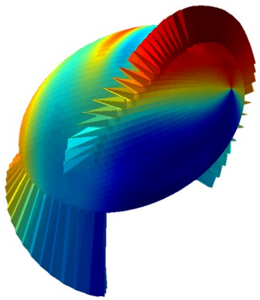
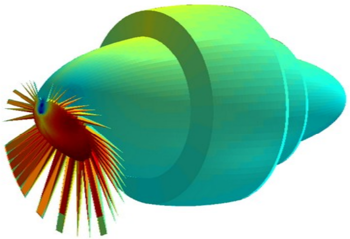
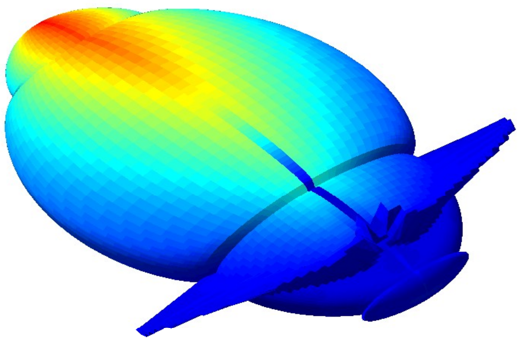
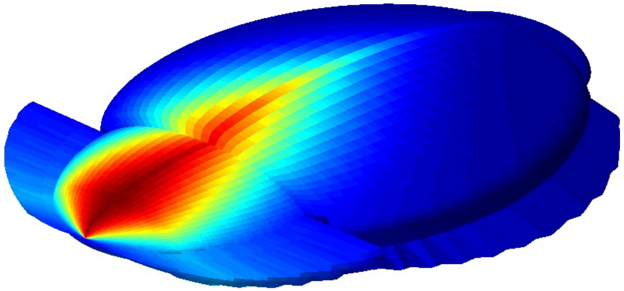


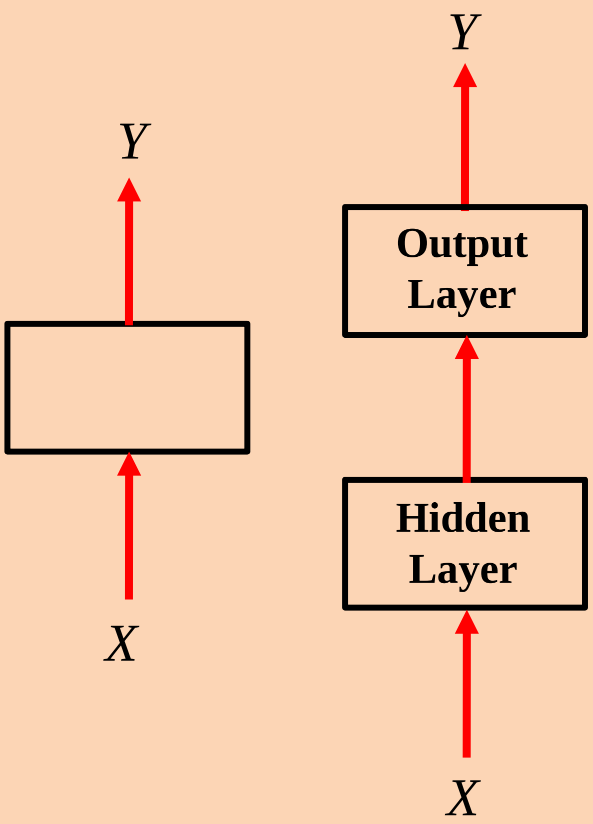


Lecture 14
Deep Learning II
Recurrent Networks

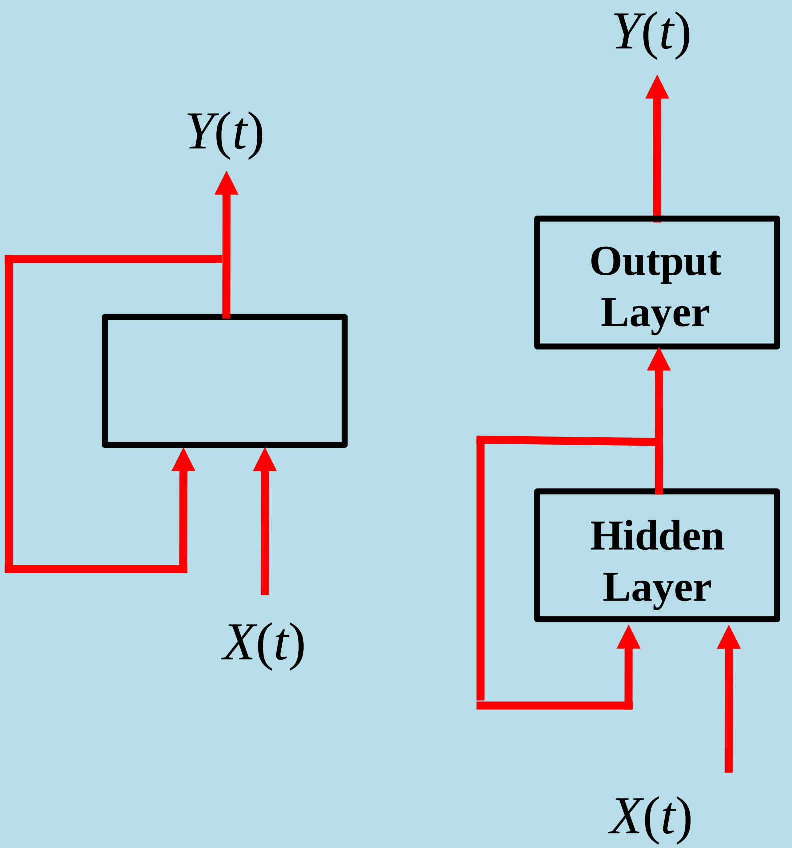




Recurrent Networks



Feed-Forward Networks



Recurrent Networks

Recurrent Network Applications

Sequence recognition

- Identify a word from its audio recording.
- Recognize a song from its audio.

Sequence labeling

- Label all proper nouns in a text.
- Label all R-peaks in an ECG

Sequence/Time-Series Prediction

- Predict the price of a stock based on previous prices.
- Predict the next word in a text given the words so far (Language models).

Sequence Generation

- Generate a sequence of actions to perform a task.
- Play a piece of music given its title.

Sequence-to-Sequence Learning

- Output a sequence in response to an input sequence.
- Output the words of a song given the lyrics.
- Machine translation.

Language Representation

- Embed sentences into a vector space (sentence embedding).

Associative Memory

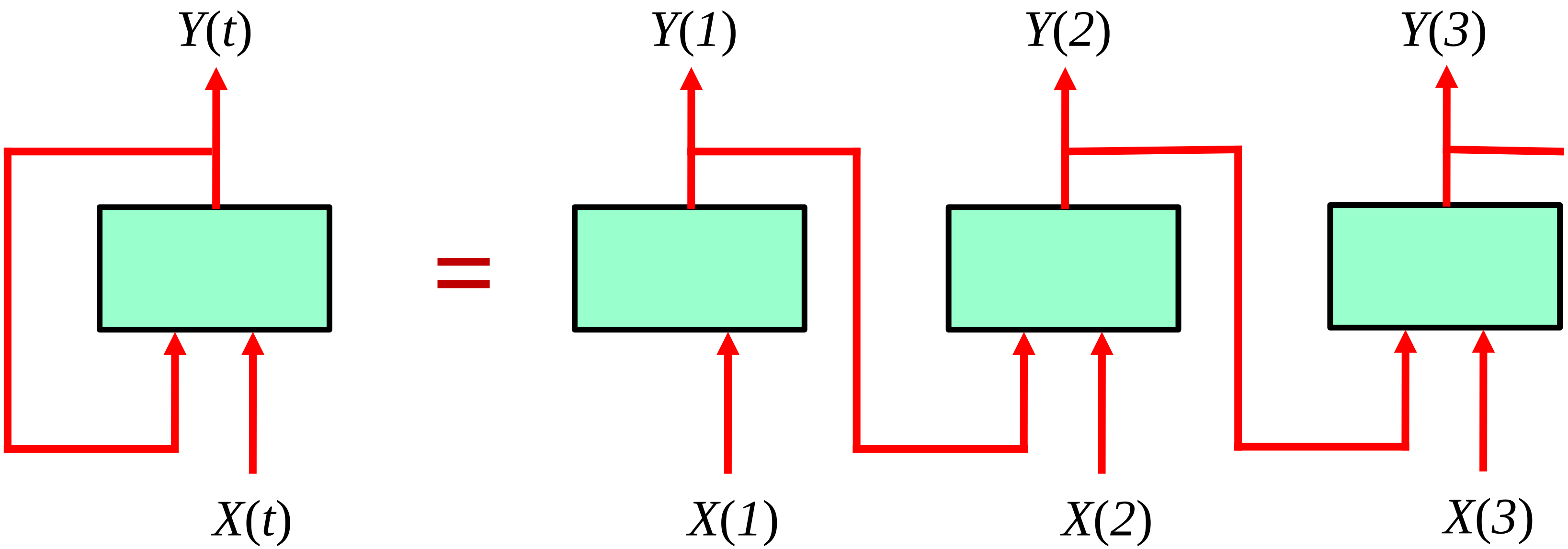
- Store and recall associative memories as attractors.

