3) LSTM

13 December 2025 02:37 PM

for GRU

2 gates. update gate and reset gate

$$r_t = w_{r_x} \cdot x_t + w_{r_h} \cdot h_{t-1} + b_r$$

$$z_t = w_{z_x} \cdot x_t + w_{z_h} \cdot h_{t-1} + b_z$$

candidate memory. $\tilde{h}_t = \tanh(w \cdot x_t + (h_{t-1} \circledast r_t) + b)$

$$h_t = \tilde{h}_t z_t + h_{t-1}(1 - z_t)$$ → update

next state memory.

forget gate.

$$f_t = \sigma\left(w_f [x_t, h_{t-1}] + b_f\right)$$

$$i = \sigma\left(w_i [x_i, h_{t-1}] + b_i\right)$$

$$h_t = z_t \tilde{h}_t + (1 - z_t) h_{t-1}$$

$$h_t = z_t \tilde{h}_t + h_{t-1} - z_t h_{t-1}$$

$$h_t = z_t (\tilde{h}_t - h_{t-1}) + h_{t-1}$$

$z_t = 0.001$

$$h_t = z_t (\tilde{h} - h_{t-1}) + h_{t-1}$$

$-0.009 h_{t-1} + h_{t-1}$

$$h_t = 0.001 (\tilde{h} - h_{t-1}) + h_{t-1}$$

$$h_t = 0.001 \tilde{h} + 0.999 h_{t-1}$$

$h_t \approx 0.999 h_{t-1}$ → only 99.9% of the memory is retained.

f_t → fraction of current input.

$$c_t = c_{t-1}$$

$$c_t = c_{t-1} + \Delta t$$

$$c_t = f_t \circledast c_{t-1} + i_t \circledast g_t$$

i_t: input gate; g_t: candidate content; f_t: forget gate

$$\tilde{h}_t = \tanh\left(w_x \cdot x_t + w_h (h_{t-1} * r_t) + b_h\right)$$

equations for LSTM.

$$\begin{cases} f_t = \sigma\left(w_{f_x} x_t + w_{f_h} h_{t-1} + b_f\right) \\ i_t = \sigma\left(w_{i_x} \cdot x_t + w_{i_h} \cdot h_{t-1} + b_i\right) \end{cases}$$

$$i_t = \sigma\left(W_{i_x} \cdot x_t + W_{i_h} \cdot h_{t-1} + b_i\right)$$

$$g_t = \tanh\left(W_{g_x} \cdot x_t + W_{g_h} \cdot h_{t-1} + b_g\right)$$

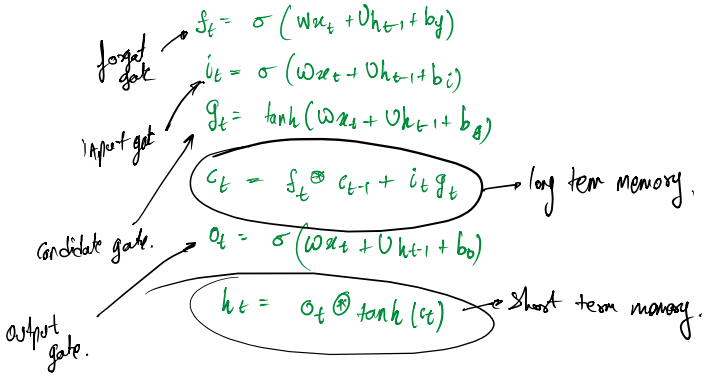
$$c_t = f_t \circledast c_{t-1} + i_t \circledast g_t$$ → expose it as long term memory.

