Module IV (8 hours)

Recurrent Neural Networks: Back Propagation, vanishing gradients, exploding gradients, truncated backpropagation through time, Gate Recurrent Units (GRUs), Long Short-Term Memory (LSTM) cells, solving the vanishing gradient problem with LSTMs

Sequence Learning problem:-

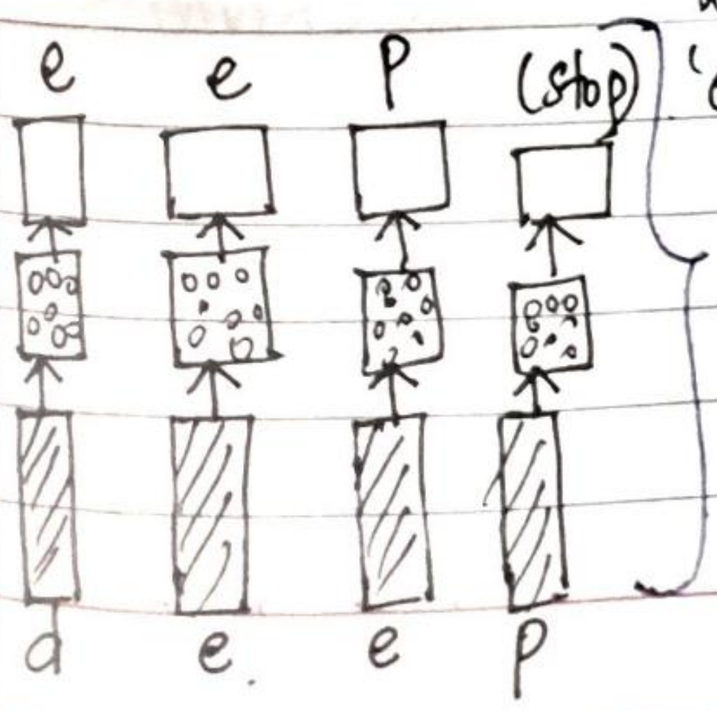
* In feedforward and convolutional neural n/ws the size of the input was always fixed

  eg:- fed fixed size (32 × 32) images to convolutional neural n/ws for image classification

* Also, each input to the network was independent of the previous or future inputs

* In many applications the input is not of a fixed size. Moreover, successive inputs may not be independent of each other

  eg:- Consider the task of auto completion
  ↳ Given the first character 'd' you want to predict the next character 'e' and so on



↳ First, successive inputs are no longer independent (to predict 'e' in the ~~second~~ ~~day~~ we would want to know what the previous input was)

o/ps depends on previous i/ps also

↳ Second, the length of the inputs and the number of predictions you need to make is not fixed (eg (deep, learn, machine, have different number of characters)

↳ Third, each n/w (RNN) is performing the same task (i/p = a character, o/p = a character)

↑ the length of the input is not fixed

eg:- ↳ Sentimental Analysis
↳ Speech Recognition
↳ Video Labeling

How to model the tasks involving sequences

Target points:

* Account for dependence b/w inputs
* Account for variable number of inputs
* The function executed at each time step is the same.

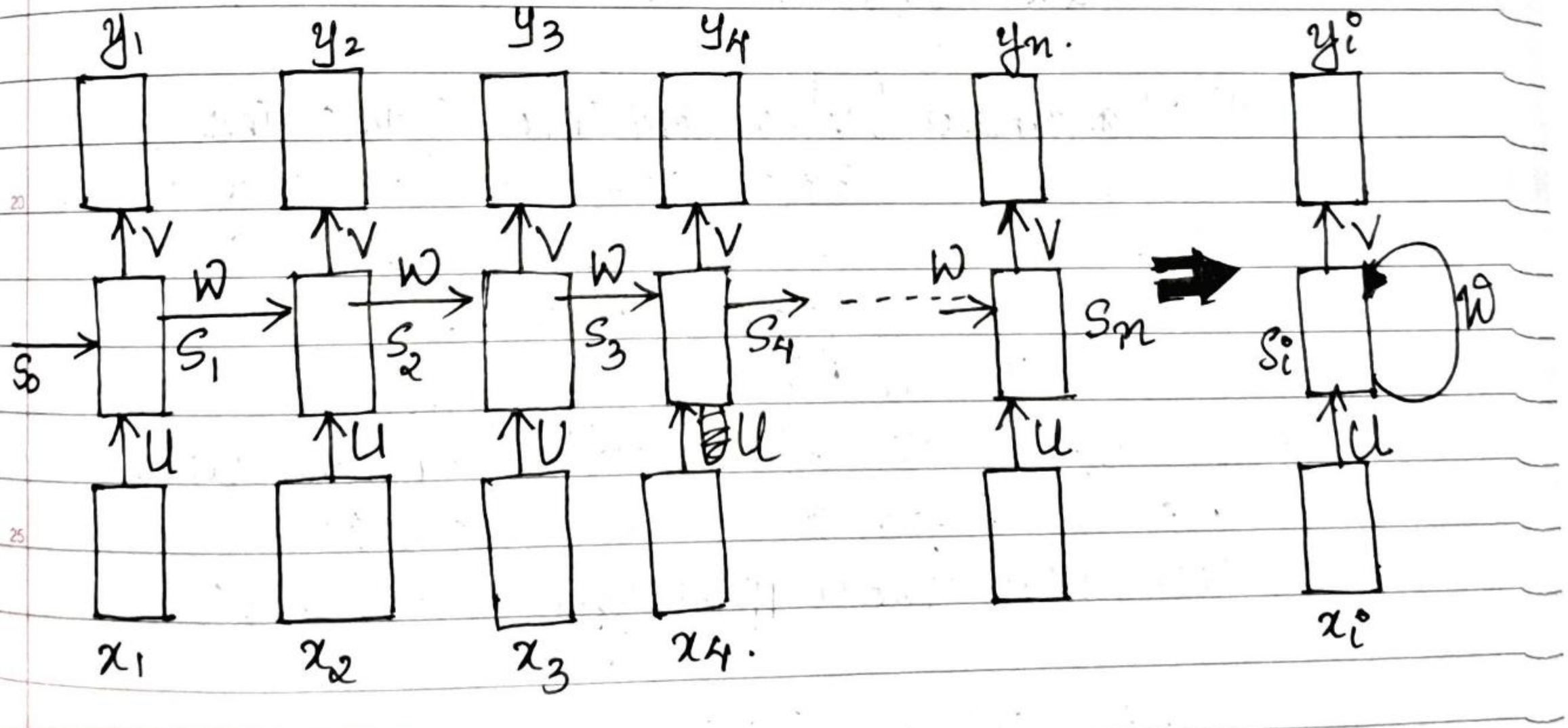
To train a model that has a kind of sequence inputs need Recurrent Neural Network (RNN)

Recurrent Neural Network:-

* RNN is a type of neural network in which the o/p from the previous step is fed as input to the current step. / is a type of ANN which uses sequential data or time series data
* RNN is normally used in NLP and some other domains.

* RNN's also have a hidden stage which used to capture information about a sentence (Scenario=NLP)

* RNN's have a memory, which is used to capture information about the calculations made so far.



$$S_i = \sigma(Ux_i + WS_{i-1} + b)$$

$$y_i = O(VS_i + C)$$

Or

$$y_i = f(x_i, S_i, U, V, W)$$

Camlin Page
Date / /

where S_i = state of the network at timestep 'i'. ← hidden state at time 'i'

c = true class

* The parameters are W, U, V which are shared across timesteps.

* Same network can be used to compute $y_1, y_2, \ldots, y_{10}$ or y_n.

Backpropagation.

* Parameter 'U' transformation the input x_t to the state S_t

* Parameter 'W' transforms the previous state S_{t-1} to the present state S_t.

* Parameter 'V' maps the computed internal state S_t to the output O_t.

[S is replaced with h]

* If tanh is the activation fn used and softmax is the activation fn used in the o/p layer

$h_t = \tanh(Ux_t + Wh_{t-1})$

$\hat{y}_t = \text{softmax}(Vh_t)$

Types of RNN Architectures:

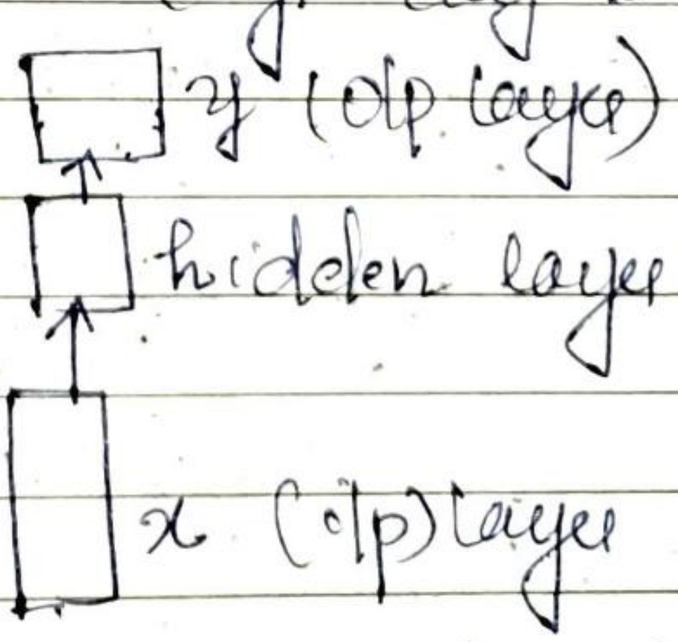
* The common architectures
  - ↳ One to One
  - ↳ One to Many
  - ↳ Many to One
  - ↳ Many to Many