Pattern Completion

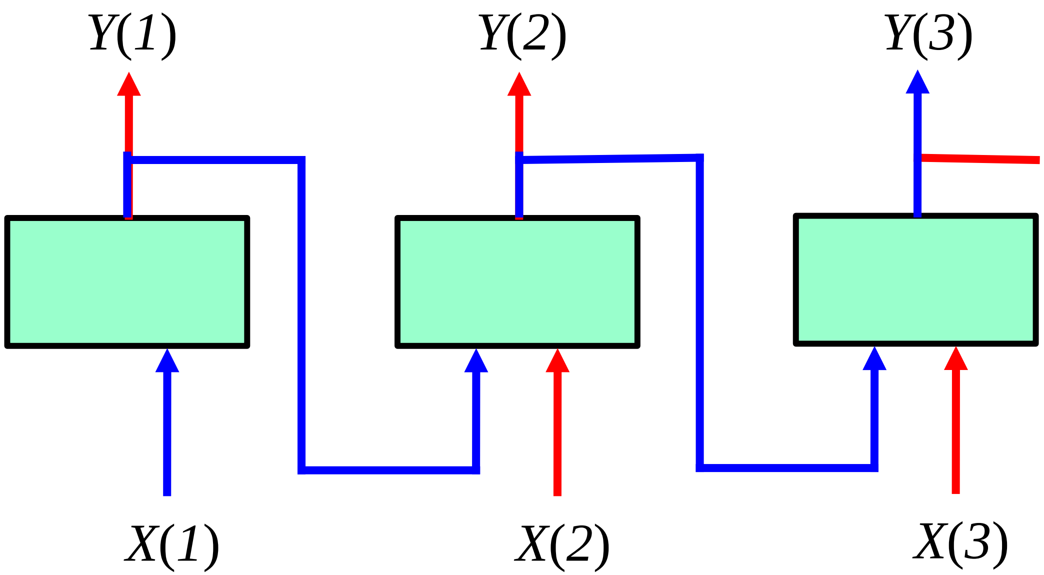
- Complete/filter a pattern given a noisy or partial version.

….. And many others.

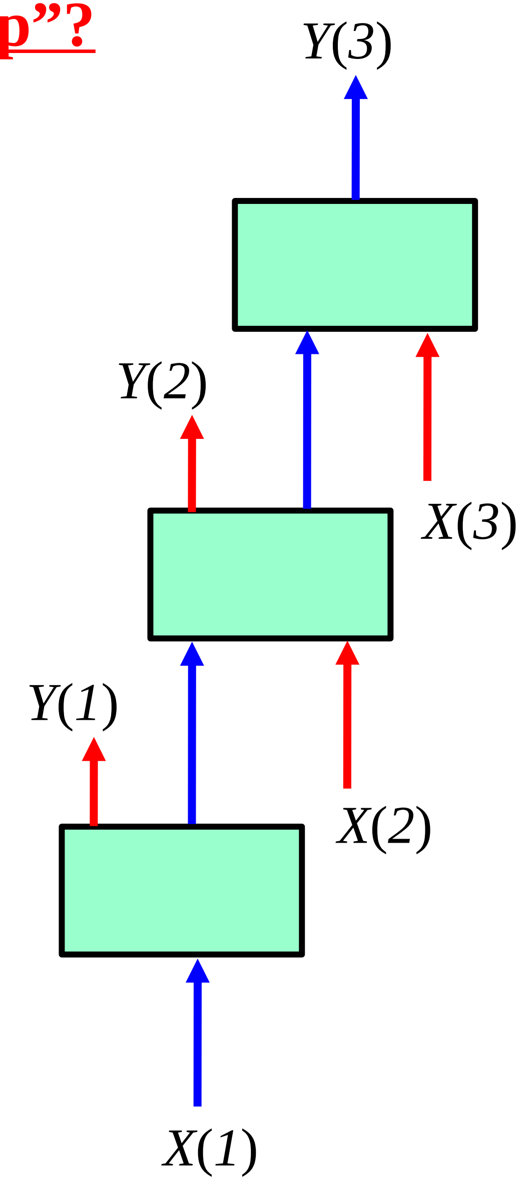
Why are recurrent networks "deep"?



Why are recurrent networks “deep”?



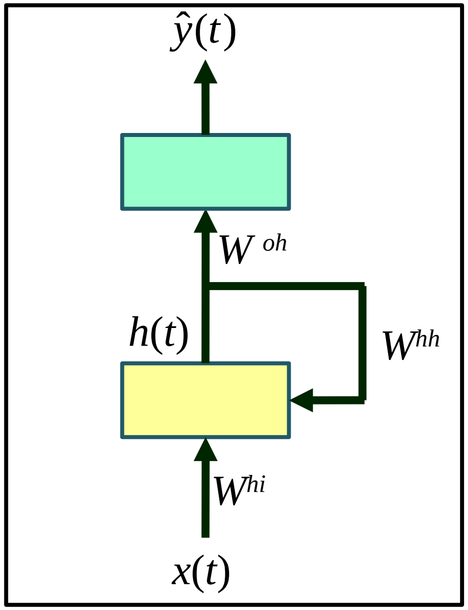
=



Each cycle of time is like going through a layer of neurons

Recurrent networks learn using ***backpropagation through time*** (BPTT)

Training Recurrent Networks



A typical, single hidden-layer recurrent network.

Weight vector: $W = \left[W^{oh} \quad W^{hh} \quad W^{hi} \right]^T$

Loss function: $J(t)$

Data set: $\{ (x(t), y(t)) \}$ → Training/validation/test sets

Typically, several shorter *sub-sequences* are grouped into a *minibatch*, and several minibatches in an *epoch*.

- Recurrent networks are trained by backpropagation.
- Each time-step adds a layer to the effective network.
- Backpropagation of δs through layers = propagation of δs back through time.

→ Backpropagation Through Time (BPTT)

Learning Procedure:

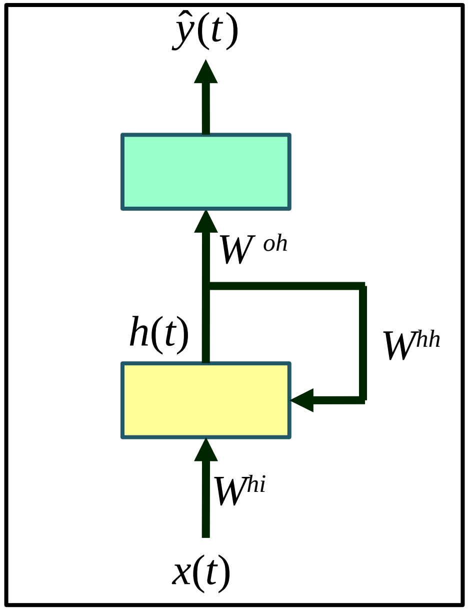
For t = 1 to T

- Present input $x(t)$
- Get the output $\hat{y}(t)$
- Compare with desired output $y(t)$ to get errors $e(t)$ and loss $J(t)$
- Calculate the gradient $\frac{\partial J(t)}{\partial W} = \left[\frac{\partial J(t)}{\partial W^{oh}} \quad \frac{\partial J(t)}{\partial W^{hh}} \quad \frac{\partial J(t)}{\partial W^{hi}} \right]^T$
- Update weights using $\Delta W = -\eta \frac{\partial J(t)}{\partial W}$ (every step, or end of sequence/epoch)

End

Main Issue:

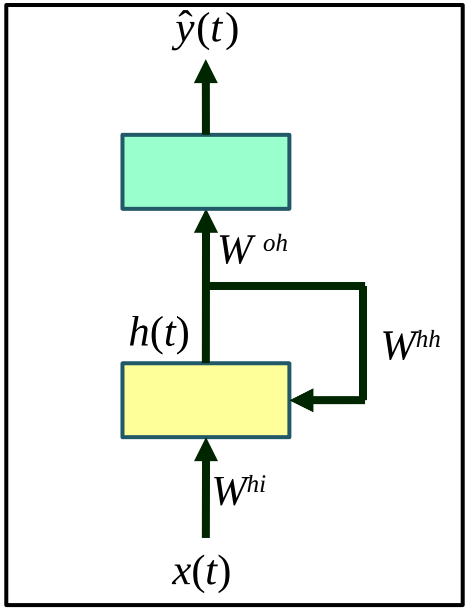
- Hidden output $h(t)$ depends not just on $x(t)$ but also on $h(t\text{-}1)$
- And $h(t\text{-}1)$ is generated through the same weights as $h(t)$
- And the same is true for $h(t\text{-}2)$ and $h(t\text{-}3)$,…., $h(1)$
- So $h(t)$ at time t depends on the recurrent weights W^{hh} through multiple paths.



