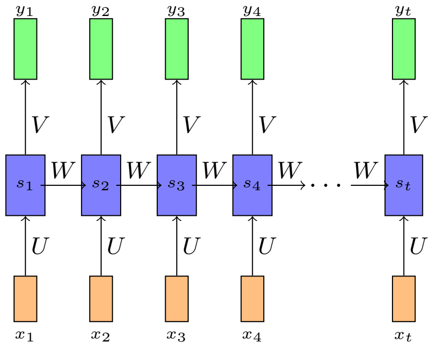
# CS7015 (Deep Learning) : Lecture 15

## Long Short Term Memory Cells (LSTMs), Gated Recurrent Units (GRUs)

Mitesh M. Khapra

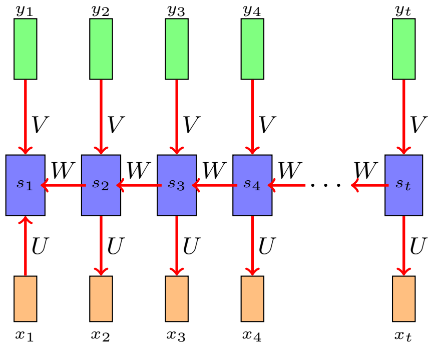
Department of Computer Science and Engineering
Indian Institute of Technology Madras

## Module 15.1: Selective Read, Selective Write, Selective Forget - The Whiteboard Analogy



- The state ($s_i$) of an RNN records information from all previous time steps
- At each new timestep the old information gets morphed by the current input
- One could imagine that after $t$ steps the information stored at time step $t - k$ (for some $k < t$) gets completely morphed

  so much that it would be impossible to extract the original information stored at time step $t - k$



- A similar problem occurs when the information flows backwards (backpropagation)
- It is very hard to assign the responsibility of the error caused at time step $t$ to the events that occurred at time step $t - k$
- This responsibility is of course in the form of gradients and we studied the problem in backward flow of gradients
- We saw a formal argument for this while discussing vanishing gradients

- Let us see an analogy for this
- We can think of the state as a fixed size memory
- Compare this to a fixed size white board that you use to record information
- At each time step (periodic intervals) we keep writing something to the board
- Effectively at each time step we morph the information recorded till that time point
- After many timesteps it would be impossible to see how the information at time step $t-k$ contributed to the state at timestep $t$

- Continuing our whiteboard analogy, suppose we are interested in deriving an expression on the whiteboard
- We follow the following strategy at each time step
- Selectively write on the board
- Selectively read the already written content
- Selectively forget (erase) some content
- Let us look at each of these in detail

$$a = 1 \quad b = 3 \quad c = 5 \quad d = 11$$

**Compute** $ac(bd + a) + ad$

Say "board" can have only 3 statements at a time.

1. $ac$
2. $bd$
3. $bd + a$
4. $ac(bd + a)$
5. $ad$
6. $ac(bd + a) + ad$

$$ac = 5$$
$$bd = 33$$

## Selective write

- There may be many steps in the derivation but we may just skip a few
- In other words we select what to **write**

$$a = 1 \quad b = 3 \quad c = 5 \quad d = 11$$

**Compute** $ac(bd + a) + ad$

Say "board" can have only 3 statements at a time.

1. $ac$
2. $bd$
3. $bd + a$
4. $ac(bd + a)$
5. $ad$
6. $ac(bd + a) + ad$

$$\begin{aligned} ac &= 5 \\ bd &= 33 \\ bd + a &= 34 \end{aligned}$$

## Selective read

- While writing one step we typically read some of the previous steps we have already written and then decide what to write next
- For example at Step 3, information from Step 2 is important
- In other words we select what to **read**

$$a = 1 \quad b = 3 \quad c = 5 \quad d = 11$$

**Compute** $ac(bd + a) + ad$

Say "board" can have only 3 statements at a time.

1. $ac$
2. $bd$
3. $bd + a$
4. $ac(bd + a)$
5. $ad$
6. $ac(bd + a) + ad$

$$ac = 5$$
$$bd = 33$$
$$bd + a = 34$$

## Selective forget

- Once the board is full, we need to delete some obsolete information
- But how do we decide what to delete? We will typically delete the least useful information
- In other words we select what to **forget**

$$a = 1 \quad b = 3 \quad c = 5 \quad d = 11$$

**Compute** $ac(bd + a) + ad$

Say "board" can have only 3 statements at a time.

