CS60010: Deep Learning

Spring 2021

Sudeshna Sarkar

Recurrent Neural Network – Part 2

LSTM

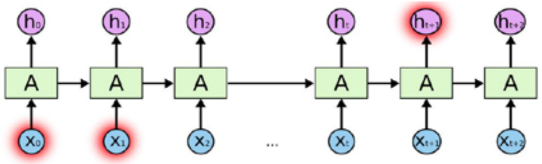
22 Feb 2021

RNN for Language Modelling

- Predict the next word

Problem of Long-Term Dependencies

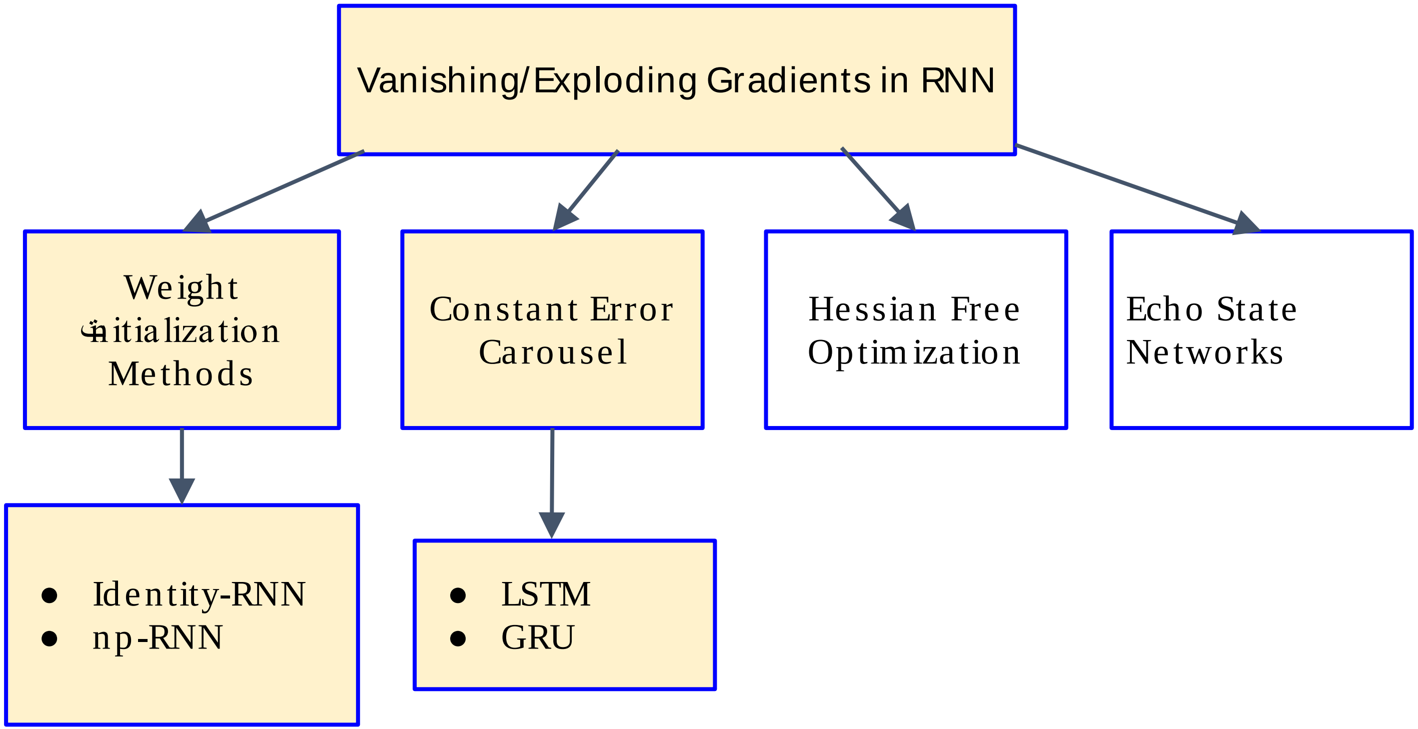
- I did my schooling from Kolkata though my family is from Tamil Nadu. I speak fluent ____.”
  - We need the context from further back.
  - Large gap between relevant information and point where it is needed



Vanishing Gradient : The problem of Long-term Dependencies

- Vanishing Gradient problem
- Multiply many small numbers together
- Errors due to further back time steps have smaller and smaller gradients

