Sequence Modeling Methods and Applications

Prof. Mingkui Tan

SCUT Machine Intelligence Laboratory (SMIL)

Outline

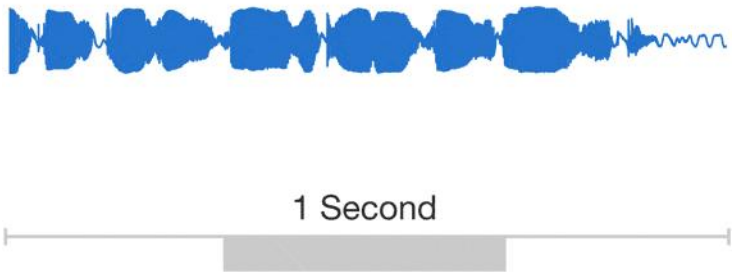
1. Backgrounds
2. Recurrent Neural Networks (RNNs)
3. Attention Mechanisms
4. Attention is All You Need: Transformer
5. Long Short-Term Memory (LSTM) Networks
6. BERT(Bidirectional Encoder Representation from Transformers)
7. Conclusions

Outline

Sequence Modeling

Definition:

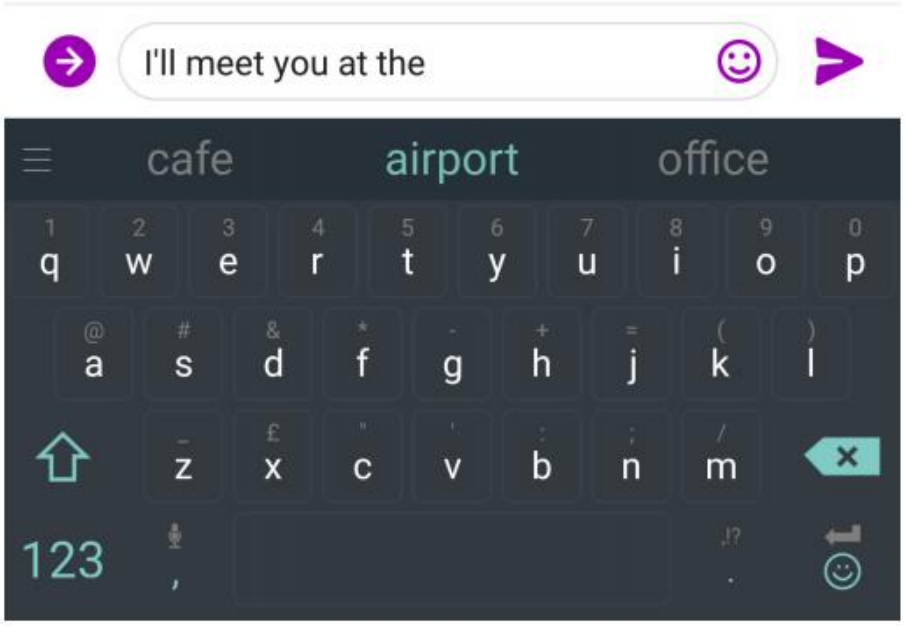
- Sequence Modeling is the task to model, interpret, make predictions about or generate any type of sequential data, such as audio, text, etc.



General Application

Auto-Completion:

- Auto-Completion is the task of predicting what word/letter comes next.

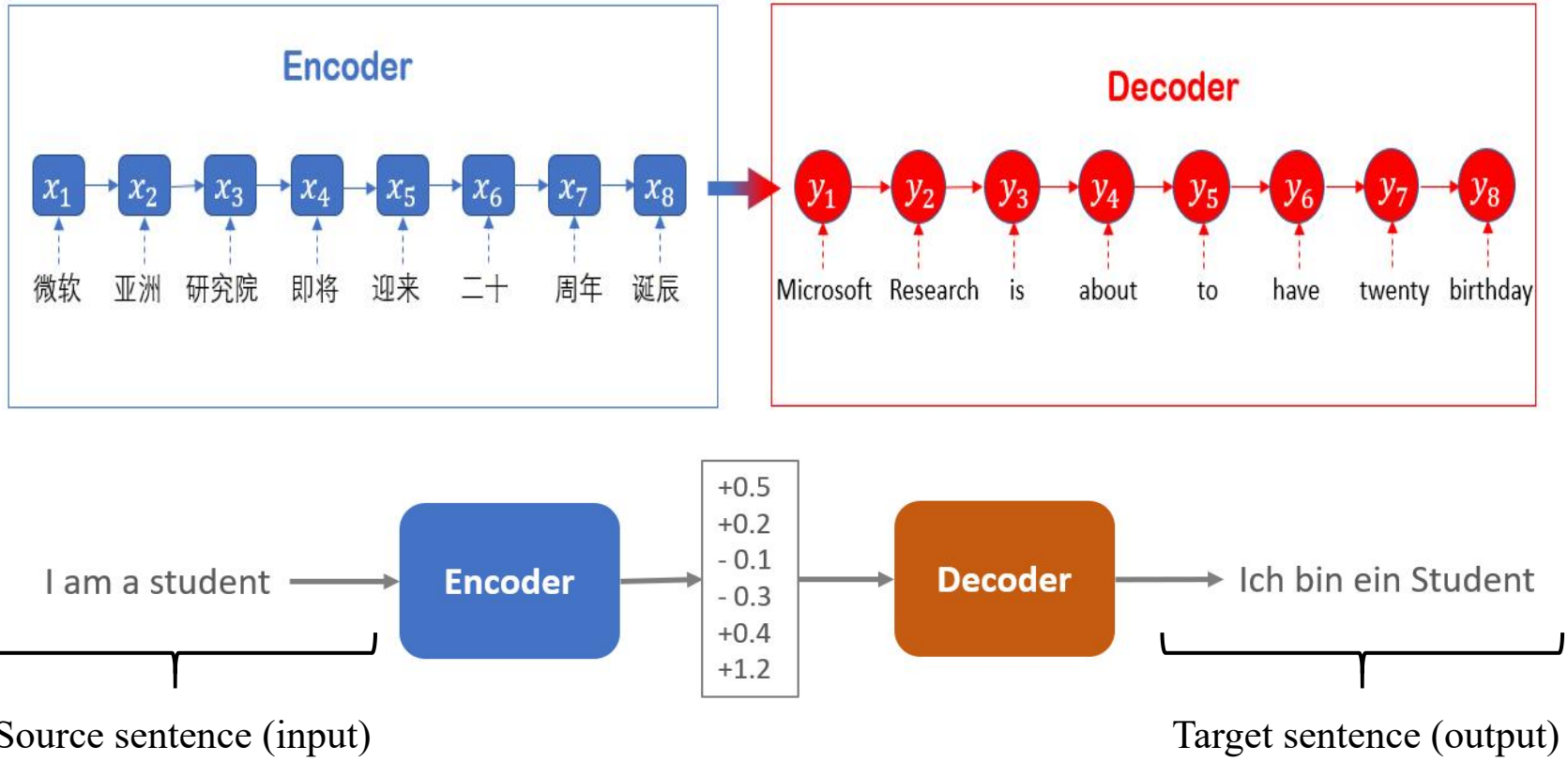


Challenges:

- The inputs and outputs have a sequential dependence.
- The current output is dependent on the previous input.

Neural Machine Translation (NMT)

- Definition: an approach to machine translation that uses an artificial neural network to predict the likelihood of a sequence of words.



Neural Machine Translation (NMT)

How to predict the likelihood of a sequence of words?

- NMT directly calculates $P(\boldsymbol{y}|\boldsymbol{x})$:

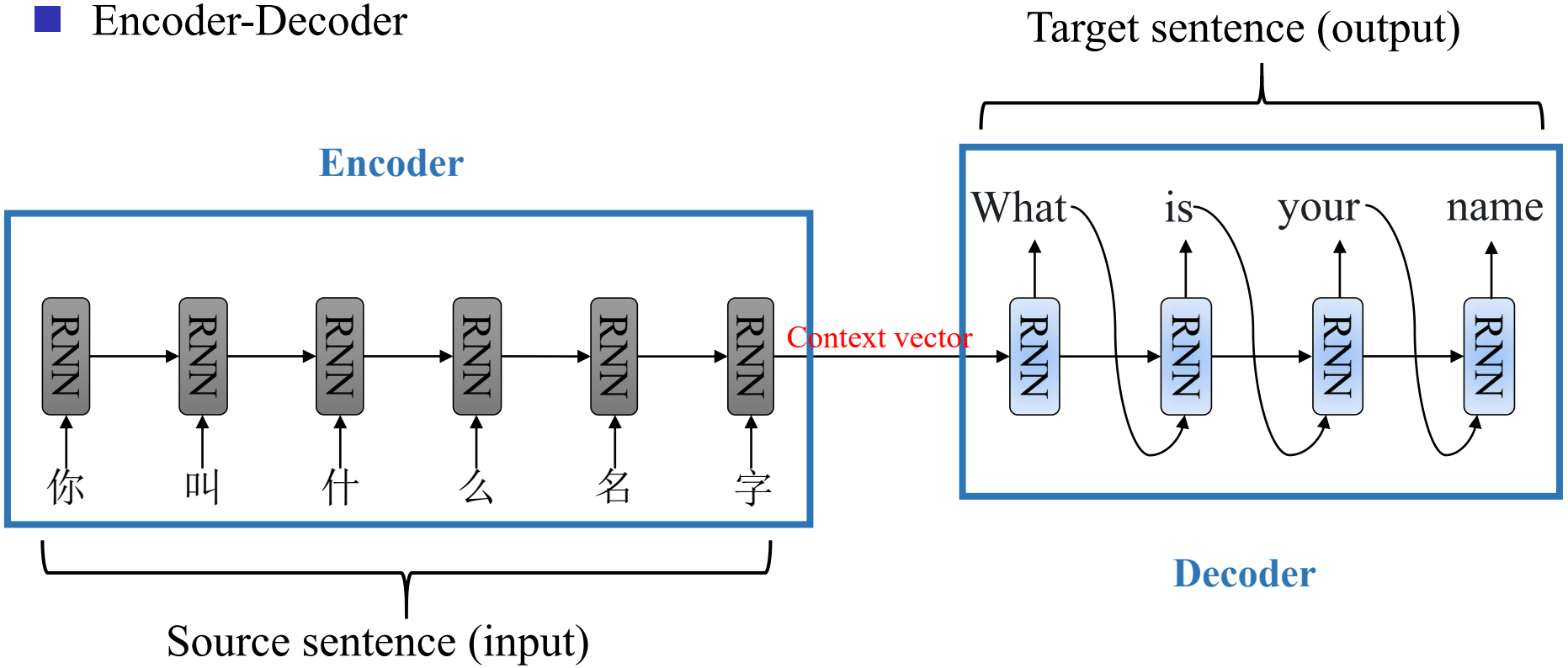
$$P(\boldsymbol{y}|\boldsymbol{x}) = P(\boldsymbol{y}_1|\boldsymbol{x})P(\boldsymbol{y}_2|\boldsymbol{y}_1,\boldsymbol{x})P(\boldsymbol{y}_3|\boldsymbol{y}_1,\boldsymbol{y}_2,\boldsymbol{x})\ldots P(\boldsymbol{y}_T|\boldsymbol{y}_1,\ldots,\boldsymbol{y}_{T-1},\boldsymbol{x})$$

Probability of next target word, given target words so far and souce sentence $\boldsymbol{x}$

- Input: encode of the source sentence $\boldsymbol{x}_1,\ldots,\boldsymbol{x}_T$.
- Output: generate target sentence $\boldsymbol{y}_1,\cdots,\boldsymbol{y}_T$.

