COMP 5630:Machine Learning

Lecture 12: LSTM, GRU

10/3/2024

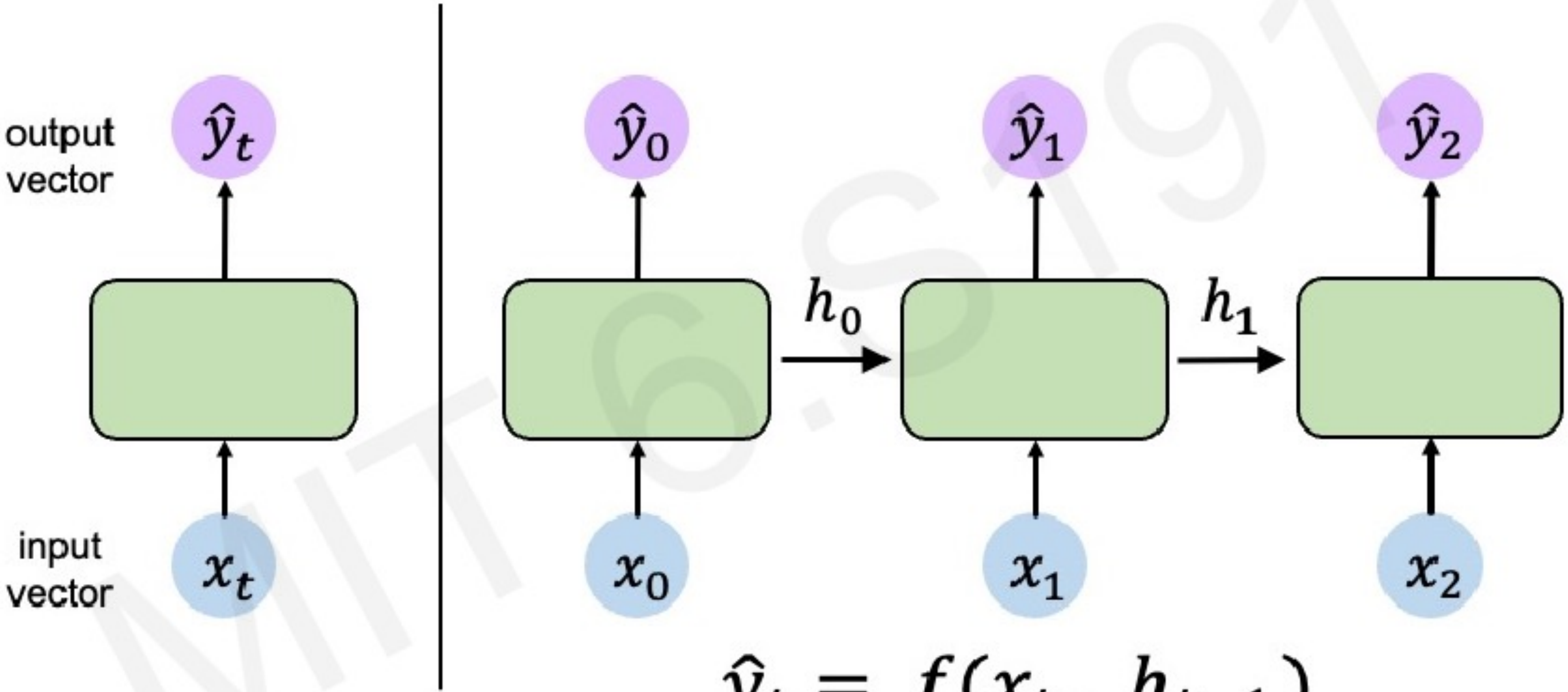
Handouts

- Deep learning
  - https://www.deeplearningbook.org/
  - Part II: Modern Practical Deep Networks

- Interpret Bayes factor for model selection
  - Uploaded on Canvas

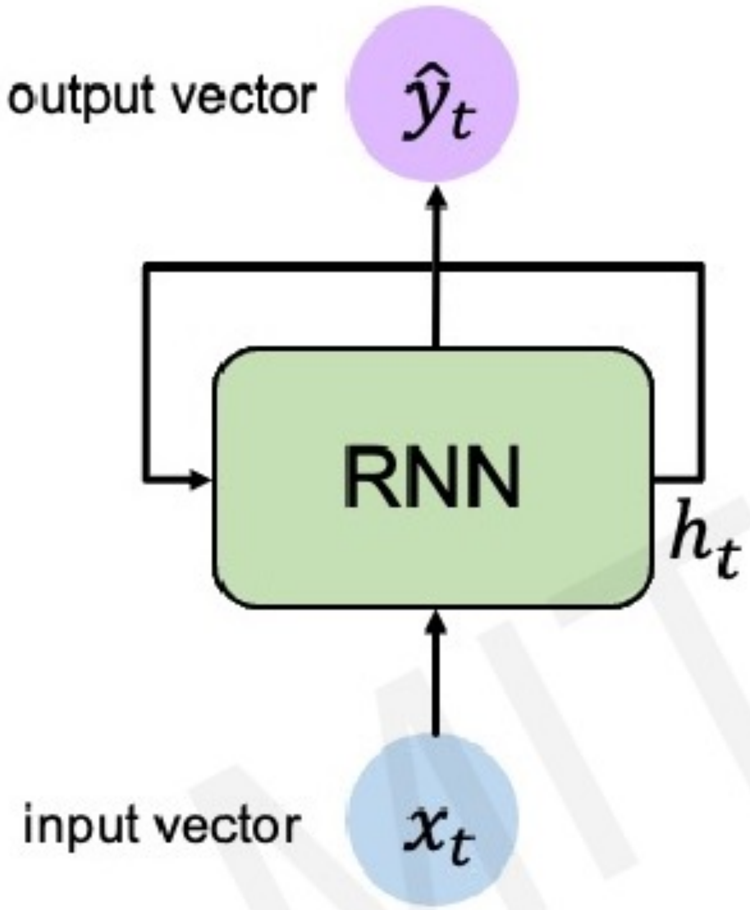
RNN Gradient Flow, limitations, LSTM

Neurons with Recurrence



$$\underbrace{\hat{y}_t}_{\text{output}} = f(\underbrace{x_t}_{\text{input}},\ \underbrace{h_{t-1}}_{\text{past memory}})$$

