



# Natural Language Processing

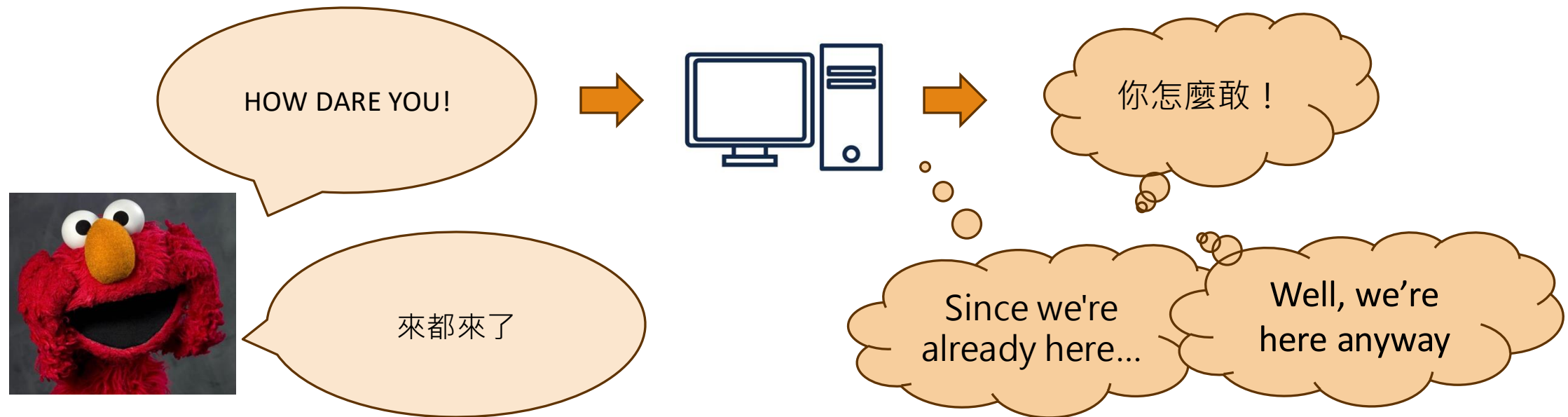
Sequence-to-sequence Models and Attention Mechanisms



# Machine Translation

Machine translation plays a pivotal role in the development of NLP.

- Many advancements in NLP language models were initially motivated by the need to address translation challenges.



# Challenges of Machine Translation

---

Unlike the other tasks like classification, the input and output of machine translation are **in different lengths**.



- ✗ We cannot just add an FFN at the end.
- ✓ The hidden state should be utilized to encode the original sequence and it should be passed to the generation process.

# Sequence to sequence Model

---

Sequence to Sequence (Seq2Seq) model

The core idea of the Seq2Seq model is to map **variable-length input sequences to variable-length output sequences**.

This flexibility enables its wide application across various tasks, such as:

- **Machine Translation**: Translating a sentence from one language to another.
- Text Summarization: Condensing long articles into shorter summaries.
- Dialogue Generation: Generating appropriate responses based on input contexts.



# Sequential Models

---

Sequence models, also known as time series models, are a class of neural network architectures designed to model sequences of data.

- **RNN**
- **LSTM**

Each output of a sequential model depends on the hidden state variable provided by the previous step.

# RNN for Sequence Generation

---

## 1. Input Encoding:

- The input sequence is fed into the RNN one element at a time.

## 2. Hidden State Update:

- The RNN updates its hidden state at each time step based on the current input element and the previous hidden state.

## 3. Output Generation:

- Using the hidden states at each time step, the RNN generates the output sequence one element at a time.

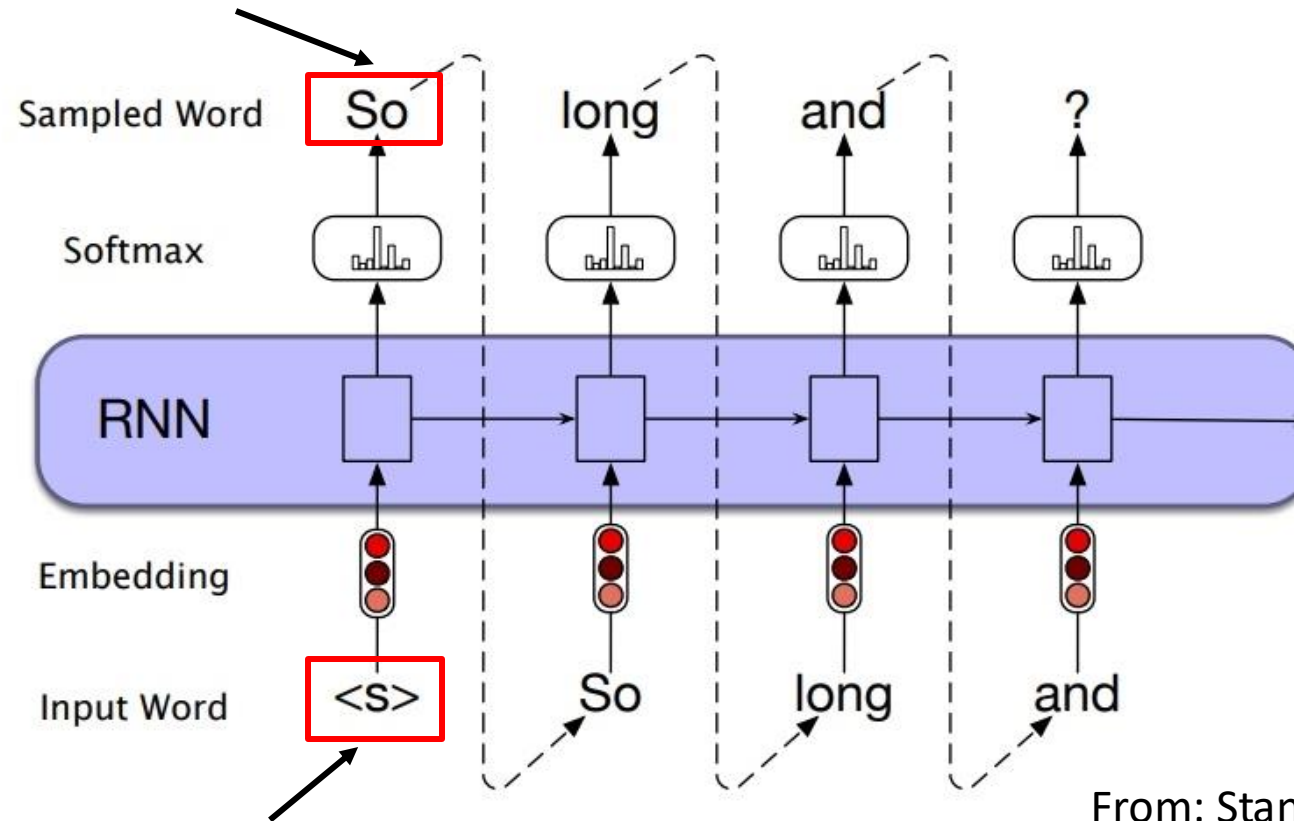
## 4. Iteration:

- Steps 2-3 are iterated until the desired length of the output sequence is reached or until a specific termination condition is met (e.g., generating an end-of-sequence token).