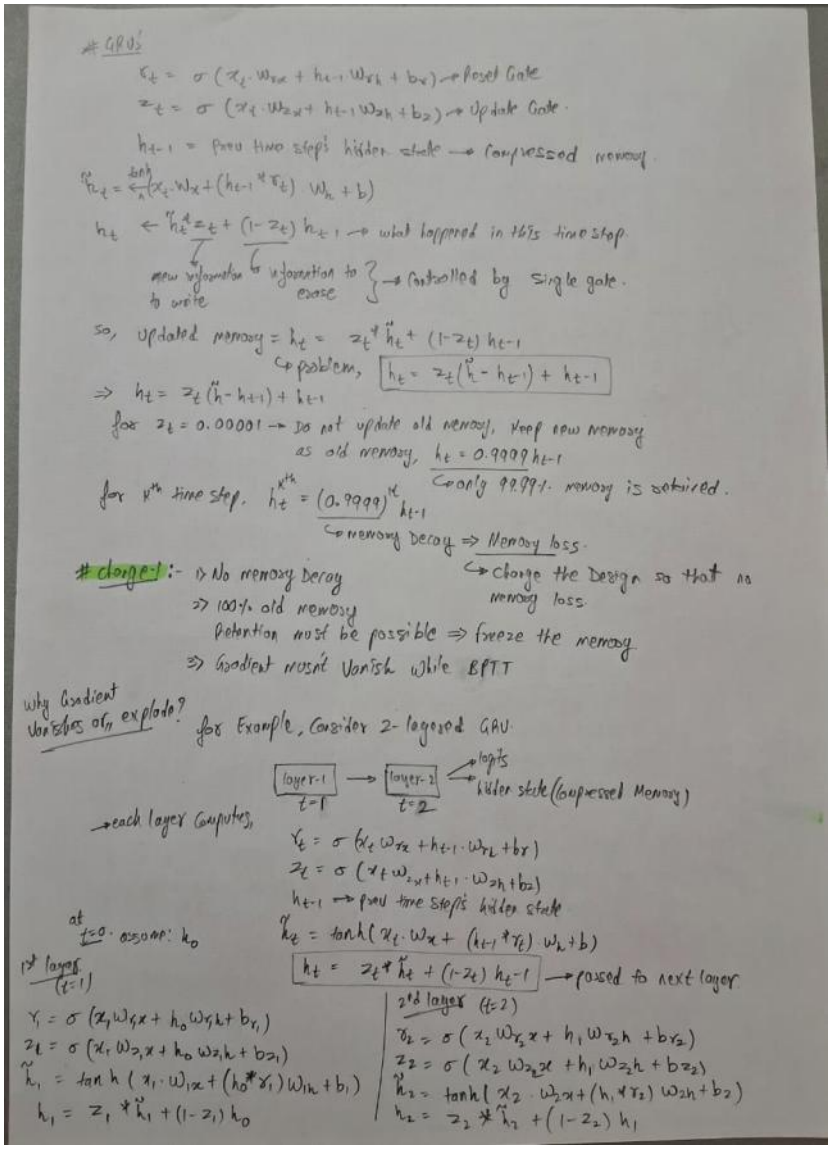
$$o_t = \sigma\left(W_{o_x} x_t + W_{o_h} h_{t-1} + b_o\right)$$