1. $ac$
2. $bd$
3. $bd + a$
4. $ac(bd + a)$
5. $ad$
6. $ac(bd + a) + ad$

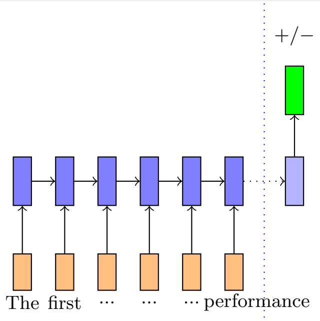
$$ad + ac(bd + a) = 181$$
$$ac(bd + a) = 170$$
$$ad = 11$$

- There are various other scenarios where we can motivate the need for selective write, read and forget
- For example, you could think of our brain as something which can store only a finite number of facts
- At different time steps we selectively read, write and forget some of these facts
- Since the RNN also has a finite state size, we need to figure out a way to allow it to selectively read, write and forget

## Module 15.2: Long Short Term Memory(LSTM) and Gated Recurrent Units(GRUs)

## Questions

- Can we give a concrete example where RNNs also need to selectively read, write and forget ?
- How do we convert this intuition into mathematical equations ? We will see this over the next few slides

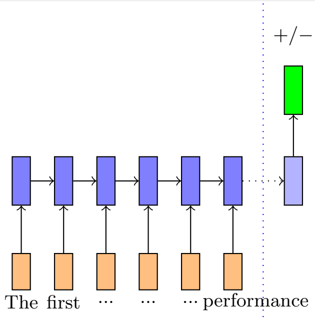


**Review:** The first half of the movie was dry but the second half really picked up pace. The lead actor delivered an amazing performance

- Consider the task of predicting the sentiment (positive/negative) of a review
- RNN reads the document from left to right and after every word updates the state
- By the time we reach the end of the document the information obtained from the first few words is completely lost
- Ideally we want to
  - **forget** the information added by stop words (a, the, etc.)
  - **selectively read** the information added by previous sentiment bearing words (awesome, amazing, etc.)
  - **selectively write** new information from the current word to the state

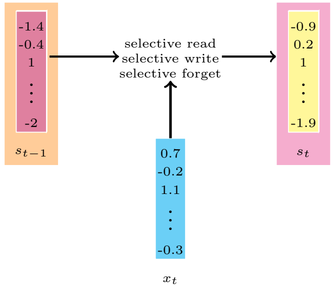
## Questions

- Can we give a concrete example where RNNs also need to selectively read, write and forget ?
- How do we convert this intuition into mathematical equations ?

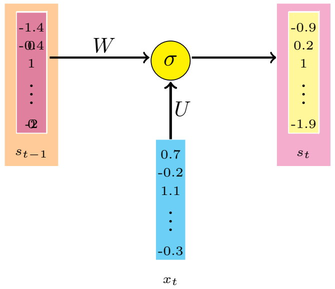


**Review:** The first half of the movie was dry but the second half really picked up pace. The lead actor delivered an amazing performance

- Recall that the blue colored vector ($s_t$) is called the state of the RNN
- It has a finite size ($s_t \in \mathbb{R}^n$) and is used to store all the information upto timestep $t$
- This state is analogous to the whiteboard and sooner or later it will get overloaded and the information from the initial states will get morphed beyond recognition
- **Wishlist:** selective write, selective read and selective forget to ensure that this finite sized state vector is used effectively



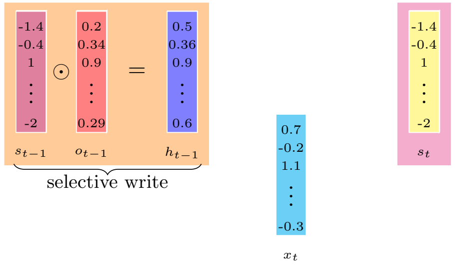
- Just to be clear, we have computed a state $s_{t-1}$ at timestep $t-1$ and now we want to overload it with new information $(x_t)$ and compute a new state $(s_t)$
- While doing so we want to make sure that we use selective write, selective read and selective forget so that only important information is retained in $s_t$
- We will now see how to implement these items from our wishlist



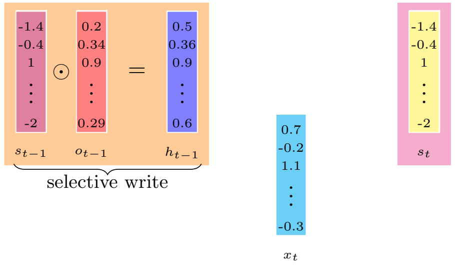
## Selective Write

- Recall that in RNNs we use $s_{t-1}$ to compute $s_t$
  $s_t = \sigma(Ws_{t-1} + Ux_t)$ *(ignoring bias)*
- But now instead of passing $s_{t-1}$ as it is to $s_t$ we want to pass (write) only some portions of it to the next state
- In the strictest case our decisions could be binary (for example, retain 1st and 3rd entries and delete the rest of the entries)
- But a more sensible way of doing this would be to assign a value between 0 and 1 which determines what fraction of the current state to pass on to the next state

## Selective Write



- We introduce a vector $o_{t-1}$ which decides what fraction of each element of $s_{t-1}$ should be passed to the next state
- Each element of $o_{t-1}$ gets multiplied with the corresponding element of $s_{t-1}$
- Each element of $o_{t-1}$ is restricted to be between 0 and 1
- But how do we compute $o_{t-1}$? How does the RNN know what fraction of the state to pass on?

