HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Lesson 9: Recurrent neural network

Viet-Trung Tran

Outline

- Sequence prediction
- Recurrent neural networks (RNN)
- Back-propagation thought time (BPTT)
- LSTM and GRU
- RNN applications

Sequence prediction

Sequence prediction

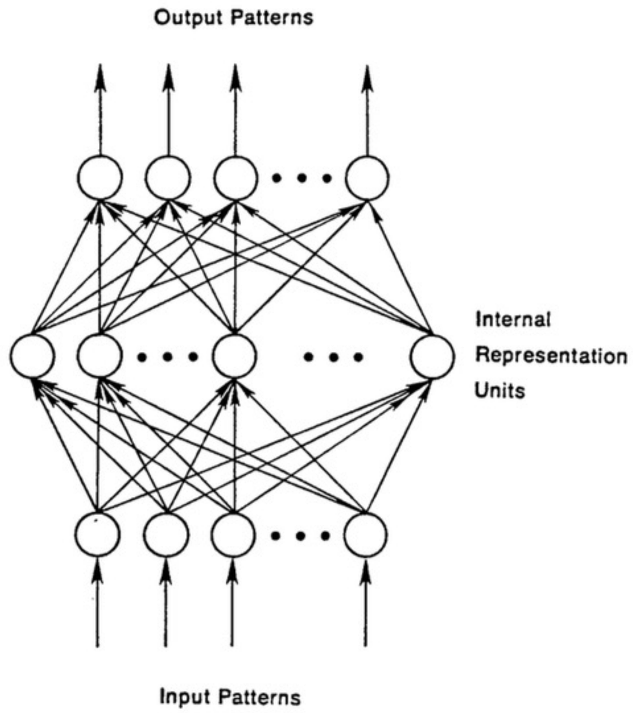
- So far, we have only focused on the prediction problem with fixed-size inputs and outputs
- What if the input and output are a variable sized sequence?

Text classification

- Sentiment classification: categorize reviews of (a restaurant or a movie or a product) as positive or negative
  - “The food was really good”
  - “The vacuum cleaner broke within two weeks”
  - “The movie has its dull parts, but overall, it is worth watching”
- What features and classification models should be used to solve this problem?
  - Inputs are variable sized sequences

Breft on feedforward neural networks

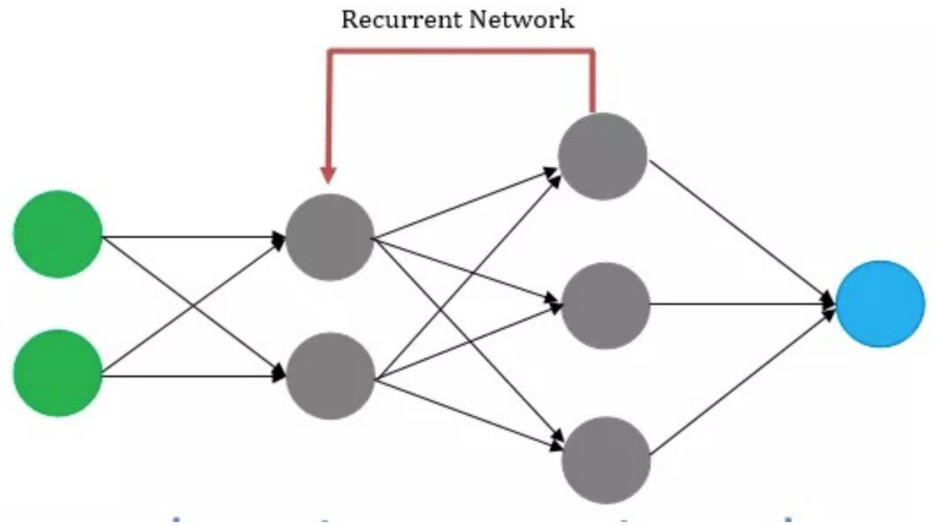
- Output relies only on current input
- Input -> hidden -> output
- Input as fixed-sized vector
- Output as fixed-sized vector
- Fixed amount of computation steps
- No memories
  - No notation of order in time
  - Totally forget the past



© https://skymind.ai/wiki/lstm

Recurrent neural network (RNN)

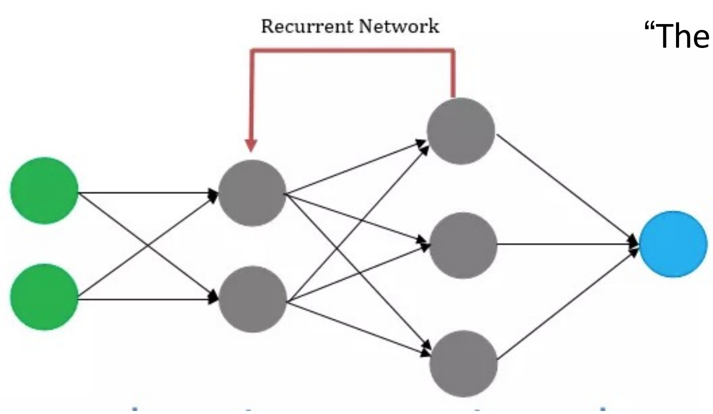
- The connections between units may form a directed cycle



© learning.maxtech4u.com

Network with “memory”

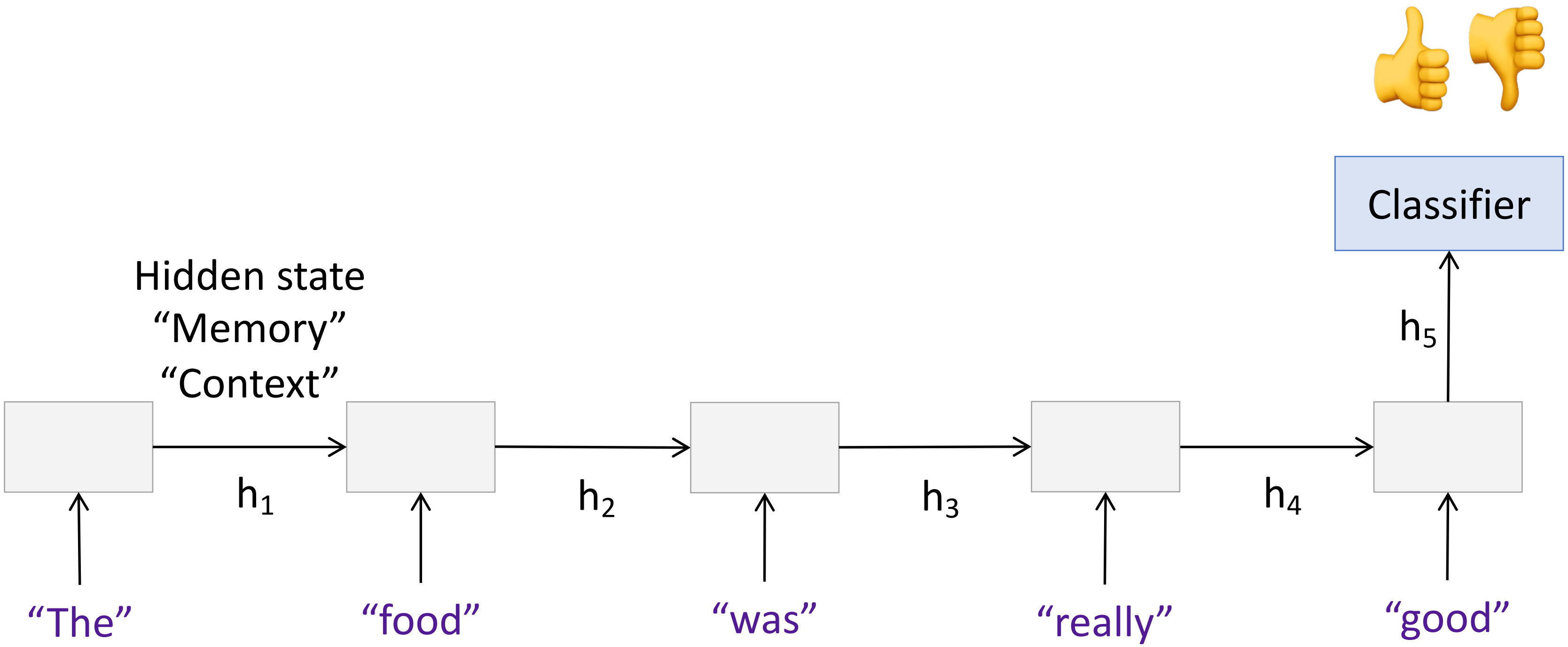
- Output relies on current input and the recent past (history information)
- Input + prev_hidden -> hidden -> output



“The past that does not pass away”

Sentiment classification

- “The food was really good”



Recurrent Neural Network (RNN)

Language model

$$P_{bo}(w_i \mid w_{i-n+1} \cdots w_{i-1})$$

RNN language model

- Character RNN

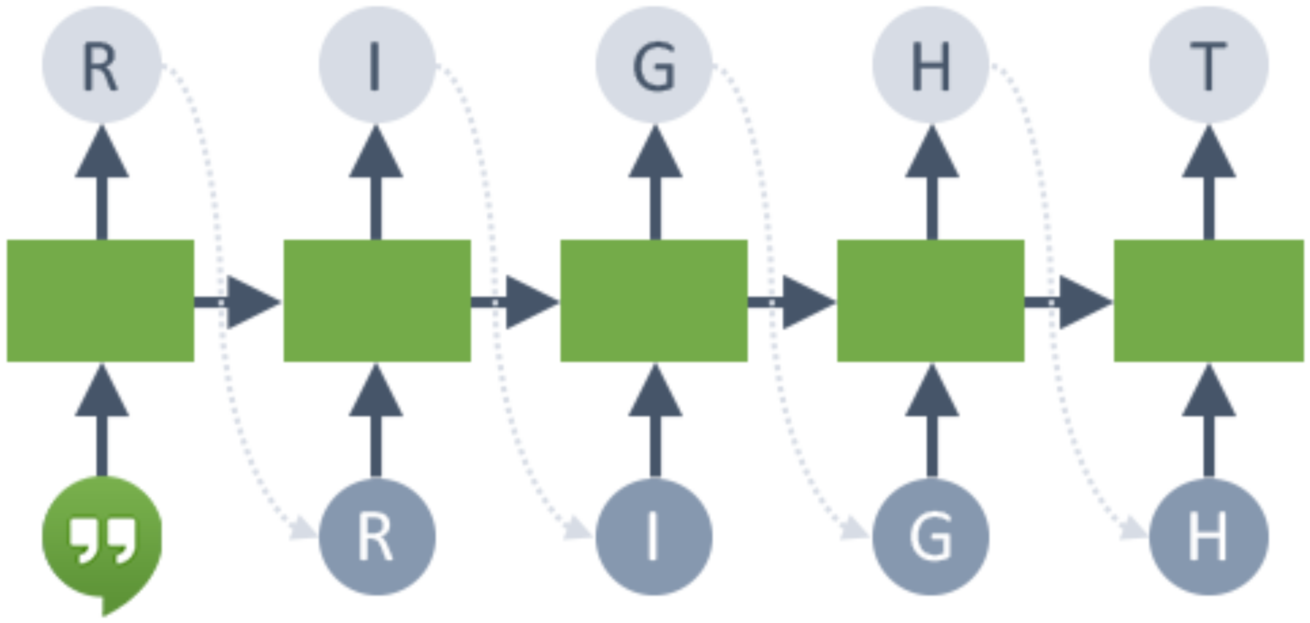


Image source

Character RNN

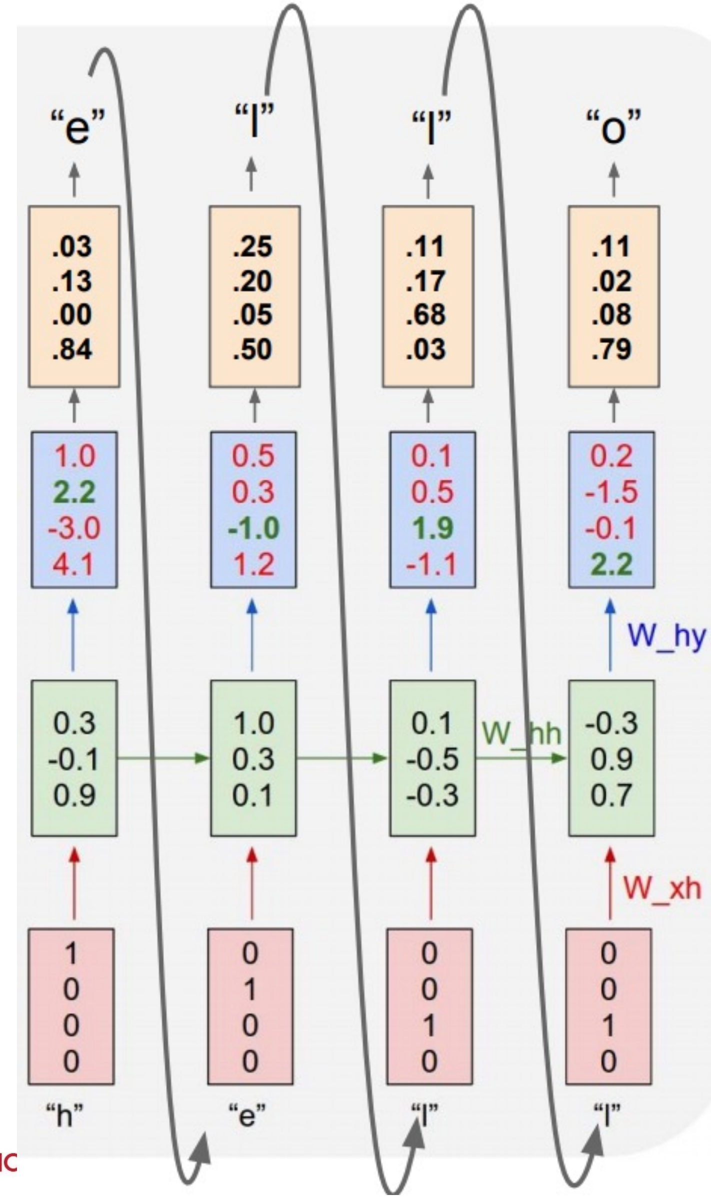
Output character y_i

Output layer (linear transformation + softmax)

