

# Recurrent Neural Networks

Minlie Huang

[aihuang@tsinghua.edu.cn](mailto:aihuang@tsinghua.edu.cn)

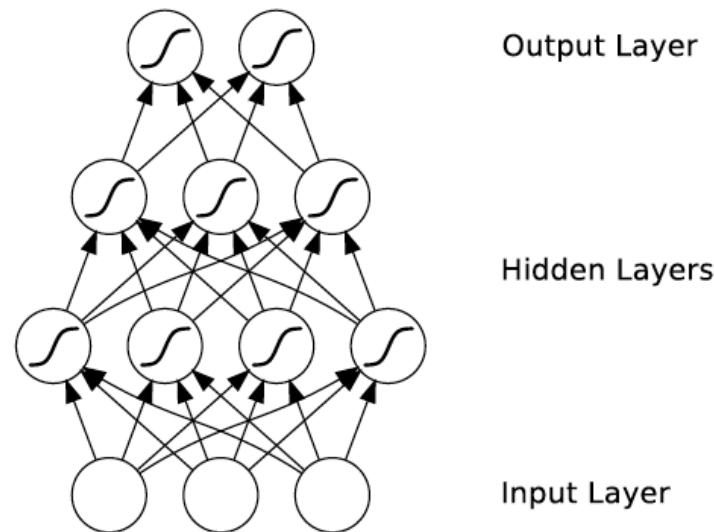
Dept. of Computer Science and Technology

Tsinghua University

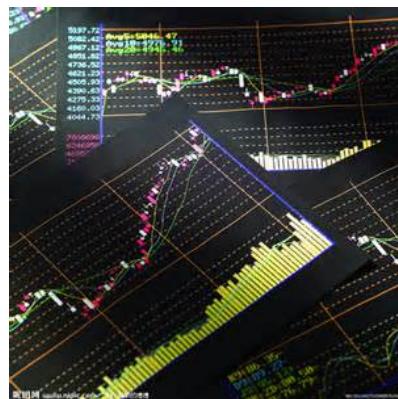
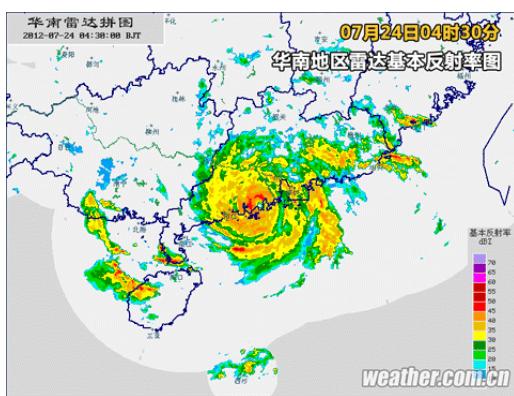
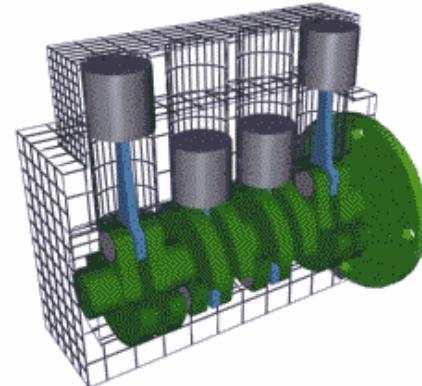
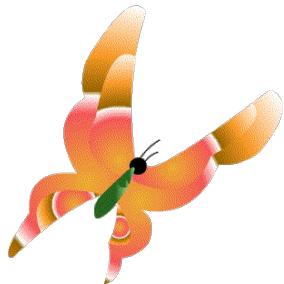
<http://coai.cs.tsinghua.edu.cn/hml/>

# What's wrong with MLPs?

- Problem 1: Can't model sequences
  - Fixed-sized inputs & outputs
  - No temporal structure, thus no dynamics
- Problem 2: Pure feed-forward processing
  - No “memory”, no feedback



# Dynamic Systems

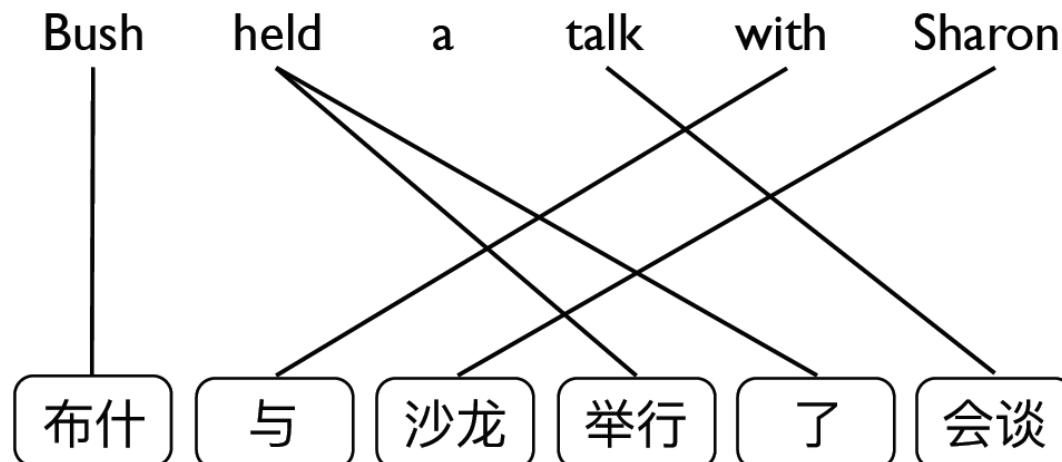


# Sequences are everywhere

- Signal to natural language

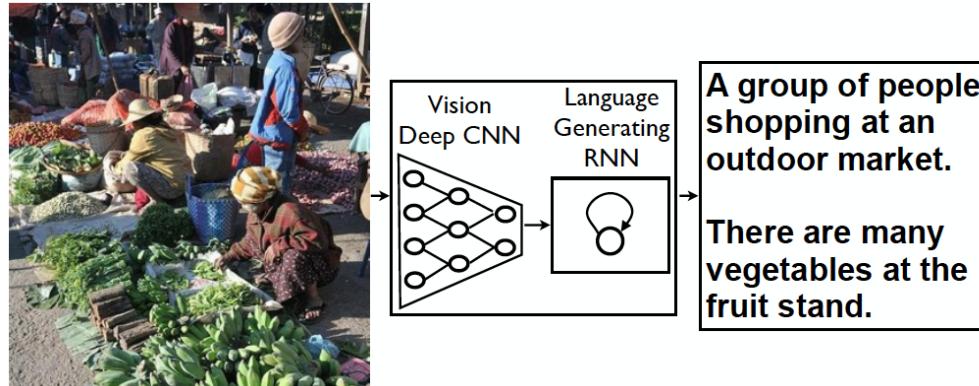


- Translation from English to Chinese

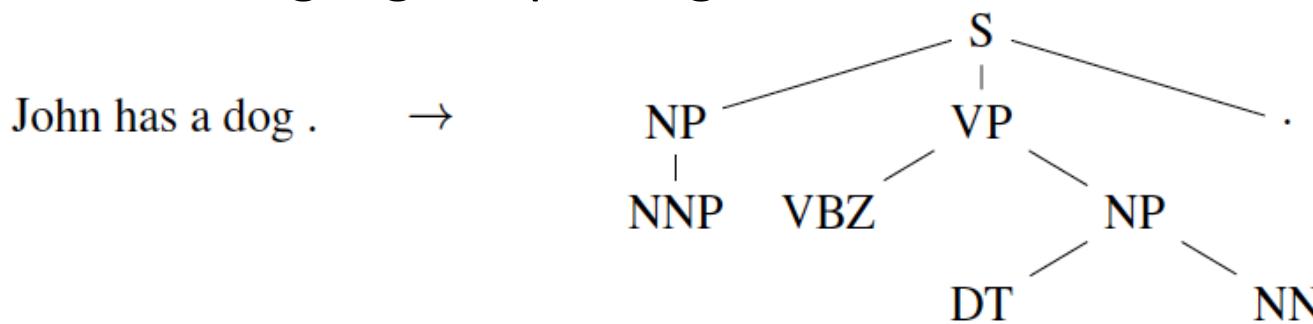


# Sequences are everywhere

- Image to description

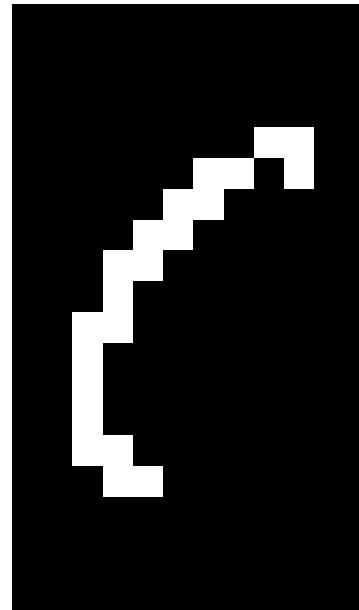
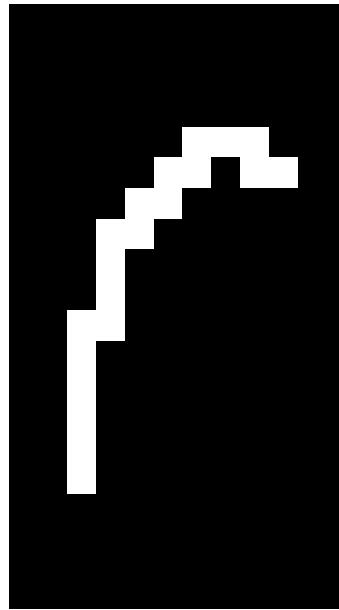