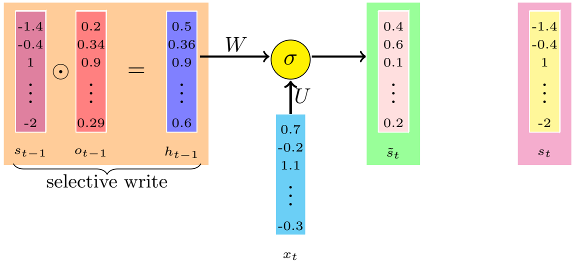


## Selective Write

- Well the RNN has to learn $o_{t-1}$ along with the other parameters $(W, U, V)$
- We compute $o_{t-1}$ and $h_{t-1}$ as

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

- The parameters $W_o, U_o, b_o$ need to be learned along with the existing parameters $W, U, V$
- The sigmoid (logistic) function ensures that the values are between 0 and 1
- $o_t$ is called the output gate as it decides how much to pass (write) to the next time step

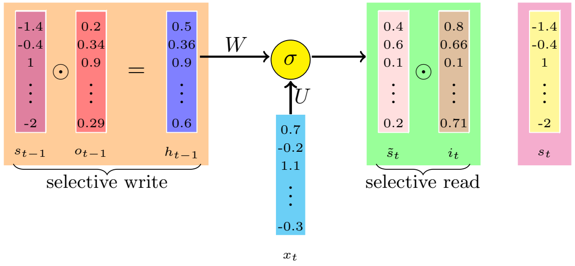


## Selective Read

- We will now use $h_{t-1}$ to compute the new state at the next time step
- We will also use $x_t$ which is the new input at time step t

$$\tilde{s}_t = \sigma(Wh_{t-1} + Ux_t + b)$$

- Note that $W, U$ and $b$ are similar to the parameters that we used in RNN (for simplicity we have not shown the bias b in the figure)

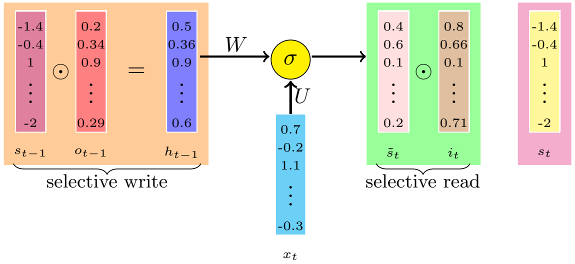


## Selective Read

- $\tilde{s}_t$ thus captures all the information from the previous state ($h_{t-1}$) and the current input $x_t$
- However, we may not want to use all this new information and only selectively **read** from it before constructing the new cell state $s_t$
- To do this we introduce another gate called the input gate

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

- and use $i_t \odot \tilde{s}_t$ as the selectively read state information



- So far we have the following

**Previous state:**

$s_{t-1}$

**Output gate:**

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

**Selectively Write:**

$h_{t-1} = o_{t-1} \odot \sigma(s_{t-1})$

**Current (temporary) state:**

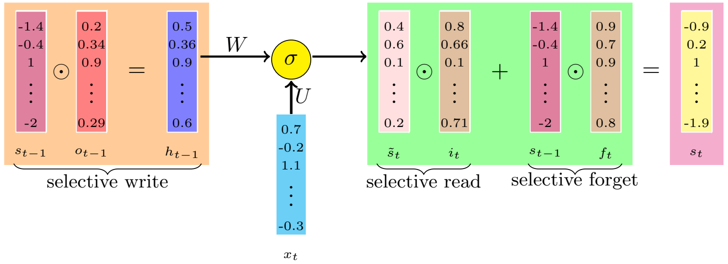
$\tilde{s}_t = \sigma(W h_{t-1} + U x_t + b)$

**Input gate:**

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

**Selectively Read:**

$i_t \odot \tilde{s}_t$



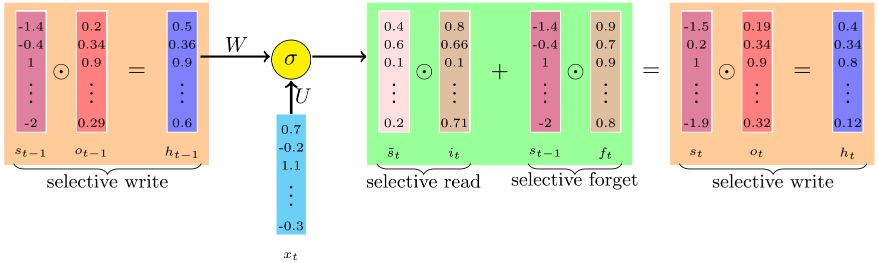
## Selective Forget

- How do we combine $s_{t-1}$ and $\tilde{s}_t$ to get the new state
- Here is one simple (but effective) way of doing this:

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

- But we may not want to use the whole of $s_{t-1}$ but forget some parts of it
- To do this we introduce the forget gate

$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$$
$$s_t = f_t \odot s_{t-1} + i_t \odot \tilde{s}_t$$



- 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:**

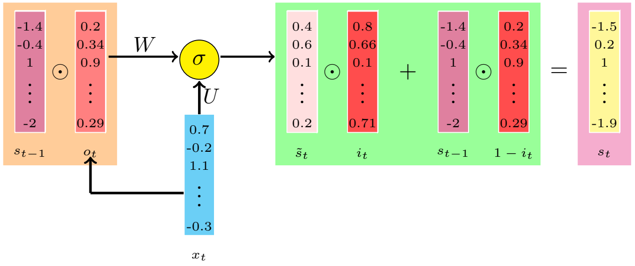
$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)$$
$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$$
$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$$

**States:**

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

## Note

- LSTM has many variants which include different number of gates and also different arrangement of gates
- The one which we just saw is one of the most popular variants of LSTM
- Another equally popular variant of LSTM is Gated Recurrent Unit which we will see next



The full set of equations for GRUs

**Gates:**

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

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

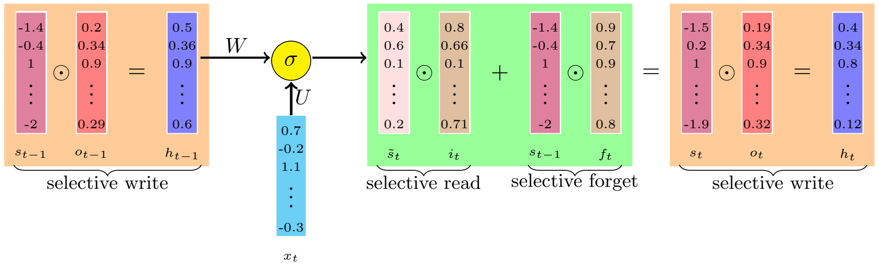
**States:**

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

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

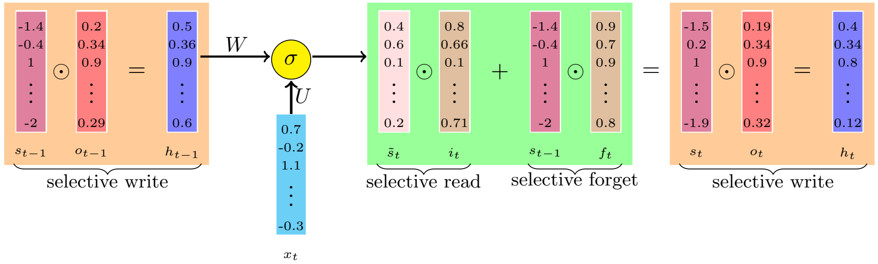
- No explicit forget gate (the forget gate and input gates are tied)
- The gates depend directly on $s_{t-1}$ and not the intermediate $h_{t-1}$ as in the case of LSTMs

## Module 15.3: How LSTMs avoid the problem of vanishing gradients



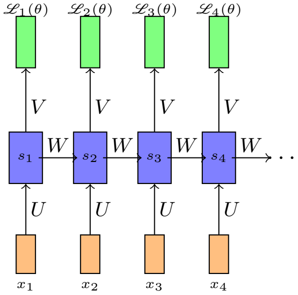
## Intuition

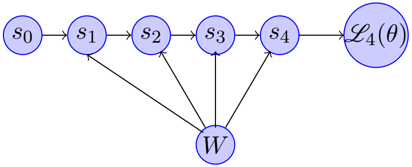
- During forward propagation the gates control the flow of information
- They prevent any irrelevant information from being written to the state
- Similarly during backward propagation they control the flow of gradients
- It is easy to see that during backward pass the gradients will get multiplied by the gate



- If the state at time $t-1$ did not contribute much to the state at time $t$ (i.e., if $\|f_t\| \to 0$ and $\|o_{t-1}\| \to 0$) then during backpropagation the gradients flowing into $s_{t-1}$ will vanish
- But this kind of a vanishing gradient is fine (since $s_{t-1}$ did not contribute to $s_t$ we don't want to hold it responsible for the crimes of $s_t$)
- The key difference from vanilla RNNs is that the flow of information and gradients is controlled by the gates which ensure that the gradients vanish only when they should (i.e., when $s_{t-1}$ didn't contribute much to $s_t$)