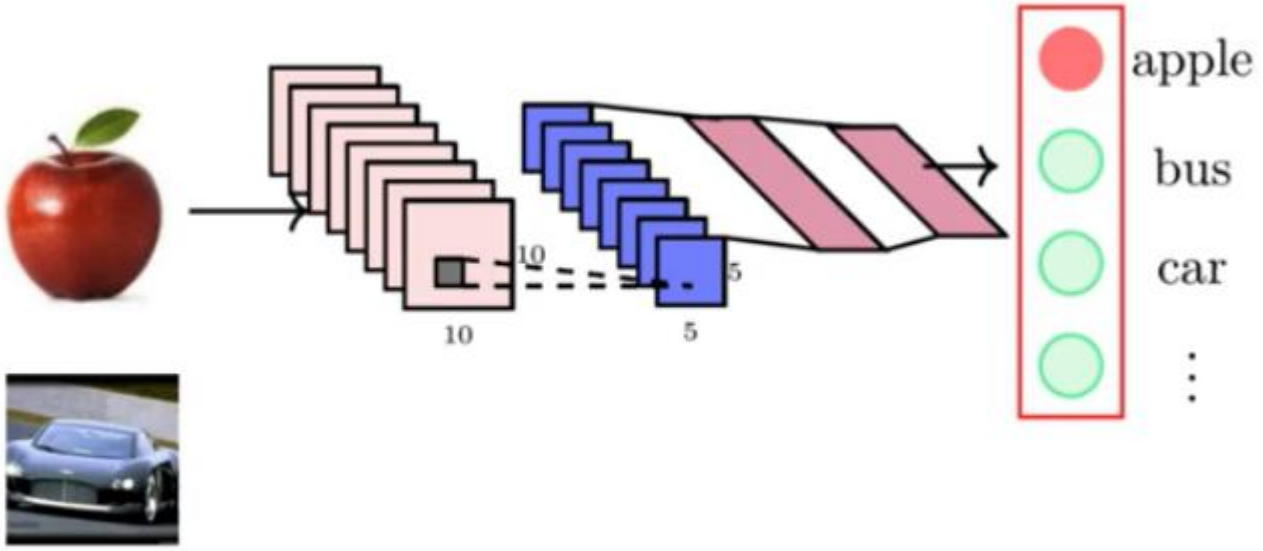
Neural Machine Translation (NMT)

- Encoder-Decoder



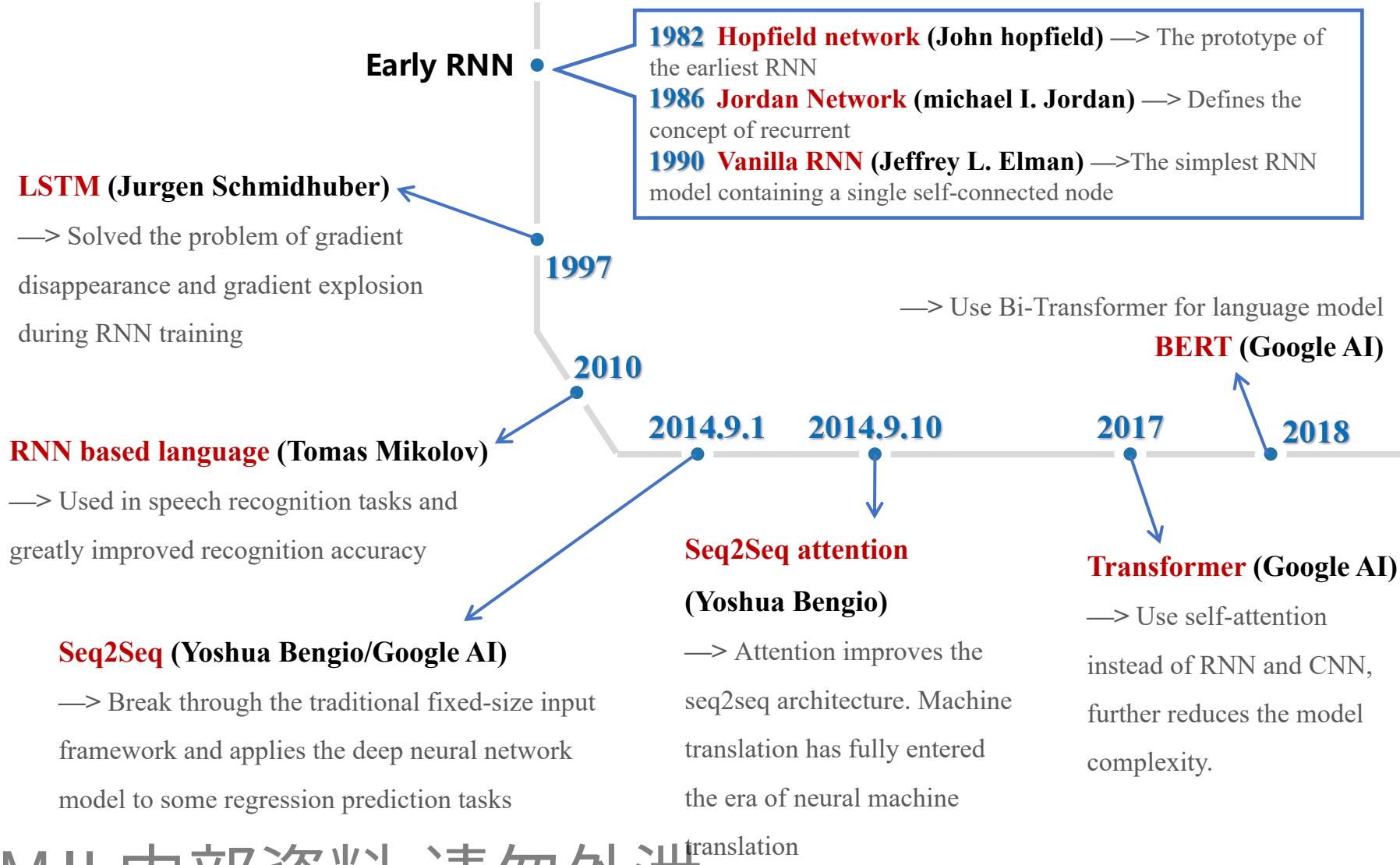
- Encoder: produce an encoding of the source sentence into a context vector.
- Decoder: a language Model that generates target sentence from the context vector iteratively.

Why Don't We Use CNN



- Traditional Convolution Neural Networks (CNNs) only consider the current input and all inputs are independent of each other.
- CNN is more suitable for image processing.

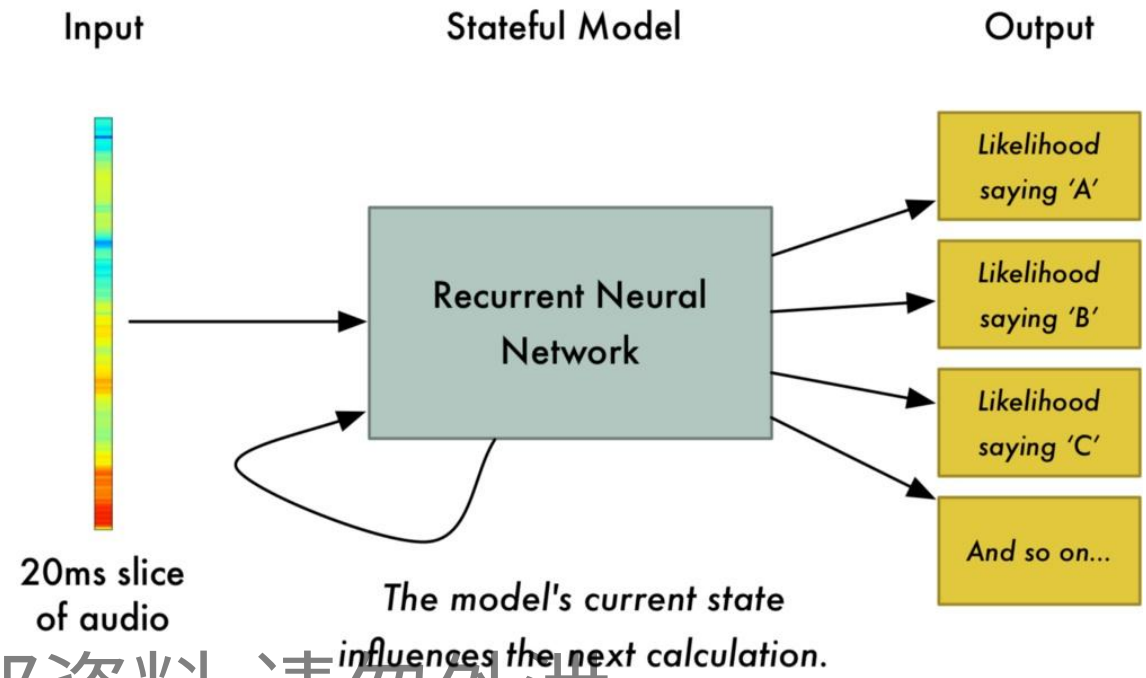
Evolution of Sequence Modeling methods



Sequence to Sequence Model Applications

Speech Recognition

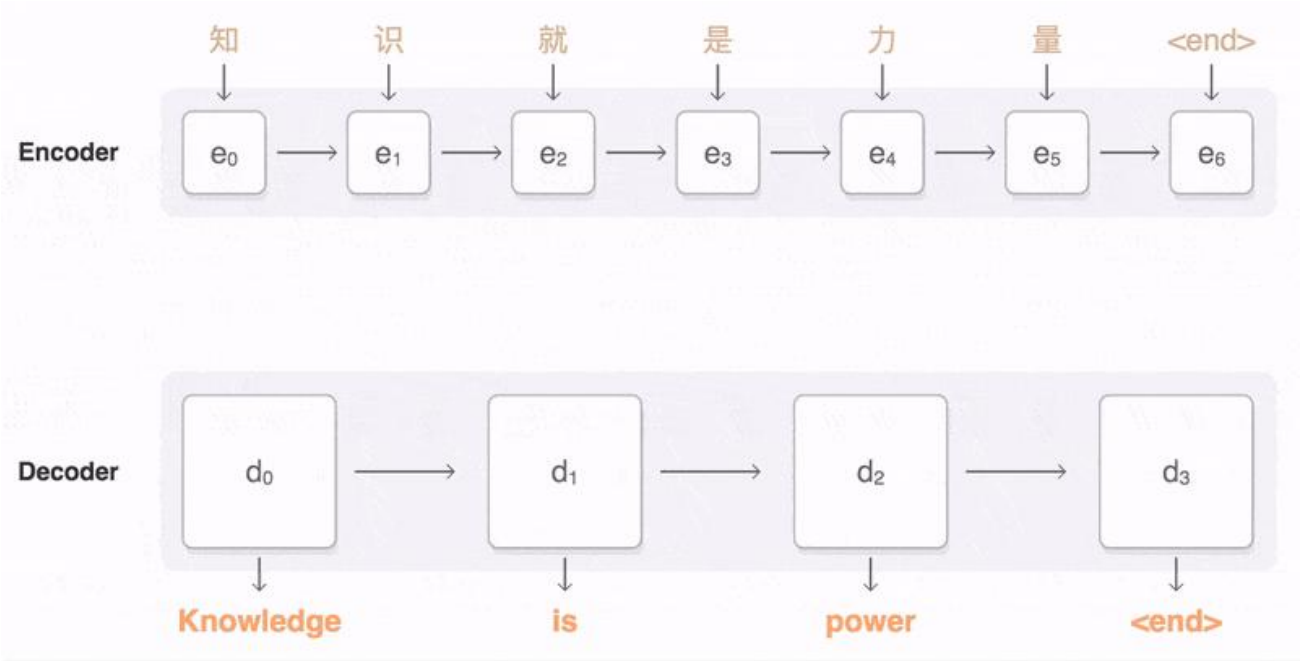
- **Definition:** Speech recognition aims to convert spoken language into readable text.
- **Difficulty:** Vocabulary is hard to recognize if the speech contains confusing words.



Sequence to Sequence Model Applications

Machine Translation

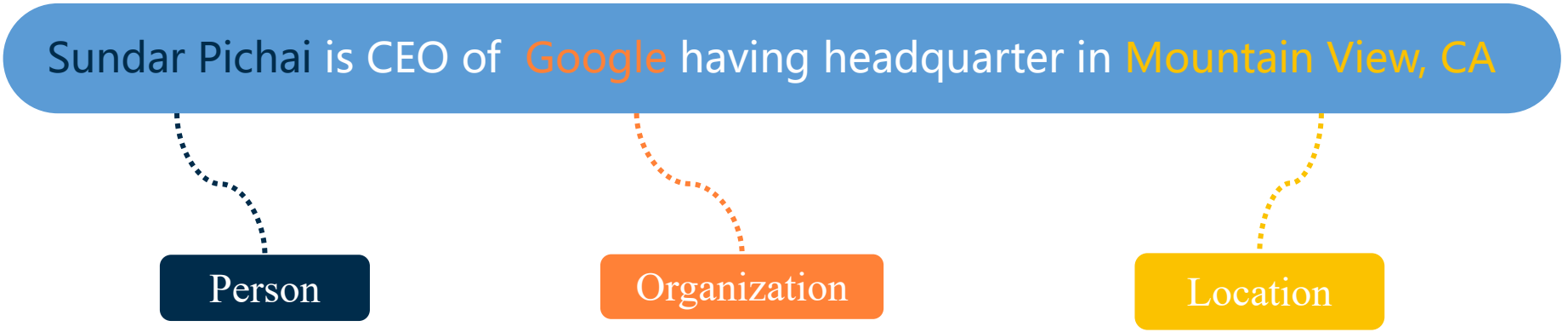
- **Definition:** Machine translation aims to translate text from one language to another.
- **Difficulties:** Word sense disambiguation and Syntactic disambiguation



