Applied Deep Learning

Chapter 6: Recurrent NNs

Ali Bereyhi

ali.bereyhi@utoronto.ca

Department of Electrical and Computer Engineering

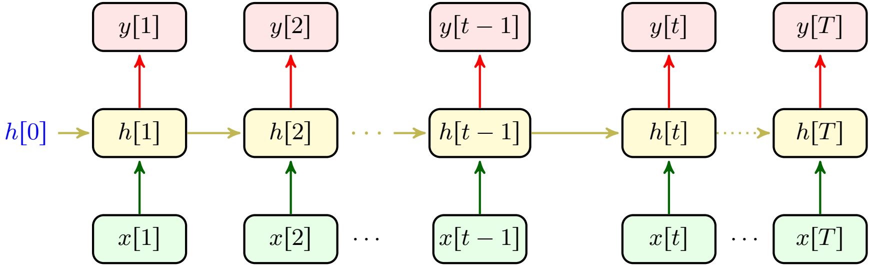
University of Toronto

Fall 2025

Principle of Gating

To understand the idea of *Gating*, let's *get back to our basic RNN*

1. *We start with hidden state* $h[0]$
2. *We have* $h[t] = f(w_1 x[t] + w_{\rm m} h[t-1])$
3. *We have* $y[t] = f(w_2 h[t])$



Looking at $h[t]$ *as memory, we can say we are always updating the memory*

Principle of Gating

Recall our motivating example: *we wanted to predict the next word*

...	$\mathbf{x}[t-6]$	$\mathbf{x}[t-5]$	$\mathbf{x}[t-4]$	$\mathbf{x}[t-3]$	$\mathbf{x}[t-2]$	$\mathbf{x}[t-1]$	$\mathbf{x}[t]$

`... Julia has been nominated to receive Alexander von Humboldt Prize for her`

Should we update the memory all the way from Julia?

- *Obviously No!*
  - ↳ *We should stop updating at Julia, since it is something we should remember*
- *How can we do it? Let's try a thought experiment*

Principle of Gating

Say we access to a sequence $u[t] \in [0,1]$: *assume the following happens*

- ↳ We arrive at *"Julia"* at time t_0, i.e., $x[t_0] \propto$ *"Julia"*
- ↳ At time t_0, we have $u[t_0] = 1$
- ↳ We want to predict at $t+1$
- ↳ From $t_0 + 1$ to t, we have $u[t_0+1] = \cdots = u[t] = 0$

Now, we update our memory like this

1. *We start with hidden state* $h[0]$
2. *We compute* $\tilde{h}[t] = f(w_1 x[t] + w_{\rm m} h[t-1])$
3. *We update* $h[t] = u[t]\tilde{h}[t] + (1 - u[t])\, h[t-1]$
4. *At each time, we have* $y[t] = f(w_2 h[t])$

Let's see what happens to the memory

Principle of Gating

At time t_0, we can say

- *We have* $x[t_0] \propto$ *"Julia" and compute* $\tilde{h}[t_0] = f(w_1 x[t_0] + w_{\mathrm{m}} h[t_0 - 1])$
  - ↳ $\tilde{h}[t_0]$ *has fresh memory about "Julia"*
  - ↳ *Since* $u[t_0] = 1$, *we update as* $h[t_0] = 1 \times \tilde{h}[t_0] + 0 \times h[t_0 - 1] = \tilde{h}[t_0]$
  - ↳ *RNN has a fresh memory about "Julia"*

At the next time, i.e., $t_0 + 1$, we have

- *No matter what* $\tilde{h}[t_0 + 1]$ *is we have* $u[t_0 + 1] = 0$
  - ↳ *We update as* $h[t_0 + 1] = 0 \times \tilde{h}[t_0 + 1] + 1 \times h[t_0] = h[t_0] = \tilde{h}[t_0]$
  - ↳ *We still have a fresh memory about "Julia"*

This repeats from $t_0 + 1$ till t, so at time t

- *No matter what* $\tilde{h}[t]$, *we have* $u[t] = 0$
  - ↳ *We update as* $h[t] = 0 \times \tilde{h}[t] + 1 \times h[t - 1] = h[t - 1] = \ldots = \tilde{h}[t_0]$
  - ↳ *We still have a fresh memory about "Julia"*