We will now see an illustrative proof of how the gates control the flow of gradients

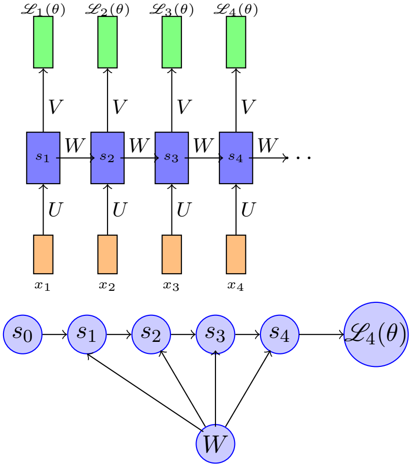




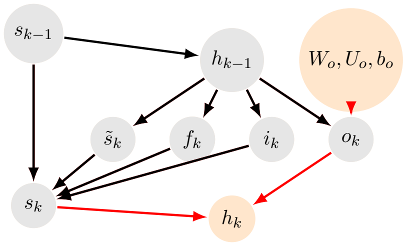
- Recall that RNNs had this multiplicative term which caused the gradients to vanish

$$\frac{\partial \mathscr{L}_t(\theta)}{\partial W} = \frac{\partial \mathscr{L}_t(\theta)}{\partial s_t} \sum_{k=1}^{t} \prod_{j=k}^{t-1} \frac{\partial s_{j+1}}{\partial s_j} \frac{\partial^+ s_k}{\partial W}$$

- In particular, if the loss at $\mathscr{L}_4(\theta)$ was high because $W$ was not good enough to compute $s_1$ correctly then this information will not be propagated back to $W$ as the gradient $\frac{\partial \mathscr{L}_t(\theta)}{\partial W}$ along this long path will vanish



- In general, the gradient of $\mathscr{L}_t(\theta)$ w.r.t. $\theta_i$ vanishes when the gradients flowing through **each and every path** from $L_t(\theta)$ to $\theta_i$ vanish.
- On the other hand, the gradient of $\mathscr{L}_t(\theta)$ w.r.t. $\theta_i$ explodes when the gradient flowing through **at least one path** explodes.
- We will first argue that in the case of LSTMs there exists at least one path through which the gradients can flow effectively (and hence no vanishing gradients)

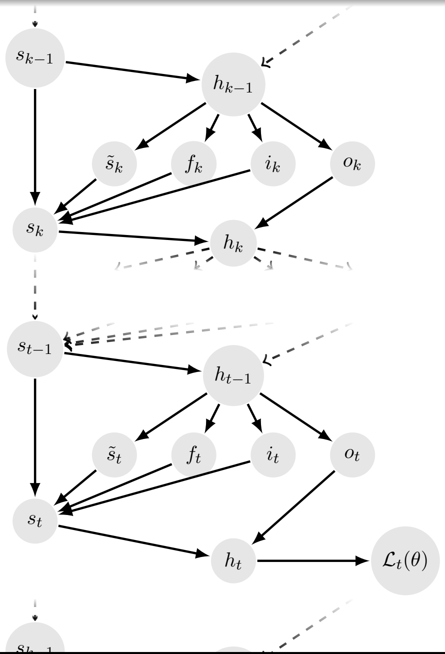


- We will start with the dependency graph involving different variables in LSTMs
- Starting with the states at timestep $k-1$

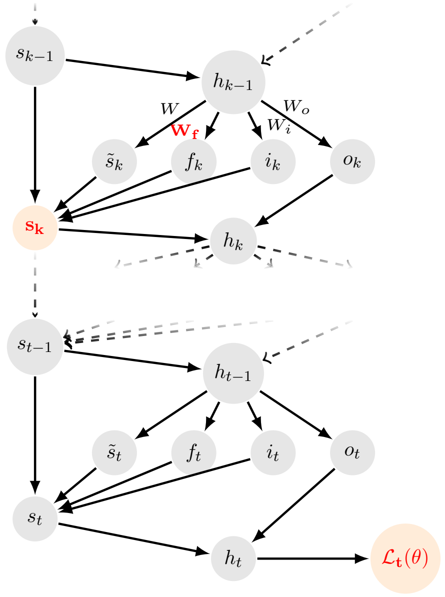
$$o_k = \sigma(W_o h_{k-1} + U_o x_k + b_o)$$

- For simplicity we will omit the parameters for now and return back to them later

