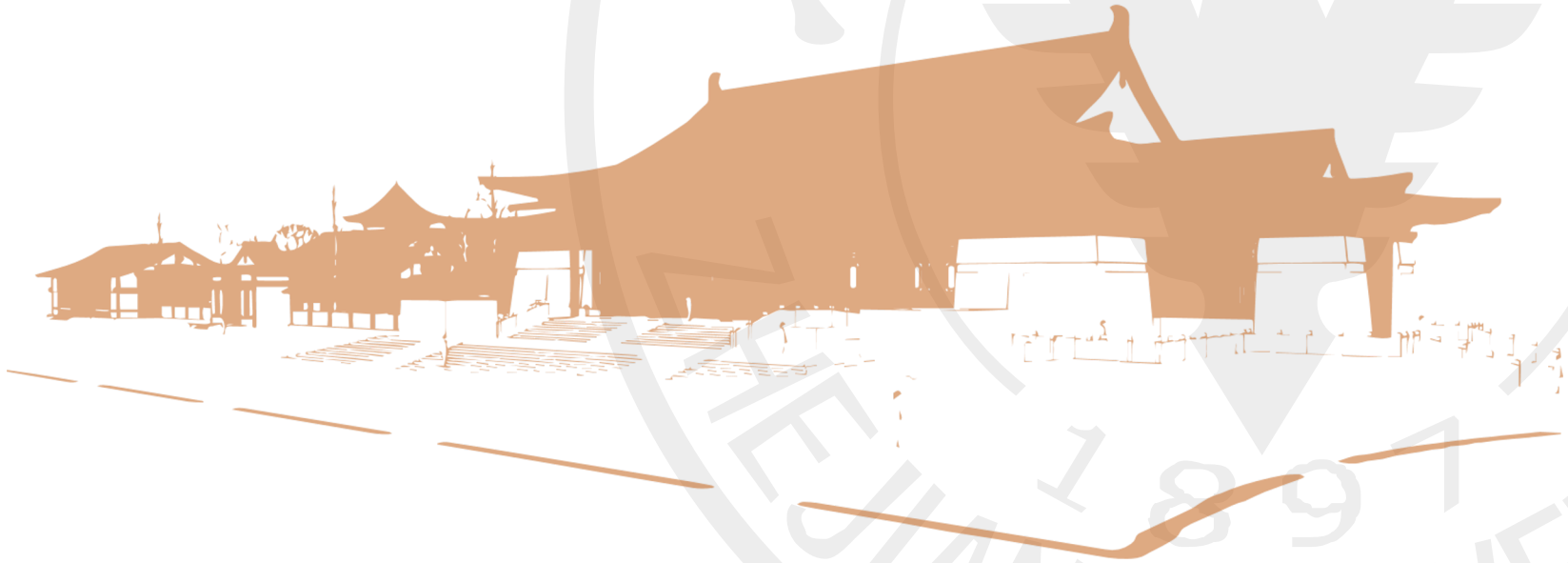
LSTM and Transformer



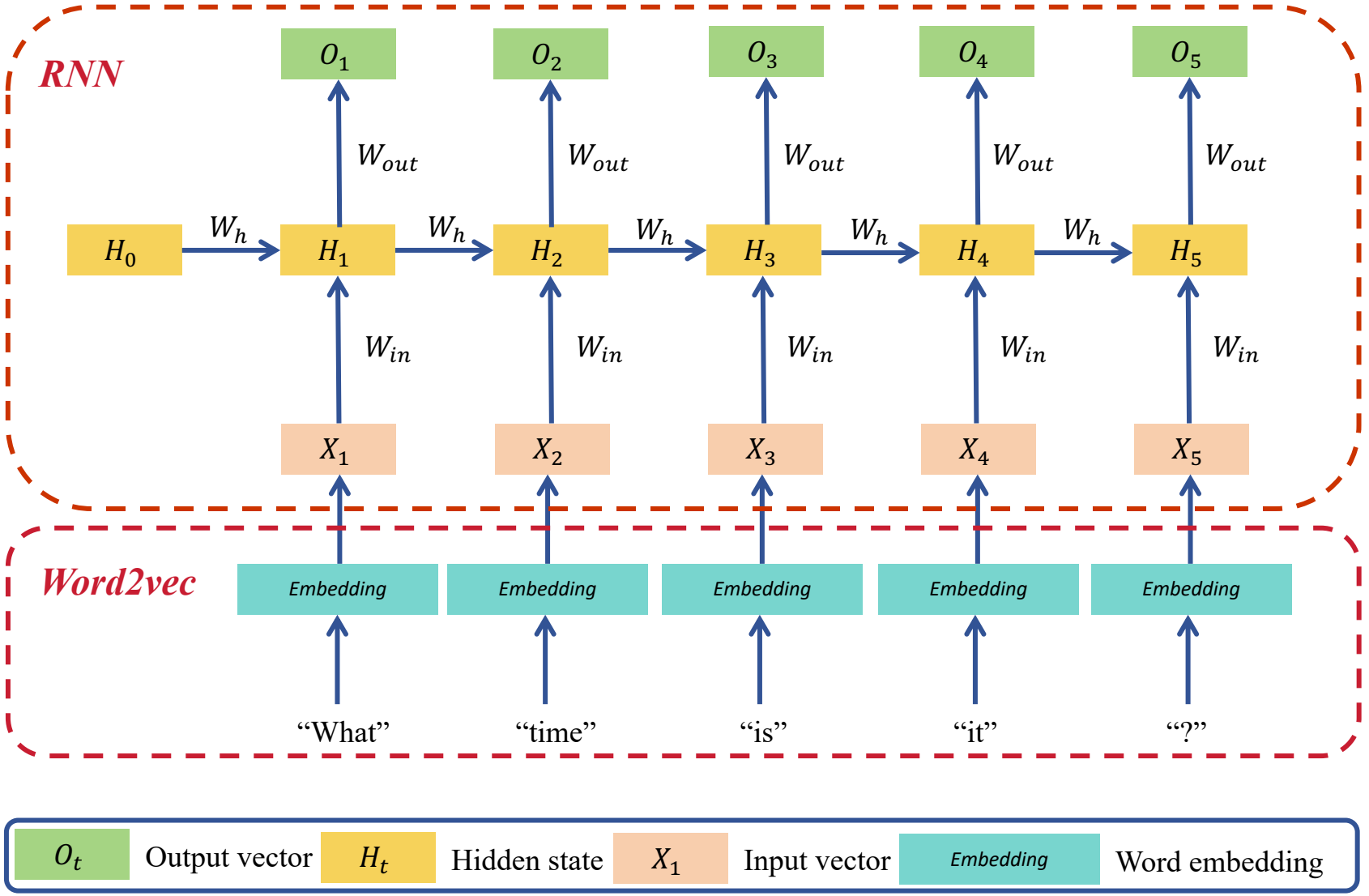
Review

What time is it?

Review

- Use **RNN (Recurrent Neural Network)** to predict outputs

- Use **Word2vec** to obtain continual representation

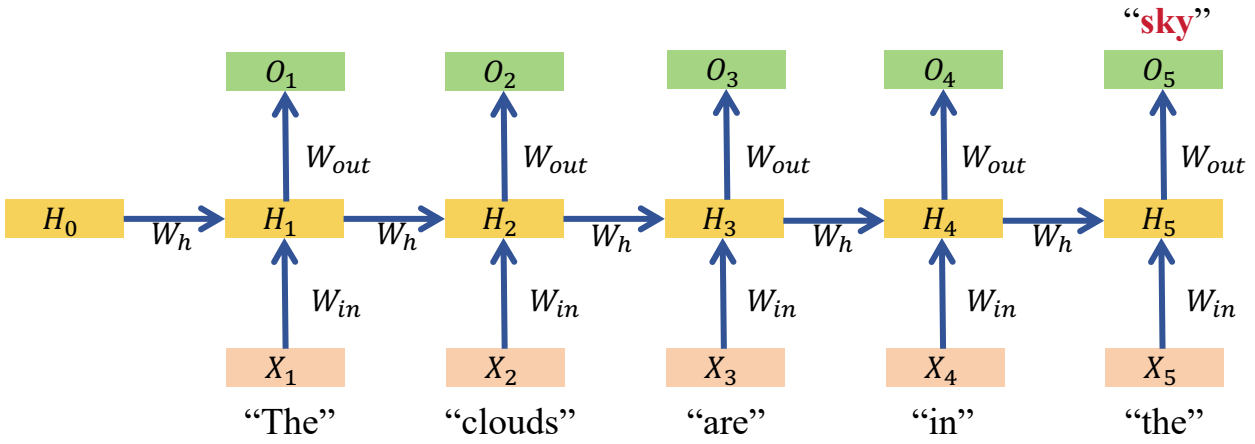


Overview

- **Limitation of RNN (Recurrent Neural Network)**
- **LSTM (Long Short-Term Memory network)**
- **Attention Mechanism**
- **Transformer**

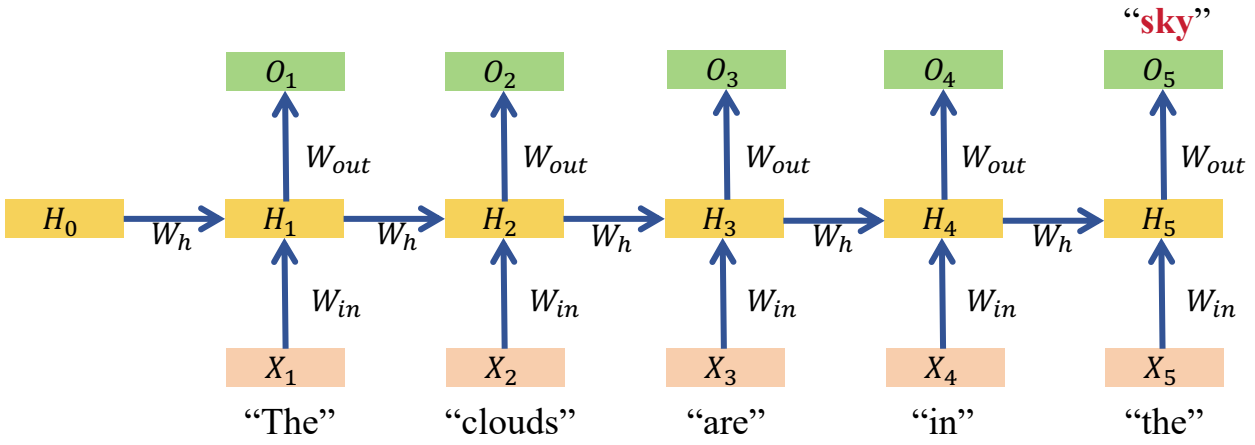
Limitation of RNN

- RNN cannot handle long-term dependencies.
  - For a short sequence, like "***The clouds are in the ???***", RNN can predict the next word "***sky***".

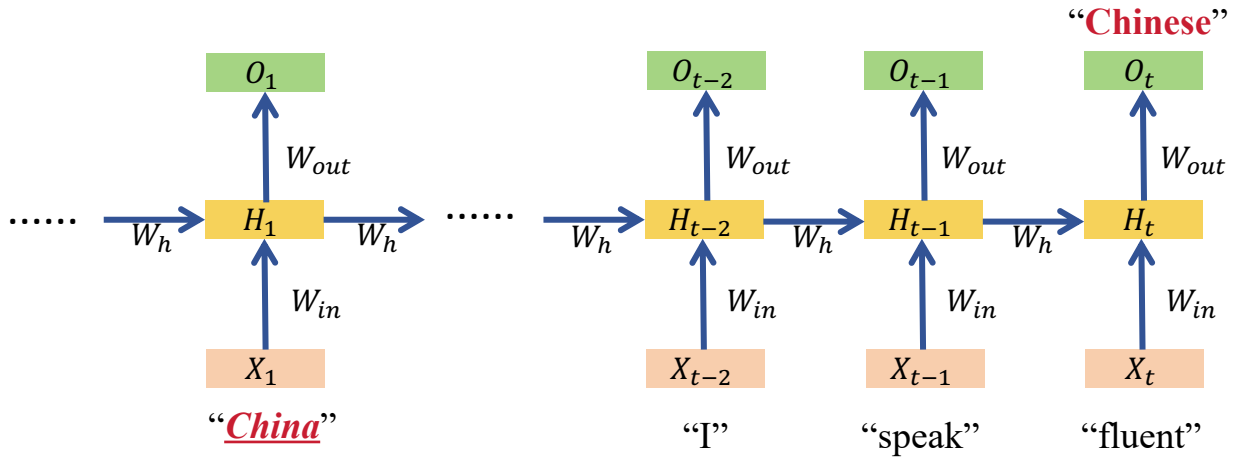


Limitation of RNN

- RNN cannot handle long-term dependencies.
  - For a short sequence, like "***The clouds are in the ???***", RNN can predict the next word "***sky***".