Principle of Gating: *Updating via Gates*

$u[t]$ *gates the memory: it decides how much memory we should pass and forget*

- *We update* $h[t] = u[t]\tilde{h}[t] + (1 - u[t])\, h[t-1]$

\+ *How does it help with vanishing gradient?*

– Well! *It is implicitly making skip connections through time*

Recall that *with standard BPTT, we have for* $i = t, t-1, \dots, t_0$

$$\frac{\partial h[i]}{\partial w_{\mathrm{m}}} = \dot{f}\left(z[i]\right)\left(h[i-1] + w_{\mathrm{m}} \frac{\partial h[i-1]}{\partial w_{\mathrm{m}}}\right)$$

But, now we skip multiple time slots, as we have for $i = t, t-1, \dots, t_0$

$$\frac{\partial h[i]}{\partial w_{\mathrm{m}}} = \frac{\partial h[i-1]}{\partial w_{\mathrm{m}}}$$

Principle of Gating: *A Generic Gate*

+ *Sounds inspiring! But, how could you get $u[t]$?*
– Well! *Like what we did the whole time: we learn it!*

Gate

Let $\mathbf{x}[t]$ be input and $\mathbf{h}[t-1]$ be last hidden state: a gate $\mathbf{\Gamma}[t]$ is computed as

$$\mathbf{\Gamma}[t] = \sigma\left(\mathbf{W}_{\Gamma,\mathrm{in}}\mathbf{x}[t] + \mathbf{W}_{\Gamma,\mathrm{m}}\mathbf{h}[t-1] + \mathbf{b}_{\Gamma}\right)$$

where $\sigma(\cdot)$ is sigmoid function, and $\mathbf{W}_{\Gamma,\mathrm{in}}$, $\mathbf{W}_{\Gamma,\mathrm{m}}$ and $\mathbf{b}_{\Gamma}$ are learnable[1]

+ *Why do we use sigmoid function?*
- Simply because *it is between 0 and 1*
+ *What should we set the dimension of $\mathbf{\Gamma}[t]$?*
- Same as the *variable (memory component) that we want to gate*

[1]We are going to drop bias and you all know why!

Practical Gated Architectures

There are various *gated* architectures: *we look into two of them*

1. *Gated Recurrent Unit (GRU)*
2. *Long Short-Term Memory (LSTM)*

Before we start, let's recall their basic RNN counterpart

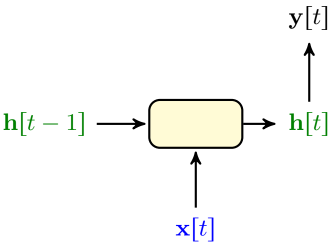
Basic RNN Counterpart

Say we set activation to $f(\cdot)$ $\rightsquigarrow$ we usually set it to $\tanh(\cdot)$

1. *Start with an initial hidden state*
   - ↳ *we can learn* $\mathbf{h}[0]$
2. *Compute memory as* $\mathbf{h}[t] = f(\mathbf{W}_1 \mathbf{x}[t] + \mathbf{W}_{\mathrm{m}} \mathbf{h}[t-1])$
   - ↳ *we can learn* $\mathbf{W}_1$ *and* $\mathbf{W}_{\mathrm{m}}$
3. *Compute output* $\mathbf{y}[t] = f_{\mathrm{out}}(\mathbf{W}_2 \mathbf{h}[t])$ $\rightsquigarrow$ f_{out} and f could be different
   - ↳ *we can learn* $\mathbf{W}_2$

Classical Diagram: *Hidden Layer as Unit*

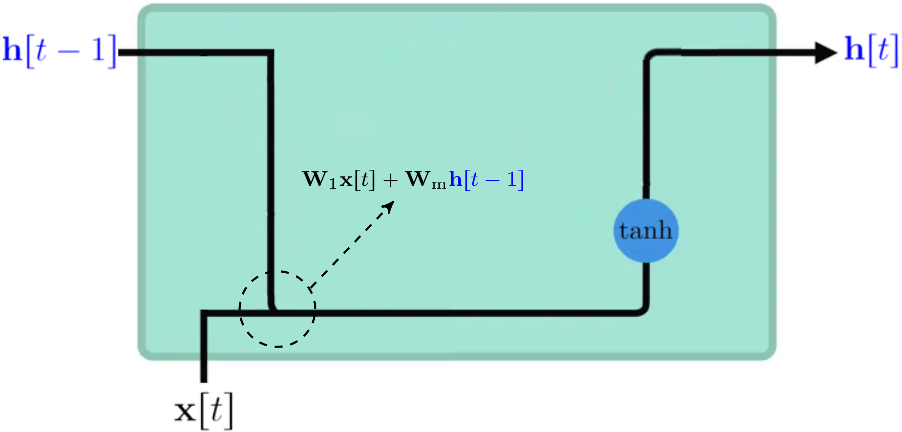
When we study a gated architecture: *it is common to look at the hidden layer as a unit which takes some inputs and returns some outputs*