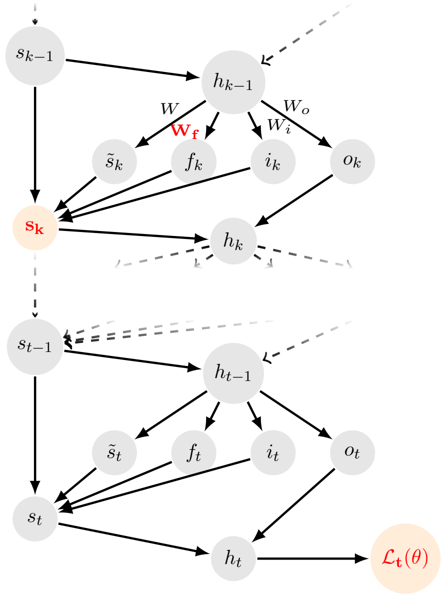
$$\begin{aligned} i_k &= \sigma(W_i h_{k-1} + U_i x_k + b_i) \\ f_k &= \sigma(W_f h_{k-1} + U_f x_k + b_f) \\ \tilde{s_k} &= \sigma(W h_{k-1} + U x_k + b) \\ s_k &= f_k \odot s_{k-1} + i_k \odot \tilde{s_k} \\ h_k &= o_k \odot \sigma(s_k) \end{aligned}$$



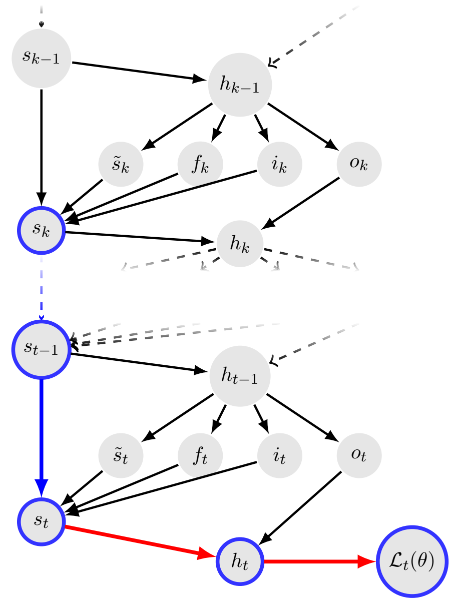
- Starting from $h_{k-1}$ and $s_{k-1}$ we have reached $h_k$ and $s_k$
- And the recursion will now continue till the last timestep
- For simplicity and ease of illustration, instead of considering the parameters ($W$, $W_o$, $W_i$, $W_f$, $U$, $U_o$, $U_i$, $U_f$) as separate nodes in the graph we will just put them on the appropriate edges. (We show only a few



- For example, we are interested in knowing if the gradient flows to $W_f$ through $s_k$
- In other words, if $\mathscr{L}_t(\theta)$ was high because $W_f$ failed to compute an appropriate value for $s_k$ then this information should flow back to $W_f$ through the gradients
- We can ask a similar question about the other parameters (for example, $W_i$, $W_o$, $W$, etc.)
- How does LSTM ensure that this gradient does not vanish even at arbitrary time steps? Let us see



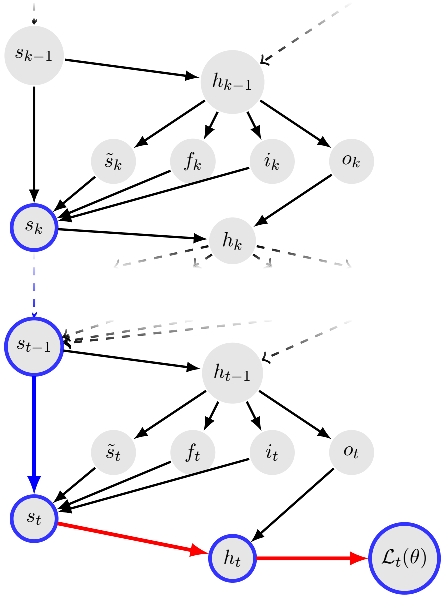
- It is sufficient to show that $\frac{\partial \mathcal{L}_t(\theta)}{\partial s_k}$ does not vanish (because if this does not vanish we can reach $W_f$ through $s_k$)
- First, we observe that there are multiple paths from $\mathscr{L}_t(\theta)$ to $s_k$ (you just need to reverse the direction of the arrows for backpropagation)
- For example, there is one path through $s_{k+1}$, another through $h_k$
- Further, there are multiple paths to reach to $h_k$ itself (as should be obvious from the number of outgoing arrows from $h_k$)
- So at this point just convince yourself that there are many paths from $\mathscr{L}_t(\theta)$ to $s_k$



- Consider one such path (highlighted) which will contribute to the gradient
- Let us denote the gradient along this path as $t_0$

$$t_0 = \frac{\partial \mathscr{L}_t(\theta)}{\partial h_t} \frac{\partial h_t}{\partial s_t} \frac{\partial s_t}{\partial s_{t-1}} \cdots \frac{\partial s_{k+1}}{\partial s_k}$$