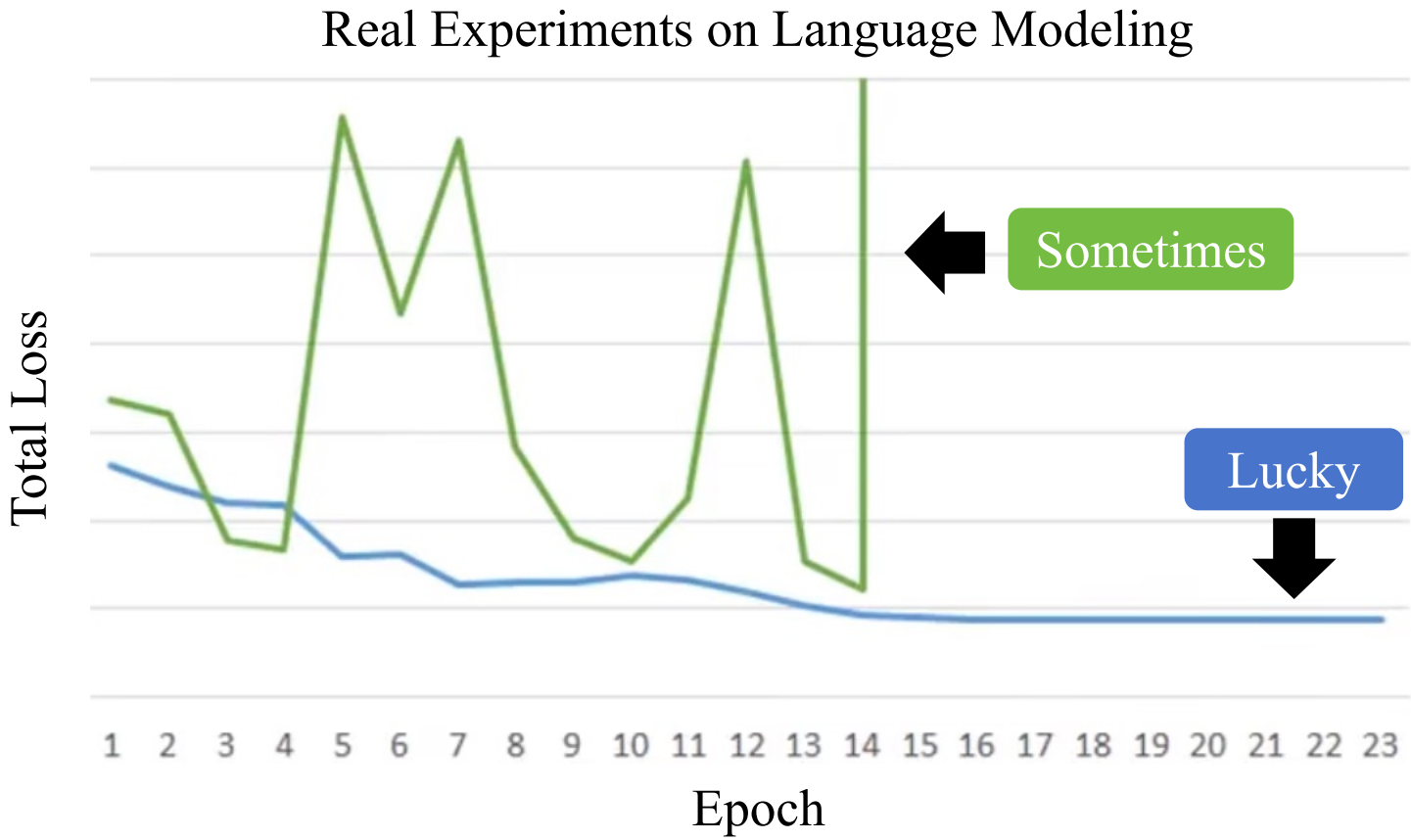


  - For a very long sequence, like "***I grew up in China and had .....(a long sentence), so I speak fluent ???***", it very hard for RNN to predict the next word "***Chinese***".



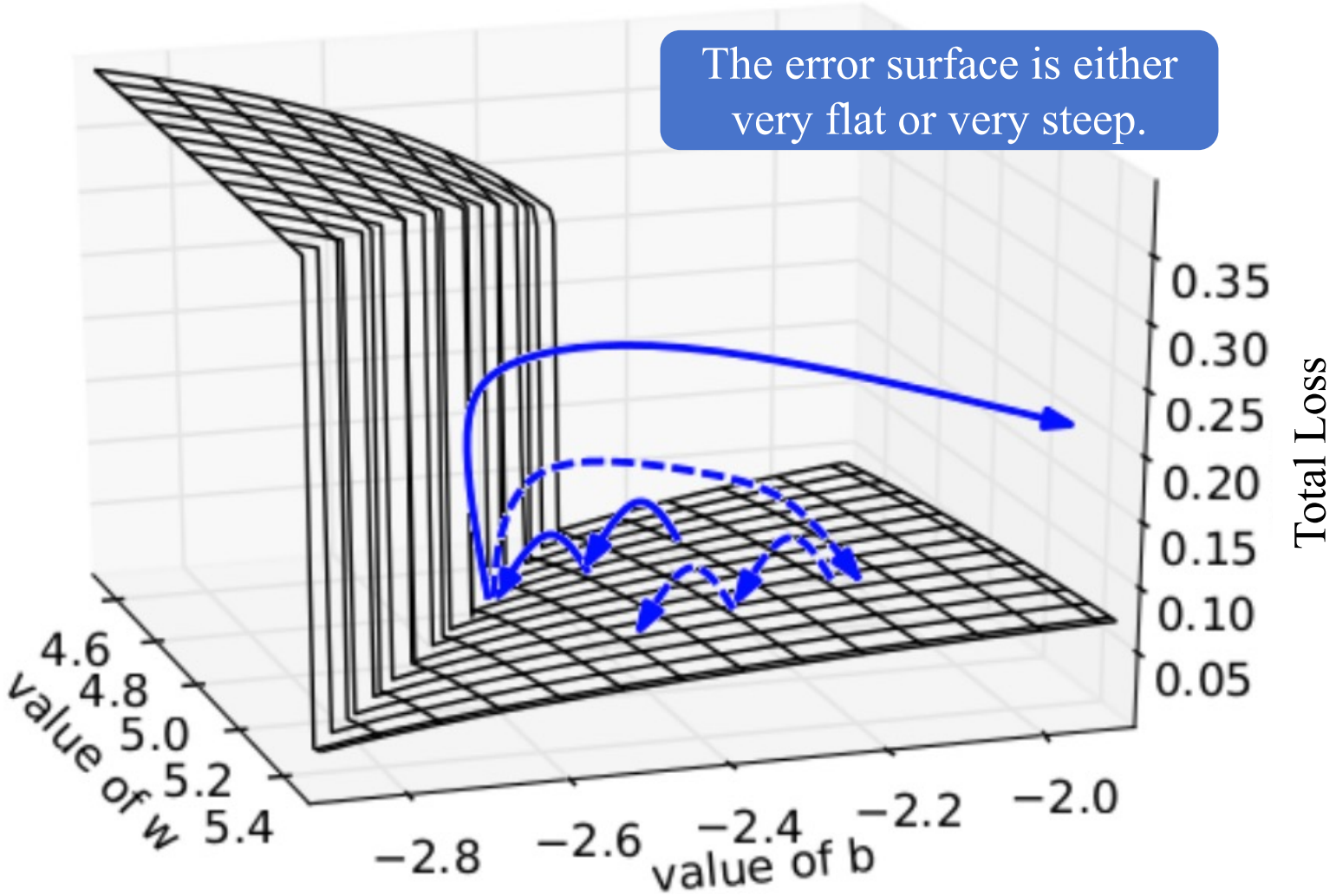
Limitation of RNN

- RNN-based network is not always easy to train :(



Limitation of RNN

- The error surface is rough



求是创新 On the difficulty of training recurrent neural networks, *ICML*'13

Limitation of RNN

- A toy example to explain the difficulty during gradient descent

$W_h = 1 \rightarrow O_{t=1000} = 1$

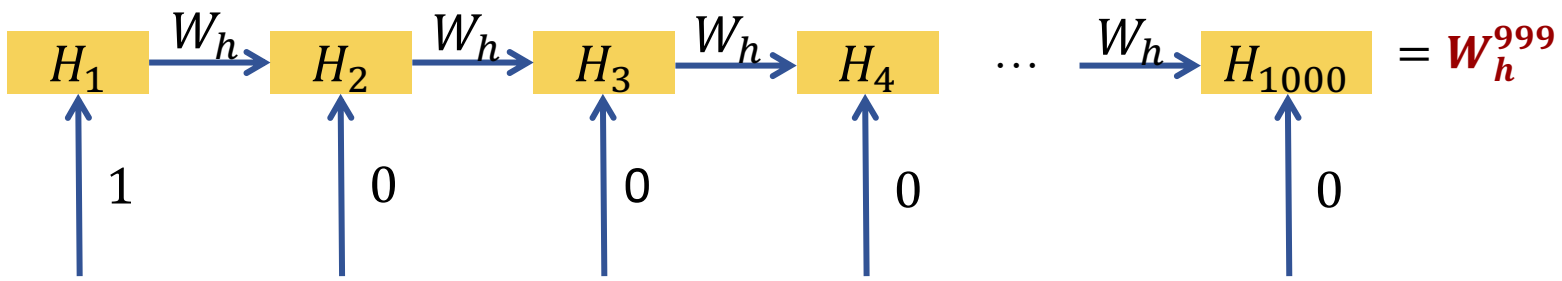
$W_h = 1.01 \rightarrow O_{t=1000} \approx 20000$

Large $\partial L/\partial W$ → Small Learning Rate?

$W_h = 0.99 \rightarrow O_{t=1000} \approx 0$

$W_h = 0.01 \rightarrow O_{t=1000} \approx 0$

Small $\partial L/\partial W$ → Large Learning Rate?



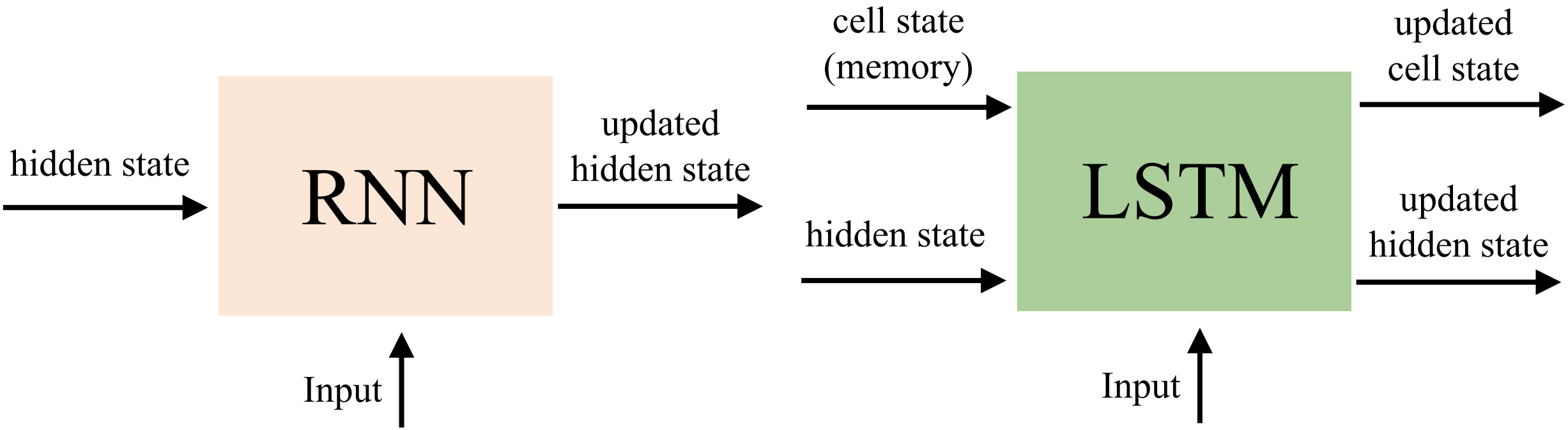
LSTM

How to address the long-dependency problem?

How to address the optimization problem?

Use Gated RNN, *e.g.*, LSTM (Long Short-Term Memory networks).