Recurrent Neural Networks (RNNs)



Apply a **recurrence relation** at every time step to process a sequence:

$$\boxed{h_t} = \boxed{f_W}(\boxed{x_t}, \boxed{h_{t-1}})$$

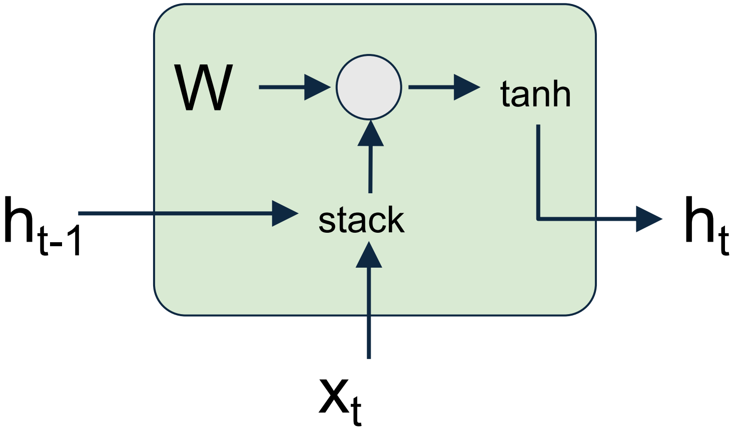
cell state — function with weights W — input — old state

Note: the same function and set of parameters are used at every time step

RNNs have a **state**, h_t, that is updated **at each time step** as a sequence is processed

Vanilla RNN Gradient Flow

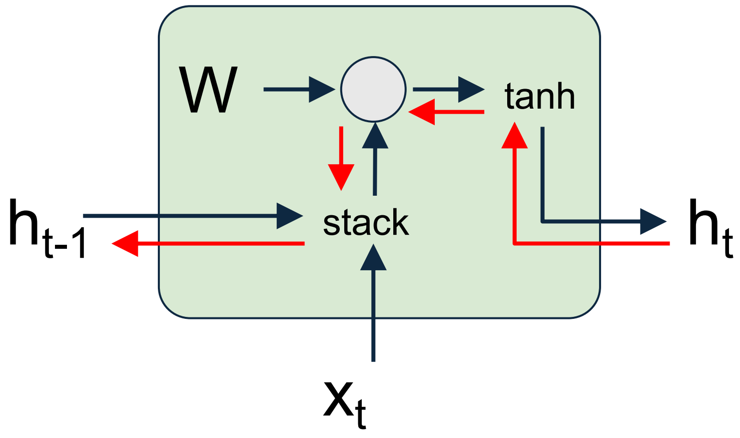
Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013



$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$
$$= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

Vanilla RNN Gradient Flow

Backpropagation from h_t to h_{t-1} multiplies by W (actually W_{hh}^T)

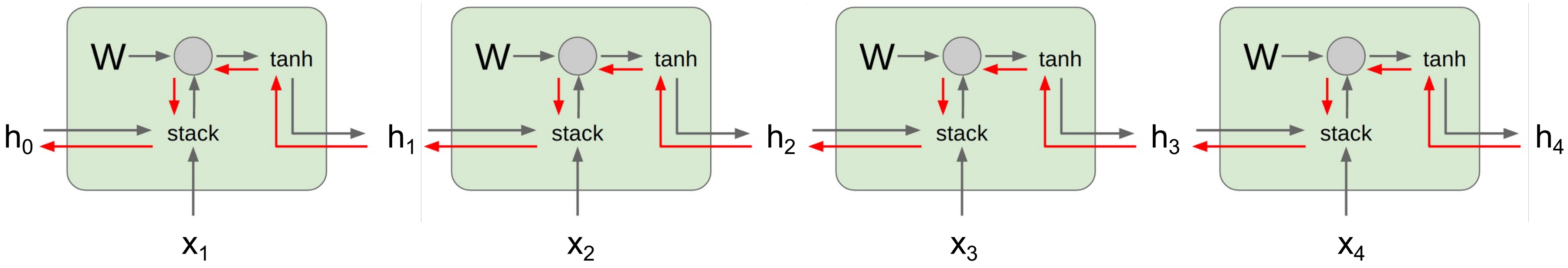


Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

$$
\begin{aligned}
h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\
&= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \\
&= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)
\end{aligned}
$$