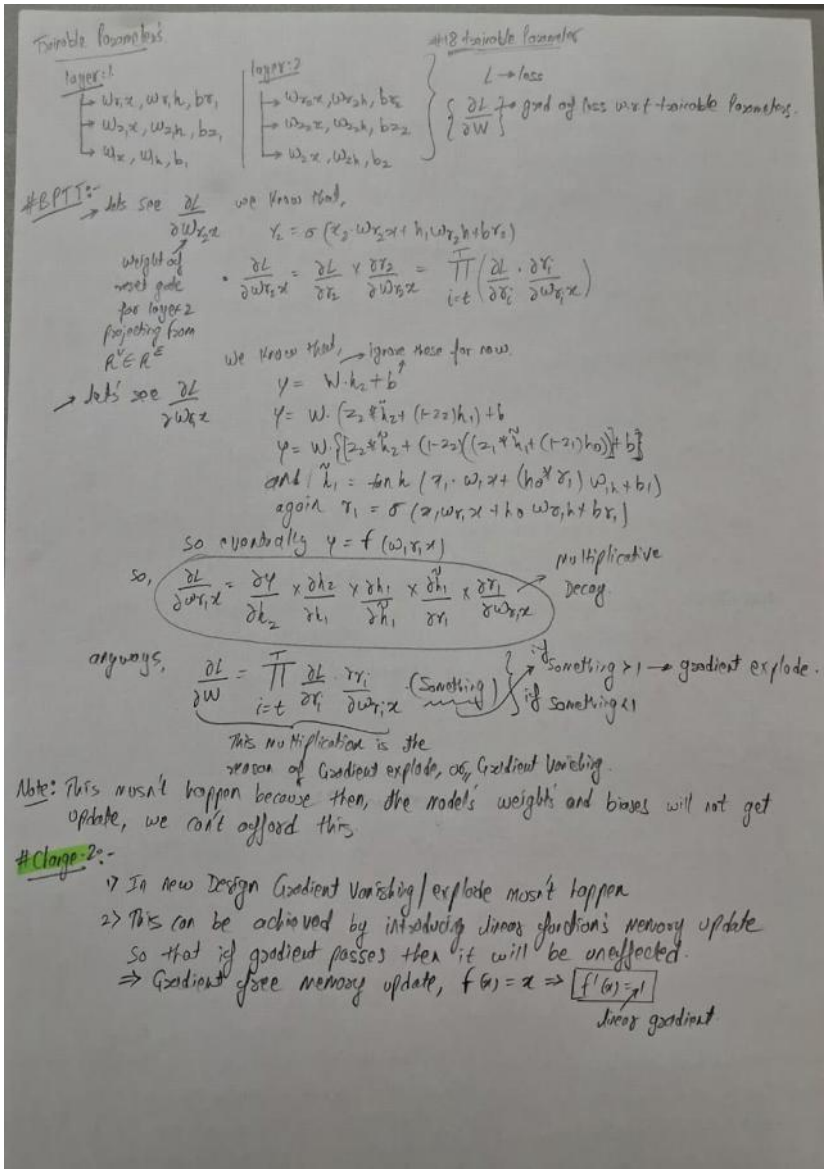
$$h_t = o_t \circledast \tanh(c_t)$$

Return long term Memory, Short term Memory

$$c_t = c_{t-1} + \Delta\varepsilon$$




GRUs
$r_t = \sigma(x_t \cdot w_{rx} + h_{t-1} \cdot w_{rh} + b_r)$ → Reset Gate
$z_t = \sigma(x_t \cdot w_{zx} + h_{t-1} \cdot w_{zh} + b_z)$ → Update Gate.
h_{t-1} = prev time step's hidden state → compressed memory.
$\tilde{h}_t = \tanh(x_t \cdot w_x + (h_{t-1} * r_t)\, w_h + b)$
$h_t = \tilde{h}_t * z_t + (1 - z_t)\, h_{t-1}$ → what happened in this time step.
new information to write
information to erase ? → Controlled by single gate.
so, Updated memory = $h_t = z_t * \tilde{h}_t + (1 - z_t)\, h_{t-1}$
problem, $h_t = z_t(\tilde{h}_t - h_{t-1}) + h_{t-1}$
$\Rightarrow h_t = z_t(\tilde{h} - h_{t-1}) + h_{t-1}$
for $z_t = 0.00001$ → Do not update old memory, Keep new memory as old memory, $h_t = 0.9999\, h_{t-1}$
Only 99.99% memory is retained.
for k^{th} time step, $h_t^{k^{th}} = (0.9999)^k\, h_{t-1}$
memory decay ⇒ Memory loss.
change-1 :- ⇒ No memory decay
⇒ 100% old memory
Retention must be possible ⇒ freeze the memory
⇒ Gradient must not vanish while BPTT
Change the design so that no memory loss.
Why Gradient Vanishes or, explode?
for Example, Consider 2-layered GRU
layer-1 (t=1) → layer-2 (t=2) → logits, hidden state (Compressed Memory)
→ each layer Computes,
$r_t = \sigma(x_t w_{rx} + h_{t-1} \cdot w_{rL} + b_r)$
$z_t = \sigma(x_t w_{zx} + h_{t-1} \cdot w_{zh} + b_z)$
h_{t-1} → prev time step's hidden state
$\tilde{h}_t = \tanh(x_t \cdot w_x + (h_{t-1} * r_t)\, w_h + b)$
$h_t = z_t * \tilde{h}_t + (1 - z_t)\, h_{t-1}$ → passed to next layer
at t=0, assume: h_0
1st layer (t=1)
$r_1 = \sigma(x_1 w_{r_1 x} + h_0 w_{r_1 h} + b_{r_1})$
$z_1 = \sigma(x_1 w_{z_1 x} + h_0 w_{z_1 h} + b_{z_1})$
$\tilde{h}_1 = \tanh(x_1 \cdot w_{1x} + (h_0 * r_1)\, w_{1h} + b_1)$
$h_1 = z_1 * \tilde{h}_1 + (1 - z_1)\, h_0$
2nd layer (t=2)
$r_2 = \sigma(x_2 w_{r_2 x} + h_1 w_{r_2 h} + b_{r_2})$
$z_2 = \sigma(x_2 w_{z_2 x} + h_1 w_{z_2 h} + b_{z_2})$
$\tilde{h}_2 = \tanh(x_2 \cdot w_{2x} + (h_1 * r_2)\, w_{2h} + b_2)$
$h_2 = z_2 * \tilde{h}_2 + (1 - z_2)\, h_1$


Trainable Parameters
layer:1
layer:2
#18 trainable Parameters
L → loss