# RNN for Sequence Generation

Predict the next token.



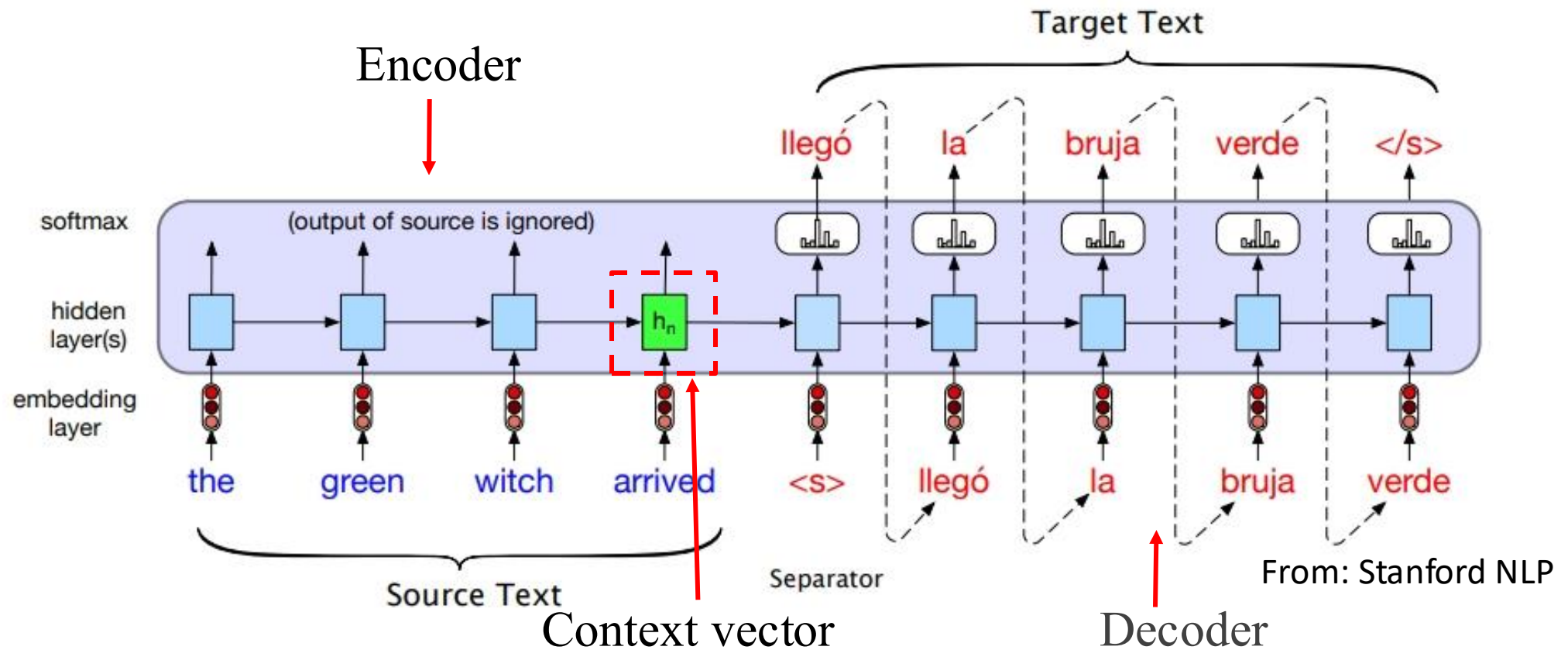
From: Stanford NLP

The sequence generation begin with a special token.



# RNN for Machine Translation

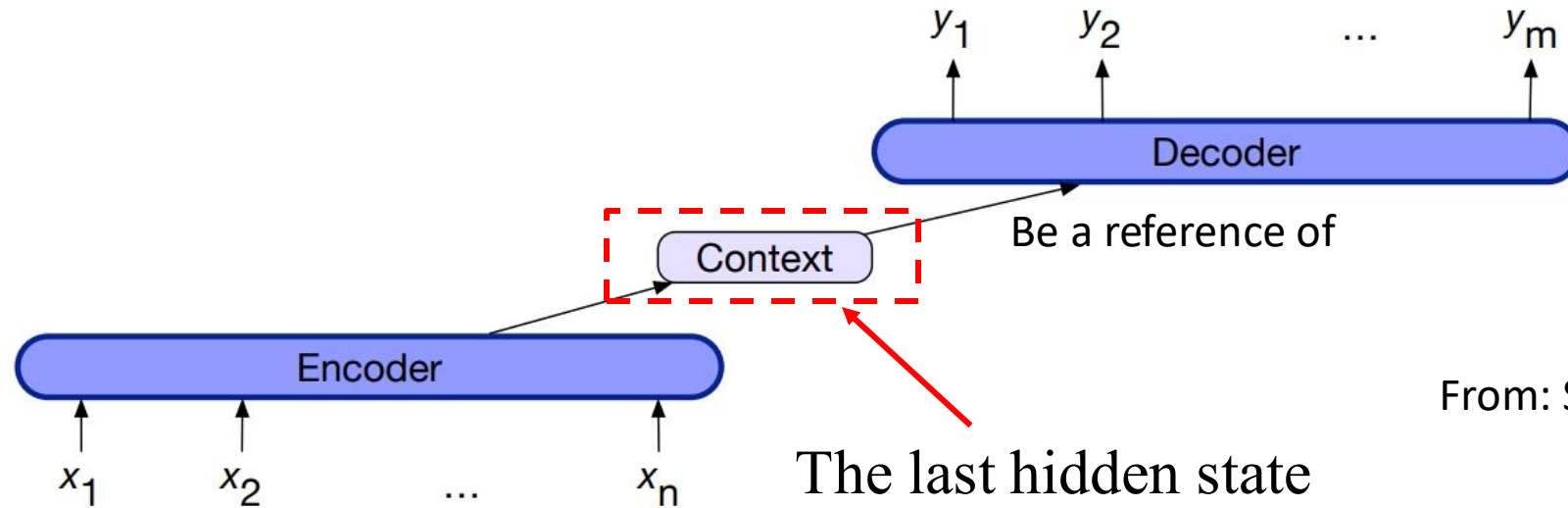
- Machine translation is a classic application of encoder-decoder architecture.





# RNN for Machine Translation

- The context vector is the last hidden state of the encoder and it contains all the information of the input.



From: Stanford NLP

# Encoder-decoder RNN

---

Encoder-decoder networks consist of three conceptual components:

- **An encoder** that accepts an input sequence,  $x_{1:n}$ , and generates a corresponding sequence of *contextualized representations*,  $h_{1:n}$ .
- **A context vector**, which is a function of  $h_{1:n}$ , and conveys the essence of the input to the decoder.
- **A decoder**, which accepts the context vector as input and generates an arbitrary length sequence of hidden states  $h_{1:m}$ , from which a corresponding sequence of output states  $y_{1:m}$ , can be obtained.