Sequence to Sequence Model Applications

Named Entity Recognition

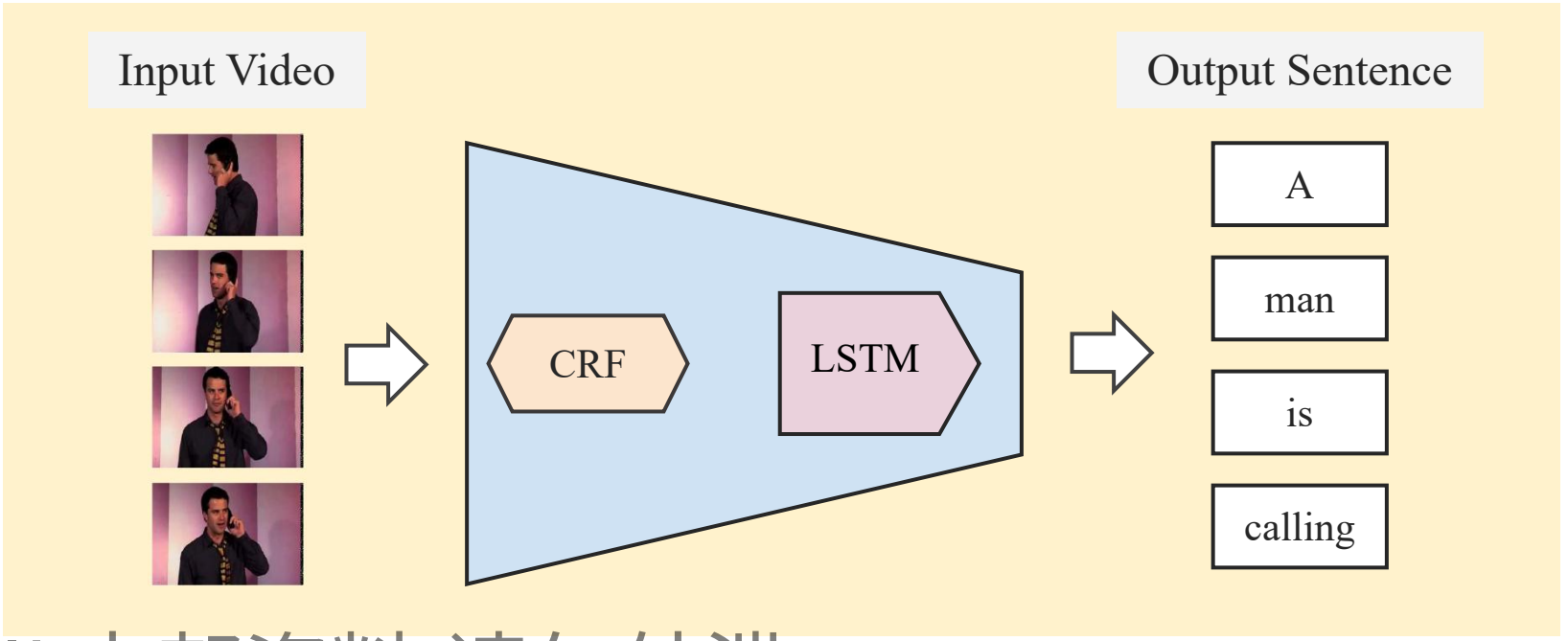
- **Definition:** Named-entity recognition seeks to locate and classify named entities mentioned in unstructured text into pre-defined categories.
- **Challenge:** Devise models to deal with linguistically complex contexts.



Sequence to Sequence Model Applications

Video Captioning

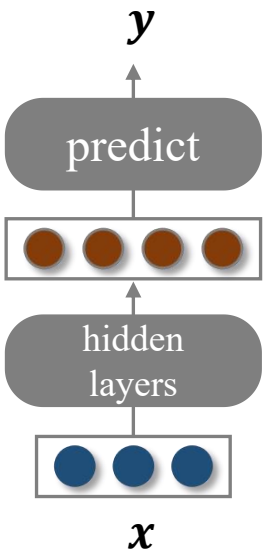
- **Definition:** Video Captioning aims to generate text descriptions for videos.
- **Challenge:** Model variable length videos with a fixed number of parameters.



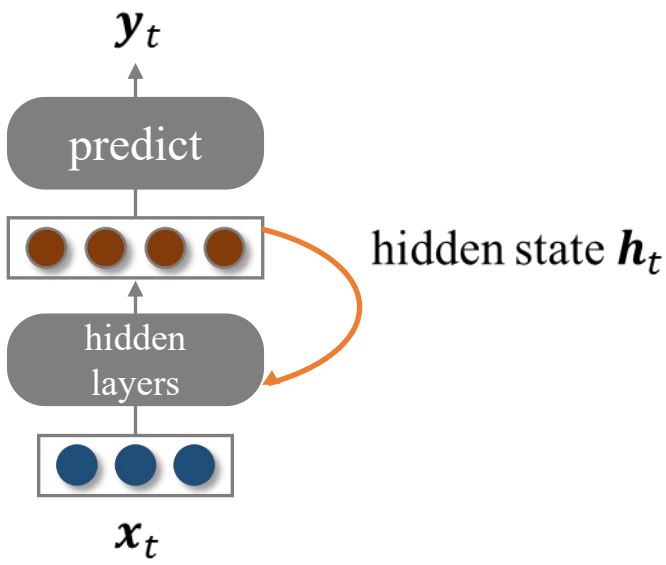
Outline

RNNs

RNNs are capable of remembering information in time dimension.



Feedforward Networks



Recurrent Networks

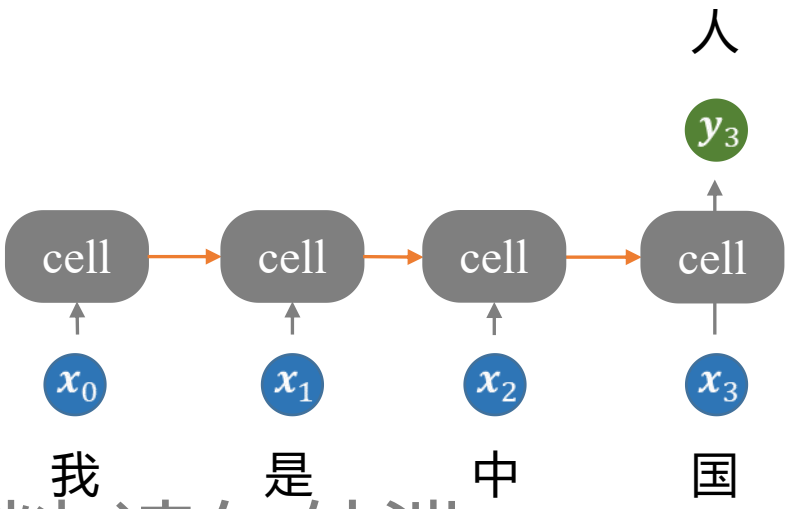
- Input: sequence of words $x_1, \cdots, x_t$
- Output: sequence of words $y_1, \ldots, y_t$

RNNs

- $\boldsymbol{x}_t$ ← input: sequence of words
- $\boldsymbol{y}_t$ ← output: sequence of words
- $\boldsymbol{h}_t$ ← hidden node
- $\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}$ ← coefficients
- $\boldsymbol{E}$ ← embedding matrix

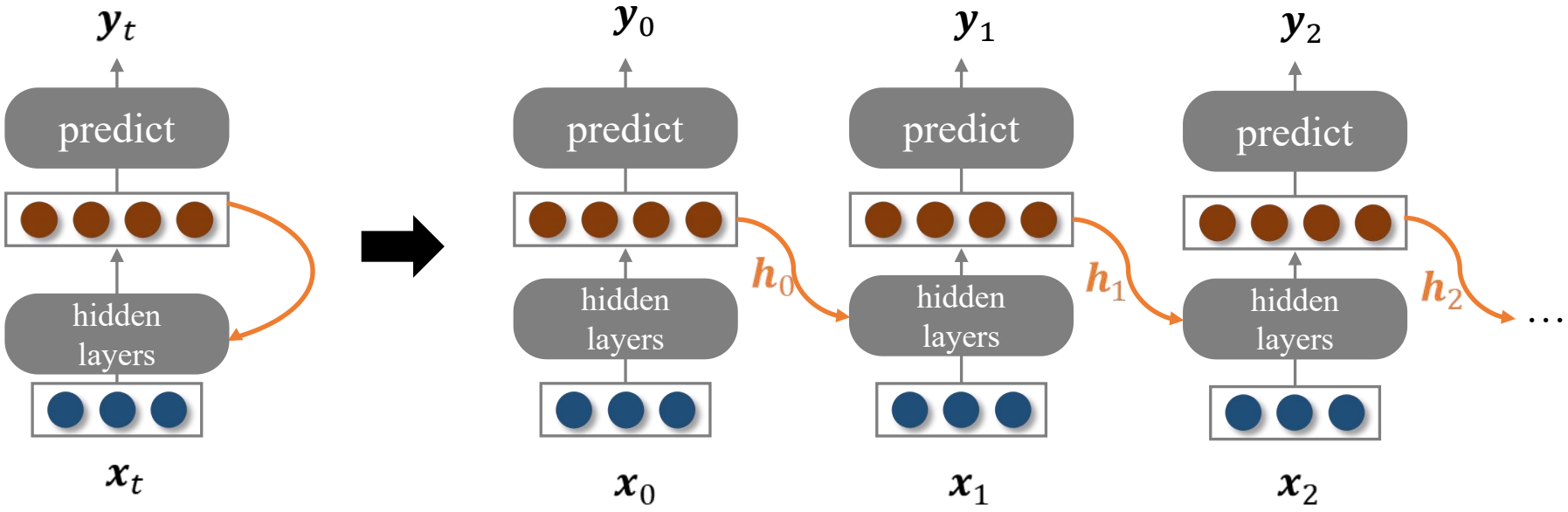
- Predict a single output:

Maximize the likely-hood of $\mathbf{P}(\boldsymbol{y}_3 | \boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3)$



RNNs

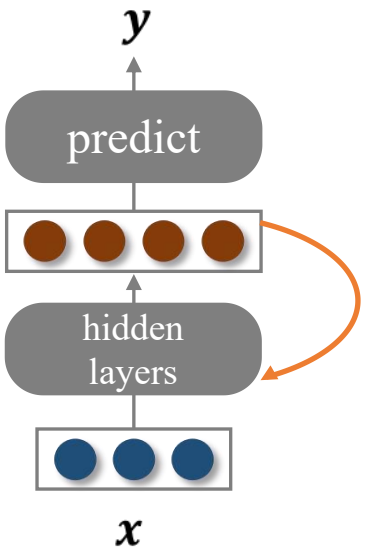
Unroll the recurrent loop:



A RNN can be thought of as multiple copies of the same network, each passing a message to a successor.

RNNs

We can model the recurrence formula by a sequence of vectors $\boldsymbol{x}$ at every time step:



old hidden node

$$\boldsymbol{h}_t = g(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)$$

new hidden node

some activation function

input vector at *t*h time step

The same function and the same set of parameters are used at every time step.

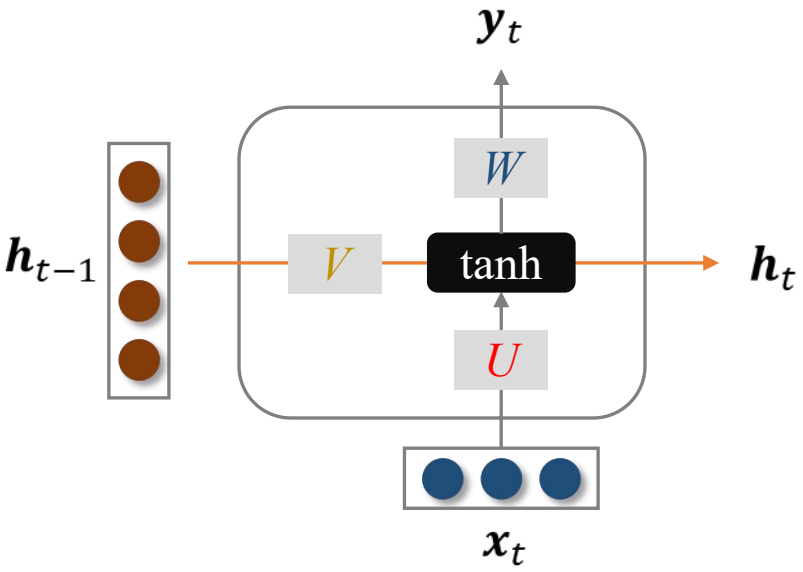
RNNs

A typical RNN cell and time step t:

$$\boldsymbol{h}_t = g(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)$$

$$\boldsymbol{h}_t = \tanh(\boldsymbol{U}^{\mathbf{T}}\boldsymbol{x}_t + \boldsymbol{V}^{\mathbf{T}}\boldsymbol{h}_{t-1} + b)$$

