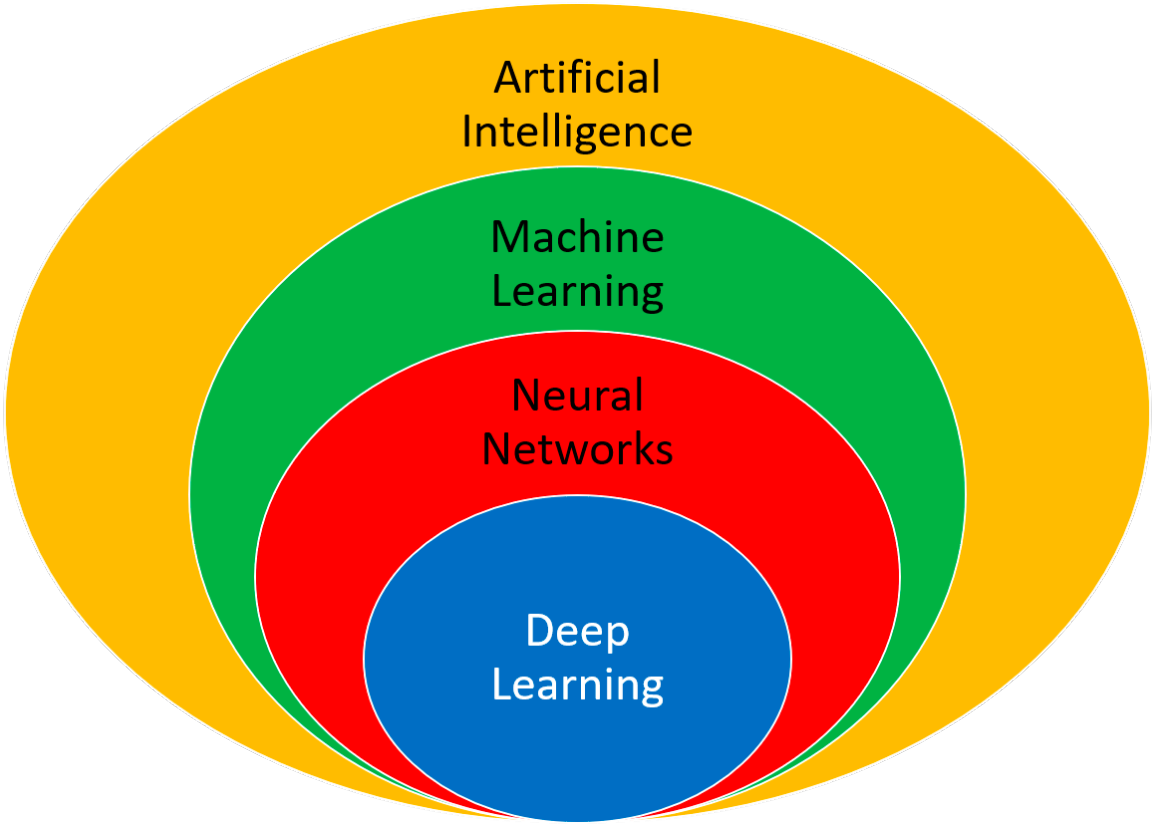
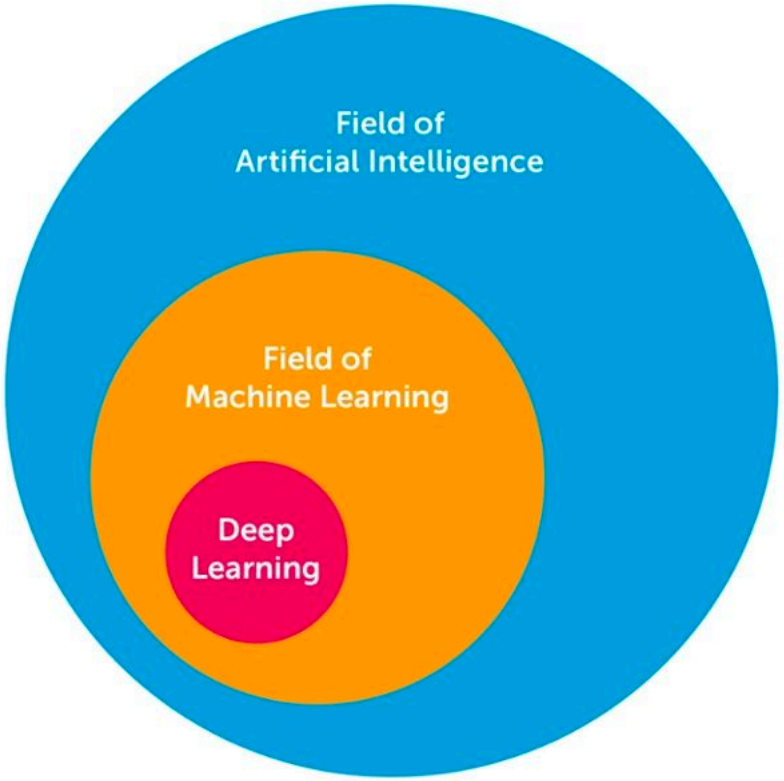
Introduction of Deep Learning with RNN and LSTM