Overview of LSTM



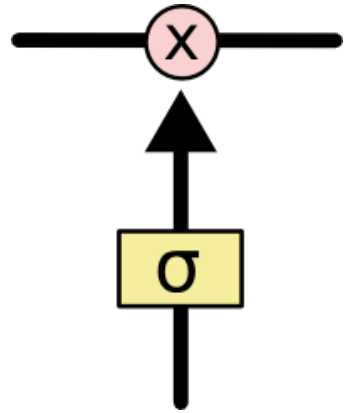
Long short-term memory. *Neural computation"97*

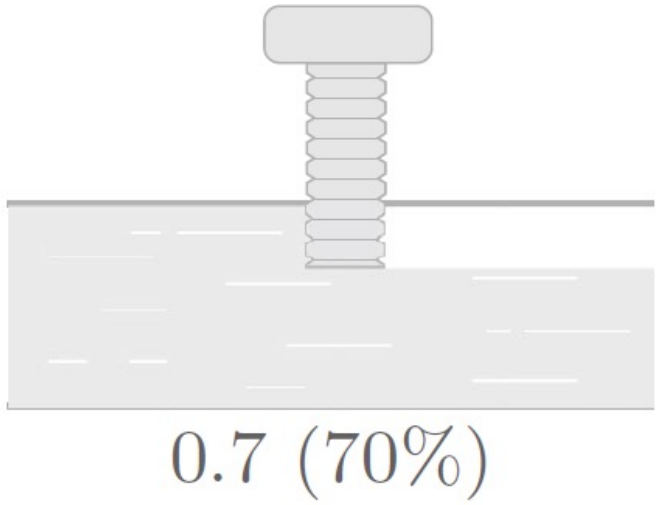
Two core ideas behind LSTM

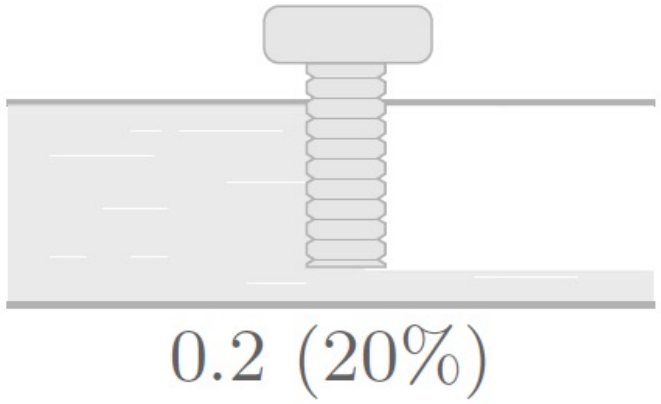
◆ Gate

- A mechanism to control the flow of information
- Use sigmoid (*i.e.*, $\boldsymbol{\sigma}$) to regulate information amount

  i.e., 0 means 'block all', 1 means 'allow all'







LSTM

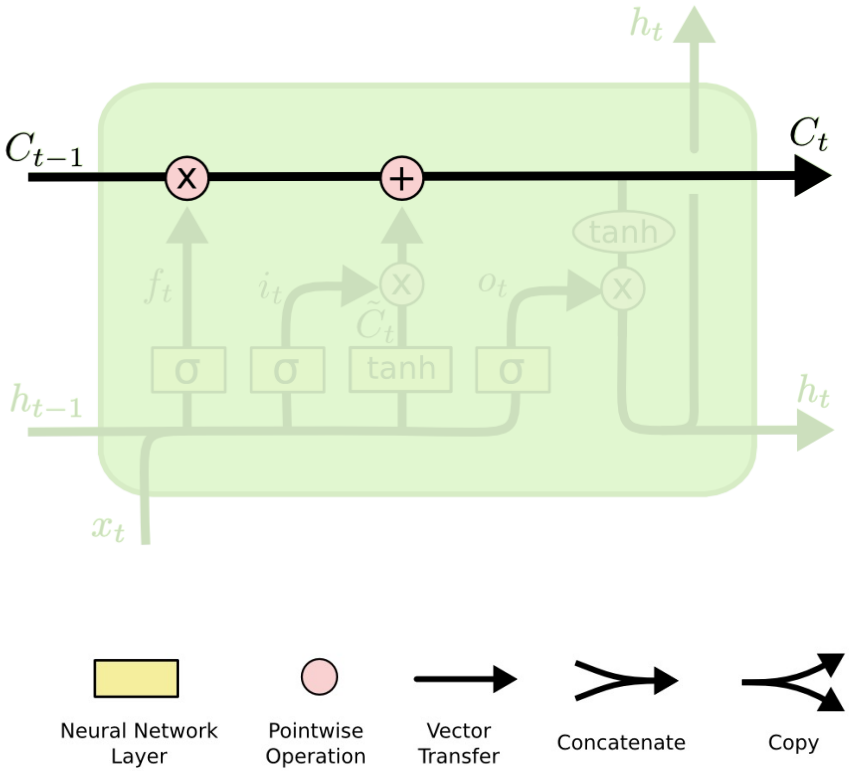
Two core ideas behind LSTM

- **Gate**
  - A mechanism to control the flow of information
  - Use sigmoid (*i.e.*, σ) to regulate information amount

    i.e., 0 means 'block all', 1 means 'allow all'

- **Cell state** (memory)
  - Run through the top of the diagram
  - Transmit information **unchanged**

- LSTM has three gates to update and control the cell state.

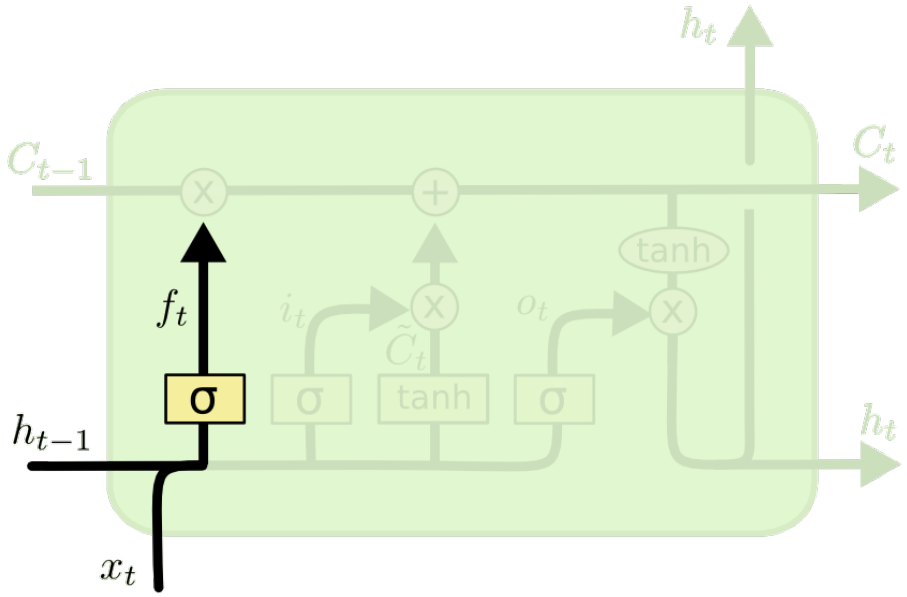


Step-by-step LSTM

First Step: Decide what information we're going to **throw** away from the cell state.

- **Tool:** ***forget gate layer***
  - **Component**: a **sigmoid** layer
  - **Function**: Look at old ***hidden state*** and ***input***

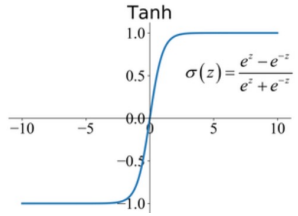
    output a number between 0 and 1 for the old cell state.



Step-by-step LSTM

Second Step: Decide what new information we're going to **store** in the cell state.

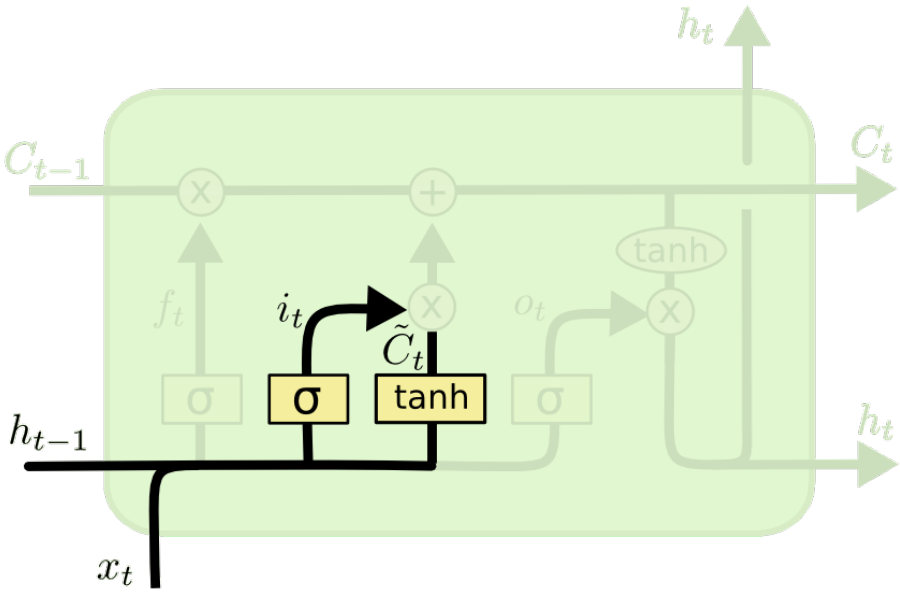
- **Tool-1:** ***a tanh layer***



  - **Function**: Generate a candidate cell state

- **Tool-2:** ***input gate layer***

  - **Component**: a sigmoid layer
  - **Function**: Decide which values we'll use in candidate cell state

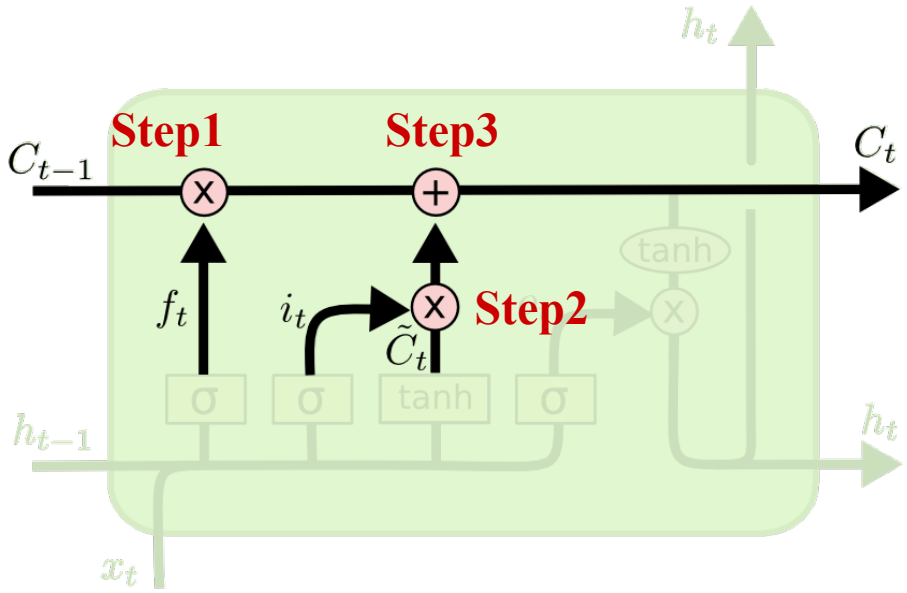


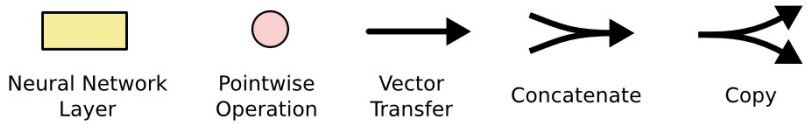
Neural Network Layer | Pointwise Operation | Vector Transfer | Concatenate | Copy

Step-by-step LSTM

Third Step: **Update** the **cell state**.

- **Sub-step-1**
  - **Function**: Forget things in cell state not related to input
- **Sub-step-2**
  - **Function**: Select values in candidate to be used.
- **Sub-step-3**
  - **Function:** Add them to obtain the **updated** cell state C_t.

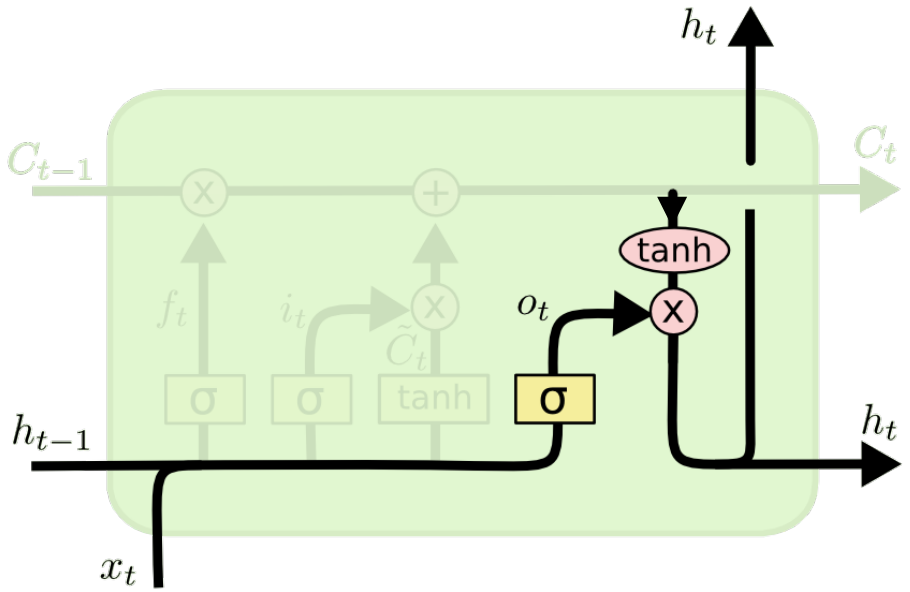




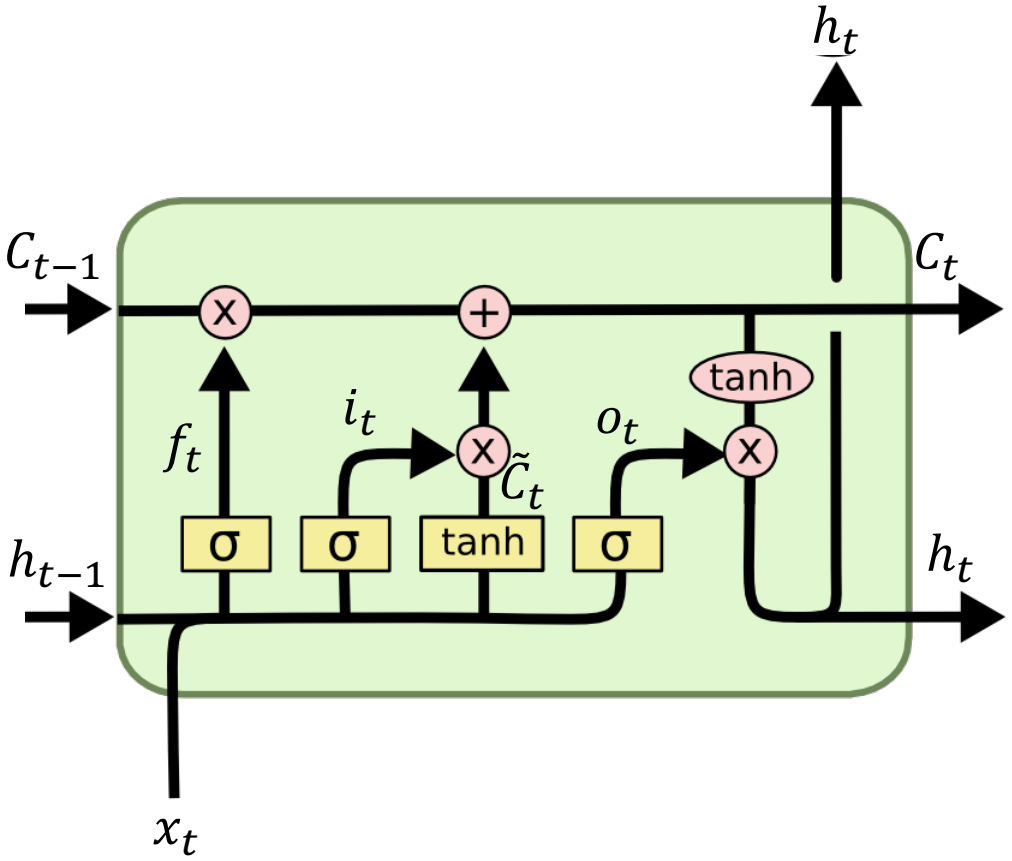
Step-by-step LSTM

Fourth Step: Decide what we're going to **output**.