Hidden state h_i

One-hot encoding x_i

Input characters



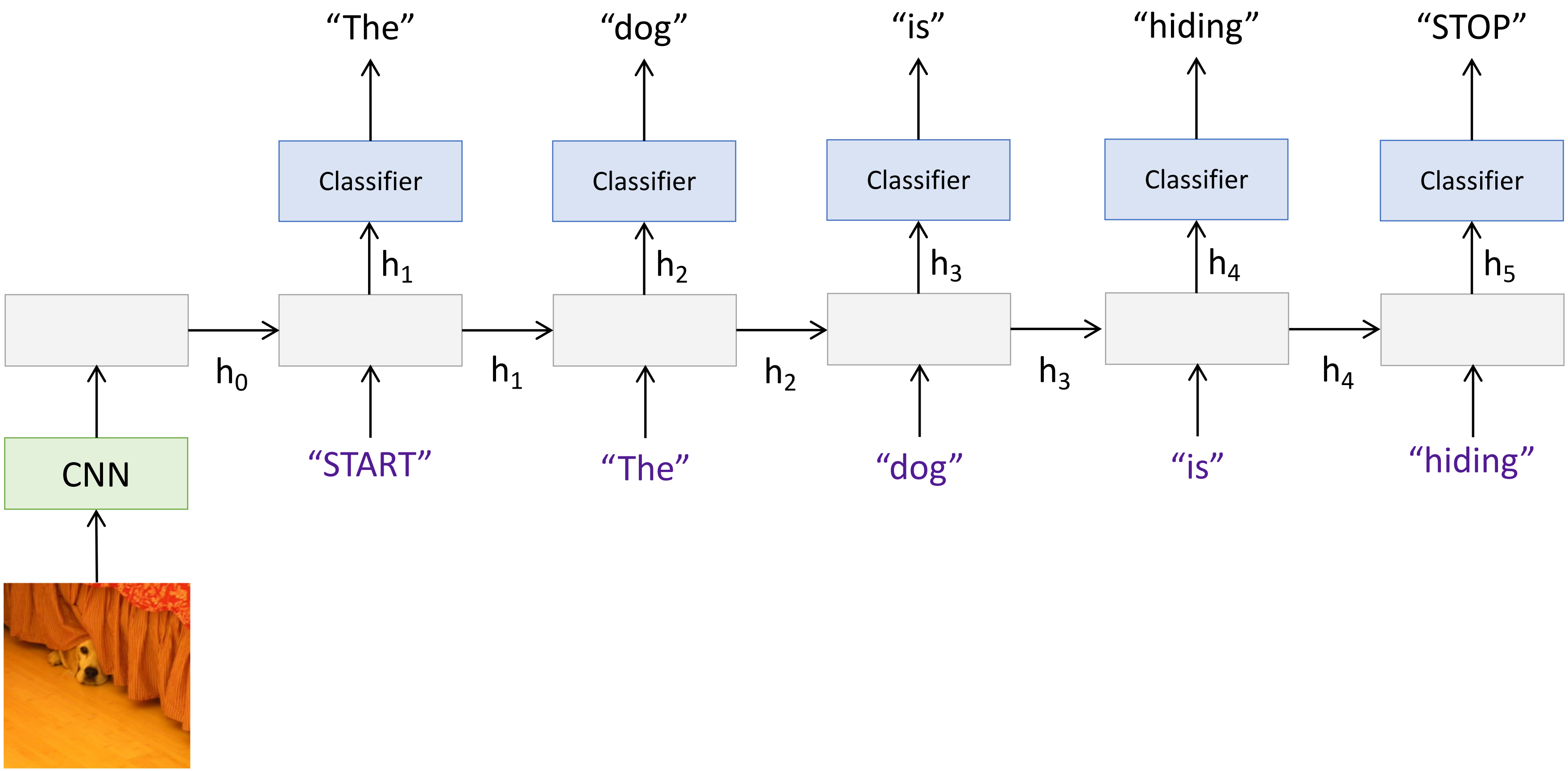
$$
\begin{aligned}
& p(y_1, y_2, \dots, y_n) \\
&= \prod_{i=1}^{n} p(y_i | y_1, \dots, y_{i-1}) \\
&\approx \prod_{i=1}^{n} P_W(y_i | h_i)
\end{aligned}
$$

Generation of photo captions

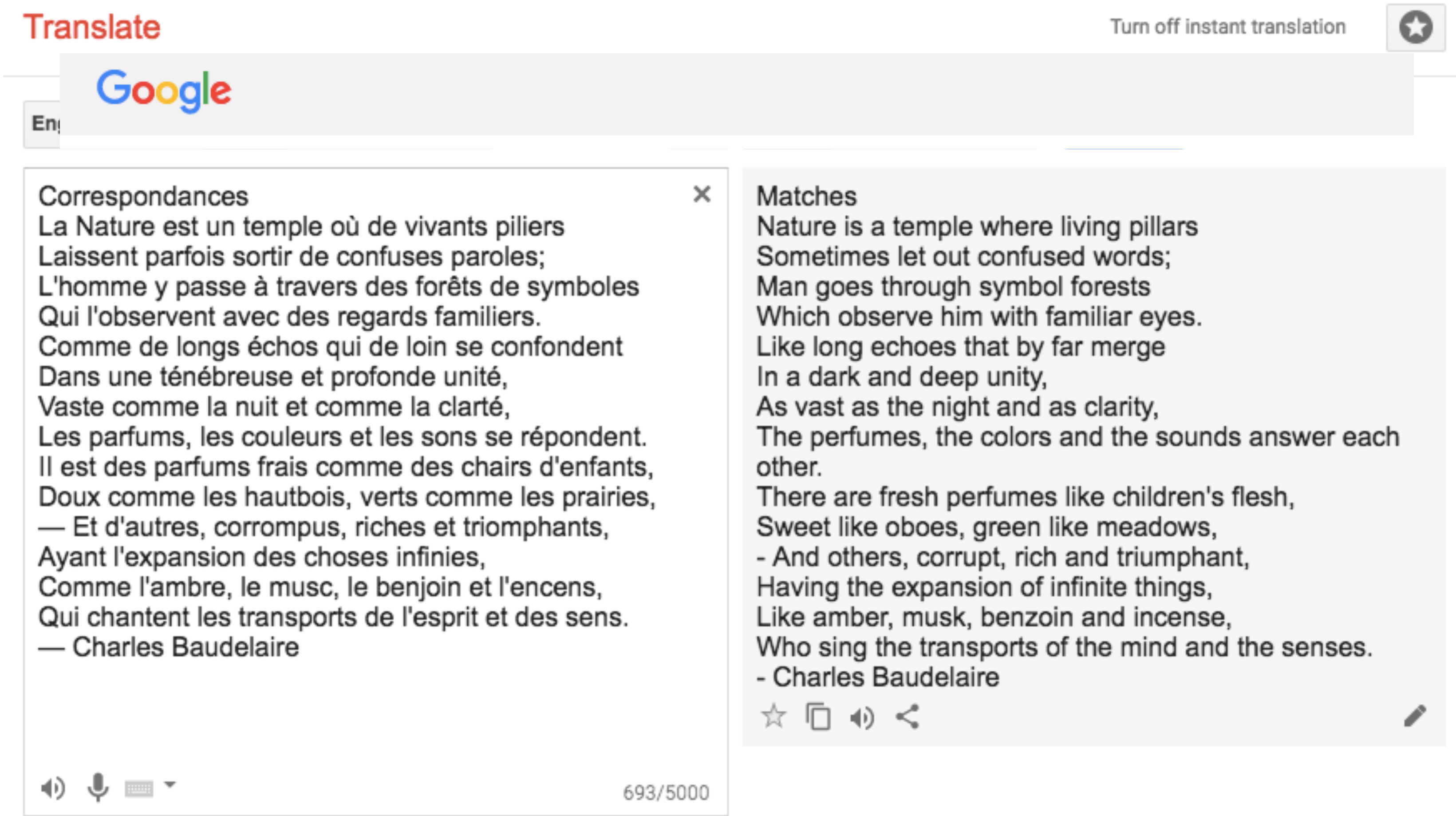
- Given an image, we must generate a sentence describing the content of the image

"The dog is hiding"

Generation of photo captions (2)



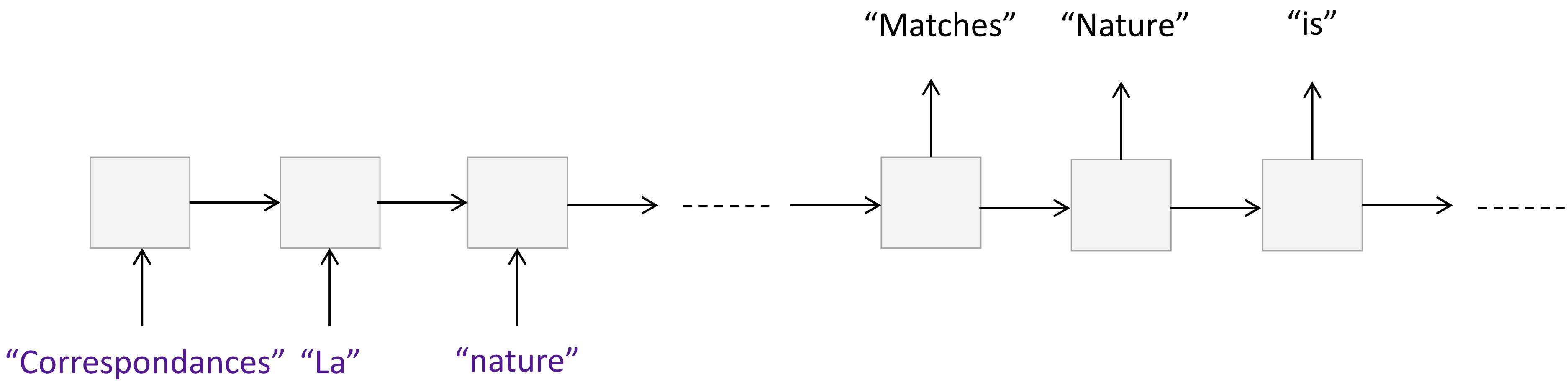
Machine translation



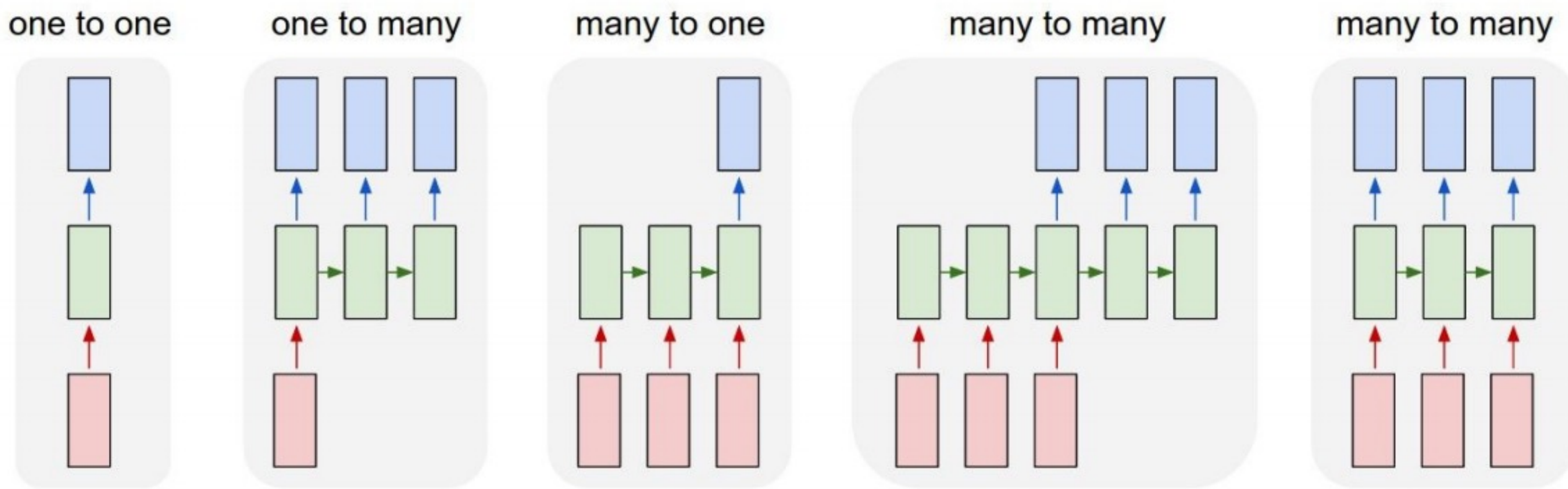
https://translate.google.com/

Machine translation (2)

- Sequence to sequence



Summary of prediction types



Phân lớp ảnh

Sinh mô tả ảnh

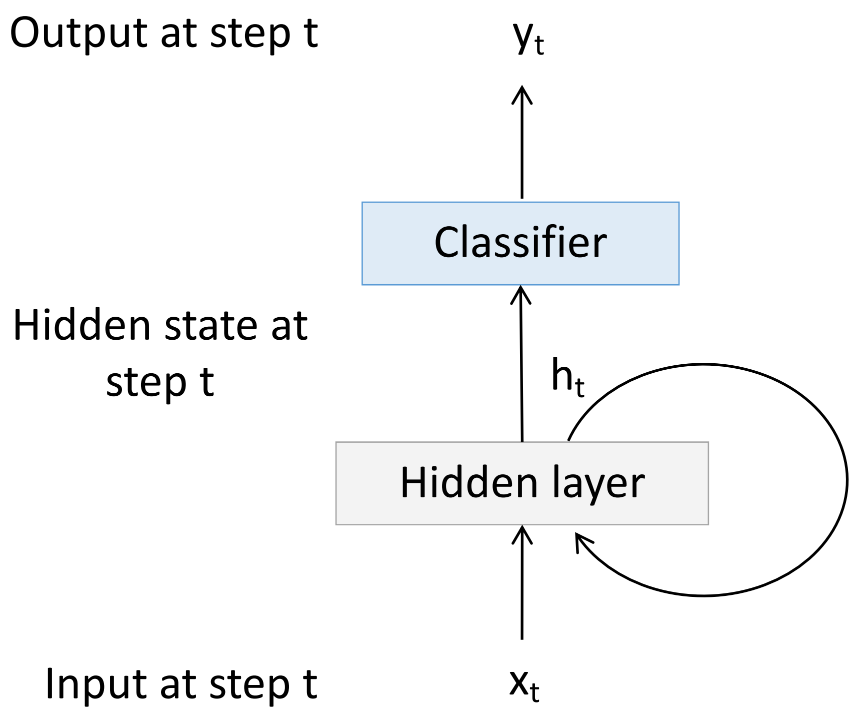
Phân loại sắc thái câu

Dịch máy

Phân loại video mức frame

Recurrent Neural Network (RNN)

Recurrent Neural Network (RNN)

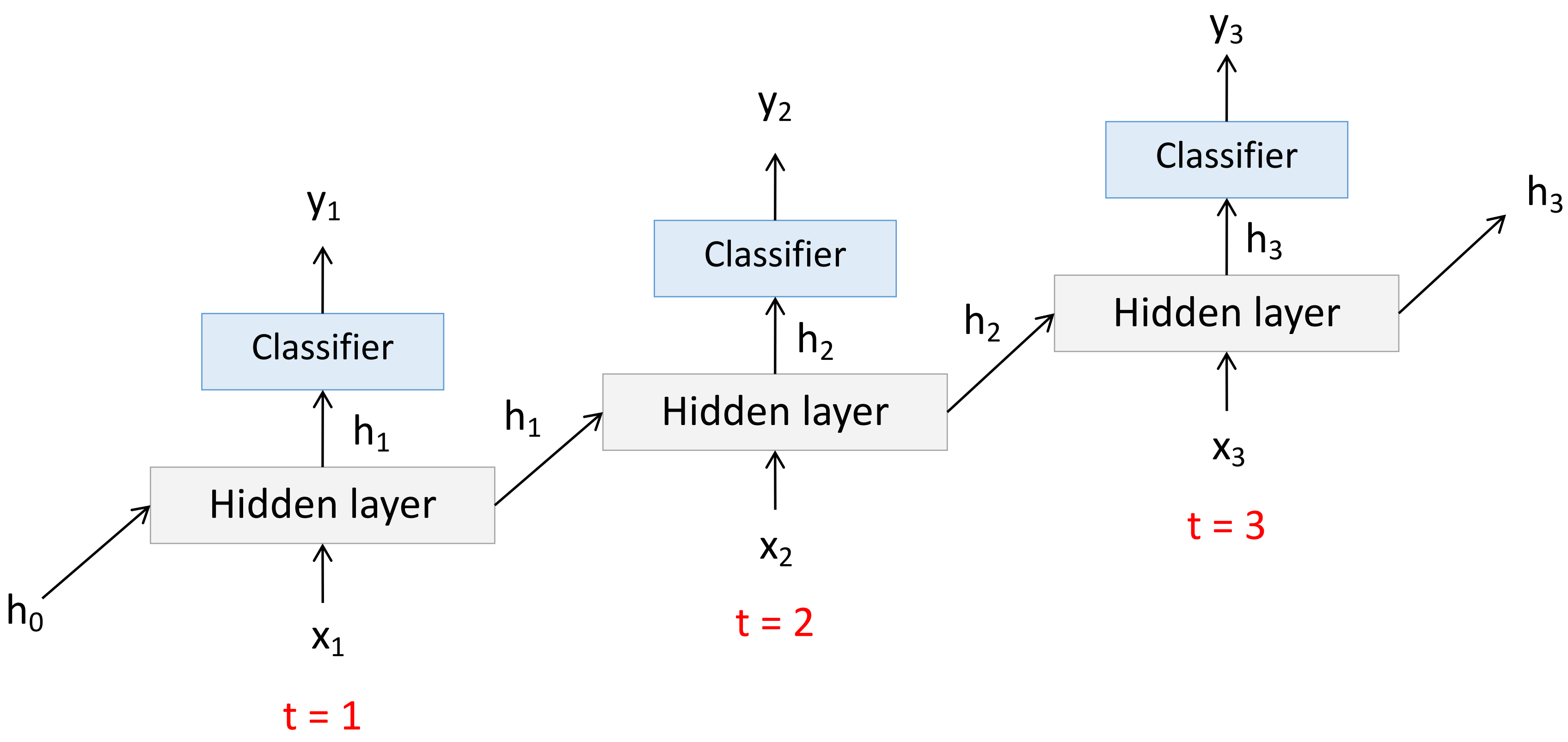


Hồi quy:

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

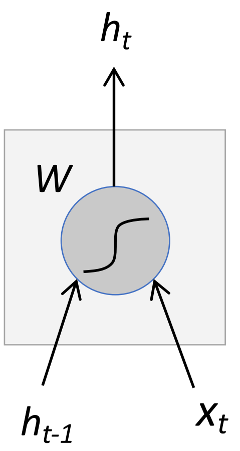
new state | function of W | input at time t | old state

Unroll RNN



y_1
Classifier
h_1
Hidden layer
h_0
x_1
$t = 1$
h_1
y_2
Classifier
h_2
Hidden layer
x_2
$t = 2$
h_2
y_3
Classifier
h_3
Hidden layer
x_3
$t = 3$
h_3

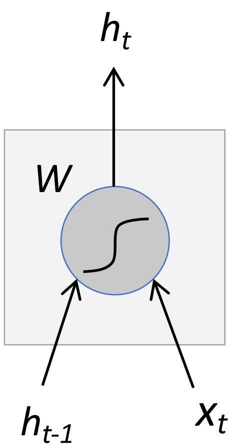
Vanilla RNN



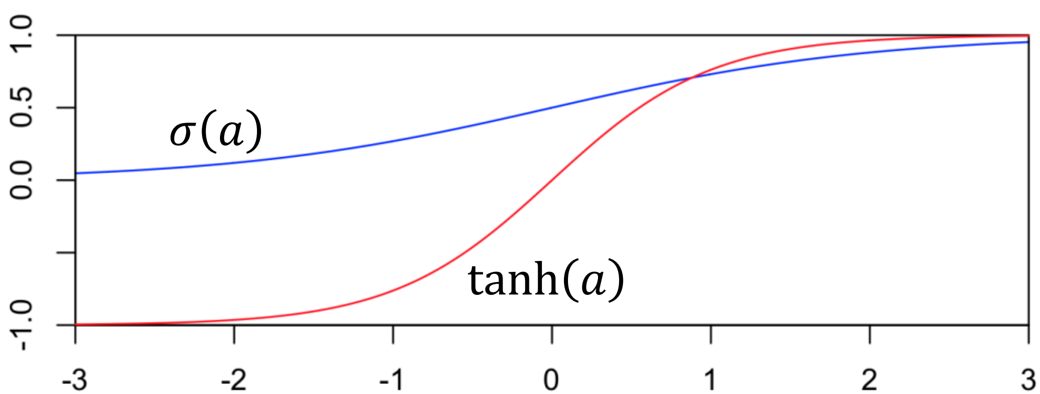
$$h_t = f_W(x_t, h_{t-1})$$
$$= \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

J. Elman, [Finding structure in time](), Cognitive science 14(2), pp. 179–211, 1990

Vanilla RNN (2)



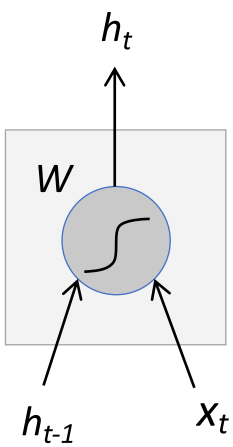
$$h_t = f_W(x_t, h_{t-1})$$
$$= \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$



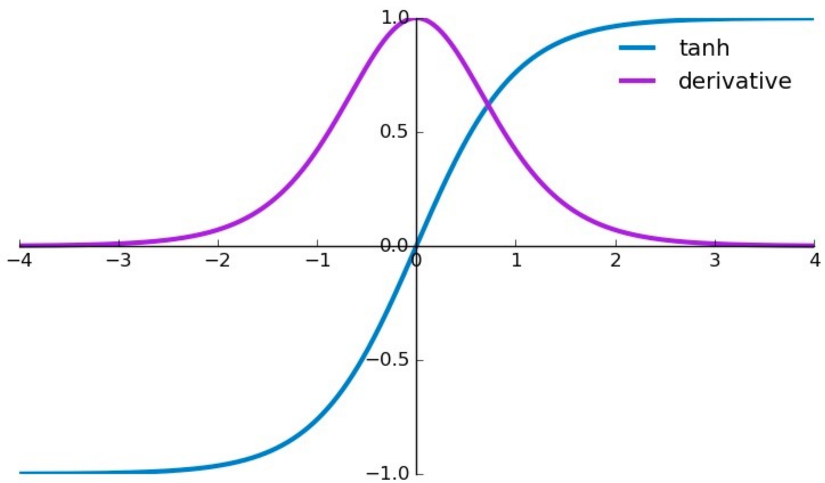
$$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$
$$= 2\sigma(2a) - 1$$

Image source

Vanilla RNN (3)



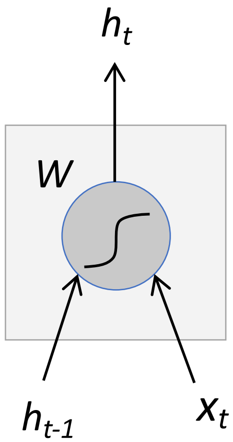
$$h_t = f_W(x_t, h_{t-1})$$
$$= \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$



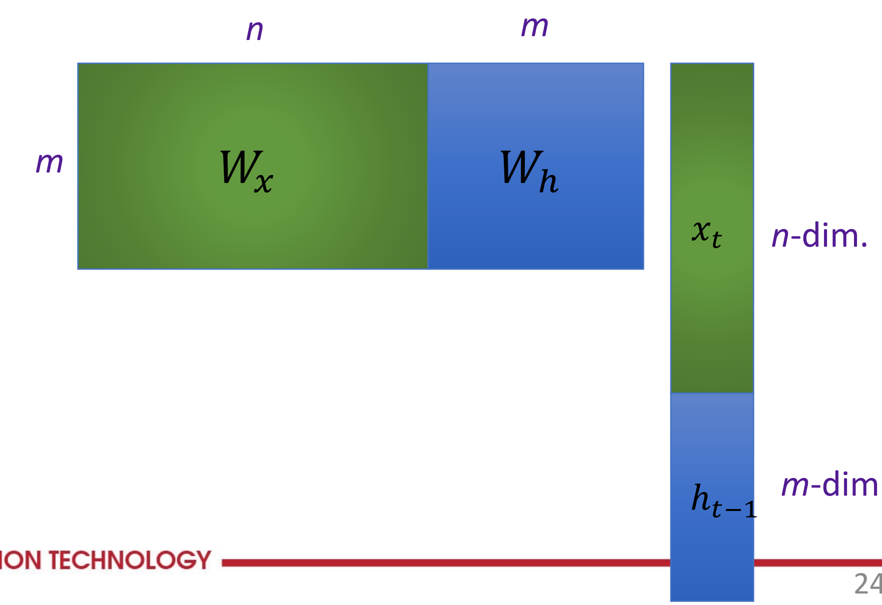
$$\frac{d}{da}\tanh(a) = 1 - \tanh^2(a)$$

Image source

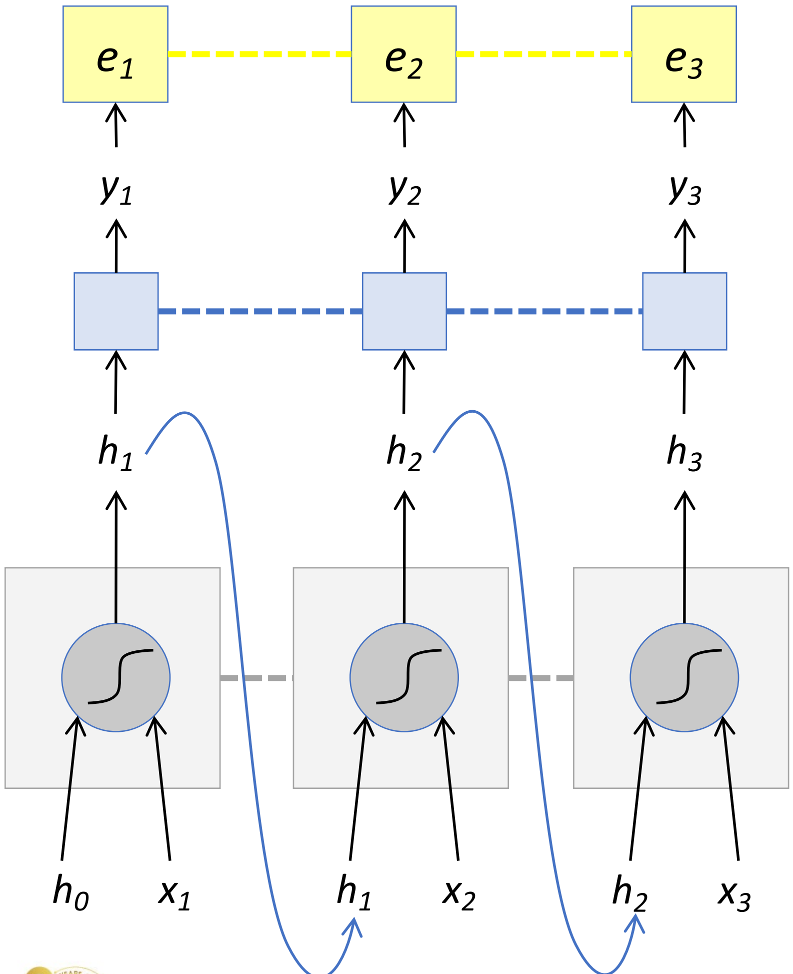
Vanilla RNN (4)



$$
\begin{aligned}
h_t &= f_W(x_t, h_{t-1}) \\
&= \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} \\
&= \tanh(W_x x_t + W_h h_{t-1})
\end{aligned}
$$



RNN Forward Pass



$$e_t = -\log(y_t(GT_t))$$

$$y_t = \text{softmax}(W_y h_t)$$

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

------ (shared weights)

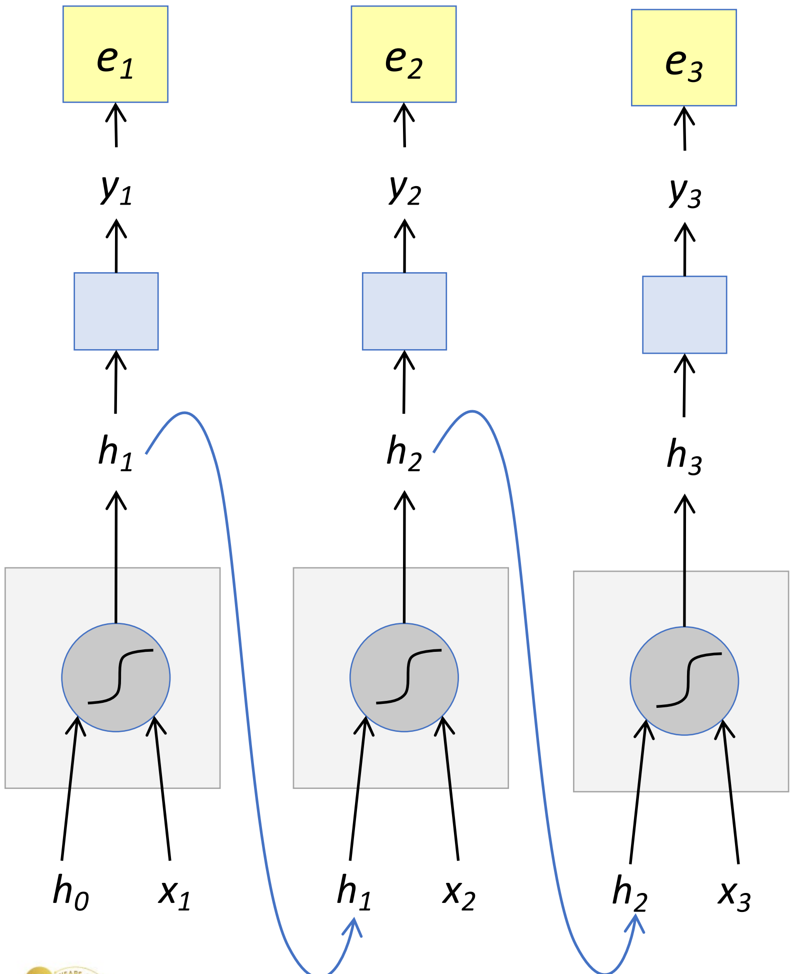
Backpropagation through time (BPTT)

Backpropagation through time (BPTT)

- Most common method for training RNNs.
- The network after unrolling is considered as a large feed-forward neural network that receives a whole data series as the input.
- The gradient for an RNN weight is computed at each of its replicas in the unfolded network, then summed (or averaged) and used to update the network weights.
- In practice, truncated BPTT is used: run the RNN forward k_1 time steps, propagate backward for k_2 time steps

https://machinelearningmastery.com/gentle-introduction-backpropagation-time/
http://www.cs.utoronto.ca/~ilya/pubs/ilya_sutskever_phd_thesis.pdf