Lê Anh Cường

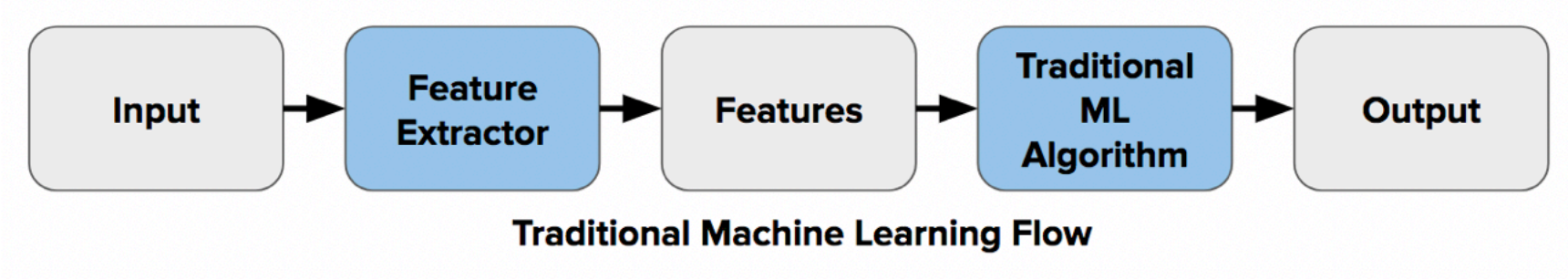
TDTU

AI, ML, NN and DL

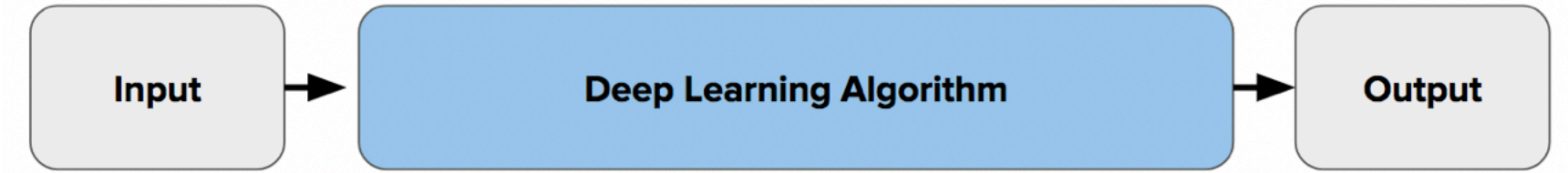




Traditional Statistical Machine Learning vs Deep Learning

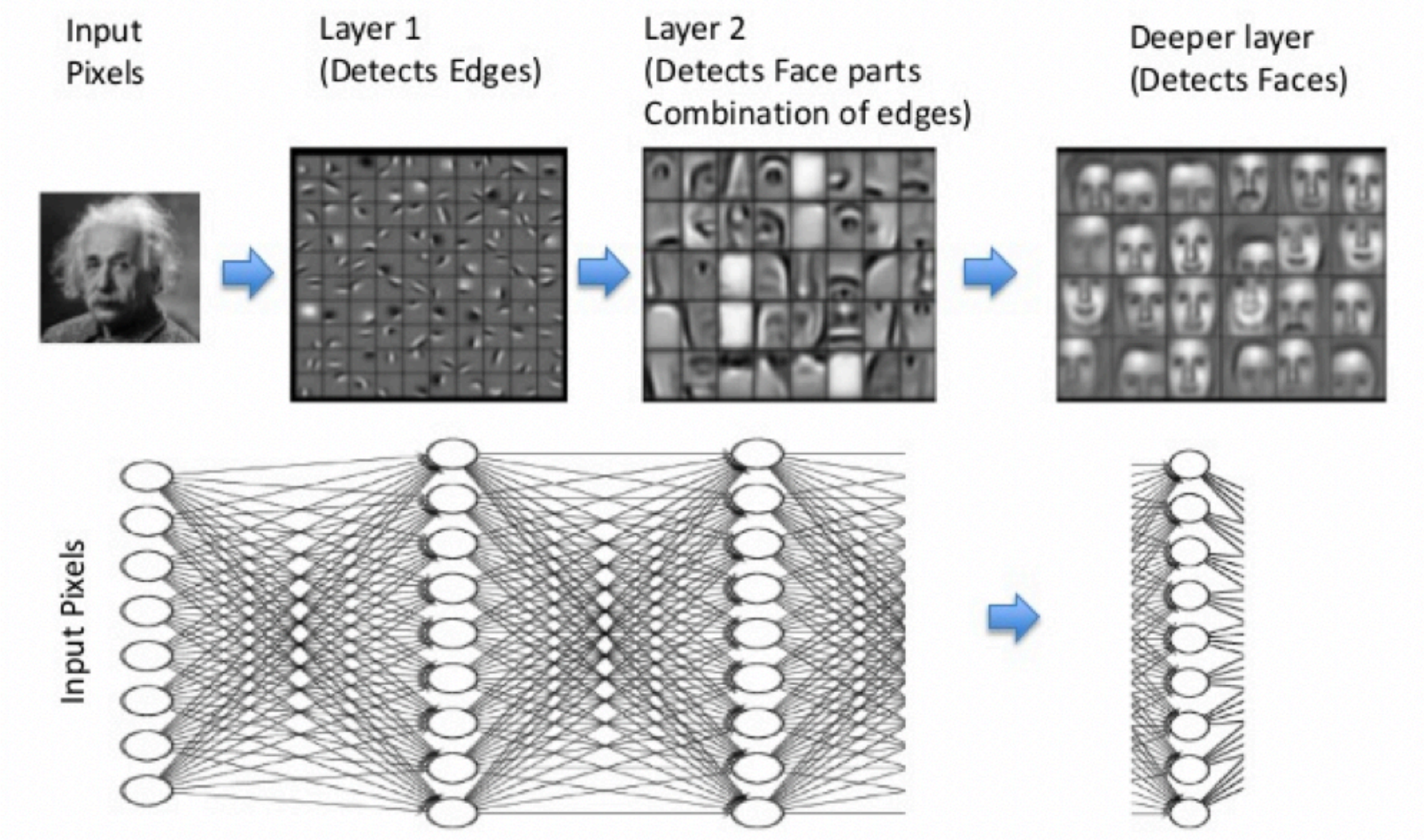


Traditional Machine Learning Flow

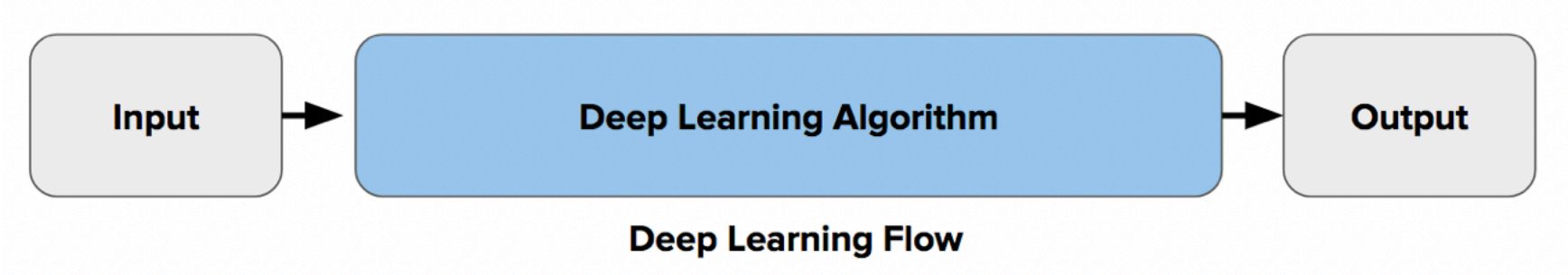


Deep Learning Flow

DL as Representation Learning

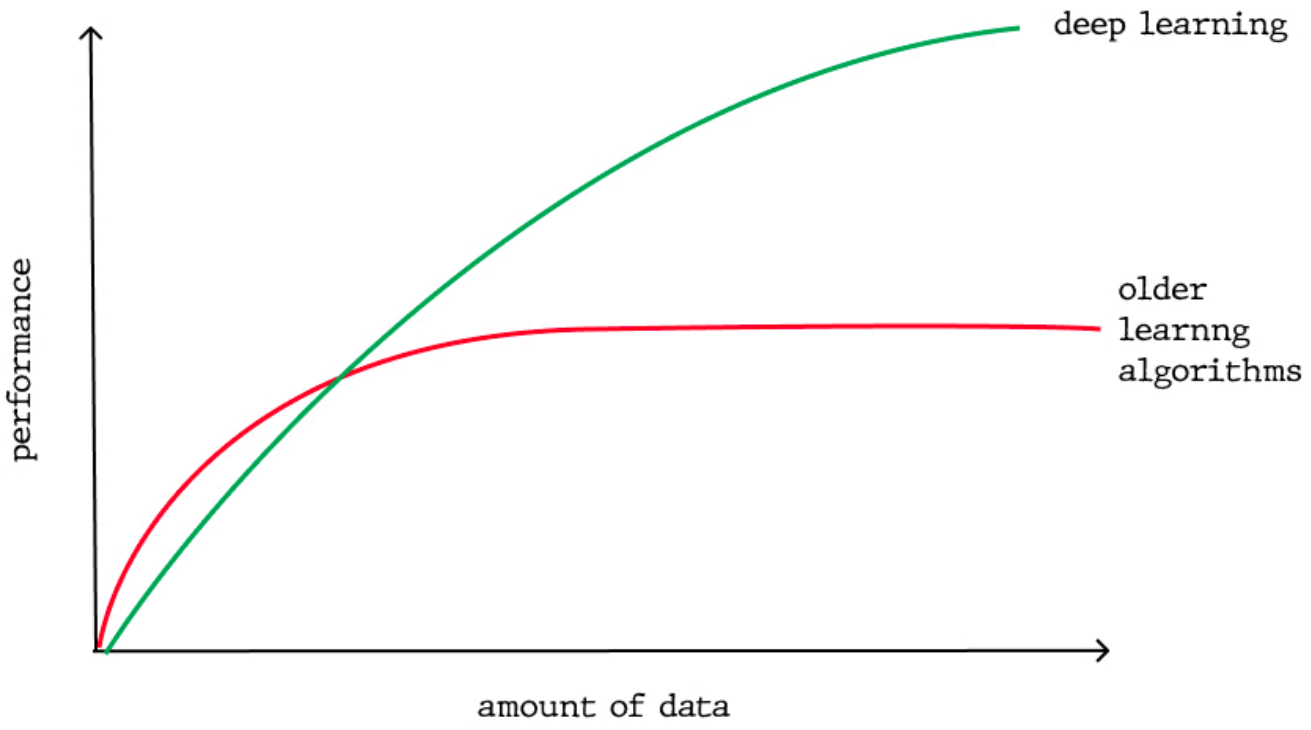


End-to-End Model

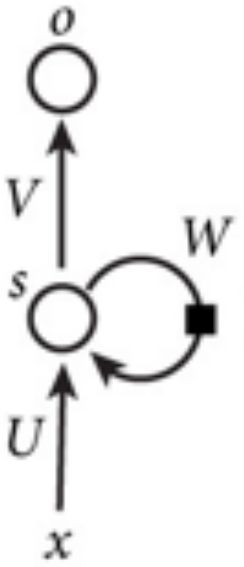


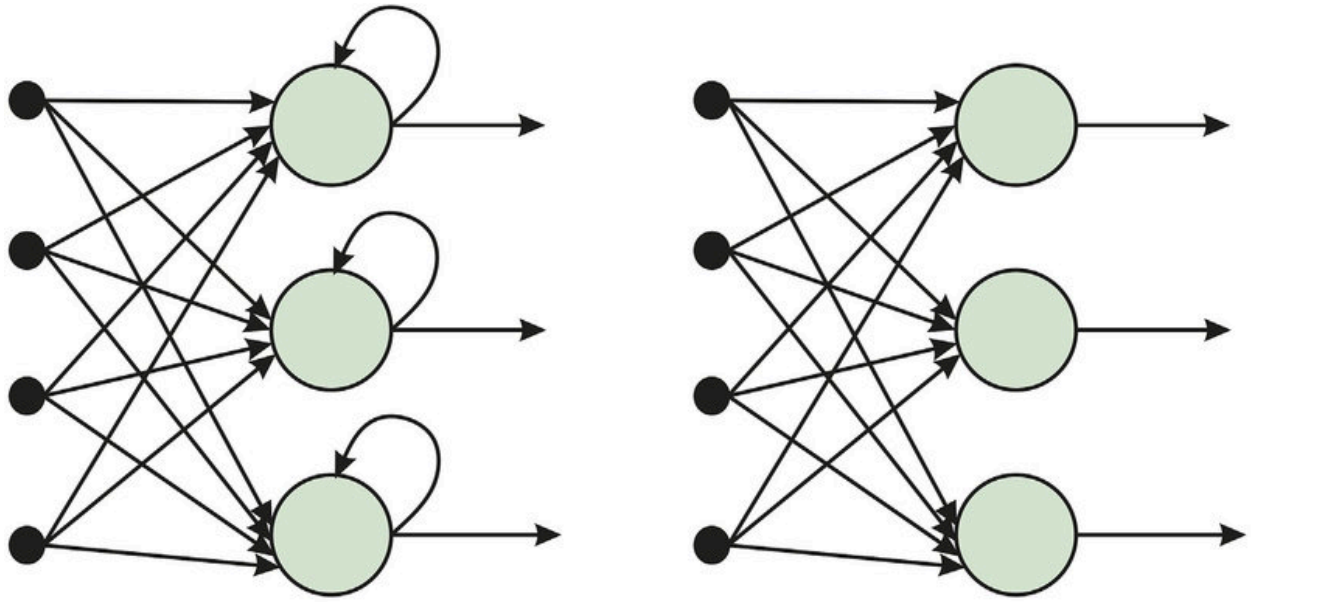
Deep Learning Flow

ML vs DL in Performance



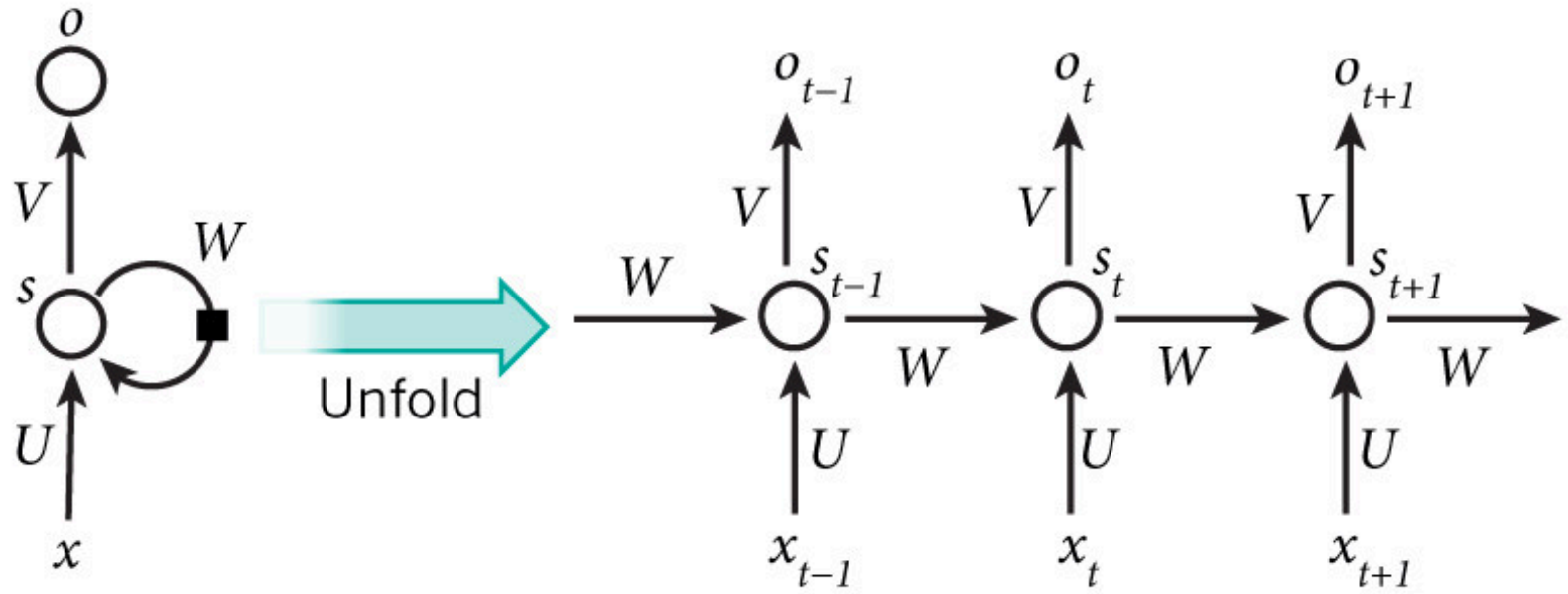
RECURRENT NEURAL NETWORKS





(a) Recurrent Neural Network (b) Feed-Forward Neural Network

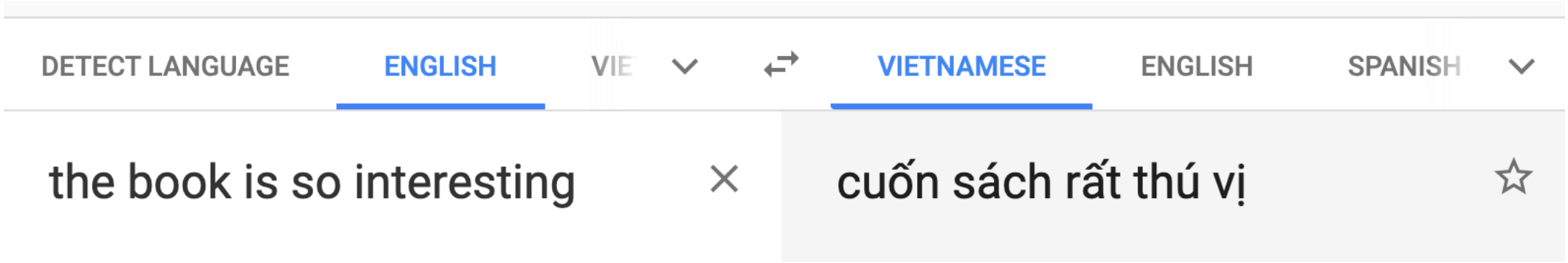
RNN for Sequential Data

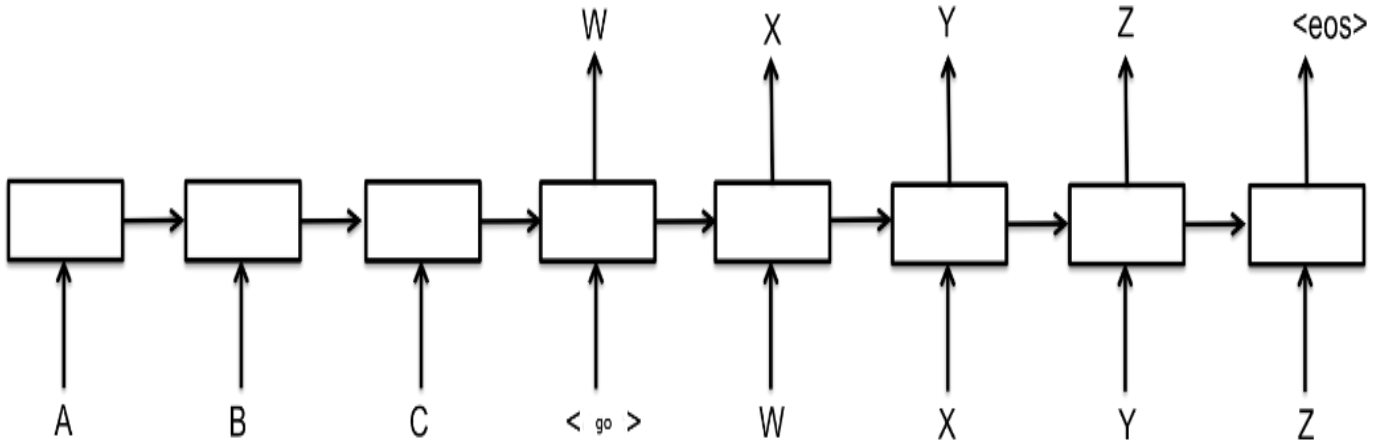


Outline

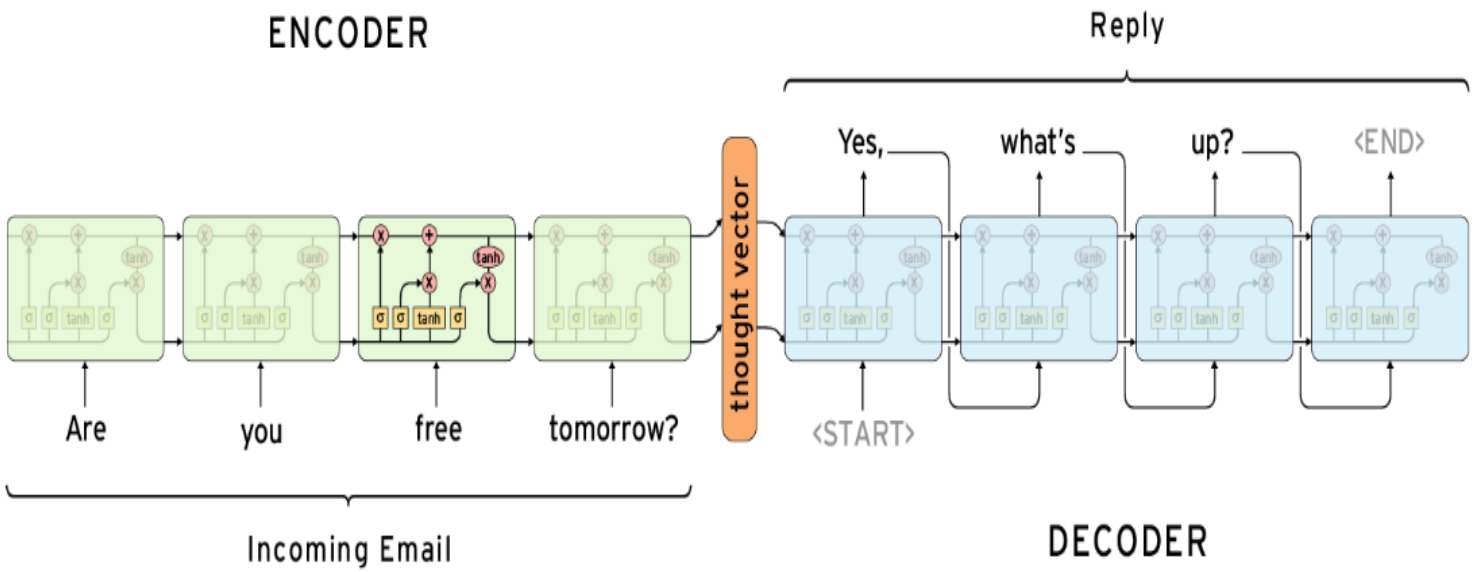
- What is RNN?
- RNN: the Architecture and Forward Computation
- RNN with Progapagation algorithm
- Long-Short Term Memory

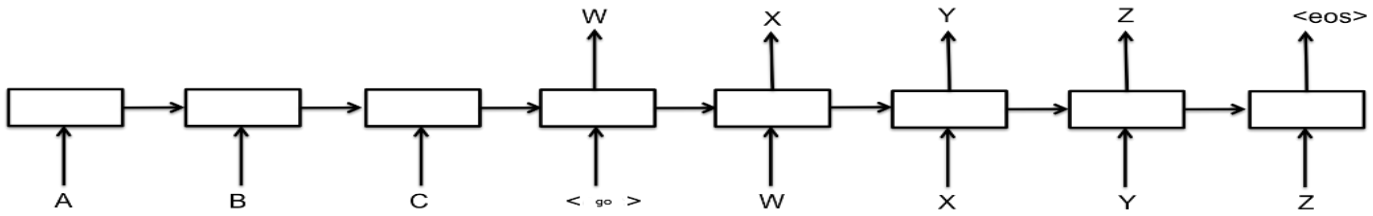
What is Recurrent Neural Network (RNN)?