We are mainly interested on this block: we want to know that given last state and new input

1. *How does this unit update hidden state?*
2. *What components are passed to the next time interval?*
   - ↳ *Here, we have only* $\mathbf{h}[t]$
   - ↳ *But, we may have other components*
     - ↳ *We will see it in LSTM*

Classical Diagram: *Basic RNN*



Practical Gated Architectures: *Gated Recurrent Unit*

Gated Recurrent Unit (GRU)

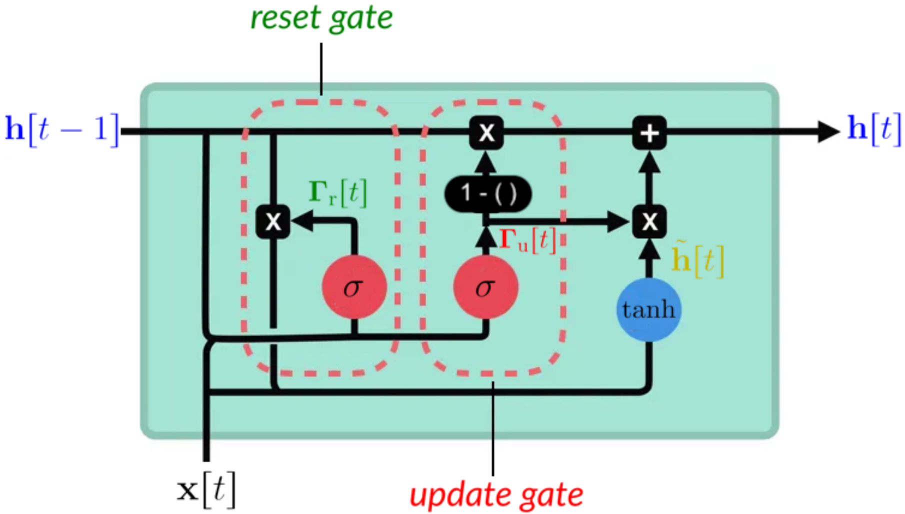
Say we set activation to $f(\cdot)$ ⇝ we usually set it to $\tanh(\cdot)$

1. *Start with an initial hidden state*
2. *Compute update gate* $\boldsymbol{\Gamma}_{\rm u}[t] = \sigma\left(\mathbf{W}_{\rm u,in}\mathbf{x}[t] + \mathbf{W}_{\rm u,m}\mathbf{h}[t-1]\right)$
3. *Compute reset gate* $\boldsymbol{\Gamma}_{\rm r}[t] = \sigma\left(\mathbf{W}_{\rm r,in}\mathbf{x}[t] + \mathbf{W}_{\rm r,m}\mathbf{h}[t-1]\right)$
4. *Compute actual memory* $\tilde{\mathbf{h}}[t] = f(\mathbf{W}_1\mathbf{x}[t] + \mathbf{W}_{\rm m}\boldsymbol{\Gamma}_{\rm r}[t] \odot \mathbf{h}[t-1])$
5. *Update hidden state as* $\mathbf{h}[t] = (1 - \boldsymbol{\Gamma}_{\rm u}[t]) \odot \mathbf{h}[t-1] + \boldsymbol{\Gamma}_{\rm u}[t] \odot \tilde{\mathbf{h}}[t]$
6. *Compute output* $\mathbf{y}[t] = f_{\rm out}(\mathbf{W}_2\mathbf{h}[t])$ ⇝ $f_{\rm out}$ and f could be different

Or we could give $\mathbf{h}[t]$ *to a new layer: for instance a new GRU whose input is* $\mathbf{h}[t]$ *and has its own state*