To train the W^{hi} and W^{hh} weights, we need to calculate:

$$\frac{\partial e(t)}{\partial W^{hh}} = \frac{\partial e(t)}{\partial \hat{y}(t)} \frac{\partial \hat{y}(t)}{\partial W^{hh}} = \frac{\partial e(t)}{\partial \hat{y}(t)} \frac{\partial \hat{y}(t)}{\partial h(t)} \frac{\partial h(t)}{\partial W^{hh}}$$

$$\frac{\partial e(t)}{\partial W^{hi}} = \frac{\partial e(t)}{\partial \hat{y}(t)} \frac{\partial \hat{y}(t)}{\partial W^{hi}} = \frac{\partial e(t)}{\partial \hat{y}(t)} \frac{\partial \hat{y}(t)}{\partial h(t)} \frac{\partial h(t)}{\partial W^{hi}}$$

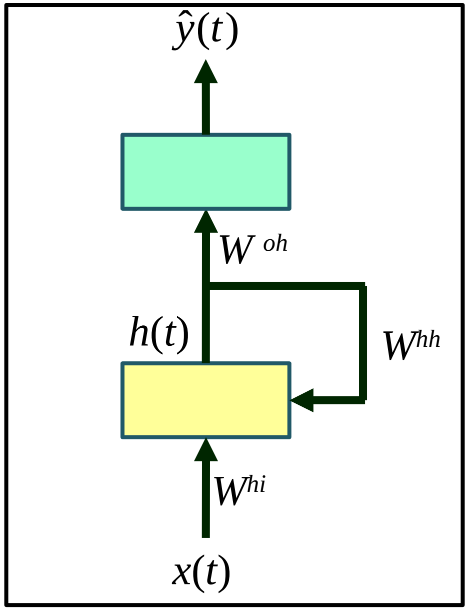


To train the W^{hi} and W^{hh} weights, we need to calculate:

$$\frac{\partial e(t)}{\partial W^{hh}} = \frac{\partial e(t)}{\partial \hat{y}(t)} \frac{\partial \hat{y}(t)}{\partial W^{hh}} = \frac{\partial e(t)}{\partial \hat{y}(t)} \frac{\partial \hat{y}(t)}{\partial h(t)} \boxed{\frac{\partial h(t)}{\partial W^{hh}}}$$

$$\frac{\partial e(t)}{\partial W^{hi}} = \frac{\partial e(t)}{\partial \hat{y}(t)} \frac{\partial \hat{y}(t)}{\partial W^{hi}} = \frac{\partial e(t)}{\partial \hat{y}(t)} \frac{\partial \hat{y}(t)}{\partial h(t)} \boxed{\frac{\partial h(t)}{\partial W^{hi}}}$$

Multiple paths of dependency

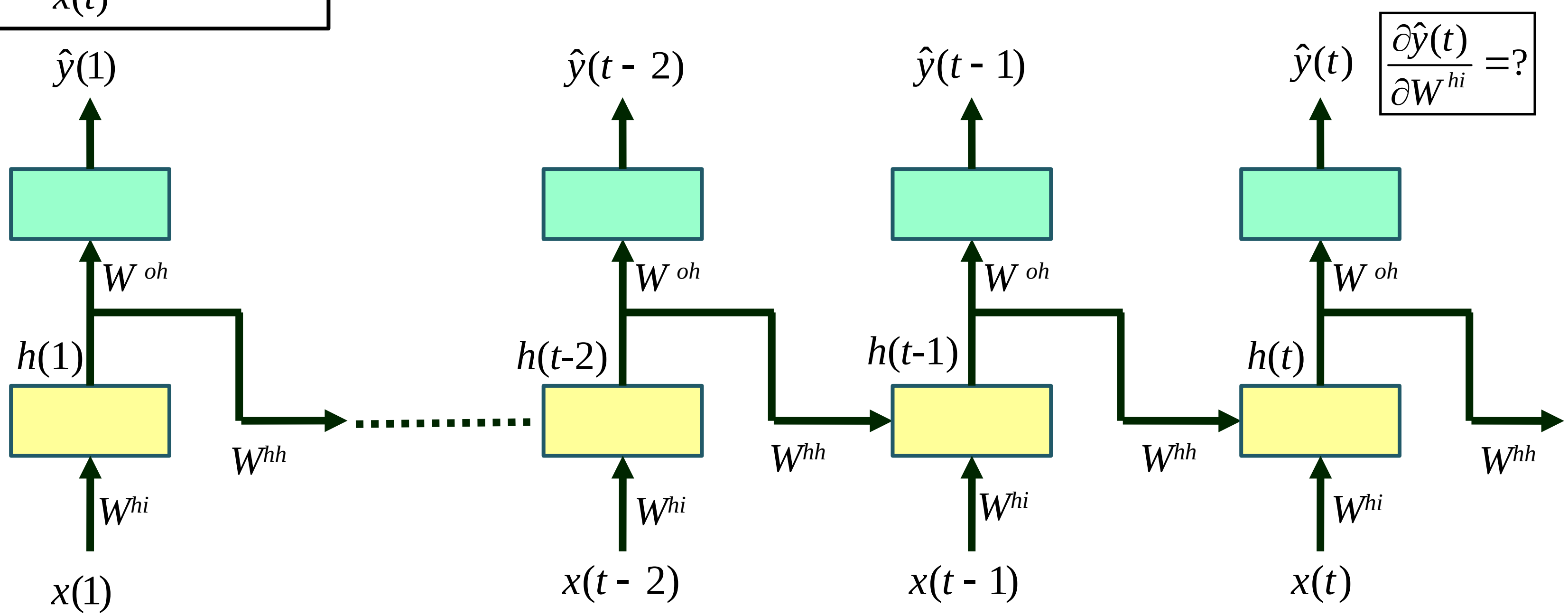


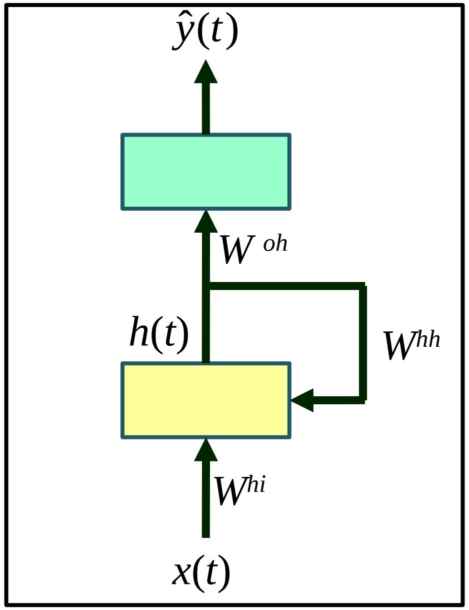
To train the W^{hi} and W^{hh} weights, we need to calculate:

$$\frac{\partial e(t)}{\partial W^{hh}} = \frac{\partial e(t)}{\partial \hat{y}(t)} \frac{\partial \hat{y}(t)}{\partial W^{hh}} = \frac{\partial e(t)}{\partial \hat{y}(t)} \frac{\partial \hat{y}(t)}{\partial h(t)} \boxed{\frac{\partial h(t)}{\partial W^{hh}}}$$

$$\frac{\partial e(t)}{\partial W^{hi}} = \frac{\partial e(t)}{\partial \hat{y}(t)} \frac{\partial \hat{y}(t)}{\partial W^{hi}} = \frac{\partial e(t)}{\partial \hat{y}(t)} \frac{\partial \hat{y}(t)}{\partial h(t)} \boxed{\frac{\partial h(t)}{\partial W^{hi}}}$$