gradient of loss w.r.t trainable Parameters.
#BPTT:-
Let's see
we know that,
weight of reset gate for layer 2 projecting from
We know that, ignore these for now
Let's see
So eventually
So,
Multiplicative Decay
anyways,
if Something > 1 → gradient explode.
if Something < 1
This multiplication is the reason of Gradient explode, or Gradient Vanishing.
Note: This mustn't happen because then, the model's weights and biases will not get update, we can't afford this.
#Change 2:-
1) In new Design Gradient Vanishing / explode mustn't happen
2) This can be achieved by introducing linear function's memory update so that if gradient passes then it will be unaffected.
⇒ Gradient free memory update, f(x) = x ⇒ f'(x) = 1
linear gradient

#LSTMs

By Implementing Those changes Discussed in GRUs Architecture, we can Implement new Architecture such that,

1) No memory Decay will happen
2) Short old memory Retention.
3) Gradient mostly Vanish / explode

In GRU, we have one gate that handled both the removal and Updation of old memory and new memory respectively, we now need seperate gates for that,

so,

$h_t = z_t * \tilde{h}_t + (1 - z_t) * h_{t-1}$

⇒ no Decay ⇒ $\rightarrow h_t = h_{t-1}$ (must be possible)

$\rightarrow h_t = h_{t-1} + \Delta t$ → new information.

⇒ must be linear (No Gradient Decay/explode)

⇒ $h_t = f(h_{t-1} + \Delta t)_t$; where $f_t \rightarrow$ linear function.

for Control over memory introduce gates,

$c_t = f_t * c_{t-1} + i_t * g_t$

- $o_t \rightarrow$ output gate
- $i_t \rightarrow$ input gate
- $g_t \rightarrow$ Candidate gate
- $c_{t-1} \rightarrow$ prev long term memory
- $f_t \rightarrow$ forget gate

and expose,

Updated memory ← $o_t * \tanh(c_t)$

⇒
- $f_t = \sigma(x_t \cdot W_{fx} + h_{t-1} W_{fh} + b_f)$ → how much to forget from c_{t-1}
- $i_t = \sigma(x_t \cdot W_{ix} + h_{t-1} W_{ih} + b_i)$ → How strongly we need it to update old memory
- $g_t = \tanh(x_t \cdot W_{gx} + h_{t-1} W_{gh} + b_g)$ → what to write in old memory, as a new information
- $c_t = f_t * c_{t-1} + i_t * g_t$ → new information → "Long Term Memory"
- $o_t = \sigma(x_t \cdot W_{ox} + h_{t-1} W_{oh} + b_o)$
- $h_t = o_t * \tanh(c_t)$ → "Short Term Memory"

$y = W \cdot h_t + b$.