- Natural language to parsing tree

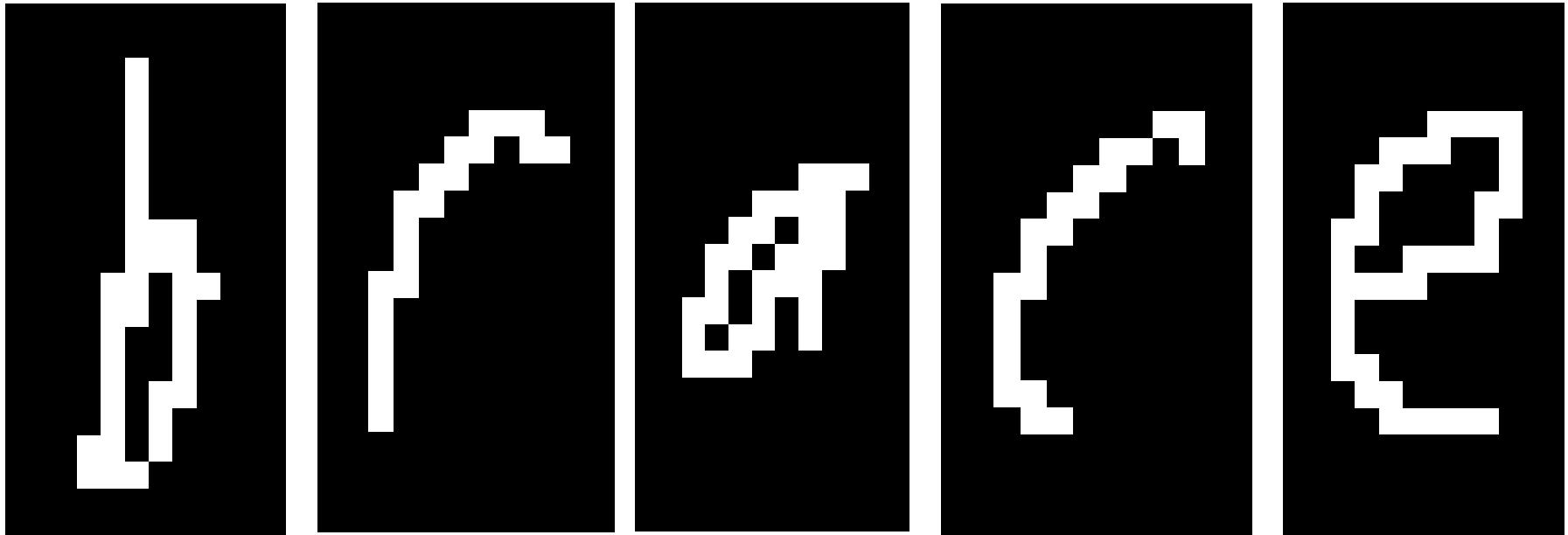


John has a dog . → (S (NP NNP )<sub>NP</sub> (VP VBZ (NP DT NN )<sub>NP</sub> )<sub>VP</sub> . )<sub>S</sub>

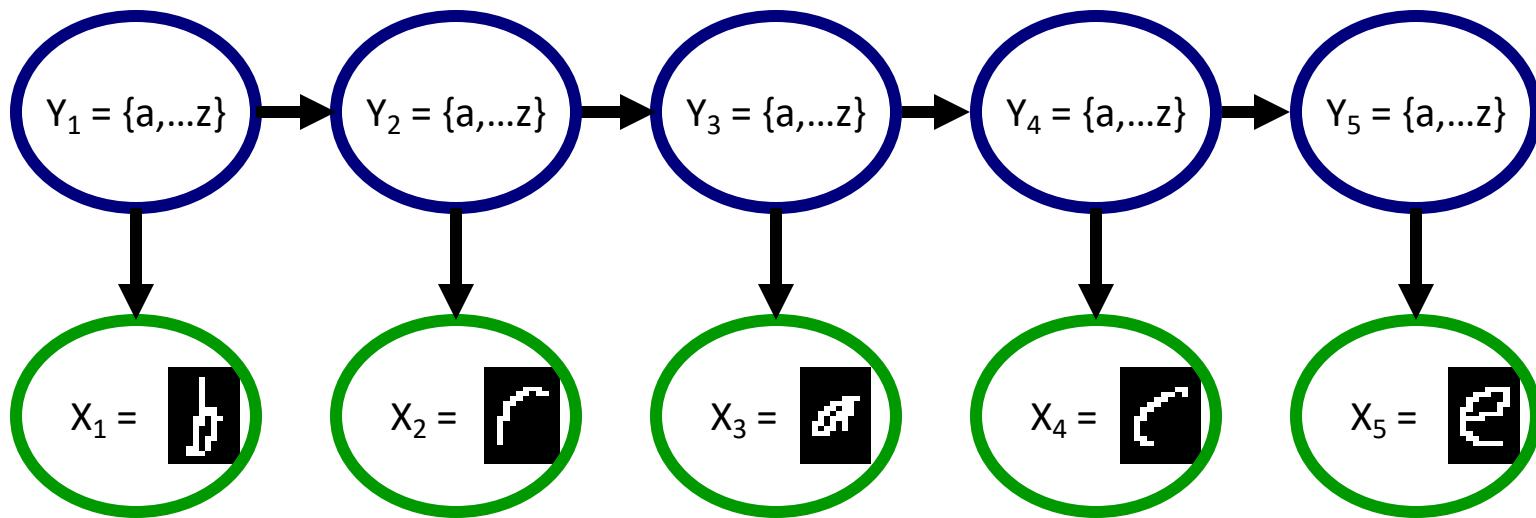
# Why model sequences?



# Why model sequences?



# Why model sequences?

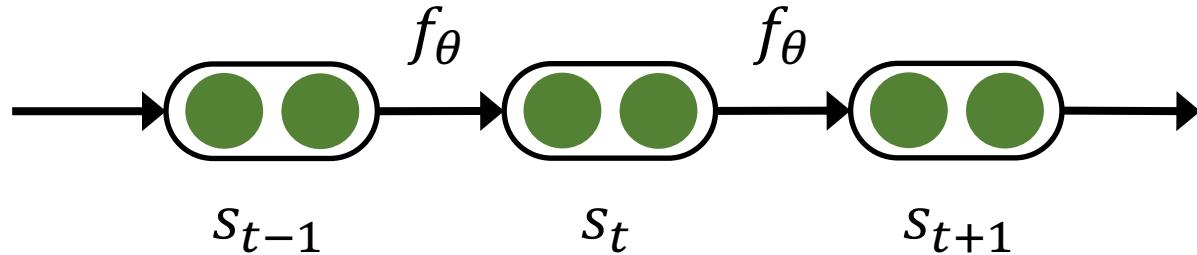


Hidden Markov Model (HMM):

$$P(X_{1..n}, Y_{1..n}) = P(Y_1)P(X_1|Y_1) \cdot \prod_{i=2}^n P(Y_i|Y_{i-1})P(X_i|Y_i)$$