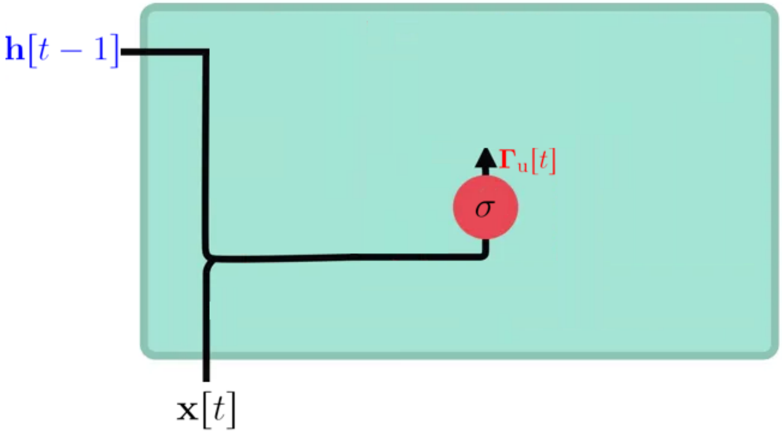
Practical Gated Architectures: *GRU*

This is what's going on in a GRU cell



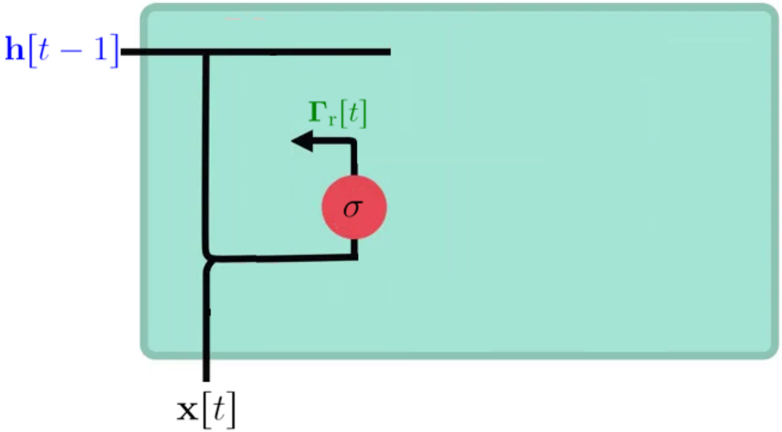
Practical Gated Architectures: *GRU*

Compute update gate $\mathbf{\Gamma}_{\mathrm{u}}[t] = \sigma\left(\mathbf{W}_{\mathrm{u,in}}\mathbf{x}[t] + \mathbf{W}_{\mathrm{u,m}}\mathbf{h}[t-1]\right)$



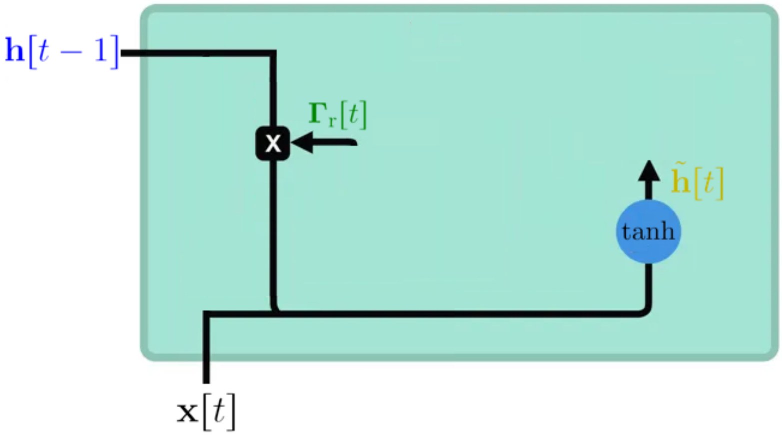
Practical Gated Architectures: *GRU*

Compute reset gate $\mathbf{\Gamma}_{\rm r}[t] = \sigma\left(\mathbf{W}_{\rm r,in}\mathbf{x}[t] + \mathbf{W}_{\rm r,m}\mathbf{h}[t-1]\right)$