Note: It is necessary to keep passing short term Memory as well as long Term memory to next time step in Training as well as Inference mode.

C_t → What happened before 100th time step | h_t → what happened before '2,3' time step

→ GRU$_x$, GRU$_f$, GRU$_y$

→ Astra$_x$, Astra$_f$, Astra$_y$

→ Scribe$_x$, Scribe$_f$, Scribe$_y$

→ Levanthian

Gradient Analysis

Let's take 2-Layer Example and unfold the LSTM for BPTT

layer-1

$f_1 = \sigma(W_{f_1} \cdot x_1 + U_{f_1} \cdot h_0 + b_{f_1})$

$i_1 = \sigma(W_{i_1} \cdot x_1 + U_{i_1} \cdot h_0 + b_{i_1})$

$g_1 = \sigma(W_{g_1} \cdot x_1 + U_{g_1} \cdot h_0 + b_{g_1})$

$O_1 = \sigma(W_{O_1} \cdot x_1 + U_{O_1} \cdot h_0 + b_{O_1})$

$C_1 = f_1 * c_0 + i_1 * g_1$

$h_1 = O_1 * \tanh(C_1)$

layer-2

$f_2 = \sigma(W_{f_2} \cdot x + U_{f_2} \cdot h_1 + b_{f_2})$

$i_2 = \sigma(W_{i_2} \cdot x + U_{i_2} \cdot h_1 + b_{i_2})$

$g_2 = \sigma(W_{g_2} \cdot x + U_{g_2} \cdot h_1 + b_{g_2})$

$O_2 = \sigma(W_{O_2} \cdot x + U_{O_2} \cdot h_1 + b_{O_2})$

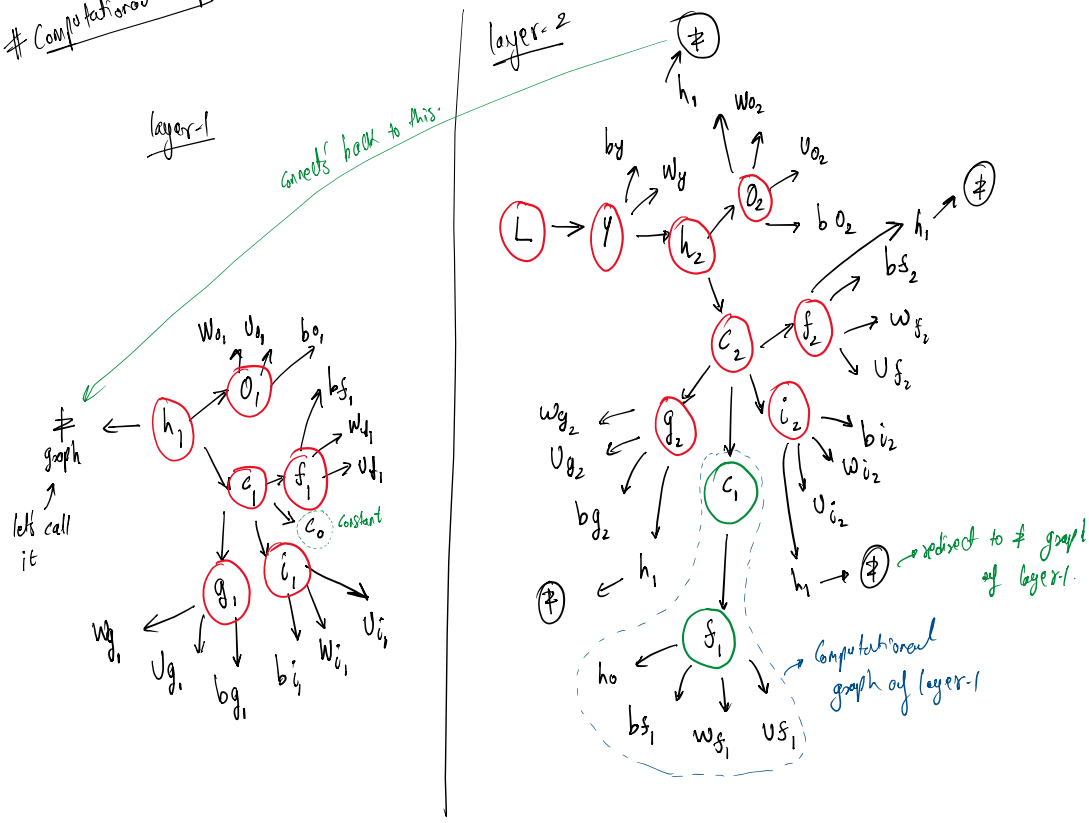
$C_2 = f_2 * c_1 + i_2 * g_2$

$h_2 = O_2 * \tanh(C_2)$

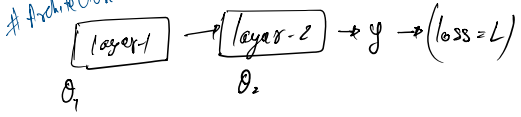
$y = W_y \cdot h_2 + b_y$

$L = \mathcal{L}(y)$

Computational Graph