* One to One :-

↳ This model is similar to a single layer neural n/w as it only provides single predictions

↳ It is mostly used fixed-size i/p 'x' and fixed-size o/p 'y'

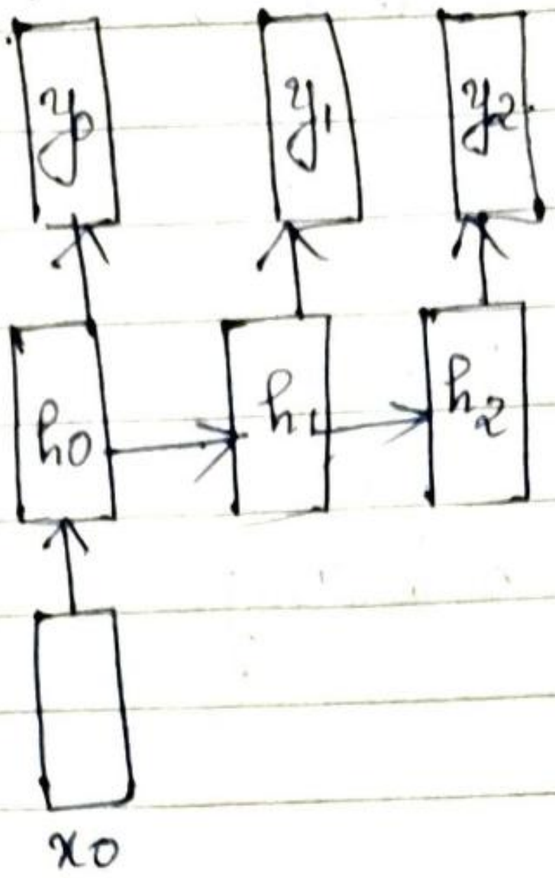
eg :- image classification (eg: dog vs cat)



* One to Many :-

↳ This consist of a single i/p 'x', activation 'a' and multiple o/p 'y'.

eg :- Generating an audio stream. It takes a single audio sbeam as i/p and generates new tones or new music based on that stream.



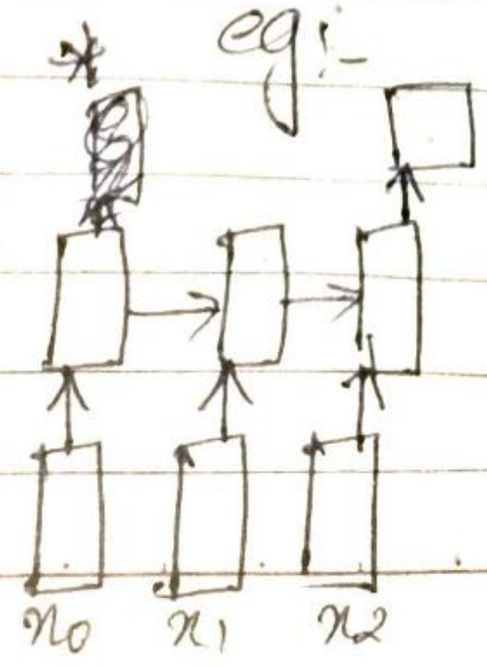
eg:- Image captioning
(i/p as image and o/p tells the content in the image)

Many to One:-

* This consist of multiple inputs 'x' (such as words or sentence), activation 'a' and produce a single o/p 'y' at the end.

* This type of architecture mostly used to perform sentiment analysis as it processes the entire i/p (collection of words) to produce a single o/p (positive, negative, or neutral sentiment)

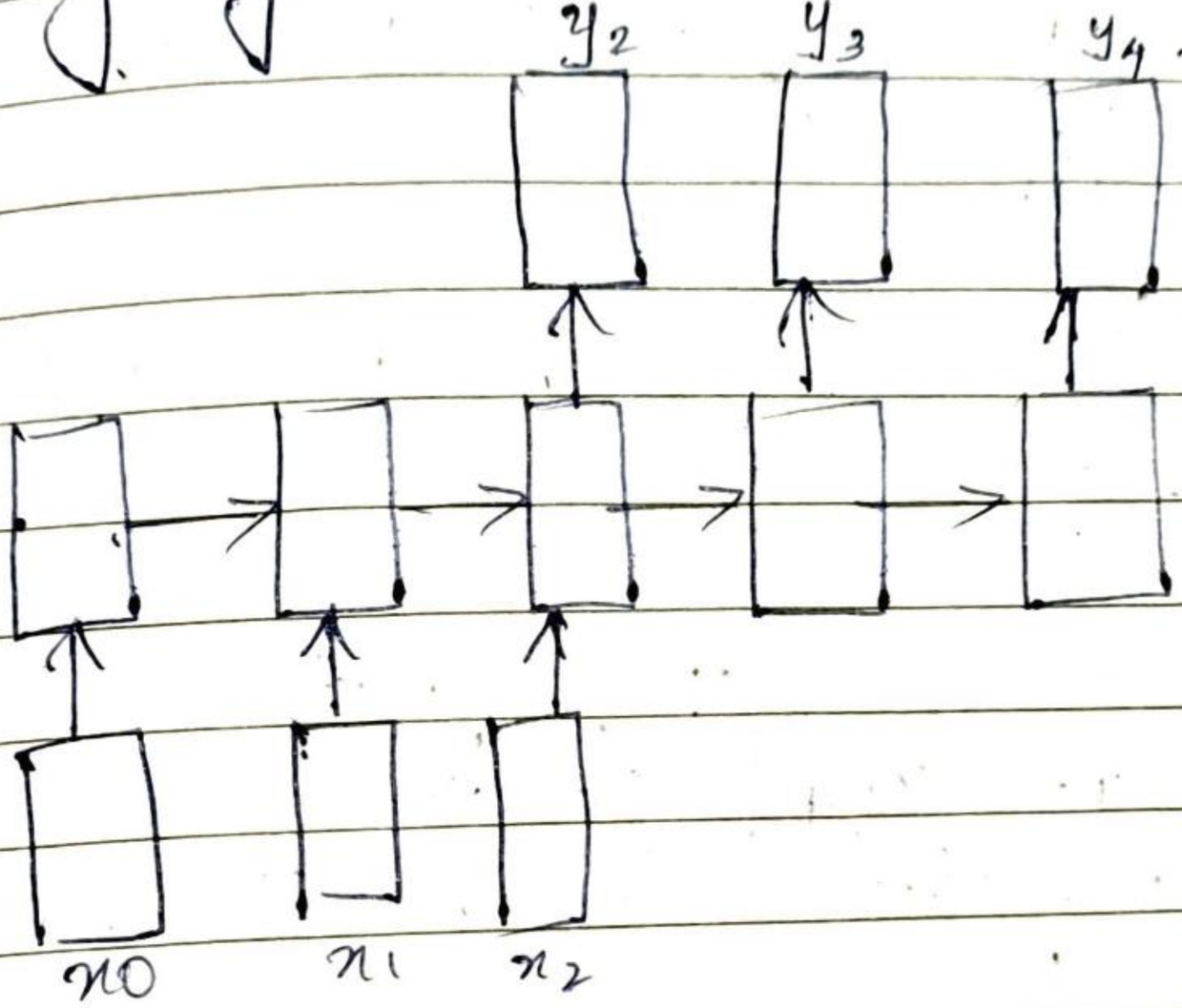
* eg:- Stock price prediction



Camlin Page
Date / /

Many to Many :-

↳ Many-to-Many sequence learning can be used for machine translation where the ip sequence is in some language and the op sequence is in some other language.