- **Tool:** ***output gate layer***
  - **Component**: a sigmoid layer
  - **Function**: Decide what parts of the cell state we're going to **output** → hidden state.



Summary of LSTM



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

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

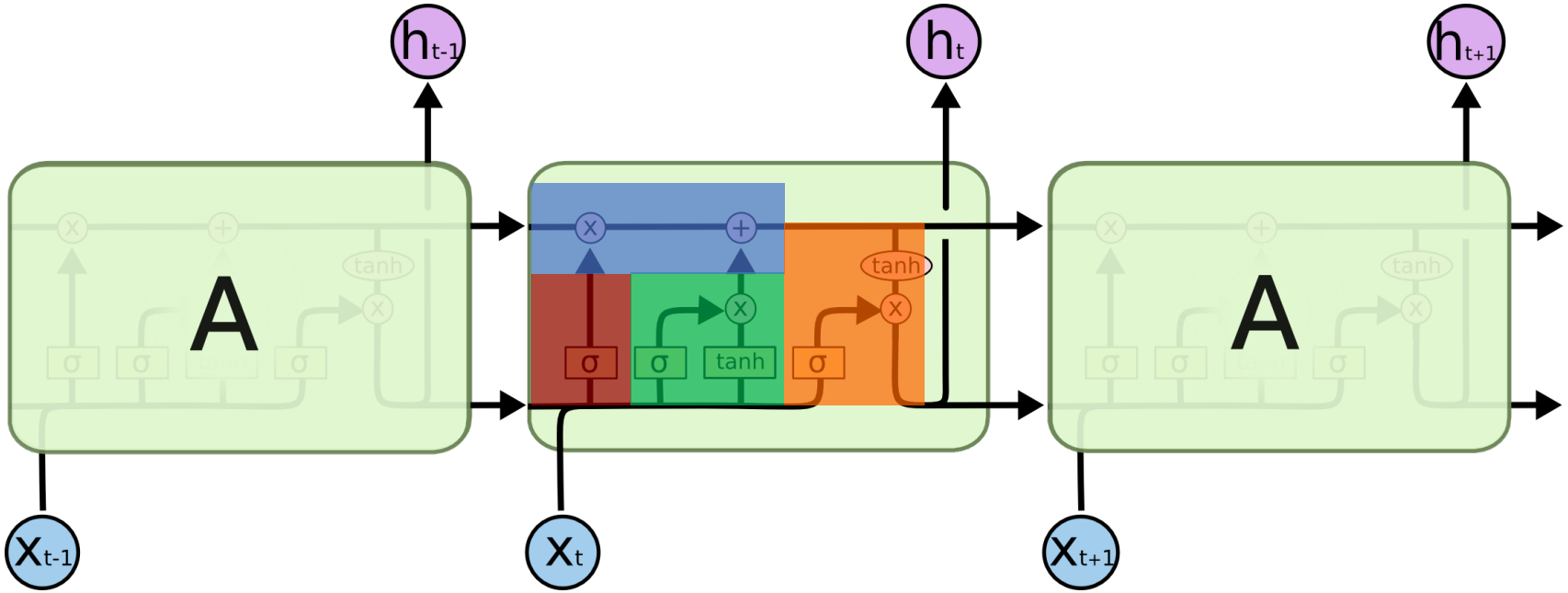
Candidate Cell State $\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$

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

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

Updated Hidden State $h_t = o_t * tanh(C_t)$

Summary of LSTM



(1) ***Forget gate:*** decide what information we will **throw** away from the cell state.

(2) ***Input gate:*** decide what **new information** we will **store** in the cell state.

(3) *Cell state*: store **short-term** memory in a **long** time

(4) ***Output gate:*** **output** the **new hidden state** h_t

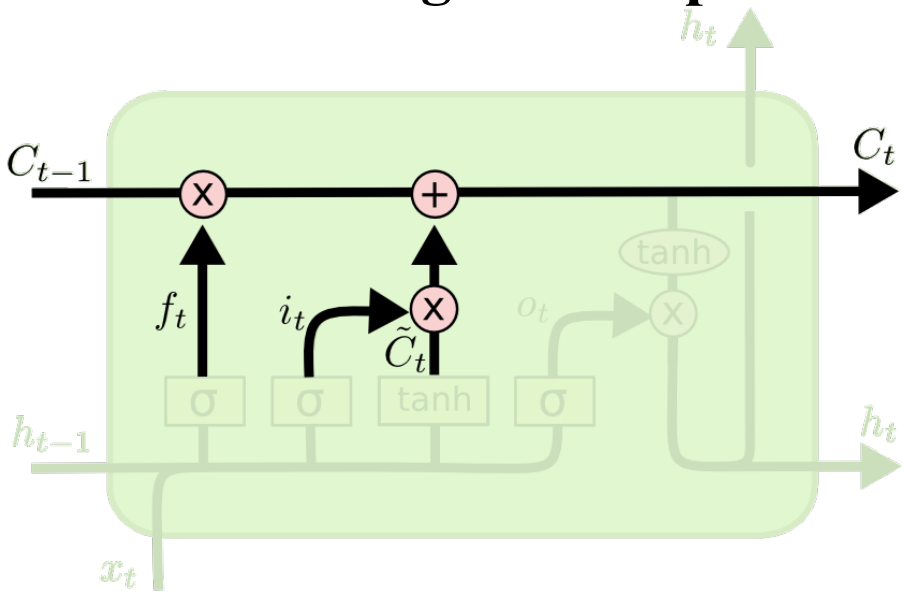
Problems

How to address the long-dependency problem?

How to address the optimization problem?

Advantage of LSTM

- **Alleviate the long-term dependency issue**



Short-term memory can be transmitted in a **long** time through the **cell state**

- **Alleviate the optimization problem**
  - LSTM: the **gradient** is controlled by gates
  - The long-distance gradient will **NOT** completely disappear

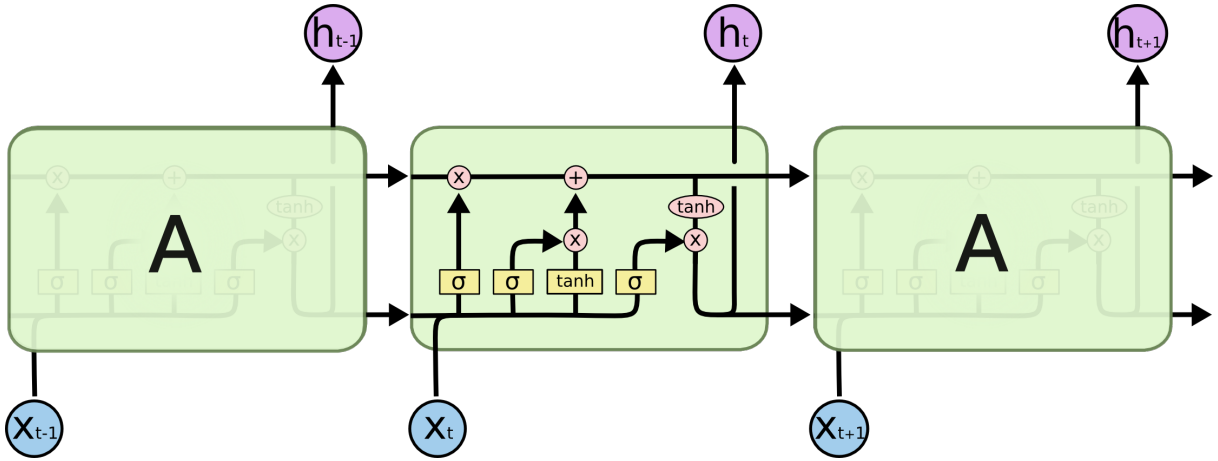
Limitation of LSTM

- **The long-term dependency of LSTM will still be lost.**

For instance:

- "*I grew up in China and had ..... I speak fluent* ***???***".

Chinese



"China" … "I" "speak" "fluent"

- The hidden state can **NOT** store too much information due to its limited capacity
- Due to the sequential nature, the importance of each word during prediction follows the relationship: ***fluent > speak > I >>> China.***

Attention Mechanism

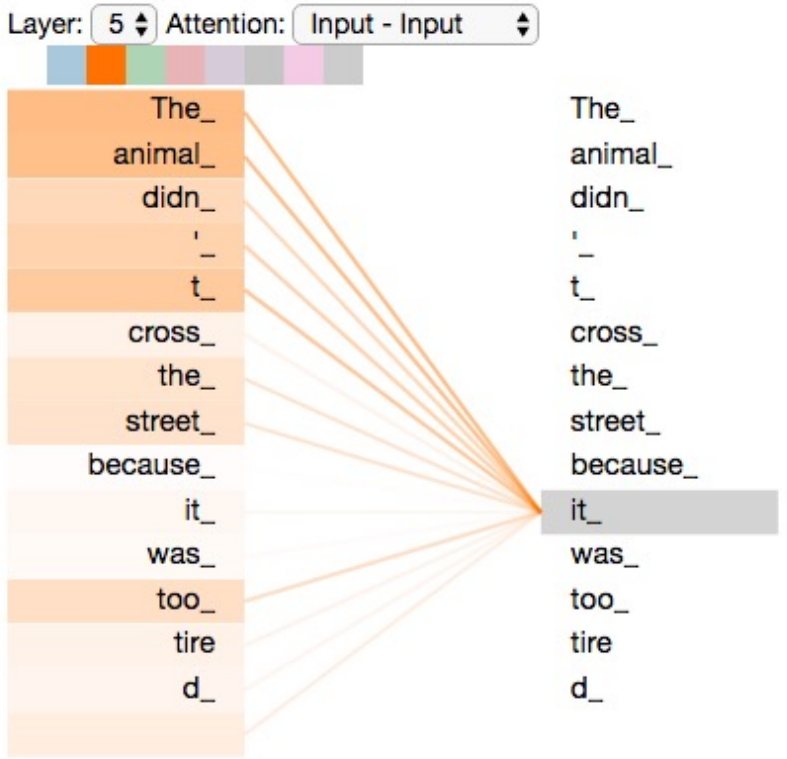
Q: How to enable models to handle longer sequences?

A: Attention Is All You Need.

Attention mechanism: enable models to focus on relevant information even with long-term dependency.

Example

- "The ***animal*** didn't cross the ***street*** because ***it*** was too tired."
- What does ***it*** refer to?
  The **animal** or the **street**?
- Associating ***it*** with all words,
- **The attention mechanism** can tell us the relations with different **attention values**.