Must be summed over all paths



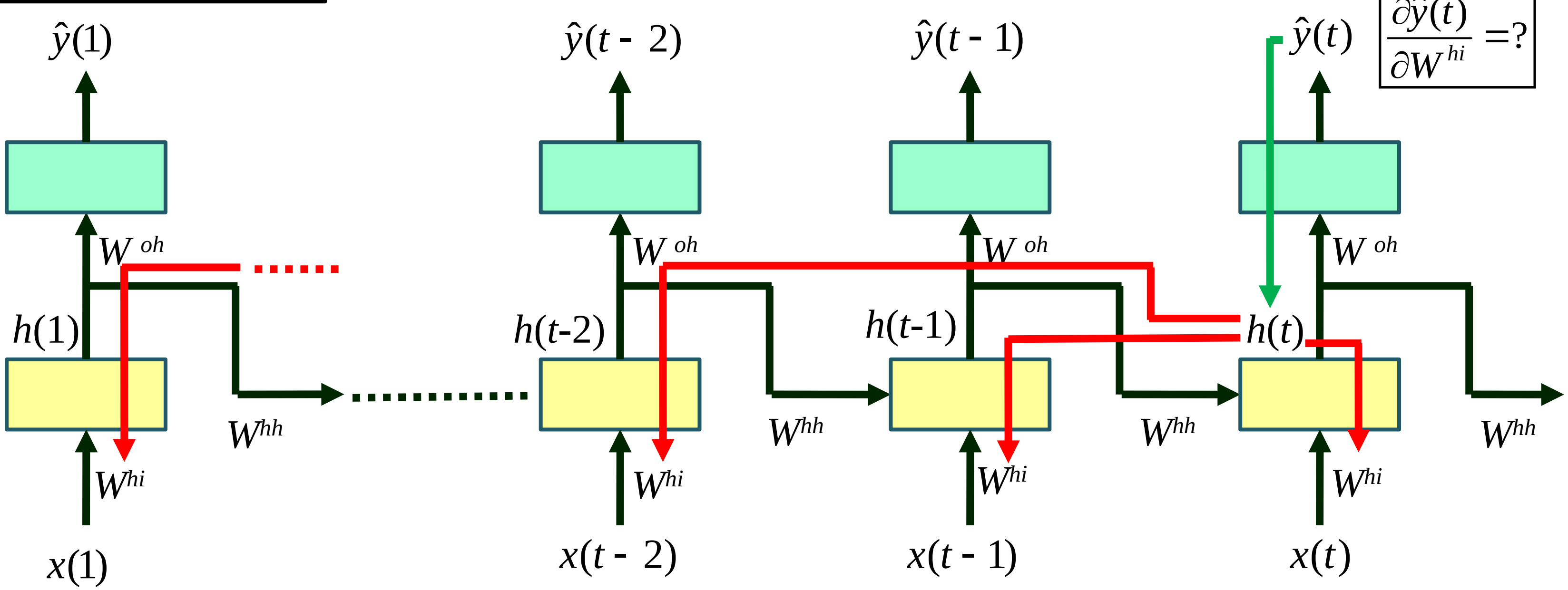


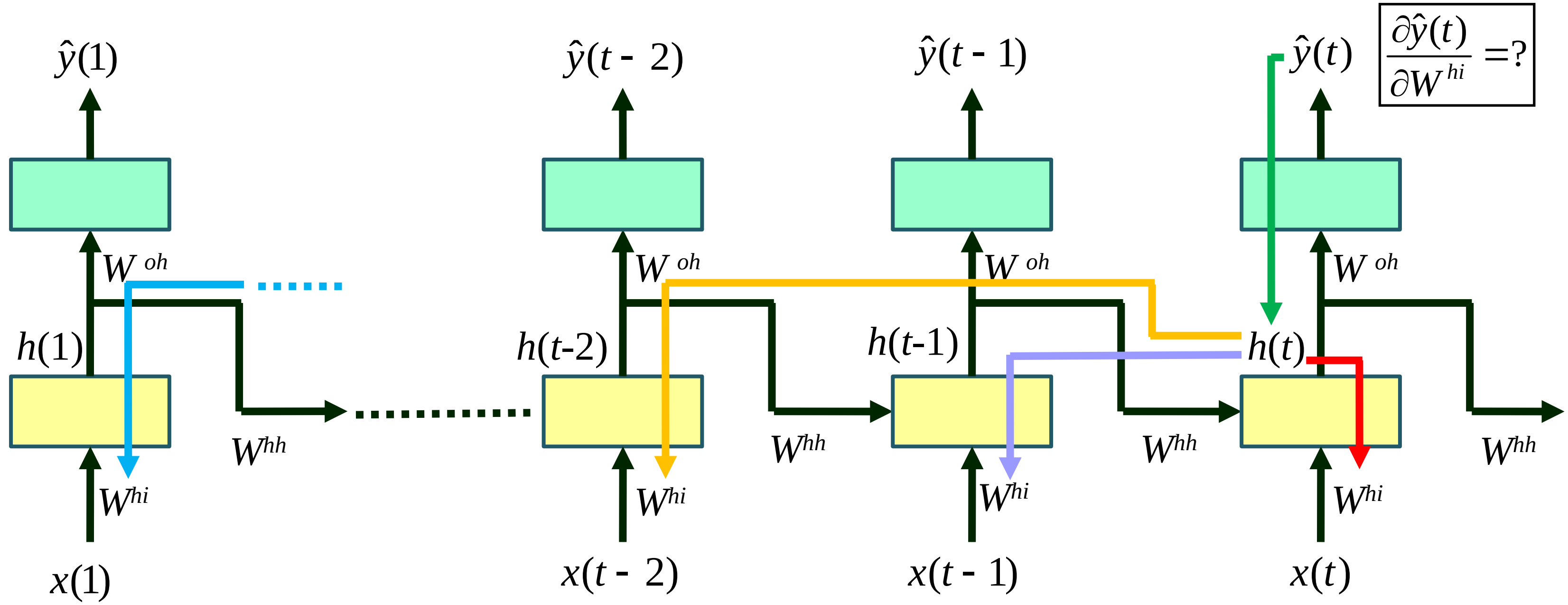
To train the W^{hi} and W^{hh} weights, we need to calculate:

$$\frac{\partial e(t)}{\partial W^{hh}} = \frac{\partial e(t)}{\partial \hat{y}(t)} \frac{\partial \hat{y}(t)}{\partial W^{hh}} = \frac{\partial e(t)}{\partial \hat{y}(t)} \frac{\partial \hat{y}(t)}{\partial h(t)} \frac{\partial h(t)}{\partial W^{hh}}$$

$$\frac{\partial e(t)}{\partial W^{hi}} = \frac{\partial e(t)}{\partial \hat{y}(t)} \frac{\partial \hat{y}(t)}{\partial W^{hi}} = \frac{\partial e(t)}{\partial \hat{y}(t)} \frac{\partial \hat{y}(t)}{\partial h(t)} \frac{\partial h(t)}{\partial W^{hi}}$$

$$\boxed{\frac{\partial \hat{y}(t)}{\partial W^{hi}} = ?}$$



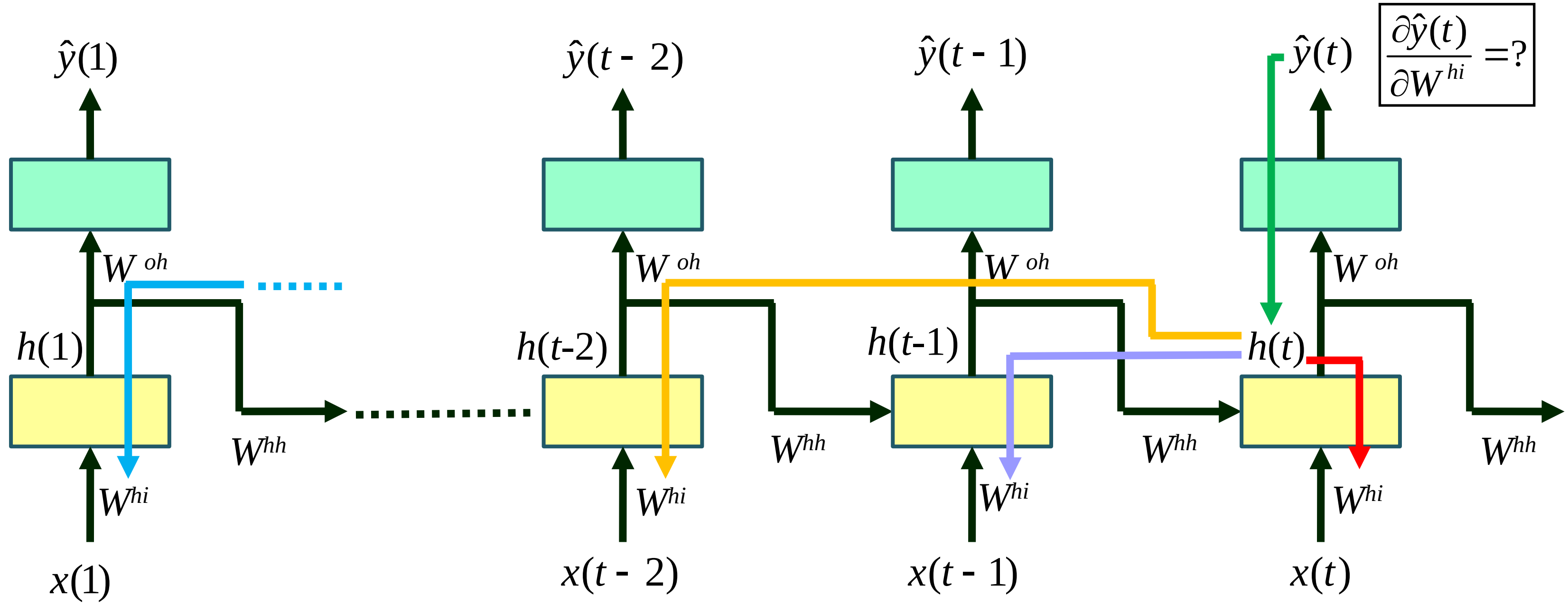


Consider the paths shown in different colors:

$$\frac{\partial h(t)}{\partial W^{hi}} = \boxed{\frac{\partial f(x(t), h(t-1))}{\partial W^{hi}}} + \boxed{\frac{\partial h(t)}{\partial h(t-1)} \frac{\partial h(t-1)}{\partial W^{hi}}} + \boxed{\frac{\partial h(t)}{\partial h(t-1)} \frac{\partial h(t-1)}{\partial h(t-2)} \frac{\partial h(t-2)}{\partial W^{hi}}}$$

$$+ \cdots\cdots\cdots + \boxed{\left(\prod_{d=2}^{t} \frac{\partial h(d)}{\partial h(d-1)} \right) \frac{\partial h(1)}{\partial W^{hi}}}$$

chain rule



Consider the three paths shown in different colors:

$$\frac{\partial h(t)}{\partial W^{hi}} = \boxed{\frac{\partial f(x(t), h(t-1))}{\partial W^{hi}}} + \boxed{\frac{\partial h(t)}{\partial h(t-1)} \frac{\partial h(t-1)}{\partial W^{hi}}} + \boxed{\frac{\partial h(t)}{\partial h(t-1)} \frac{\partial h(t-1)}{\partial h(t-2)} \frac{\partial h(t-2)}{\partial W^{hi}}}$$

$$+ \cdots\cdots\cdots + \boxed{\left(\prod_{d=2}^{t} \frac{\partial h(d)}{\partial h(d-1)} \right) \frac{\partial h(1)}{\partial W^{hi}}}$$

Note that this could be written as $\dfrac{\partial h(t)}{\partial W^{hi}}$ but that would confuse it with the LHS

Thus, at time step *t*:

$$\boxed{\frac{\partial h(t)}{\partial W^{hi}} = \frac{\partial f(x(t), h(t-1))}{\partial W^{hi}} + \sum_{q=1}^{t-1} \left(\prod_{d=q+1}^{t} \frac{\partial h(d)}{\partial h(d-1)} \right) \frac{\partial h(q)}{\partial W^{hi}}}$$

from which, we can calculate: $\frac{\partial e(t)}{\partial W^{hi}} = \frac{\partial e(t)}{\partial \hat{y}(t)} \frac{\partial \hat{y}(t)}{\partial h(t)} \frac{\partial h(t)}{\partial W^{hi}}$

Similarly for $\frac{\partial e(t)}{\partial W^{hh}}$

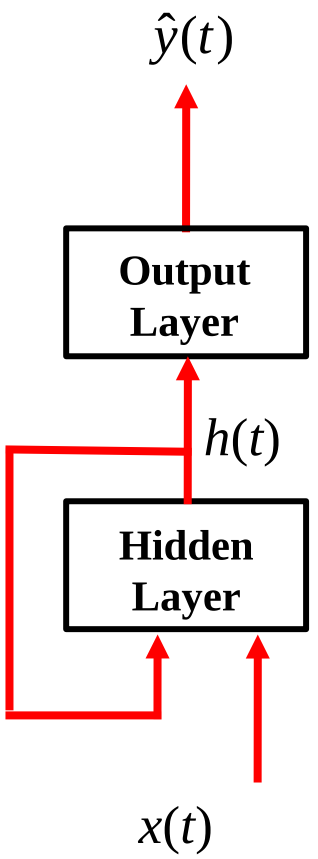
Problem: As *t* gets larger, we need to calculate longer and longer chains.