Backpropagation through time (BPTT)

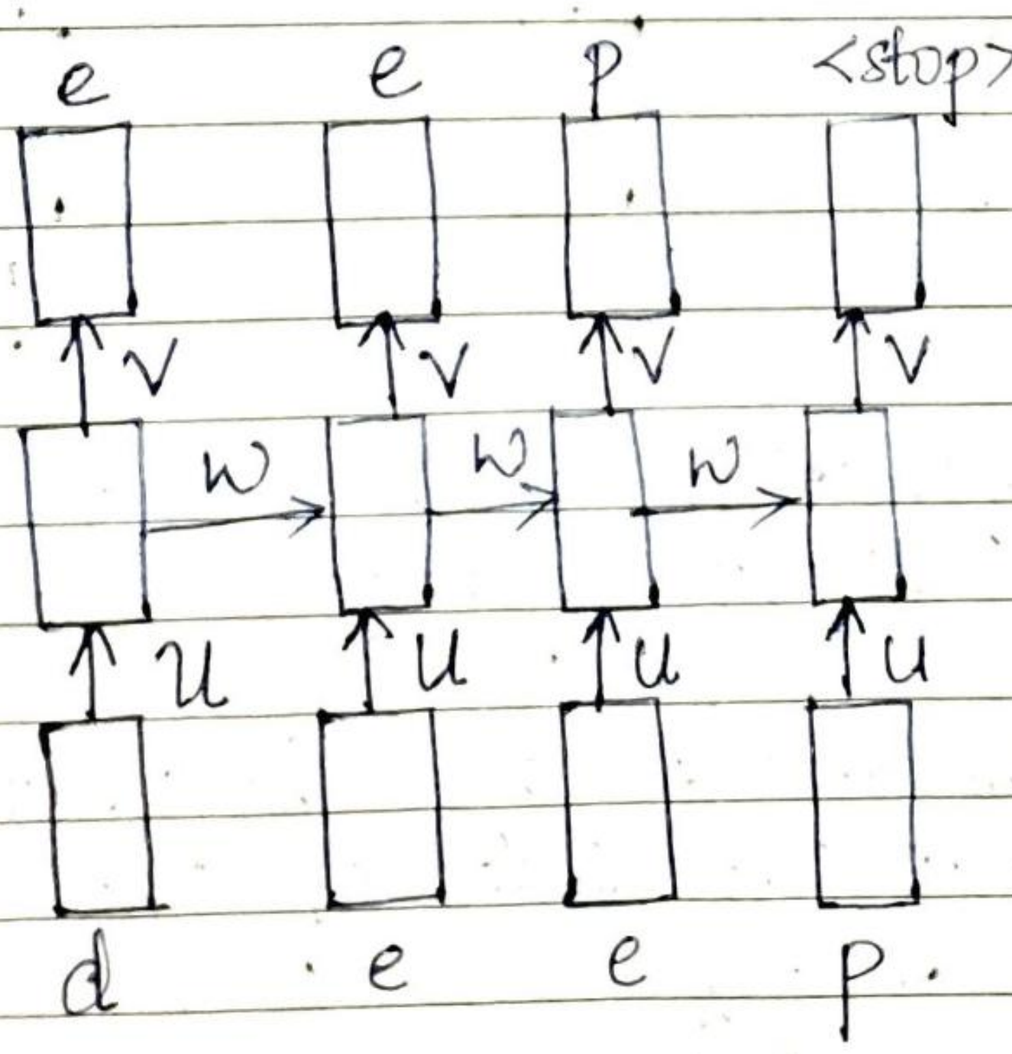
* Dimensions of the parameters

$x_i \in \mathbb{R}^n$ (n dimensional i/p)
$s_i \in \mathbb{R}^d$ (d-dimensional state)
$y_i \in \mathbb{R}^k$ (say 'k' classes)
$U \in \mathbb{R}^{n \times d}$
$V \in \mathbb{R}^{d \times k}$
$W \in \mathbb{R}^{d \times d}$

* Let us consider an example "task of auto completion (predicting the next character)

↳ For simplicity we assume that there are only 4 characters in our vocabulary ('d, e, p, <stop>)

↳ At each timestep we want to predict one of these 4 characters

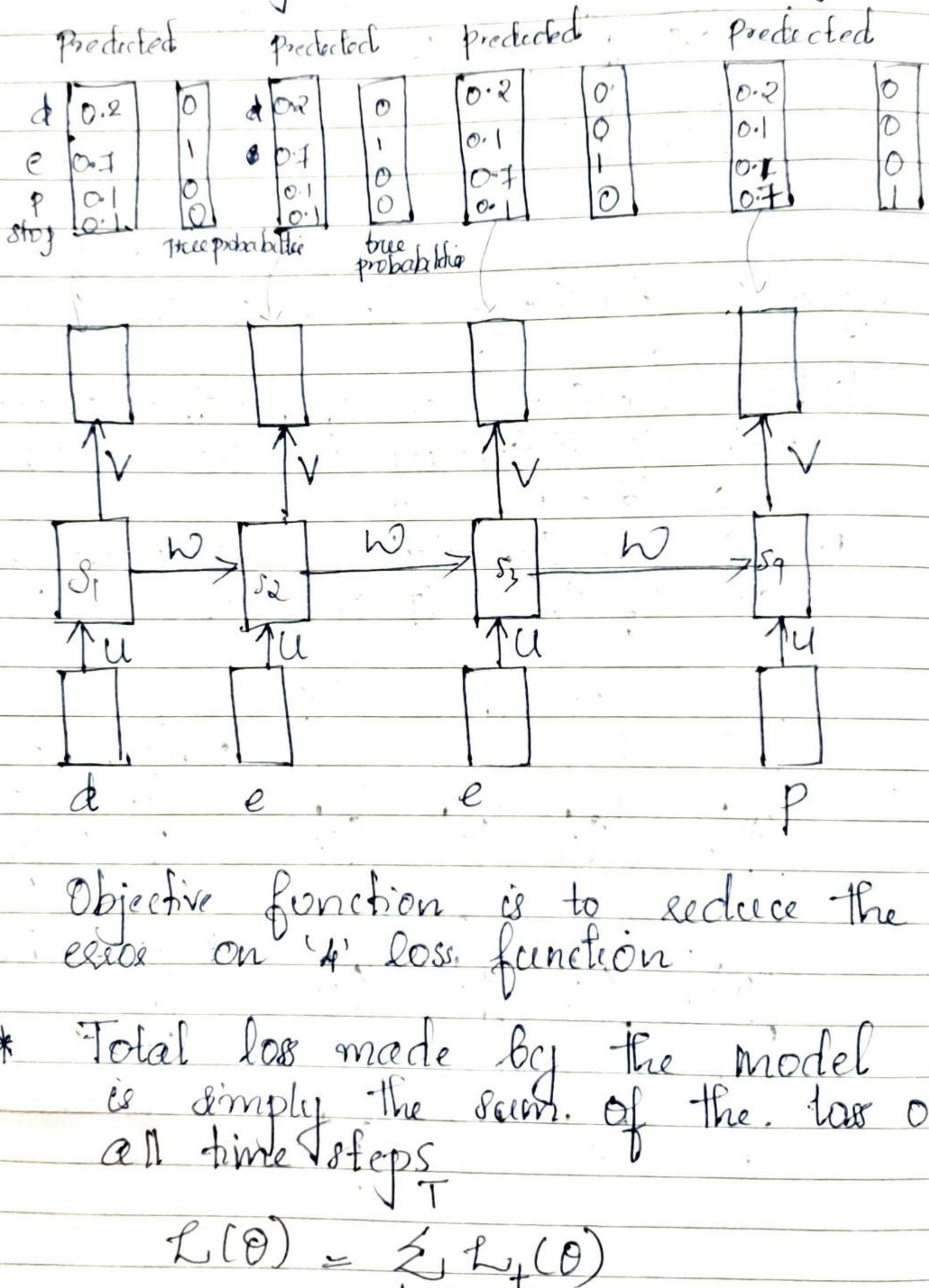


* Suitable o/p fn for this task is softmax

* Suitable loss fn for this task is cross entropy

* Suppose we initialize U, V, W randomly and the n/w predicts the probabilities

* Initially predict 'd' as input, U, V, W are randomly initialized except matrix which predicts some probability distribution



* Objective function is to reduce the error on 'd'. loss function.

* Total loss made by the model is simply the sum of the loss over all time steps

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

Camlin | Page
Date / /

'eat time step 1
e at time step 2
p at time step 3
stop at time step 4

$L_t(\theta) = -\log(y_{tc})$

↳ true class at time step 't'
↳ loss at time step 't'

y_{tc} = Predicted probability of true character at time step 'i'.

T = number of time steps.