# Gradient Vanishing/Exploding Problem of RNN

---

- During backpropagation, gradients are computed with respect to the hidden states through the loss function at each time step.

$$y_t = g(Vh_t)$$

$$h_t = f(Ux_t + Wh_{t-1})$$

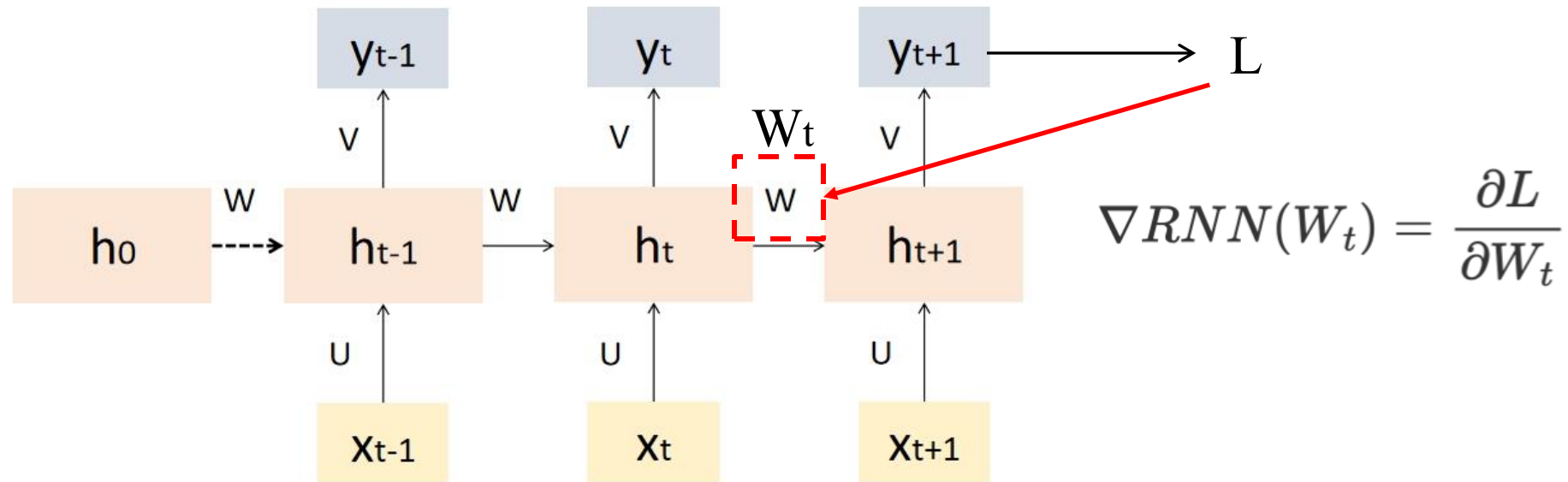
- Compute the gradient (use the parameter  $W$  as an example):

$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L}{\partial h_T} \cdot \boxed{\frac{\partial h_T}{\partial h_{T-1}}} \cdots \frac{\partial h_t}{\partial W}$$

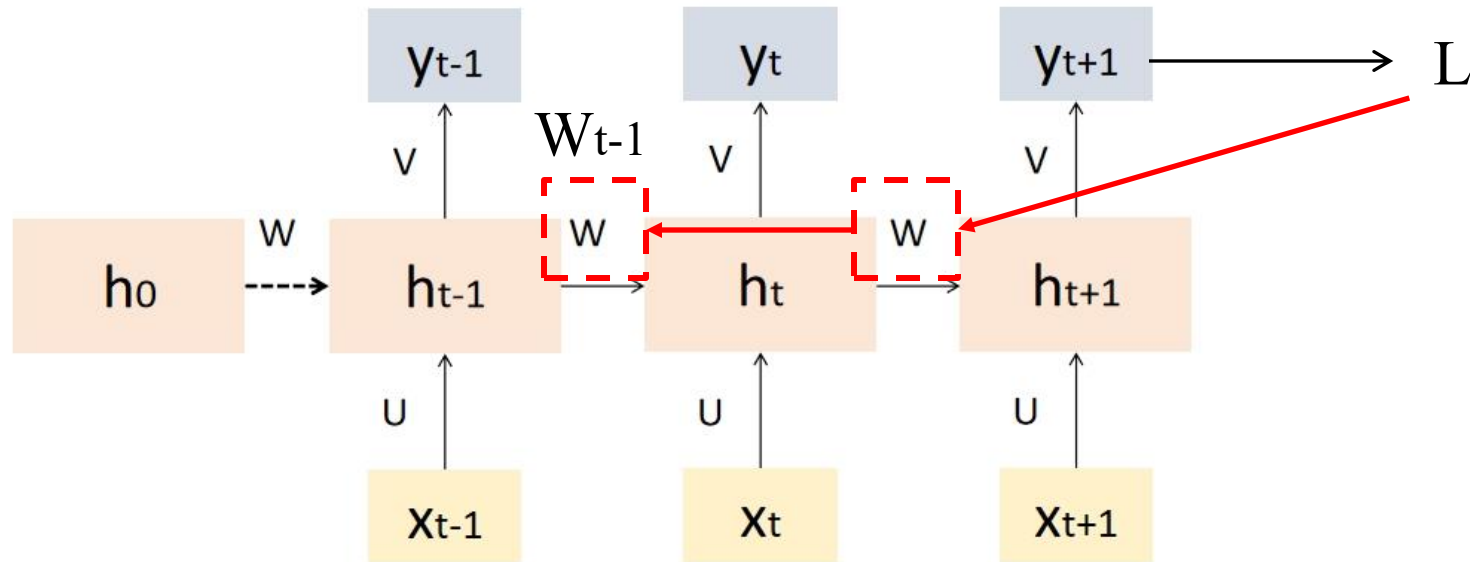
The further back in time an input is, the more factors it needs to be multiplied by.