Addressing Vanishing / exloding gradients



Some strategies

- Design a model that operates at multiple time scales
  - Some parts of the model operate at fine-grained time scales and can handle small details
  - Other parts operate at coarse time scales and transfer information from the distant past to the present more efficiently.
- Strategies for building both fine and coarse time scales: Addition of skip connections across time
  - Add direct connections from variables in the distant past to variables in the present: Construct RNNs with longer delays
  - Introduce time delay of d
  - Gradients diminish as a function of τ/d rather than τ

Weight Initialization using Identity-RNN

- Identity RNN:
- Basic RNN with ReLU as nonlinearity
- Initialize hidden-to-hidden matrix to identity matrix
- RNNs composed of rectified linear units are relatively easy to train and are good at modeling long-range dependencies.

A Simple Way to Initialize Recurrent Networks of Rectified Linear Units by Quoc V. Le, Navdeep Jaitly, Geoffrey E. Hinton, Google; arxiv, April 2015

Gated RNNs

- **Gated RNNs** contain **gates** to control what information is passed through.

Gated Cells

LSTM, GRU, etc

LSTMs and GRUs : Gates

- Think of $\boldsymbol{h}$ as a state memory: you read the content h_{t-1} to write new content at h_t.
- At each computation step, the entire state is read, and the entire is possibly rewritten
- Solution: provide more controlled “memory access” with gates

$$\underset{\mathbf{s}'}{\begin{bmatrix}8\\11\\3\\7\\5\\15\end{bmatrix}} \leftarrow \underset{\mathbf{g}}{\begin{bmatrix}0\\1\\0\\0\\0\\1\end{bmatrix}} \odot \underset{\mathbf{x}}{\begin{bmatrix}10\\11\\12\\13\\14\\15\end{bmatrix}} + \underset{(\mathbf{1}-\mathbf{g})}{\begin{bmatrix}1\\0\\1\\1\\1\\0\end{bmatrix}} \odot \underset{\mathbf{s}}{\begin{bmatrix}8\\9\\3\\7\\5\\8\end{bmatrix}}$$

Using binary gate vector g to control access to memory s'.

- Gate **g** specifies what element of **x** should be copied in memory **s**.
- **(1 – g)** specifies what should be kept.

$\odot$: Hadamard product
Elementwise multiplication

These gates serve as building blocks : But they have to be differentiable.

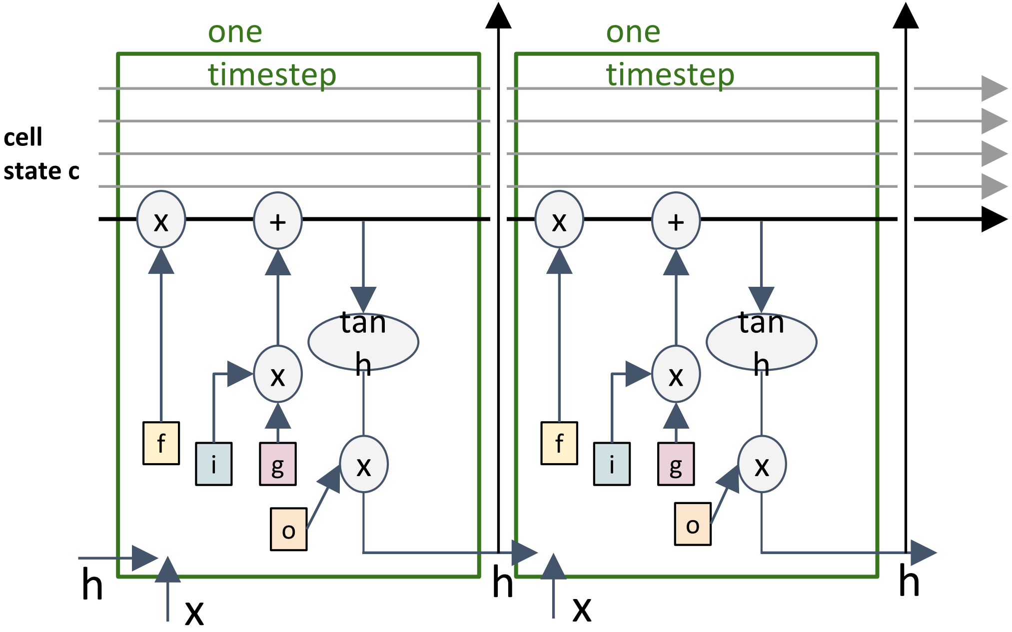
Replace $g \in \{0,1\}^n$ with $g' \in R^n$ which is then passed through a sigmoid function $\sigma(g') \in \{0,1\}^n$