* For backpropagation we need to compute the gradient w.r.t W, U, V

→ (1) $\frac{\partial L(\theta)}{\partial V}$

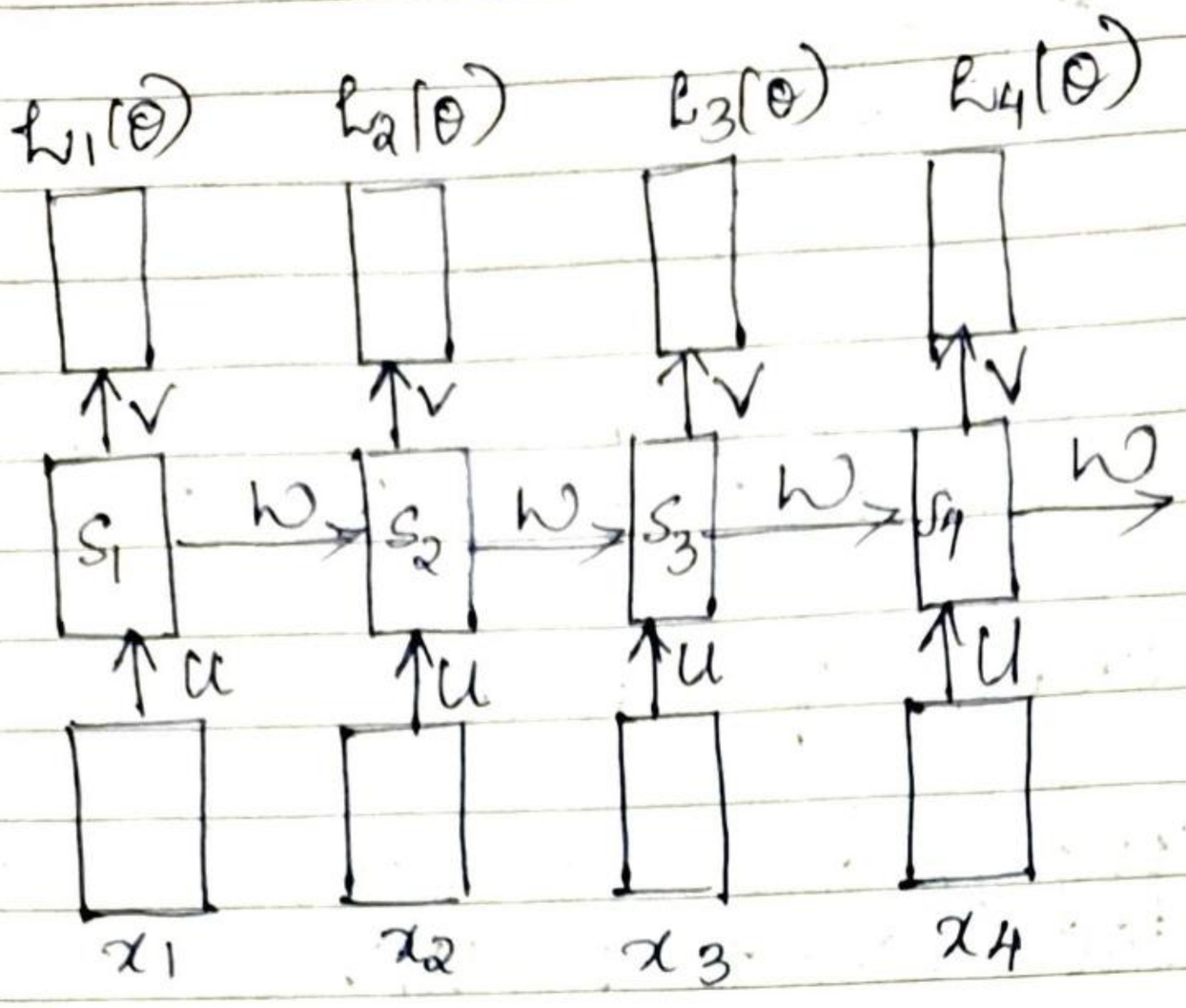
$$\frac{\partial L(\theta)}{\partial V} = \sum_{t=1}^{T} \frac{\partial L_t(\theta)}{\partial V}$$

* Each term is the summation is simply the derivative of the loss w.r.t the weights in the o/p layer

→ (2) $\frac{\partial L(\theta)}{\partial W}$

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

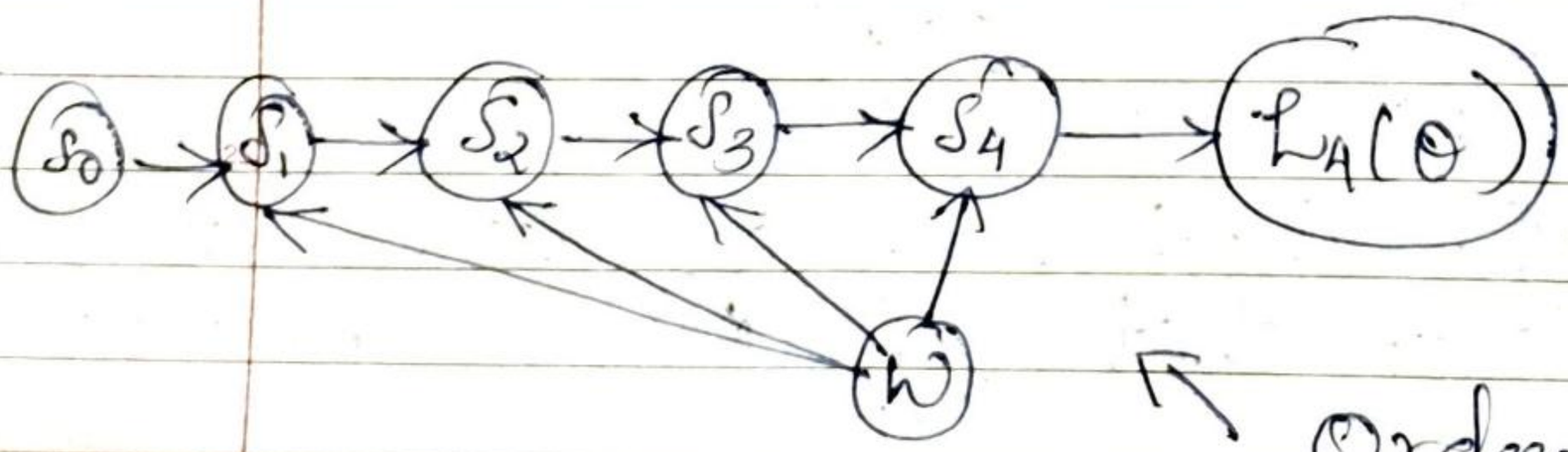
↳ By the chain rule of derivatives, we know that $\frac{\partial L_t(\theta)}{\partial W}$ is obtained by summing gradients along all the paths from $L_t(\theta)$ to W.



No: of Paths connecting $L_t(\theta)$ to W = 4 (according to eg)

* $L_4(\theta)$ depends on δ_4
* δ_4 depends on W and δ_3
* δ_3 depends on W and δ_2
* δ_2 depends on W and δ_1
* δ_1 depends on W and δ_0

where δ_0 is a constant starting state



↖ Ordered n/w.

* In an ordered n/w each state variable is computed at a time in a specified order (first δ_1, then δ_2 and so on)

* Now we have

$$\frac{\partial L_4(\theta)}{\partial w} = \frac{\partial L_4(\theta)}{\partial s_4} \frac{\partial s_4}{\partial w}$$

↓

Can be computed easily using chain rule. Since $L_4(\theta)$ depends on V and V depends on s_4

$$\frac{\partial L_4(\theta)}{\partial s_4} = \frac{\partial L_4(\theta)}{\partial V} * \frac{\partial V}{\partial s_4}$$

* $\frac{\partial s_4}{\partial w} = ?$ $\quad s_4 = \sigma(U x_4 + W s_3 + b)$

(lets eliminate σ)

$$\frac{\partial s_4}{\partial w} = \frac{\partial}{\partial w}(U x_4 + W s_3 + b)$$

$$= \otimes \underline{\underline{s_3}}$$

s_4 depends on s_3, which depends on s_2 and so on

(In such an ordered n/w, we can't compute $\frac{\partial s_4}{\partial w}$ by simply treating s_3 as constant (bcz it also depends on w)

↳ In such ordered n/ws the total derivative $\frac{\partial s_4}{\partial w}$ has two parts

↳ Explicit and Implicit

Explicit:- $\frac{\partial^+ s_4}{\partial w}$, treating all other inputs as constant

Implicit :- Summing over all indirect paths from s_4 to w

$$\frac{ds_4}{dw} = \underbrace{\frac{\partial^+ s_4}{\partial w}}_{\text{Explicit}} + \underbrace{\frac{ds_4}{ds_3}\frac{ds_3}{dw}}_{\text{Implicit}}$$

$$= \frac{\partial^+ s_4}{\partial w} + \frac{ds_4}{ds_3}\left[\underbrace{\frac{\partial^+ s_3}{\partial w}}_{\text{explicit}} + \underbrace{\frac{ds_3}{ds_2}\frac{ds_2}{dw}}_{\text{implicit}}\right]$$

$$= \frac{\partial^+ s_4}{\partial w} + \frac{ds_4}{ds_3}\frac{\partial^+ s_3}{\partial w} + \frac{ds_4}{ds_3}\frac{ds_3}{ds_2}\left[\frac{\partial^+ s_2}{\partial w} + \frac{ds_2}{ds_1}\frac{ds_1}{dw}\right]$$

$$= \frac{\partial^+ s_4}{\partial w} + \frac{ds_4}{ds_3}\frac{\partial^+ s_3}{\partial w} + \frac{ds_4}{ds_3}\frac{ds_3}{ds_2}\frac{\partial^+ s_2}{\partial w} + \frac{ds_4}{ds_3}\frac{ds_3}{ds_2}\frac{ds_2}{ds_1}\left[\frac{ds_1}{dw}\right]$$

$$= \frac{\partial^+ s_4}{\partial w} + \frac{ds_4}{ds_3}\frac{\partial^+ s_3}{\partial w} + \frac{ds_4}{ds_3}\frac{ds_3}{ds_2}\frac{\partial^+ s_2}{\partial w} + \frac{ds_4}{ds_3}\frac{ds_3}{ds_2}\frac{ds_2}{ds_1}\frac{ds_1}{dw}$$

$$= \frac{\partial^+ s_4}{\partial w} + \frac{ds_4}{ds_3}\frac{\partial^+ s_3}{\partial w} + \frac{ds_4}{ds_3}\frac{ds_3}{ds_2}\left[\frac{\partial^+ s_2}{\partial w} + \frac{ds_2}{ds_1}\frac{ds_1}{dw}\right]$$

$$= \frac{\partial^+ s_4}{\partial w} + \frac{ds_4}{ds_3}\frac{\partial^+ s_3}{\partial w} + \frac{ds_4}{ds_3}\frac{ds_3}{ds_2}\frac{\partial^+ s_2}{\partial w} + \frac{ds_4}{ds_3}\frac{ds_3}{ds_2}\frac{ds_2}{ds_1}\frac{ds_1}{dw}$$

tensor

$$= \frac{\partial^+ s_4}{\partial w} + \frac{ds_4}{ds_3}\frac{\partial^+ s_3}{\partial w} + \frac{ds_4}{ds_3}\frac{ds_3}{ds_2}\frac{\partial^+ s_2}{\partial w} + \frac{ds_4}{ds_3}\frac{ds_3}{ds_2}\frac{ds_2}{ds_1}\frac{\partial^+ s_1}{\partial w} + \frac{ds_4}{ds_3}\frac{ds_3}{ds_2}\frac{ds_2}{ds_1}\frac{ds_1}{ds_0}\frac{\partial^+ s_0}{\partial w}$$