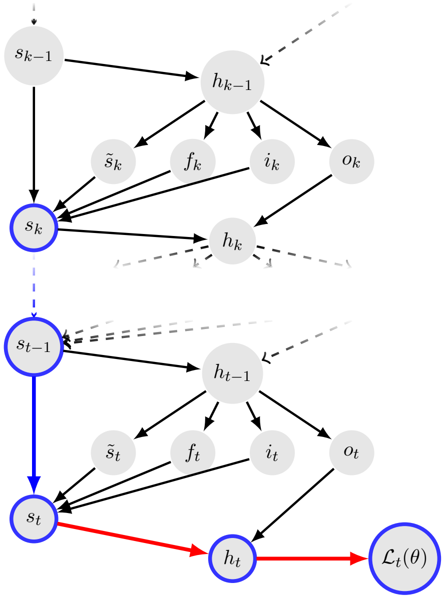
- The first term $\frac{\partial \mathscr{L}_t(\theta)}{\partial h_t}$ is fine and it doesn't vanish ($h_t$ is directly connected to $\mathscr{L}_t(\theta)$ and there are no intermediate nodes which can cause the gradient to vanish)
- We will now look at the other terms $\frac{\partial h_t}{\partial s_t}$ $\frac{\partial s_t}{\partial s_{t-1}}$ $(\forall t)$



- Let us first look at $\frac{\partial h_t}{\partial s_t}$
- Recall that

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

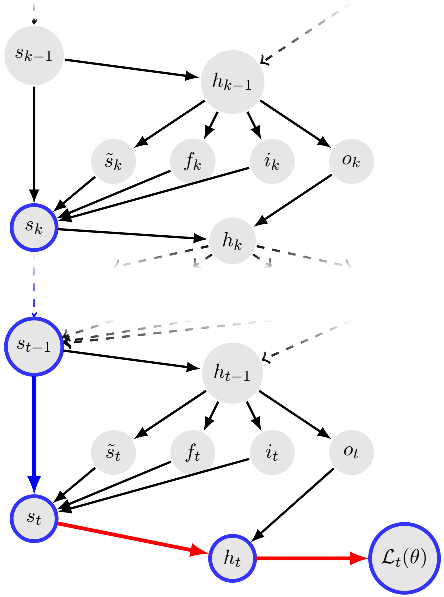
- Note that $h_{ti}$ only depends on $o_{ti}$ and $s_{ti}$ and not on any other elements of $o_t$ and $s_t$
- $\frac{\partial h_t}{\partial s_t}$ will thus be a square diagonal matrix $\in \mathbb{R}^{d \times d}$ whose diagonal will be $o_t \odot \sigma'(s_t) \in \mathbb{R}^d$ (see slide 35 of Lecture 14)
- We will represent this diagonal matrix by $\mathcal{D}(o_t \odot \sigma'(s_t))$



- Now let us consider $\frac{\partial s_t}{\partial s_{t-1}}$
- Recall that

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

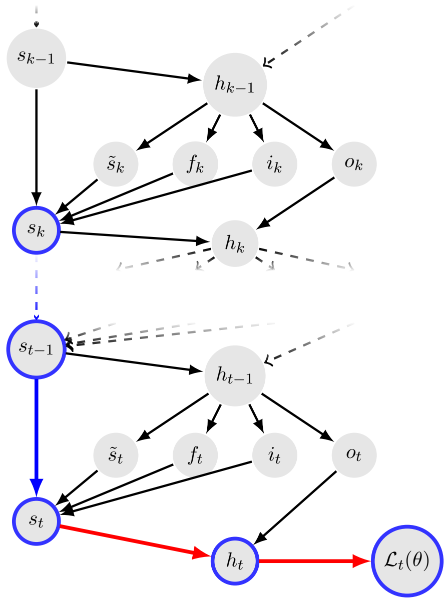
- Notice that $\tilde{s}_t$ also depends on $s_{t-1}$ so we cannot treat it as a constant
- So once again we are dealing with an ordered network and thus $\frac{\partial s_t}{\partial s_{t-1}}$ will be a sum of an explicit term and an implicit term (see slide 37 from Lecture 14)
- For simplicity, let us assume that the gradient from the implicit term vanishes (we are assuming a worst case scenario)
- And the gradient from the explicit term (treating $\tilde{s}_t$ as a constant) is given by $\mathcal{D}(f_t)$