# Gradient Vanishing/Exploding Problem of RNN

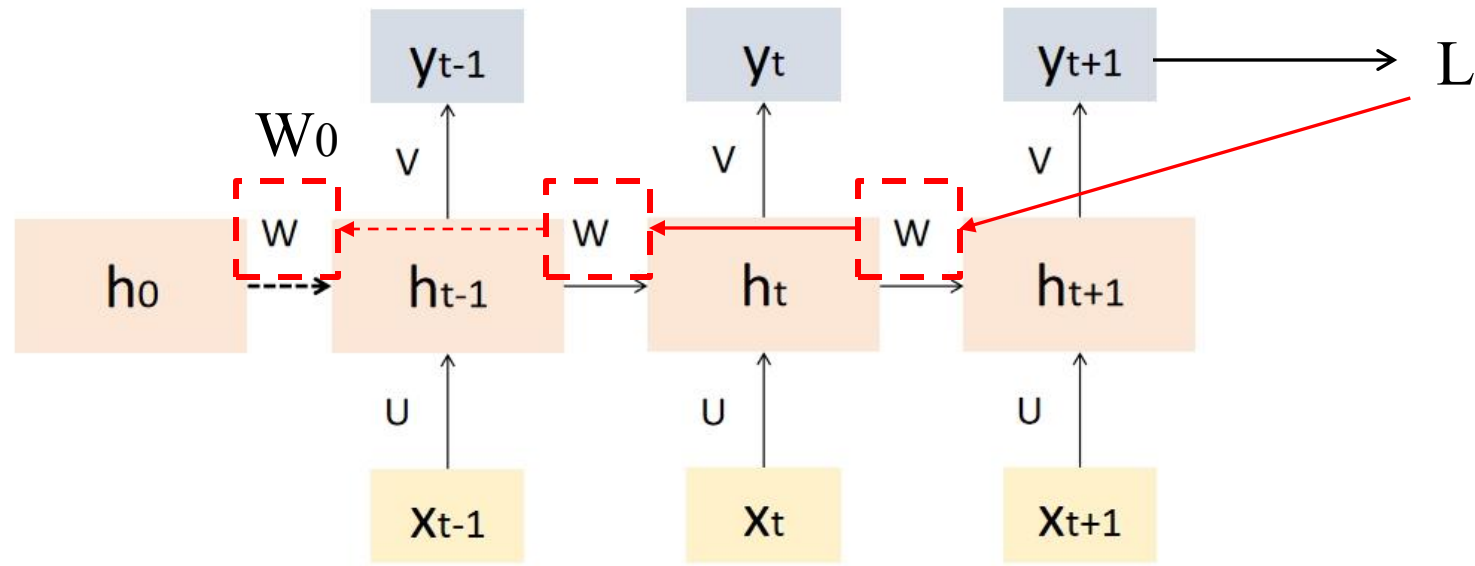


# Gradient Vanishing/Exploding Problem of RNN



$$\nabla RNN(W_{t-1}) = \frac{\partial L}{\partial W_{t-1}} = \frac{\partial L}{\partial W_t} \frac{\partial W_t}{\partial W_{t-1}}$$

# Gradient Vanishing/Exploding Problem of RNN



$$\nabla RNN(W_0) = \frac{\partial L}{\partial W_0} = \underbrace{\frac{\partial L}{\partial W_t} \frac{\partial W_t}{\partial W_{t-1}} \frac{\partial W_{t-1}}{\partial W_{t-2}} \cdots \frac{\partial W_1}{\partial W_0}}_{\text{gradient terms}}$$

If these terms  $< 1$ , the gradient decays exponentially (gradient vanishing)

If these terms  $> 1$ , the gradient grows rapidly (gradient exploding)

# Gradient Vanishing Problem of RNN

---

Gradient vanishing results in :

- **Information Loss:**

As the number of time steps increases, gradient vanishing prevents RNNs from retaining important information from earlier time steps.

- **Training Instability:**

With the disappearance of gradients in early time steps, the update of network parameters becomes slow or even stagnates.

- **Difficulty in Generating Long Sequences:**

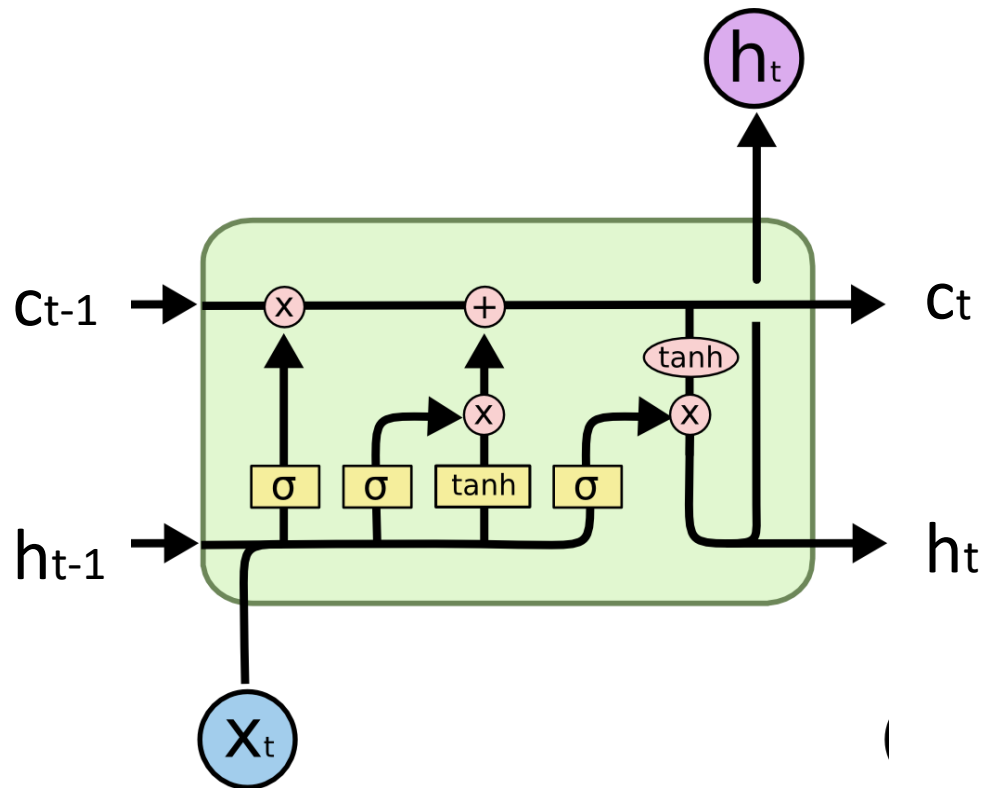
RNNs perform poorly in generating long sequences. Generated sequences may become incoherent or meaningless.





# Long Short-Term Memory

Long Short-Term Memory (LSTM) is proposed to solve Gradient Vanishing problem of RNNs.



$$R_t = \sigma_r(W_r x_t + U_r h_{t-1} + b_r) \quad \text{Forget gate}$$

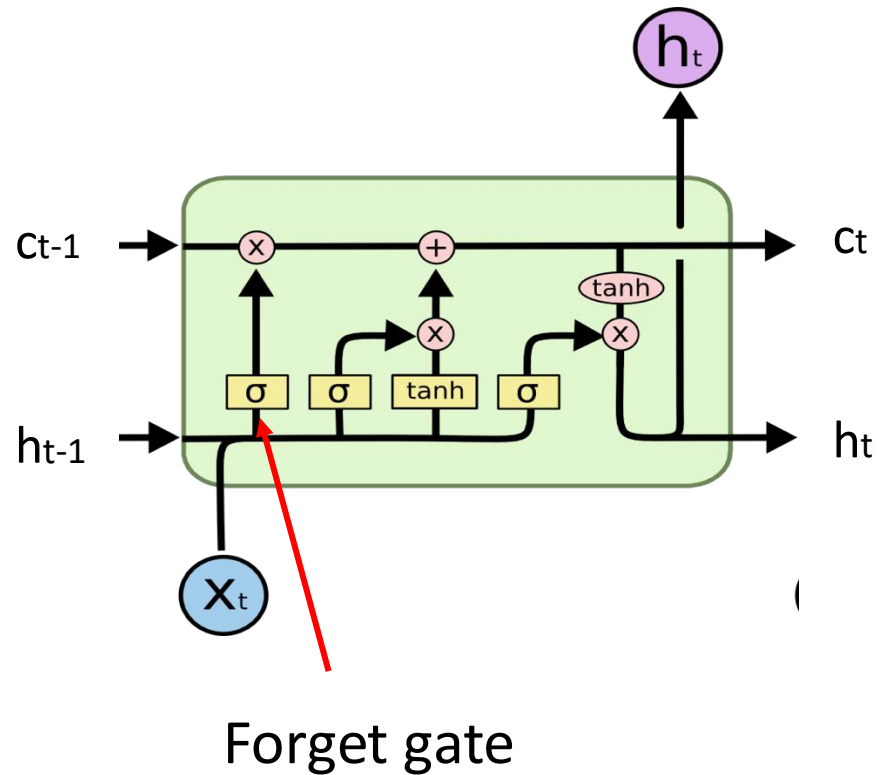
$$K_t = \sigma_r(W_k x_t + U_k h_{t-1} + b_k) \quad \text{Input gate}$$