Automatic Email Reply





Part Of Speech Tagging
Sentiment Analysis

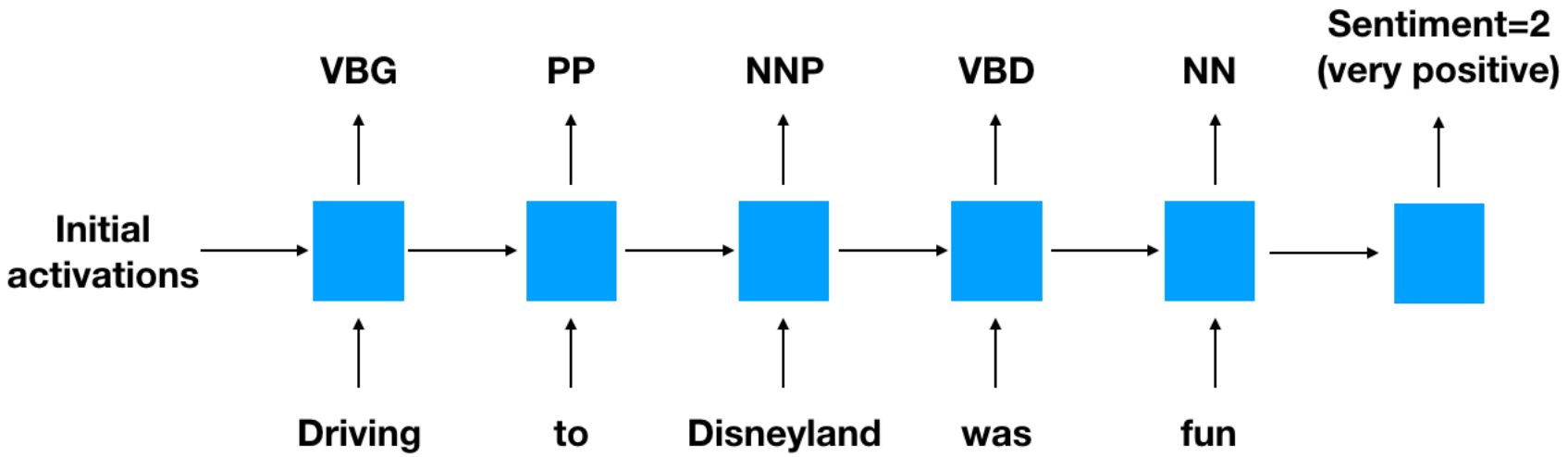
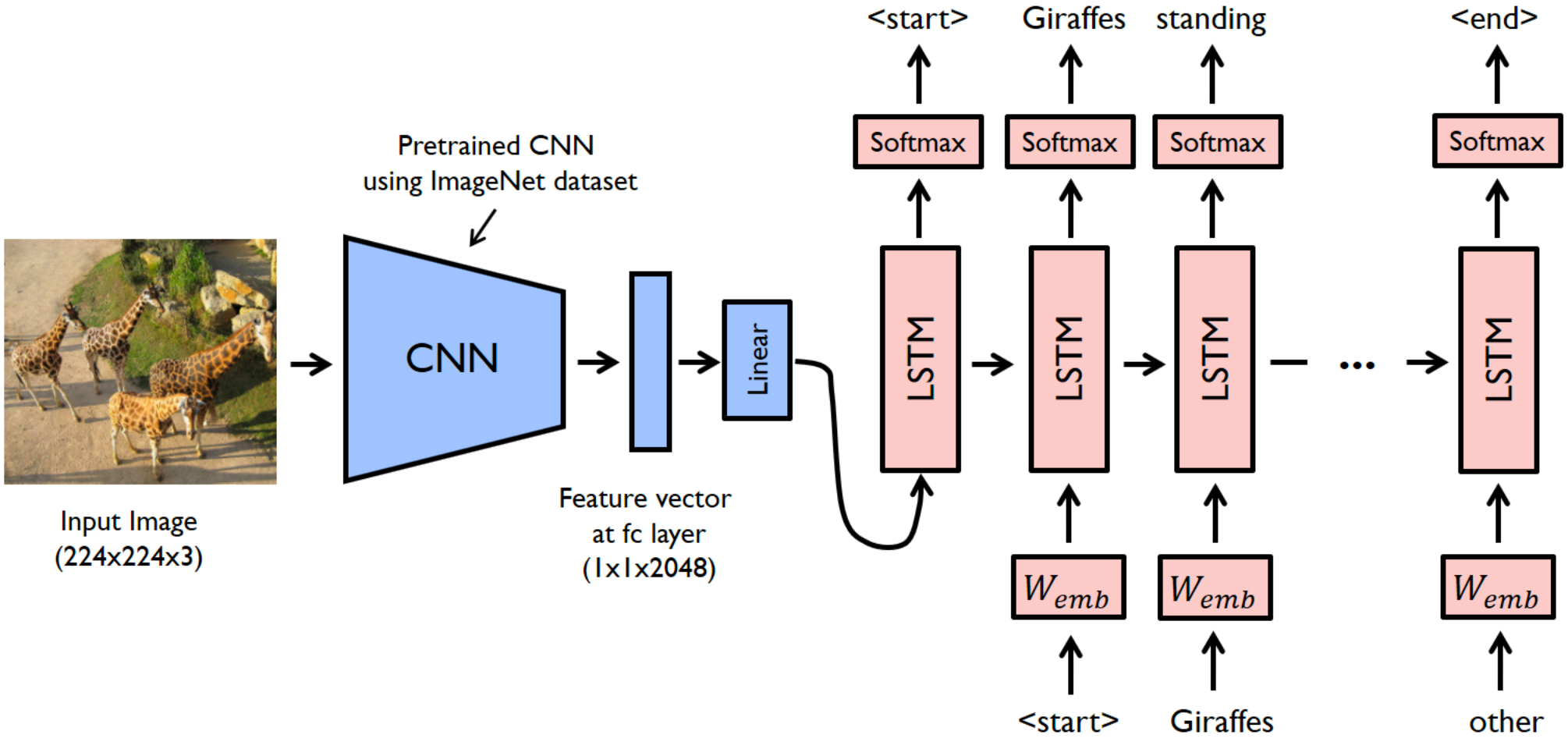
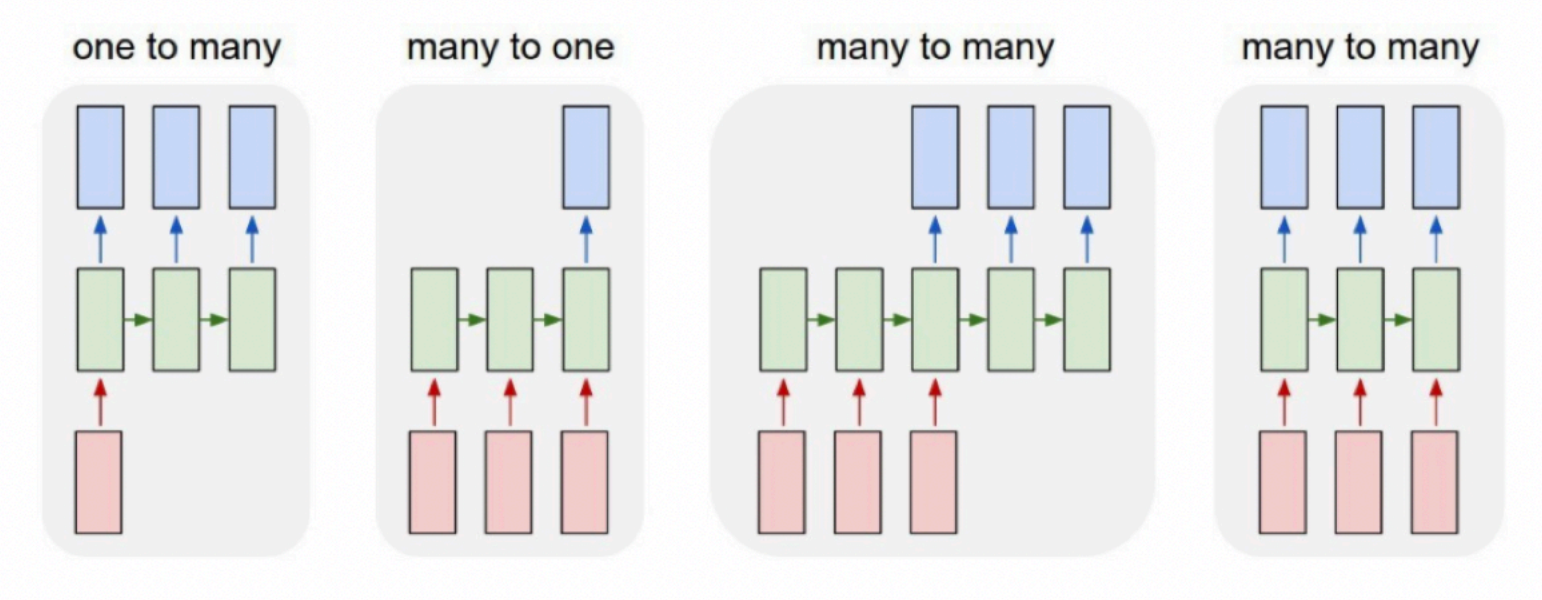


Image Captioning

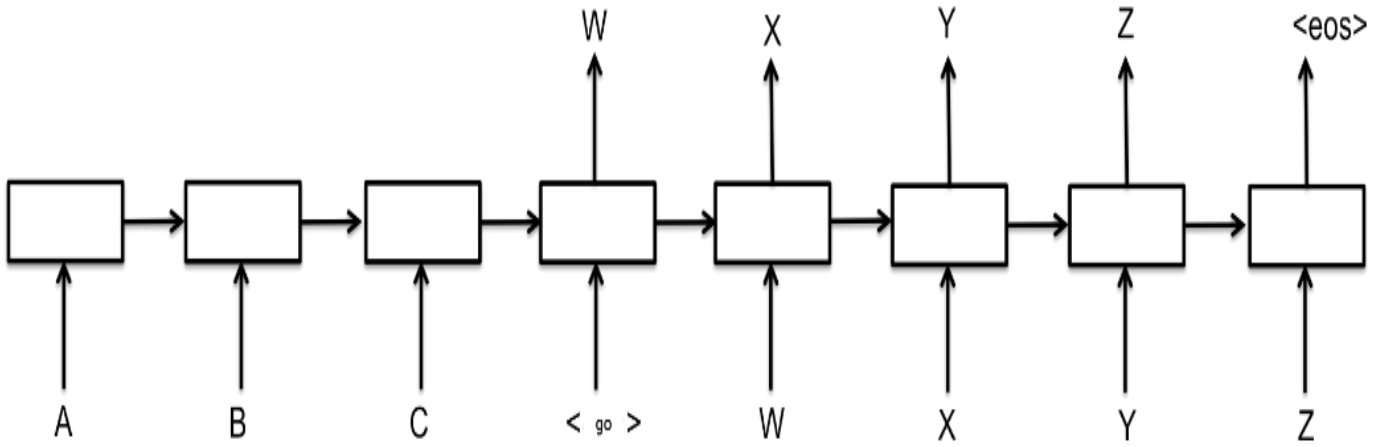


RNN Architectures

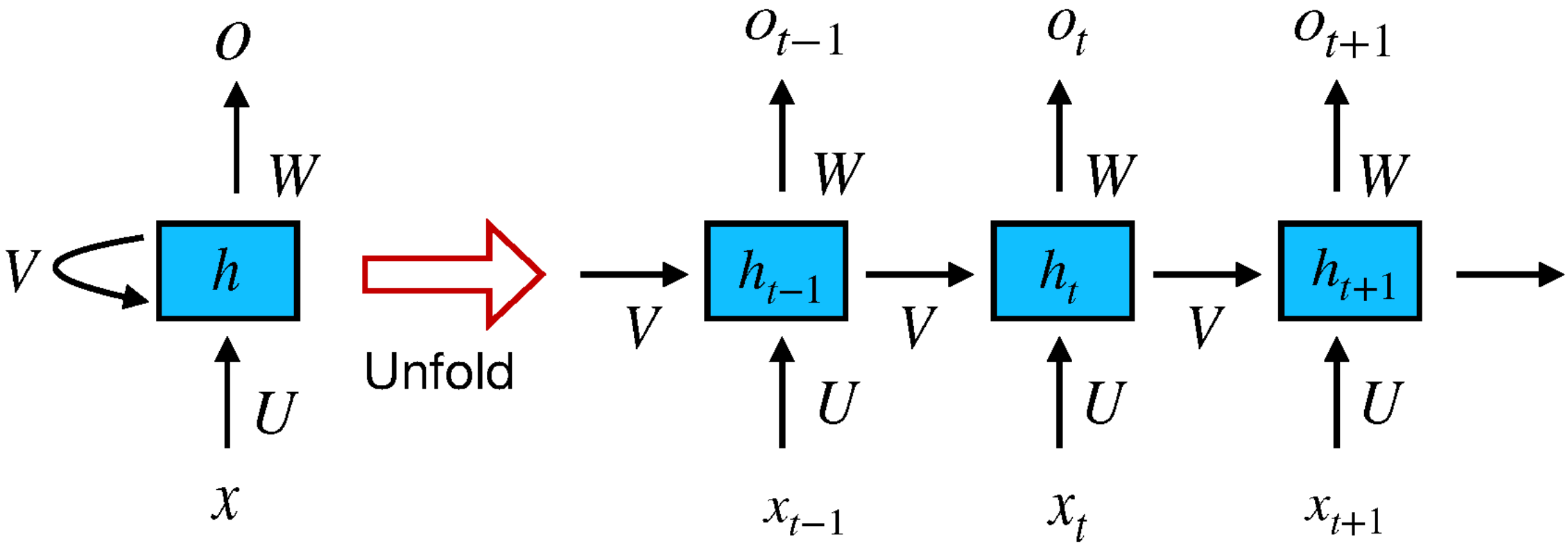


What is Recurrent Neural Network (RNN)?