Vanilla RNN Gradient Flow

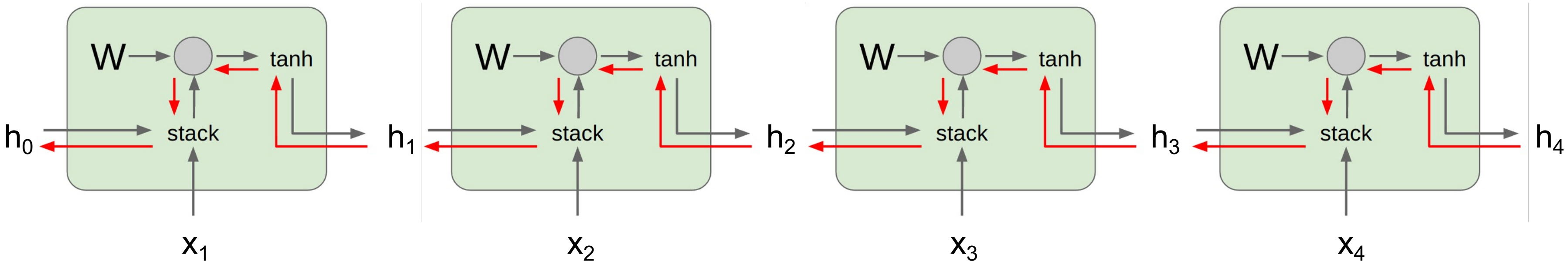
Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Vanilla RNN Gradient Flow

Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013



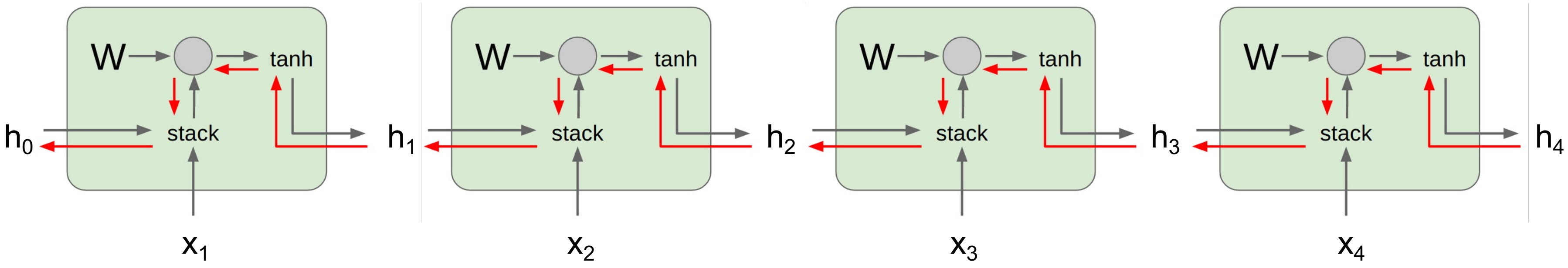
Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

Vanilla RNN Gradient Flow

Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

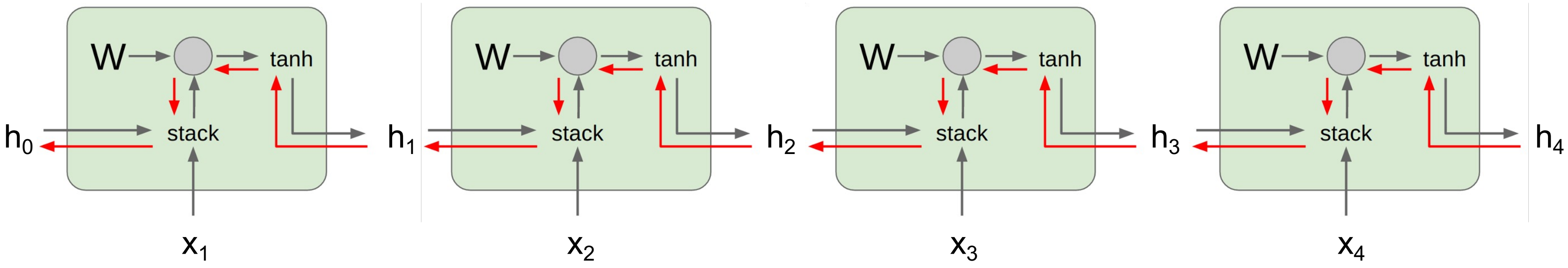
Gradient clipping: Scale gradient if its norm is too big

```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
  grad *= (threshold / grad_norm)
```

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

→ Change RNN architecture

Limitations of RNN

- RNNs can not handle long term dependencies in practice

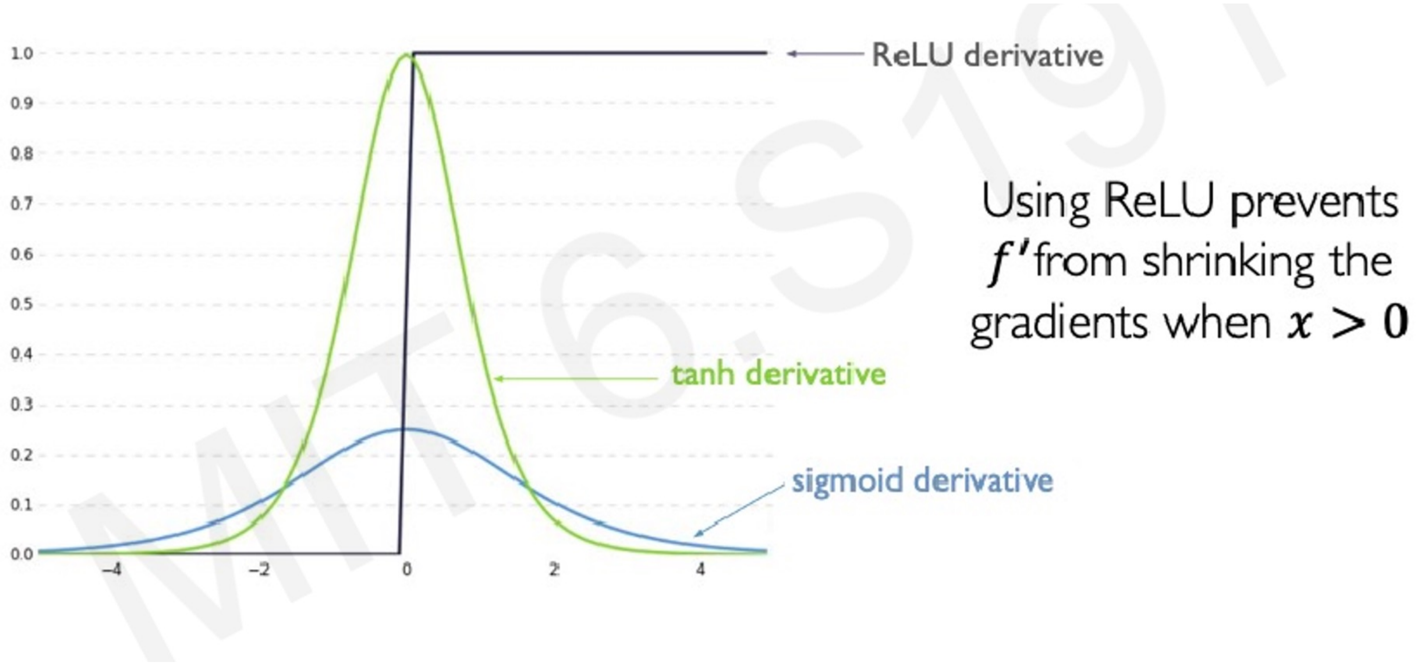
- Example task: next word prediction by RNN, vocabulary of English
  1. Bird fly in the _? [*sky*]
  2. I was born in Germany and I can speak fluently in_ ? [*German*]