Camlin | Page
Date / /

* For simplicety, we will short-circuit some of the paths

$$\frac{\partial s_4}{\partial w} = \frac{\partial s_4}{\partial s_4}\frac{\partial^+ s_4}{\partial w} + \frac{\partial s_4}{\partial s_3}\frac{\partial^+ s_3}{\partial w} + \frac{\partial s_4}{\partial s_2}\frac{\partial^+ s_2}{\partial w} + \frac{\partial s_4}{\partial s_1}\frac{\partial^+ s_1}{\partial w} + \frac{\partial s_4}{\partial s_0}\frac{\partial^+ s_0}{\partial w}$$ neglected

$$\frac{\partial s_4}{\partial w} = \sum_{k=1}^{4} \frac{\partial s_4}{\partial s_k}\frac{\partial^+ s_k}{\partial w}$$

* Finally we have

$$\frac{\partial L_4(\theta)}{\partial(w)} = \frac{\partial L_4(\theta)}{\partial s_4}\frac{\partial s_4}{\partial w}$$

we know $\frac{\partial s_4}{\partial w} = \sum_{k=1}^{4} \frac{\partial s_4}{\partial s_k}\frac{\partial^+ s_k}{\partial w}$

In general

$$\therefore \frac{\partial L_t(\theta)}{\partial(w)} = \frac{\partial L_4(\theta)}{\partial s_t} \sum_{k=1}^{t} \frac{\partial s_t}{\partial s_k}\cdot\frac{\partial^+ s_k}{\partial w}$$

* This algorithm is called Backpropagation through time (BPTT) as we backpropagate over all previous time steps

Ans.

→ (3) $\boxed{\frac{\partial L(\theta)}{\partial u}}$

$$\frac{\partial L(\theta)}{\partial u} = \sum_{K=0}^{T} \frac{\partial L(\theta)}{\partial s_K} \frac{\partial s_K}{\partial u} \qquad \left[\frac{\partial s_K}{\partial u} = x_K\right]$$

$$\sum_{K=0}^{T} \frac{\partial L(\theta)}{\partial s_K} \cdot x_K$$

$$\frac{\partial L(\theta)}{\partial s_K} = ?$$

$$s_K = \sigma(U x_K + W s_{K-1} + b)$$

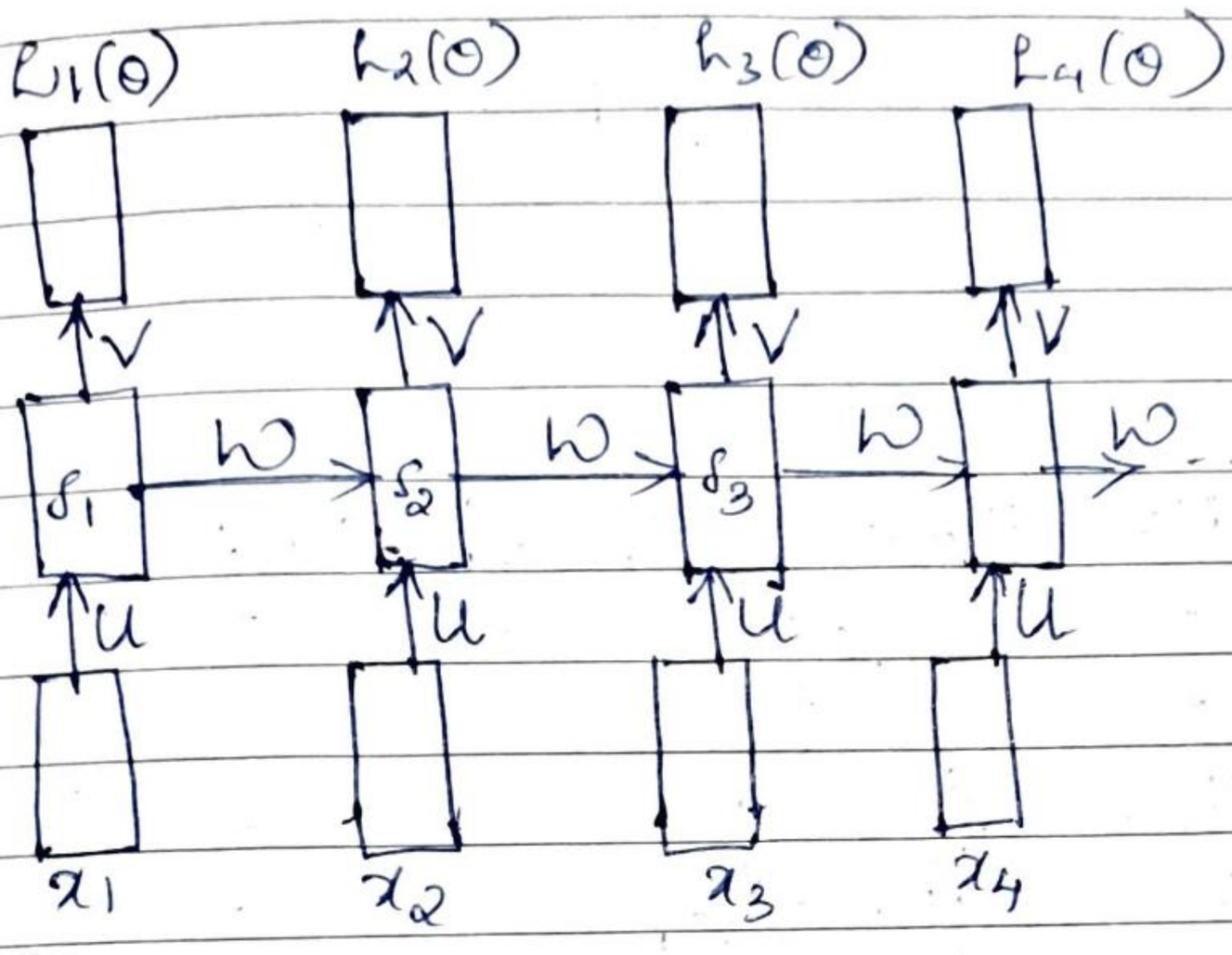
[U contributes to all s_K's up to $K=t$]

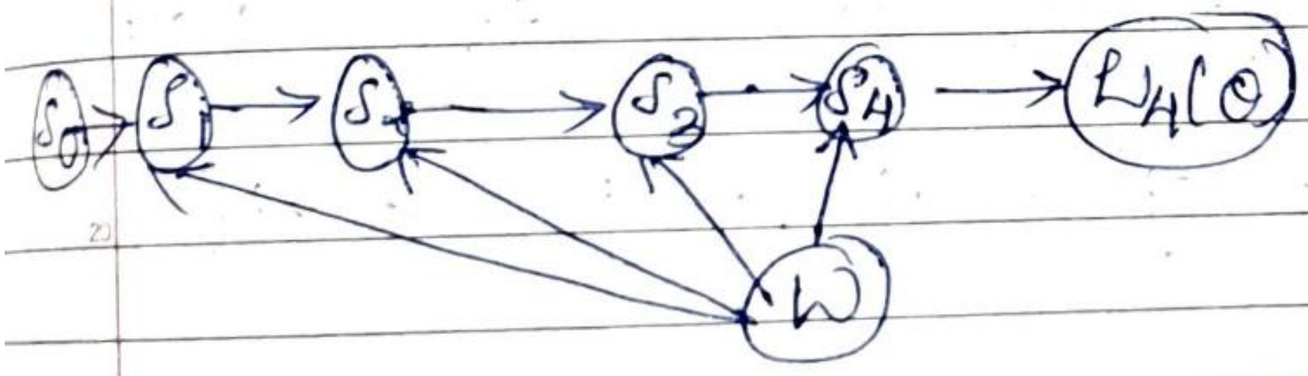
s_K depends on s_{K-1}

$$\frac{\partial L(\theta)}{\partial s_{K-1}} = \frac{\partial L(\theta)}{\partial s_K} \frac{\partial s_K}{\partial s_{K-1}}$$

So, only need to compute $\frac{\partial s_K}{\partial s_{K-1}}$ for each decrement of k till the first term so, which is a constant.