- The idea behind RNNs is to make use of Sequential Information.
- RNNs are called *recurrent* because they perform the same task for every element of a sequence.
- RNNs have a "memory" which captures information about what has been calculated so far.

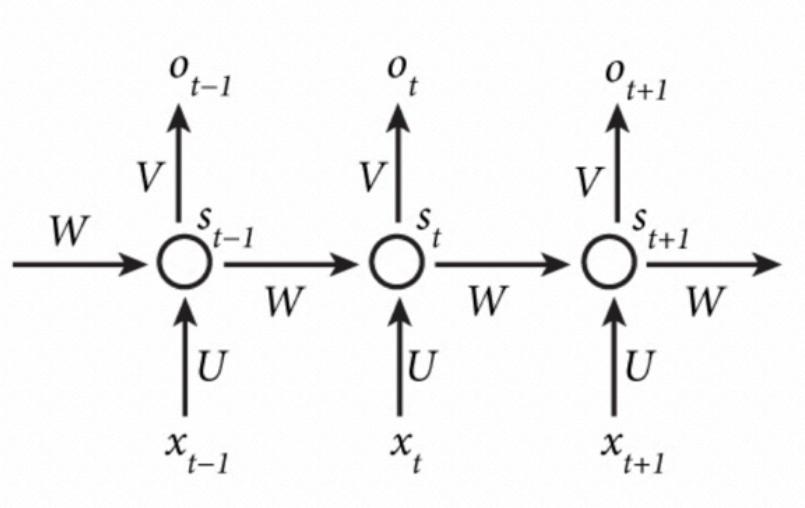


What is Recurrent Neural Network (RNN)?



Forward Computation in RNN

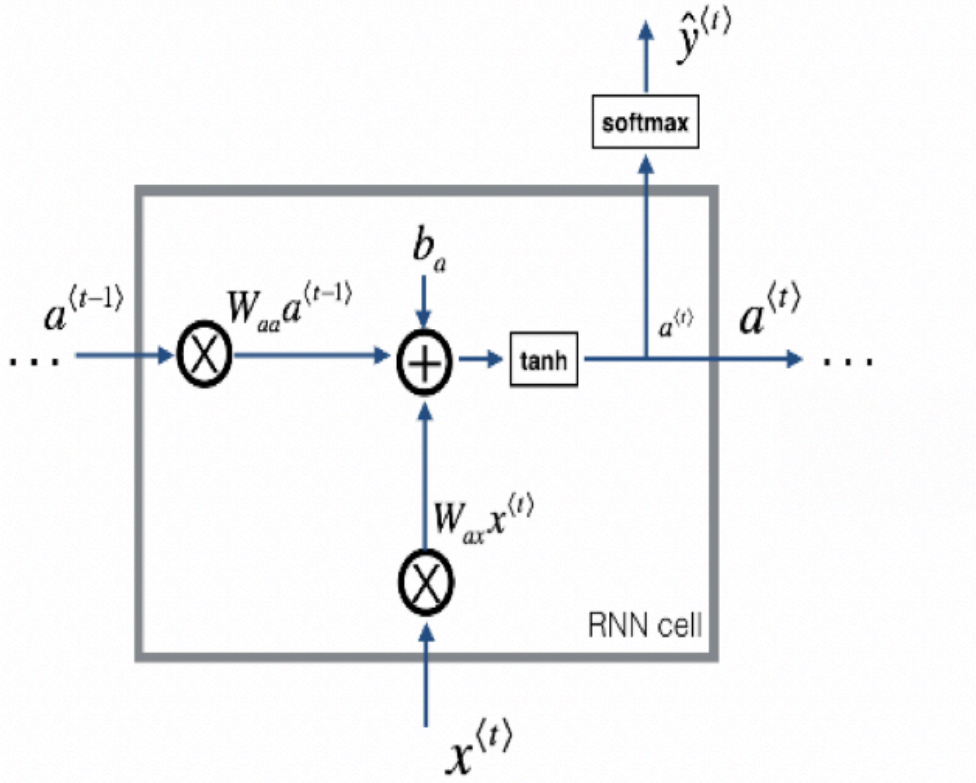
- x_t is the input at time step t. For example, x_1 could be a one-hot vector corresponding to the second word of a sentence.
- s_t is the hidden state at time step t. It's the "memory" of the network. s_t is calculated based on the previous hidden state and the input at the current step: $s_t = f(Ux_t + Ws_{t-1})$. The function f usually is a nonlinearity such as tanh or ReLU. s_{-1}, which is required to calculate the first hidden state, is typically initialized to all zeroes.
- o_t is the output at step t. For example, if we wanted to predict the next word in a sentence it would be a vector of probabilities across our vocabulary. $o_t = \text{softmax}(Vs_t)$.



$$s_t = f(Ux_t + Ws_{t-1}).$$

$$o_t = \text{softmax}(Vs_t).$$

Forward Computation in RNN



$$a^{\langle t \rangle} = \tanh(W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t-1 \rangle} + b_a)$$

$$\hat{y}^{\langle t \rangle} = soft\max(W_{ya}a^{\langle t \rangle} + b_y)$$

Example

https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/

https://medium.com/towards-artificial-intelligence/whirlwind-tour-of-rnns-a11effb7808f

1	0	0	0
0	1	0	0
0	0	1	1
0	0	0	0
h	e	l	l

$$s_t = f(Ux_t + Ws_{t-1}).$$

wxh			
0.287027	0.84606	0.572392	0.486813
0.902874	0.871522	0.691079	0.18998
0.537524	0.09224	0.558159	0.491528

×

1
0
0
0
h

=

0.287027
0.902874
0.537524

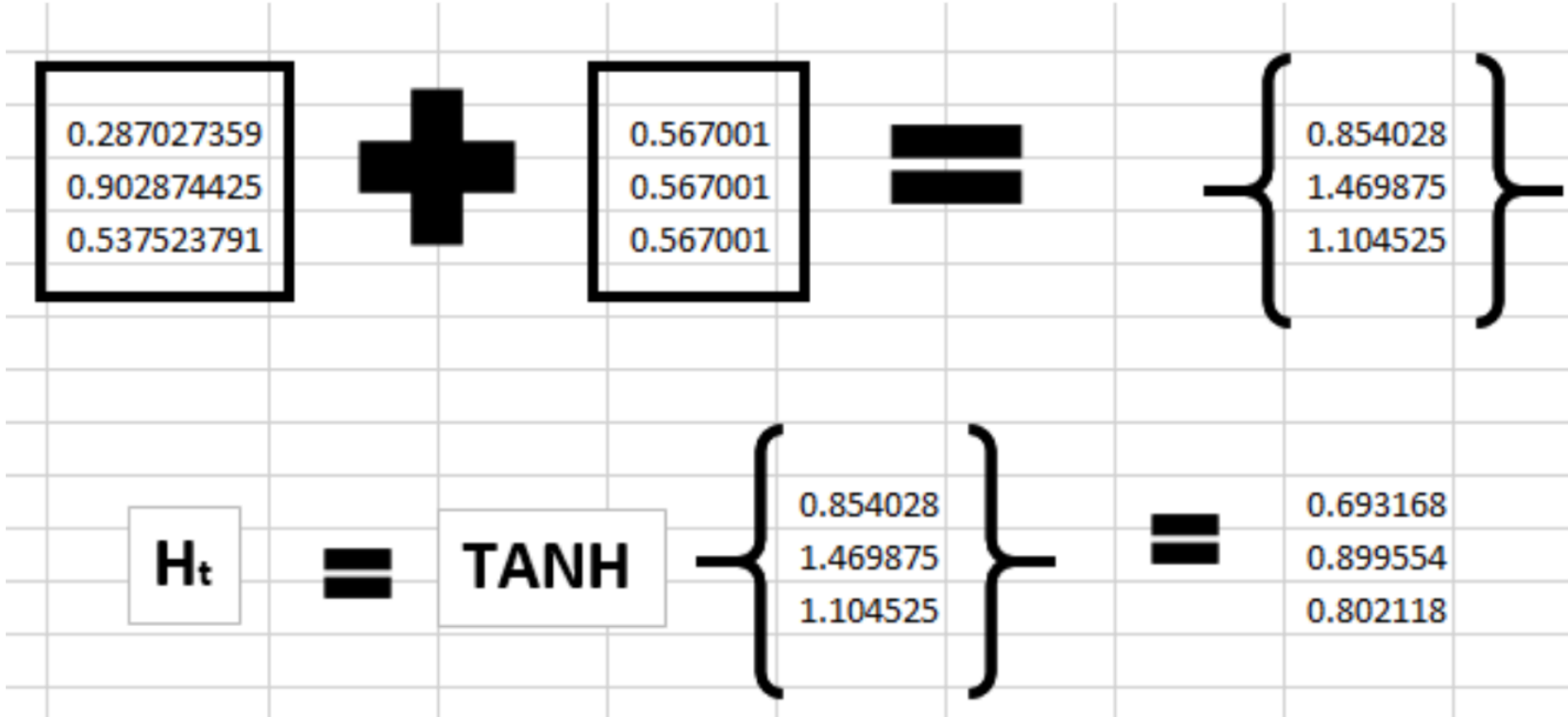
Weight(whh)	bias
0.427043	0.567001

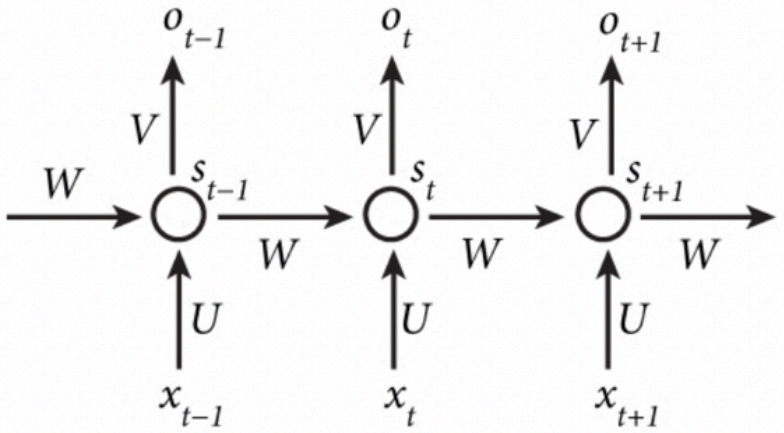
×

0
0
0
h_{t-1}

=

0.567001
0.567001
0.567001


0.287027359
0.902874425
0.537523791
0.567001
0.567001
0.567001
0.854028
1.469875
1.104525
Ht
TANH
0.854028
1.469875
1.104525
0.693168
0.899554
0.802118



$$s_t = f(Ux_t + Ws_{t-1}).$$

$$o_t = \text{softmax}(Vs_t).$$

$$a_1 = \begin{pmatrix} 0.1 & 0.5 & 0.1 \\ 0.5 & 0.9 & 0.3 \\ 0.3 & 0.2 & 0.1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0.6 & 0.8 & 0.4 & 0.8 \\ 0.2 & 0.2 & 0.8 & 0.7 \\ 0.9 & 0.8 & 0.1 & 0.2 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0.6 \\ 0.2 \\ 0.9 \end{pmatrix}$$

$$h_1 = tanh(\begin{pmatrix} 0.6 \\ 0.2 \\ 0.9 \end{pmatrix}) = \begin{pmatrix} 0.54 \\ 0.20 \\ 0.72 \end{pmatrix}$$

$$y_1 = softmax(\begin{pmatrix} 0.9 & 0.8 & 0.3 \\ 0.2 & 0.3 & 0.4 \\ 0.6 & 0.9 & 0.1 \\ 0.5 & 0.0 & 0.3 \end{pmatrix} \begin{pmatrix} 0.54 \\ 0.20 \\ 0.72 \end{pmatrix}) = \begin{pmatrix} 0.32 \\ 0.21 \\ 0.24 \\ 0.22 \end{pmatrix}$$

BackPropagation Through Time (BPTT)

The *loss*, or error, to be
the cross entropy loss, given by:

$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$

$$E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t)$$

$$= -\sum_t y_t \log \hat{y}_t$$

$$s_t = \tanh(Ux_t + Ws_{t-1})$$

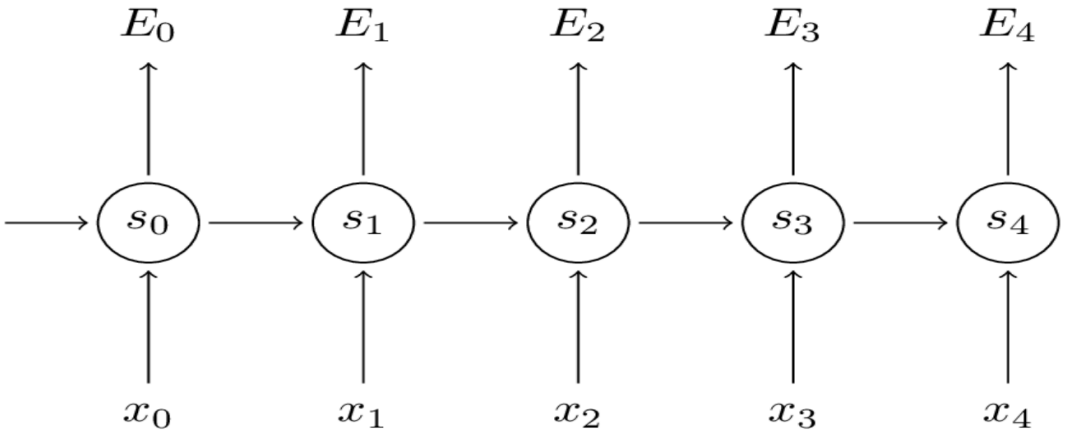
$$\hat{y}_t = \mathrm{softmax}(Vs_t)$$

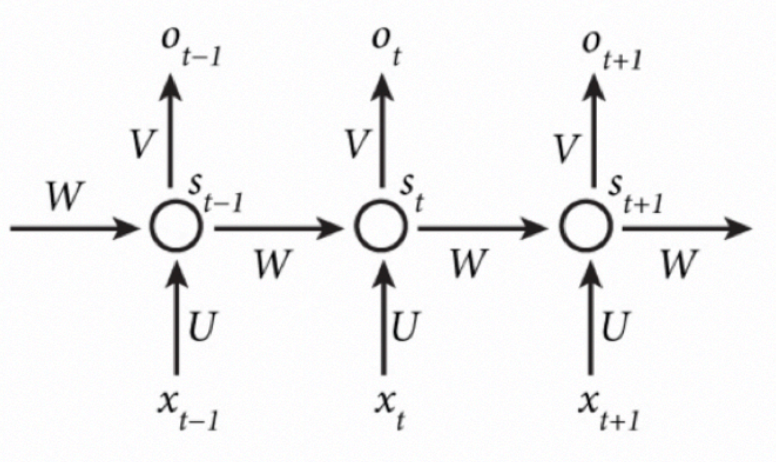
The cross entropy formula takes in two distributions, $p(x)$, the true distribution, and $q(x)$, the estimated distribution, defined over the discrete variable x and is given by

$$H(p, q) = -\sum_{\forall x} p(x) \log(q(x))$$

BackPropagation Through Time (BPTT)

Remember that our goal is to calculate the gradients of the error with respect to our parameters U, V and W and then learn good parameters using Stochastic Gradient Descent. Just like we sum up the errors, we also sum up the gradients at each time step for one training example: $\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$.



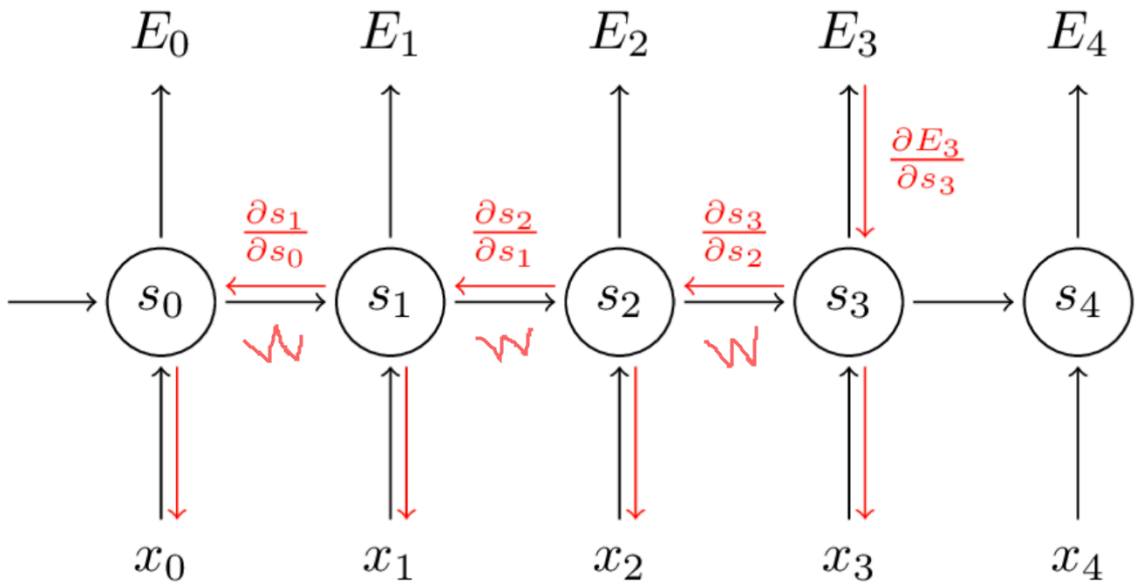


$$s_t = \tanh(Ux_t + Ws_{t-1})$$
$$\hat{y}_t = \text{softmax}(Vs_t)$$

$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$

$$E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t)$$

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W}$$



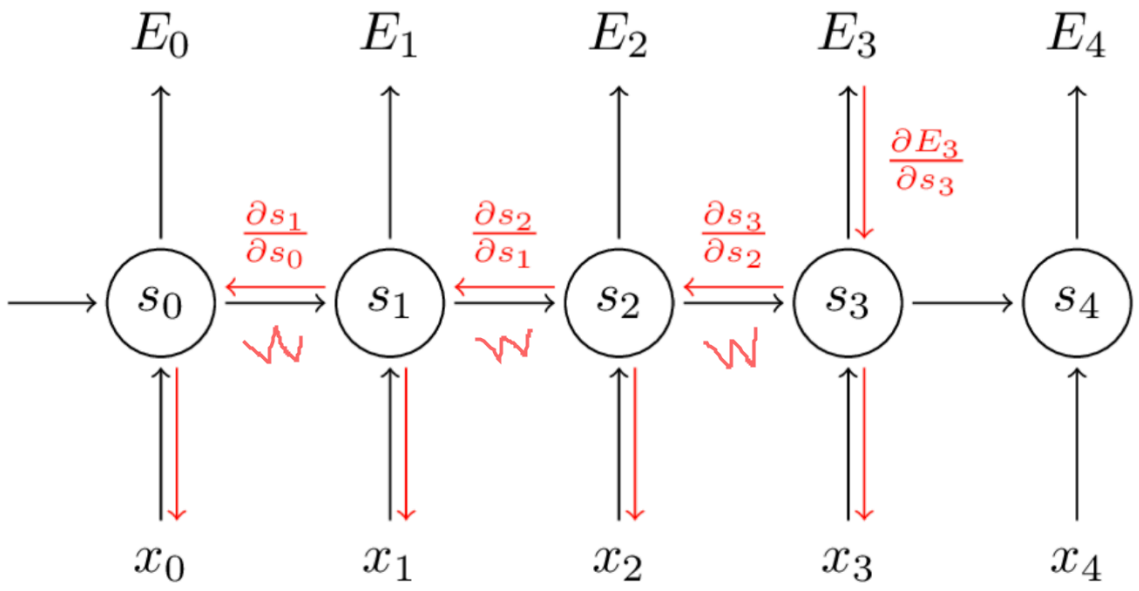
Note that $\frac{\partial s_3}{\partial s_k}$ is a chain rule in itself! For example,

$$\frac{\partial s_3}{\partial s_1} = \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1}.$$

$$s_t = \tanh(Ux_t + Ws_{t-1})$$
$$\hat{y}_t = \text{softmax}(Vs_t)$$

$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$

$$E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t)$$



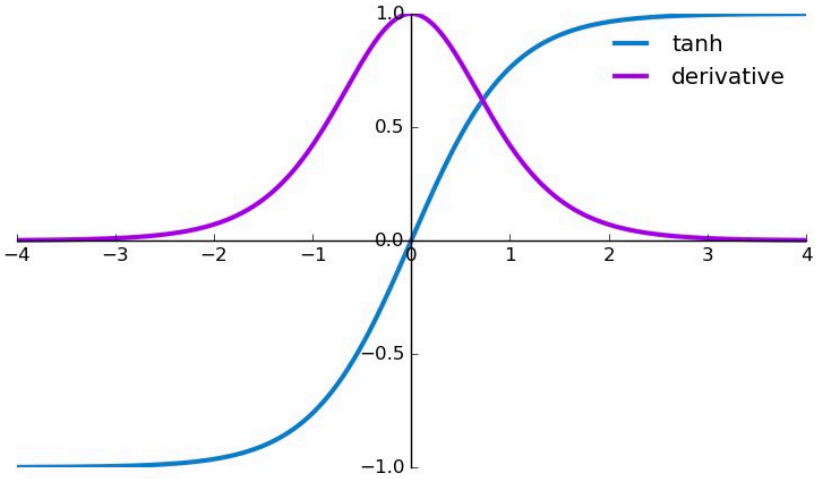
$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W}$$

$$\Rightarrow \quad \frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \left(\prod_{j=k+1}^{3} \frac{\partial s_j}{\partial s_{j-1}} \right) \frac{\partial s_k}{\partial W}$$

$$s_t = \tanh(Ux_t + Ws_{t-1})$$

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \left(\prod_{j=k+1}^{3} \frac{\partial s_j}{\partial s_{j-1}} \right) \frac{\partial s_k}{\partial W}$$

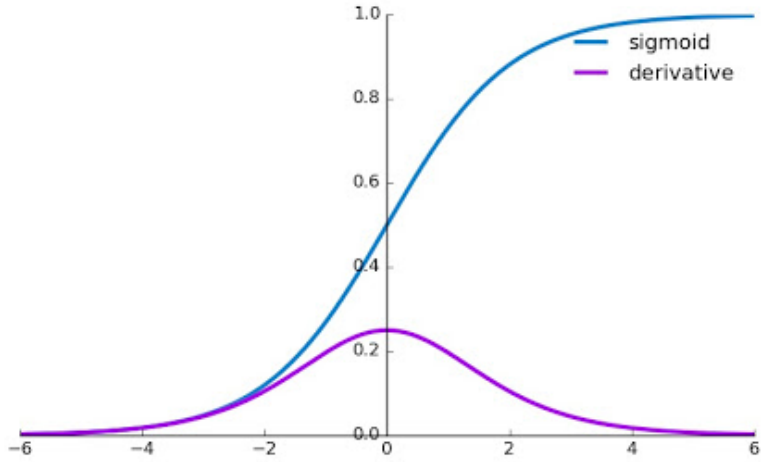
$$\frac{\partial s_j}{\partial s_{j-1}} = \tanh' * W$$

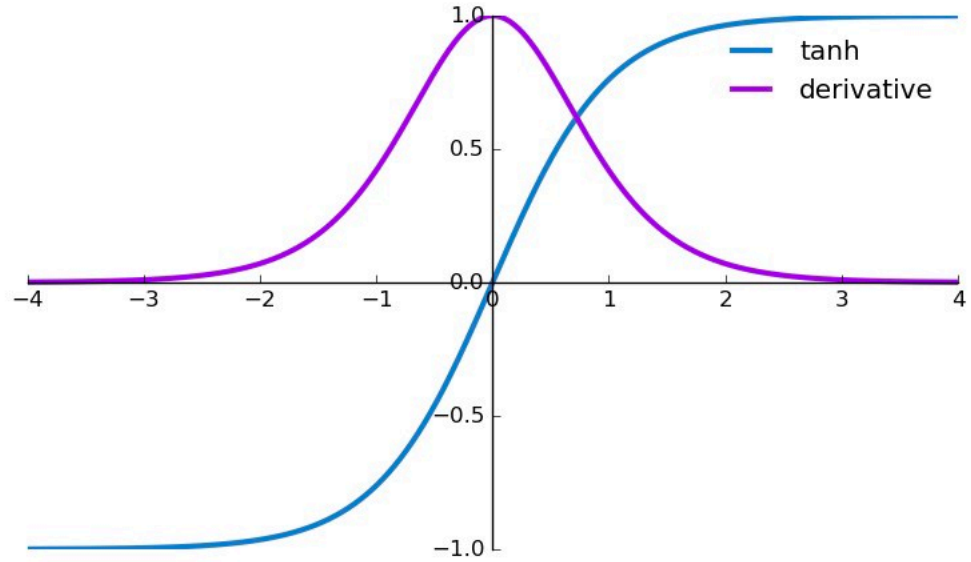


$$so \ \frac{\partial E_k}{\partial W} \longrightarrow 0$$

W ~ small -> Vanishing problem

Sigmoid and Tanh funtions





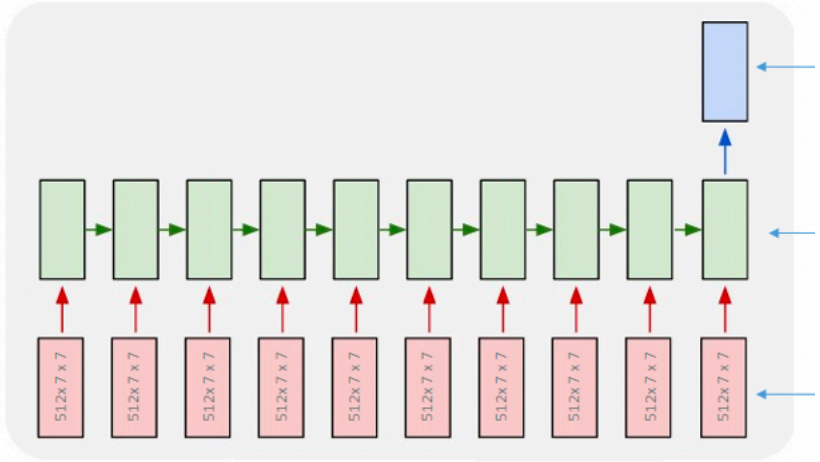
RNN: Vanishing Problem

"During back propagation, recurrent neural networks suffer from the vanishing gradient problem. Gradients are values used to update a neural networks weights.
The vanishing gradient problem is when the gradient shrinks as it back propagates through time. If a gradient value becomes extremely small, it doesn't contribute too much learning."