$$V_t = \sigma_r(W_v x_t + U_v h_{t-1} + b_v) \quad \text{Output gate}$$

$$c_t = R_t \cdot c_{t-1} + K_t \cdot \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$$

$$h_t = V_t \sigma_h(c_t)$$

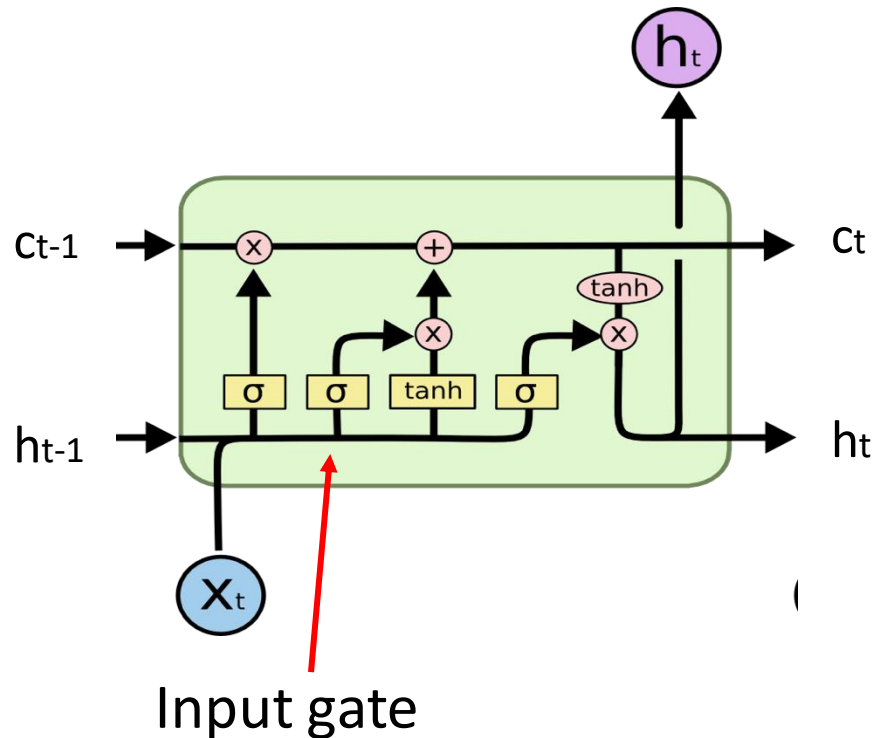
# Long Short-Term Memory



$$R_t = \sigma_r(W_r x_t + U_r h_{t-1} + b_r)$$

- **Forget Gate:** It uses a sigmoid function to determine which information from past memory should be forgotten or discarded.

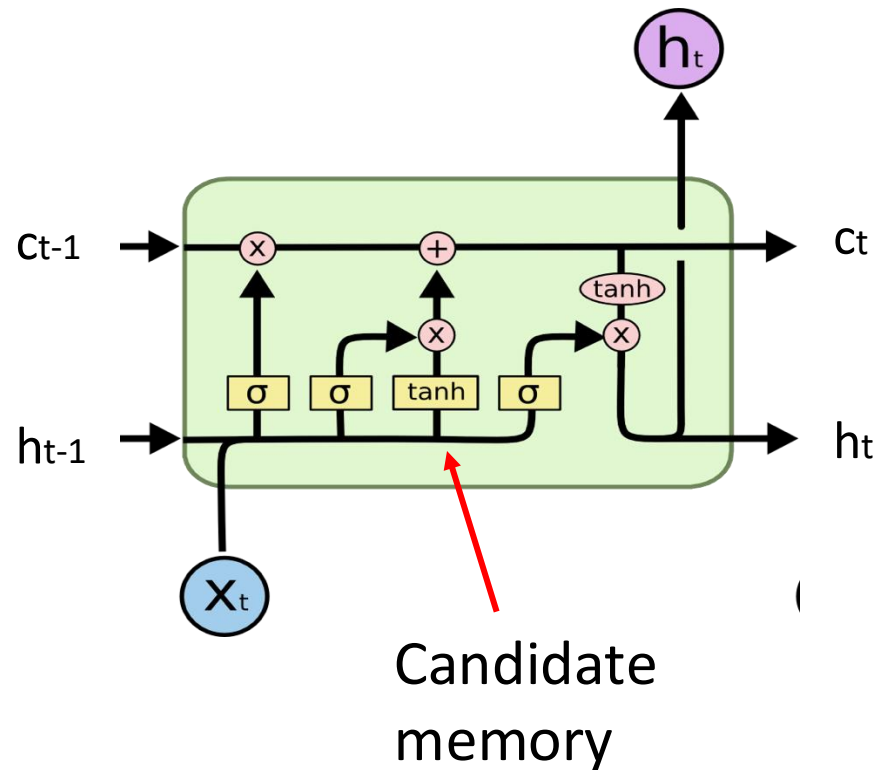
# Long Short-Term Memory



$$K_t = \sigma_r(W_k x_t + U_k h_{t-1} + b_k)$$

- **Input Gate:** It uses a sigmoid function to determine which information should be added to the cell state.