Architecture



θ_1: $\{W_{f_1}, U_{f_1}, b_{f_1}, W_{g_1}, U_{g_1}, b_{g_1}, W_{o_1}, b_{o_1}, U_{o_1}, W_{i_1}, b_{i_1}, U_{i_1}\}$

θ_2: $\{W_{f_2}, U_{f_2}, b_{f_2}, W_{g_2}, U_{g_2}, b_{g_2}, W_{o_2}, b_{o_2}, U_{o_2}, W_{i_2}, b_{i_2}, U_{i_2}\}$

to optimize W_{f_1}

$$\frac{\partial L}{\partial W_{f_1}} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial h_2} \cdot \frac{\partial h_2}{\partial c_2} \cdot \frac{\partial c_2}{\partial c_1} \cdot \frac{\partial c_1}{\partial f_1} \cdot \frac{\partial f_1}{\partial W_{f_1}}$$

but, $c_2 = f_2 * c_1 + i_2 * g_2$

$$\frac{\partial c_2}{\partial c_1} = f_2 \Rightarrow \frac{\partial c_2}{\partial c_1} \propto f_2$$

we get this by Design.

$$\Rightarrow \frac{\partial L}{\partial W_{f_1}} \propto f_2$$

now $f_2 \rightarrow$ sigmoid func$^n \rightarrow$ $0 < f_2 < 1$ $\rightarrow$ Stabilized gradient flow.

↳ so model can now learn to stablize this gradient flow.

Generally, # Constant Error Carousel

$$\frac{\partial c_t}{\partial c_{t-k}} = \prod_{j=t-k+1}^{t} f_t \; ; \quad 0 < f_t < 1$$

further Analysis

$$c_t = f_t * c_{t-1} + i_t * g_t$$

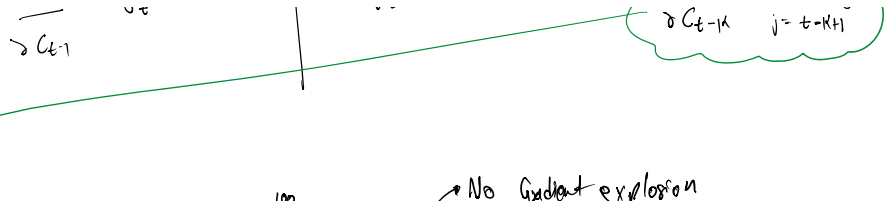
$$\Rightarrow \frac{\partial c_t}{\partial c_{t-1}} = f_t$$