Limitation of RNN

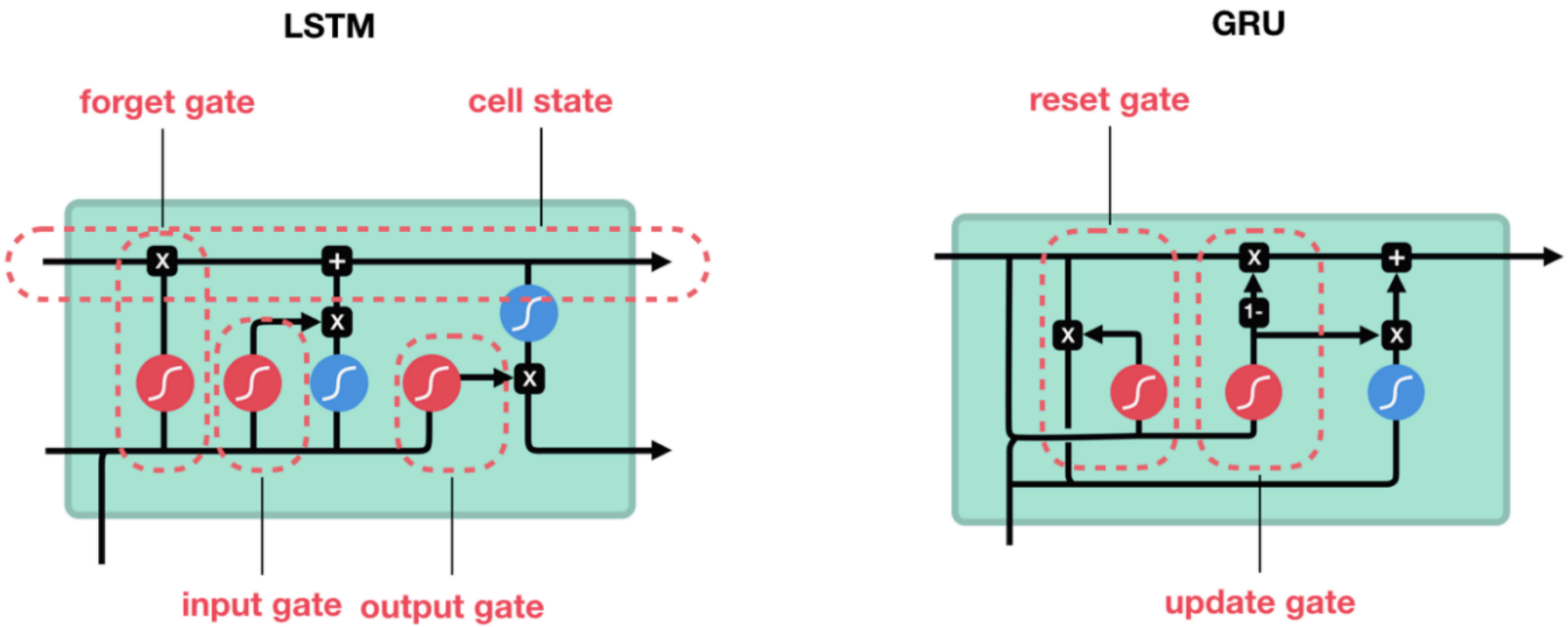
consider the sentence:

“ I grew up in France … I speak fluent French”



LSTM (Long Short Term Memory) ’s and **GRU (Gated Recurrent Unit)’s as a Solution**

These gates can learn which data in a sequence is important to keep or throw away



LSTM Networks

- 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), and were refined and popularized by many people, they work tremendously well on a large variety of problems, and are now widely used.
- LSTMs are explicitly designed to avoid the long-term dependency problem.

Customers Review 2,491

Thanos

September 2018

Verified Purchase

Amazing! This box of cereal gave me a perfectly balanced breakfast, as all things should be. I only ate half of it but will definitely be buying again!

When you read the review, your brain subconsciously only remembers important keywords. You pick up words like “amazing” and “perfectly balanced breakfast”. You don’t care much for words like “this”, “gave“, “all”, “should”, etc.

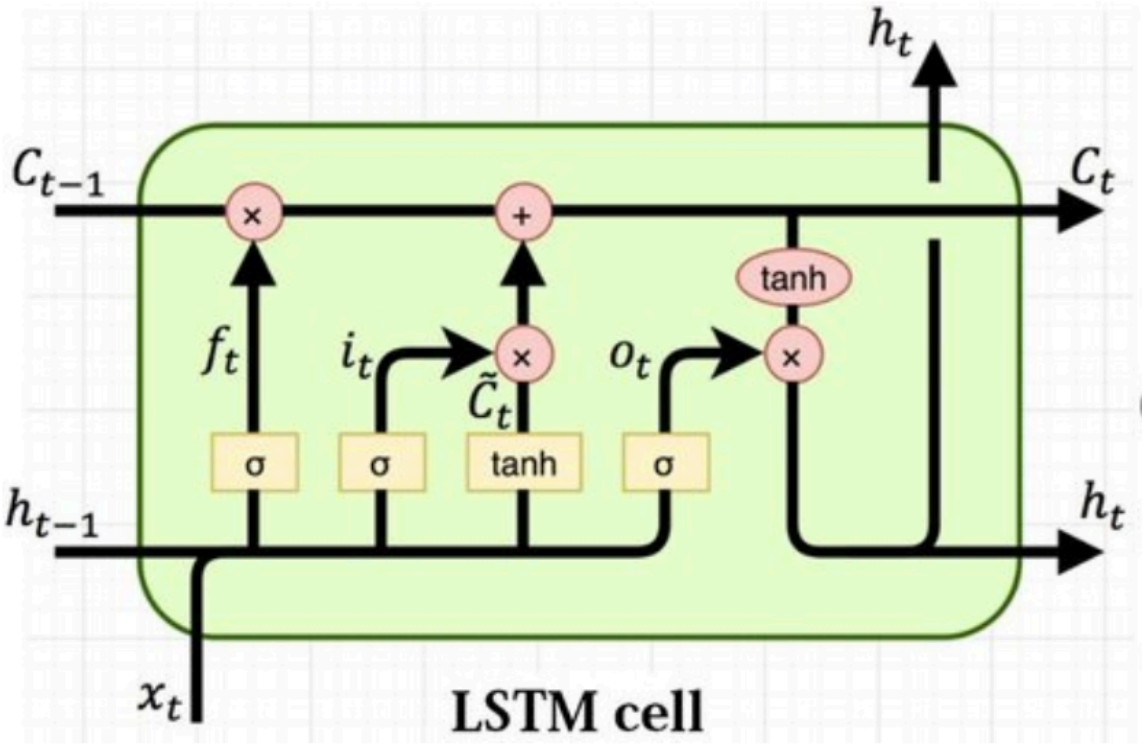
Longer Memories through LSTMs

- Adding a forgetting mechanism.
- Adding a saving mechanism.
  - When the model sees a new image, it needs to learn whether any information about the image is worth using and saving.

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

https://dominhhai.github.io/vi/2017/10/what-is-lstm/

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
\tilde{c}_t &= \sigma_h(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

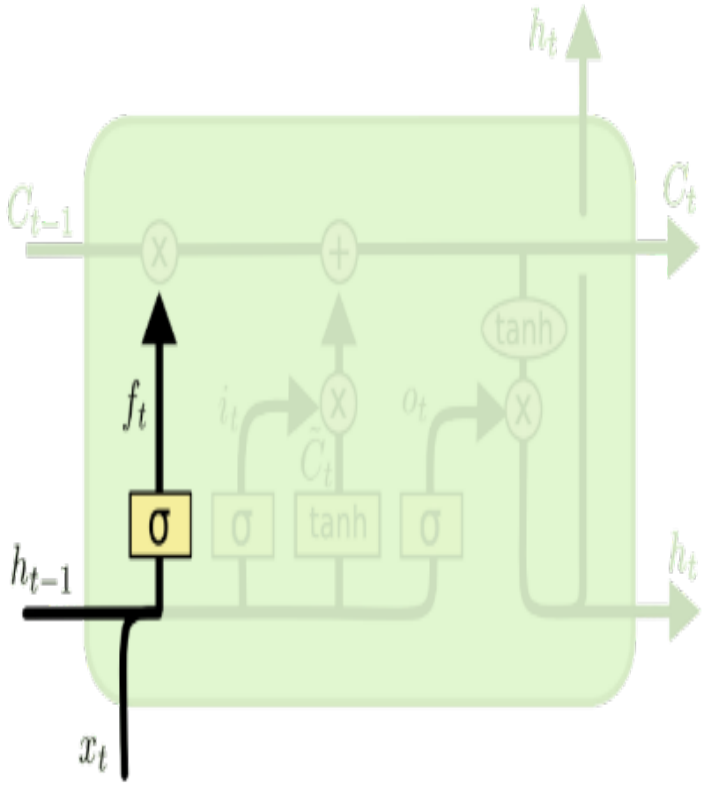


where the initial values are $c_0 = 0$ and $h_0 = 0$ and the operator $\circ$ denotes the Hadamard product (element-wise product). The subscript t indexes the time step.

https://en.wikipedia.org/wiki/Long_short-term_memory

LSTM Cell

- Forget gate

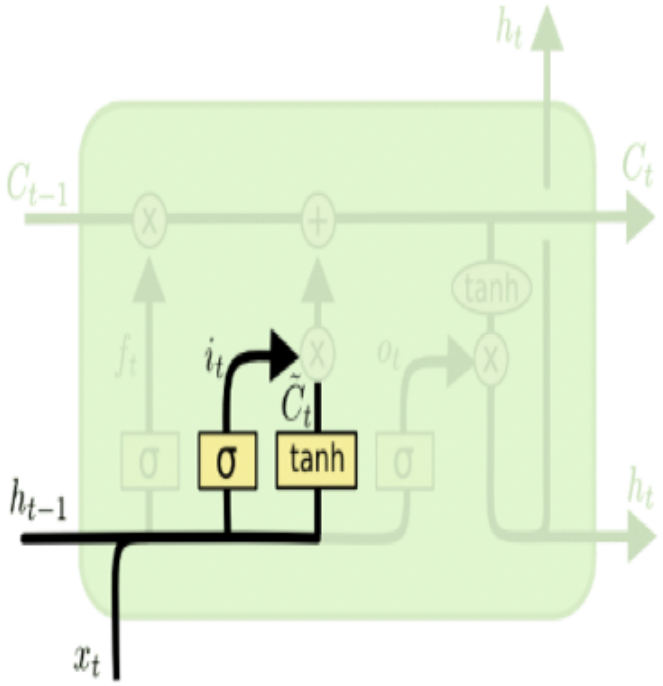


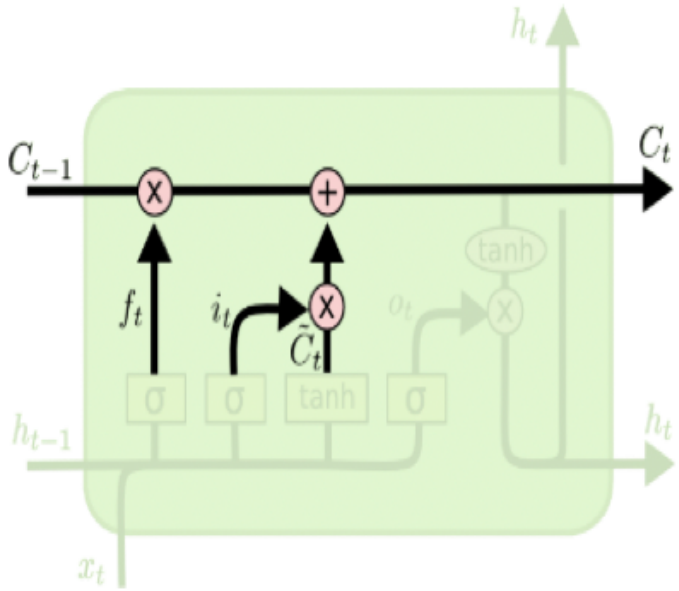
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

- Input Gate

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

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





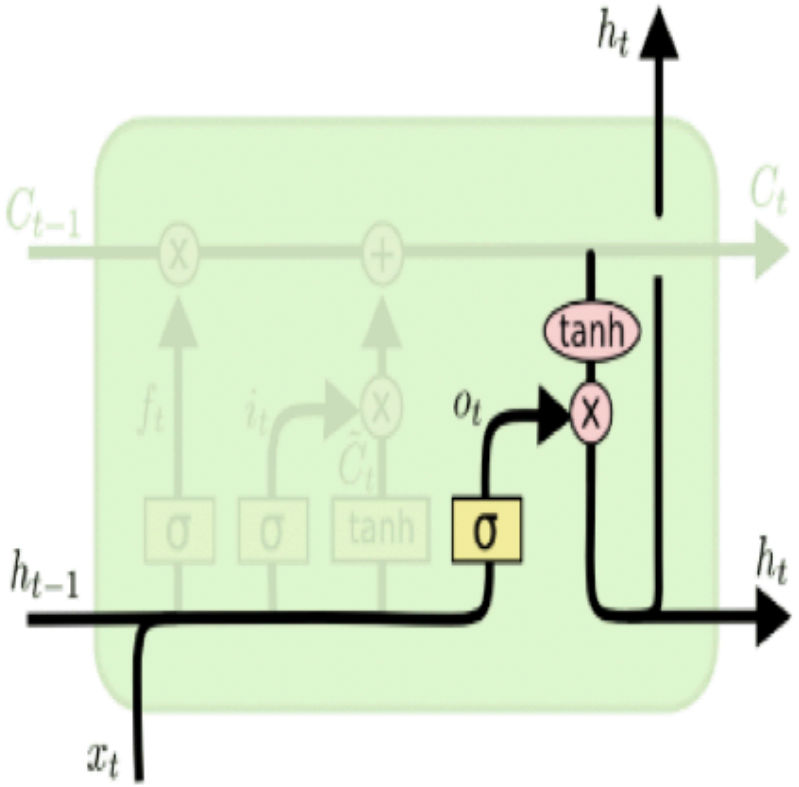
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Output Gate