# How to model sequences?

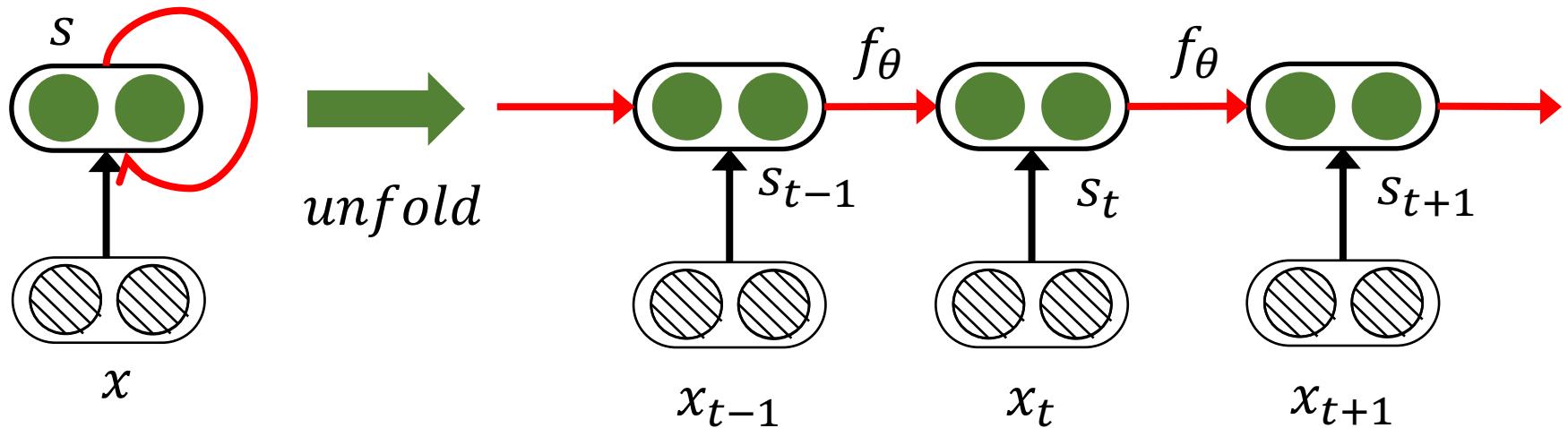
- No input



$$s_t = f_\theta(s_{t-1})$$

# How to model sequences?

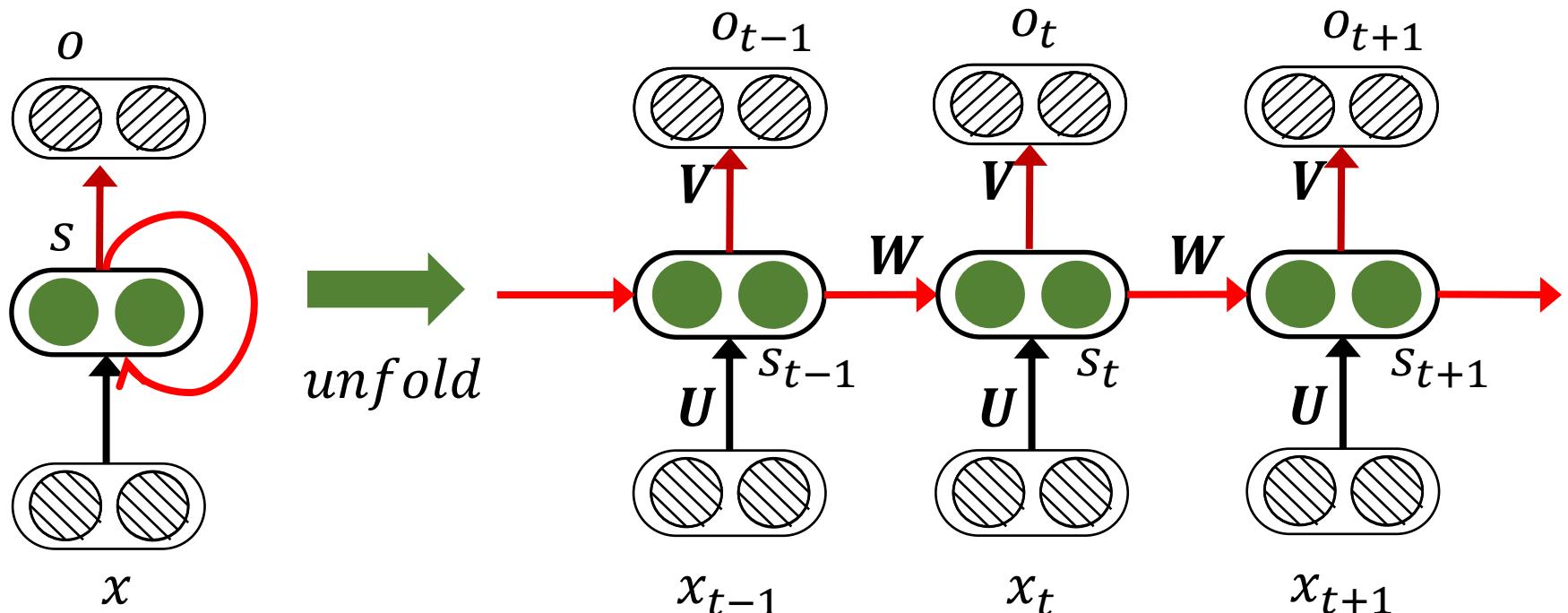
- With inputs



$$s_t = f_\theta(s_{t-1}, x_t)$$

# How to model sequences?

- With inputs and outputs

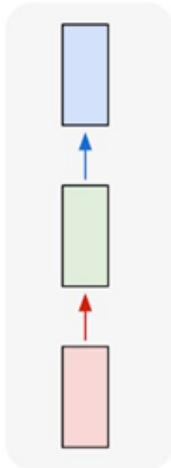


$$s_t = f(W * s_{t-1} + U * x_t)$$

$$o_t = g(V * s_t)$$

# How to model sequences?

one to one

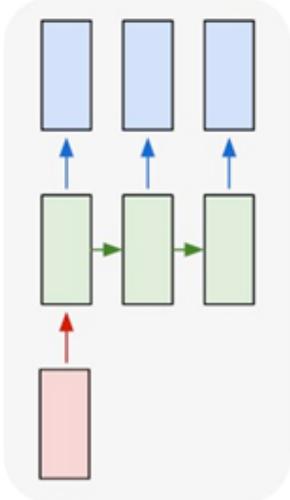


**Input:** No sequence

**Output:** No sequence

**Example:** “standard” classification / regression

one to many

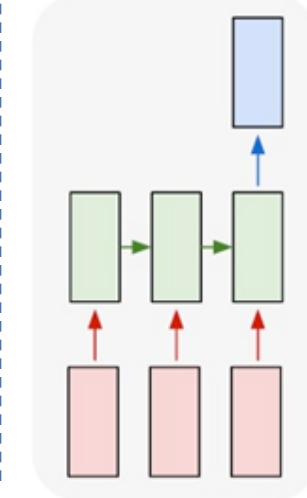


**Input:** No sequence

**Output:** Sequence

**Example:** Image2Caption

many to one

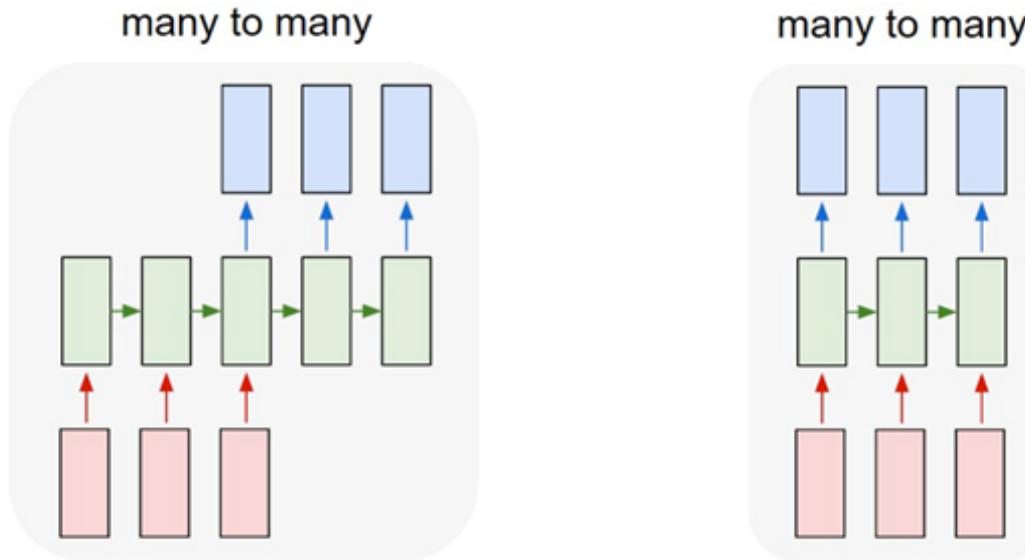


**Input:** Sequence

**Output:** No sequence

**Example:** sentence classification, multiple-choice question answering

# How to model sequences?



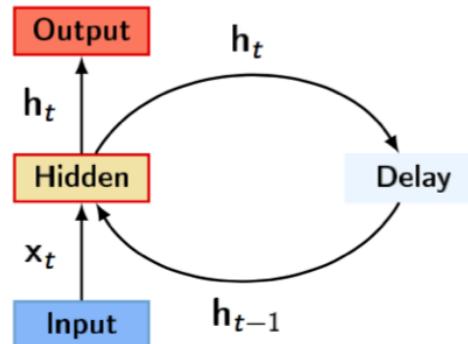
**Input:** Sequence, **Output:** Sequence

**Example:** machine translation, video captioning, open-ended question answering, video question answering

# Recurrent Neural Network (RNN)

- The RNN has recurrent hidden states whose output at each time is dependent on that of the previous time.
- Given a sequence  $x(1:n) = (x_1, x_2, \dots, x_t, \dots, x_n)$  (**each  $x_i$  is a vector**), the RNN updates its recurrent hidden state  $h(t)$  by

$$h_t = \begin{cases} 0 & \text{if } t = 0 \\ f(h_{t-1}, x_t) & \text{otherwise} \end{cases}$$



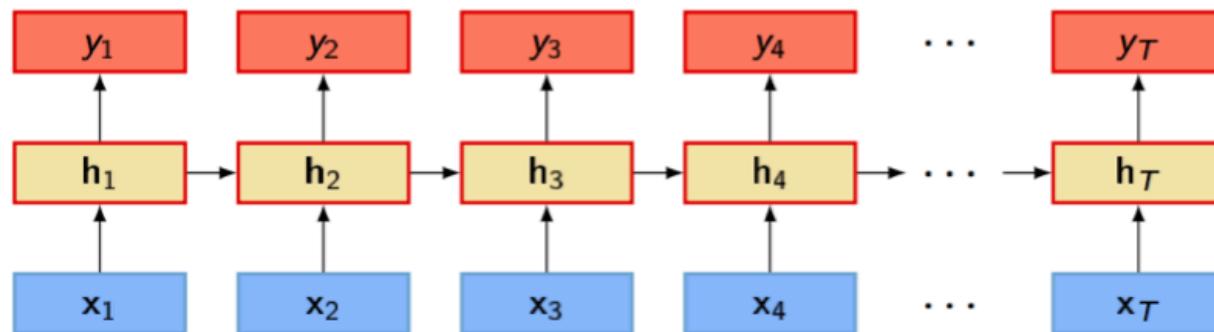
# Simple Recurrent Network

- The recurrence function

$$h_t = f(Uh_{t-1} + Wx_t + b)$$

where  $f$  is non-linear function (for instance, *sigmoid* or *tanh*). Weight  $U \in \mathbb{R}^{d \times d}$ , weight  $W \in \mathbb{R}^{d \times k}$ , hidden state  $h \in \mathbb{R}^d$ , and input  $x \in \mathbb{R}^k$

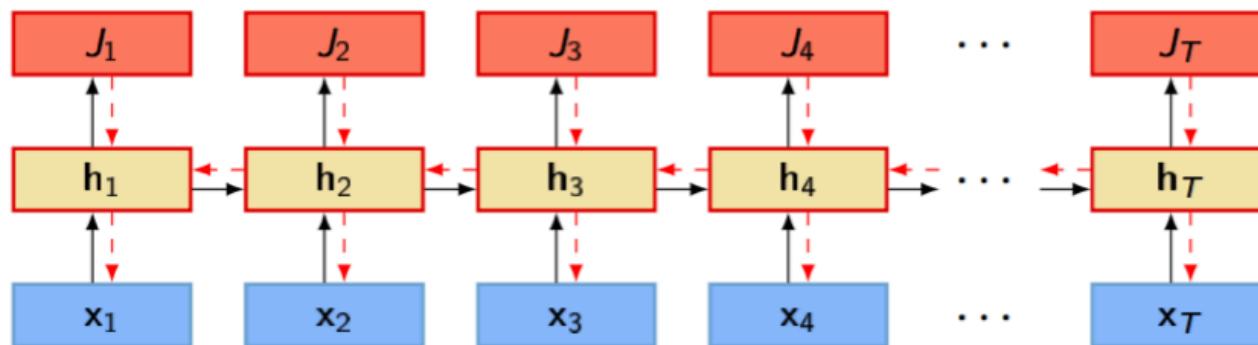
- A typical RNN structure can be illustrated as follows:



# Backpropagation Through Time, BPTT

- Suppose loss at time  $t$  is  $J_t$ , then the total loss is  $\sum_{t=1}^T J_t$ . The gradient of  $J$  is

$$\frac{\partial J}{\partial U} = \sum_{t=1}^T \frac{\partial J_t}{\partial U} = \sum_{t=1}^T \frac{\partial h_t}{\partial U} \frac{\partial J_t}{\partial h_t}$$



# Gradient of RNN

$$\frac{\partial J}{\partial U} = \sum_{t=1}^T \sum_{k=1}^t \frac{\partial h_k}{\partial U} \frac{\partial h_t}{\partial h_k} \frac{\partial y_t}{\partial h_t} \frac{\partial J_t}{\partial y_t} \quad (4)$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^t \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=k+1}^t \text{diag}[f'(h_{i-1})] * U \quad (5)$$

$$\frac{\partial J}{\partial U} = \sum_{t=1}^T \sum_{k=1}^t \frac{\partial h_k}{\partial U} \left( \prod_{i=k+1}^t \text{diag}[f'(h_{i-1})] * U \right) \frac{\partial y_t}{\partial h_t} \frac{\partial J_t}{\partial y_t} \quad (6)$$

# Matrix Calculus

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} & \frac{\partial y}{\partial x_2} & \cdots & \frac{\partial y}{\partial x_n} \end{bmatrix}$$

Figure:  $y = f(x_1, x_2, \dots, x_n)$

$$\frac{\partial \mathbf{Y}}{\partial x} = \begin{bmatrix} \frac{\partial y_{11}}{\partial x} & \frac{\partial y_{12}}{\partial x} & \cdots & \frac{\partial y_{1n}}{\partial x} \\ \frac{\partial y_{21}}{\partial x} & \frac{\partial y_{22}}{\partial x} & \cdots & \frac{\partial y_{2n}}{\partial x} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_{m1}}{\partial x} & \frac{\partial y_{m2}}{\partial x} & \cdots & \frac{\partial y_{mn}}{\partial x} \end{bmatrix}$$

Figure:  $Y$  is a matrix.

# Matrix Calculus

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix} \quad \frac{\partial \mathbf{F}}{\partial \mathbf{X}} = \begin{bmatrix} \frac{\partial \mathbf{F}}{\partial X_{1,1}} & \cdots & \frac{\partial \mathbf{F}}{\partial X_{1,n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \mathbf{F}}{\partial X_{m,1}} & \cdots & \frac{\partial \mathbf{F}}{\partial X_{m,n}} \end{bmatrix}$$

Figure:  $F, X$  are matrices. □ ▶ ↻

Figure:  $Y, X$  are column vectors.