put $c_{t-1} = f_{t-1} * c_{t-2} + i_{t-1} * g_{t-1}$

$$\Rightarrow c_t = f_t f_{t-1} * c_{t-2} + f_t * i_{t-1} * g_{t-1} + i_t * g_t$$

$$\frac{\partial c_t}{\partial c_{t-2}} = f_t * f_{t-1} \Rightarrow \frac{\partial c_t}{\partial c_{t-1}} = \prod_{j=t-k+1}^{t} f_j$$

error at time 't' is sent backwards.



- if $f_t = 0.999$
- Sequence length = 100

$$\frac{\partial C_t}{\partial C_{t-k}} = (0.999)^{100} \approx 0.9047 < 1$$

→ No Gradient explosion
→ But Gradient will eventually shrink and it's Delayed.
→ if any forget gate's output is Small then it will effect and vanishes the Gradient.

Compression of Information in LSTM's

$$C_t = f_t * C_{t-1} + i_t * g_t$$

How much old information to keep
How much new info to add.

imagine: $C_{t-1} = \begin{bmatrix} 0.9 & 0.586 \\ -0.43 & 0.12 \end{bmatrix}$ → old memory

we need to update this old memory by adding new memory

$$C_t = f_t * \begin{bmatrix} 0.9 & 0.58 \\ -0.43 & 0.12 \end{bmatrix} + i_t * g_t$$

then add new information

$0 < f_t < 1$ → Compress old memory

So Basically, at every layer of LSTM.

new Memory = (Compress old memory) + new information

Issue-2:- Each time step sees the Compressed version of Earlier time step. → C_{t+k} is the Compressed Version of C_{t+k-1}

⇒ Every information is Compressed.

Temporal Locality

→ Decision about memory are made when the information arrives, not when it is needed later. → model must decide now, whether some information will be useful far in future

→ LSTM must be able to Decide the importance of current information before seeing the future Context

$$\begin{aligned} f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\ i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\ g_t &= \tanh(W_g x_t + U_g h_{t-1} + b_g) \\ c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\ o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \\ h_t &= o_t \odot \tanh(c_t) \end{aligned}$$

→ This is where the model Decides whether a piece of information is useful or not, but it is only able to see current input

⇓

There is no later Revisions.

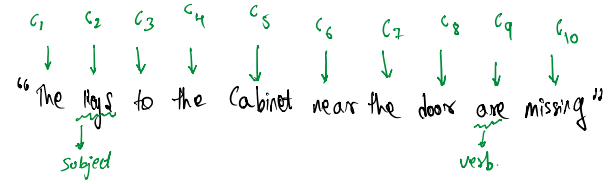
C_0 → initial Memory.

C_1 C_2 C_3 C_4 C_5 C_6 C_7 C_8 C_9 C_{10}

[redacted] There is no later Revisions.

C_0 → initial Memory.

Examples:-



at $(t=2)$ $C_2 = (C_1 * \delta_1 + i_1 * g_1)$ → must decide How strongly we need to store information related to 'keys' assume model decides to allocate δ_1 as 0.01 → (very small) ⇝ (Not important)

but, later at $t=9$ we need a Subject associated to verb 'are'.

So can we go back in time and change the importance of 'keys' ? ⇒ No.

issue-3 :- Right now, $C_t \to C_{t+1} \longrightarrow C_{t+2} \longrightarrow C_{t+3} \text{ - - - - } C_{t+k-1}$

if Something happened at time step '$t+3$' then it cannot be changed ⇒
- No mechanism to reopen any C_{t+k}
- No access to earlier time step
- each time steps sees the Compressed version of earlier timesteps

⇒ * [Memory updation is Local in time, but requirements are Global in time]