LSTMs: learnable gates

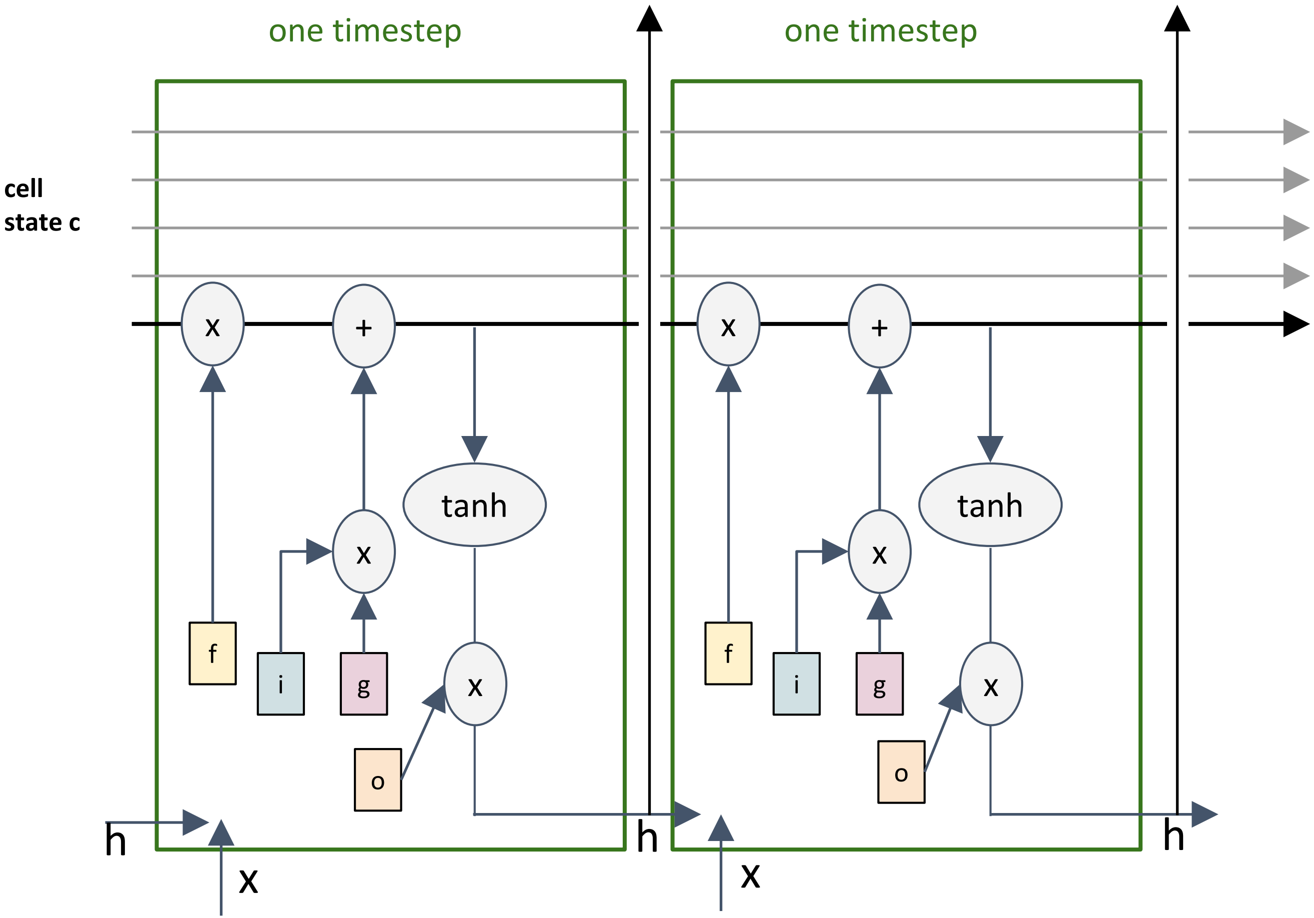
These gates serve as building blocks : But they have to be differentiable.