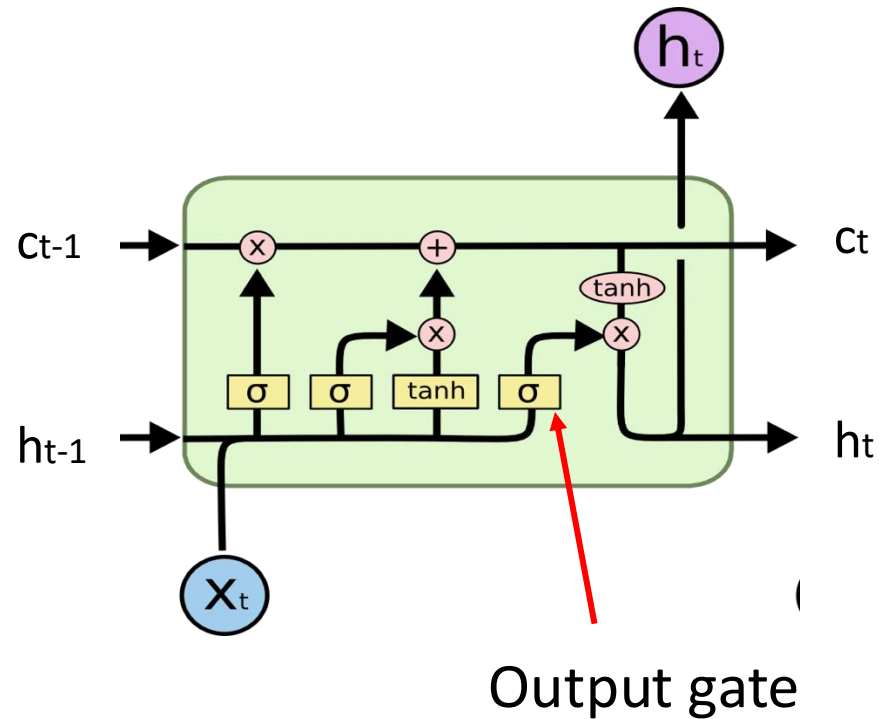
# Long Short-Term Memory



$$V_t = \sigma_r(W_v x_t + U_v h_{t-1} + b_v)$$

- **Candidate Memory:** The candidate memory represents potential **new** information that could be added to the cell state. It is generated by a tanh which is relatively closer to binary than sigmoid.

# Long Short-Term Memory



$$c_t = R_t \cdot c_{t-1} + K_t \cdot \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$$

$$h_t = V_t \sigma_h(c_t)$$

- **Output Gate:** The output gate controls the flow of information from the cell state to the hidden state.

# Long Short-Term Memory

---

LSTM can partially avoid the vanishing gradient problem due to :

- gating mechanisms & memory cells.
  - These components enable LSTM to **selectively retain or discard information** over time.
  - It maintains stable gradient flow during training and capture long-term dependencies more effectively compared to traditional RNNs.

# Problems of time-series

---

Traditional sequential models, such as simple RNNs, suffer from the following issues:

- **Vanishing/Exploding Gradients:** although LSTM alleviate this problem, it is still existed.
- **Difficulty in Parallelization:** Because sequential models **rely on the previous time step's hidden state**, they are challenging to parallelize effectively, limiting their training efficiency on large-scale datasets and the size-growth of language models.





# Attention Mechanisms

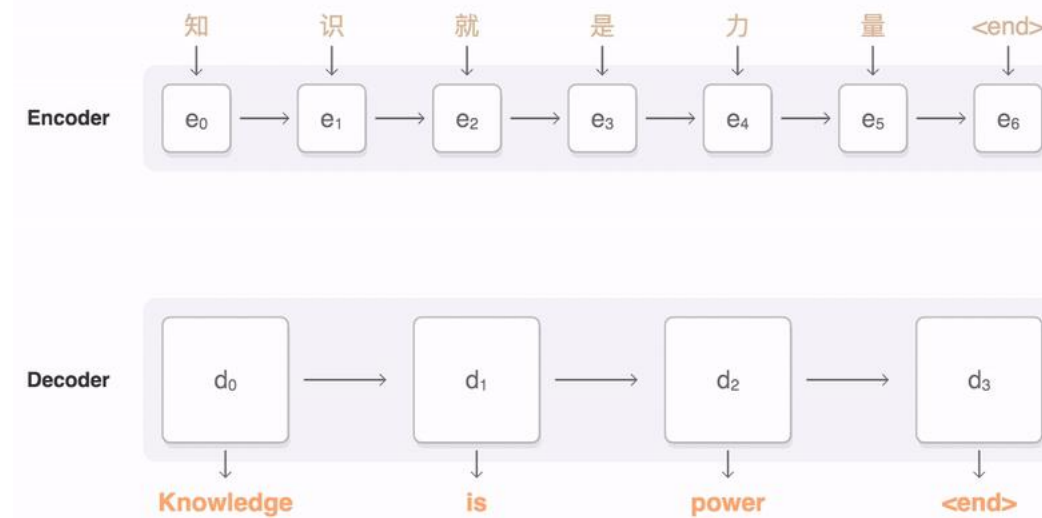
The attention mechanism solve the gradient problems and provided a parallelizable solution of LMs.

Core Idea: To enable a model to **focus on the most relevant parts** of the input sequence when making predictions or generating outputs.

- **Attention**

Attention with RNNs

Attention without RNNs

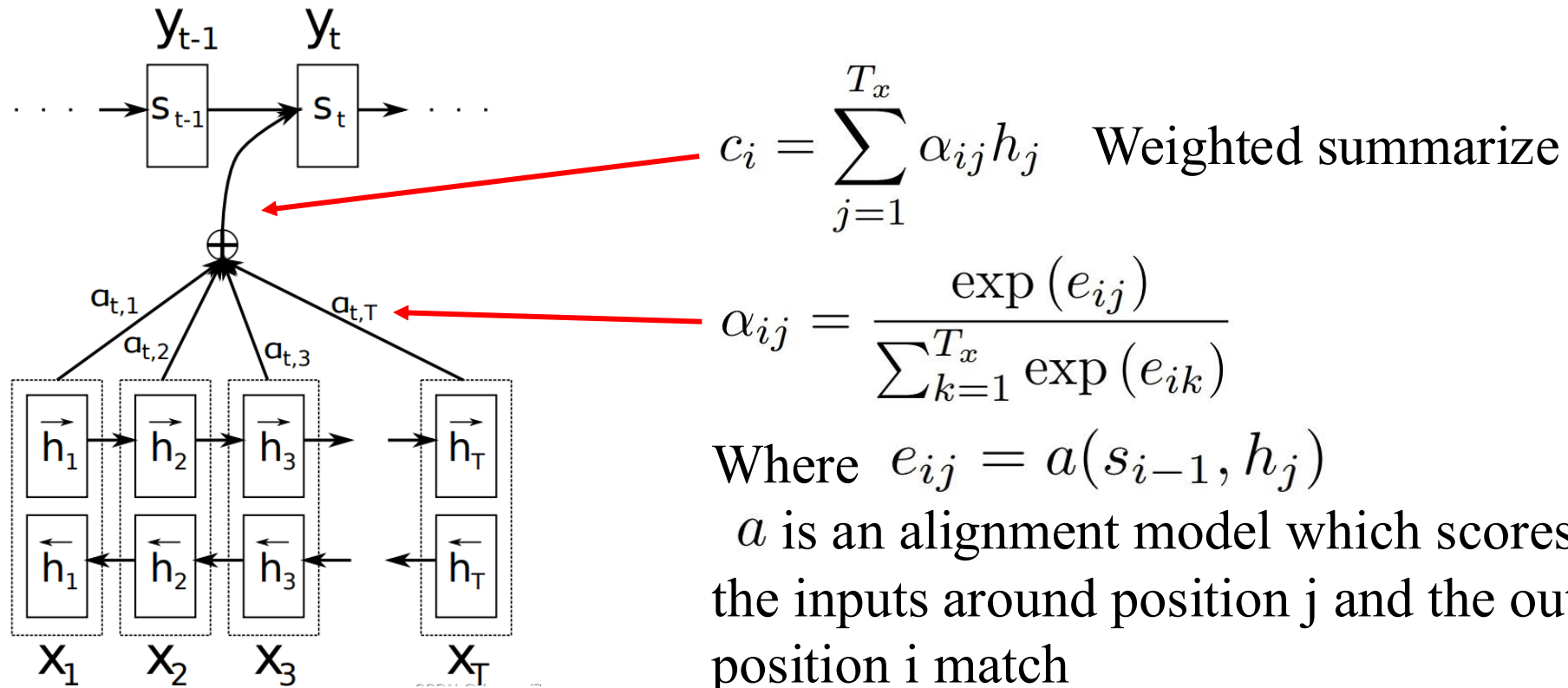


*From Google Seq2seq*



# Attention

At the beginning the attention is associated with RNNs.