  ➢ Sentence 1: sky is predicted from the previous words fly and bird
  ➢ Sentence 2: *German* needs to be predicted from the word Germany

- Gap between the relevant information and the sequence output place is large for Sentence 2
  - RNNs become unable to learn and connect the information
  - Because of vanishing or exploding gradient problem

How to RNN Handle Limitations?

1. Choose a different activation function



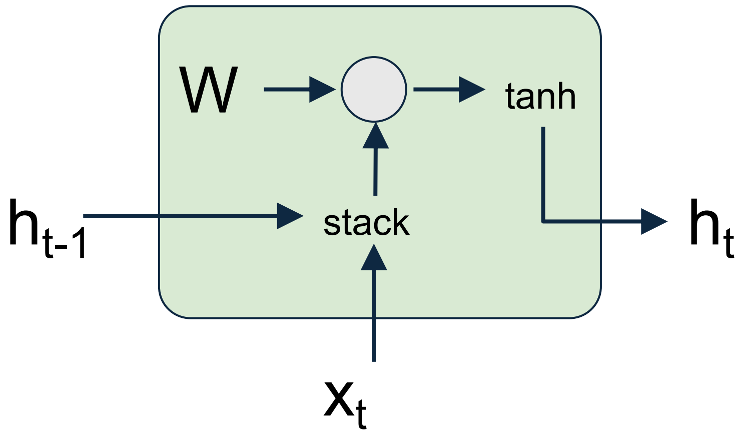
How to RNN Handle Limitations?

2. Initialize weight of the matrices to identity matrices

3. Change the RNN structure

Long Short-Term Memory (LSTM)

- The recurrence block of an RNN has only one tanh activation



Long Short Term Memory (LSTM)

The recurrence block of an LSTM has four components

Vanilla RNN

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

LSTM

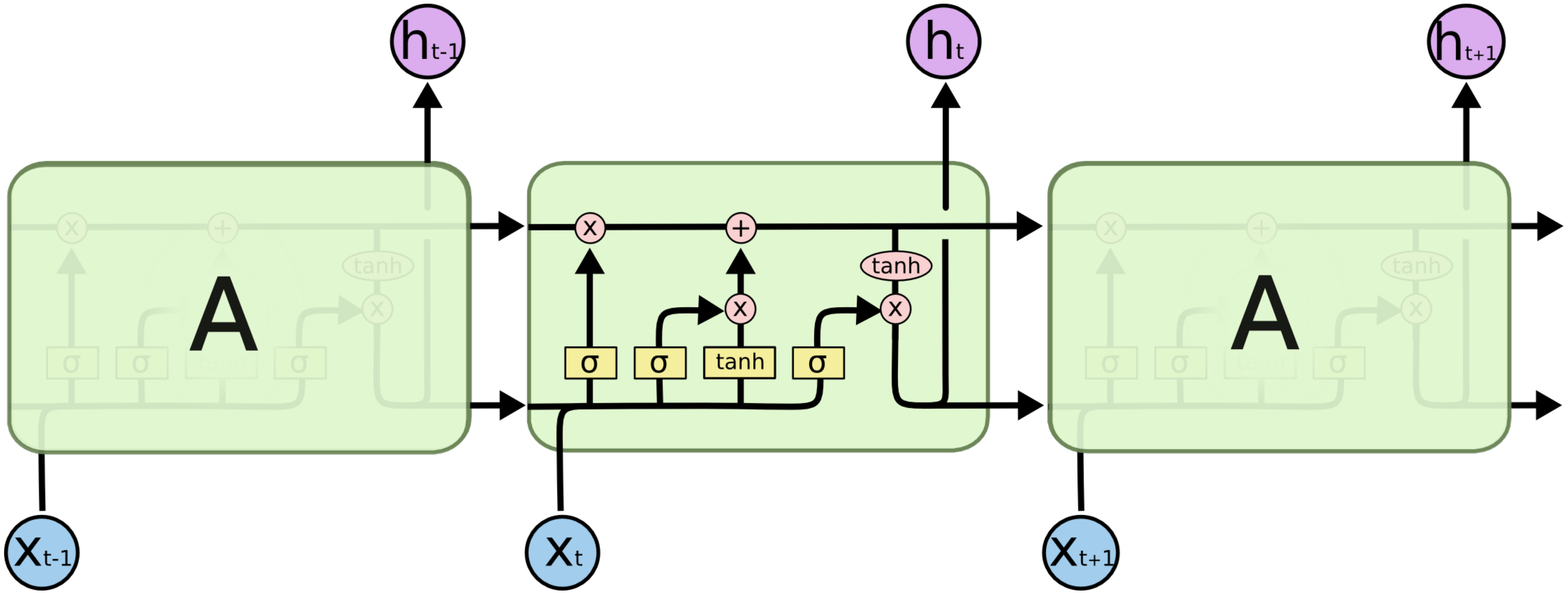
$$\begin{pmatrix}i\\ f\\ o\\ g\end{pmatrix} = \begin{pmatrix}\sigma\\ \sigma\\ \sigma\\ \tanh\end{pmatrix} W \begin{pmatrix}h_{t-1}\\ x_t\end{pmatrix}$$