Replace $g \in \{0,1\}^n$ with $g' \in R^n$ which is then passed through a sigmoid function $\sigma(g') \in \{0,1\}^n$

Split the hidden layer into two vectors **c** and **h** and have three learnable gates



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



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

Long short-term memory

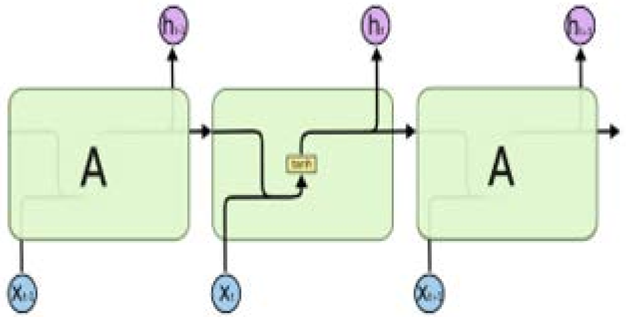
- Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber (1997)

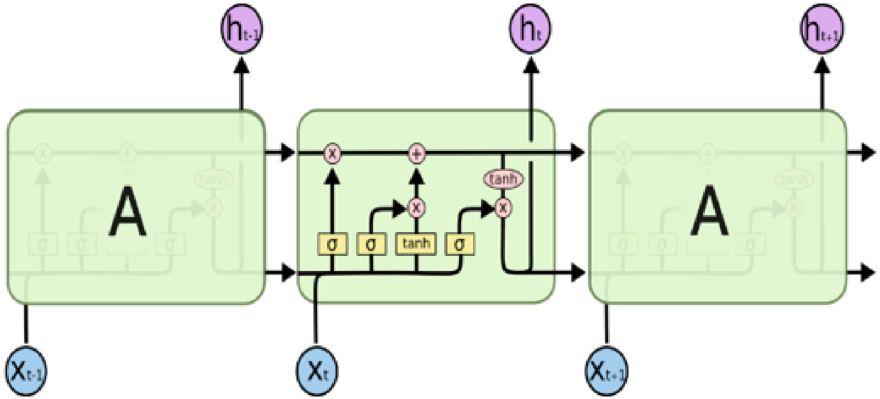
LSTMs operate using a series of gates. These gates control what information is still relevant to the network.

LSTMs introduce self-loops to produce paths where the gradient can flow for long durations.

Basic RNN unit

LSTM unit





$$h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

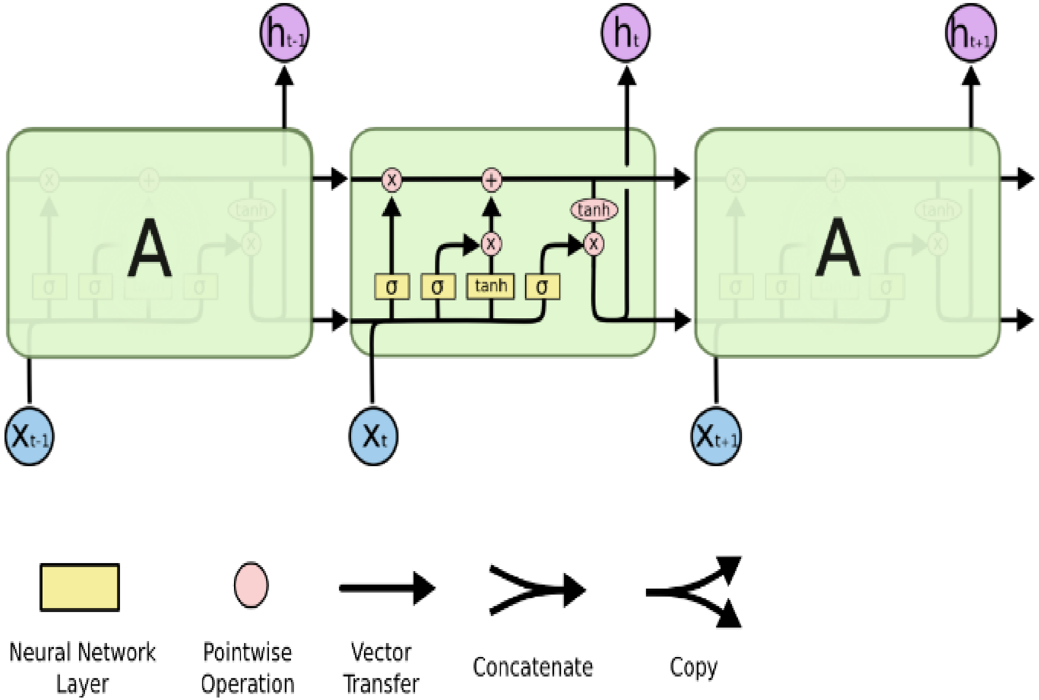
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

LSTMs operate using a series of gates. These gates control what information is still relevant to the network.