Forward pass in unrolled RNN

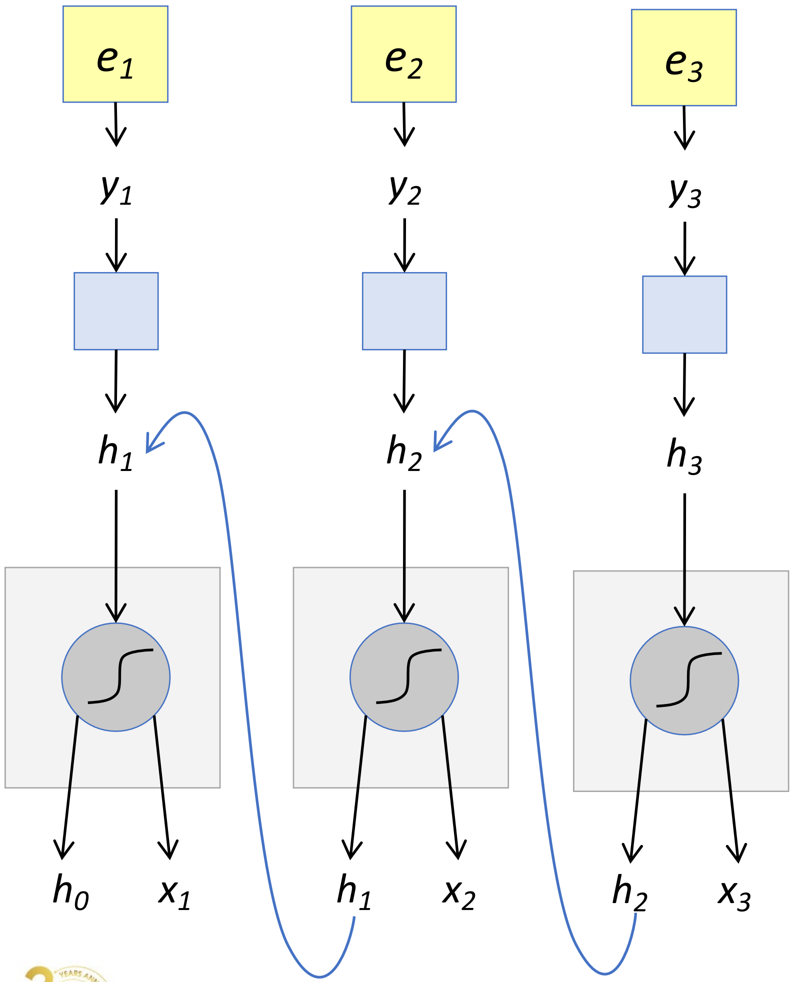


$$e_t = -\log(y_t(GT_t))$$

$$y_t = \text{softmax}(W_y h_t)$$

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

Backward pass in unrolled RNN

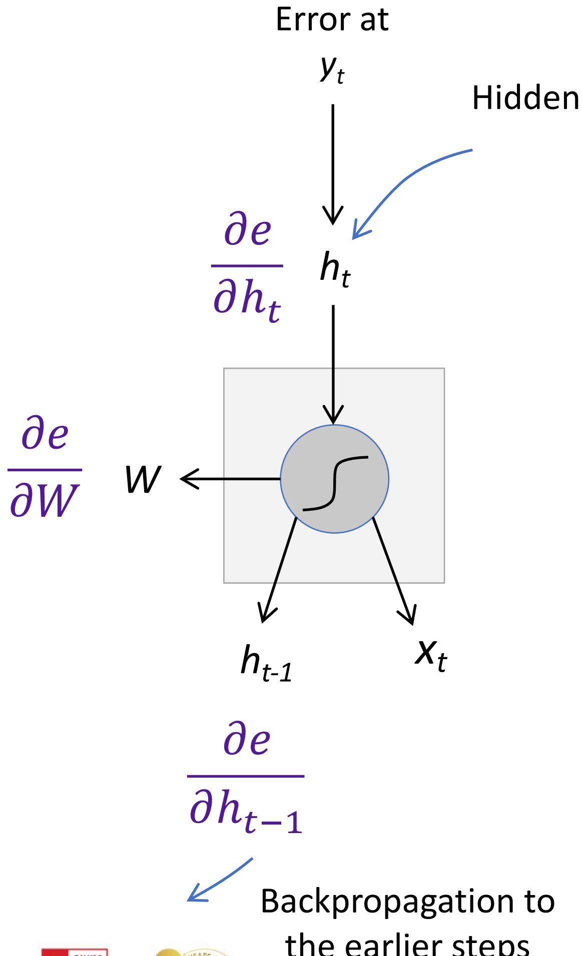


$$e_t = -\log(y_t(GT_t))$$

$$y_t = \text{softmax}(W_y h_t)$$

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

Backpropagation in RNN



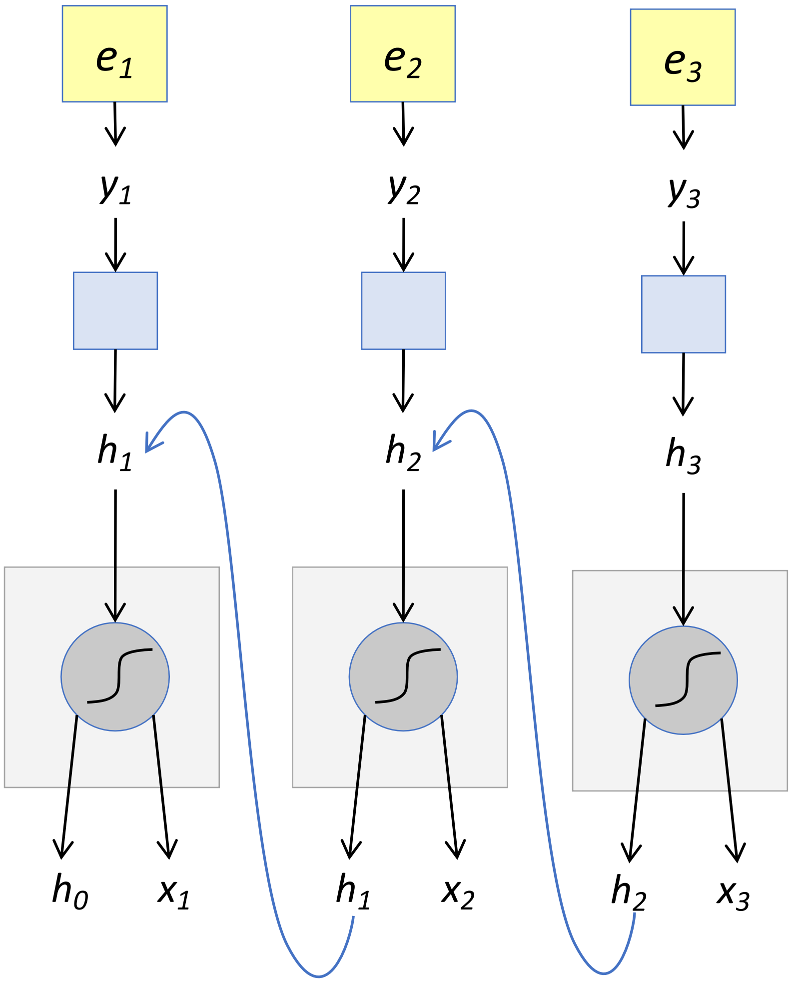
$$h_t = \tanh(W_x x_t + W_h h_{t-1})$$

$$\frac{\partial e}{\partial W_h} = \frac{\partial e}{\partial h_t} \odot \big(1 - \tanh^2(W_x x_t + W_h h_{t-1})\big)\, h_{t-1}^T$$

$$\frac{\partial e}{\partial W_x} = \frac{\partial e}{\partial h_t} \odot \big(1 - \tanh^2(W_x x_t + W_h h_{t-1})\big)\, x_t^T$$

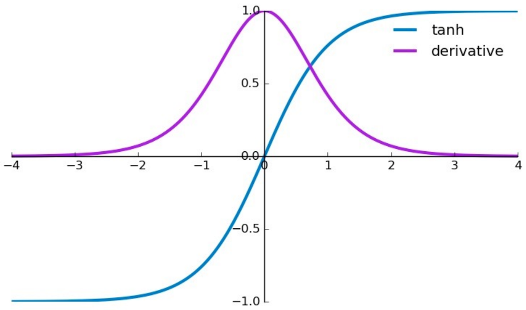
$$\frac{\partial e}{\partial h_{t-1}} = W_h^T \big(1 - \tanh^2(W_x x_t + W_h h_{t-1})\big) \odot \frac{\partial e}{\partial h_t}$$

Backpropagation in RNN (2)



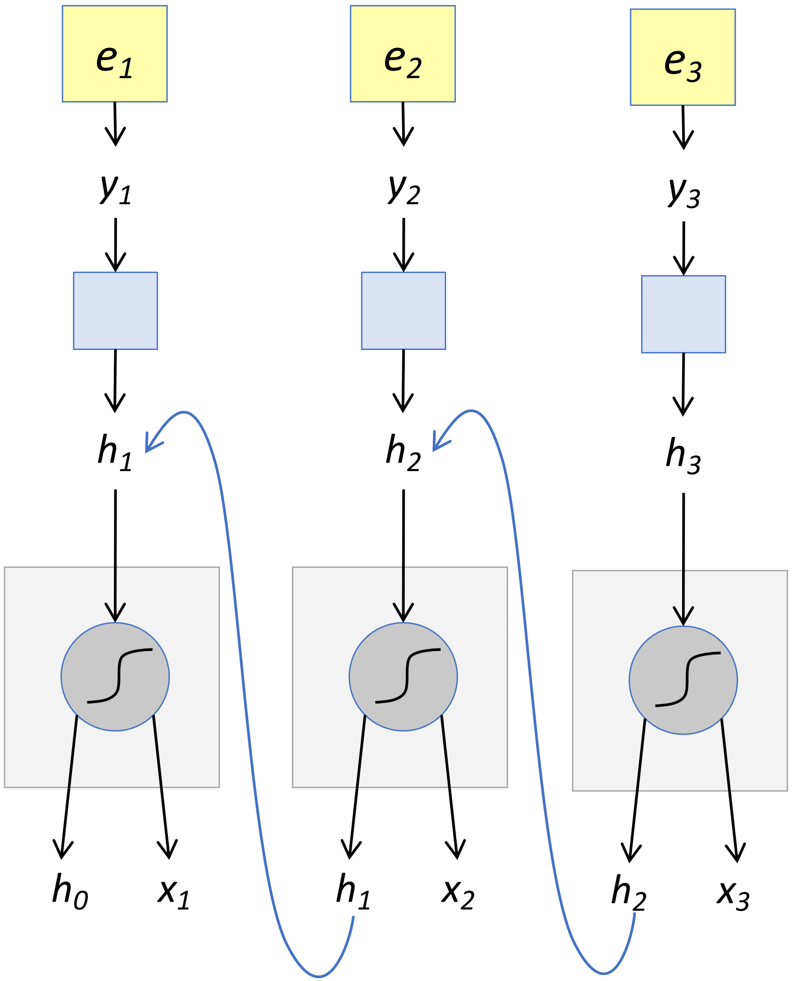
$$\frac{\partial e}{\partial h_{t-1}} = W_h^T\big(1 - \tanh^2(W_x x_t + W_h h_{t-1})\big) \odot \frac{\partial e}{\partial h_t}$$

Large tanh output value will correspond to small gradient (saturation region)



Calculate $\frac{\partial e_n}{\partial h_k}$ where $k \ll n$

Backpropagation in RNN (3)



$$\frac{\partial e}{\partial h_{t-1}} = W_h^T\left(1 - \tanh^2(W_x x_t + W_h h_{t-1})\right) \odot \frac{\partial e}{\partial h_t}$$

Largest eigenvalue of W_h < 1: Vanishing gradient

Calcuate $\frac{\partial e_n}{\partial h_k}$ where $k \ll n$

Brief thoughts

- Recall: $$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)$$
- What if σ were the identity function, $\sigma(x) = x$?

$$\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} = \operatorname{diag}\left(\sigma'\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)\right)\boldsymbol{W}_h \qquad \text{(chain rule)}$$
$$= \boldsymbol{I}\ \boldsymbol{W}_h = \boldsymbol{W}_h$$

- Consider the gradient of the loss $J^{(i)}(\theta)$ on step i, with respect to the hidden state $\boldsymbol{h}^{(j)}$ on some previous step j. *Let* $\ell = i - j$

$$\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(j)}} = \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \prod_{j<t\leq i} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} \qquad \text{(chain rule)}$$

$$= \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \prod_{j<t\leq i} \boldsymbol{W}_h = \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \boxed{\boldsymbol{W}_h^{\ell}} \qquad \text{(value of } \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}\text{)}$$

If W_h is "small", then this term gets exponentially problematic as ℓ becomes large

Brief thoughts (2)

- What's wrong with W_h^ℓ ?
- Consider if the eigenvalues of W_h are $\underbrace{\text{all less than 1}}_{\text{sufficient but not necessary}}$:

$$\lambda_1, \lambda_2, \ldots, \lambda_n < 1$$
$$\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_n \ \text{(eigenvectors)}$$

- We can write $\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \boldsymbol{W}_h^\ell$ using the eigenvectors of $\boldsymbol{W}_h$ as a basis:

$$\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \boldsymbol{W}_h^\ell = \sum_{i=1}^{n} c_i \boxed{\lambda_i^\ell} \boldsymbol{q}_i \approx \boldsymbol{0} \ \text{(for large } \ell\text{)}$$

Approaches 0 as ℓ grows so gradient vanishes

- What about nonlinear activations σ (i.e., what we use?)
  - Pretty much the same thing, except the proof requires $\lambda_i < \gamma$ for some γ dependent on dimensionality and σ

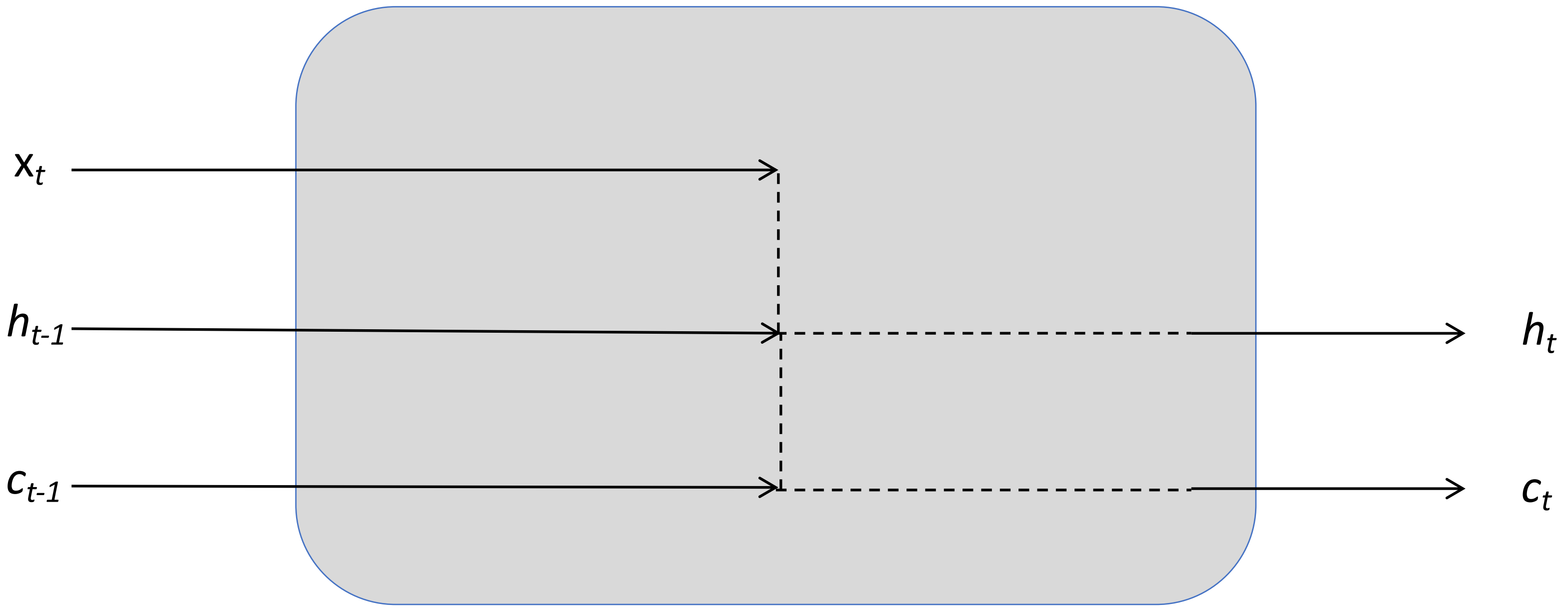
RNN tradeoffs

- RNN Advantages:
  - Can process any length input
  - Computation for step t can (in theory) use information from many steps back
  - Model size doesn’t increase for longer input
  - Same weights applied on every timestep, so there is symmetry in how inputs are processed.
- RNN Disadvantages:
  - Recurrent computation is slow
  - In practice, difficult to access information from many steps back