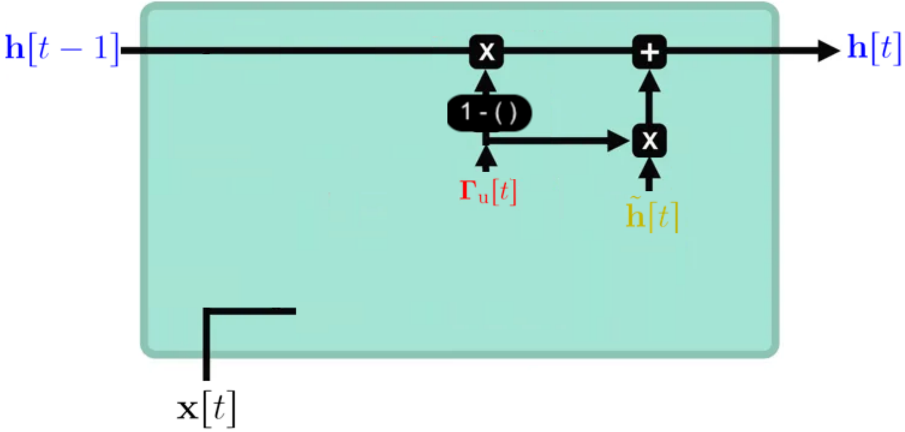
Practical Gated Architectures: *GRU*

Compute actual memory $\tilde{\mathbf{h}}[t] = f(\mathbf{W}_1\mathbf{x}[t] + \mathbf{W}_{\mathrm{m}}\boldsymbol{\Gamma}_{\mathrm{r}}[t] \odot \mathbf{h}[t-1])$



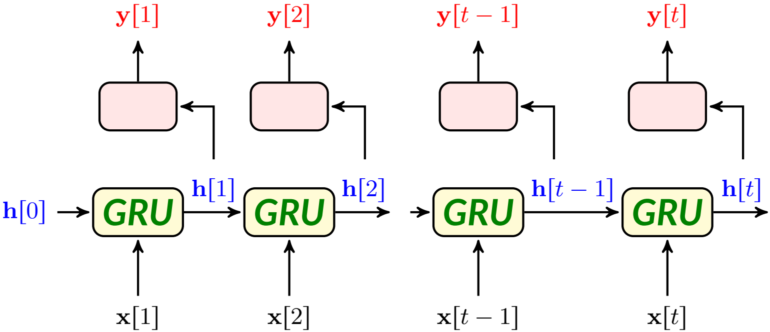
Practical Gated Architectures: *GRU*

Update hidden state as $\mathbf{h}[t] = (1 - \boldsymbol{\Gamma}_{\mathrm{u}}[t]) \odot \mathbf{h}[t-1] + \boldsymbol{\Gamma}_{\mathrm{u}}[t] \odot \tilde{\mathbf{h}}[t]$



GRU: *Forward Pass*

Starting from an initial state: GRU applies the first 5 steps each time



GRU: *Backward Pass*

Say we finished forward pass at time t. We now want to find $\nabla_{\mathbf{W}} \hat{R}$ for some $\mathbf{W}$ that is inside GRU, e.g., $\mathbf{W}_{\mathrm{u,m}}$: we start backpropagating from $\nabla_{\mathbf{y}[t]} \hat{R}$

$$\nabla_{\mathbf{W}} \hat{R} = \nabla_{\mathbf{y}[t]} \hat{R} \circ \nabla_{\mathbf{W}} \mathbf{y}[t]$$

1. *We know* $\mathbf{y}[t] = f_{\mathrm{out}}(\mathbf{W}_2 \mathbf{h}[t])$

$$\nabla_{\mathbf{W}} \mathbf{y}[t] = \nabla_{\mathbf{h}[t]} \mathbf{y}[t] \circ \nabla_{\mathbf{W}} \mathbf{h}[t]$$

2. *We know that* $\mathbf{h}[t] = (1 - \mathbf{\Gamma}_{\mathrm{u}}[t]) \odot \mathbf{h}[t-1] + \mathbf{\Gamma}_{\mathrm{u}}[t] \odot \tilde{\mathbf{h}}[t]$

$$\begin{aligned}\nabla_{\mathbf{W}} \mathbf{h}[t] = &\nabla_{\mathbf{\Gamma}_{\mathrm{u}}[t]} \mathbf{h}[t] \circ \nabla_{\mathbf{W}} \mathbf{\Gamma}_{\mathrm{u}}[t] + \nabla_{\mathbf{h}[t-1]} \mathbf{h}[t] \circ \nabla_{\mathbf{W}} \mathbf{h}[t-1] \\ &+ \nabla_{\tilde{\mathbf{h}}[t]} \mathbf{h}[t] \circ \nabla_{\mathbf{W}} \tilde{\mathbf{h}}[t]\end{aligned}$$