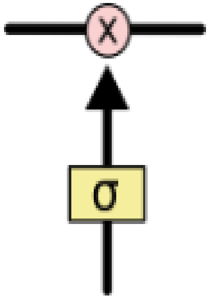
LSTMs introduce self-loops to produce paths where the gradient can flow for long durations.

Long short-term memory



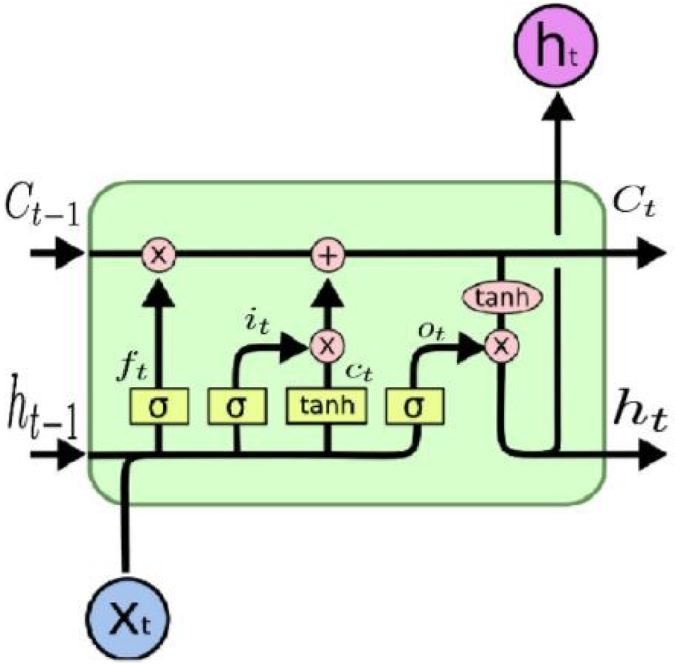
Information added or removed through gates.

Ex: sigmoid net and pointwise multiplication



Long short-term memory

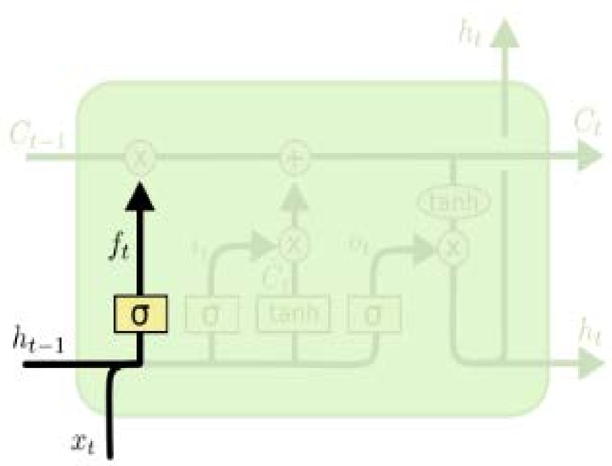
(1) Forget (2) Input (3) Update (4) Output



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

It runs straight down the entire chain, with only some minor linear interactions.

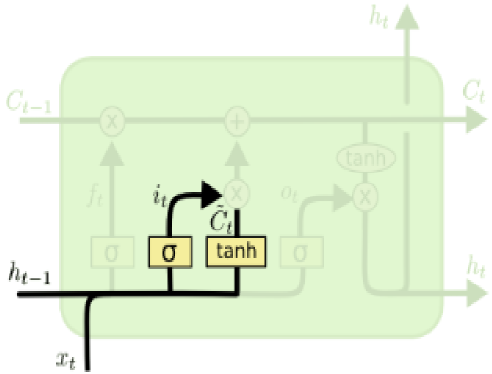
LSTM forget gate



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

Forget irrelevant parts of the previous state

Long short-term memory



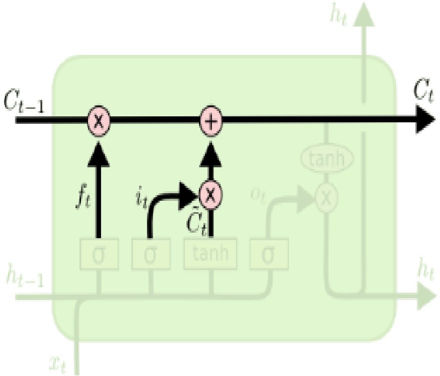
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

Next decide what new information we're going to store in the cell state. This has two parts.