LSTM and GRU

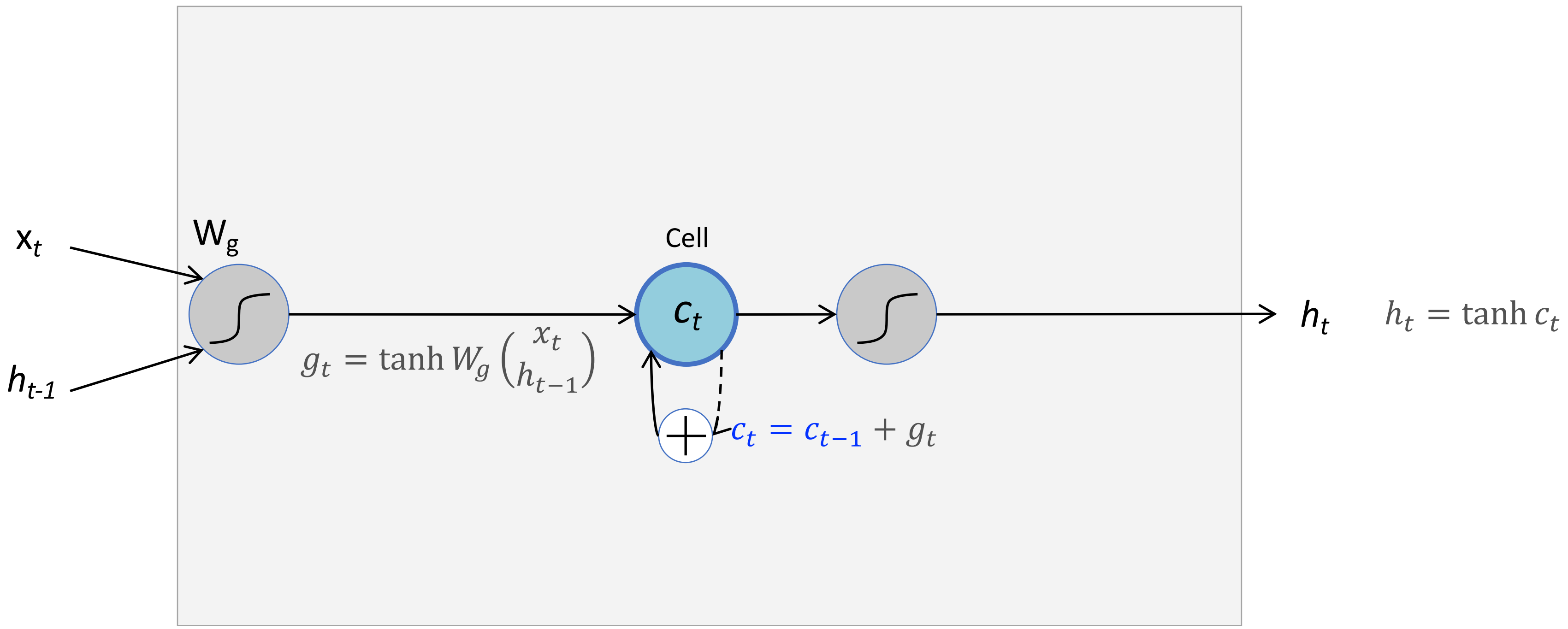
Long Short-Term Memory (LSTM)

- Use memory "cells" to avoid gradient suppression



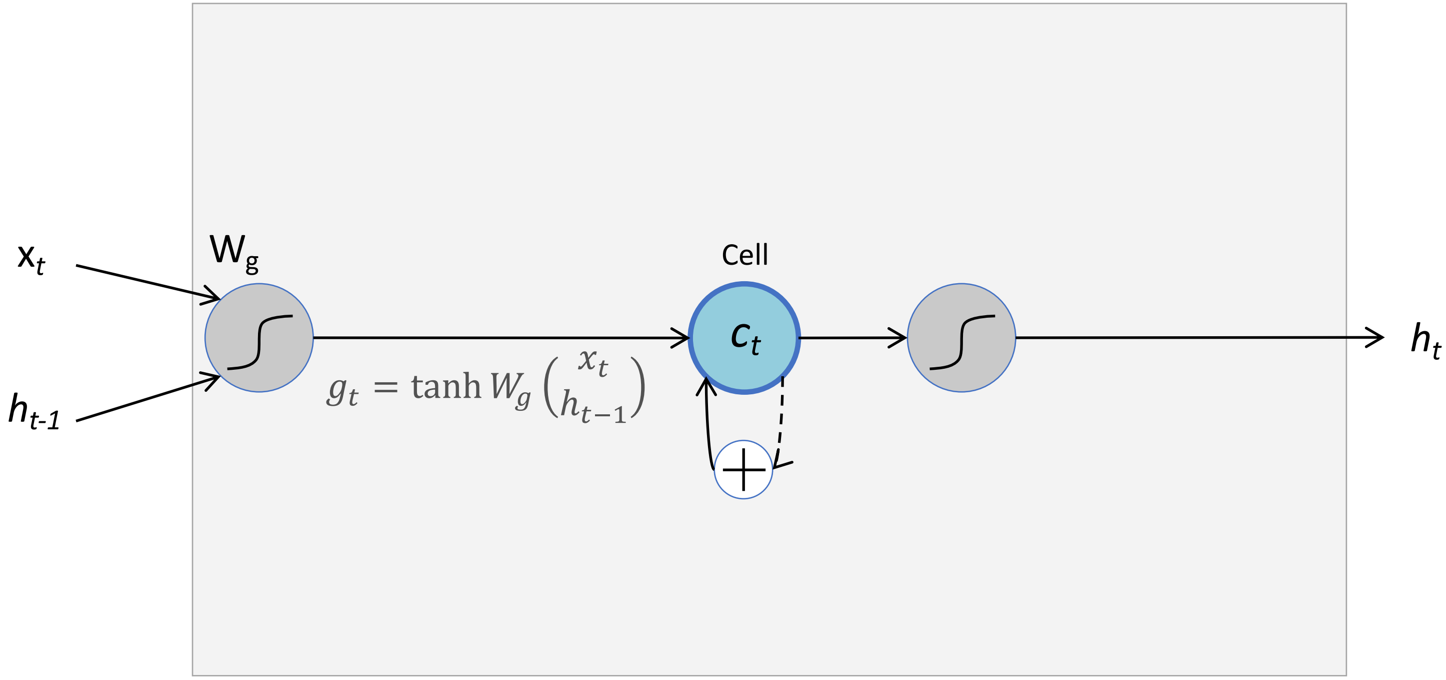
S. Hochreiter and J. Schmidhuber, [Long short-term memory](), Neural Computation 9 (8), pp. 1735–1780, 1997

LSTM Cell



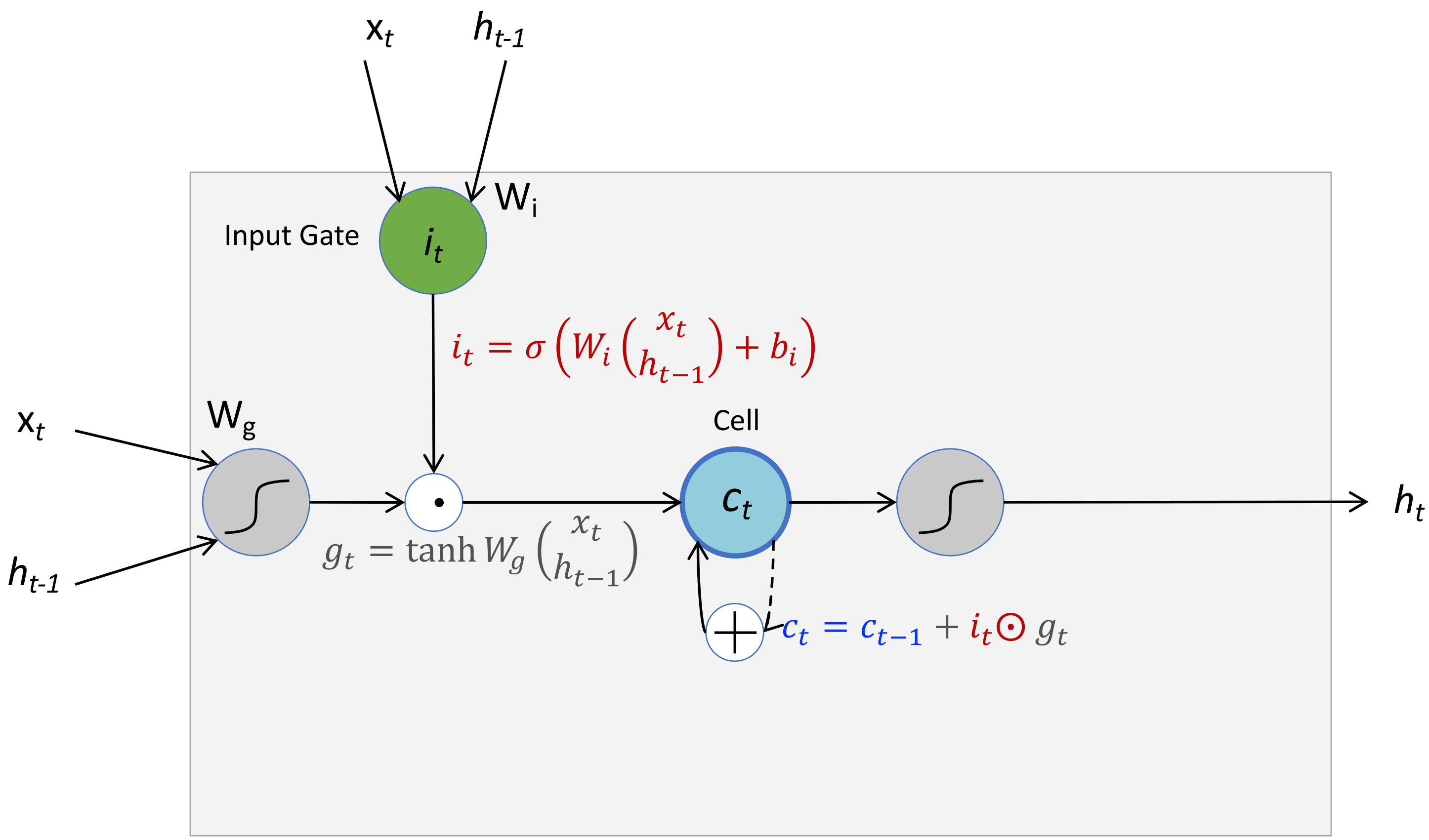
x_t
W_g
h_{t-1}
$g_t = \tanh W_g \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$
Cell
c_t
$c_t = c_{t-1} + g_t$
h_t
$h_t = \tanh c_t$

LSTM Cell



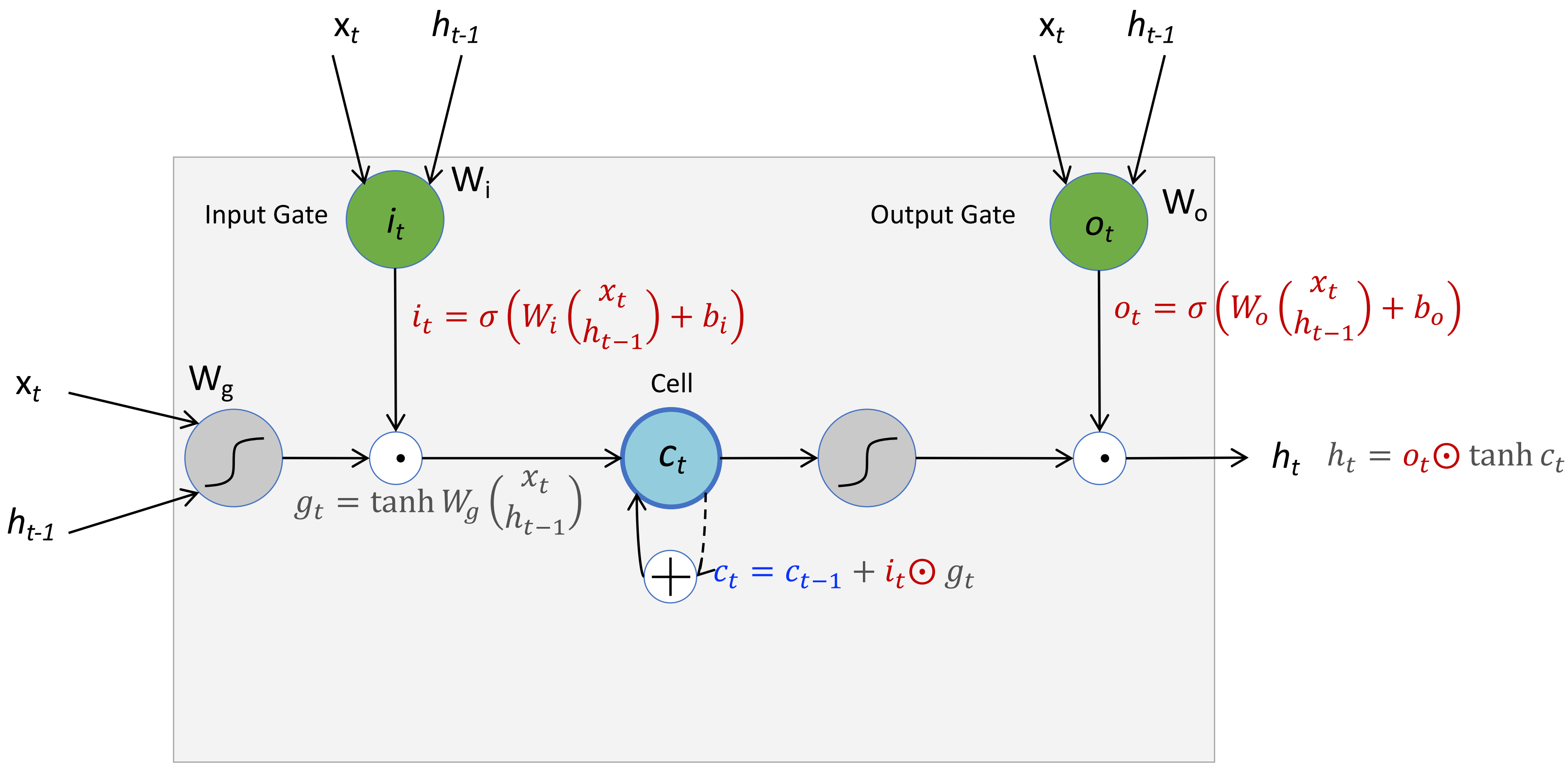
x_t
W_g
h_{t-1}
$g_t = \tanh W_g \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$
Cell
c_t
h_t