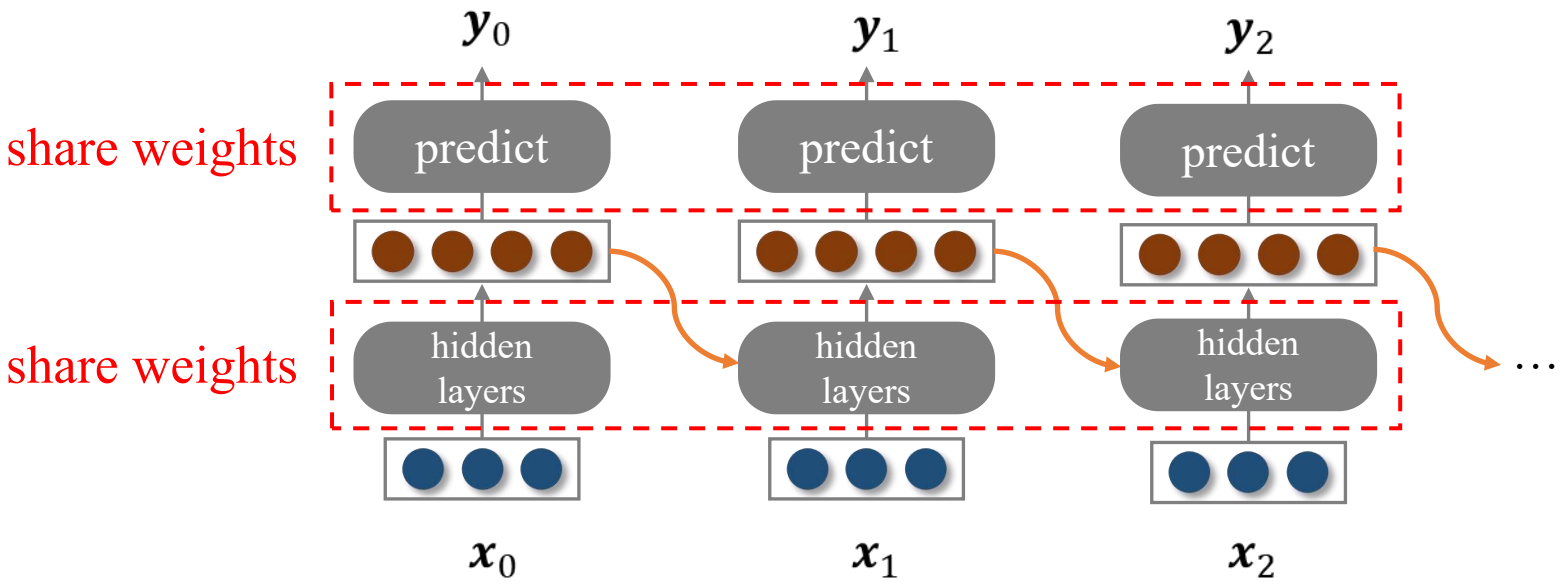
$$\boldsymbol{y}_t = \boldsymbol{W}^{\mathbf{T}}\boldsymbol{h}_t$$



- The tanh function is considered as the activation function.
- We can replace the tanh function with sigmoid or ReLU function.

RNNs

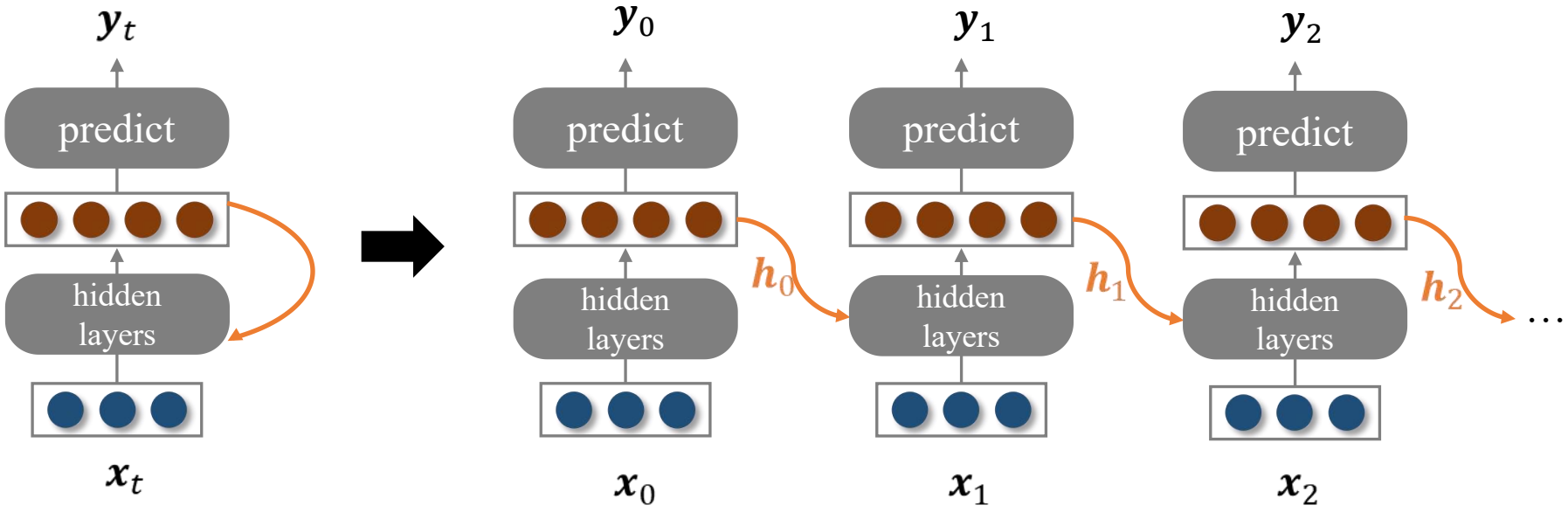
Unroll the recurrent loop:



A RNN can be thought of as multiple copies of the same network, each passing a message to a successor.

RNNs

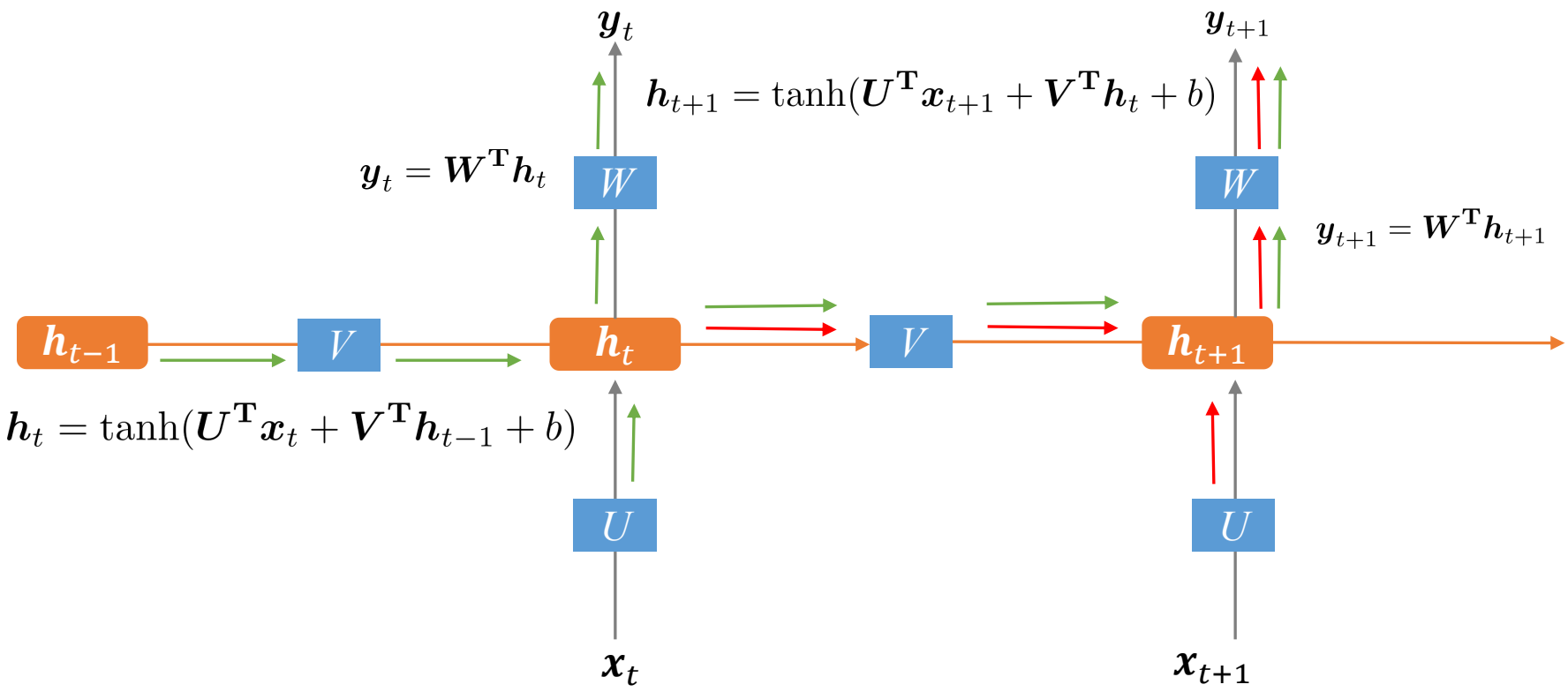
Unroll the recurrent loop:



A RNN can be thought of as multiple copies of the same network, each passing a message to a successor.

Forward Propagation: Prediction for Networks

Forward Propagation Through Time (FPTT):



Forward Propagation: Prediction for Networks

- Given input a seqence of words $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_T$, the forward propagation of a RNN cell is to predict the output $\boldsymbol{y}_1$ of $\boldsymbol{x}_1$.

- Detailed prediction process:

  for t from 1 to T do

  $$\boldsymbol{u}_t = \boldsymbol{U}^{\mathbf{T}} \boldsymbol{x}_t + \boldsymbol{V}^{\mathbf{T}} \boldsymbol{h}_{t-1} + b$$

  $$\boldsymbol{h}_t = \sigma(\boldsymbol{u}_t)$$

  $$\boldsymbol{y}_t = \boldsymbol{W}^{\mathbf{T}} \boldsymbol{h}_t$$

  end for

Example:

$\sigma = \tanh(\cdot)$

σ : some activation function

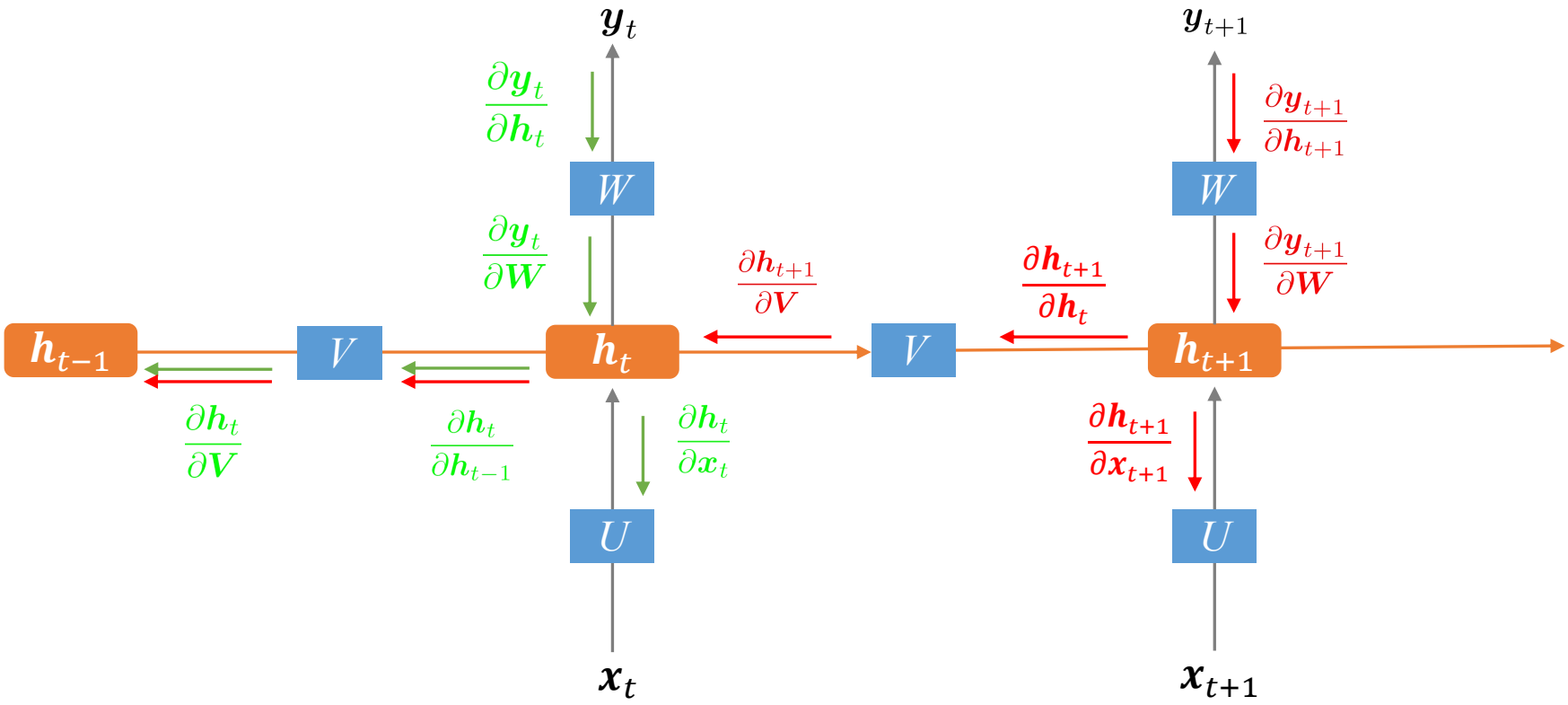
- Total loss: $L(y, \hat{y}) = \sum_{i=1}^{m} \sum_{t=1}^{T} L_i(y_t, \hat{y}_t)$

$\hat{\boldsymbol{y}}_t$: predicted label

$\boldsymbol{y}_t$: ground-truth label

Backpropagation of Neural Networks

Backward Propagation Through Time (BPTT):



Backpropagation of Neural Networks

What's the derivative of $L(\boldsymbol{y}, \hat{\boldsymbol{y}})$, the repeated weight matrix $\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}$?