1. a sigmoid layer called the "input gate layer" decides which values we'll update.
2. a tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state.

Long short-term memory

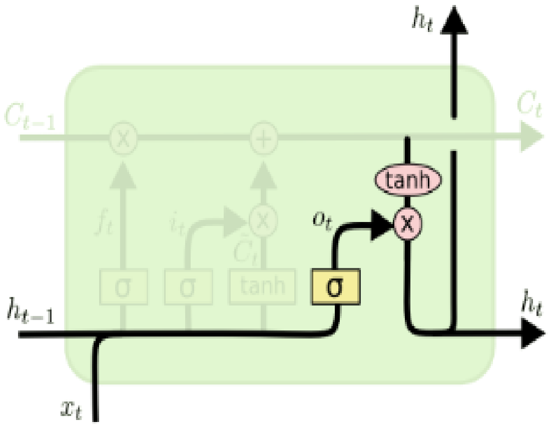


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

It’s now time to update the old cell state, C_{t-1}, into the new cell state C_t.

C_{t-1} C_t

Long short-term memory

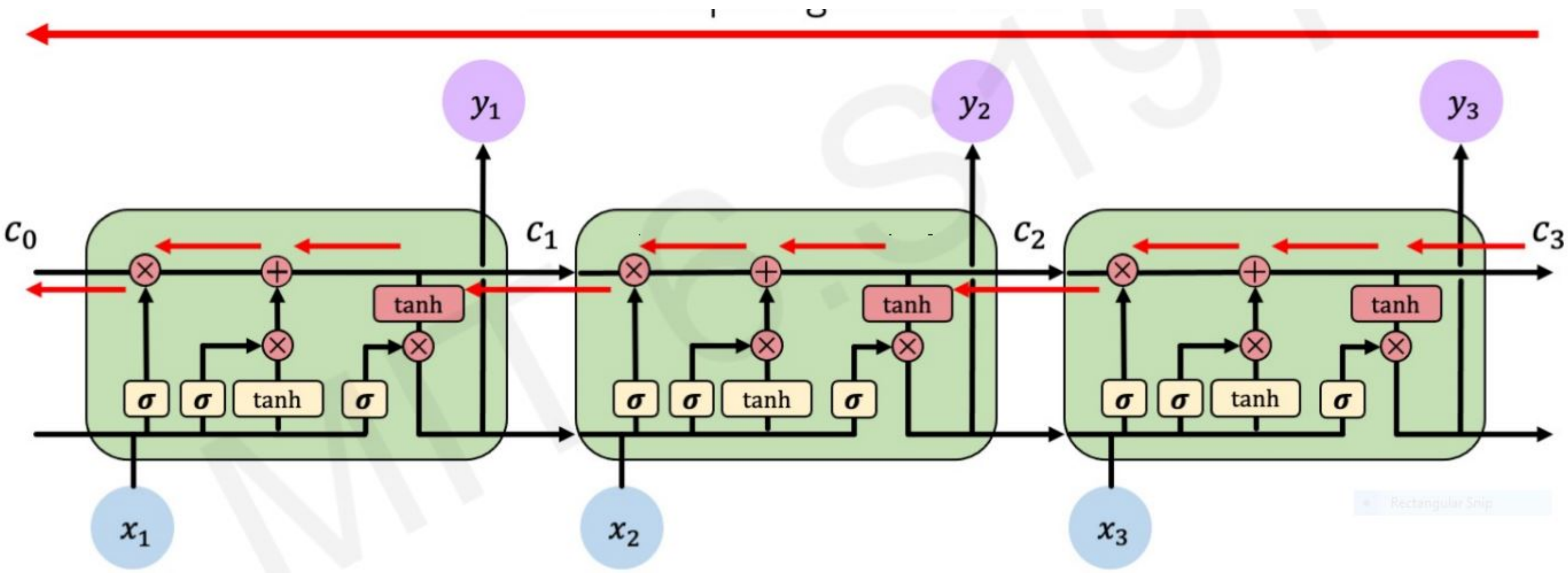


$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right)$$

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

Finally, we need to decide what we're going to output.

LSTM gradient flow is uninterrupted.



Backpropagation from c_t to c_{t-1} only elementwise multiplication by f, no matrix multiply by W

Application: Language Model

Machine Translation

Image Captioning