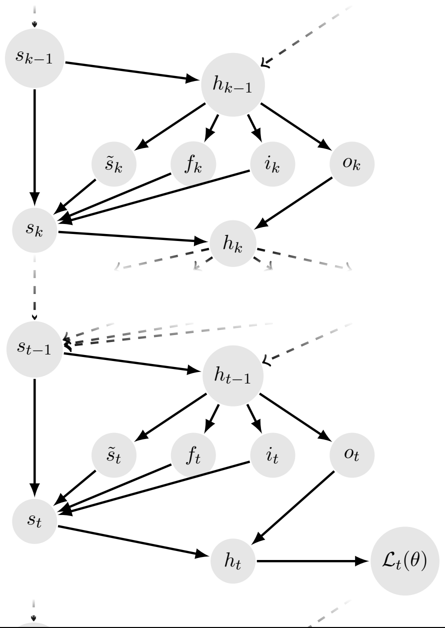
- We now return back to our full expression for $t_0$:

$$t_0 = \frac{\partial \mathscr{L}_t(\theta)}{\partial h_t} \frac{\partial h_t}{\partial s_t} \frac{\partial s_t}{\partial s_{t-1}} \dots \frac{\partial s_{k+1}}{\partial s_k}$$
$$= \mathscr{L}'_t(h_t).\mathcal{D}(o_t \odot \sigma'(s_t))\mathcal{D}(f_t) \dots \mathcal{D}(f_{k+1})$$
$$= \mathscr{L}'_t(h_t).\mathcal{D}(o_t \odot \sigma'(s_t))\mathcal{D}(f_t \odot \dots \odot f_{k+1})$$
$$= \mathscr{L}'_t(h_t).\mathcal{D}(o_t \odot \sigma'(s_t))\mathcal{D}(\odot_{i=k+1}^{t} f_i)$$

- The red terms don't vanish and the blue terms contain a multiplication of the forget gates
- The forget gates thus regulate the gradient flow depending on the explicit contribution of a state ($s_t$) to the next state $s_{t+1}$



- If during forward pass $s_t$ did not contribute much to $s_{t+1}$ (because $f_t \to 0$) then during backpropgation also the gradient will not reach $s_t$
- This is fine because if $s_t$ did not contribute much to $s_{t+1}$ then there is no reason to hold it responsible during backpropgation ($f_t$ does the same regulation during forward pass and backward pass which is fair)
- Thus there exists this one path along which the gradient doesn't vanish when it shouldn't
- And as argued as long as the gradient flows back to $W_f$ through one of the paths ($t_0$) through $s_k$ we are fine !
- Of course the gradient flows back only when required as regulated by $f_i$'s (but let me just say it one last time that *this is fair*)



- Now we will see why LSTMs do not solve the problem of exploding gradients
- We will show a path through which the gradient can explode
- Let us compute one term (say $t_1$) of $\frac{\partial \mathcal{L}_t(\theta)}{\partial h_{k-1}}$ corresponding to the highlighted path

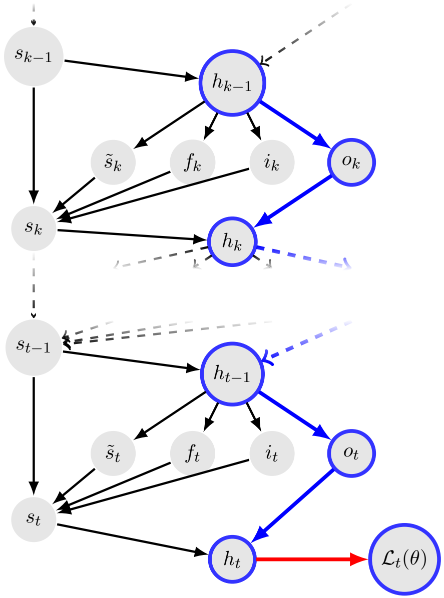
$$t_1 = \frac{\partial \mathcal{L}_t(\theta)}{\partial h_t} \left( \frac{\partial h_t}{\partial o_t} \frac{\partial o_t}{\partial h_{t-1}} \right) \dots \left( \frac{\partial h_k}{\partial o_k} \frac{\partial o_k}{\partial h_{k-1}} \right)$$

$$= \mathcal{L}'_t(h_t) \left( \mathcal{D}(\sigma(s_t) \odot o'_t).W_o \right) \dots$$

$$\left( \mathcal{D}(\sigma(s_k) \odot o'_k).W_o \right)$$

$$\|t_1\| \leq \|\mathcal{L}'_t(h_t)\| \left( \|K\| \|W_o\| \right)^{t-k+1}$$

- Depending on the norm of matrix $W_o$, the gradient $\frac{\partial \mathcal{L}_t(\theta)}{\partial h_{k-1}}$ may explode
- Similarly, $W_i$, $W_f$ and $W$ can also cause the gradients to explode



- So how do we deal with the problem of exploding gradients ?
- One popular trick is to use gradient clipping
- While backpropagating if the norm of the gradient exceeds a certain value, it is scaled to keep its norm within an acceptable threshold*
- Essentially we retain the direction of the gradient but scale down the norm

*Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks." ICML(3)28(2013):1310-1318