***darker color means higher attention**

Attention Mechanism

***Q*: How to obtain the **attention value**?**

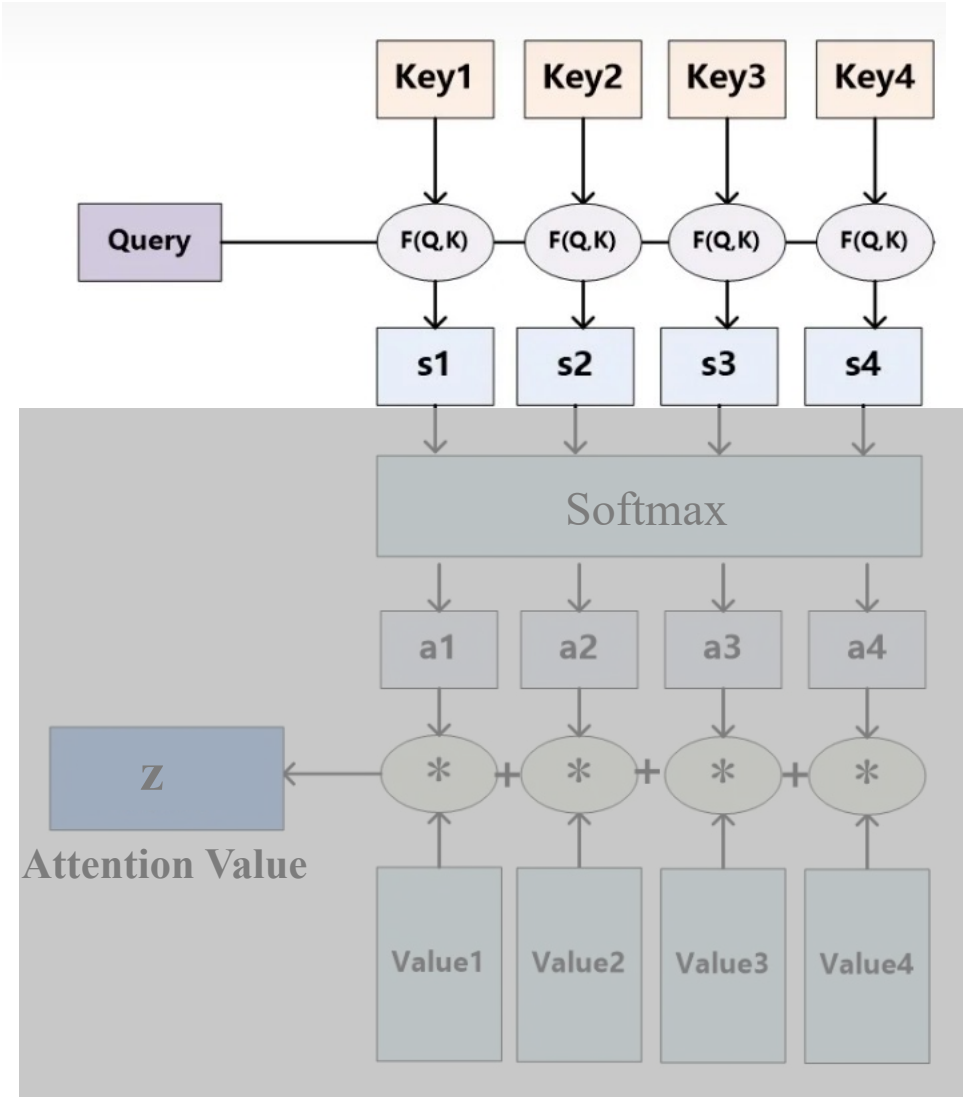
A: Calculate the **correlation** between any two words.

Attention Mechanism

Query, **Key,** and **Value:** Three embeddings for each word

(Input: word2vec X; Output: $Query$, Key, and $Value$)

- $Query, Key, Value = Q \times X, K \times X, V \times X$
  - → Q, K and V are learnable matrixes.
- $s = F(Query, Key) = \frac{Query \cdot Key}{\sqrt{d_k}}$
  - → **Score**: correlation between Query and Key

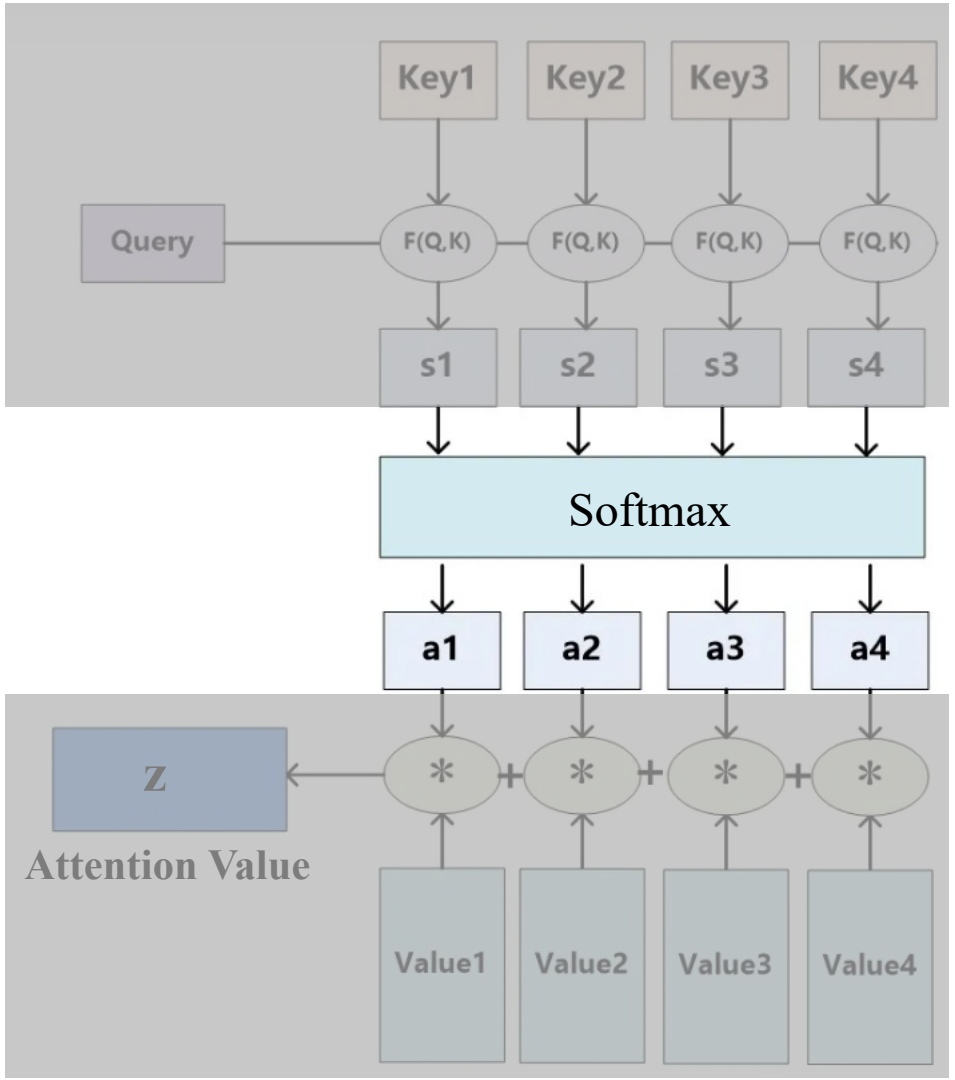


Attention Mechanism

Query, **Key,** and **Value:** Three embeddings for each word

(Input: word2vec X; Output: $Query$, Key, and $Value$)

- $Query, Key, Value = Q \times X, K \times X, V \times X$
  - → Q, K and V are learnable matrixes.
- $s = F(Query, Key) = \frac{Query \cdot Key}{\sqrt{d_k}}$
  - → Score: correlation between Query and Key
- $a = \text{Softmax}(\frac{Query \cdot Key}{\sqrt{d_k}})$.
  - → **Attention Weight:** Normalized score (0-1)

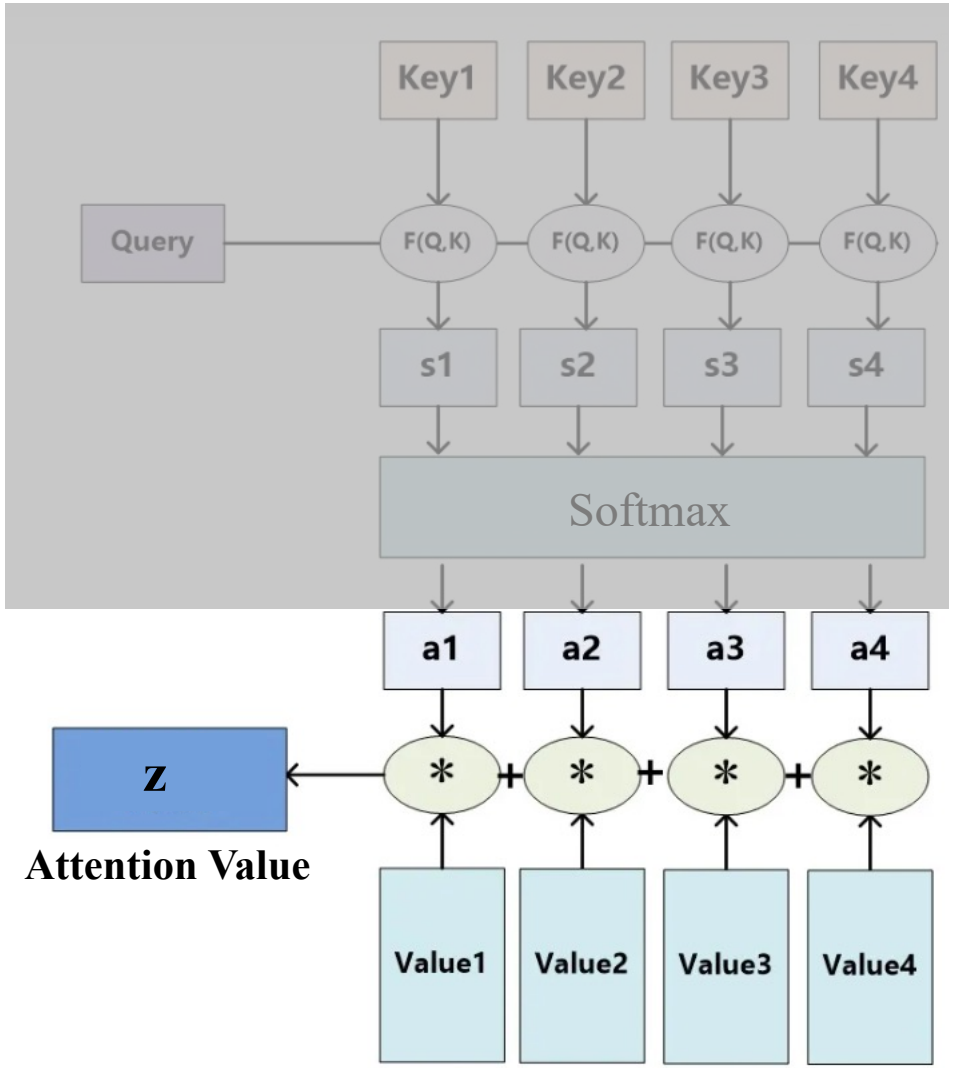


Attention Mechanism

Query, **Key,** and **Value:** Three embeddings for each word

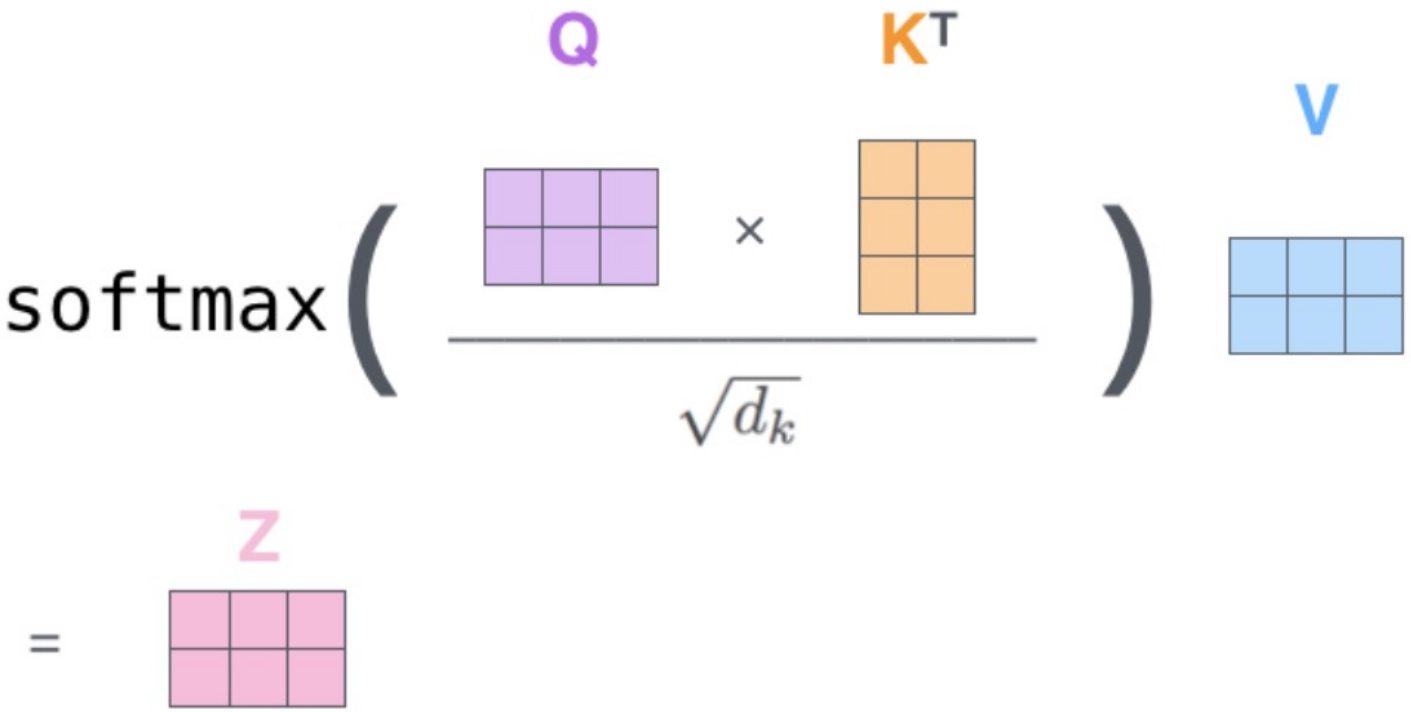
(Input: word2vec X; Output: $Query$, Key, and $Value$)