# Derive the Results

$$\vec{h}_t = f(\vec{U} * \vec{h}_{t-1} + \vec{W} * \vec{X}_t) \quad (7)$$

where weight  $\vec{U} \in \mathcal{R}^{d \times d}$ , weight  $\vec{W} \in \mathcal{R}^{d \times k}$ , hidden state  $\vec{h} \in \mathcal{R}^d$ , and input  $\vec{X} \in \mathcal{R}^k$ .

$$\begin{bmatrix} \vec{h}_t[1] \\ \vec{h}_t[2] \\ \vdots \\ \vec{h}_t[d] \end{bmatrix} = \begin{bmatrix} f(\vec{U}_{1*} \cdot \vec{h}_{t-1} + \vec{W}_{1*} \cdot \vec{X}_t) \\ f(\vec{U}_{2*} \cdot \vec{h}_{t-1} + \vec{W}_{2*} \cdot \vec{X}_t) \\ \vdots \\ f(\vec{U}_{d*} \cdot \vec{h}_{t-1} + \vec{W}_{d*} \cdot \vec{X}_t) \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_d \end{bmatrix} \quad (8)$$

where each  $f_i$  is a scalar since  $\vec{U}_{i*} \cdot \vec{h}_{t-1} + \vec{W}_{i*} \cdot \vec{X}_t$  is a scalar.

行向量\*列向量

$$\frac{\partial \vec{h}_t}{\partial \vec{h}_{t-1}} = \begin{bmatrix} \frac{\partial f_1}{\partial \vec{h}_{t-1}} \\ \frac{\partial f_2}{\partial \vec{h}_{t-1}} \\ \vdots \\ \frac{\partial f_d}{\partial \vec{h}_{t-1}} \end{bmatrix} = \begin{bmatrix} \vec{U}_{1*} \cdot f'(\vec{h}_{t-1})[1] \\ \vec{U}_{2*} \cdot f'(\vec{h}_{t-1})[2] \\ \vdots \\ \vec{U}_{d*} \cdot f'(\vec{h}_{t-1})[d] \end{bmatrix} \quad (9)$$

where each  $f'(\vec{h}_{t-1})[i]$  is a scalar,  $f'(\vec{h}_{t-1})$  is a column vector, and  $\vec{U}_{i*}$  is a row vector.

$$\frac{\partial \vec{h}_t}{\partial \vec{h}_{t-1}} = \begin{bmatrix} f'(\vec{h}_{t-1})[1] & & & & \\ & f'(\vec{h}_{t-1})[2] & & & \\ & & \ddots & & \\ & & & f'(\vec{h}_{t-1})[d] & \end{bmatrix} \cdot \begin{bmatrix} \vec{U}_{1*} \\ \vec{U}_{2*} \\ \vdots \\ \vec{U}_{d*} \end{bmatrix} \quad (10)$$

$$= \text{diag}[f'(\vec{h}_{t-1})] \cdot \vec{U} \quad (11)$$

# Gradient Vanishing/Explosion

Define  $\gamma = \|\text{diag}[f'(h_{i-1})] * U\|$ ,

- ▶ Exploding Gradient Problem: When  $\gamma > 1$  and  $t - k \rightarrow \infty$ ,  
 $\gamma^{t-k} \rightarrow \infty$ .
- ▶ Vanishing Gradient Problem: When  $\gamma < 1$  and  $t - k \rightarrow \infty$ ,  
 $\gamma^{t-k} \rightarrow 0$ .

There are various ways to solve Long-Term Dependency problem.

# Vanishing/Exploding Gradient Problem

- Consider a **linear** recurrent net with zero inputs

$$h_t = W_{hh} h_{t-1} = W_{hh}^t h_0$$

- Singular value of  $W < 1 \Rightarrow$  gradient Vanishes
- Singular value of  $W > 1 \Rightarrow$  gradient Explodes
- “It is **sufficient** for the largest eigenvalue  $\lambda_1$  of the recurrent weight matrix to be smaller than 1 for long term components to vanish (as  $t \rightarrow \infty$ ) and **necessary** for it to be larger than 1 for gradients to explode.”

# Vanishing/Exploding Gradient Problem

- Gradient norm clipping (Mikolov thesis 2012; Pascanu, Mikolov, Bengio, ICML 2013)

$$\hat{g} \leftarrow \frac{\partial \mathcal{J}}{\partial \theta}$$

IF  $\|\hat{g}\| > \text{Threshold}$

THEN  $\hat{g} \leftarrow \frac{\text{Threshold}}{\|\hat{g}\|} * \hat{g}$

- Gradient propagation regularizer (avoid vanishing gradient)

# Connections to MLP?

- Why is RNN a deep structure?

# Connections to MLP?

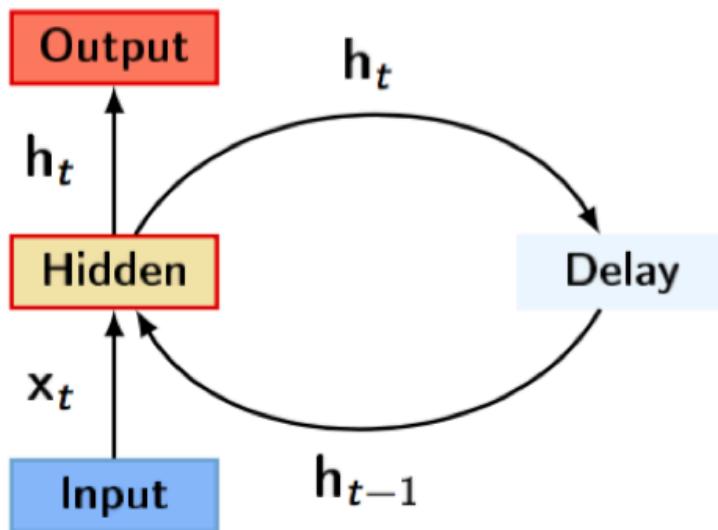


Figure: Traditional View.

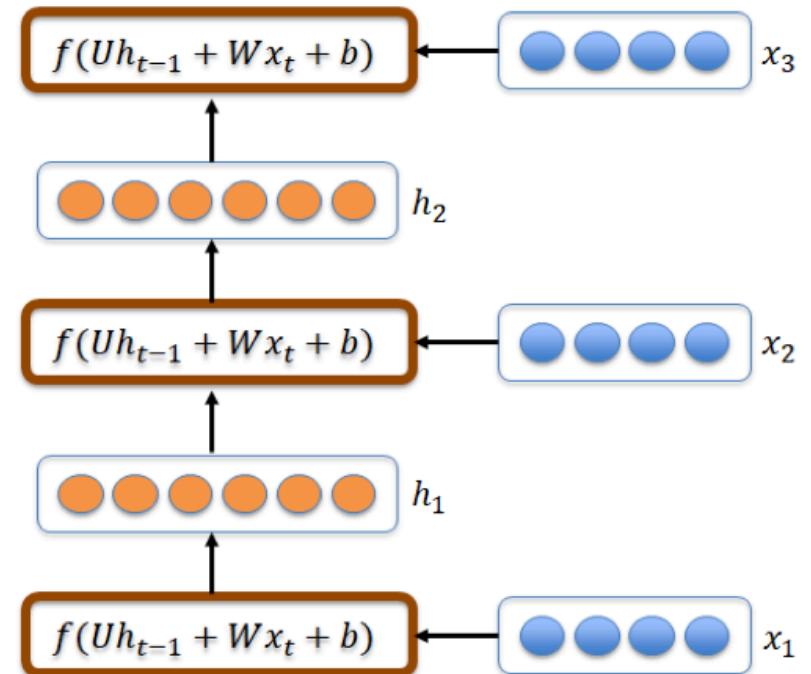


Figure: Deep View.

# Long Short-Term Memory (LSTM)

Key difference to RNN: a memory cell  $c$  which is controlled by three gates:

- Input gate  $i$ :

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$

- Output gate  $o$ :

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$

- Forget gate  $f$ :

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$

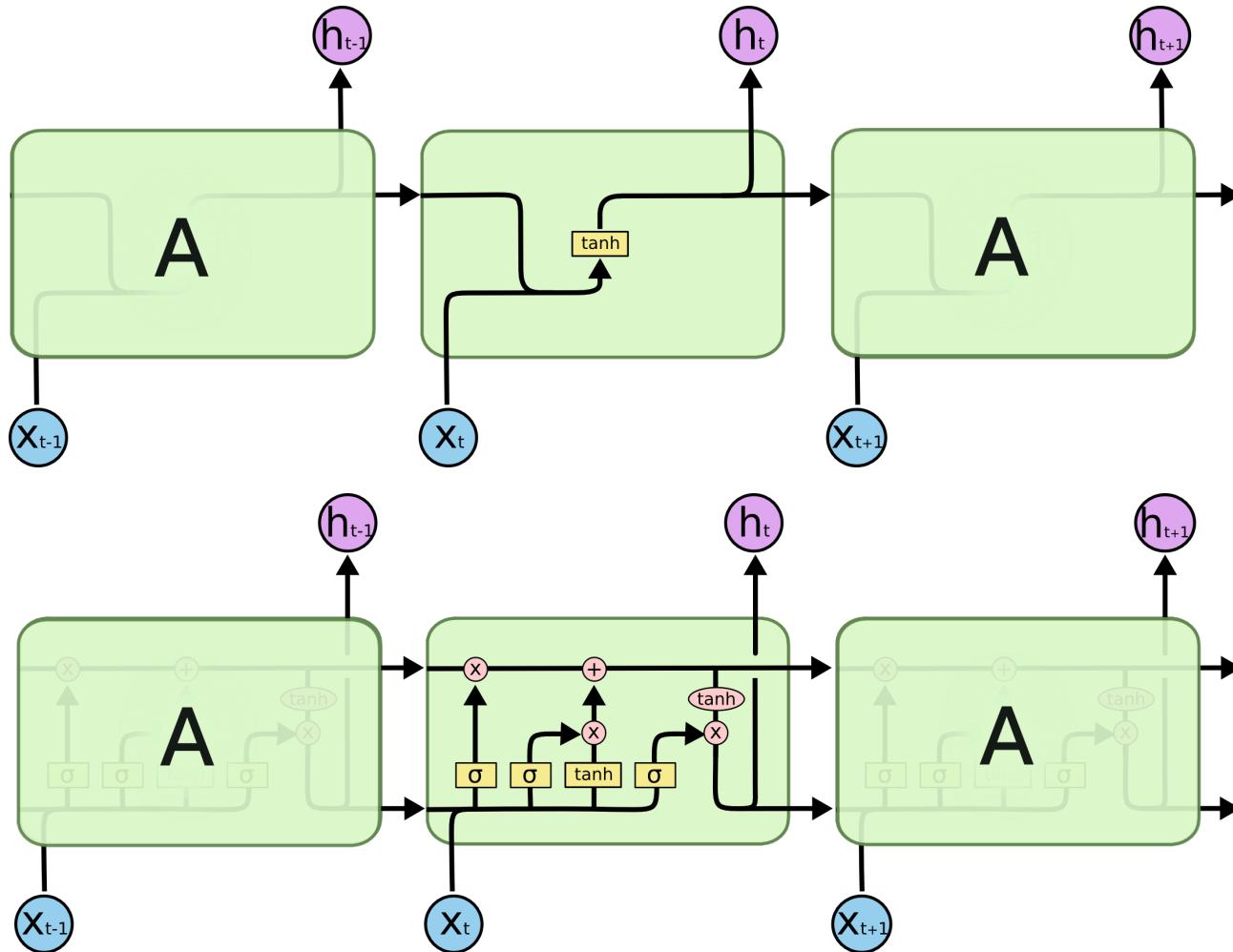
- Update of memory cell and hidden state:

$$\tilde{C}_t = \tanh(W_C x_t + U_C h_{t-1} + b_C)$$

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

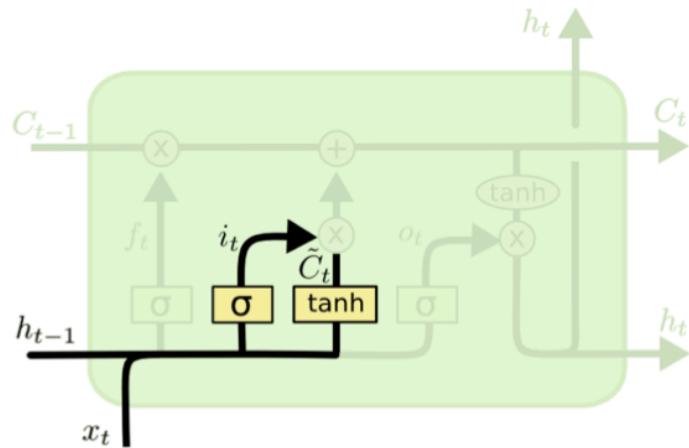
$$h_t = o_t \otimes \tanh(C_t)$$

# Standard RNN vs. LSTM



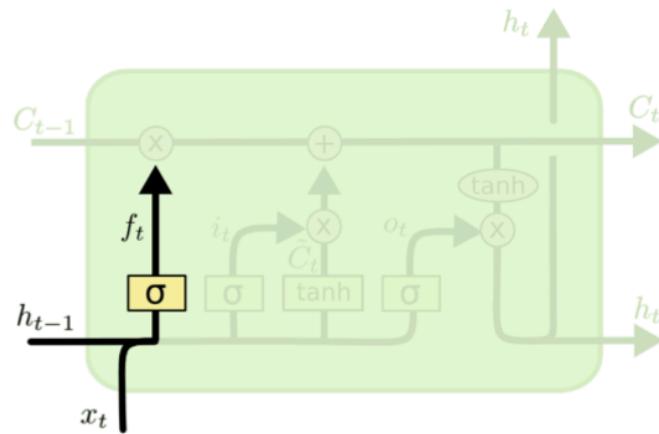
The pictures are from <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

# LSTM: Input and Forget Gates



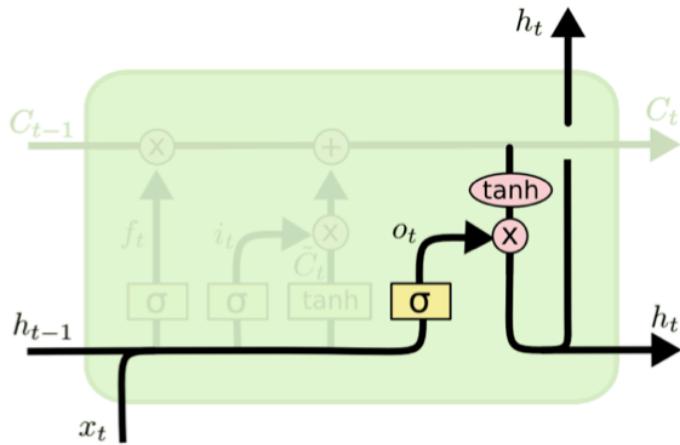
$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$

$$\tilde{C}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c)$$



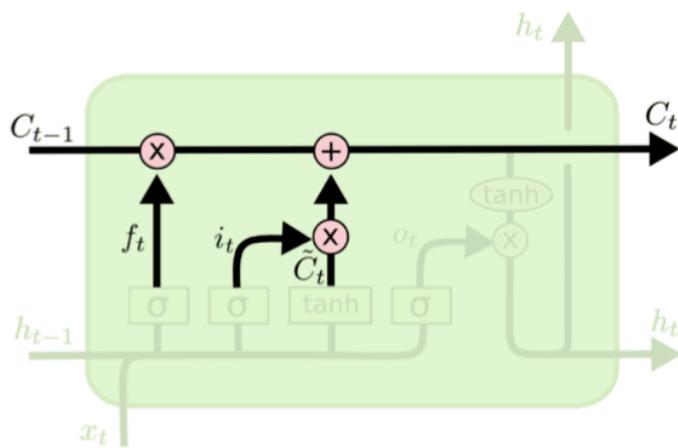
$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$

# LSTM: Output Gates and State Update



$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$

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



$$C_t = f_t \otimes C_{t-1} + i_t \otimes \widetilde{C}_t$$

# Empirical Feasible Length of LSTM

RNN

100

LSTM

500

# Gated Recurrent Unit (GRU)

Instead of using three gates, only two gates are employed:

- Update gate  $z$ :

$$z_t = \sigma(W_z x_t + U_z h_{t-1})$$

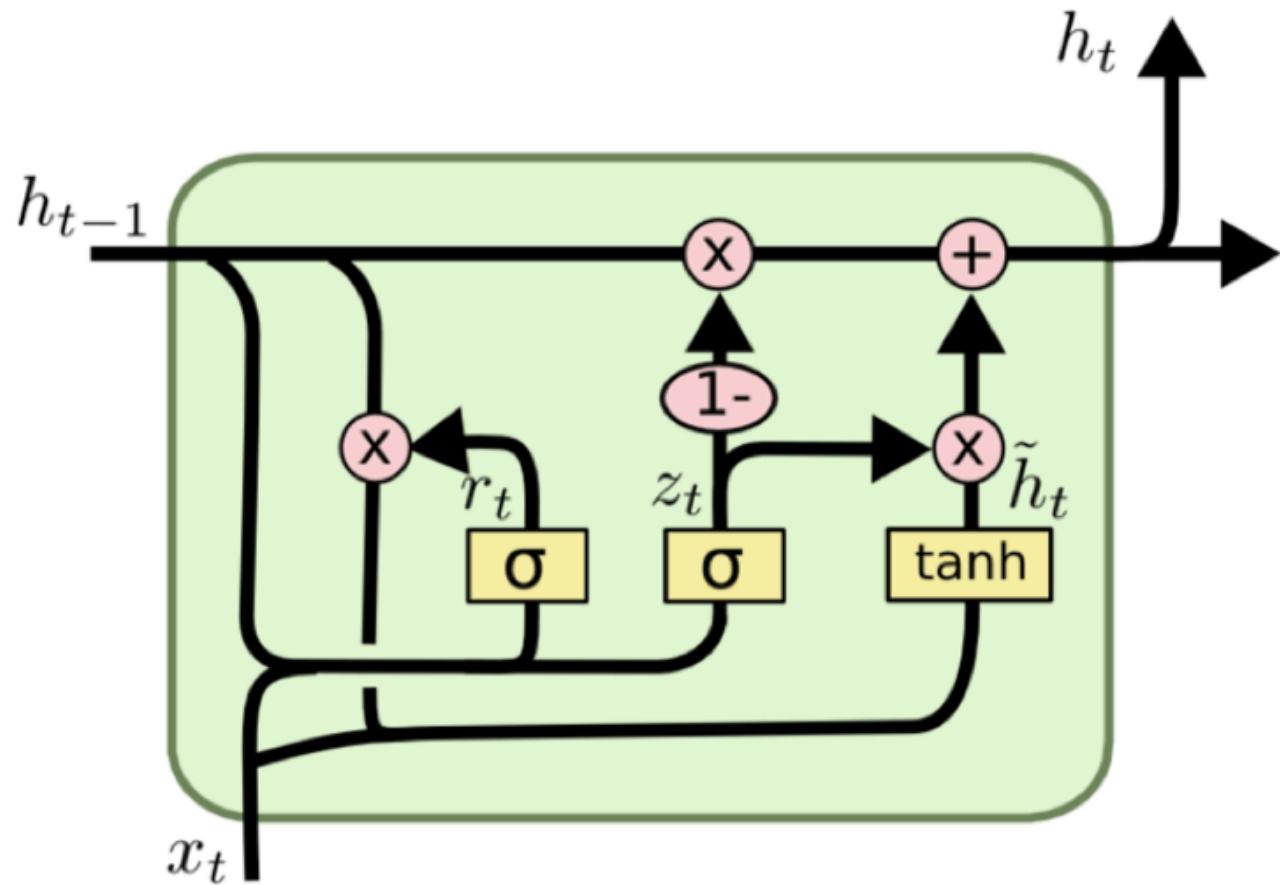
- Reset gate  $r$ :

$$r_t = \sigma(W_r x_t + U_r h_{t-1})$$

- State update:

$$\begin{aligned}\tilde{h}_t &= \tanh(W_C x_t + U(r_t \otimes h_{t-1})) \\ h_t &= (1 - z_t) \otimes h_{t-1} + z_t \otimes \tilde{h}_t\end{aligned}$$

# GRU Architecture



# Empirical Feasible Length of GRU

RNN

100

LSTM

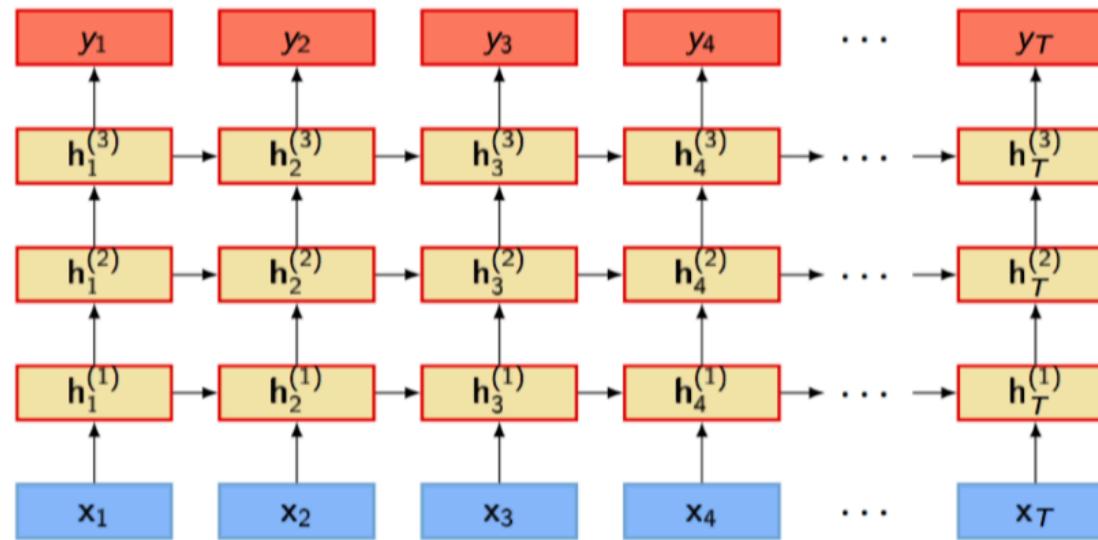
500

GRU

1000

# Stacked RNN

- Stack multiple RNN/LSTM in vertical direction



- More common is seen that raw input ( $x_i$ ) is connected to all layers as input.