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

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

Hochreiter and Schmidhuber, "Long Short Term Memory", Neural Computation 1997

LSTM Structure



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTMs: Key Concepts

1. Maintain a **cell state**
2. Use **gates** to control the **flow of information**
   - **Forget** gate gets rid of irrelevant information
   - **Store** relevant information from current input
   - Selectively **update** cell state
   - **Output** gate returns a filtered version of the cell state
3. Backpropagation through time with partially **uninterrupted gradient flow**

LSTM

- **Step 1:** The LSTM receives the input vector (xt) and the previous state (ht-1, ct-1).

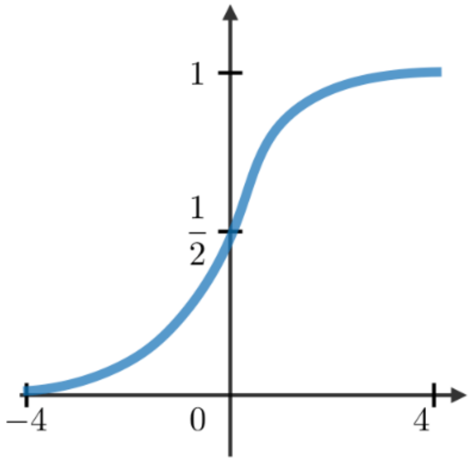
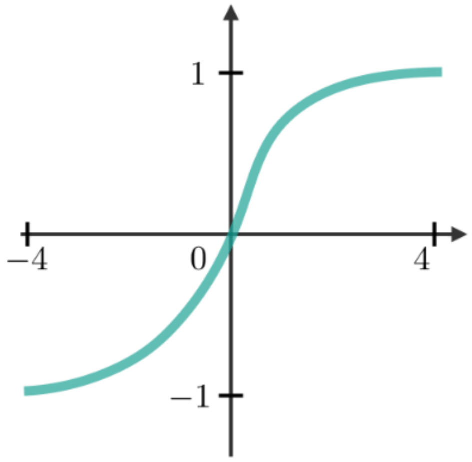
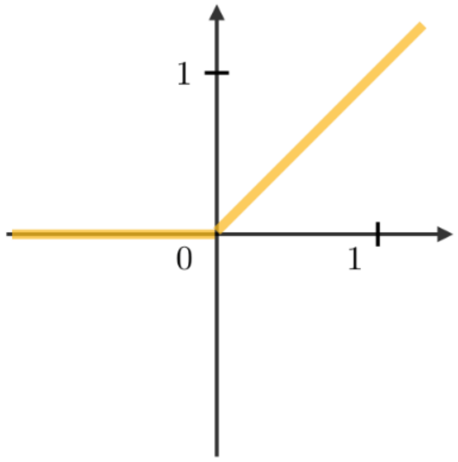
- **Step 2:** The forget gate (ft) decides what information to discard from the cell state. It uses the input vector and the previous hidden state to generate a number between 0 and 1 for each number in the cell state ct-1.
  - A 1 represents "completely keep this“
  - A 0 represents "completely get rid of this".

  - *Which activation function is good for this task?*

https://www.linkedin.com/pulse/understanding-lstm-in-depth-look-its-architecture-pros-babu-thomas/

Activation Functions

❐ **Commonly used activation functions** — The most common activation functions used in RNN modules are described below:

Sigmoid	Tanh	RELU
$g(z)=\dfrac{1}{1+e^{-z}}$	$g(z)=\dfrac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	$g(z)=\max(0,z)$
		

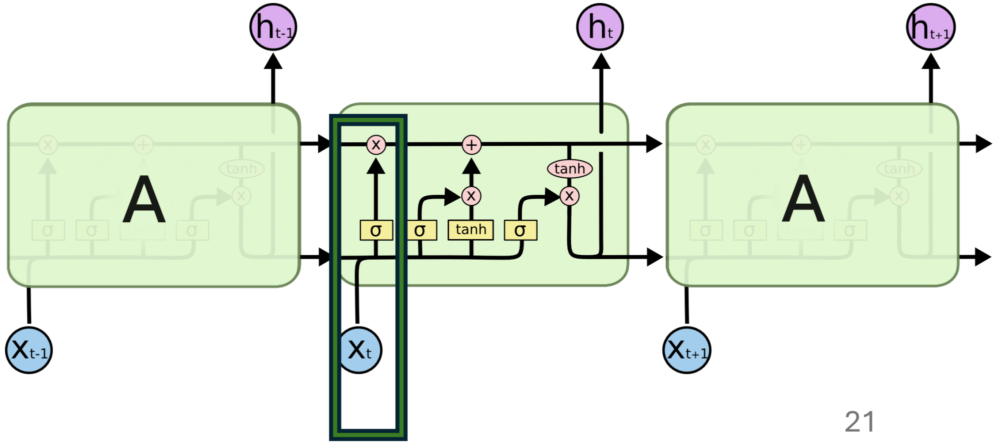
https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

LSTM

- **Step 1:** The LSTM receives the input vector (xt) and the previous state (ht-1, ct-1).