- $Query, Key, Value = Q \times X, K \times X, V \times X$
  - → Q, K and V are learnable matrixes.
- $s = F(Query, Key) = \frac{Query \cdot Key}{\sqrt{d_k}}$
  - → Score: correlation between Query and Key
- $a = \text{Softmax}(\frac{Query \cdot Key}{\sqrt{d_k}})$.
  - → **Attention Weight:** Normalized score (0-1)
- $z = \sum(a * Value)$
  - → **Attention Value** (the sum of weighted value)



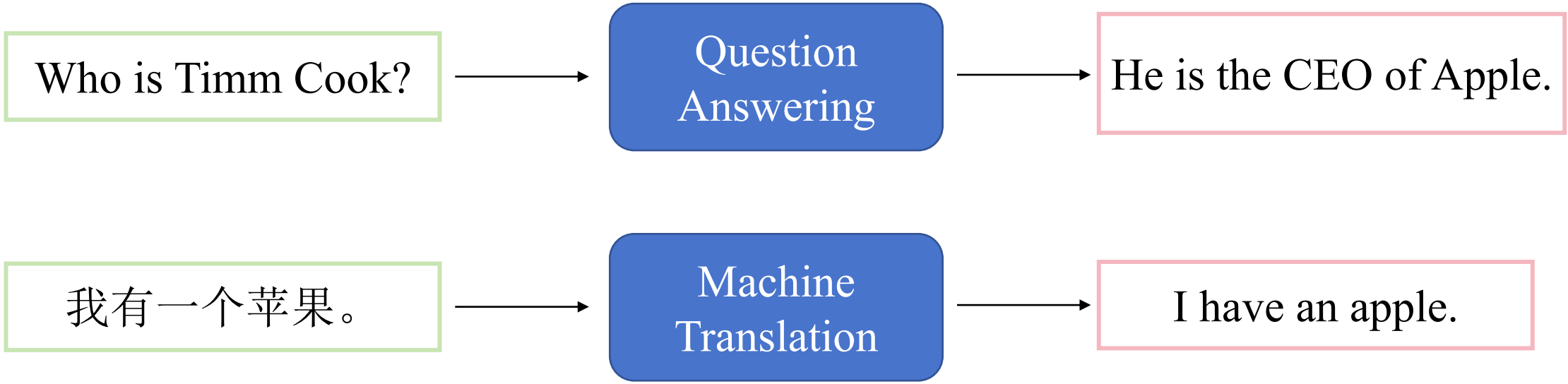
Matrix Calculation for Self-attention

In practice, we use matrix operations for parallel computing.



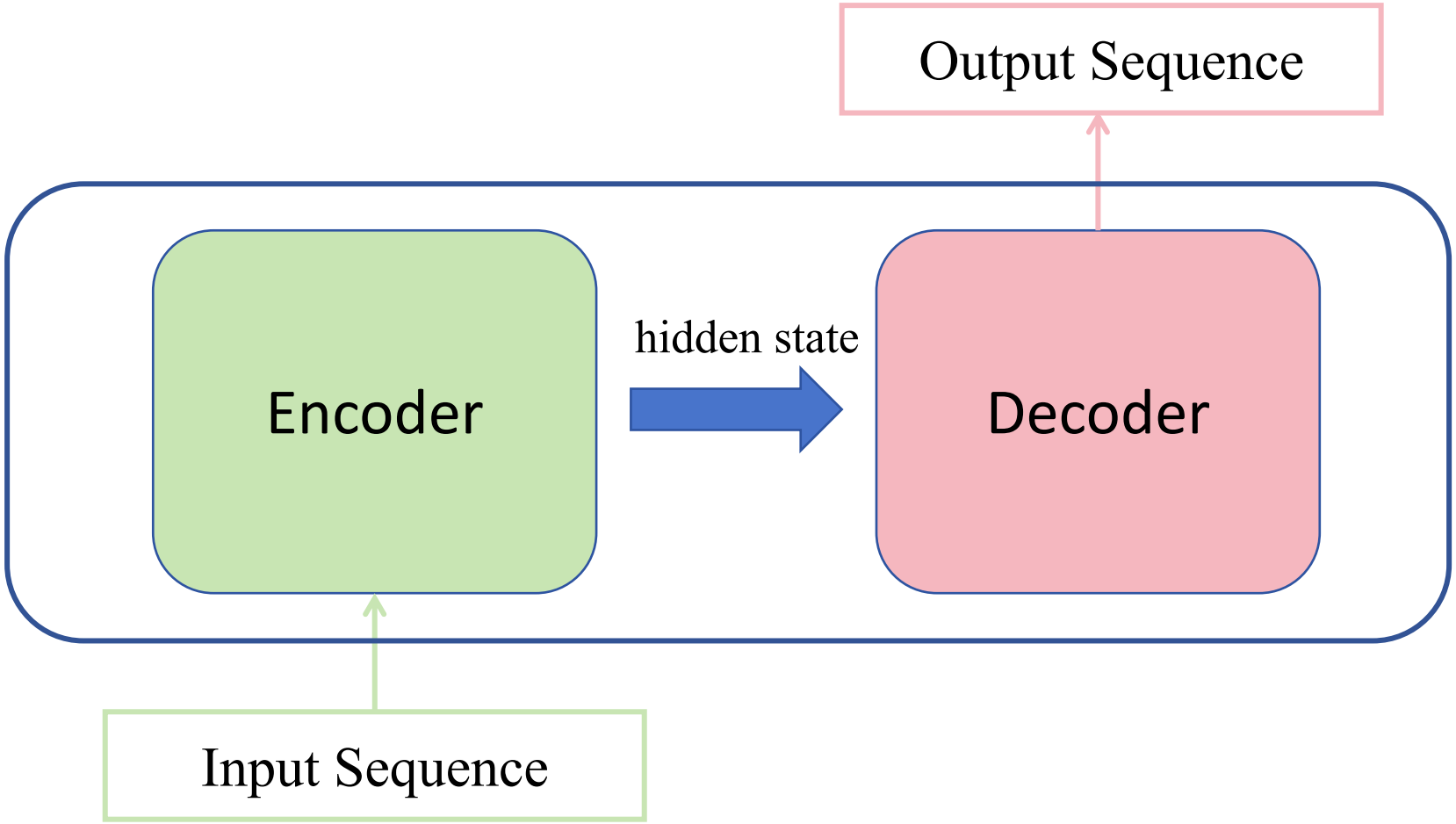
Sequence-to-sequence (Seq2seq)

- Input a sequence, output a sequence
- The output length is determined by model



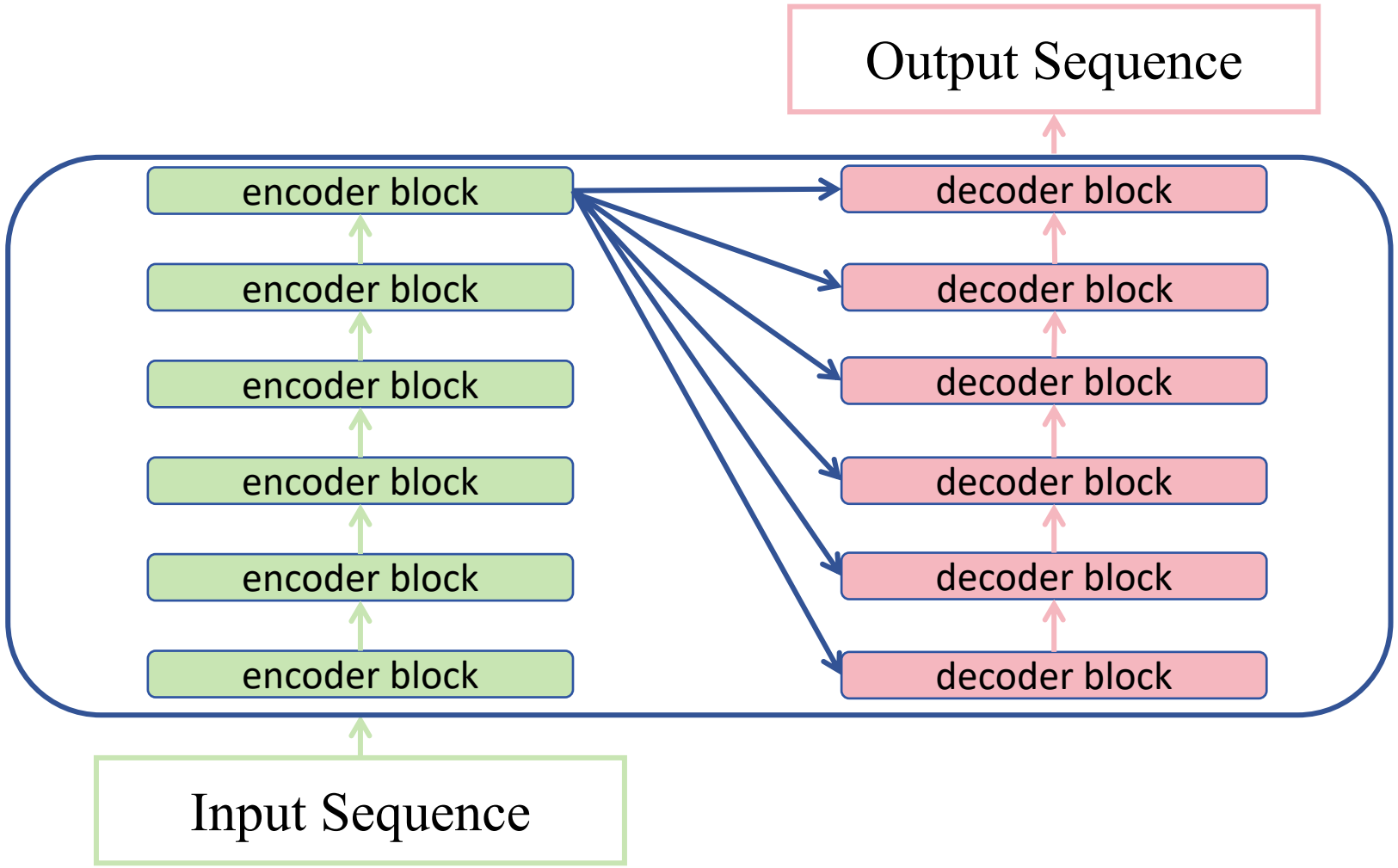
Encoder-decoder

- Encoder-decoder is a universal structure for seq2seq
- Transformer is an implementation of encoder-decoder



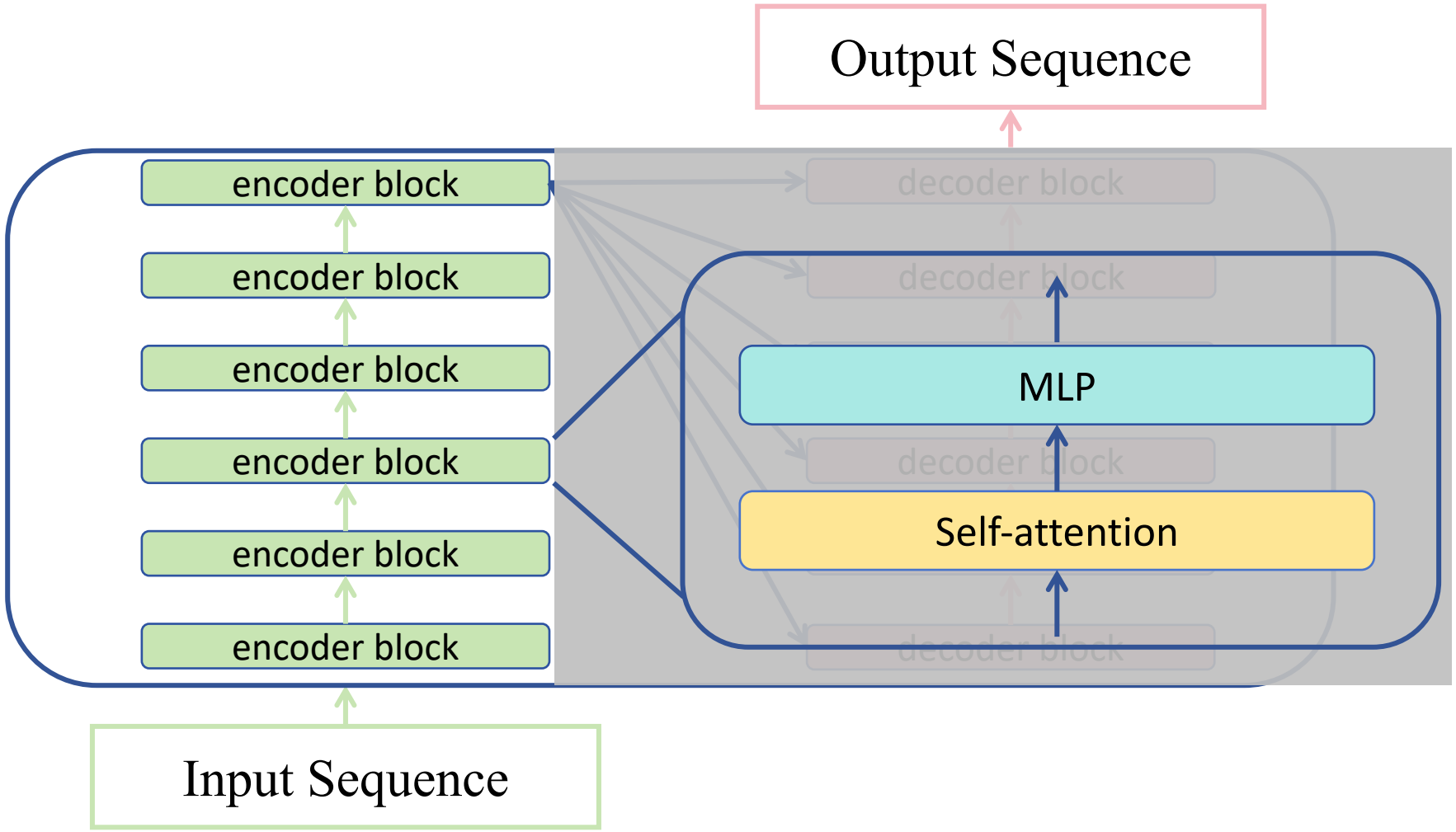
Transformer

- The encoder and decoder consist of a set of blocks.