Solution: Use shorter training sub-sequences, limit how far back to chain
This is called ***truncated BPTT*** with look-back limited to τ steps back.

Problem: Back-propagating through time causes vanishing or exploding gradients.

Solution: Clip the gradient (to prevent explosion), use ReLU (to prevent vanishing).

Recurrent networks are good at remembering recent context to generate outputs:



$$\hat{y}(t) = f_{out}\left(h(t)\right)$$

$$h(t) = f_h(x(t), h(t-1))$$

$$\Rightarrow \ \hat{y}(t) \text{ depends on } x(t), x(t-1), x(t-2), \ldots.$$

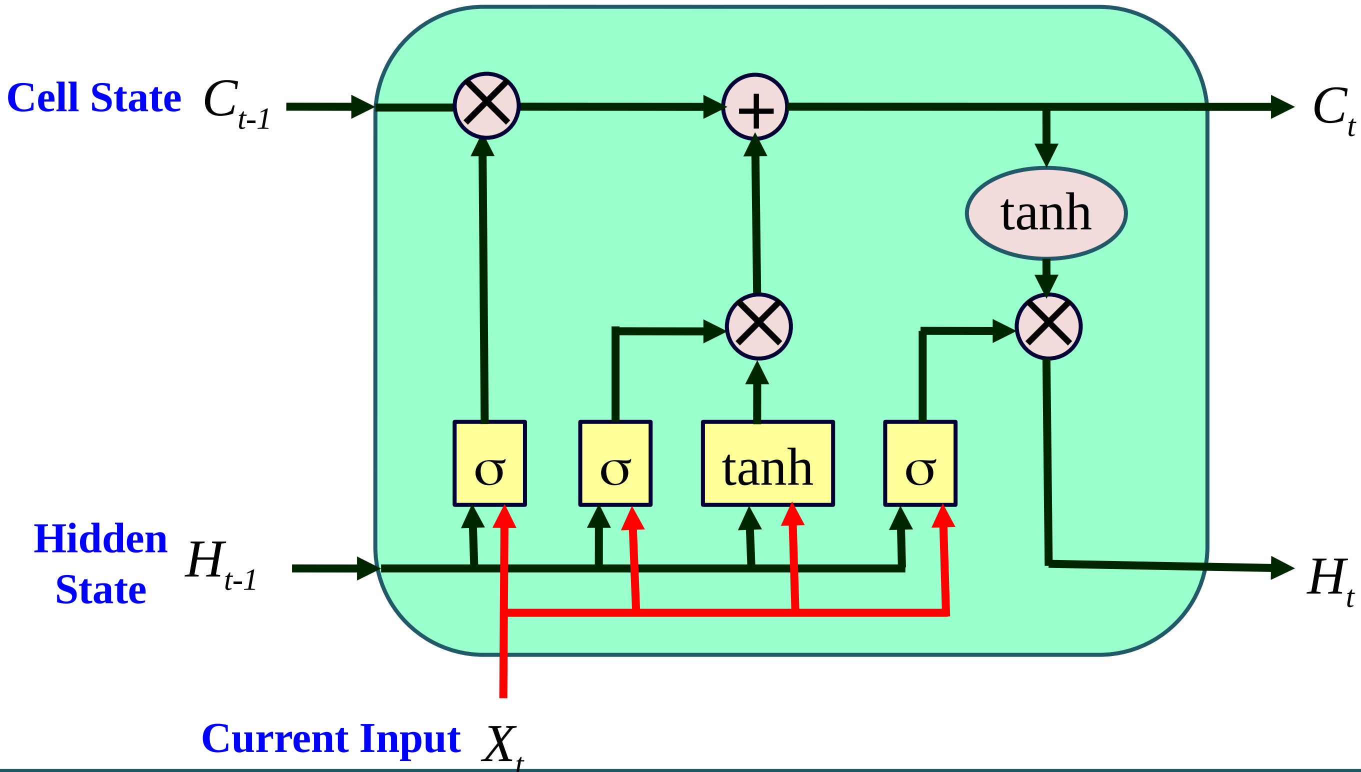
But how much each input is remembered depends only on how long ago it occurred, ***not on its meaning or significance***.

There is need for a recurrent network that can control which past data to remember and which to forget based on its meaning and significance.

Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM)

Hochreiter & Schmidhuber (1997) http://www.bioinf.jku.at/publications/older/2604.pdf

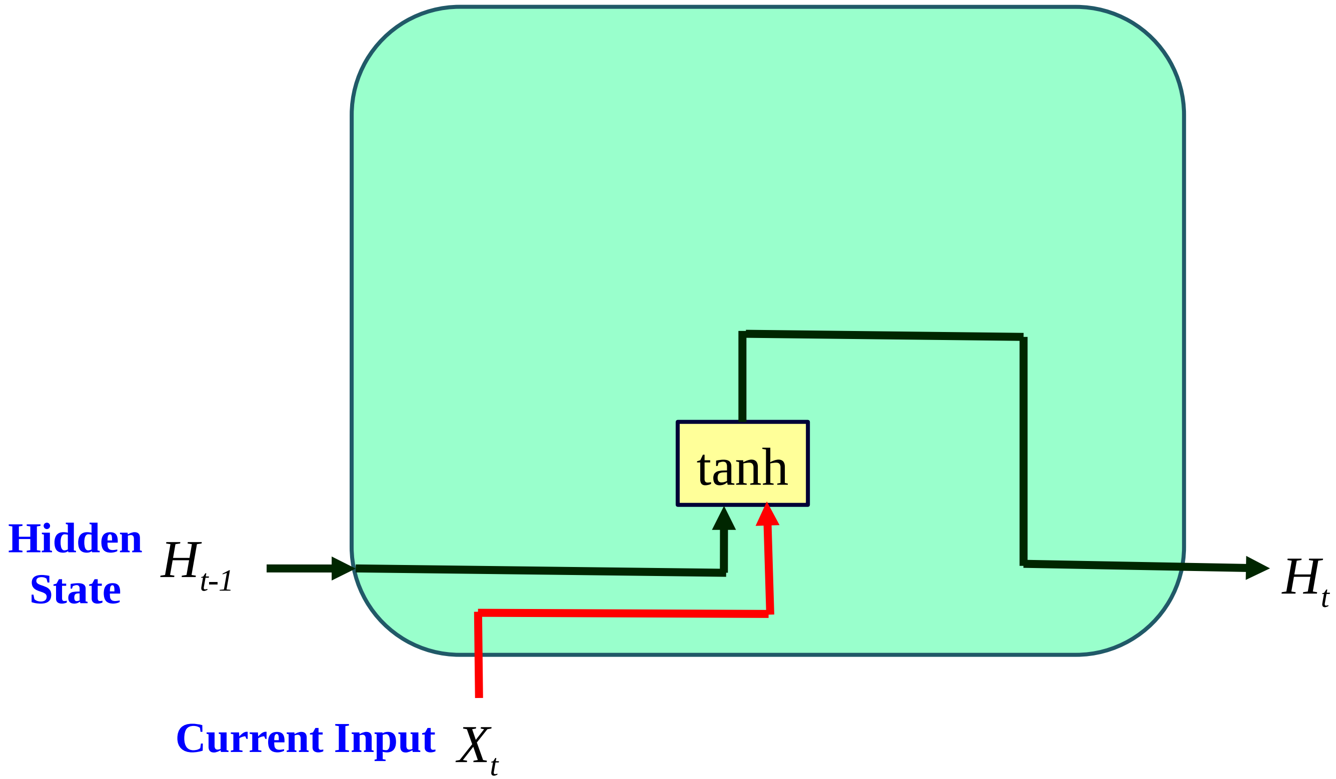


An LSTM Cell ↔ RNN Hidden Layer:

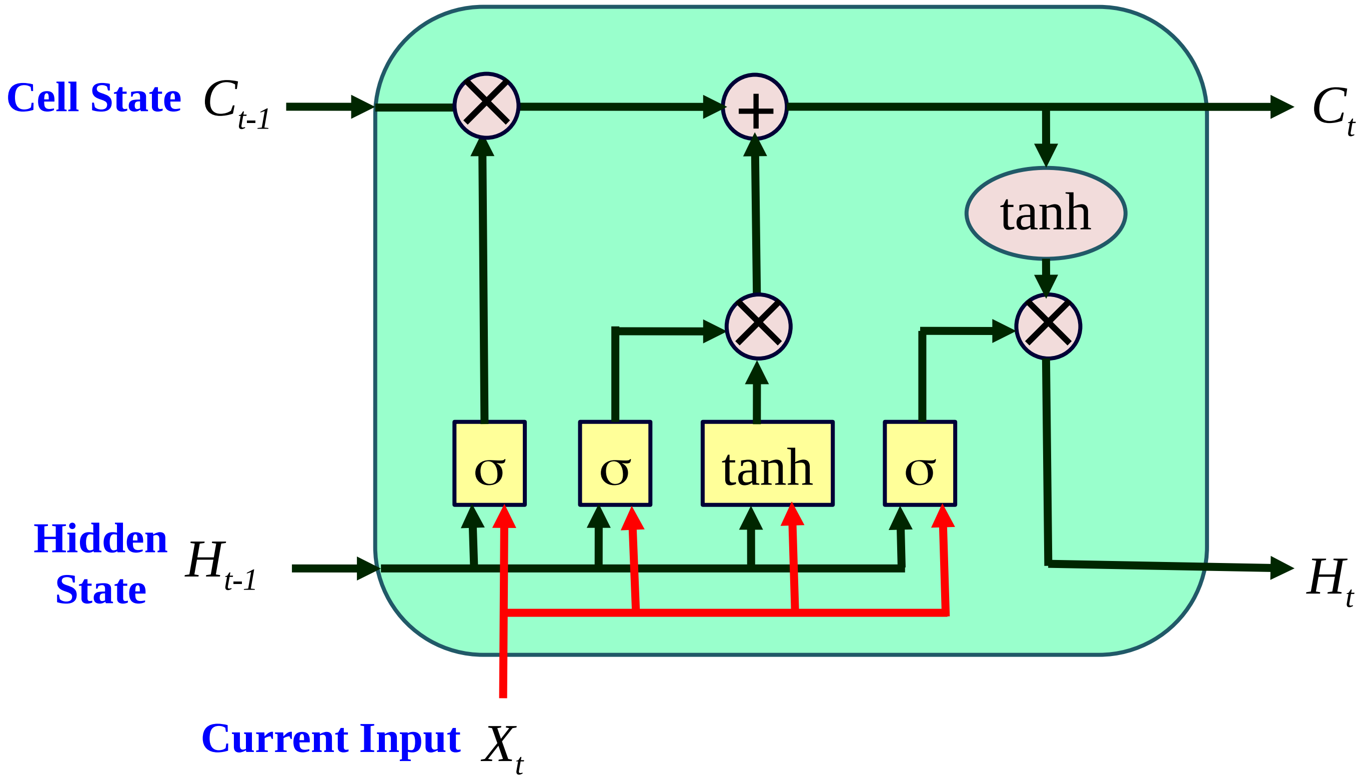
Each LSTM cell is a neural network with several layers shown as σ and tanh

The ⊗, + and tanh denote element-wise operations.

Comparison with Standard RNN

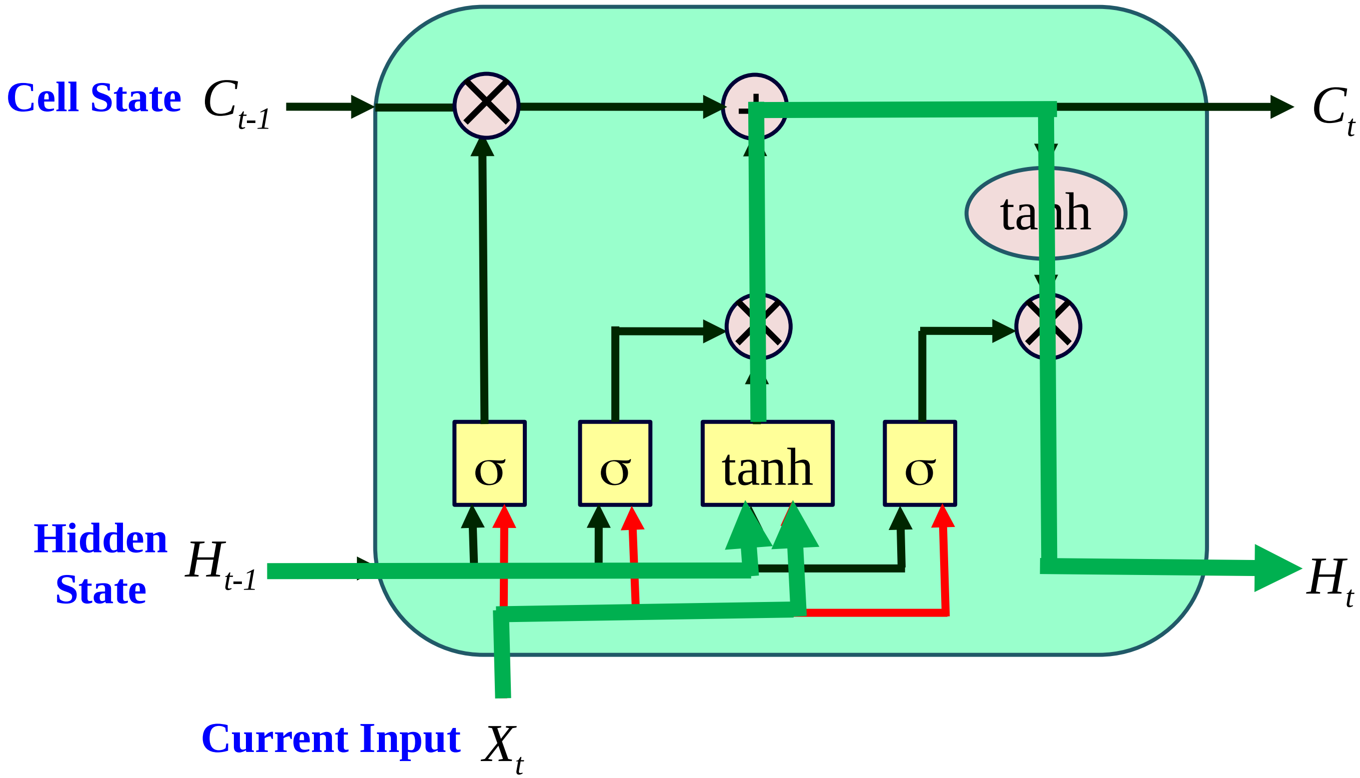


Comparison with Standard RNN



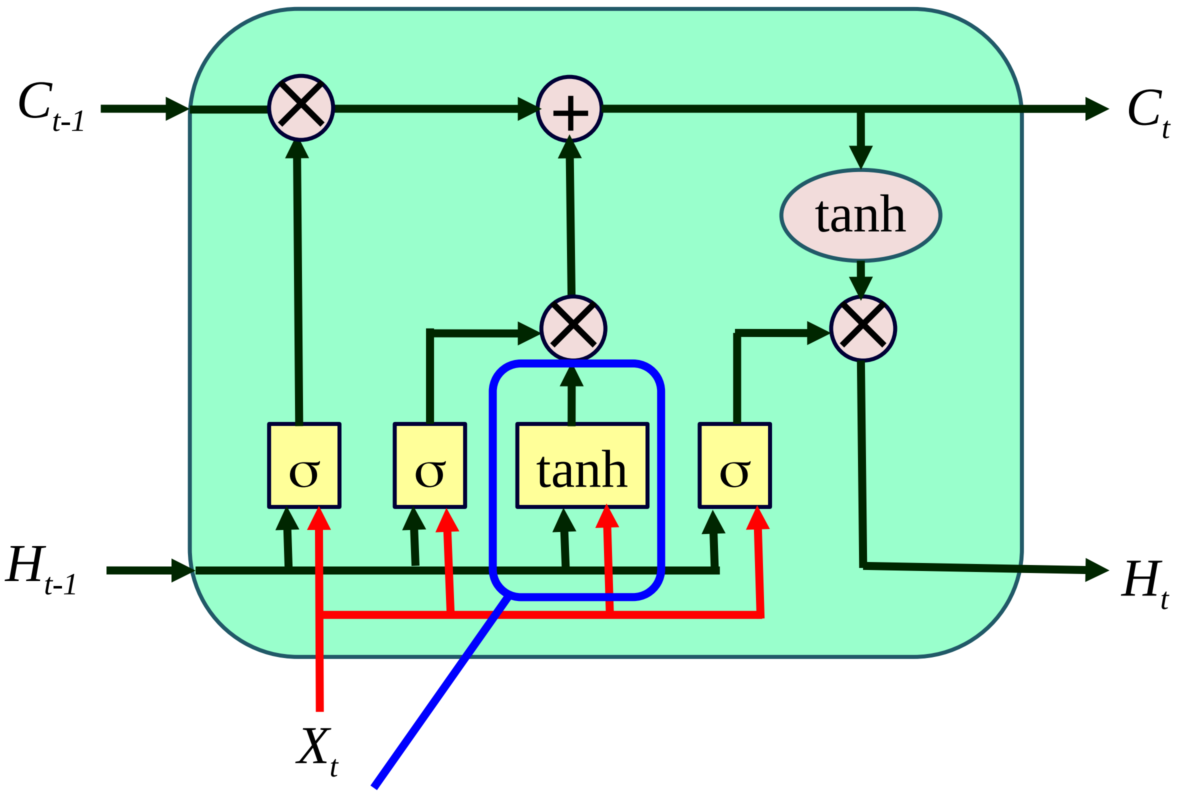
The standard hidden layer is replaced by a 4-part hidden layer, with each sub-layer of the same dimension.