- **Step 2:** The forget gate (ft) decides what information to discard from the cell state. It uses the input vector and the previous hidden state to generate a number between 0 and 1 for each number in the cell state ct-1.
  - A 1 represents "completely keep this“
  - A 0 represents "completely get rid of this".

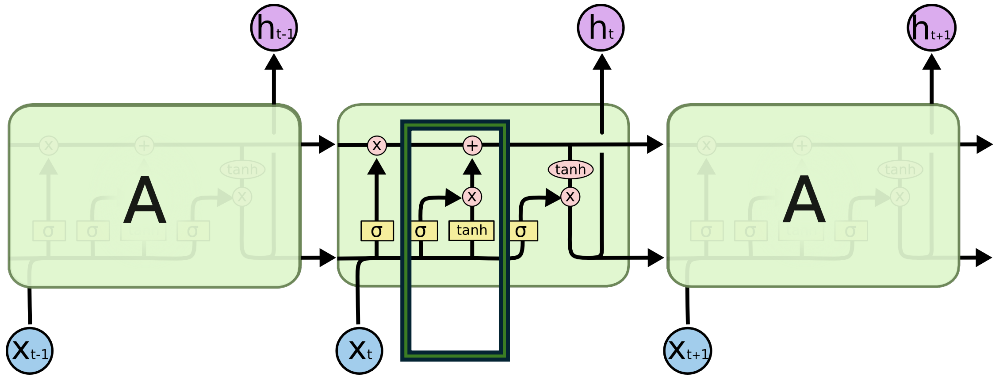
- Forget Gate: ft = σ(Wf.[ht-1, xt] + bf)



LSTM

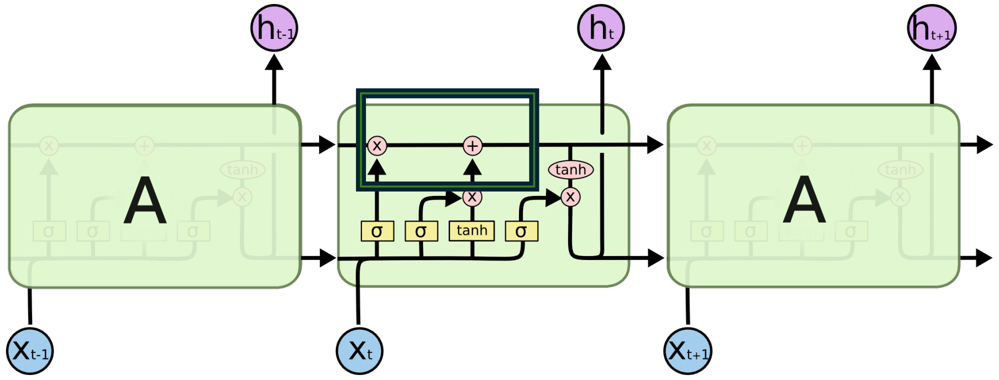
- **Step 3:** The input gate (it) decides what new information to store in the cell state. It has two parts.
  - A sigmoid layer called the "input gate layer" decides which values we'll update,
  - A tanh layer creates a vector of new candidate values (Ct~) that could be added to the state.

- Input Gate: it = σ(Wi.[ht-1, xt] + bi)
- Candidate Values(Cell State Update): Ct~ = tanh(Wc.[ht-1, xt] + bc)



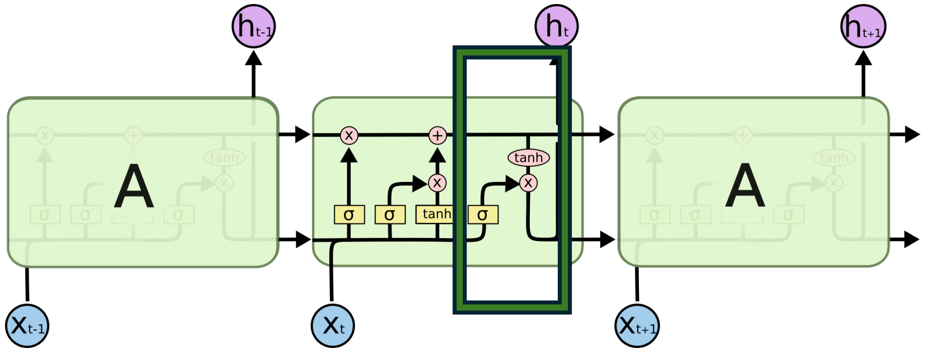
LSTM

- **Step 4:** Update the old cell state (ct-1) to the new cell state (ct).
  - The old cell state is multiplied by ft to forget the things we decided to forget earlier.
  - Then we add the new candidate values, scaled by how much we decided to update each state value.
- Cell State(Final Cell State): ct = ft * (ct-1) + it * Ct~



LSTM

- **Step 5:** Output: a filtered version of the cell state
  - First, we run a sigmoid layer to decide what parts of the cell state we're going to output.
  - Then, we put the cell state through tanh (to push the values to be between -1 and 1) and multiply it by the output of the sigmoid gate, so we only output the parts we decided to.
- Output Gate: $o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o)$
- Hidden State: $h_t = o_t * \tanh(c_t)$