3. $\ldots$

Practical Gated Architectures: *Long Short-Term Memory*

Long Short-Term Memory (LSTM)

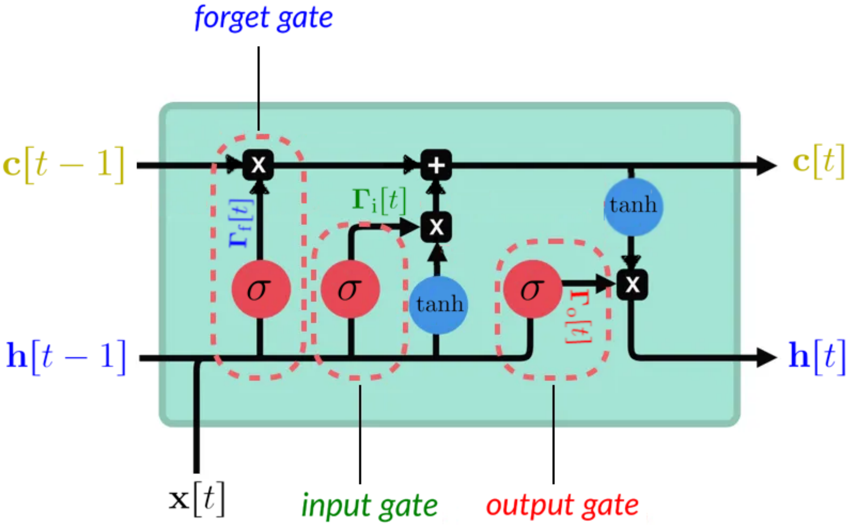
Say we set activation to $f(\cdot)$ $\rightsquigarrow$ we usually set it to $\tanh(\cdot)$

1. *Start with initial hidden state and cell state*
2. *Compute forget gate* $\mathbf{\Gamma}_{\mathrm{f}}[t] = \sigma\left(\mathbf{W}_{\mathrm{f,in}}\mathbf{x}[t] + \mathbf{W}_{\mathrm{f,m}}\mathbf{h}[t-1]\right)$
3. *Compute input gate* $\mathbf{\Gamma}_{\mathrm{i}}[t] = \sigma\left(\mathbf{W}_{\mathrm{i,in}}\mathbf{x}[t] + \mathbf{W}_{\mathrm{i,m}}\mathbf{h}[t-1]\right)$
4. *Compute output gate* $\mathbf{\Gamma}_{\mathrm{o}}[t] = \sigma\left(\mathbf{W}_{\mathrm{o,in}}\mathbf{x}[t] + \mathbf{W}_{\mathrm{o,m}}\mathbf{h}[t-1]\right)$
5. *Compute actual cell state* $\tilde{\mathbf{c}}[t] = f(\mathbf{W}_1\mathbf{x}[t] + \mathbf{W}_{\mathrm{m}}\mathbf{h}[t-1])$
6. *Update cell state as* $\mathbf{c}[t] = \mathbf{\Gamma}_{\mathrm{f}}[t]\mathbf{c}[t-1] + \mathbf{\Gamma}_{\mathrm{i}}[t] \odot \tilde{\mathbf{c}}[t]$
7. *Update hidden state as* $\mathbf{h}[t] = \mathbf{\Gamma}_{\mathrm{o}}[t] \odot f\left(\mathbf{c}[t]\right)$
8. *Compute output* $\mathbf{y}[t] = f_{\mathrm{out}}(\mathbf{W}_2\mathbf{h}[t])$ $\rightsquigarrow$ f_{out} and f could be different

Or we could give $\mathbf{h}[t]$ *to a new layer: for instance a new LSTM whose input is* $\mathbf{h}[t]$ *and has its own hidden and cell states*