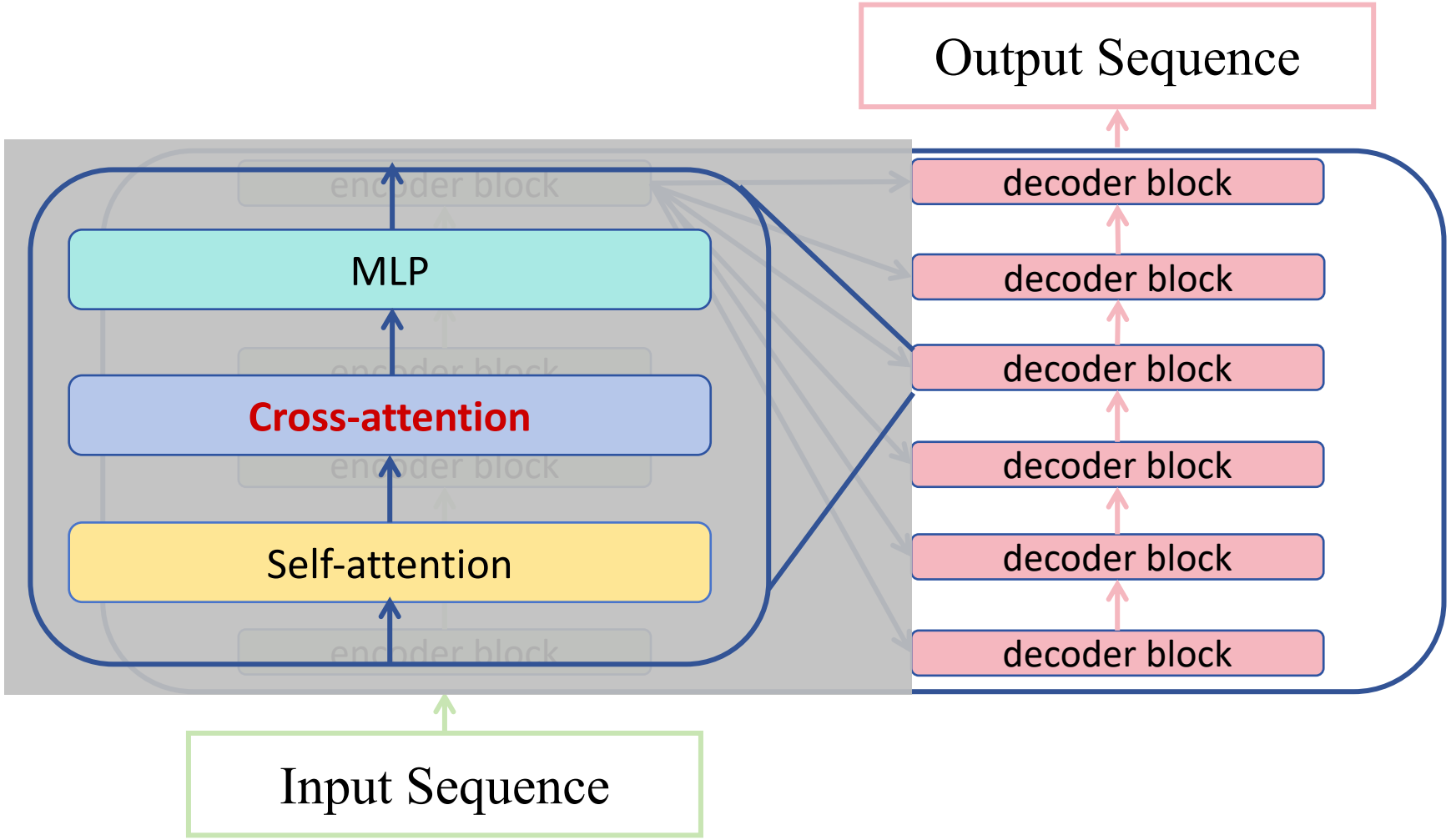
Transformer – Encoder

- Each encoder block consists of a self-attention layer and a MLP layer.



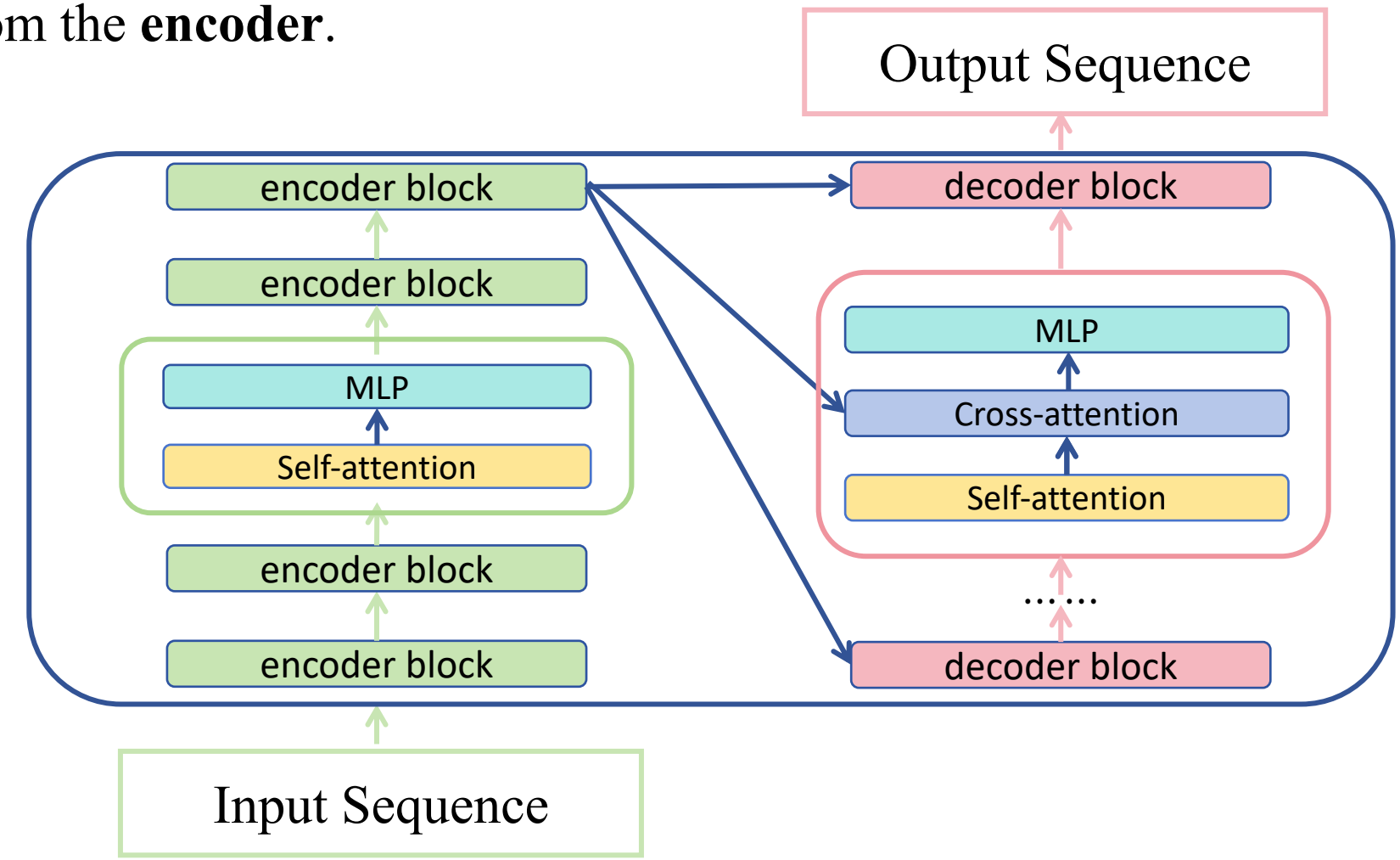
Transformer – Decoder

- Each decoder block consists of a self-attention layer, a cross-attention layer, and a MLP layer.



Transformer – Cross-attention

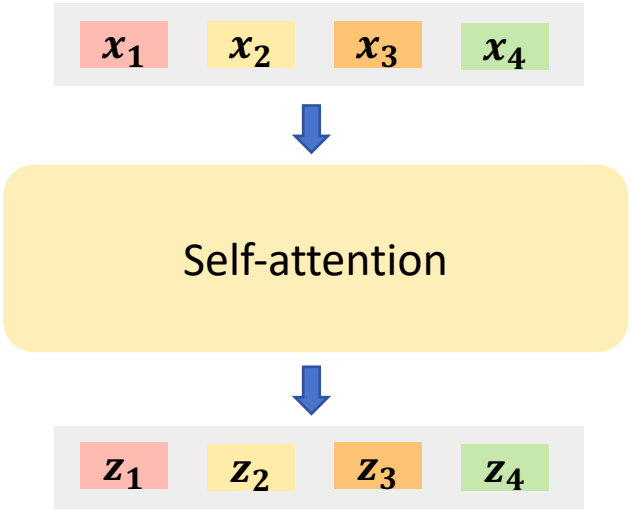
- **Self-attention**: the *query*, *key*, and *value* are taken from **the same sequence**.
- **Cross-attention**: the *query* is taken from the **decoder**, while the *key* and *value* are taken from the **encoder**.



Transformer – Positional Encoding

- **The limitation of attention:**
  - The attention for input sequence is entirely **parallelized**;
  - Lose the **sequential** information of input sequence.

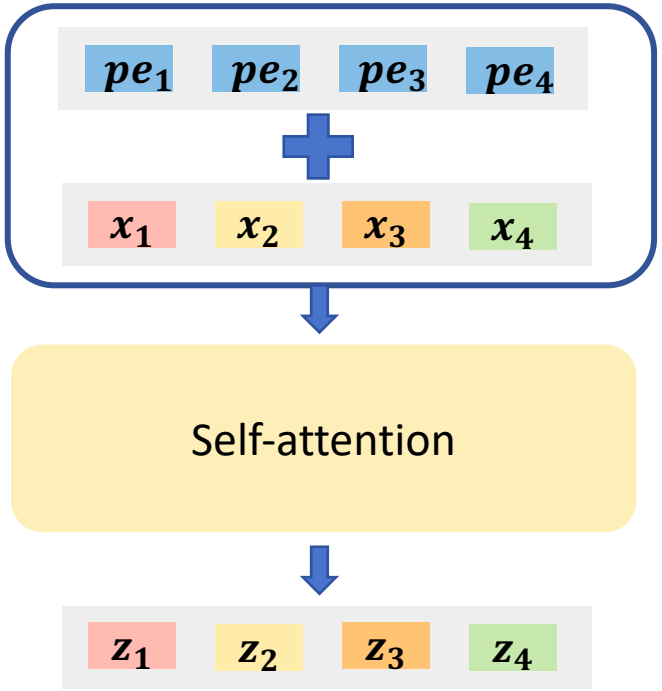
How should we know that x_1 is in front of x_4 ?



Transformer – Positional Encoding

- **The limitation of attention:**
  - The attention for input sequence is entirely **parallelized**;
  - Lose the **sequential** information of input sequence.

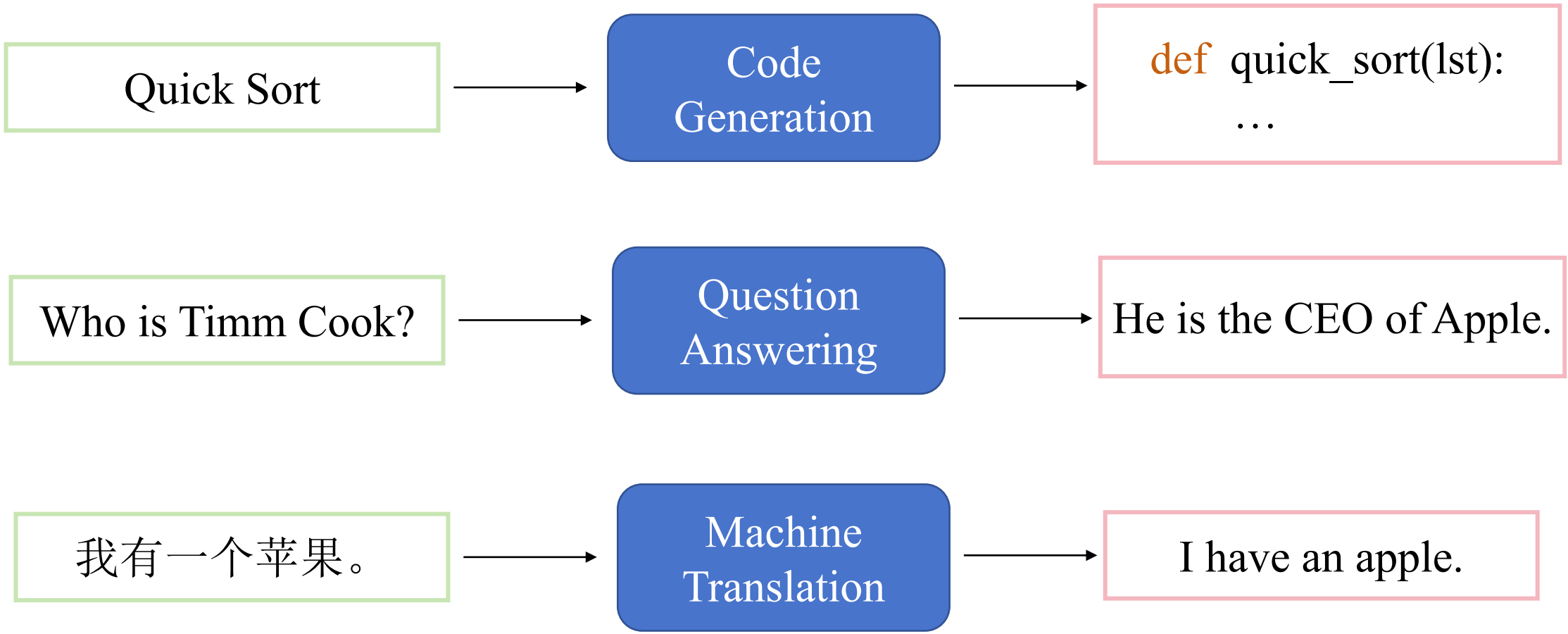
positional encoding is employed.