- Let us resort to Chain Rule to compute the gradient.

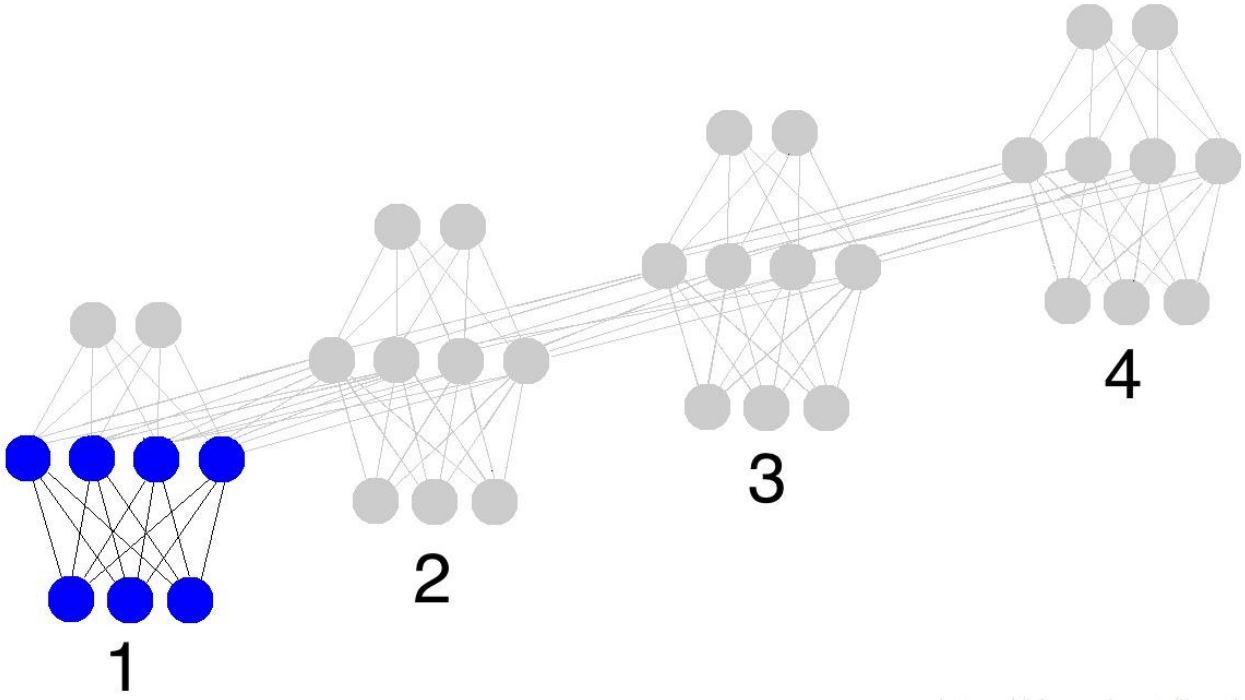
$$\frac{\partial L_3(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial W} = \frac{\partial L_3(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial y_3} \frac{\partial y_3}{\partial W}$$

Use $L_3(\boldsymbol{y}, \hat{\boldsymbol{y}})$ as an example

$$\frac{\partial L_3(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial U} = \sum_{k=0}^{3} \frac{\partial L_3(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial y_3} \frac{\partial y_3}{\partial h_3} \frac{\partial h_3}{\partial h_k} \frac{\partial h_k}{\partial U}$$

$$\frac{\partial L_3(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial V} = \sum_{k=0}^{3} \frac{\partial L_3(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial y_3} \frac{\partial y_3}{\partial h_3} \frac{\partial h_3}{\partial h_k} \frac{\partial h_k}{\partial V}$$

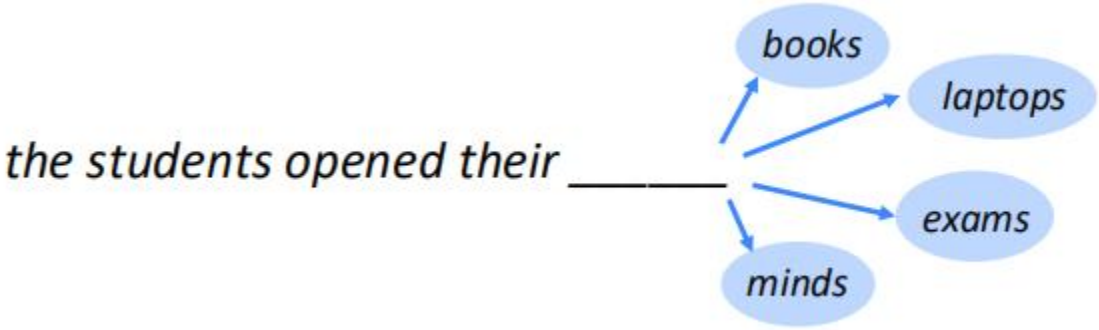
Backward Propagation Through Time (BPTT):



Black represents the predicted value, yellow represents the prediction error, and dark yellow represents gradient back propagation.

Application: Language Modeling (LM)

- Language Modeling: predict the coming word basis on a piece of words.



- Given a sequence of words $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_t$, compute the probability distribution of the next word $\boldsymbol{x}_{t+1}$:

$$\mathbf{P}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t)$$

where $\boldsymbol{x}_{t+1}$ can be any word in the vocabulary $V = \{\mathbf{w}_1, \cdots, \mathbf{w}_{|v|}\}$.

- We need to consider lots of words at a time if we want to model language well.

Training RNN Base on a Language Model

- output distribution

$$\hat{\boldsymbol{y}}_t = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}_t + \boldsymbol{b_2}\right) \in \mathbb{R}^{|V|}$$

- hidden states

$$\boldsymbol{h}_t = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}_{t-1} + \boldsymbol{W}_e e_t + \boldsymbol{b_1}\right)$$

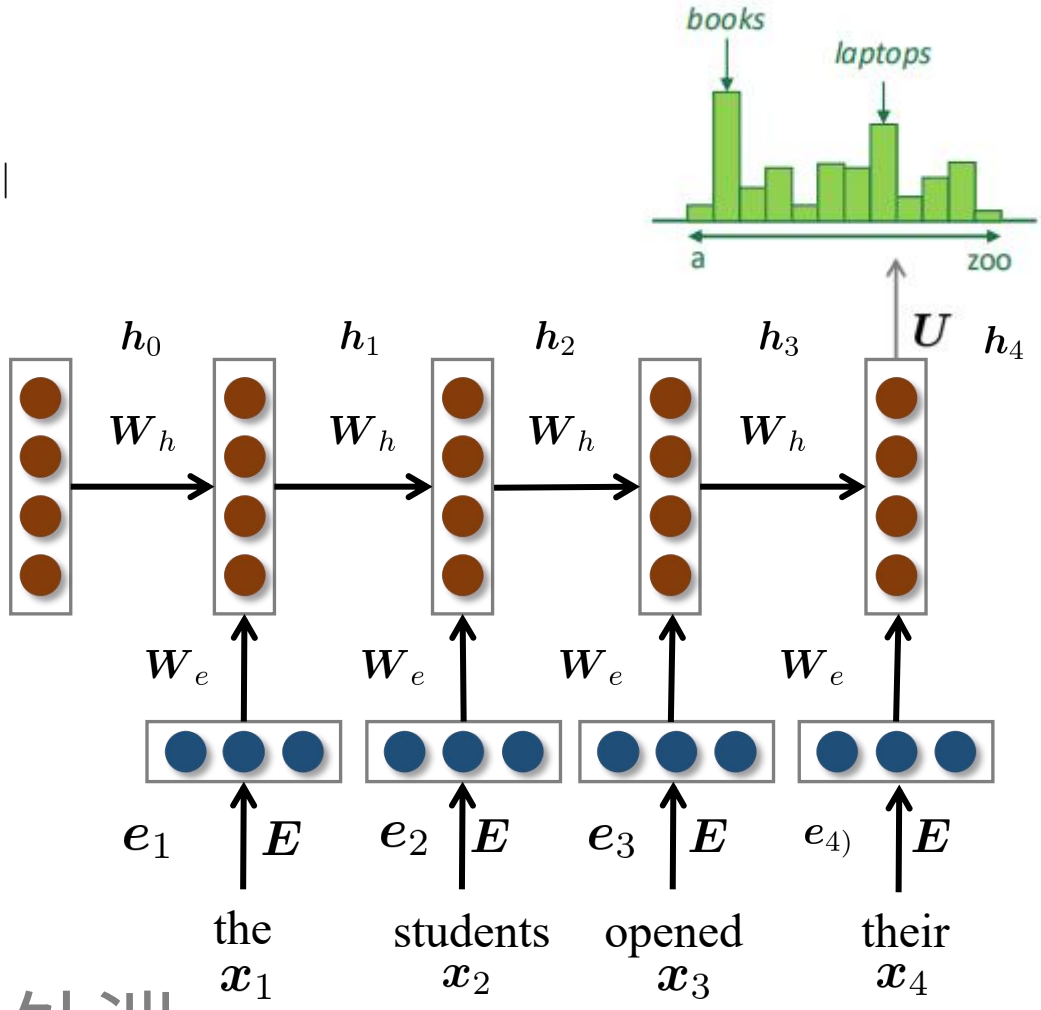
- word embedding

$$\boldsymbol{e}_t = \boldsymbol{E}\boldsymbol{x}_t$$

- words/one-hot vectors

$$\boldsymbol{x}_t \in \mathbb{R}^{|V|}$$

$\hat{\boldsymbol{y}}_4 = P(\boldsymbol{x}_5|\text{the students opened their})$



Training RNN Base on a Language Model

- Get a big corpus of text which is a sequence of words $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T$.
- Feed into RNN-LM, compute output distribution $\hat{\boldsymbol{y}}_t$ for every step t .
- Apply the cross-entropy function between predicted probability distribution $\hat{\boldsymbol{y}}_t$, and the true next word $\boldsymbol{y}_t$:

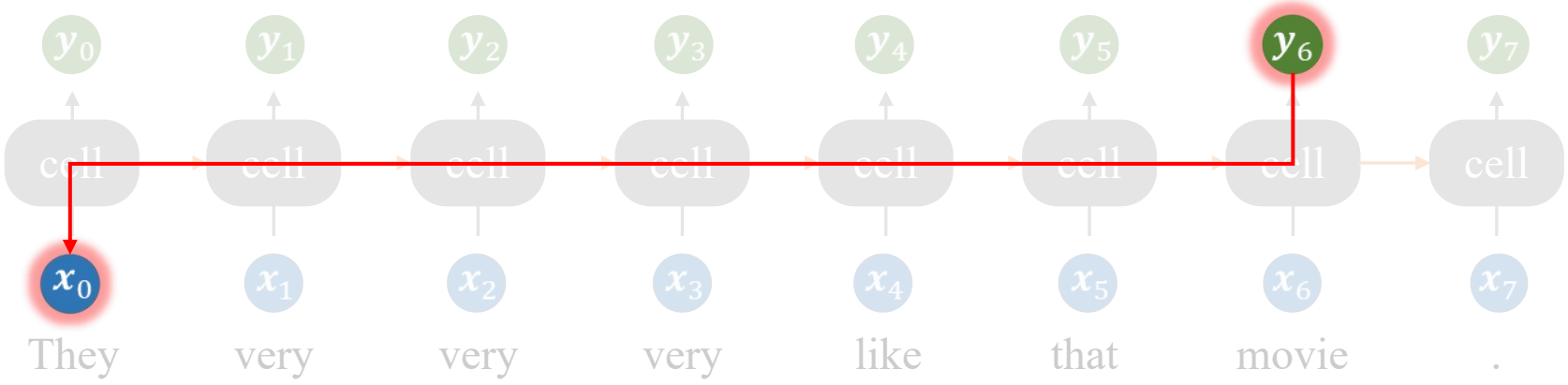
$$J_t(\theta) = CE(\boldsymbol{y}_t, \hat{\boldsymbol{y}}_t) = -\sum \boldsymbol{y}_t \log \hat{\boldsymbol{y}}_t.$$

- Total loss for the entire training set:

$$J(\theta) = \frac{1}{T} \sum_{\mathrm{i}=1}^{\mathrm{m}} \sum_{t=1}^{T} J_{i,t}(\theta)$$

Problems with RNN

The problem of long-term dependencies:



$$y_6 = W_3 \cdot \tanh(W_1 \cdot x_6 + W_2 \cdot \tanh(\cdots W_2 \cdot \tanh(W_1 \cdot x_0 + W_2 \cdot h_0 + b) \cdots) + b)$$

6 times!