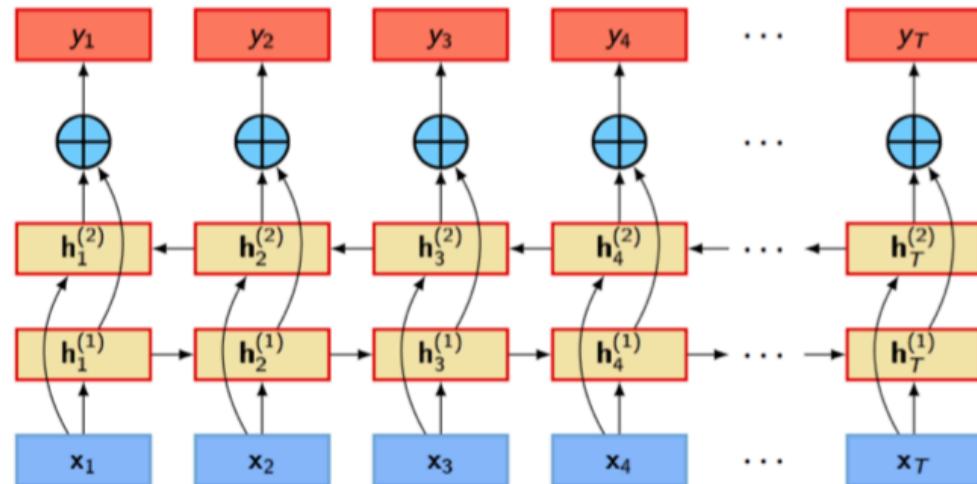
# Bidirectional RNN

- Modeling the prefix context and suffix context at the same time.

$$h_t^{(1)} = f(U^{(1)} h_{t-1}^{(1)} + W^{(1)} x_t + b^{(1)})$$

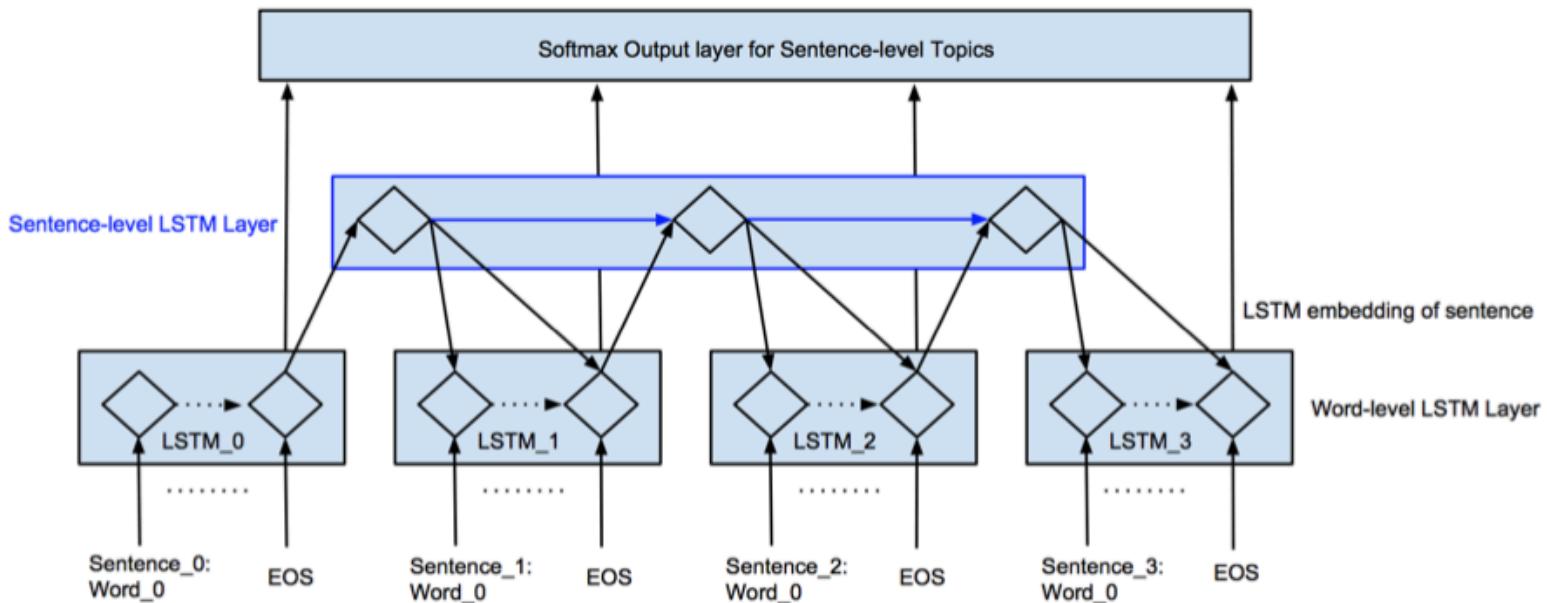
$$h_t^{(2)} = f(U^{(2)} h_{t+1}^{(2)} + W^{(2)} x_t + b^{(2)})$$

$$h_t = h_t^{(1)} \oplus h_t^{(2)}$$



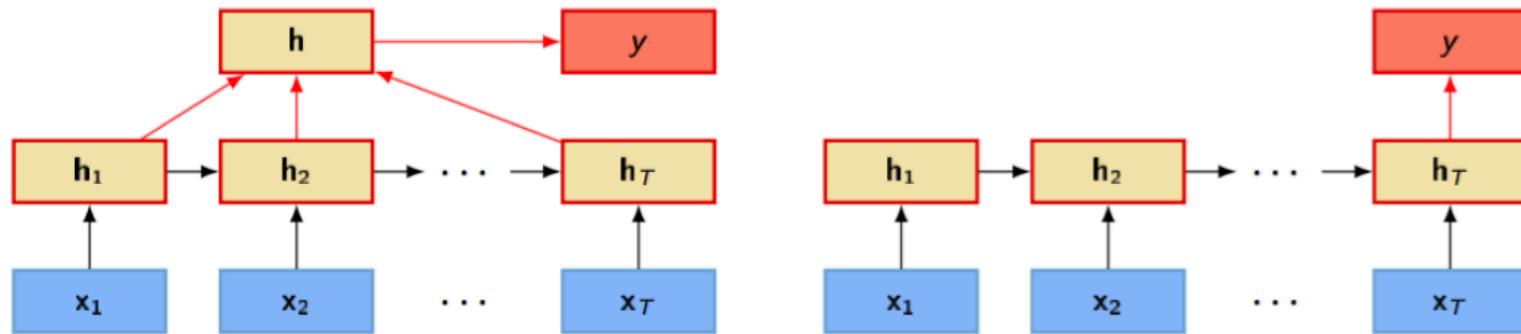
# Hierarchical LSTM

- Hierarchical LSTM (low-level LSTMs are taken as input to the high-level LSTM)



# Application: Classification

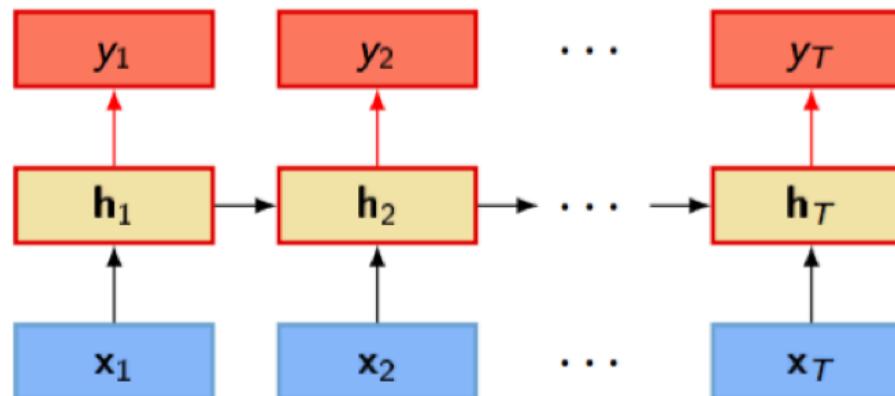
- Applicable to many classification problems:
  - Text Classification
  - Sentiment Classification



- The left: mean (can be enhanced with attention mechanisms)
- The right: last

# Application of RNN: Sequence Labeling

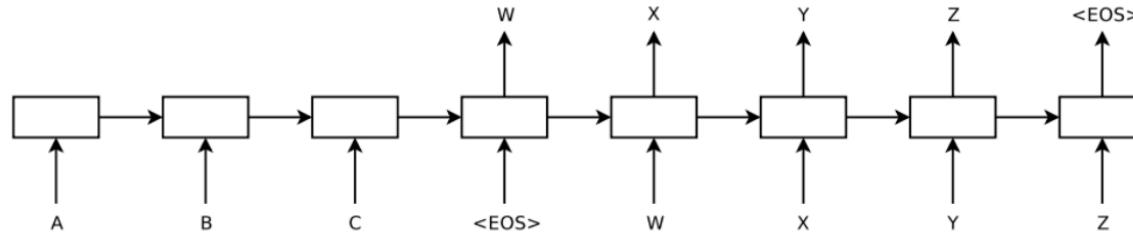
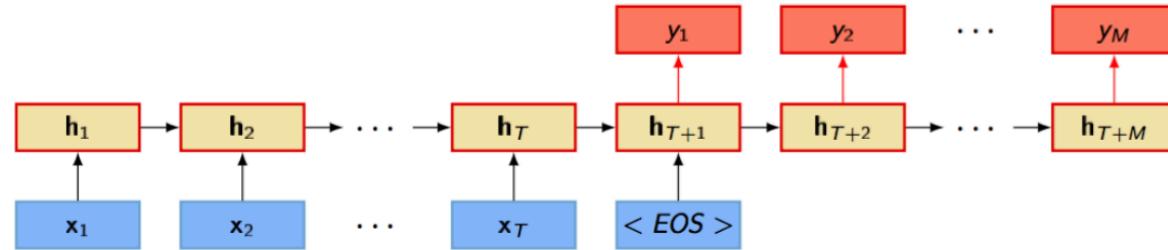
- Sequence Labeling, such as Chinese word segmentation, part-of-speech tagging, semantic role labeling