- Positional encodings are directly added to the input.

- **Various positional encoding:**
  - Absolute Positional Encoding (e.g., sine and cosine functions)
  - Learnable Positional Encoding
  - …

Transformer + Supervised Learning



Pre-train & Fine-tune

Pre-training: find good model parameters (initializations) with massive (unlabeled) data

Fine-tune: update model parameters with training data of downstream tasks

Pre-train & Fine-tune

Pre-training: find good model parameters (initializations) with massive (unlabeled) data

Fine-tune: update model parameters with training data of downstream tasks

Why this two-stage approach ?

Pre-train & Fine-tune

Pre-training: find good model parameters (initializations) with massive (unlabeled) data

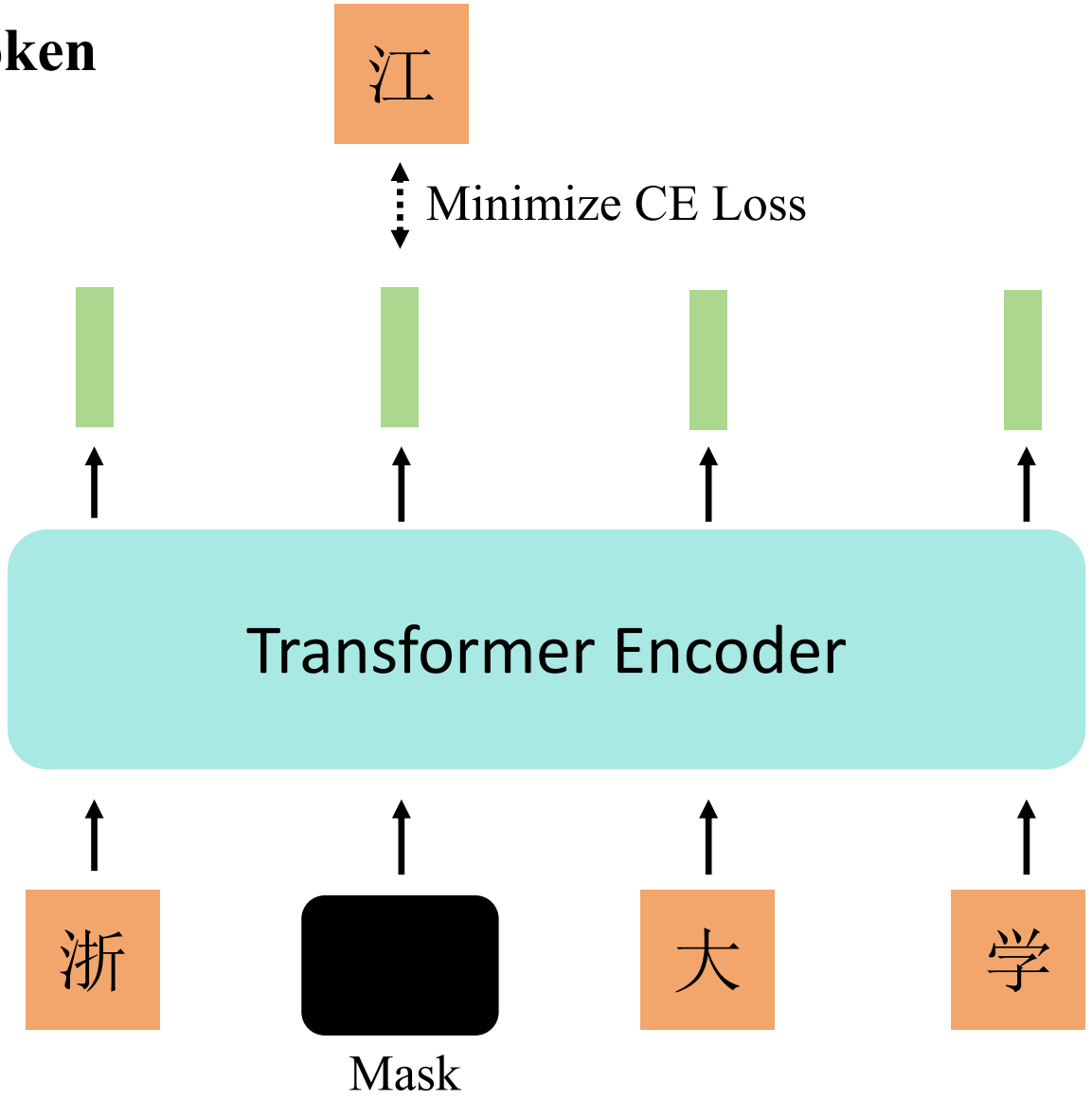
Fine-tune: update model parameters with training data of downstream tasks

Why this two-stage approach ?

- The cost of collecting data
- Human cognitive and learning processes
- Learning representations for language tokens
- …

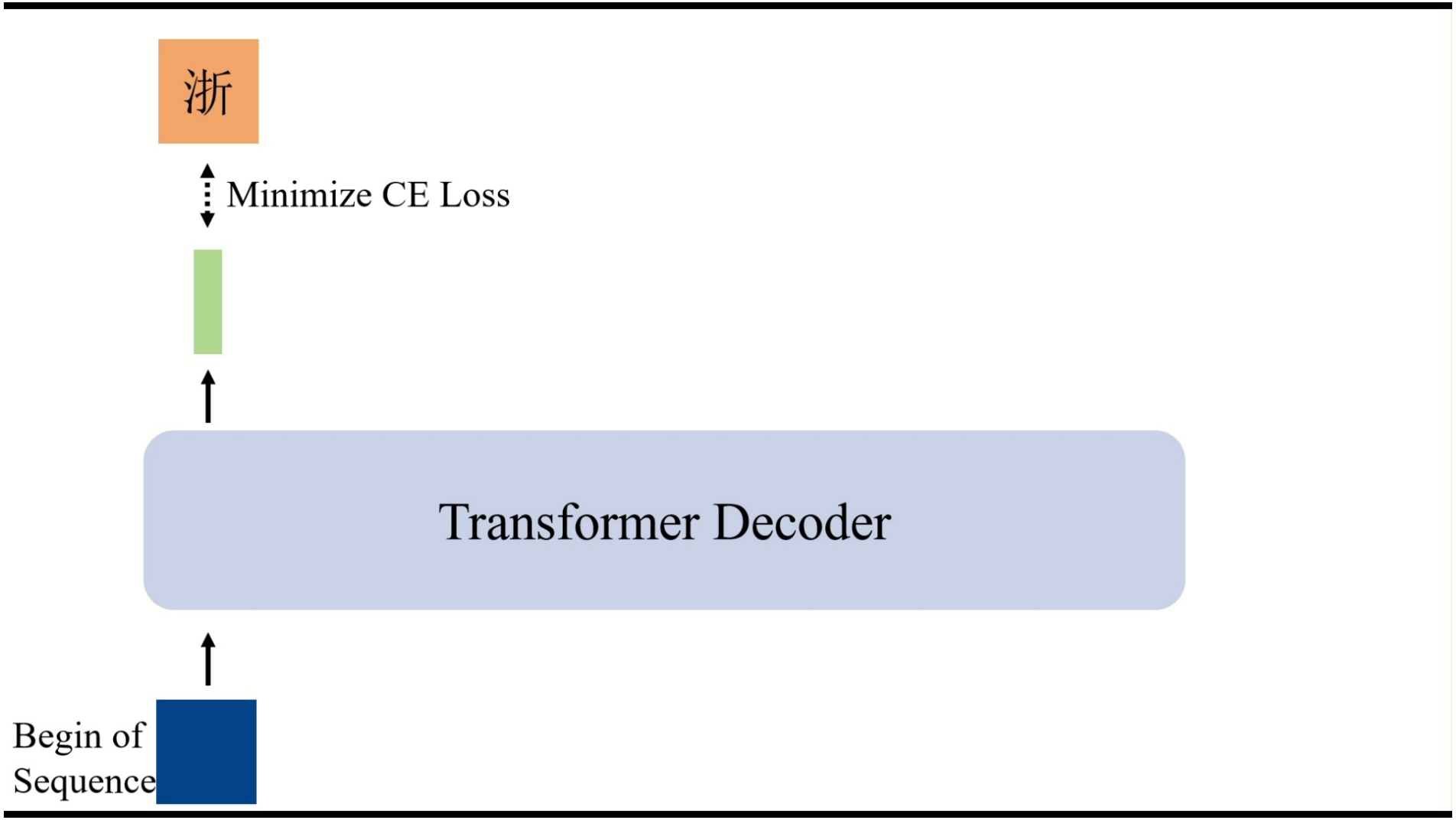
BERT (Bidirectional Encoder Representation from Transformers)

- **Predict the masked token**



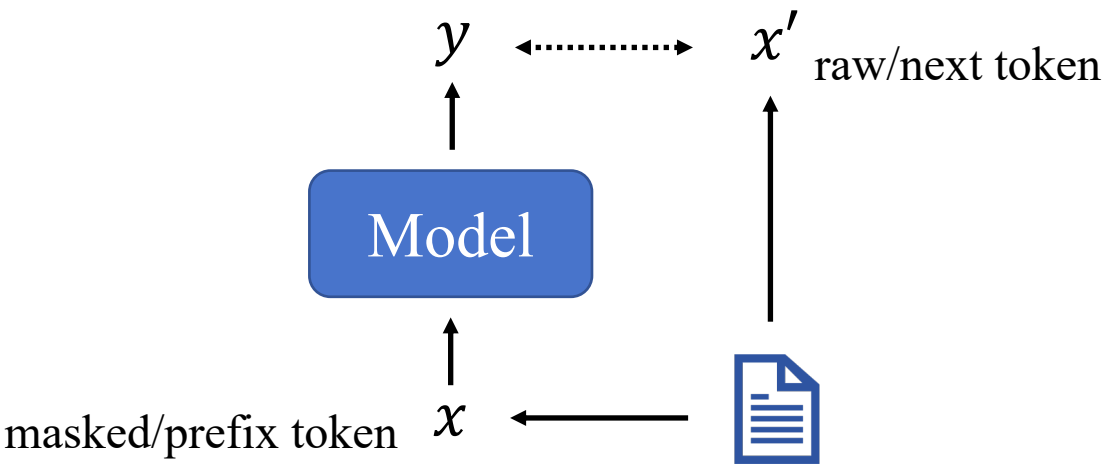
GPT (Generative Pre-trained Transformer)

- **Predict the next token**

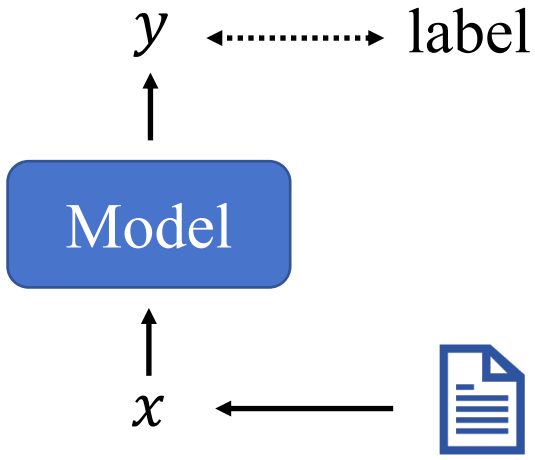


 Language models are few-shot learners, *NeurIPS'20*

Language Understanding



Self-supervised learning
(pre-train)



Supervised learning
(fine-tune)

By pre-training, the model can be used for a variety of NLP tasks (after fine-tuning) with **low training cost** (paired data, training time, etc.).

Language Understanding

Which type of pre-training do you think is better ?

Language Understanding

Which type of pre-training do you think is better ?

Before ChatGPT, BERT is more favored.

Now, ChatGPT is all you need.

Summary

Q1: Why we use LSTM (Gated RNN)?

A1: Addressing the long-term dependency and optimization problem.

Q2: Why we introduce attention mechanism?

A2: The long-term dependency of LSTM will still be lost

Q3: What is the most famous application of attention mechanism?

A3: Transformer (Attention is all you need).

Q4: What are the most popular pre-train methods for Transformer?

A4: BERT and GPT.