The Problem of Exploding and Vanishing Gradients





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

* We will now focus on $\frac{\partial s_t}{\partial s_k}$ and highlight an important problem in training RNN's using BPTT

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

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

* Let's explore One such term in the product ie $\frac{ds_{j+1}}{ds_j}$

can also written as

$$\frac{ds_j}{ds_{j-1}}$$

* $\boxed{\frac{ds_j}{ds_{j-1}}}$:-

a_j (pre activation) $= W s_{j-1} + b + U x_j$

(activation) $s_j = \sigma(a_j)$

$$\therefore \frac{ds_j}{ds_{j-1}} = \frac{ds_j}{da_j} \frac{da_j}{ds_{j-1}}$$

① → present, ② → previous

ds_j = vector, da_j = matrix

da_j = vector, $\frac{ds_j}{da_j}$ = matrix

let $a_j = [a_{j1}, a_{j2}, a_{j3} \cdots a_{jd}]$ [d dimension]

$s_j = [\sigma(a_{j1}), \sigma(a_{j2}), \sigma(a_{j3}) \cdots \sigma(a_{jd})]$

① $\frac{ds_j}{da_j} = \begin{bmatrix} \frac{ds_{j1}}{da_{j1}} & \frac{ds_{j2}}{da_{j1}} & \frac{ds_{j3}}{da_{j1}} & \cdots \\ \frac{ds_{j1}}{da_{j2}} & \frac{ds_{j2}}{da_{j2}} & \cdots & \\ & & \frac{ds_{j3}}{da_{j3}} & \\ \cdots & & & \frac{ds_{j4}}{da_{j4}} \cdots \end{bmatrix}$

$$= \begin{bmatrix} \sigma'(a_{j1}) & 0 & 0 & 0 \\ 0 & \sigma'(a_{j2}) & 0 & 0 \\ 0 & 0 & \sigma'(a_{j3}) & 0 \\ 0 & 0 & 0 & \sigma'(a_{j4}) \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \sigma'(a_{jd})$$

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

$$* \quad \frac{\partial s_j}{\partial s_{j-1}} = \frac{\partial s_j}{\partial a_j} \cdot \frac{\partial a_j}{\partial s_{j-1}} \rightarrow \text{Weight Matrix}$$

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

* Lets find the magnitude of $\frac{\partial s_j}{\partial s_{j-1}}$

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

Rule:- $\|c\| = \|ab\|$
$\leq \|a\| \|b\|$

$$\leq \|\text{diag}(\sigma'(a_j))\| \|W\|$$

↳ Lets see find out $\|\text{diag}(\sigma'(a_j))\|$

$\sigma(a_j)$ is a bounded fn (Sigmoid, tanh)
$\sigma'(a_j)$ is also bounded

↳ $\sigma'(aj) \leq \frac{1}{4} = \gamma$ (if σ is logestic)

↳ $\sigma'(aj) \leq 1 = \gamma$ (if σ is tanh)

↳ $\|W\|$ ← eeeeght matrix is also bounded by some value λ, then -

$$\left\| \frac{ds_j}{ds_{j-1}} \right\| \leq \gamma\lambda$$

* Lets consider the whole prodeeef

$$\left\| \frac{ds_t}{ds_k} \right\| = \left\| \prod_{j=k+1}^{t} \frac{ds_j}{ds_{j-1}} \right\|$$

$$\leq \prod_{j=k+1}^{t} \gamma\lambda \quad \left[(t-k) \text{ terms} \right]$$

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

* If γ and λ greater than 1, exploding gradeent problem aeises
ie

1st Case (*) If γ and λ less than 1 or $\gamma\lambda < 1$, the geadeent will vanish

2nd Case → If $\gamma\lambda > 1$, the gradeent will explode

* To avoid exploding / vanishing gradient problem is to use truncated Backpropagation

(Instead of selecting (t-k) steps @

T 1

↓ ↓

occurs vanishing / exploding

we will restrict to some minimum previous state or terms.

* In the first case, the term goes to zero exponentially fast, which makes it difficult to learn some long period dependencies. This problem is called the vanishing gradient.

* In the second case, the term goes to infinity exponentially fast, and their value becomes a NaN due to the unstable process. This problem is called the Exploding gradient.

Truncated Backpropagation Through Time

* Backpropagation is the learning algorithm used to update the weights in a neural n/w in order to minimize the error b/w the expected o/p and the predicted o/p for a given i/p

* For sequence prediction problems, where there is

an order dependence b/w observations, RNN are used instead of classical feed forward neural n/ws.

* RNNs are trained using a variation of the backpropagation algorithm called BPTT

* BPTT unrolls the RNN and propagates the error backward over the entire i/p sequence, one timestep at a time. The weights are then updated with the accumulated gradients.

* BPTT can be slow to train RNNs on pblms with very long input sequences.

* In addition to speed, the accumulation of gradients over so many timesteps can result in a shrinking of values to zero, or a growth of values that eventually overflow, or explode.

* A modification of BPTT is to limit the no: of timesteps used on the backward pass and in effect estimate the gradient used to update the weights rather than calculate it fully. This variation is called Truncated Back Propagation through time (TBPTT)

Page
Date / /

learning algorithm

* The TBPTT has two parameters

- k_1 : Defines the no: of timesteps shown to the n/w on the forward pass
- k_2 : Defines the no: of timesteps to look at when estimating the gradient on the backward pass.

LSTM Networks
(Long Short Term Memory)

* RNN's idea is that they might be able to connect previous information to the present task.

* Sometimes, we need only to look at recent information to perform the present task

only one content { eg:- If we are trying to predict the last word in "the clouds are in the sky"

* In this case, the gap b/w the relevant information and the place that it is needed is small

* But there are also cases where we need more context
eg:- "I grew up in France. I speak fluent French".

* Recent information suggests that the next word is probably the name of a language but if we want to narrow down which language, we need the context of "France" from further back.

* Here in this case, the gap b/w the relevant information and the point where it is needed to become very large.
As the gap grows
* RNN, become unable to learn to connect the information.

* This problem of RNN is called short term Memory Problem.

* This problem is alleviated by the new n/w called LSTM.

↳ LSTM Networks:-

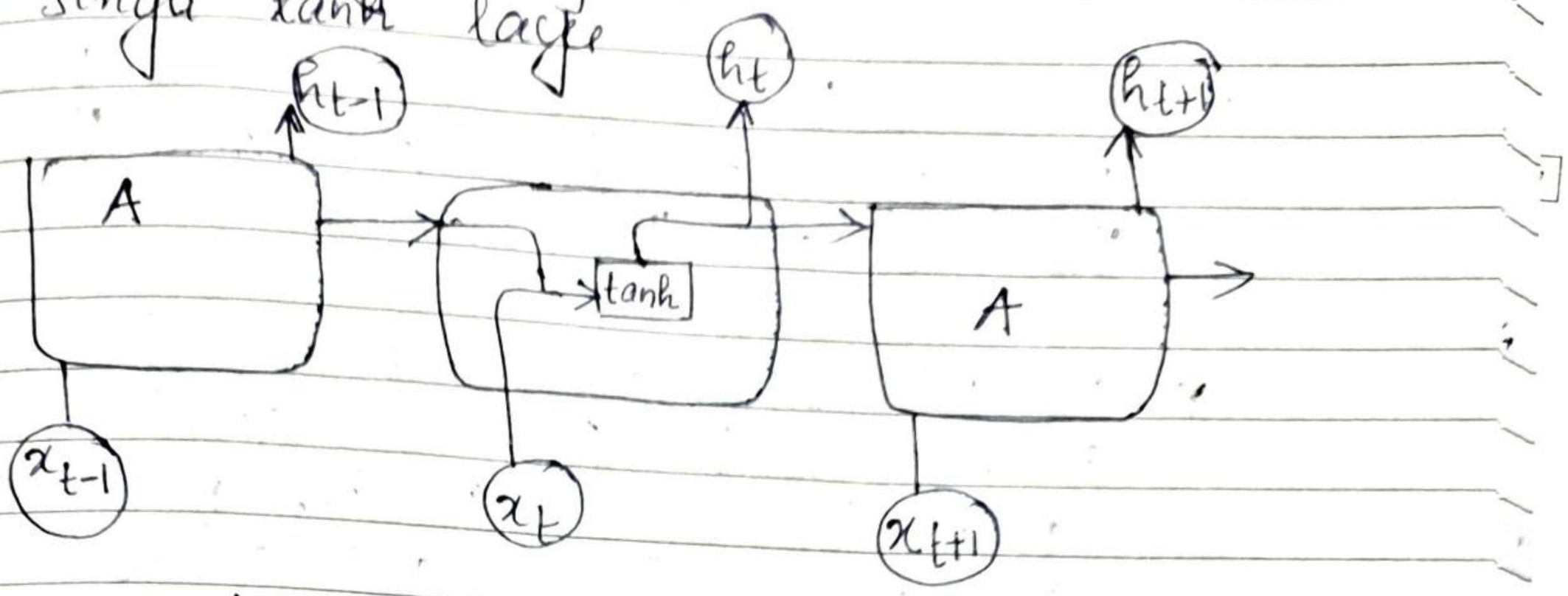
* Long short term memory n/ws - are a special kind of RNN, capable of learning long-term dependencies.