Problems with RNN

The problem of long-term dependencies:

The gradient of ε_t on $\boldsymbol{x}_k$ $(k < t)$:

$$\frac{\partial \varepsilon_t}{\partial \boldsymbol{x}_k} = \boldsymbol{W}_3 \cdot \left(\prod_{i=k}^{t} \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{h}_{i-1}}\right) \cdot \frac{\partial \boldsymbol{h}_k}{\partial \boldsymbol{x}_k} = \boldsymbol{W}_3 \cdot \left(\prod_{i=k}^{t} \boldsymbol{W}_2 \cdot diag\left(\tanh'(\boldsymbol{h}_i)\right)\right) \cdot \boldsymbol{W}_1$$

can vanish or explode due to the Matrix-chain multiplication.

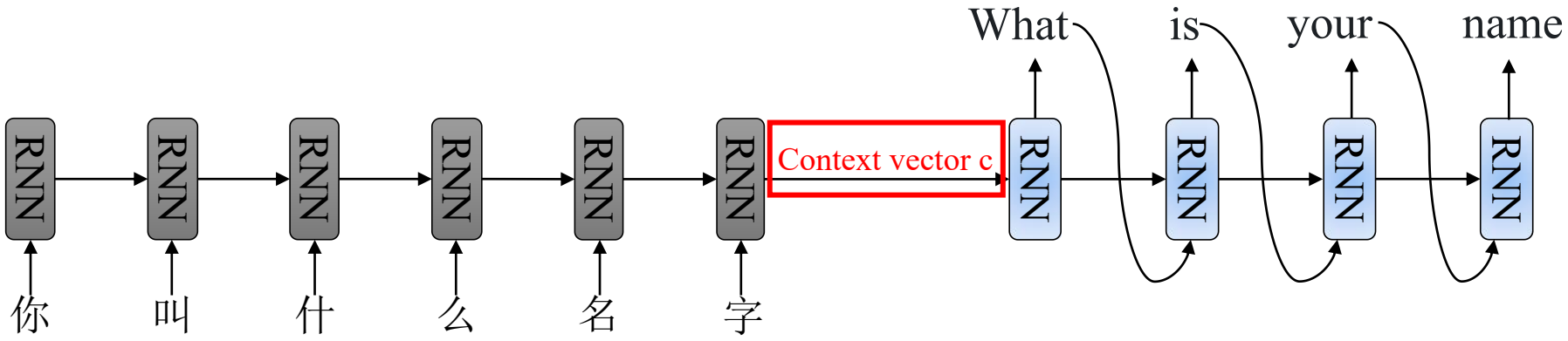
- Lots of operations are hard to capture in the hidden state path.

Outline

1. Backgrounds
2. Recurrent Neural Networks (RNNs)
3. Attention Mechanisms
4. Attention is All You Need: Transformer
5. Long Short-Term Memory (LSTM) Networks
6. BERT(Bidirectional Encoder Representation from Transformers)
7. Conclusions

Motivation

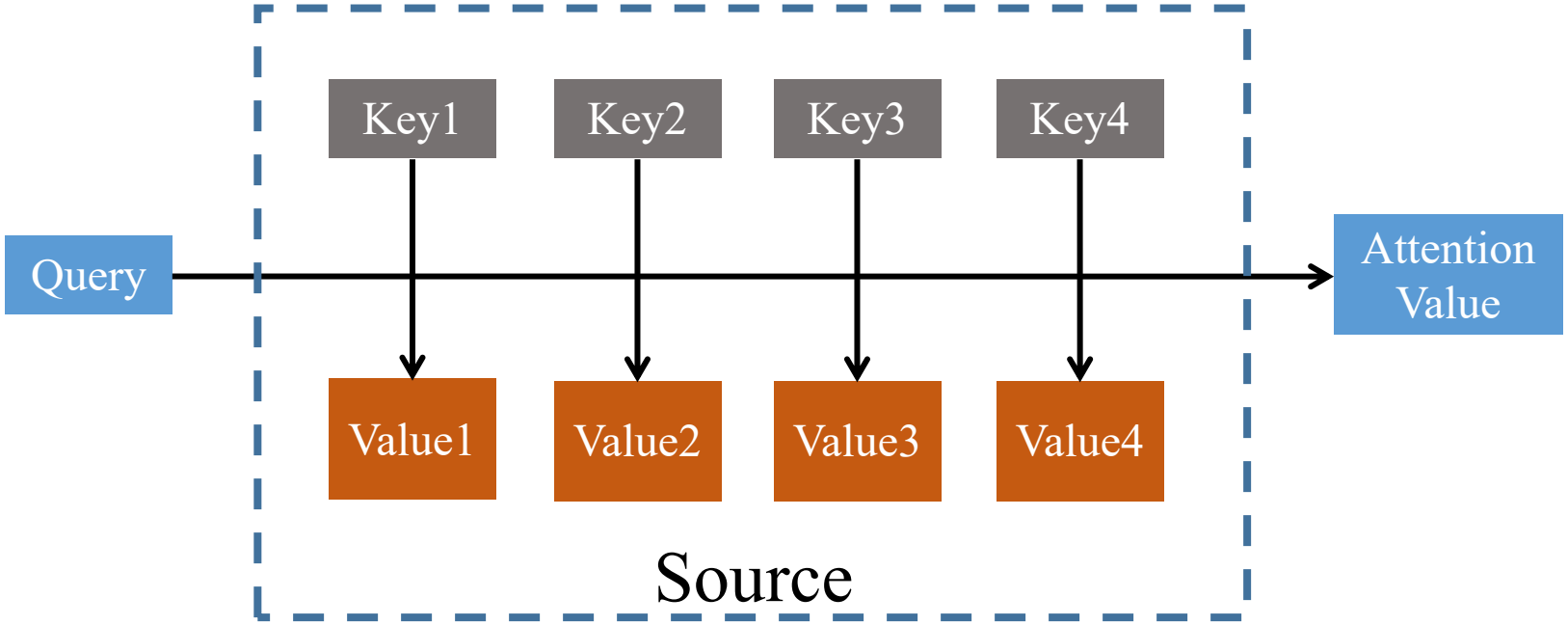
Compress all the necessary information of a source sentence into a static context vector $\boldsymbol{c}$.



- The problem of long-term dependencies: the translation effect will deteriorate rapidly as the length of text increases.

Motivation

On each step of the decoder, use direct connection to the encoder to focus on a particular part of the source sequence.



$$\text{Attention}(\text{Query}, \text{Source}) = \sum_{i=1}^{t} Similarity(Query, Key_i) * Value_i$$

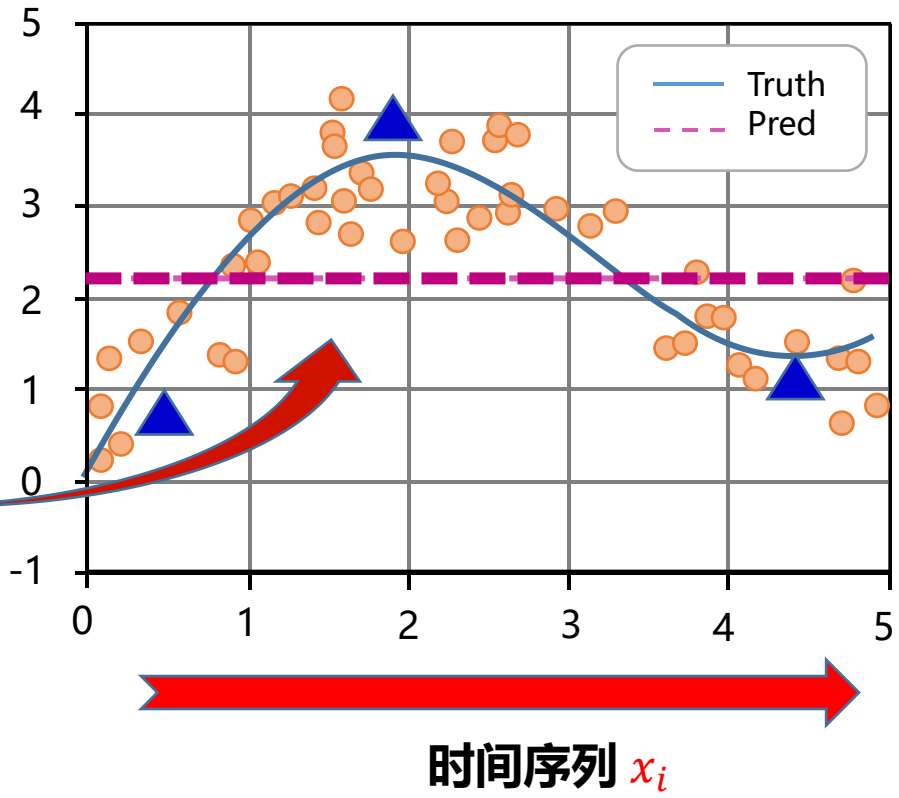
Motivation of Attention

Considering a function fitting task: **Ground Truth:** $y_i = 2\sin(x_i) + x_i^{0.8}$

Linear model $y = w'x$ no longer works

Simple Prediction for a new point x :

$$f(x) = \sum_{i=1}^{n} y_i$$



Simply using the **average of y_i** as prediction will lead to **very poor performance!**

Motivation of Attention

Considering a function fitting task: **Ground Truth:** $y_i = 2\sin(x_i) + x_i^{0.8}$

Weighted Predation for a new point x :

$$f(x) = \sum_{i=1}^{n} \alpha_i \, y_i$$

How to compute α_i?

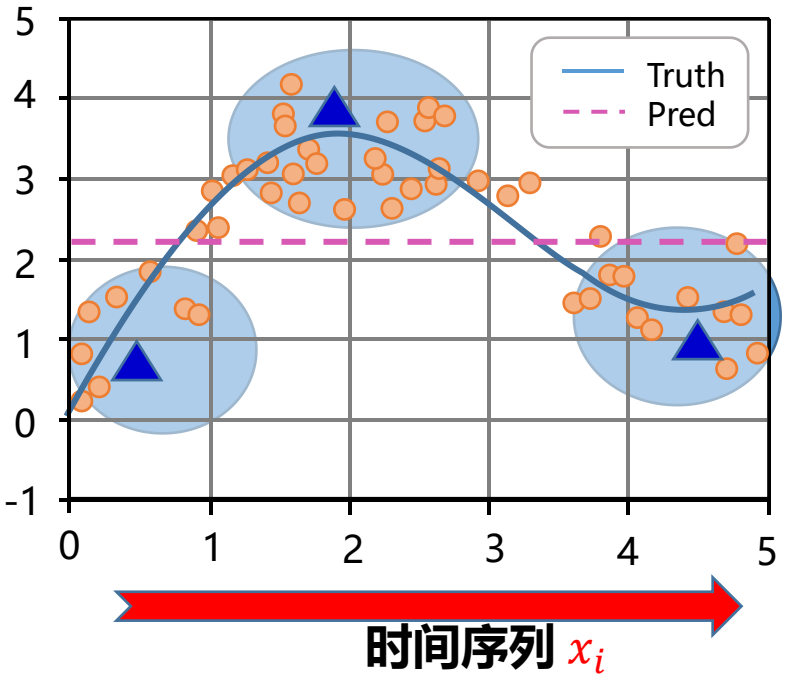
A solution is to get information from **the position of** samples, which can determine where is important:

$$f(x) = \sum_{i=1}^{n} \frac{K(x - x_i)}{\sum_{j=1}^{n} K(x - x_j)} y_i$$

where K is a kernel function. Then, we have

$$f(x) = \sum_{i=1}^{n} \alpha(x, x_i) \, y_i$$

where $\alpha(x, x_i)$ denotes the weights of y_i.



Relation with Kernel SVM:

$$f(\mathbf{x}) = \sum_{i}^{N} \alpha_i y_i \, k(\mathbf{x}_i, \mathbf{x}) + b$$

https://www.robots.ox.ac.uk/~az/lectures/ml/lect3.pdf

Motivation of Attention

- Nonparametric attention pooling

$$f(x) = \sum_{i=1}^{n} \alpha(x, x_i)\, y_i$$

where $\alpha(x, x_i)$ denotes the weights of y_i.