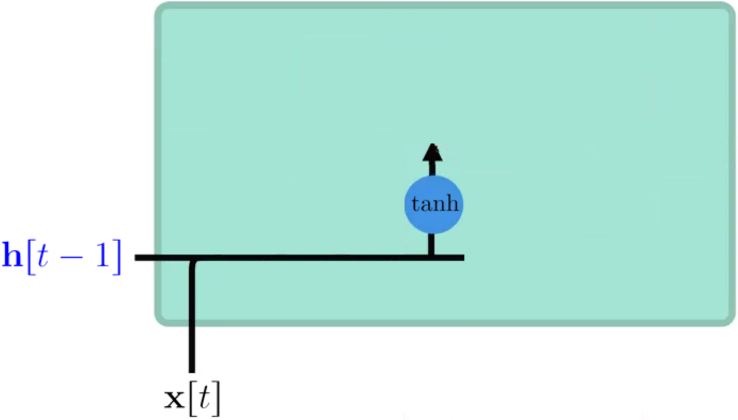
Practical Gated Architectures: *LSTM*

This is how inside an LSTM unit looks like



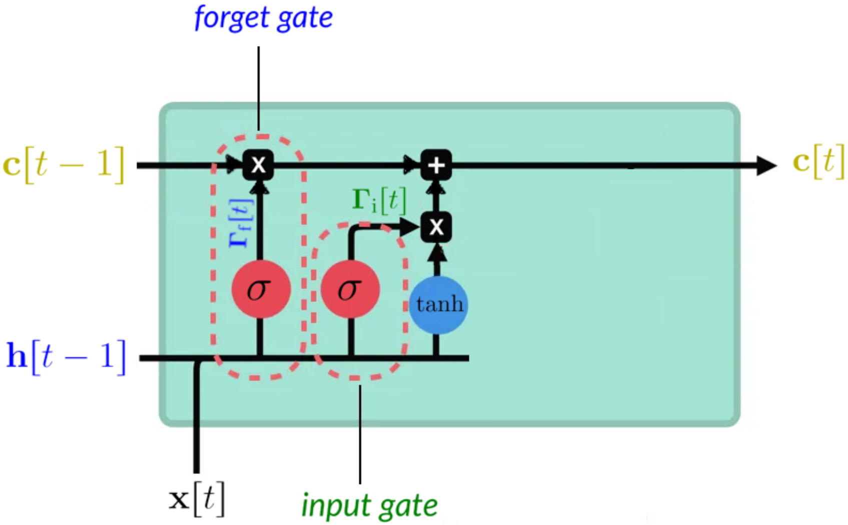
Practical Gated Architectures: *LSTM*

Actual cell state $\tilde{\mathbf{c}}[t] = f(\mathbf{W}_1\mathbf{x}[t] + \mathbf{W}_\mathrm{m}\mathbf{h}[t-1])$



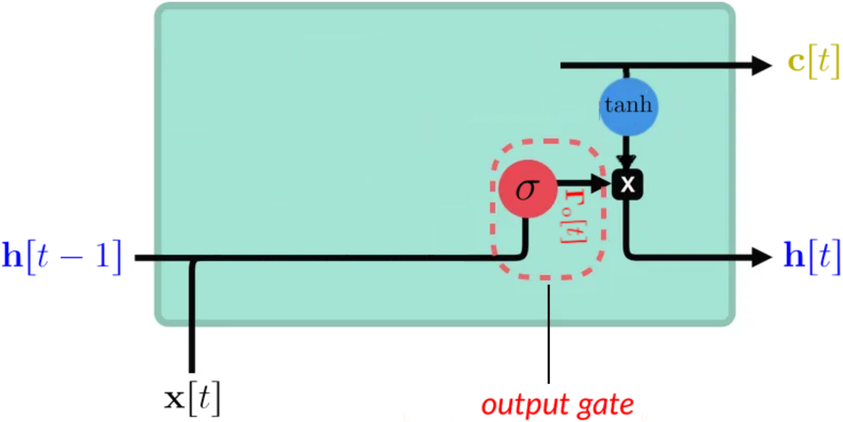
Practical Gated Architectures: *LSTM*

We use forget gate and update gate to update cell state



Practical Gated Architectures: *LSTM*

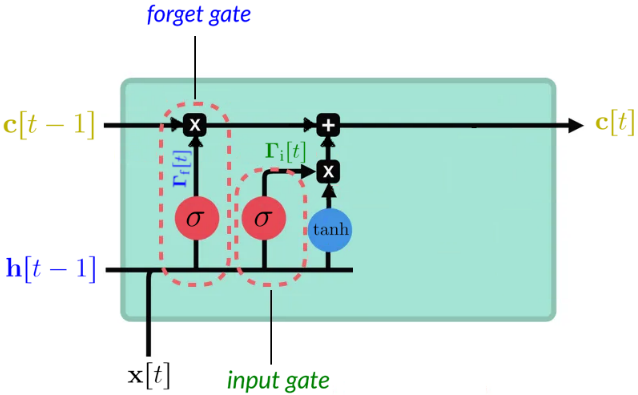
We use output gate to control fellow of memory to the hidden state