LSTM Cell



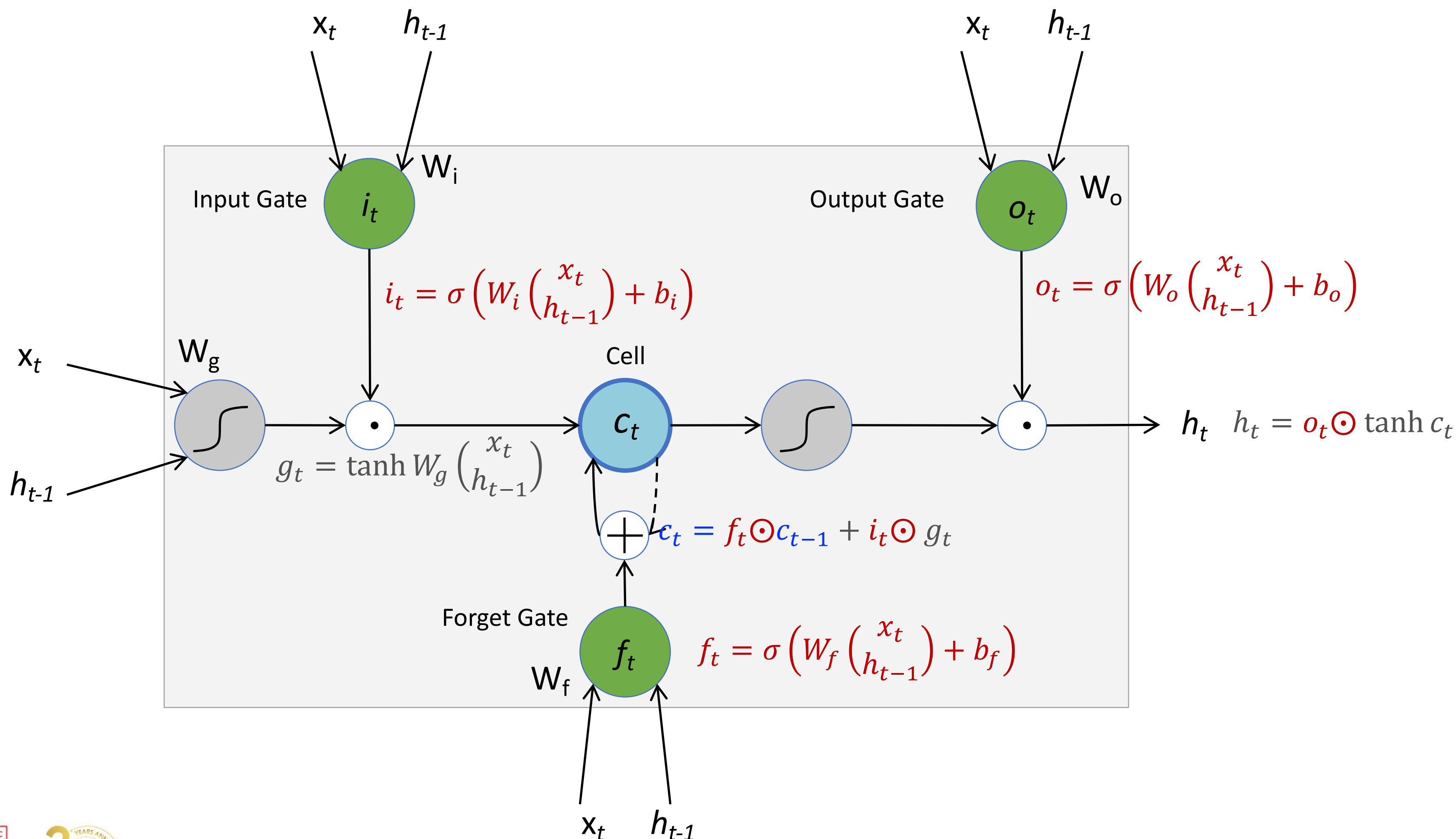
x_t
h_{t-1}
W_i
Input Gate
i_t
$i_t = \sigma\left(W_i \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_i\right)$
x_t
W_g
h_{t-1}
Cell
c_t
h_t
$g_t = \tanh W_g \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$
$c_t = c_{t-1} + i_t \odot g_t$

LSTM Cell



x_t
h_{t-1}
W_i
Input Gate
i_t
x_t
h_{t-1}
W_o
Output Gate
o_t
$i_t = \sigma\left(W_i \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_i\right)$
$o_t = \sigma\left(W_o \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_o\right)$
x_t
W_g
Cell
c_t
h_{t-1}
$g_t = \tanh W_g \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$
h_t
$h_t = o_t \odot \tanh c_t$
$c_t = c_{t-1} + i_t \odot g_t$

LSTM Cell



x_t
h_{t-1}
W_i
Input Gate
i_t
$i_t = \sigma\left(W_i \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_i\right)$
x_t
h_{t-1}
W_o
Output Gate
o_t
$o_t = \sigma\left(W_o \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_o\right)$
x_t
h_{t-1}
W_g
$g_t = \tanh W_g \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$
Cell
c_t
$c_t = f_t \odot c_{t-1} + i_t \odot g_t$
h_t
$h_t = o_t \odot \tanh c_t$
Forget Gate
W_f
f_t
$f_t = \sigma\left(W_f \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_f\right)$
x_t
h_{t-1}

LSTM Forward Pass Summary

- $\begin{pmatrix} g_t \\ i_t \\ f_t \\ o_t \end{pmatrix} = \begin{pmatrix} \tanh \\ \sigma \\ \sigma \\ \sigma \end{pmatrix} \begin{pmatrix} W_g \\ W_i \\ W_f \\ W_o \end{pmatrix} \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$
- $c_t = f_t \odot c_{t-1} + i_t \odot g_t$
- $h_t = o_t \odot \tanh c_t$

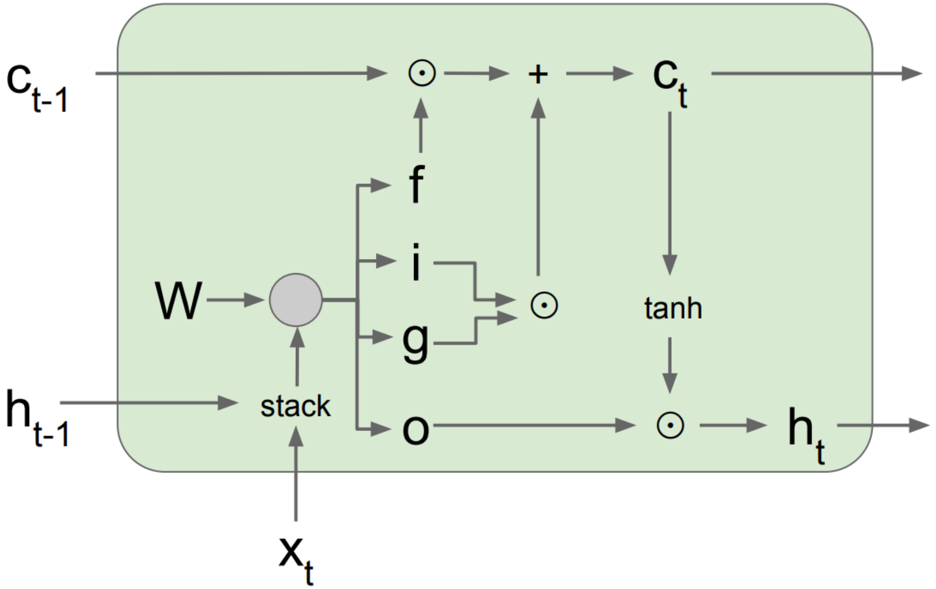
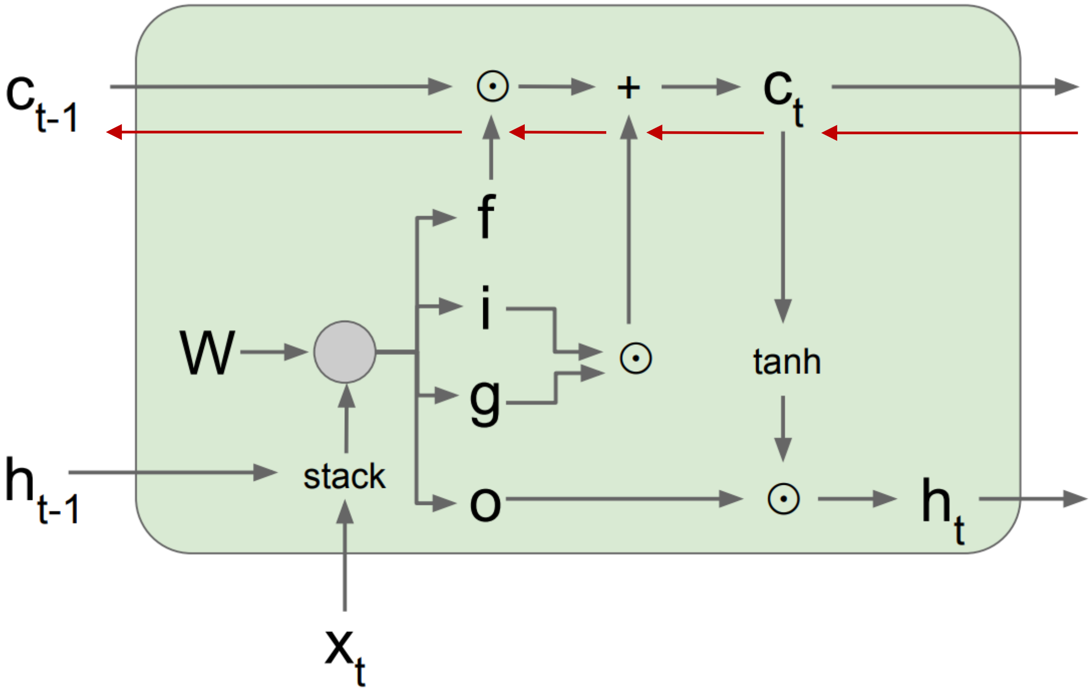


Figure source

Backpropagation in LSTM

- Gradient from c_t to c_{t-1} propagates back only through element-by-element multiplication, not through matrix multiplication and tanh functions.



Tham khảo thêm: [Illustrated LSTM Forward and Backward Pass]

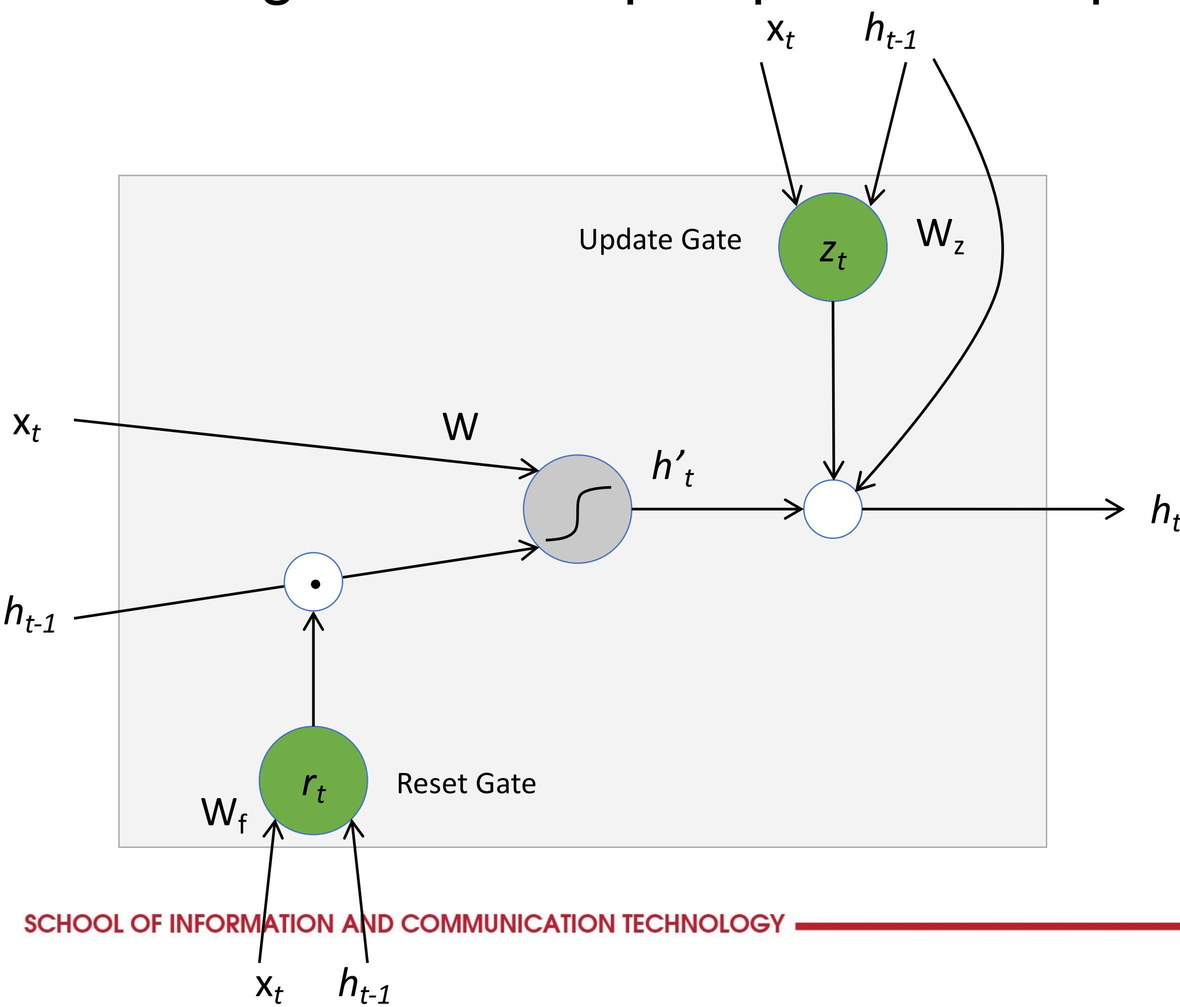
[Figure source]

Do LSTMs solve the vanishing gradient problem?

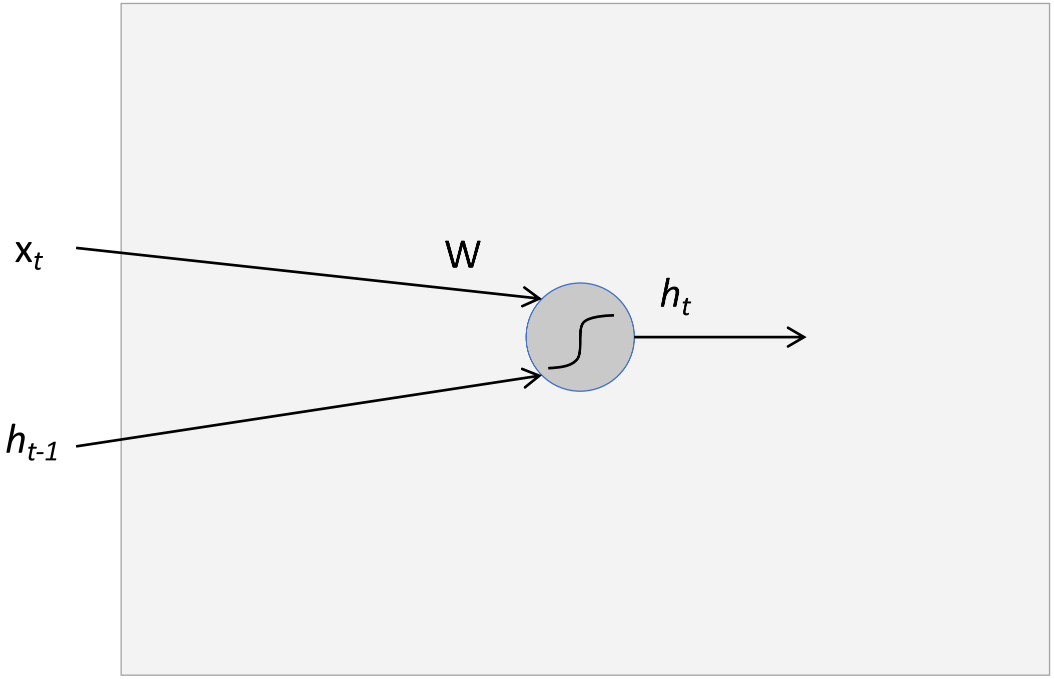
- The LSTM architecture makes it easier for the RNN to preserve information over many timesteps
  - e.g. if the f = 1 and the i = 0, then the information of that cell is preserved indefinitely.
  - By contrast, it's harder for vanilla RNN to learn a recurrent weight matrix Wh that preserves info in hidden state
- LSTM doesn't guarantee that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies

Gated Recurrent Unit (GRU)

- Do not use separate "cell state", combined with hidden state
- Combine "forget" and "output" ports into "update" ports