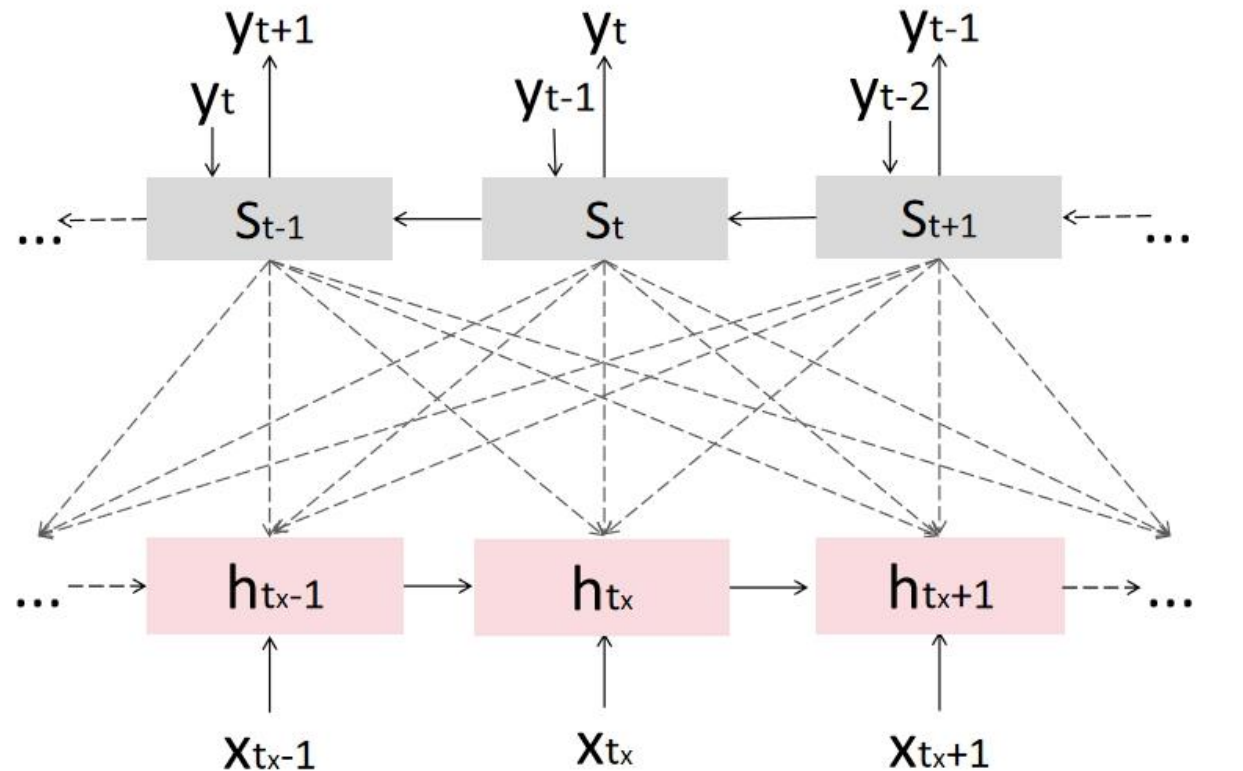


From: Attention 2015



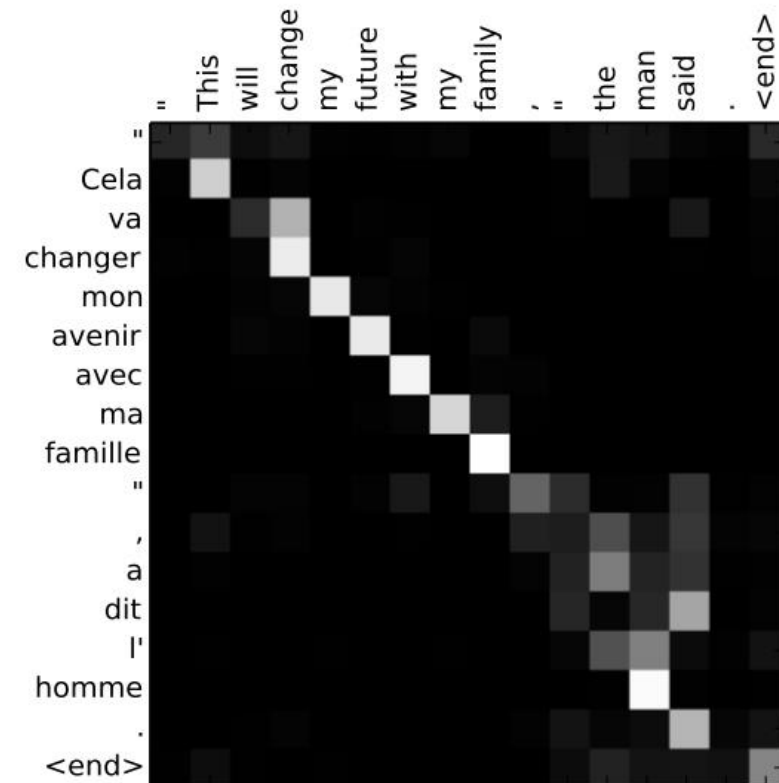
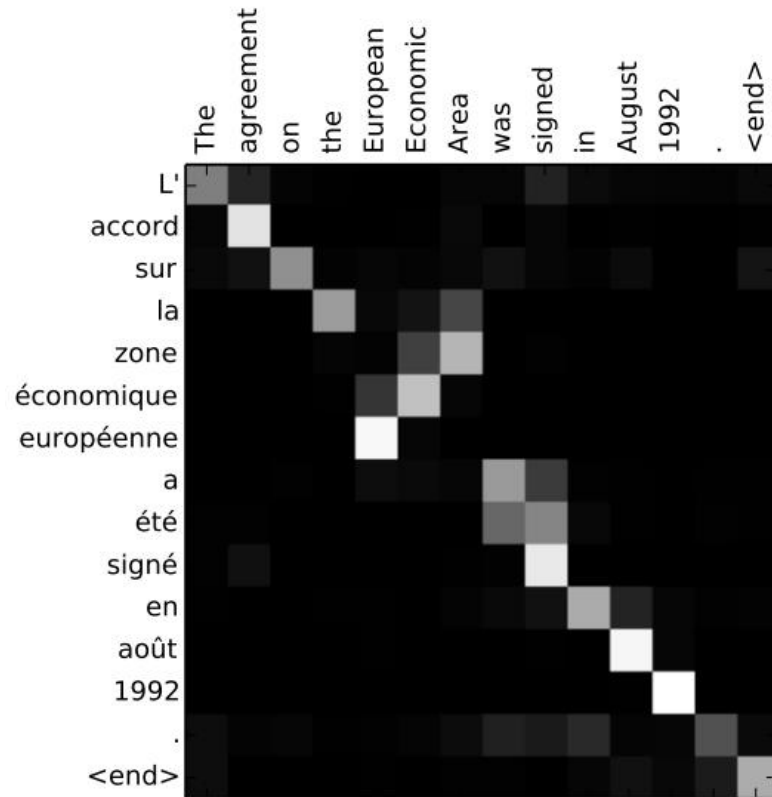
# Attention

When computing the new hidden state  $s_t$ , attention mechanism compute attention scores with all the input tokens (from a bidirectional RNN).