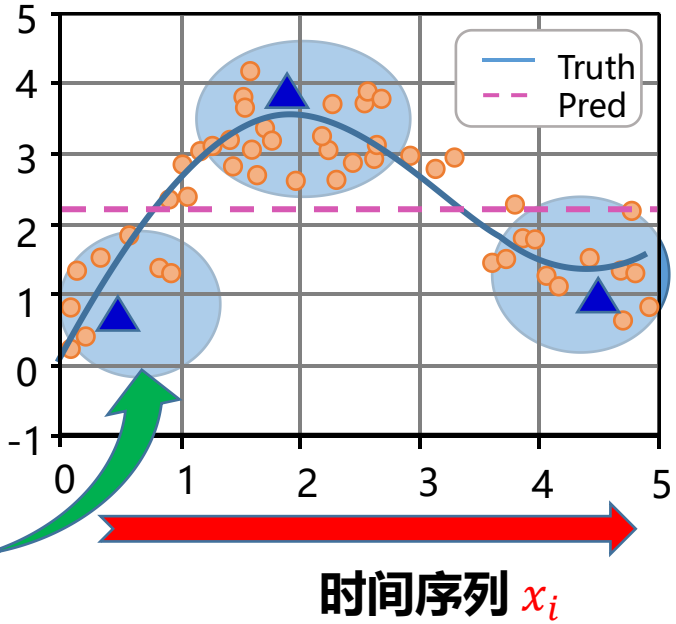
$$\alpha_i = \frac{K(x - x_i)}{\sum_{j=1}^{n} K(x - x_j)}$$

Normalization term

$$f(x) = \sum_{i=1}^{n} \frac{K(x - x_i)}{\sum_{j=1}^{n} K(x - x_j)} y_i$$

We can use Gaussian kernel to compute the distance of x to any point x_i:

$$K(x - x_i) = \exp\left(-\frac{1}{2}(x - x_i)^2\right)$$

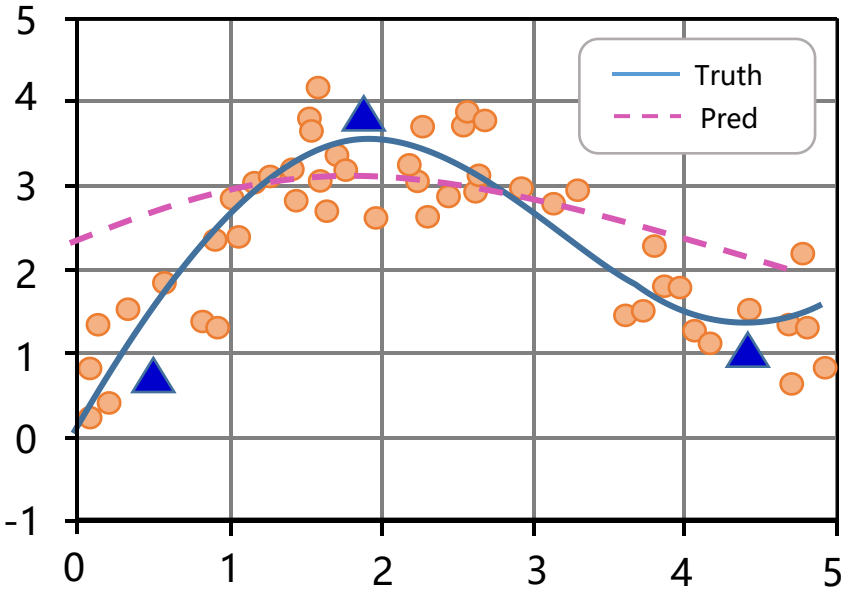


Motivation of Attention

- Nonparametric attention pooling

Then, the prediction becomes

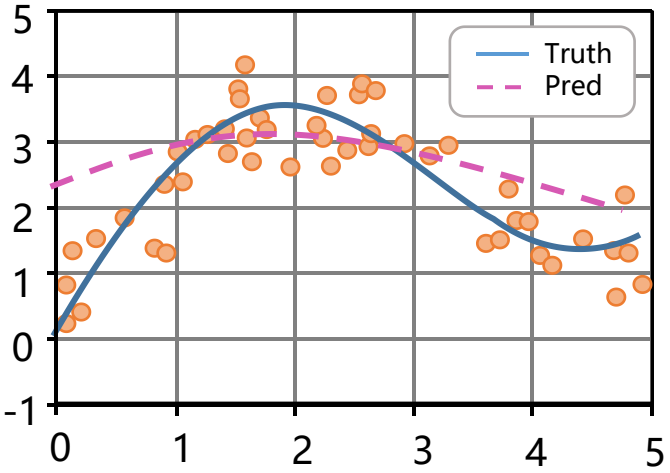
$$f(x) = \sum_{i=1}^{n} \alpha(x, x_i) y_i$$
$$= \sum_{i=1}^{n} \frac{\exp\left(-\frac{1}{2}(x - x_i)^2\right)}{\sum_{j=1}^{n} \exp\left(-\frac{1}{2}(x - x_j)^2\right)} y_i$$
$$= \sum_{i=1}^{n} \mathrm{softmax}\left(-\frac{1}{2}(x - x_i)^2\right) y_i.$$

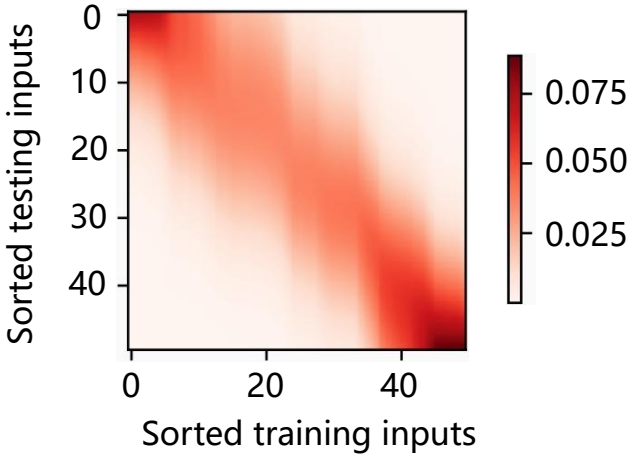


- The performance is improved with nonparametric attention pooling.

Core Idea of Attention

- Nonparametric attention pooling





$$f(x) = \sum_{i=1}^{n} \mathrm{softmax}\left(-\frac{1}{2}(x - x_i)^2\right) y_i .$$

- When x_i is more closer to x, the weight of corresponding y_i is larger, also means more important.
- x, x_i, y_i correspond to Query, Key, Value in Attention Mechanism, respectively

Core Idea of Attention

- Parametric attention pooling

A variant of attention pooling: Project $(x - x_i)$ by trainable matrix w.

Nonparametric attention pooling

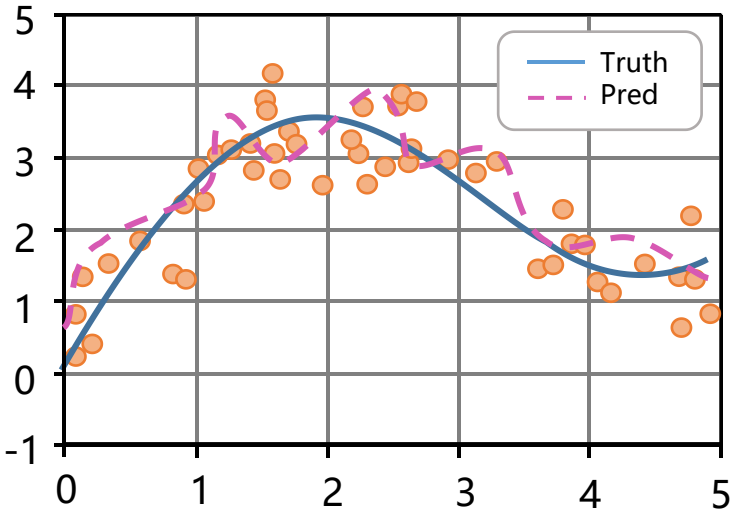
$$f(x) = \sum_{i=1}^{n} \text{softmax}\left(-\frac{1}{2}(x - x_i)^2\right) y_i.$$

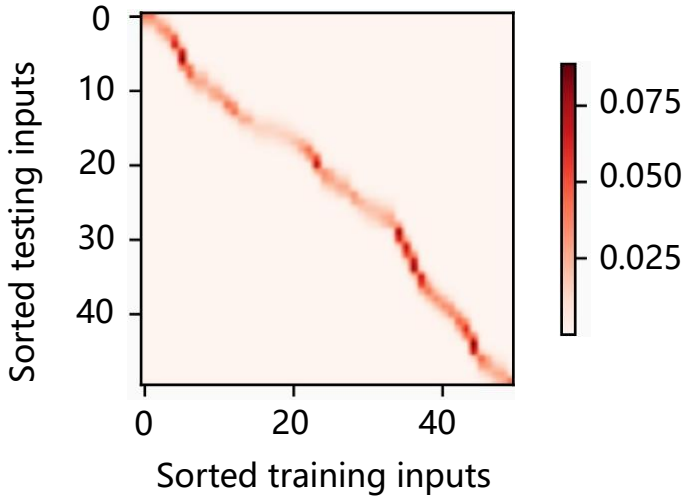
Parametric attention pooling

$$\begin{aligned} f(x) &= \sum_{i=1}^{n} \alpha(x, x_i)\, y_i \\ &= \sum_{i=1}^{n} \frac{\exp\left(-\frac{1}{2}((x - x_i)w)^2\right)}{\sum_{j=1}^{n} \exp\left(-\frac{1}{2}((x - x_j)w)^2\right)} y_i \\ &= \sum_{i=1}^{n} \text{softmax}\left(-\frac{1}{2}((x - x_i)w)^2\right) y_i. \end{aligned}$$

Core Idea of Attention

- Parametric attention pooling





$$f(x) = \sum_{i=1}^{n} \mathrm{softmax}\left(-\frac{1}{2}((x - x_i)\, w)^2\right) y_i.$$

- **Parametric** attention pooling can make attention more concentration, but probability over-fitting.

Core Idea of Attention

Summary

- Attention pooling can be parametric or nonparametric.
- In attention pooling, each $Value_i$ corresponds to a weight $\alpha(Query, Key_i)$.

- Formulation of Attention Function

$$f\left(\boldsymbol{q}, (\boldsymbol{k_1}, \boldsymbol{v_1}), \ldots, (\boldsymbol{k_n}, \boldsymbol{v_n})\right) = \sum_{i=1}^{n} \alpha(\boldsymbol{q}, \boldsymbol{k_i})\boldsymbol{v}_i$$

where $\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v}$ denote Query, Key and Value, respectively.

$\alpha(\boldsymbol{q}, \boldsymbol{k_i})$ is scoring function.

Core Idea of Attention

- An example of scoring function

$$f\big(\boldsymbol{q}, (\boldsymbol{k_1}, \boldsymbol{v_1}), \ldots, (\boldsymbol{k_n}, \boldsymbol{v_n})\big) = \sum_{i=1}^{n} \alpha(\boldsymbol{q}, \boldsymbol{k_i})\boldsymbol{v}_i$$

$\alpha(\boldsymbol{q}, \boldsymbol{k}_i)$ is usually $softmax$ function:

$$\alpha(\mathbf{q}, \mathbf{k}_i) = softmax\big(\alpha(\mathbf{q}, \mathbf{k}_i)\big) = \frac{\exp(\alpha(\mathbf{q}, \mathbf{k}_i))}{\sum_{j=1}^{m} \exp(\alpha(\mathbf{q}, \mathbf{k}_j))} \in \mathbb{R}$$

- More forms of scoring function $\alpha(\boldsymbol{q}, \boldsymbol{k}_i)$
  - Additive Attention:

$$\alpha(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{w}_v^{\mathsf{T}} \tanh\left(\boldsymbol{W}_q \boldsymbol{q} + \boldsymbol{W}_k \boldsymbol{k}\right) \in \mathbb{R}$$

  - Scaled Dot-Product Attention:

$$\alpha(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{q}^T \boldsymbol{k}/d$$