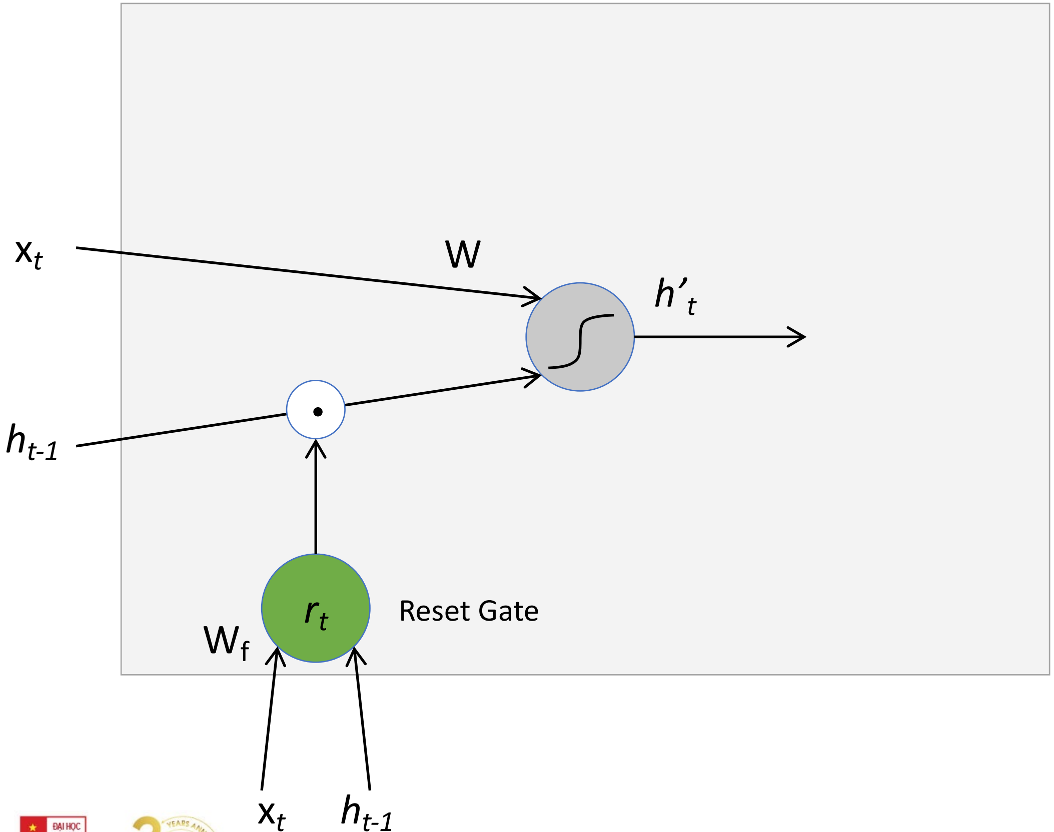


GRU (1)



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

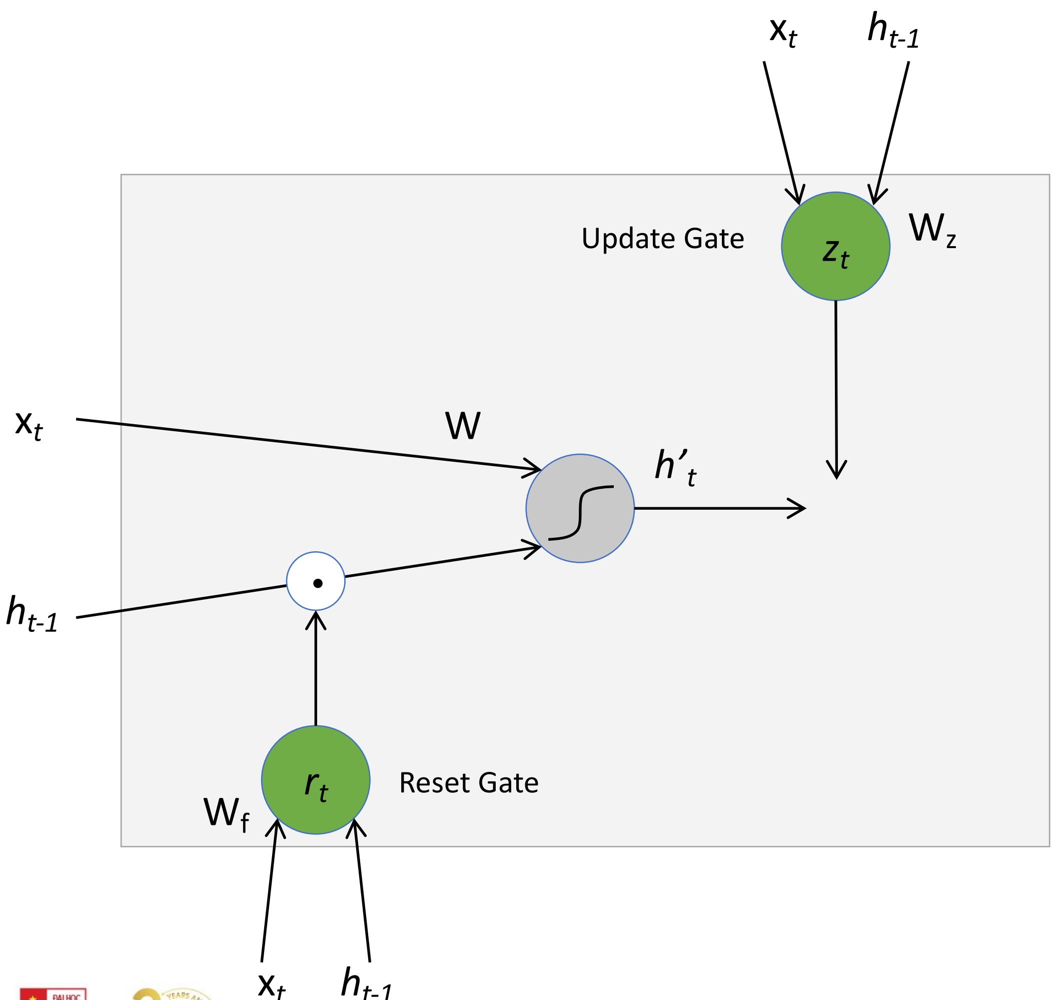
GRU (2)



$$r_t = \sigma\left(W_r\begin{pmatrix}x_t\\h_{t-1}\end{pmatrix} + b_t\right)$$

$$h'_t = \tanh W\begin{pmatrix}x_t\\r_t \odot h_{t-1}\end{pmatrix}$$

GRU (3)

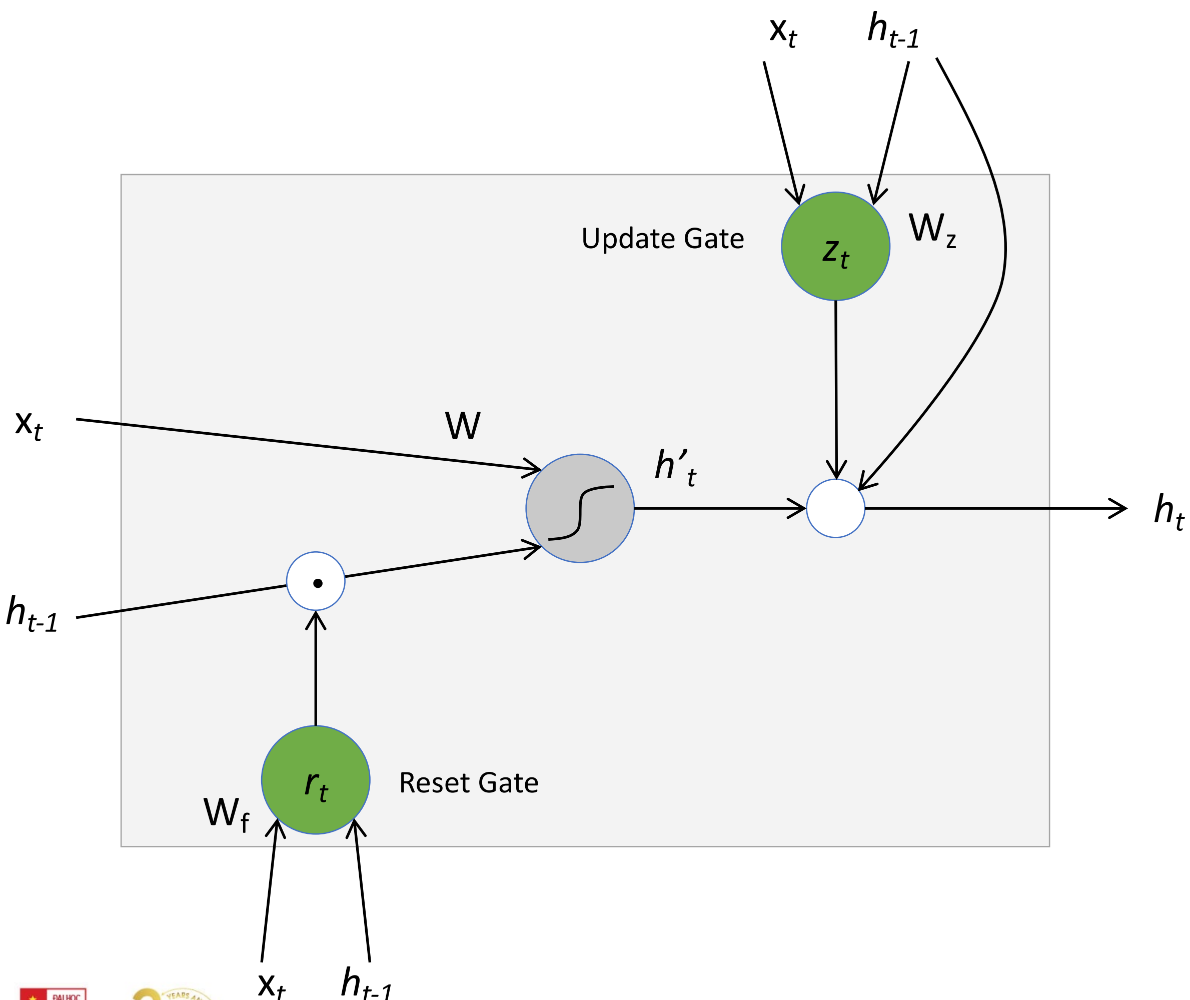


$$r_t = \sigma\left(W_r \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_t\right)$$

$$h'_t = \tanh W \begin{pmatrix} x_t \\ r_t \odot h_{t-1} \end{pmatrix}$$

$$z_t = \sigma\left(W_z \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_z\right)$$

GRU (4)



$$r_t = \sigma\left(W_r\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_t\right)$$

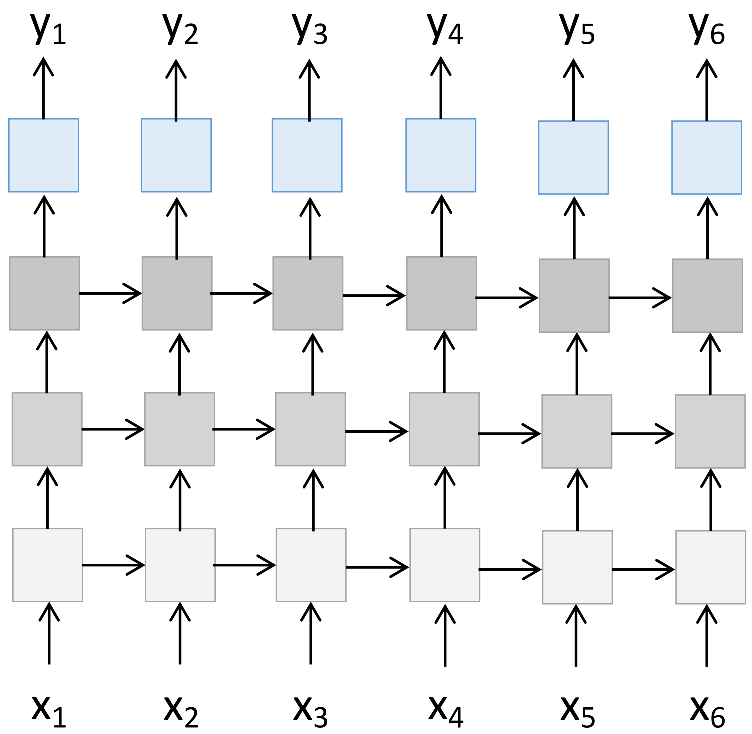
$$h'_t = \tanh W\begin{pmatrix} x_t \\ r_t \odot h_{t-1}\end{pmatrix}$$

$$z_t = \sigma\left(W_z\begin{pmatrix} x_t \\ h_{t-1}\end{pmatrix} + b_z\right)$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot h'_t$$

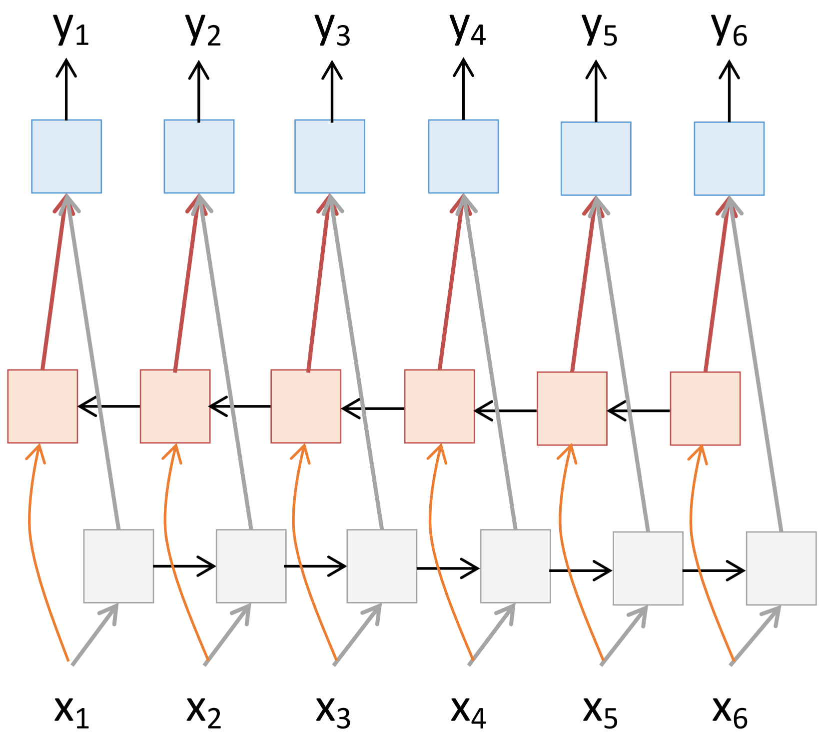
Multi-layer RNNs

- It is possible to design RNNs with many hidden layers



Bidirectional RNNs

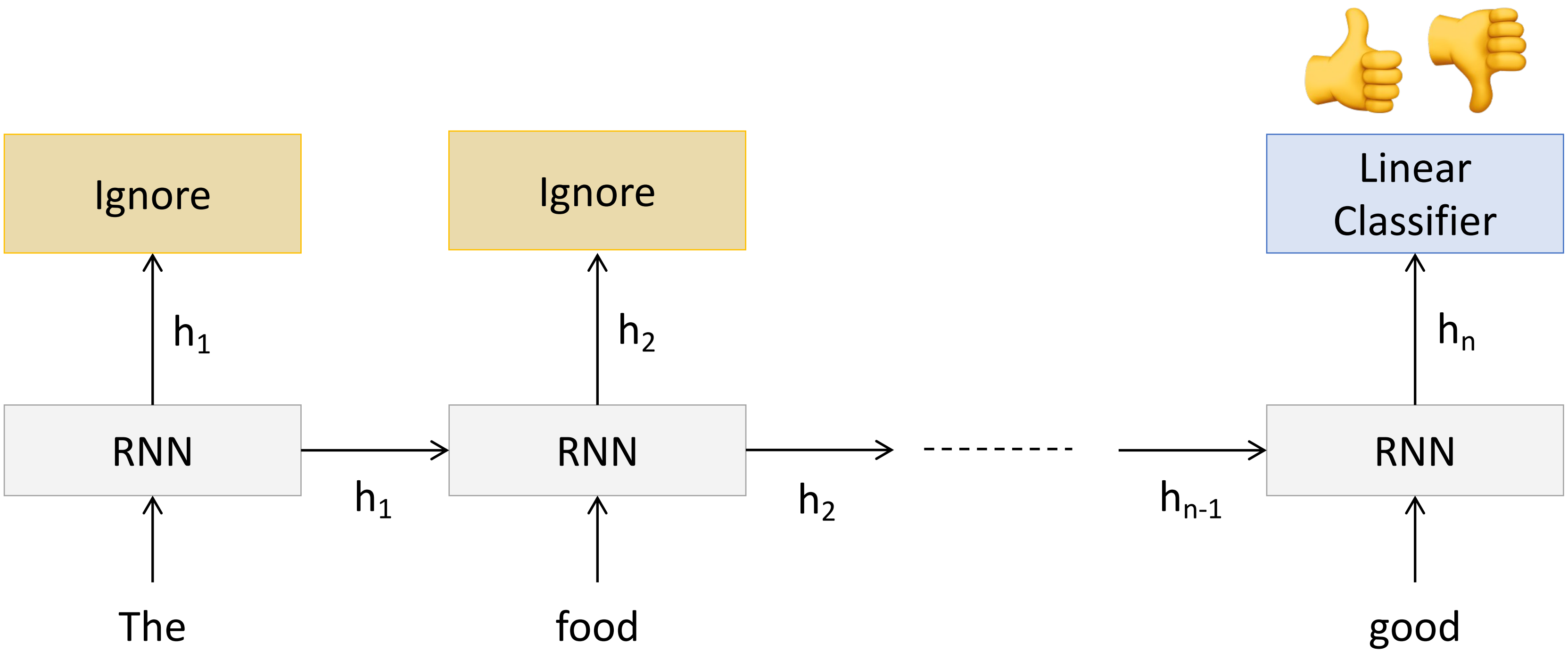
- RNNs can process the input sequence in reverse and forward direction