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

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

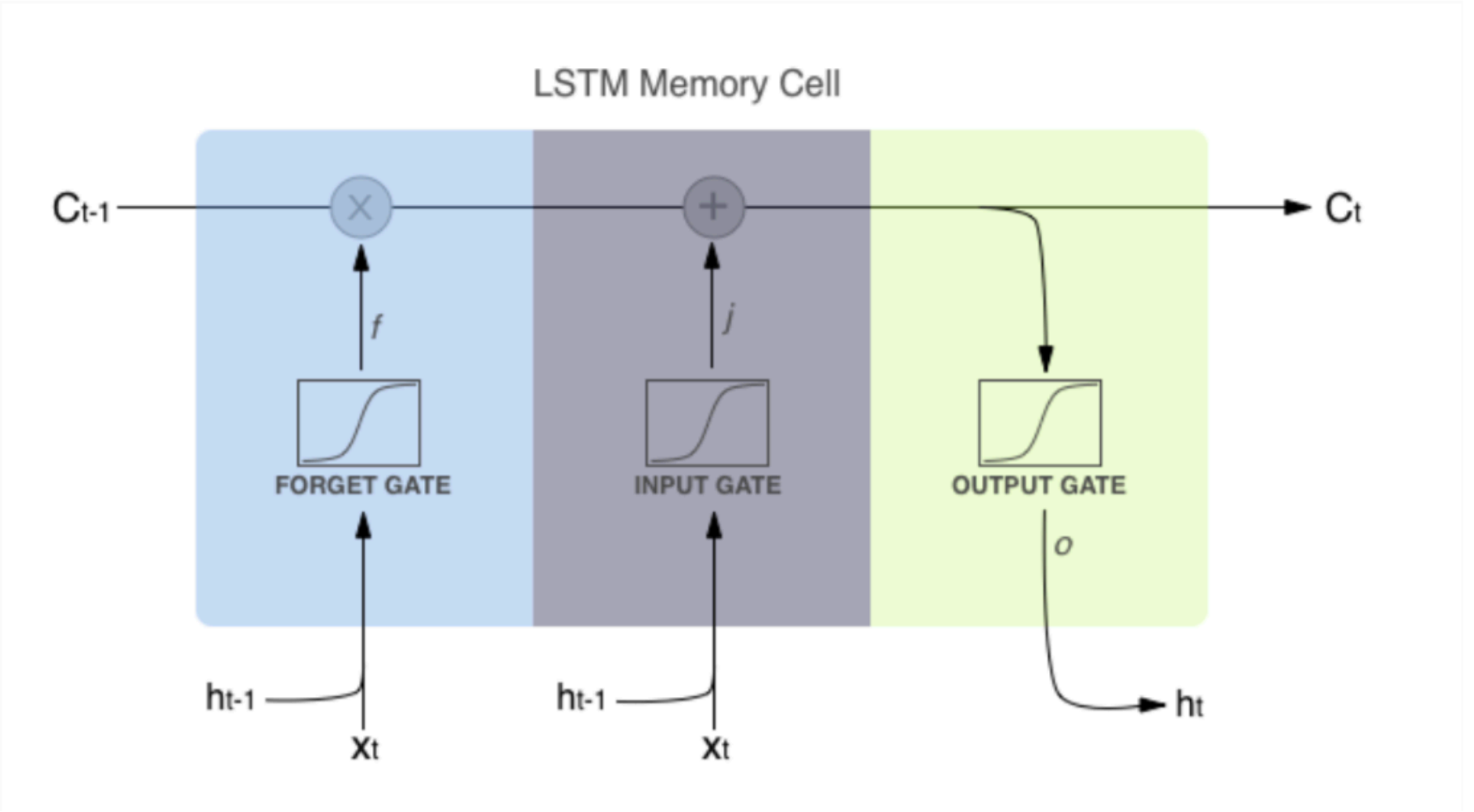
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



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

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

LSTM Memory Cell



How does LSTM prevent Vanishing

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

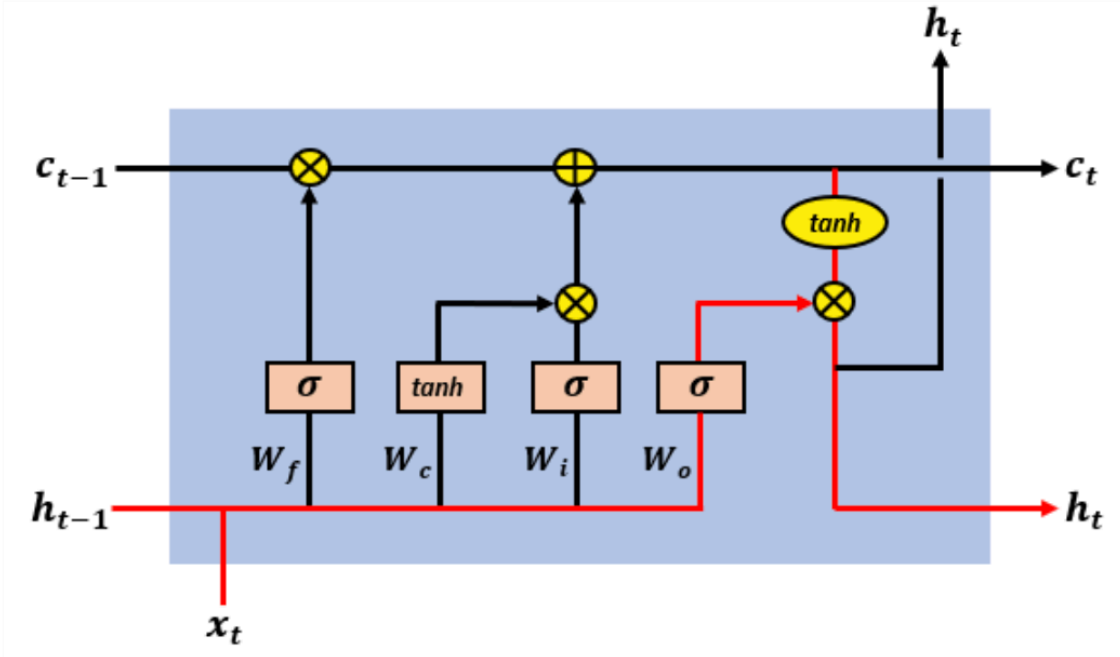
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t])$$

$$\tilde{c}_t = tanh\ (W_c \cdot [h_{t-1}, x_t])$$

$$c_t = c_{t-1} \otimes f_t \oplus \tilde{c}_t \otimes i_t$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t])$$

$$h_t = o_t \otimes tanh(c_t)$$



$$tanh(W_c \cdot [h_{t-1}, x_t]) \otimes \sigma(W_i \cdot [h_{t-1}, x_t])$$

BackPropagation Through Time in LSTM

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W} \qquad (3)$$

The gradient of the error in an LSTM

$$\frac{\partial E_k}{\partial W} = \frac{\partial E_k}{\partial h_k} \frac{\partial h_k}{\partial c_k} \dots \frac{\partial c_2}{\partial c_1} \frac{\partial c_1}{\partial W}$$

$$= \frac{\partial E_k}{\partial h_k} \frac{\partial h_k}{\partial c_k} \left(\prod_{t=2}^{k} \frac{\partial c_t}{\partial c_{t-1}} \right) \frac{\partial c_1}{\partial W} \qquad (4)$$

Backpropagation through time in LSTM

In an LTSM, the state vector c(t), has the form:

$$c_t = c_{t-1} \otimes \sigma(W_f \cdot [h_{t-1}, x_t]) \oplus$$

$$tanh\,(W_c \cdot [h_{t-1}, x_t]) \otimes \sigma(W_i \cdot [h_{t-1}, x_t])$$

which can be written compactly as

$$c_t = c_{t-1} \otimes f_t \oplus \tilde{c}_t \otimes i_t \qquad (5)$$

$$c_t = c_{t-1} \otimes f_t \oplus \tilde{c}_t \otimes i_t$$

vanilla RNN

$$C_t = tanh(W_c C_{t-1} + W_x X_t)$$

Compute the derivative of (5) and get:

$$\frac{\partial c_t}{\partial c_{t-1}} = \frac{\partial}{\partial c_{t-1}} [c_{t-1} \otimes f_t \oplus \tilde{c}_t \otimes i_t]$$

$$= \frac{\partial}{\partial c_{t-1}} [c_{t-1} \otimes f_t] + \frac{\partial}{\partial c_{t-1}} [\tilde{c}_t \otimes i_t]$$

$$= \frac{\partial f_t}{\partial c_{t-1}} \cdot c_{t-1} + \frac{\partial c_{t-1}}{\partial c_{t-1}} \cdot f_t + \frac{\partial i_t}{\partial c_{t-1}} \cdot \tilde{c}_t + \frac{\partial \tilde{c}_t}{\partial c_{t-1}} \cdot i_t$$

Backpropagating through time for gradient computation

$$\frac{\partial C_t}{\partial C_{t-1}} \approx \sigma(W_f \cdot [H_{t-1}, X_t]) \qquad (2)$$

Now, notice equation (2) means that **the gradient behaves similarly to the forget gate**, and if the forget gate decides that a certain piece of information should be remembered, it will be open and have values closer to 1 to allow for information flow.
For simplicity, we can think of the forget gate's action as:

$$\sigma(W_f \cdot [H_{t-1}, X_t]) \approx \vec{1}$$

So we get:

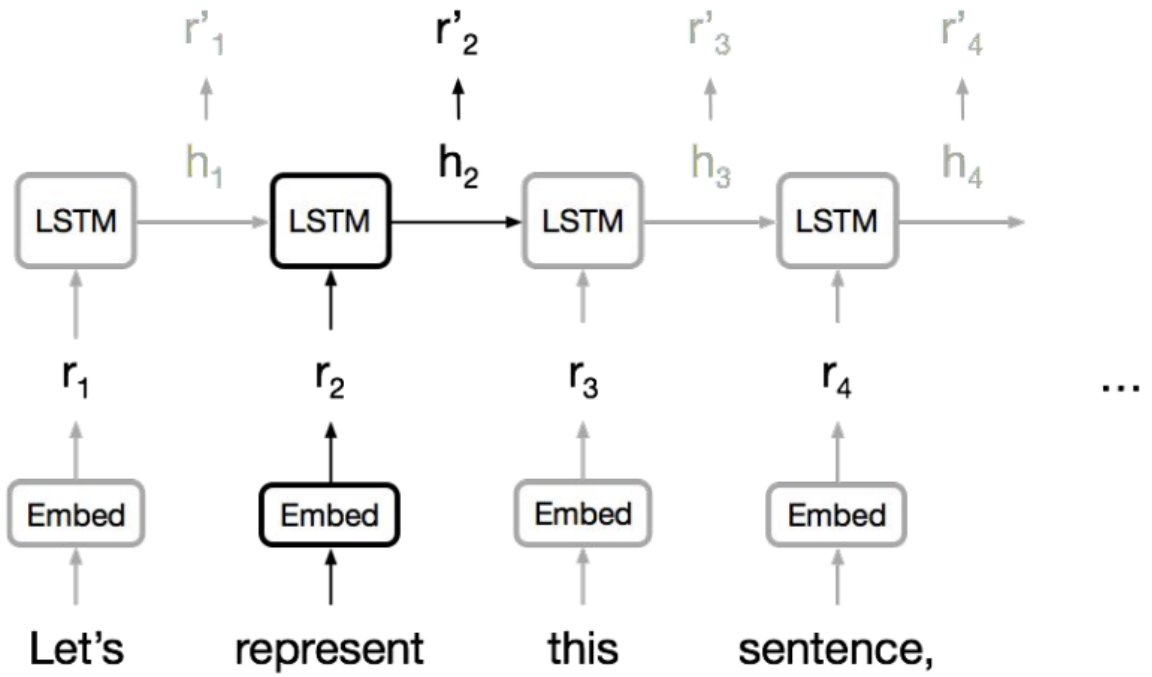
$$\frac{\partial C_t}{\partial C_{t-1}} \nrightarrow 0$$

Finally:

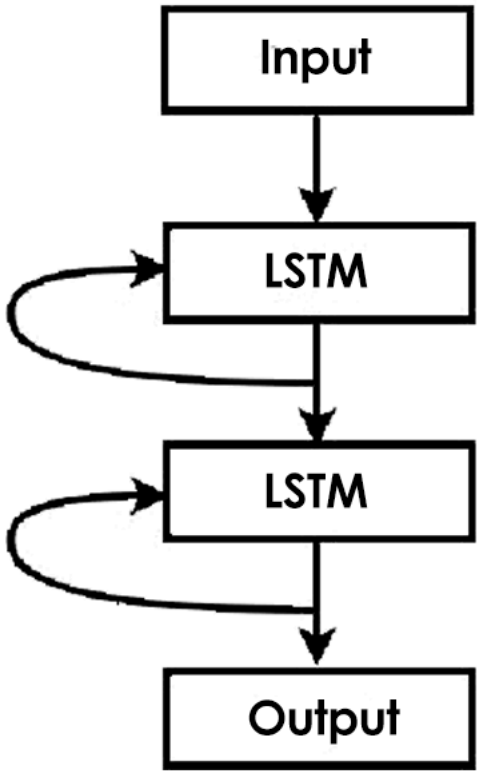
$$\frac{\partial E_k}{\partial W} \approx \frac{\partial E_k}{\partial H_k}\frac{\partial H_k}{\partial C_k}\left(\prod_{t=2}^{k} \sigma(W_f \cdot [H_{t-1}, X_t])\right)\frac{\partial C_1}{\partial W} \nrightarrow 0$$

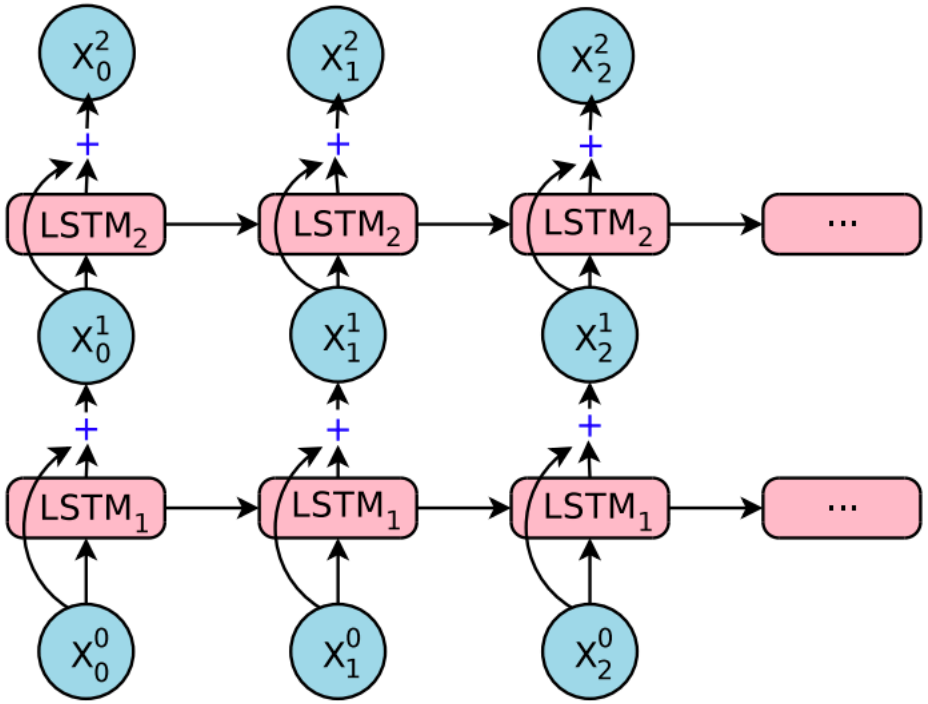
And the gradients do not vanish!