Practical Gated Architectures: *LSTM*

Intuitively, the gates in LSTM impact *the flow of information as follows*

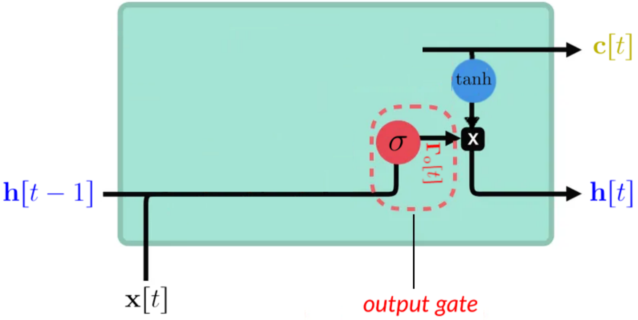
- *Forget gate controls how much we forget from last state*
  - ↳ *Assume* $\mathbf{\Gamma}_{\mathrm{f}}[t] = \mathbf{0}$*: then, we remember nothing of* $\mathbf{c}[t-1]$
- *Input gate controls how much we remember from new cell state*
  - ↳ *Assume* $\mathbf{\Gamma}_{\mathrm{i}}[t] = \mathbf{0}$*: then, we remember nothing of* $\tilde{\mathbf{c}}[t]$



Practical Gated Architectures: *LSTM*

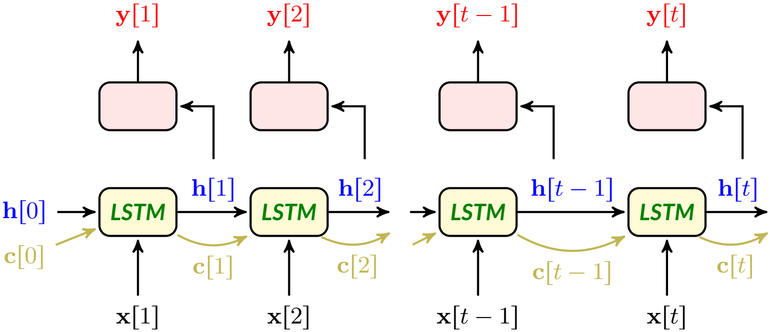
Intuitively, the gates in LSTM impact *the flow of information as follows*

- *Output gate controls how much we let from updated state to go out*
  - ↳ *Assume* $\boldsymbol{\Gamma}_0[t] = \mathbf{0}$*: then, we send nothing of* $\mathbf{c}[t]$ *out*



LSTM: *Forward Pass*

Starting from initial hidden and cell state: LSTM passes forward as



Pay Attention

Note that unlike other architectures, LSTM does not keep all memory inside hidden state but it carries it also in cell state. This state is only for memory and is not directly used by higher layers, e.g., output layer of the NN

LSTM: *Backward Pass*

Say we finished forward pass at time t. We now want to find $\nabla_{\mathbf{W}} \hat{R}$ for some $\mathbf{W}$ that is inside LSTM, e.g., $\mathbf{W}_{\mathrm{i,m}}$: we start backpropagating from $\nabla_{\mathbf{y}[t]} \hat{R}$

$$\nabla_{\mathbf{W}} \hat{R} = \nabla_{\mathbf{y}[t]} \hat{R} \circ \nabla_{\mathbf{W}} \mathbf{y}[t]$$

1. *We know* $\mathbf{y}[t] = f_{\mathrm{out}}(\mathbf{W}_2 \mathbf{h}[t])$

$$\nabla_{\mathbf{W}} \mathbf{y}[t] = \nabla_{\mathbf{h}[t]} \mathbf{y}[t] \circ \nabla_{\mathbf{W}} \mathbf{h}[t]$$

2. $\ldots$

Suggestion

Try writing it once to see the impact of gates!