Comparison with Standard RNN



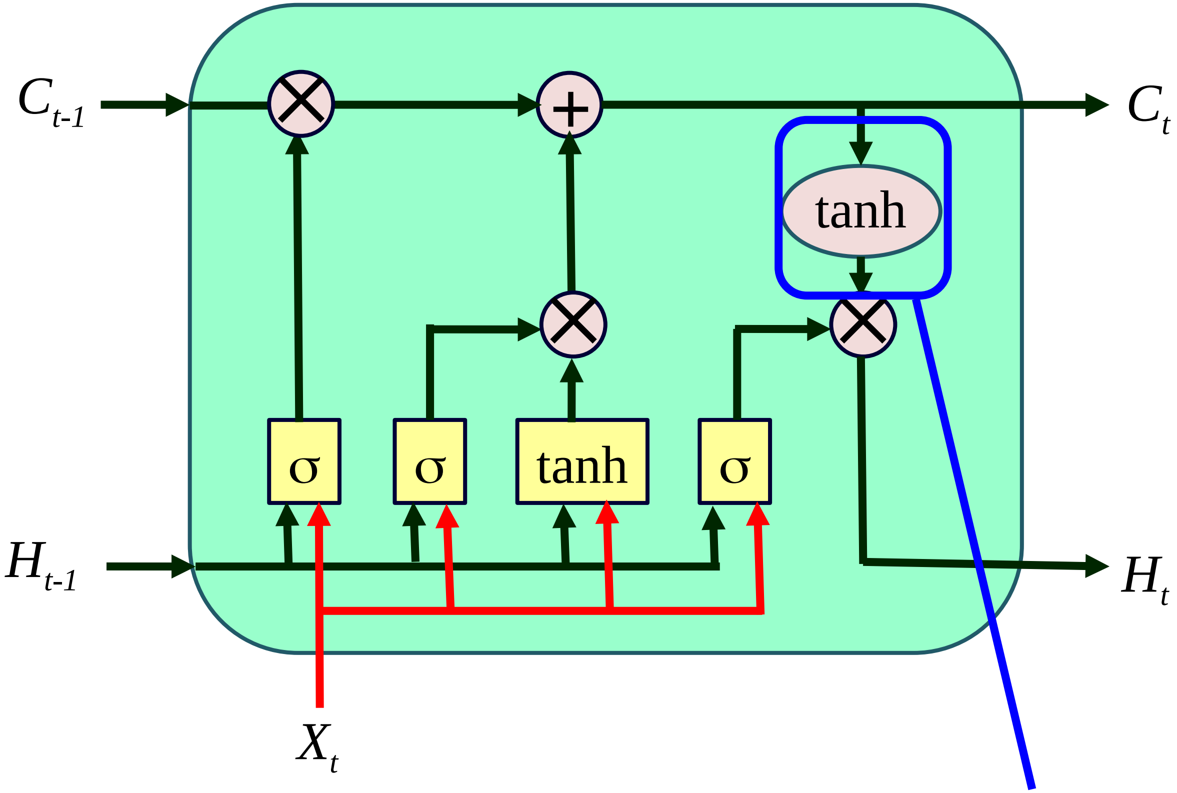
The standard hidden layer is replaced by 4-part hidden layer, with each sub-layer of the same dimension. The **green** line shows the ***main path*** through the cell.

Long Short-Term Memory (LSTM)



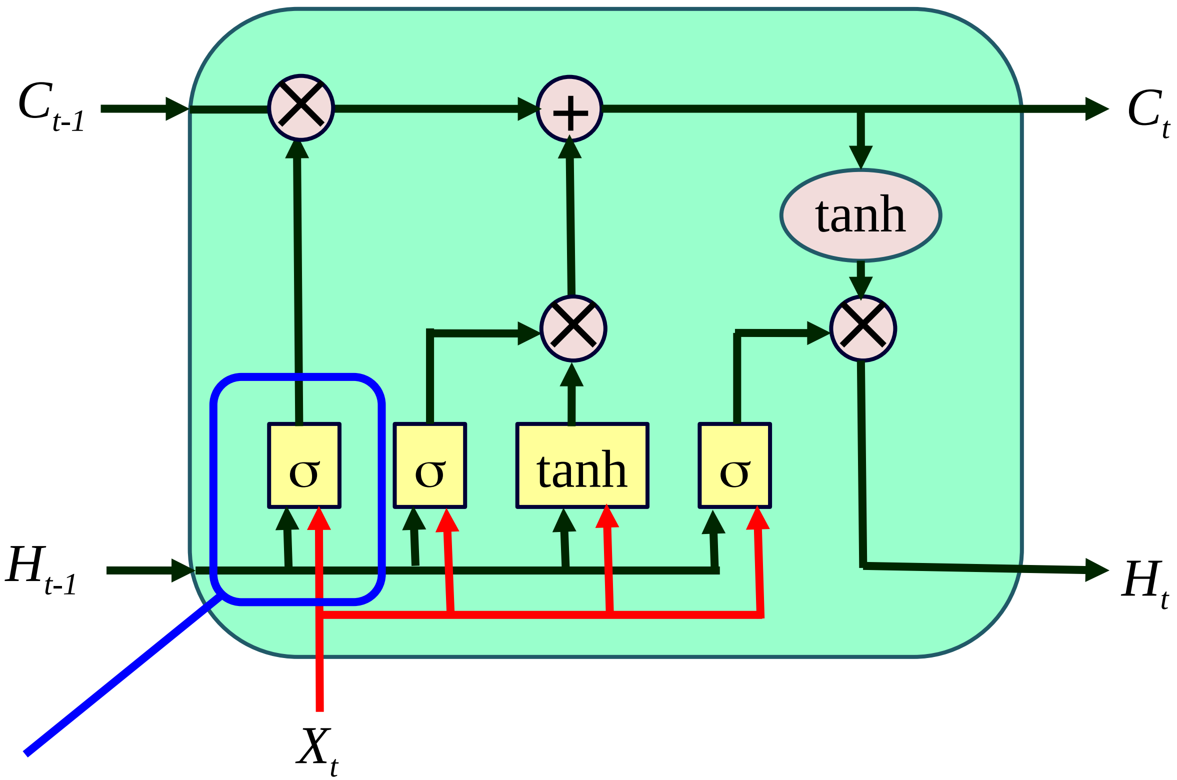
Squashed Net Input: The update to the cell state based on previous output state and current input transformed by a layer of *tanh* ( ) neurons. This is basically the hidden layer of the cell.

Long Short-Term Memory (LSTM)



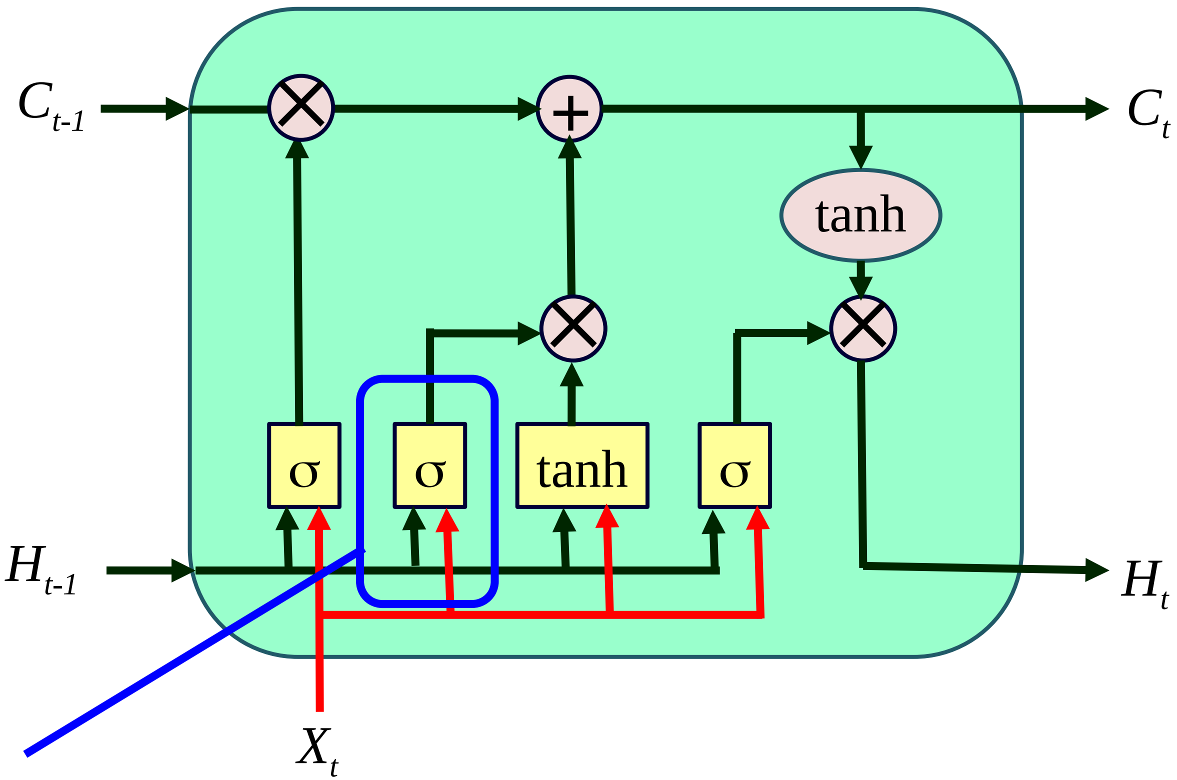
Squashed Output: The updated hidden state/output of the cell based on the new cell state squashed element-wise by a *tanh* ( ) function.