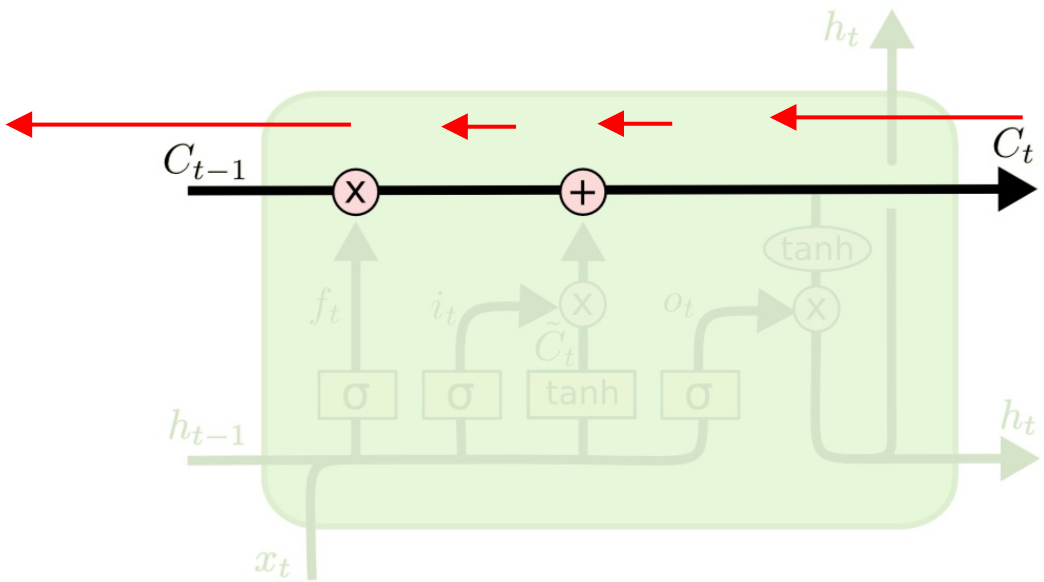
LSTM

- Why tanh activation in the input gate processing?
  - To overcome the vanishing gradient problem
  - The tanh activation function's derivative can sustain for a long range before going to zero.
  - The tanh, with-1 or +1 outputs, determines whether to decrement or increment items in the cell state.
    - Over time, the cell state may get exceedingly large or small, which could hinder the network's ability to learn.
    - The network may learn more quickly since the tanh function makes sure that the values in the cell state are always within a suitable range (-1 to 1).

LSTM

- Why sigmoid activation in the forget gate?
  - Sigmoid outputs 0 or 1, it can be used to forget or remember the information.

LSTMs Intuition: Additive Updates



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

Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTN

LSTMs Intuition: Additive Updates

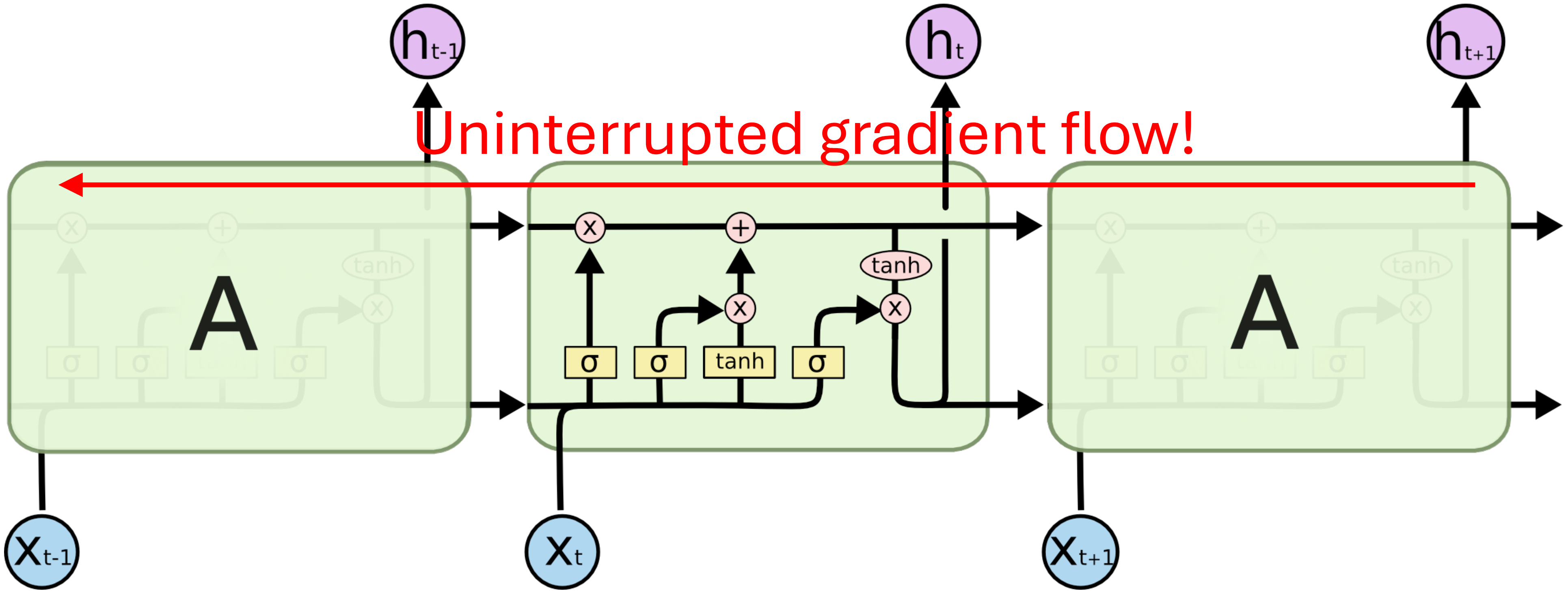
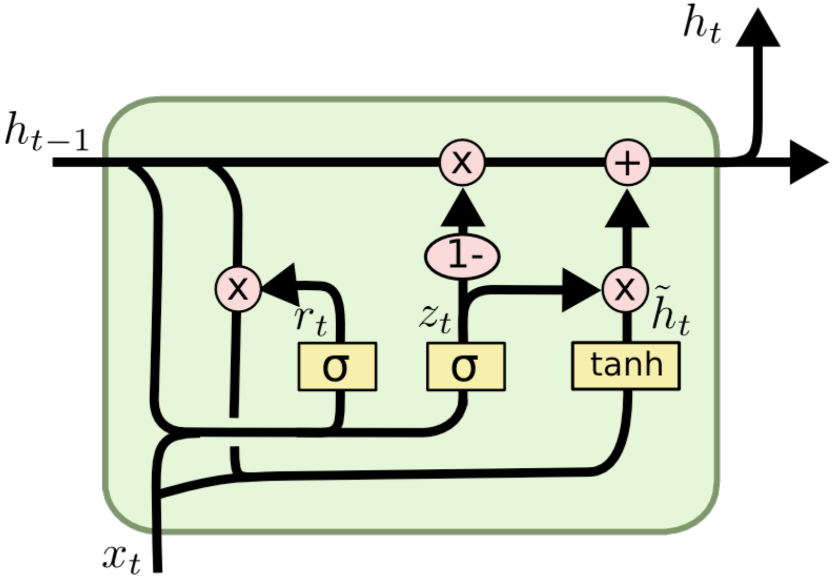


Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTN

LSTM Variants: Gated Recurrent Units (GRU)

- Changes:
  - No explicit memory; memory = hidden output
  - Z = memorize new and forget old



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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

Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTN

GRU Architecture

- 2 gates
- Update gate (z)
  - The update gate controls how much of the previous hidden state should be retained
- Reset gate (r)
  - The reset gate determines how much of the past information to forget.
- Both gates have sigmoid activation function

GRU vs LSTM

- GRU has fewer parameters
  - Does not have the forget gate.

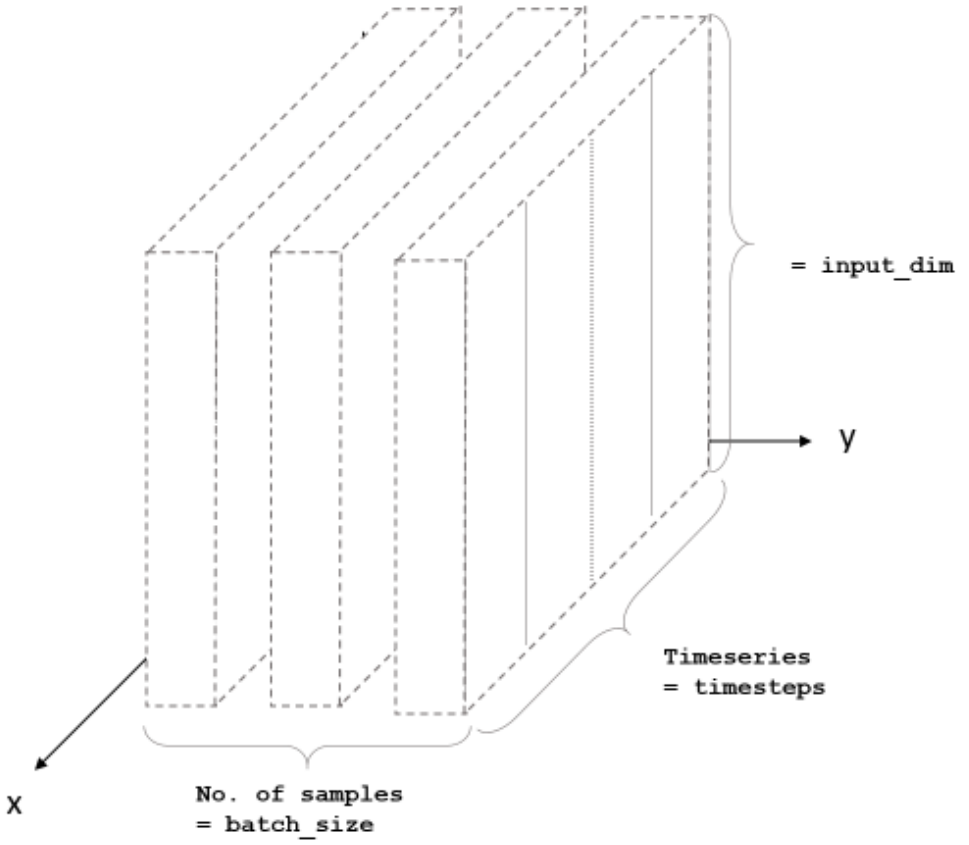
  - Computationally efficient than LSTM

  - Less prone to overfitting, making it a good choice for smaller datasets.

LSTM Demo

LSTM Shape in Keras

- the first dimension represents the batch size,
- the second dimension represents the number of time-steps
- the third dimension represents the number of units/features per time-step (input_dim)



https://www.kaggle.com/code/shivajbd/input-and-output-shape-in-lstm-keras

LSTM for Human Activity Recognition

- https://github.com/effat/MLP-Demo/blob/main/LSTM.ipynb