* They were introduced by Hochreiter and Schmidhuber in 1997.

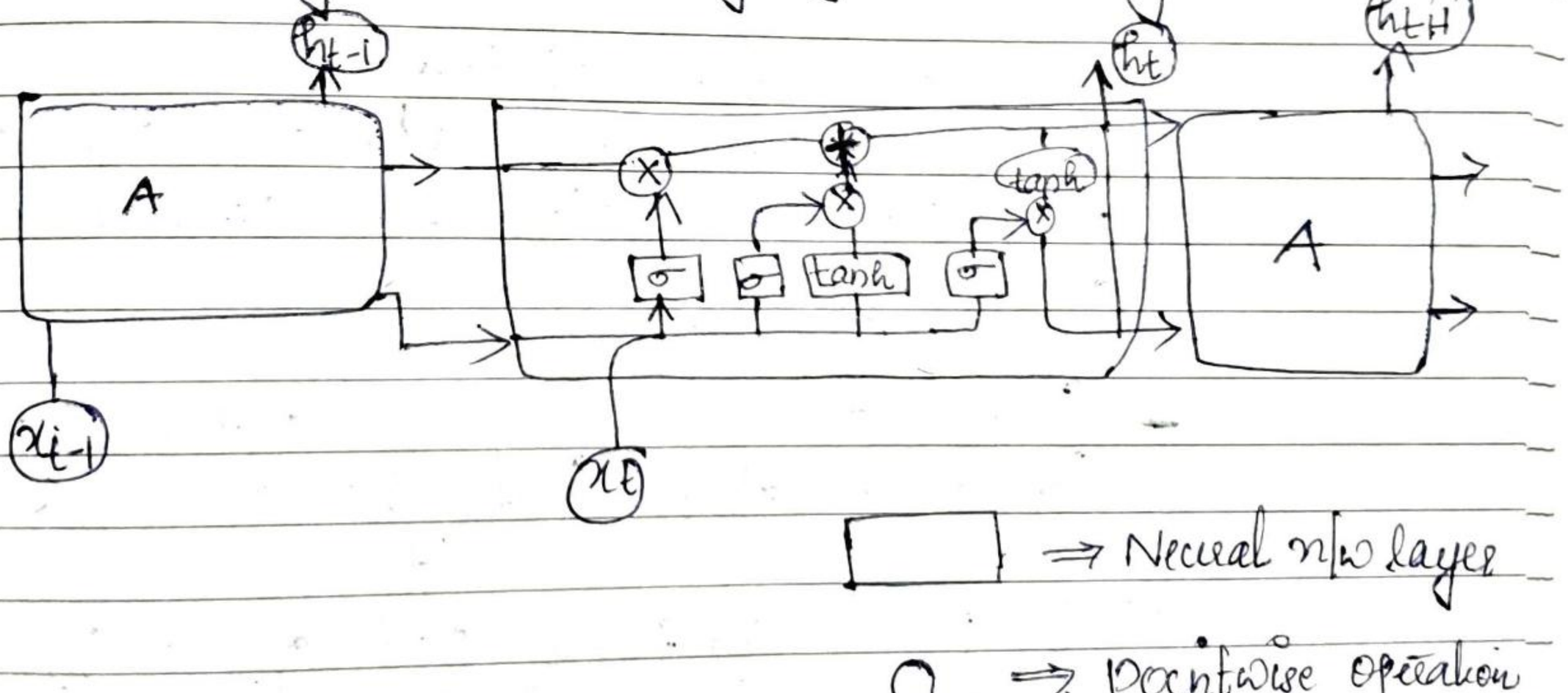
* All RNN have the form of a chain of repeating modules of neural n/w. In RNN's, this repeating module will

Camlin | Page
Date / /

have a veed simple structuree, seech as a singli tanh layer.



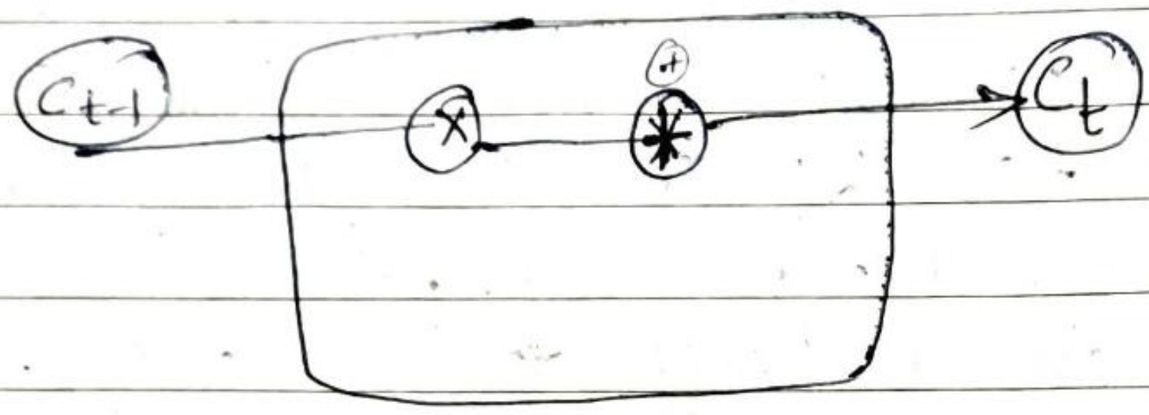
* RSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural n/w layer, there are four, interacting in a very special way.



Step 1 :- (Cell state)

* The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.

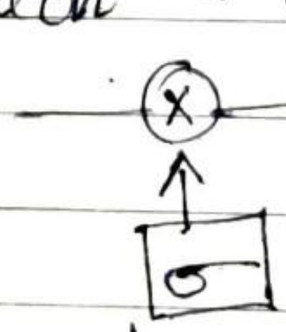
* The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions.



* LSTM have the ability to remove or add information to the cell state, carefully regulated by structures called gates

→ Gates are a way to optionally let information through.

→ They are composed out of a sigmoid neural net layer and a pointwise multiplication operation



→ Sigmoid layer o/ps numbers b/w 0 & 1 and one describing how much of each component should be let through.

↳ A value of zero means " lets nothing through "
↳ A value of One means " let everything through "

* LSTM has three of these gates to protect and control the cell state.

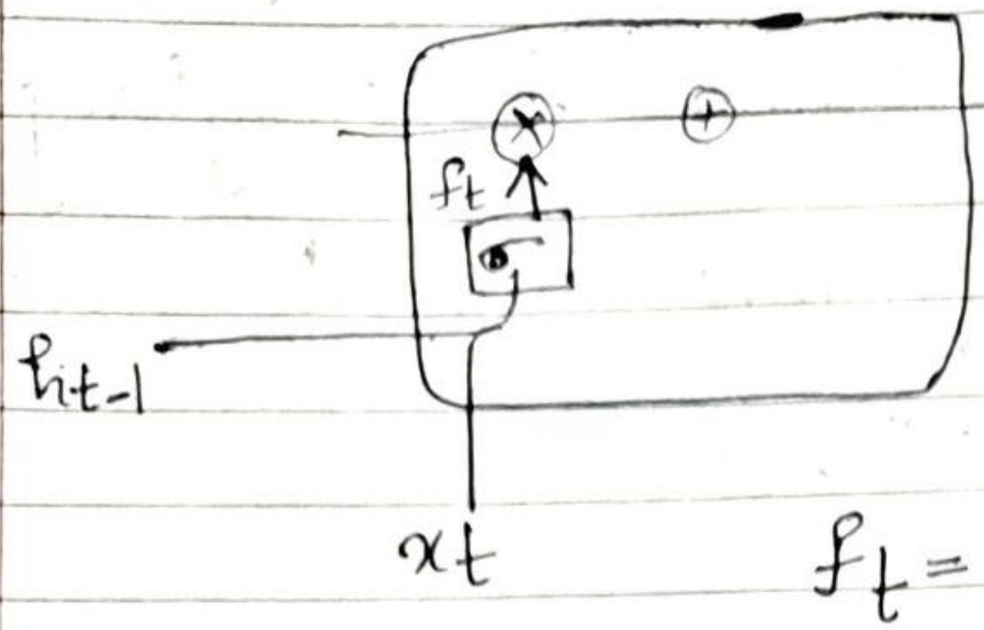
① Forget Gate layer
② Input Gate layer
③ Output Gate layer.

Step 1:-
Forget Gate layer.

* Decide what information to throw away / forget from the cell state. This decision is made by a Sigmoid layer called " forget Gate layer "

* It looks at h_{t-1} and x_t and o/p a no: b/w 0 and 1 for each no: in the cell state C_{t-1}.