LSTM Architecture

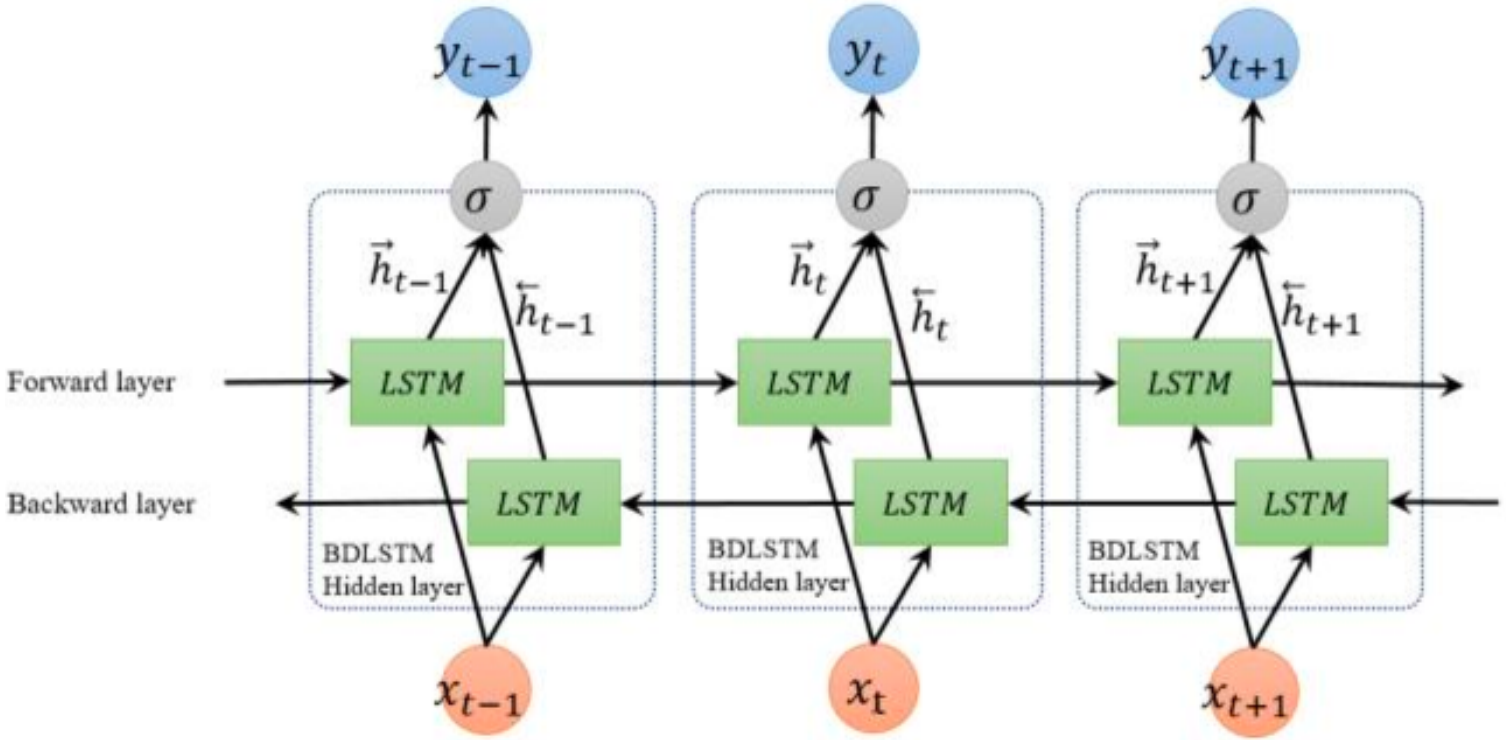


Stacked LSTM

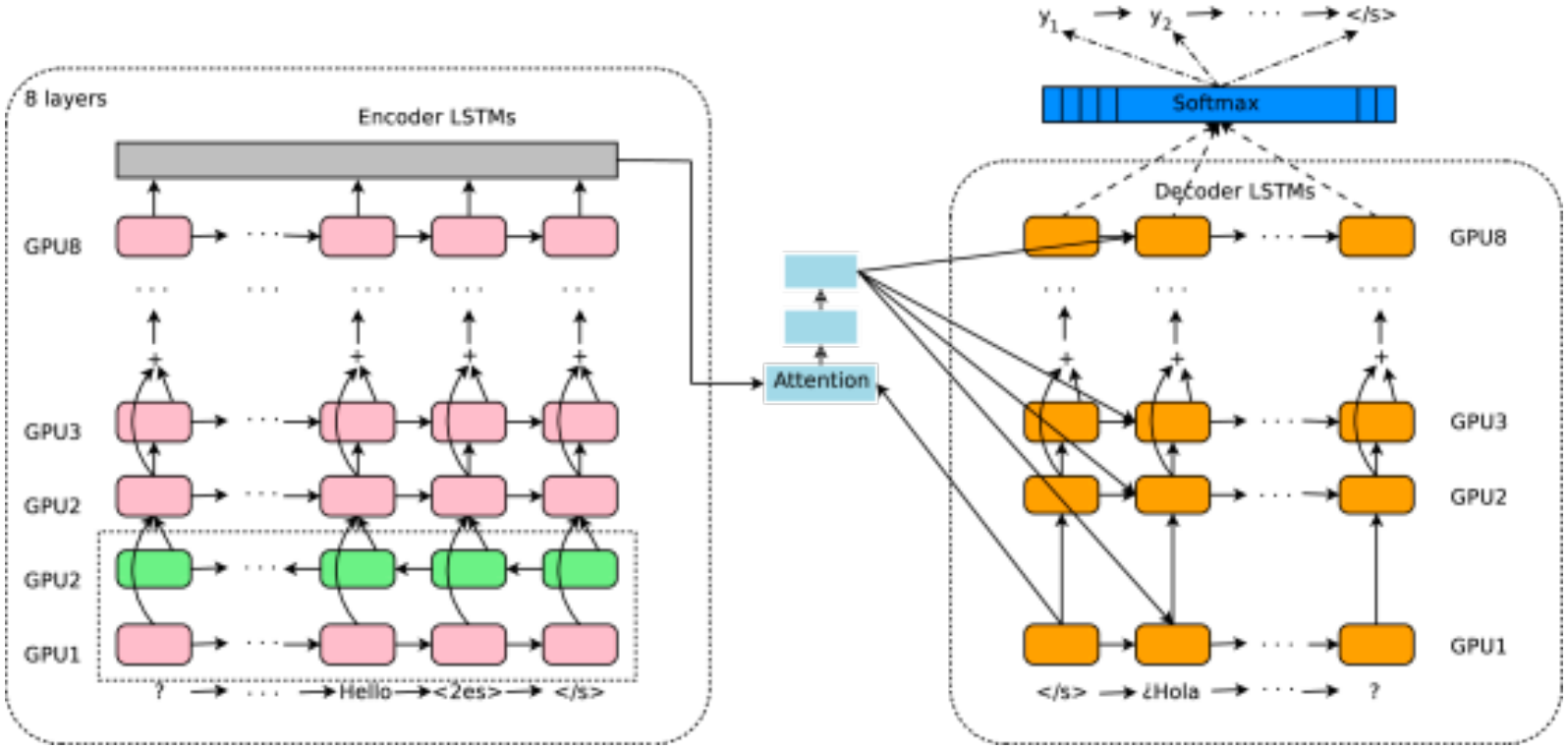




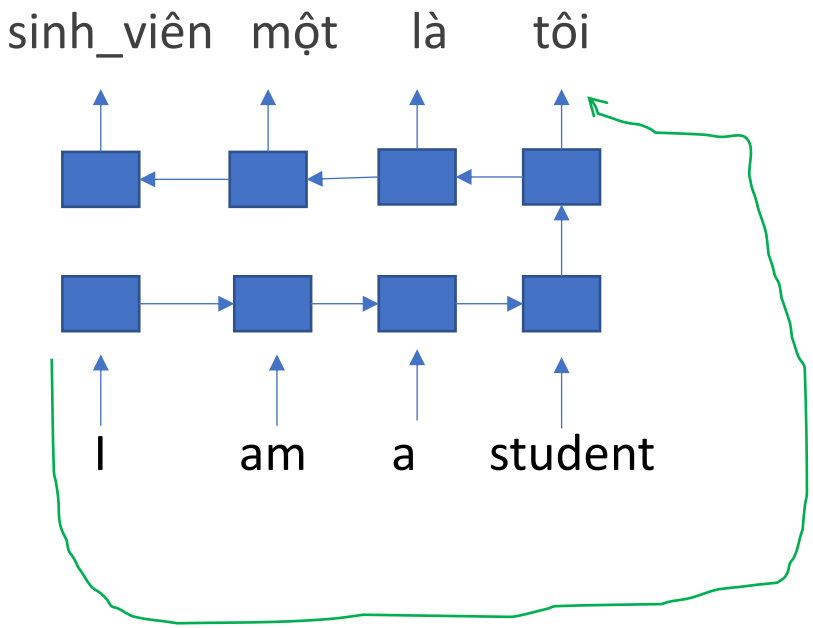
BiLSTM

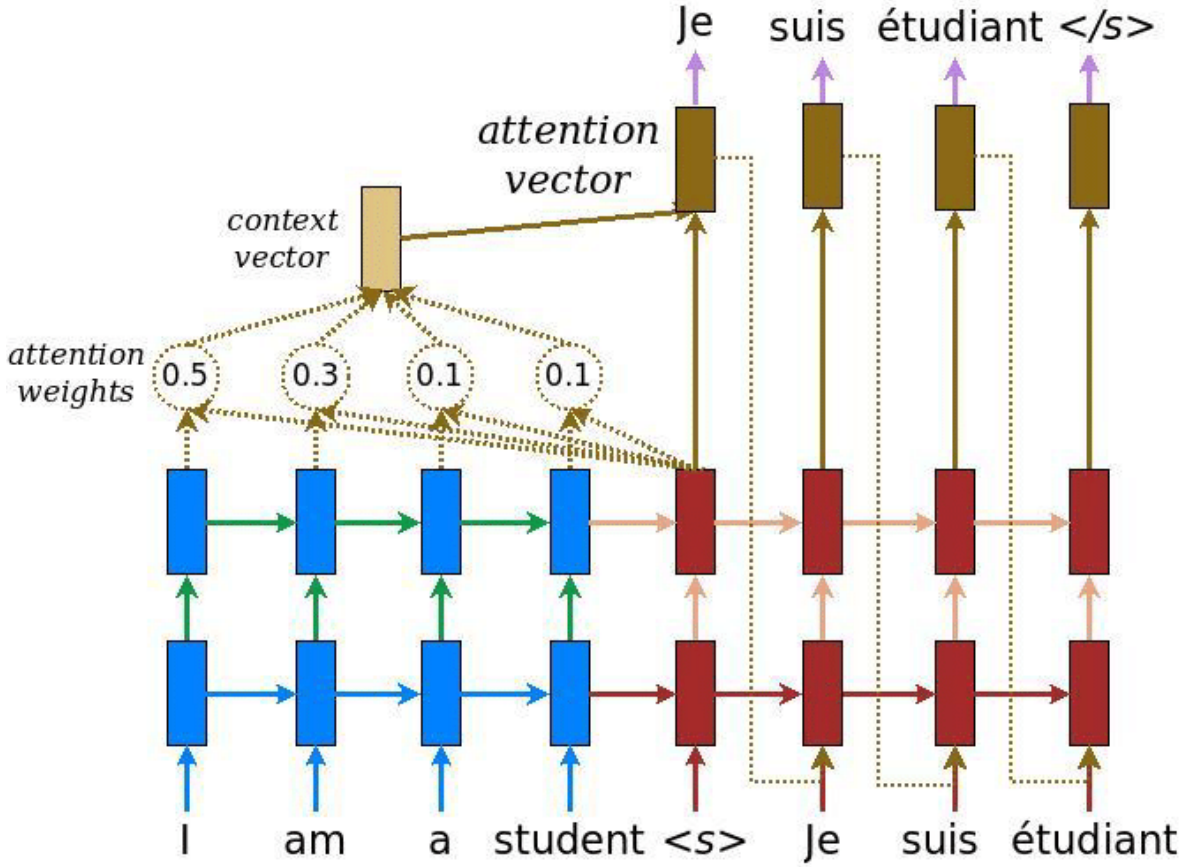

y_{t-1}
y_t
y_{t+1}
σ
$\vec{h}_{t-1}$
$\overleftarrow{h}_{t-1}$
$\vec{h}_t$
$\overleftarrow{h}_t$
$\vec{h}_{t+1}$
$\overleftarrow{h}_{t+1}$
Forward layer
Backward layer
LSTM
BDLSTM
Hidden layer
x_{t-1}
x_t
x_{t+1}

Neural Machine Translation



Attention based NMT





THANK YOU!