Long Short-Term Memory (LSTM)



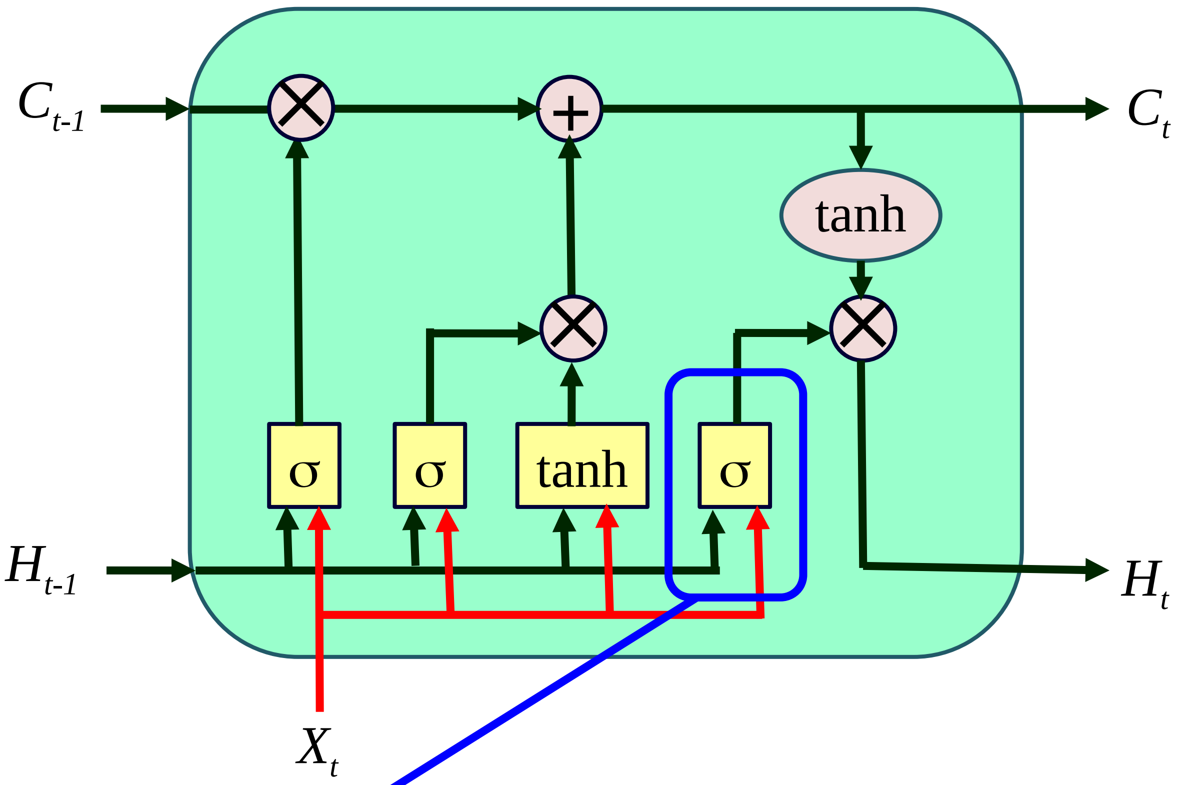
Forget Gate: Controls which elements of the cell state are remembered how much

Long Short-Term Memory (LSTM)



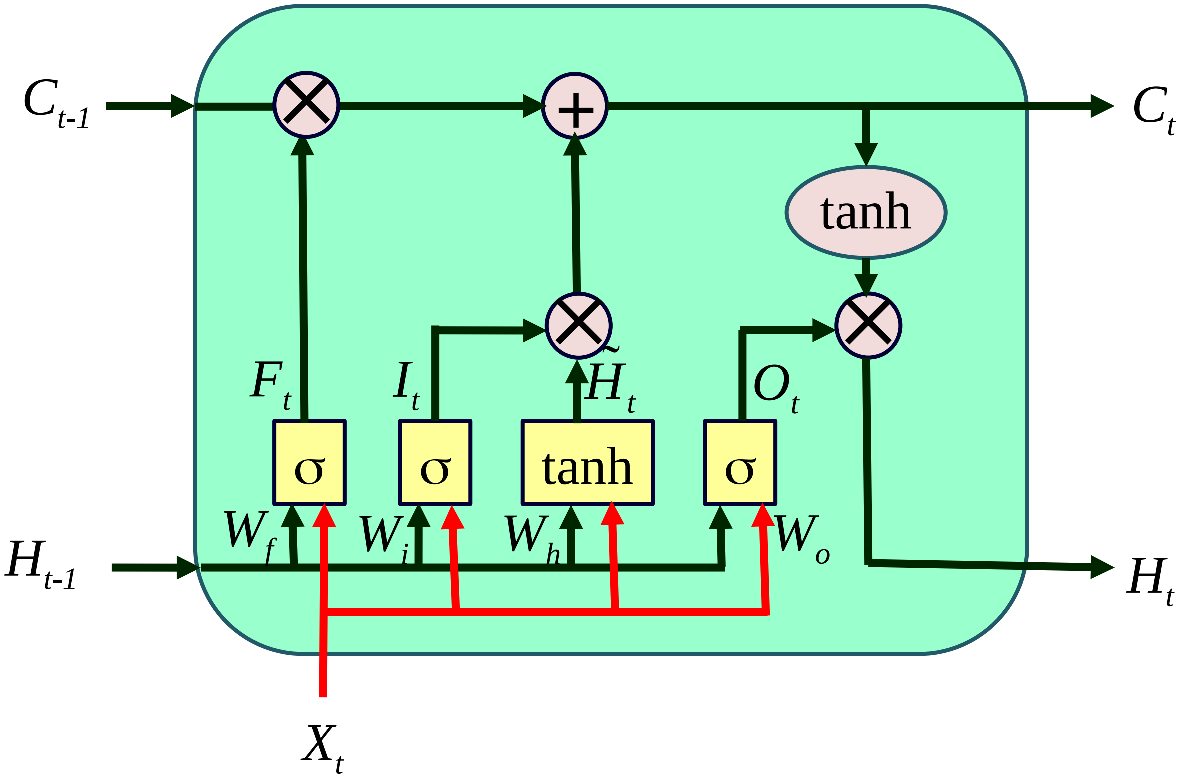
Input Gate: Controls how much of each element of the squashed net input is added to the cell state.

Long Short-Term Memory (LSTM)



Output Gate: Controls how much of each element of the squashed cell state is included in the new hidden state.

LSTM Equations



$$F_t = \sigma\left(W_f \cdot \left[X_t \; H_{t-1}\right] + b_f\right)$$

$$I_t = \sigma\left(W_i \cdot \left[X_t \; H_{t-1}\right] + b_i\right)$$

$$O_t = \sigma\left(W_o \cdot \left[X_t \; H_{t-1}\right] + b_o\right)$$

$$\tilde{H}_t = \tanh\left(W_h \cdot \left[X_t \; H_{t-1}\right] + b_h\right)$$

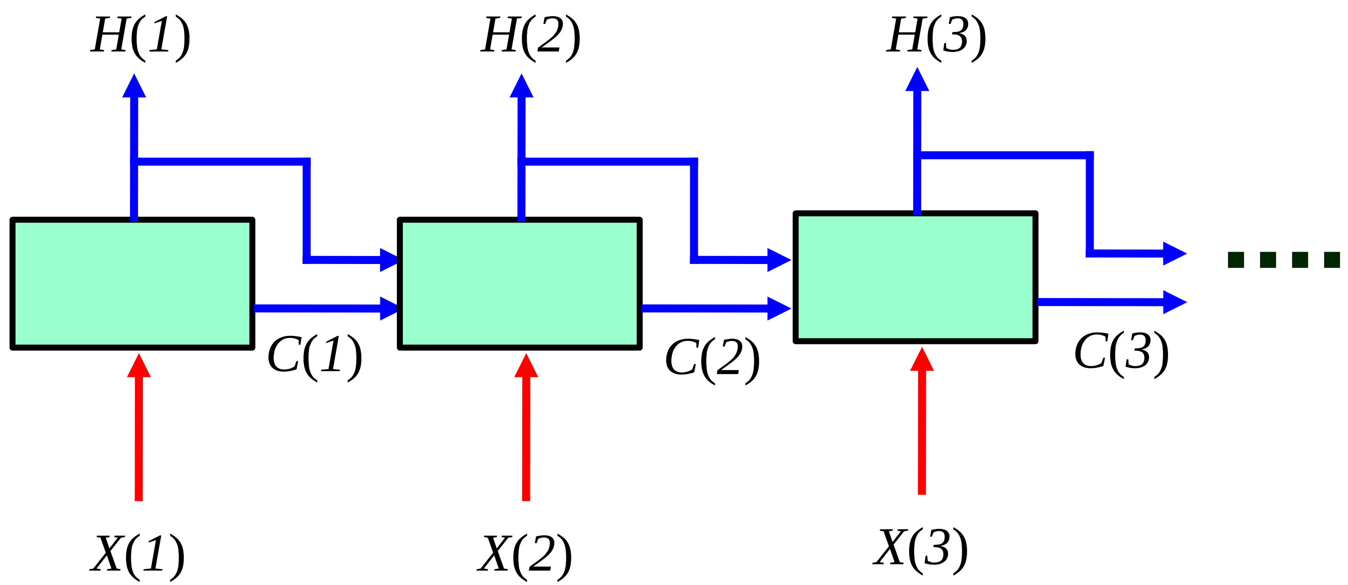
$$C_t = \left(F_t \otimes C_{t-1}\right) \oplus \left(I_t \otimes \tilde{H}_t\right)$$

$$H_t = O_t \otimes \tanh\left(C_t\right)$$

$\oplus$ = Element-wise addition

$\otimes$ = Element-wise multiplication

LSTM Unfolding



For more on LSTM…..

Tutorials:
http://colah.github.io/posts/2015-08-Understanding-LSTMs/
https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714
https://skymind.ai/wiki/lstm#long
https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/

Beyond the LSTM

The LSTM system is too complicated:
Simpler versions of gated recurrent networks have been proposed. A very commonly used one is the Gated Recurrent Unit (GRU) model

Even LSTM is not that great with learning the right context:
When an output depends on context from several steps ago, LSTM and GRU still have problems learning the right dependences.
Solution: ***Attention*** – Look explicitly at not only the current hidden state but also at past hidden states, and learn which ones are important.

Attention is computationally expensive:
Keeping track of current and past hidden states explicitly makes the learning problem very large.
Solution: The ***transformer model*** – uses a CNN-style method to turn attention-based learning into a parallel rather than sequential process.

For more, see: https://towardsdatascience.com/transformers-141e32e69591