* eg:- language model trying to predict the next word based on all the previous ones. In such a problem, the cell state might include the gender of the present ~~state~~ subject, so that correct pronoces can be used when we see a new subject, we want to forget the genders of the old subject



$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Step 2:- Input Gate layer:-

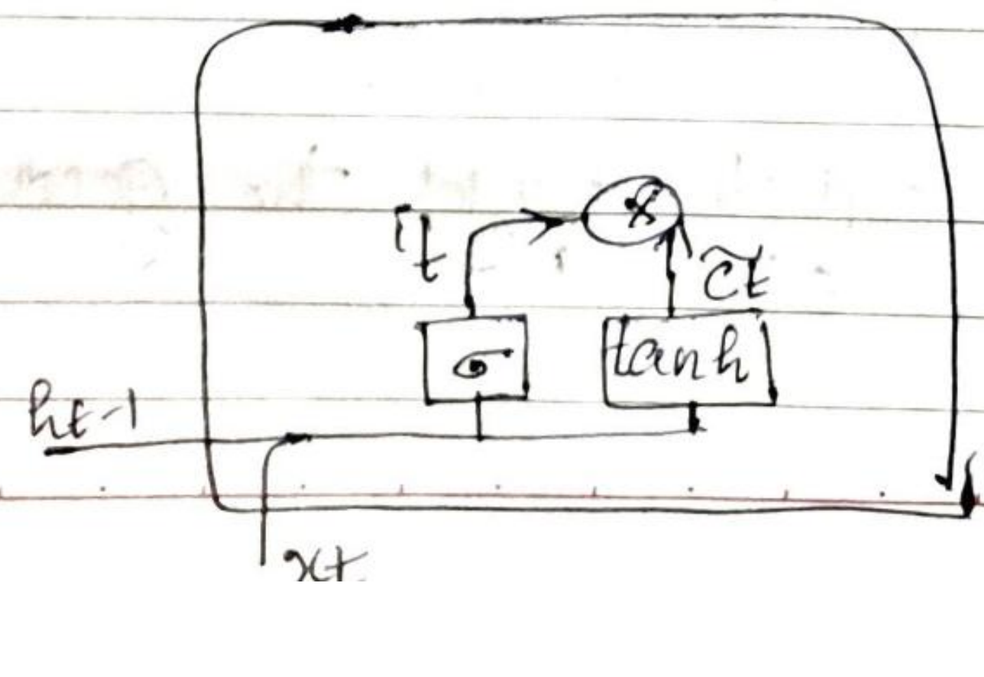
* This step decides what new information is going to store in the cell state.

* This has two parts

① ↳ sigmoid layer called the input gate layer decides which value will get updated

② ↳ A 'tanh' layer creates a vector of new candidate value $\tilde{C}_t$, that could be added to the state.

* eg:- In our language model, want to add the gender of the new subject to the cell, to replace the old one we are forgetting



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

Camlin | Page
Date / /

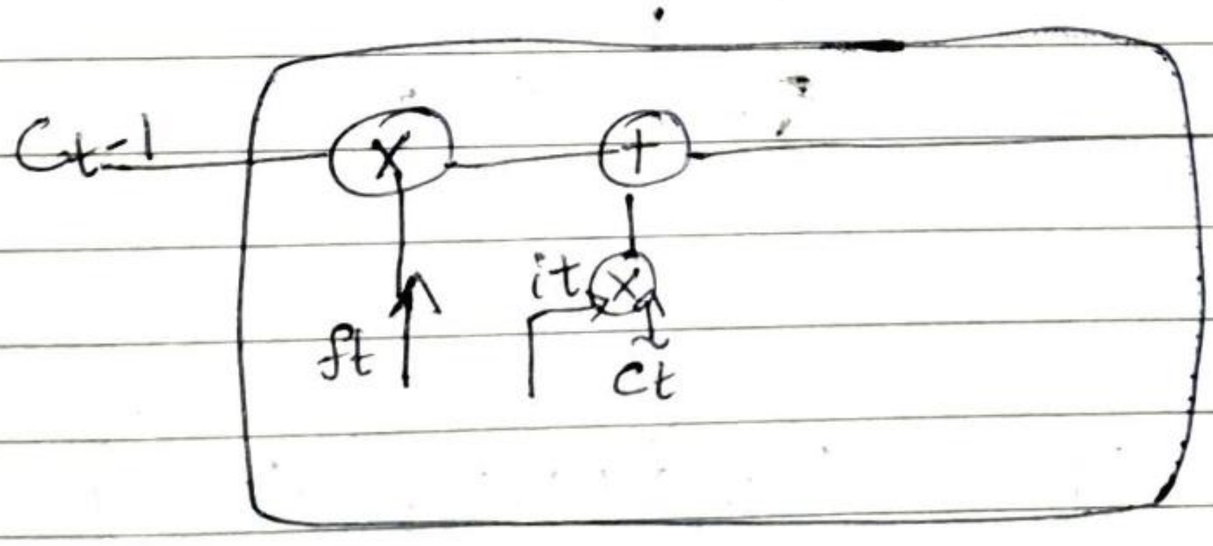
Steps:- Output Gate layer

* Update the old cell state C_{t-1}, into the new cell state C_t. Previous steps already decided what to do, in this step, we just need to actually do it.

* Multiply the old state by f_t, forgetting the things we decided to forget earlier. Then we add $i_t * \tilde{C}_t$.

$i_t * \tilde{C}_t$ → new candidate values to be updated

eg:- In the language model, this is where we'd actually drop the information about the old subject's gender and add the new information.



$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$

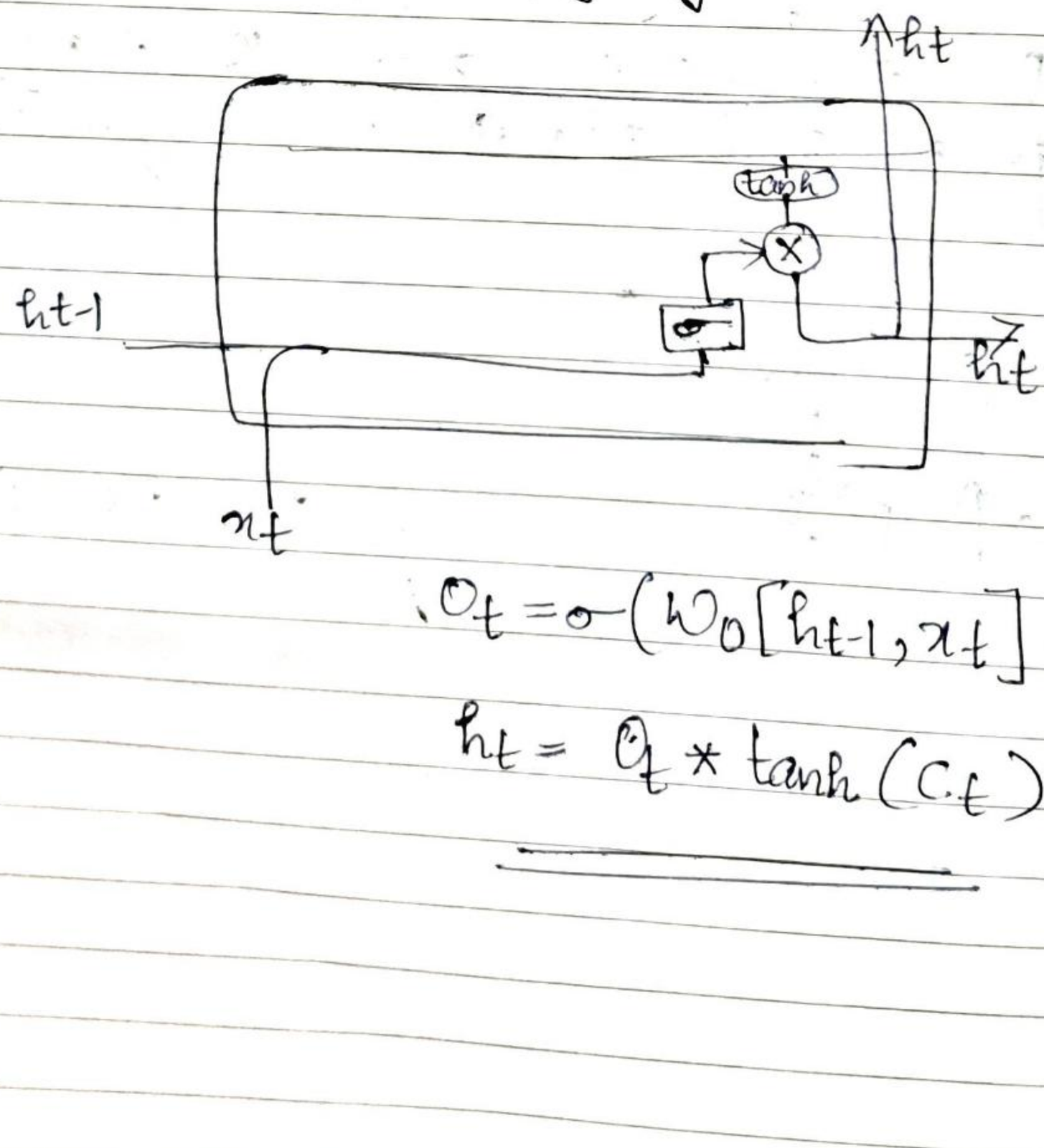
Step 3 :- Output Gate layer.

* decides what we are going to do

① * Run sigmoid layer which decides what parts of the cell state we are going to output.

② * Then, we put the cell state through tanh (b/w -1 & 1) and multiply it by the o/p of the sigmoid gate, so that we only o/p the parts we decided to

eg :- In the language model,



$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$