- Linguistic Tools:
  - Natural Language Toolkit (NLTK)
  - Stanford CoreNLP
  - THU Lexical Analyzer for Chinese (THULAC)

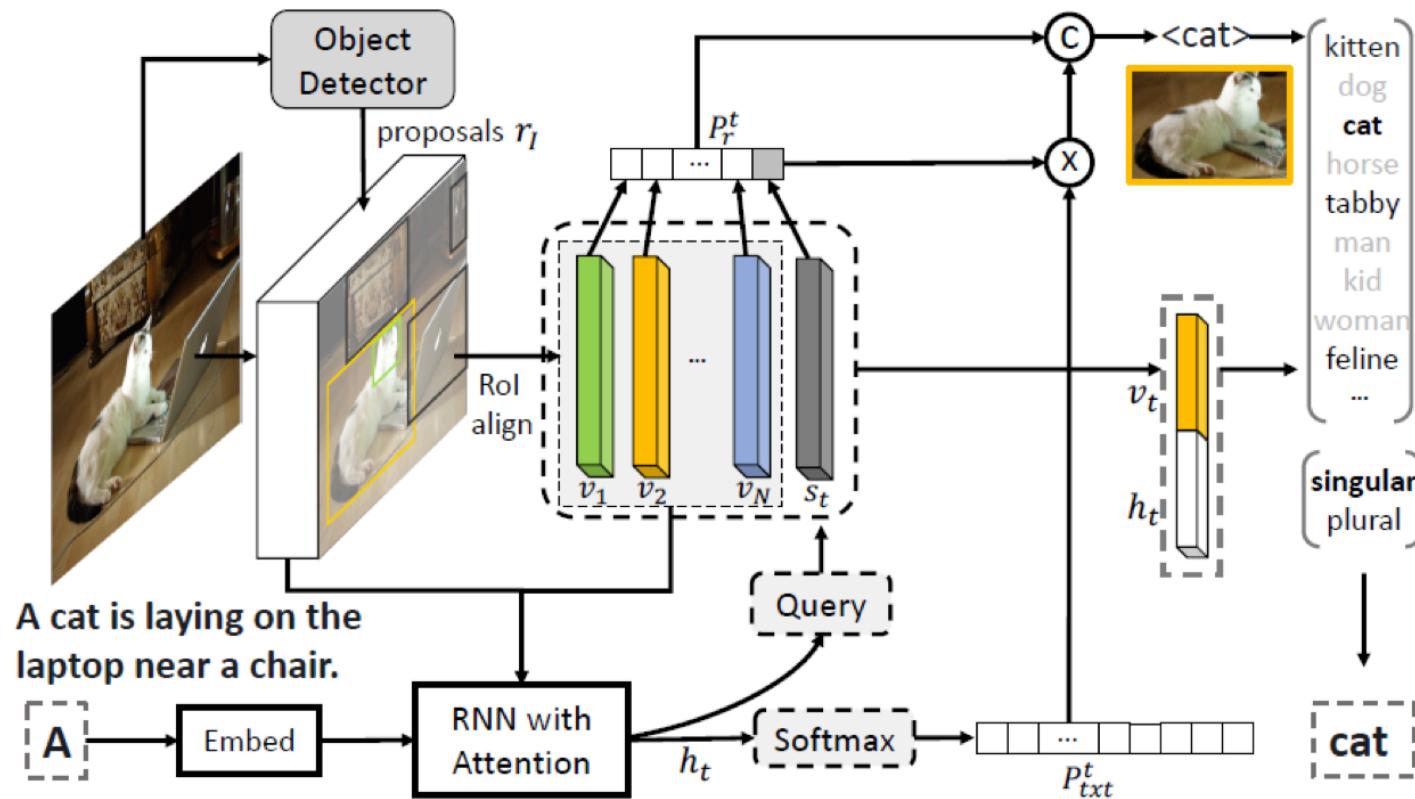
# Application: Sequence to Sequence Learning

- Machine Translation
- Sequence to sequence generation (for dialogue, question answering, etc.)
- Image captioning: from image to text



# Application: Sequence to Sequence Learning

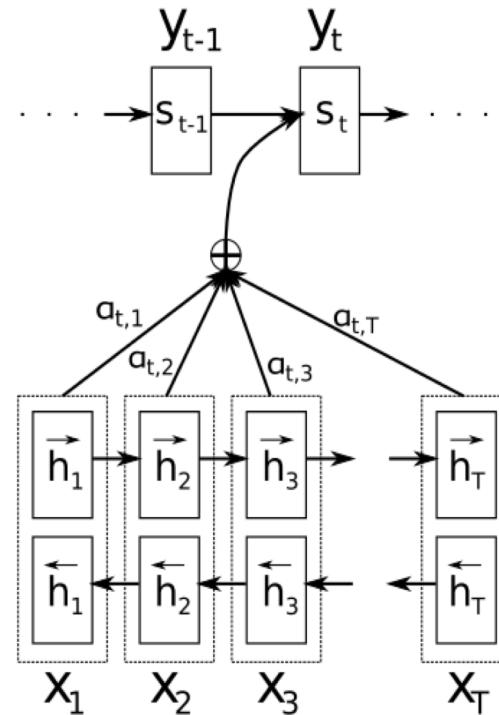
- Example: Image Caption



# What is Attention?

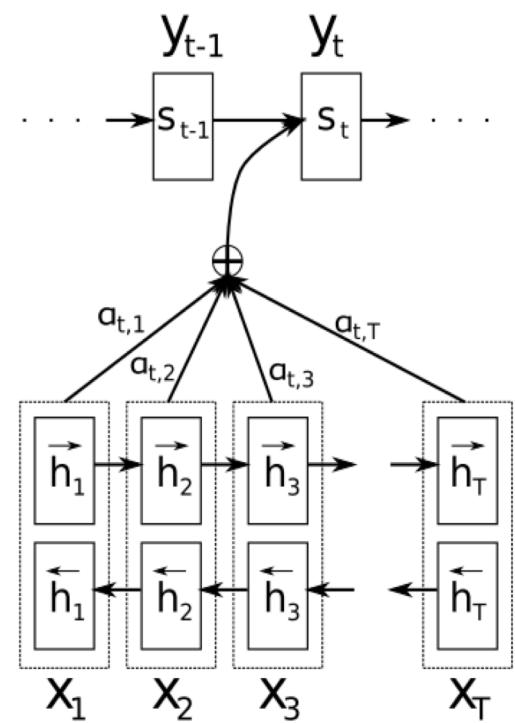
- Differentiate entities by its importance
  - spatial attention is related to location
  - temporal attention is related to causality

$$\sum \alpha_i x_i$$
$$0 \leq \alpha_i \leq 1$$



# What is Attention?

- To decode state  $s_t$ , attend to encoder states  $h_{1 \sim j}$ 
  - Similarity function:  $e_{tj} = a(s_{t-1}, h_j)$
  - Softmax:  $\alpha_{tj} = \frac{\exp(e_{tj})}{\sum_{k=1}^{T_x} \exp(e_{tk})}$
  - Context:  $c_t = \sum_{j=1}^{T_x} \alpha_{tj} h_j$
  - Merge:  $s_t = f(s_{t-1}, y_{t-1}, c_t)$



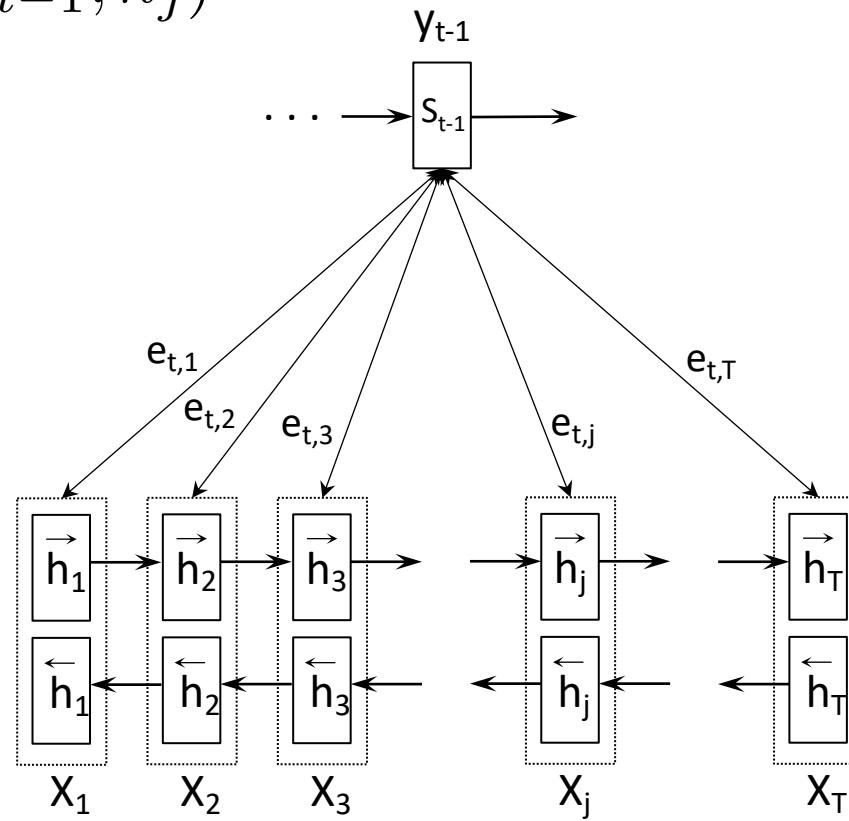
# Attention mechanism

One step decoding:

$$e_{tj} = a(s_{t-1}, h_j)$$

Decoder

Encoder

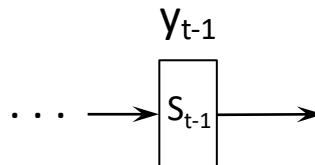


# Attention mechanism

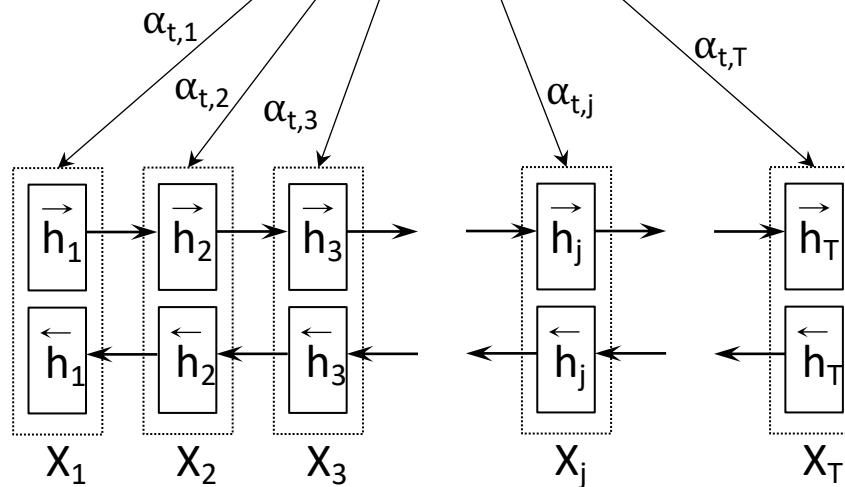
One step decoding:

$$\alpha_{tj} = \frac{\exp(e_{tj})}{\sum_{k=1}^{T_x} \exp(e_{tk})}$$

Decoder



Encoder

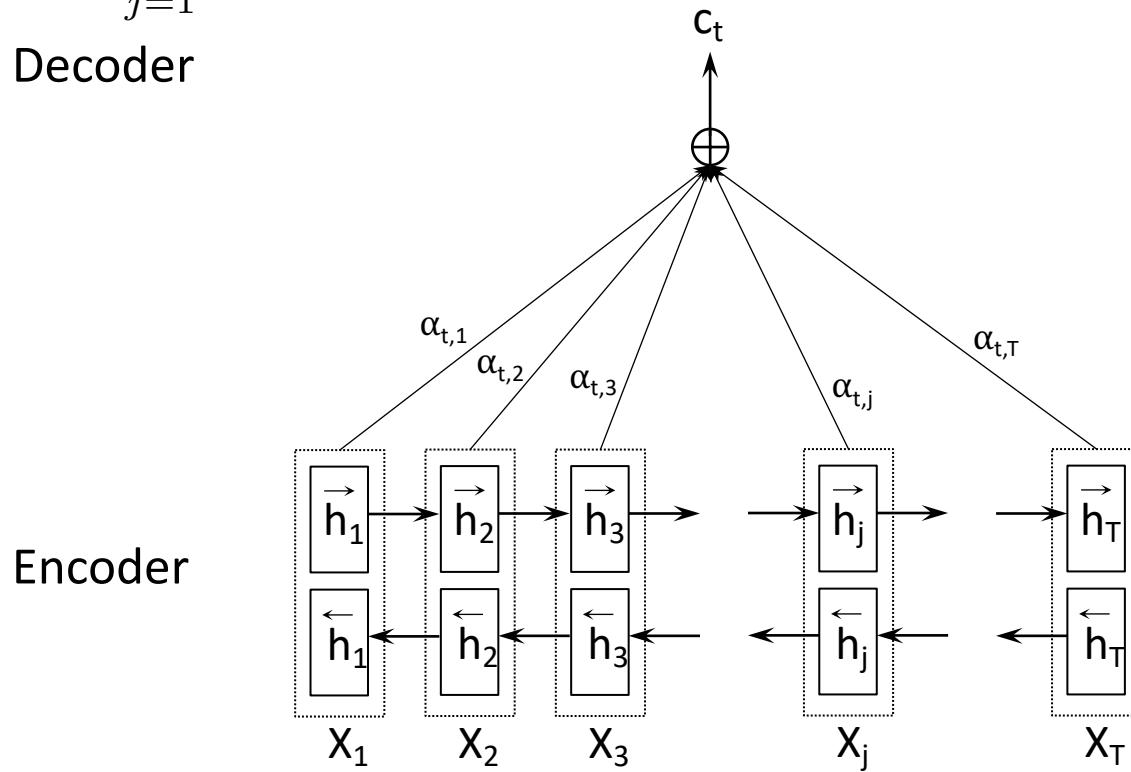


# Attention mechanism

One step decoding:

$$c_t = \sum_{j=1}^{T_x} \alpha_{tj} h_j$$

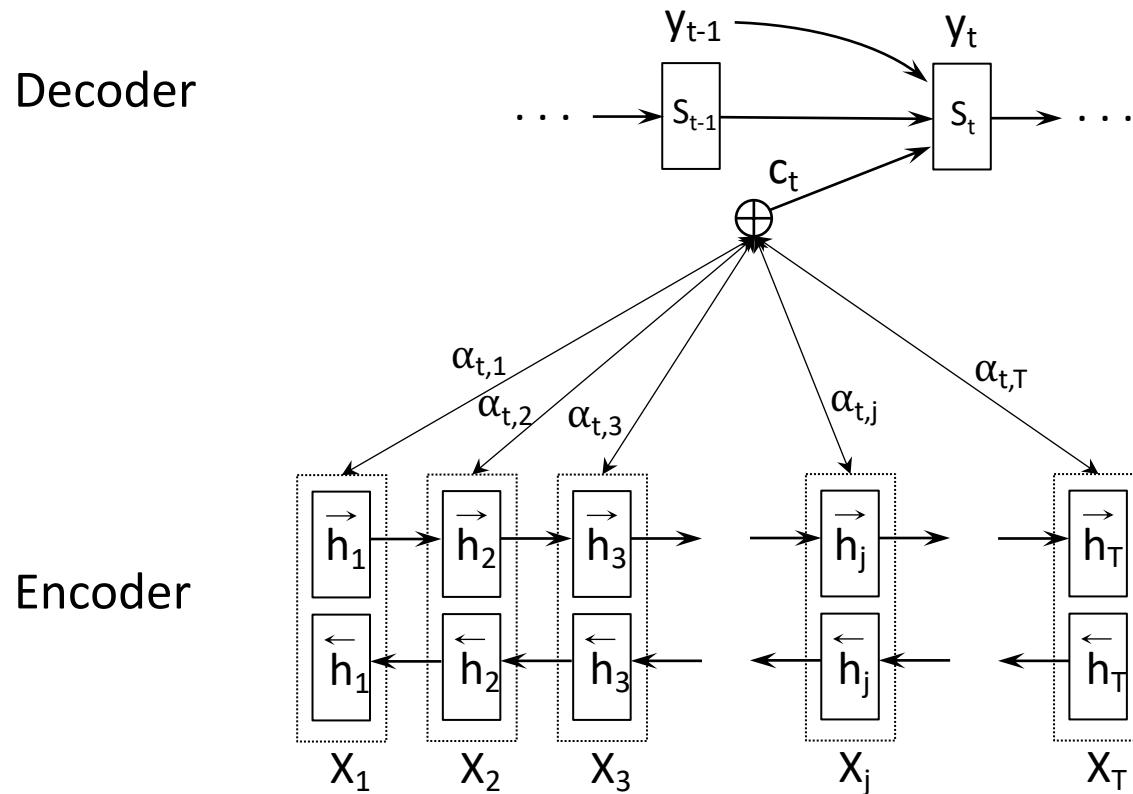
Decoder



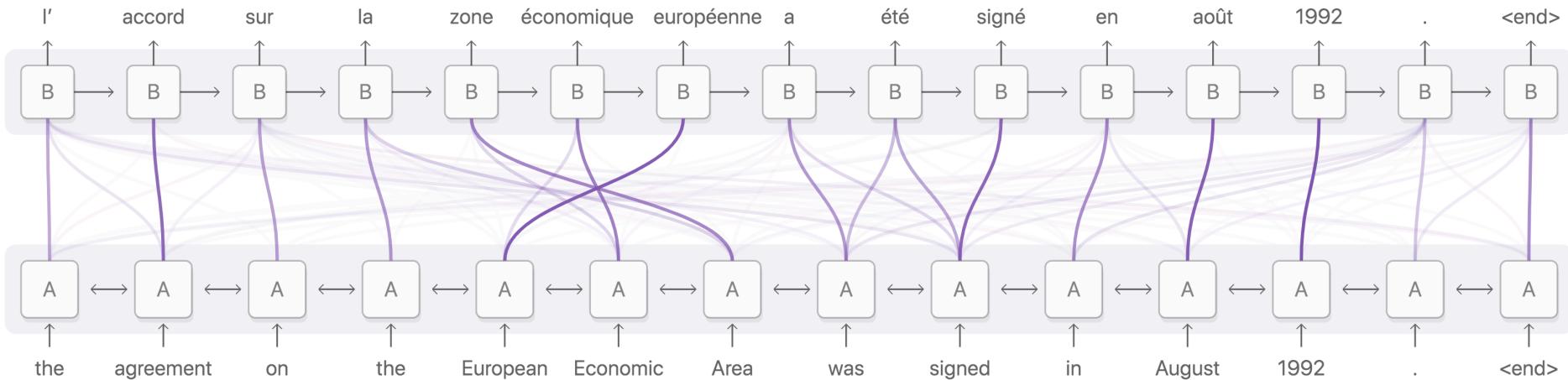
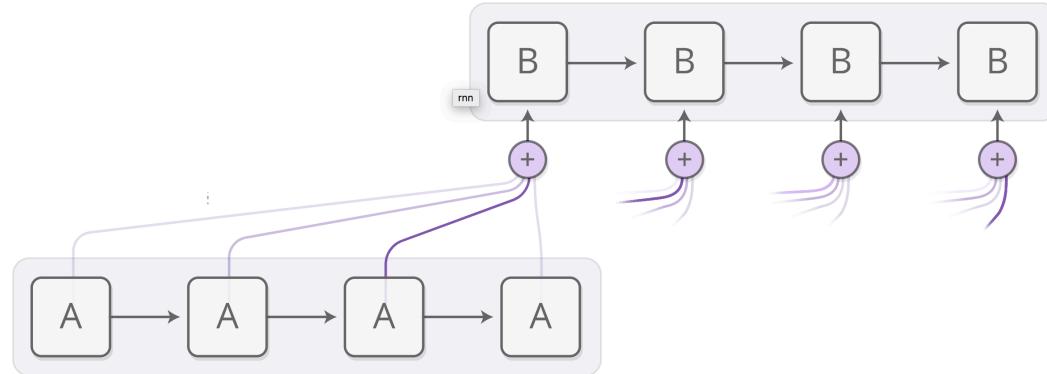
# Attention mechanism

One step decoding:

$$s_t = f(s_{t-1}, y_{t-1}, c_t)$$



# Example: NMT



# Summary

- RNN for sequence modeling
  - RNN
  - LSTM
  - GRU
- Application
  - Classification
  - Sequence Labeling
    - Sequence-to-sequence learning
- Attention