# Attention

An example of machine translation.



From: Attention 2015

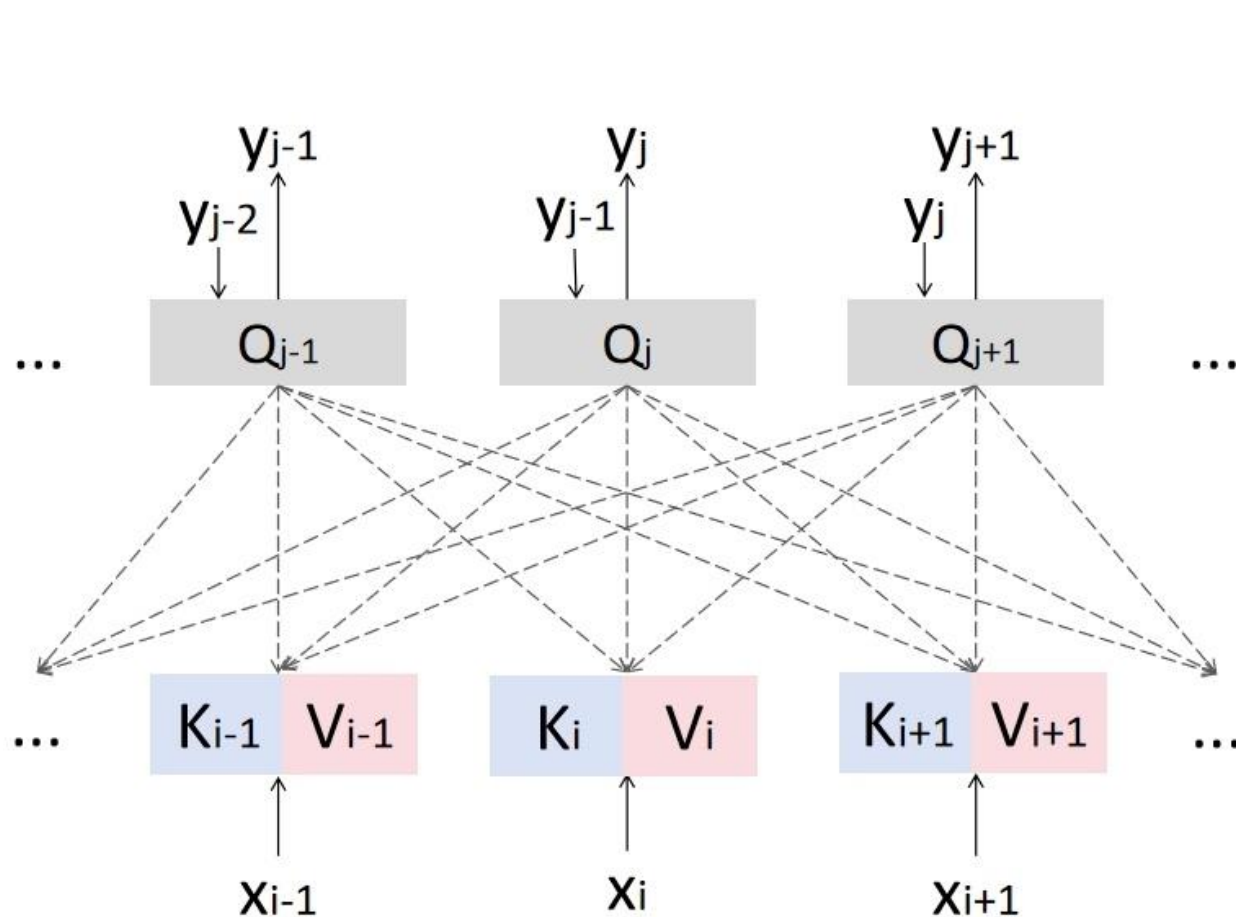
# Attention without RNNs

---

It has been proposed to eliminate the RNN component due to:

- The primary function of the RNN is to extract and process sequential features. However, this functionality can be achieved using simpler methods, leading to **reduced computational complexity**.
- RNNs are **not parallelizable**, meaning they cannot efficiently process multiple input sequences simultaneously, which limits their scalability and efficiency in large-scale applications.

# Attention without RNNs



Learnable weight matrix

$$\begin{aligned} Q &= W_Q \begin{bmatrix} y \end{bmatrix} \text{ output} \\ K &= W_K \begin{bmatrix} x \end{bmatrix} \text{ input} \\ V &= W_V \begin{bmatrix} x \end{bmatrix} \end{aligned}$$

Attention score:

$$\alpha_i = \text{Softmax}\left(\frac{Q^\top K}{\sqrt{d}}\right)$$

$$y_{\text{output}} = \sum_{i=1}^N \alpha_i v_i$$



# Why $\sqrt{d}$

---

The model's parameters should undergo normalization, ensuring their average is 0 and variance is 1.

Assume that  $q_i$  and  $k_i$  are random variable with average 0 and variance 1:

$$E(q_i k_i) = E(q_i)E(k_i) = 0$$

$$\begin{aligned} Var(q_i k_i) &= E(q_i^2 k_i^2) - E(q_i k_i)^2 \\ &= E(q_i^2 - 0^2)E(k_i^2 - 0^2) \\ &= E(q_i^2 - E(q_i)^2)E(k_i^2 - E(k_i)^2) \\ &= Var(q_i)Var(k_i) = 1 \end{aligned}$$



# Why $\sqrt{d}$

---

However, after we multiply Q and K, the variance becomes d:

$$\text{Var}(Q^\top K) = \text{Var}\left(\sum_{i=0}^d q_i k_i\right) = d \cdot 1 = d$$

So it should be divided by  $\sqrt{d}$ :

$$\text{Var}\left(\frac{Q^\top K}{\sqrt{d}}\right) = \frac{d}{(\sqrt{d})^2} = 1$$



# Summary

---

Sequential Models:

**RNN:** Recurrent Neural Network, the fundamental NN in NLP tasks.

**LSTM:** A modification of RNNs designed to alleviate gradient issues.

Attentions:

**Attention with RNN:** Avoid gradient vanishing in RNNs.

**Attention without RNN:** Parallelizable and simpler.



# Reference

---

## Stanford NLP:

- <https://web.stanford.edu/~jurafsky/slp3/>

## Attention 2015:

- Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.