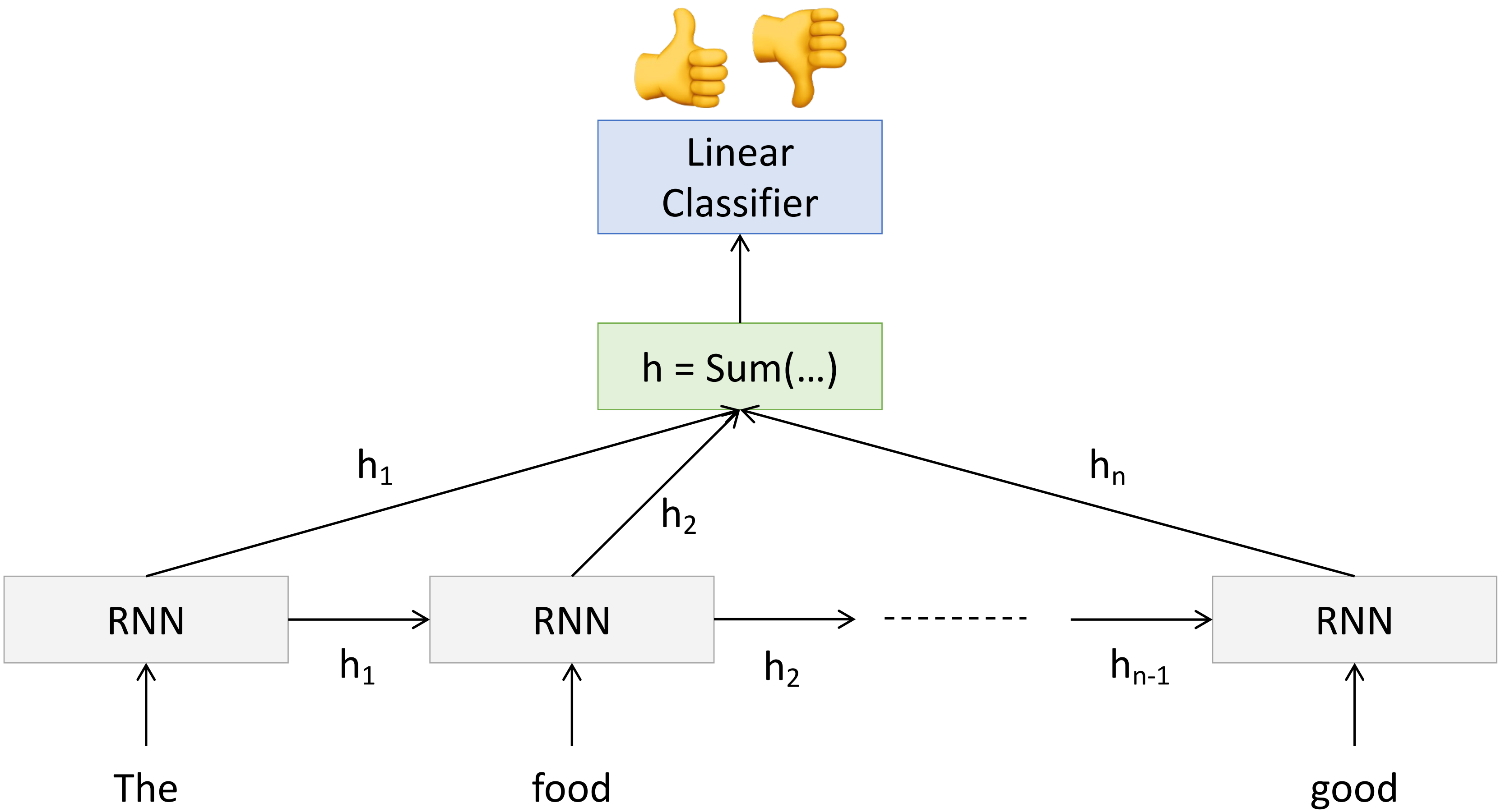
- Popular in sound recognition

RNN applications

Sequence classification

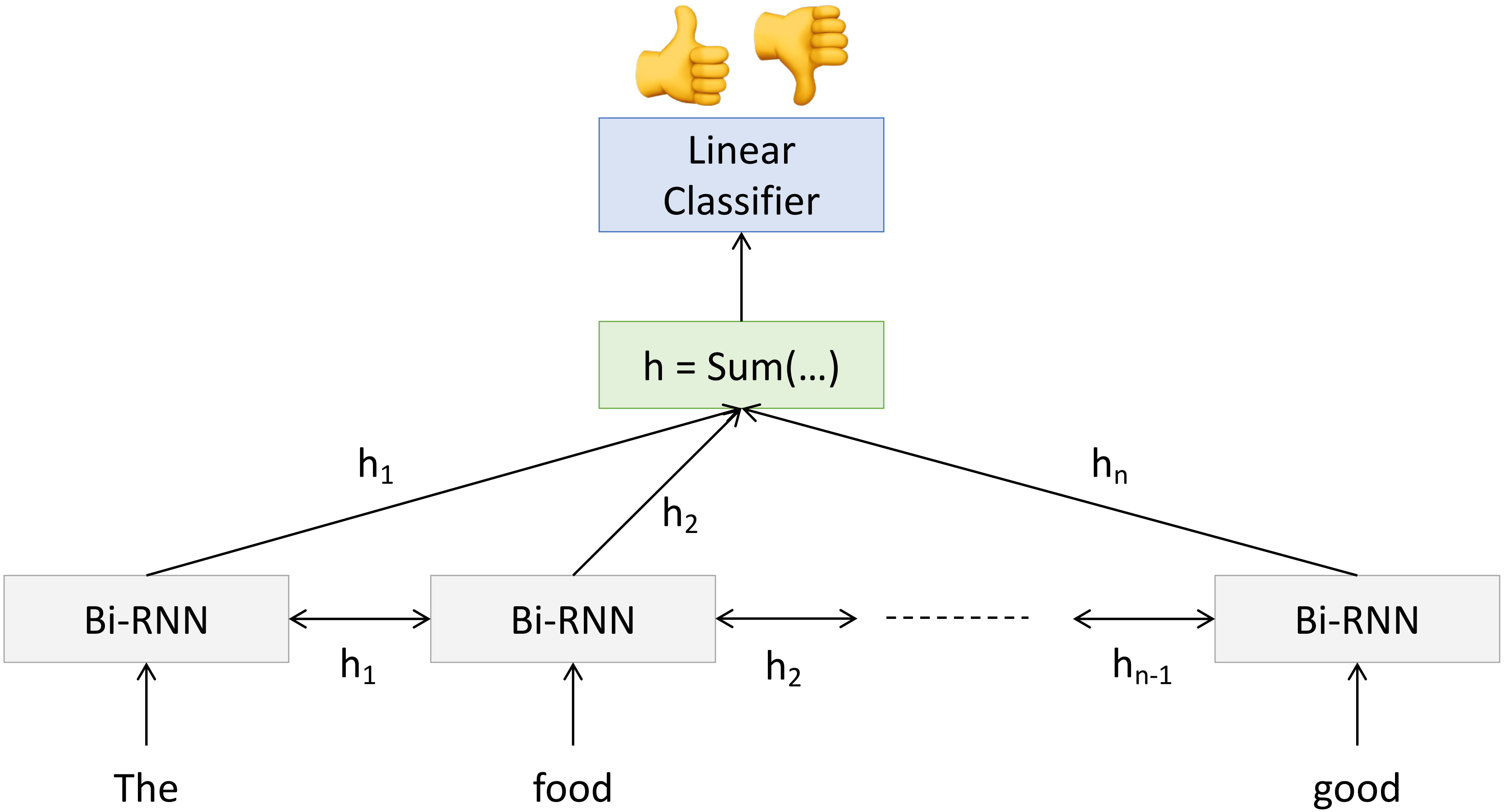


Sequence classification (2)



http://deeplearning.net/tutorial/lstm.html

Sequence classification (3)



Character RNN

100th iteration

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

↓ train more

300th iteration

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

↓ train more

700th iteration

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

↓ train more

2000th iteration

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Image source

Image caption generation

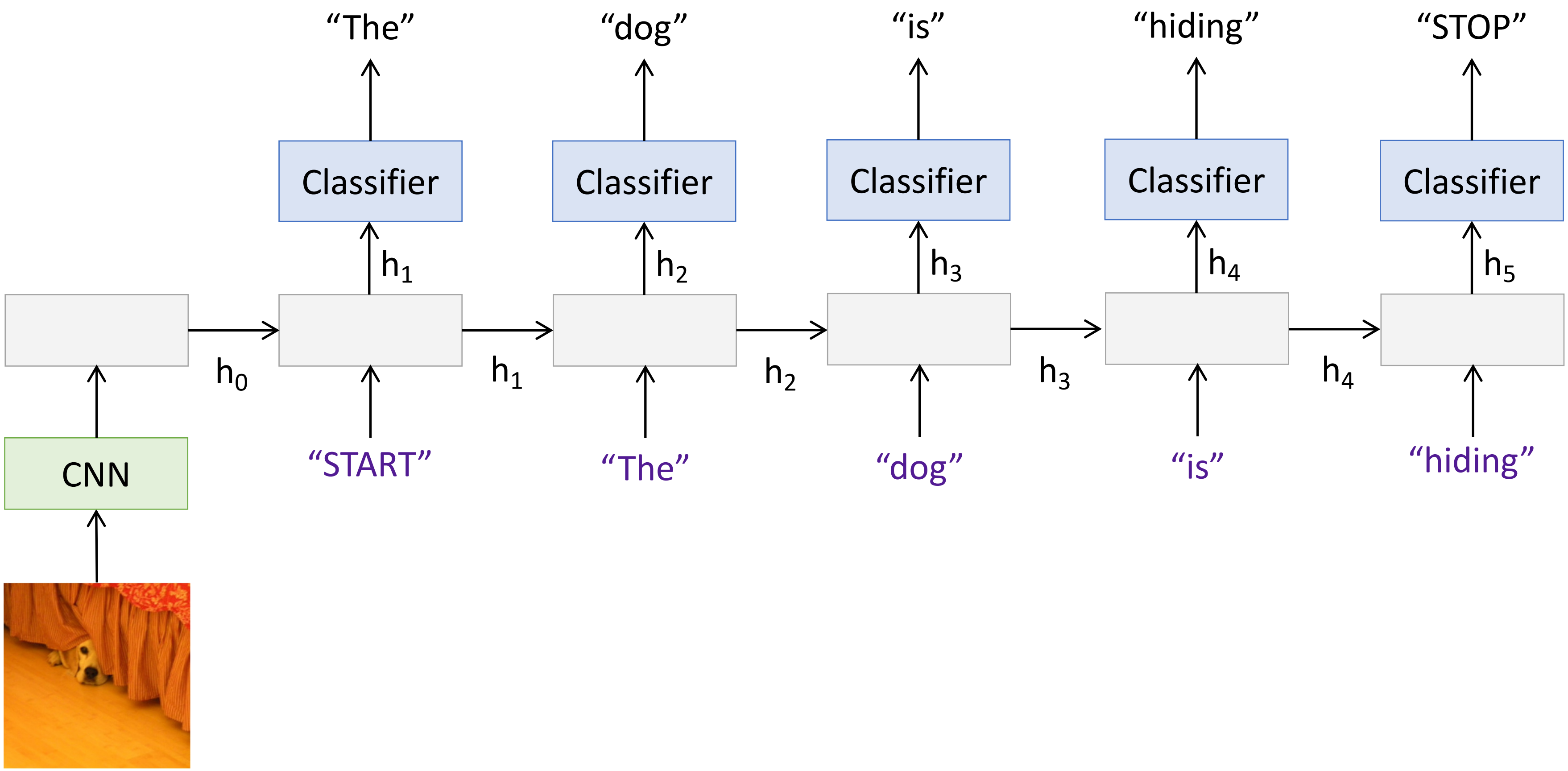
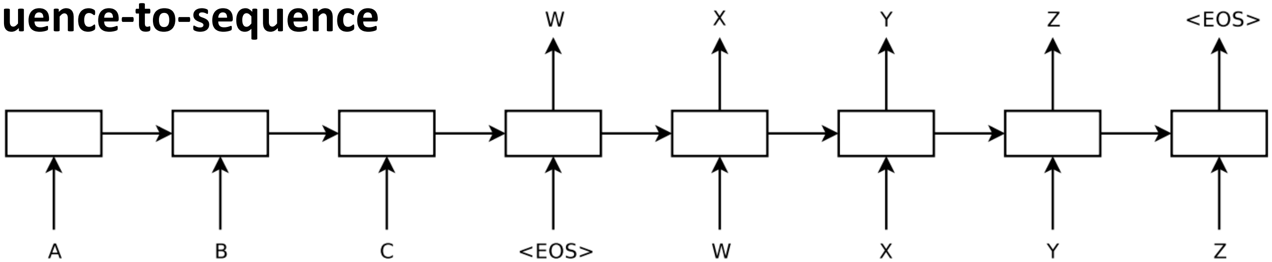


Image caption generation (2)

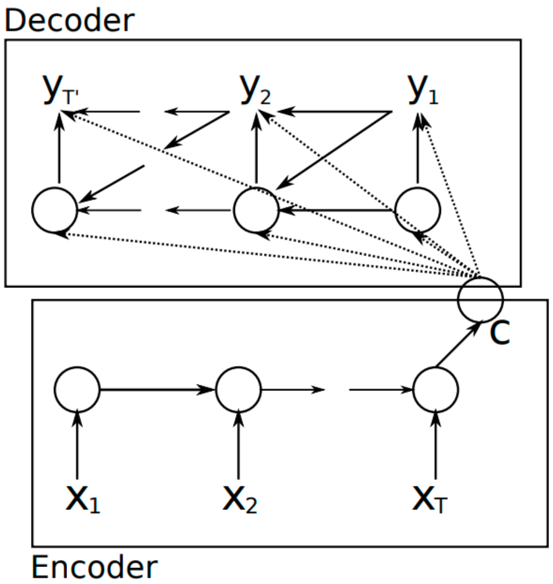
Neural machine translation

Sequence-to-sequence



I. Sutskever, O. Vinyals, Q. Le, [Sequence to Sequence Learning with Neural Networks](), NIPS 2014

Encoder-decoder



K. Cho, B. Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio, Learning phrase representations using RNN encoder-decoder for statistical machine translation, ACL 2014

References

1. http://cs231n.stanford.edu

2. http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture06-rnnlm.pdf

http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture07-fancy-rnn.pdf

3. Training RNNs: http://www.cs.toronto.edu/~rgrosse/csc321/lec10.pdf