Implementation of Attention Layer

- Additive Attention:

$$\alpha(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{w}_v^\top \tanh\left(\boldsymbol{W}_q \boldsymbol{q} + \boldsymbol{W}_k \boldsymbol{k}\right) \in \mathbb{R}$$

```python
class AdditiveAttention(nn.Module):
    def __init__(self, key_size, query_size, num_hiddens, dropout, **kwargs):
        super(AdditiveAttention, self).__init__(**kwargs)
        self.W_k = nn.Linear(key_size, num_hiddens, bias=False)
        self.W_q = nn.Linear(query_size, num_hiddens, bias=False)
        self.w_v = nn.Linear(num_hiddens, 1, bias=False)
        self.dropout = nn.Dropout(dropout)

    def forward(self, queries, keys, values, valid_lens):
        # 下面一步将Q和K分别对独自的W进行点乘
        queries, keys = self.W_q(queries), self.W_k(keys)
        # 输出的：queries[batch_size,num_qeury,num_hidden]
        # 输出的：keys[batch_size,num_keys,num_hidden]
        features = queries.unsqueeze(2) + keys.unsqueeze(1)
        # broadcasting 相加，features[batch_size,num_qeury,num_keys,num_hidden]
        features = torch.tanh(features)
        scores = self.w_v(features).squeeze(-1)
        # scores[batch_size,num_qeury,num_keys]
        self.attention_weights = masked_softmax(scores, valid_lens)
        return torch.bmm(self.dropout(self.attention_weights), values)
        # values的最后两维和keys的长度是一样的。
```

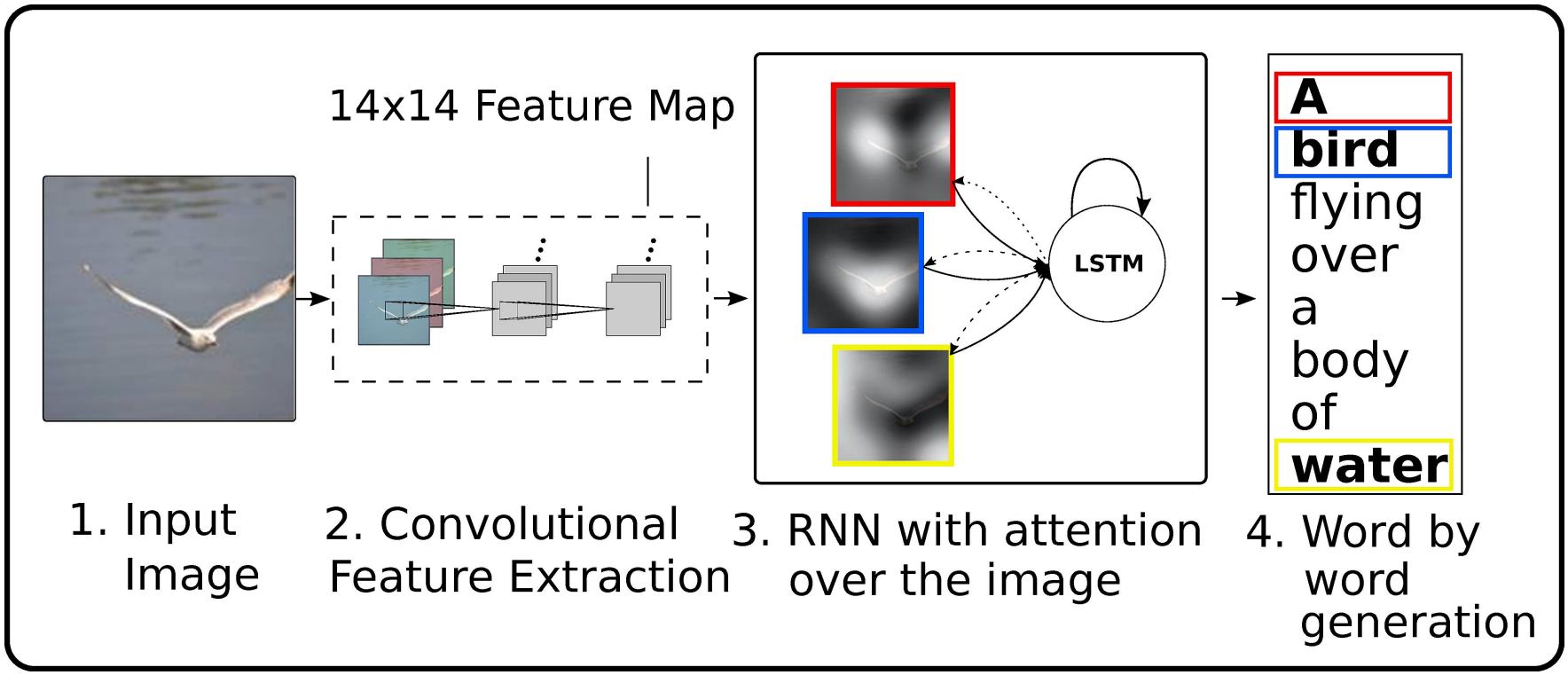
Implementation of Attention Layer

- Scaled Dot-Product Attention:

$$\alpha(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{q}^T \boldsymbol{k}/d$$

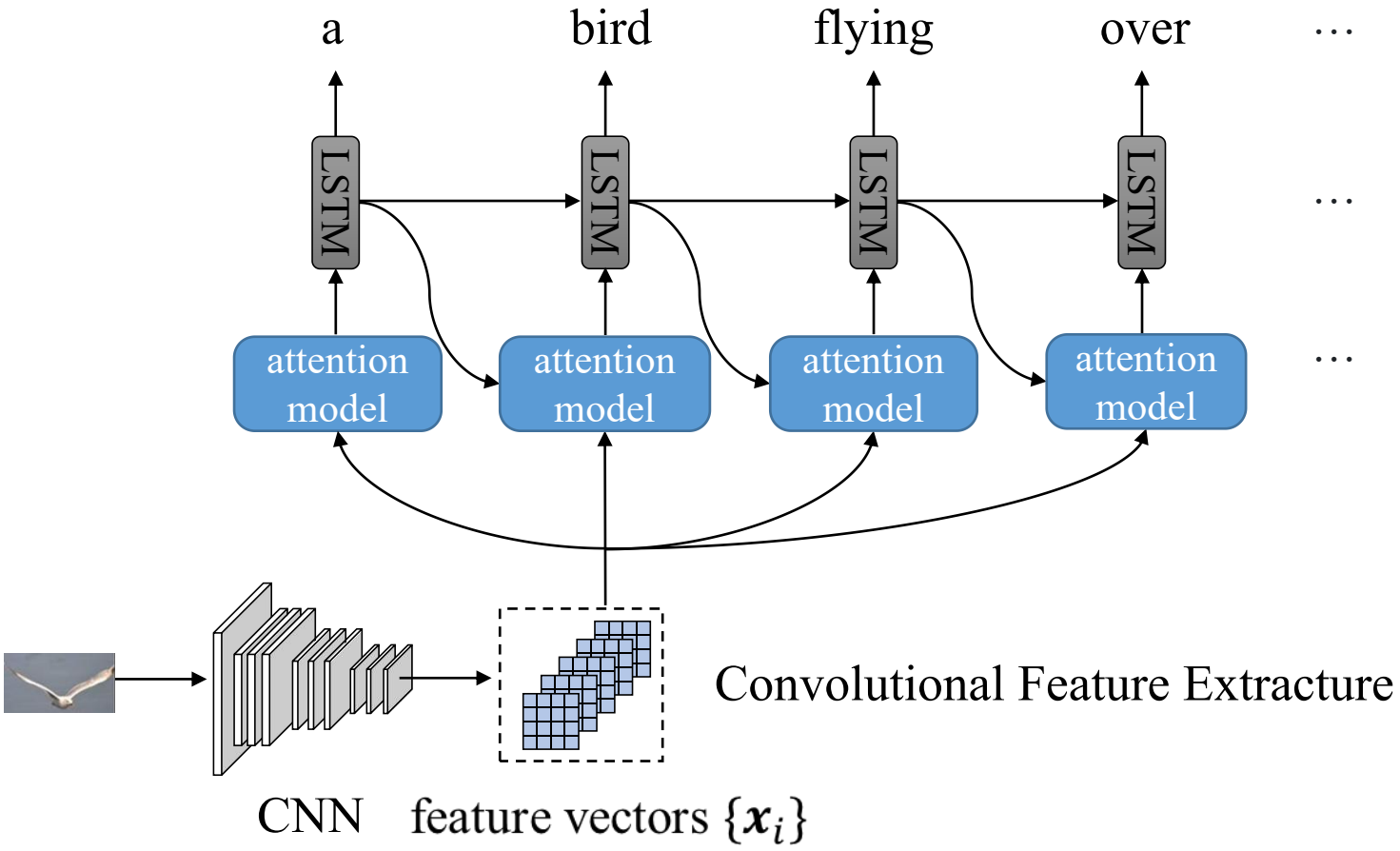
```
class DotProductAttention(nn.Module):
    def __init__(self, dropout, **kwargs):
        super(DotProductAttention, self).__init__(**kwargs)
        self.dropout = nn.Dropout(dropout)
    def forward(self, queries, keys, values, valid_lens=None):
        d = queries.shape[-1]
        scores = torch.bmm(queries, keys.transpose(1, 2)) / math.sqrt(d)
        self.attention_weights = masked_softmax(scores, valid_lens)
        return torch.bmm(self.dropout(self.attention_weights), values)
```

Attention in Image Captioning



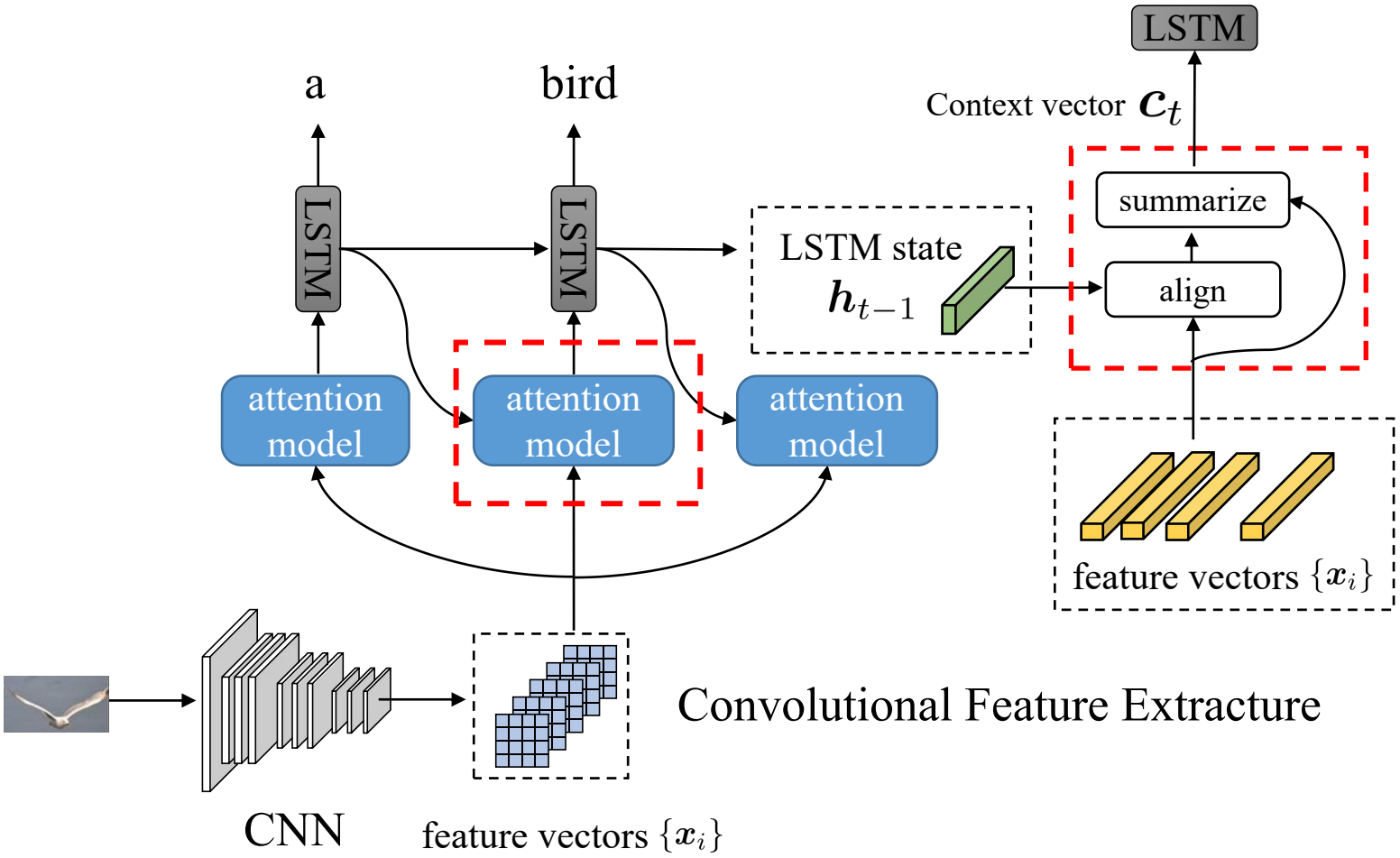
Attention in Image Caption

LSTM with attention for image captioning.



Attention in Image Caption

The attention module



Attention in Image Caption

align decodes which encoder state $\boldsymbol{x}_i$ should pay attention to according the decoder state $\boldsymbol{h}_{t-1}$.

- Compute the alignment score between $\boldsymbol{x}_i$ and $\boldsymbol{h}_{t-1}$:

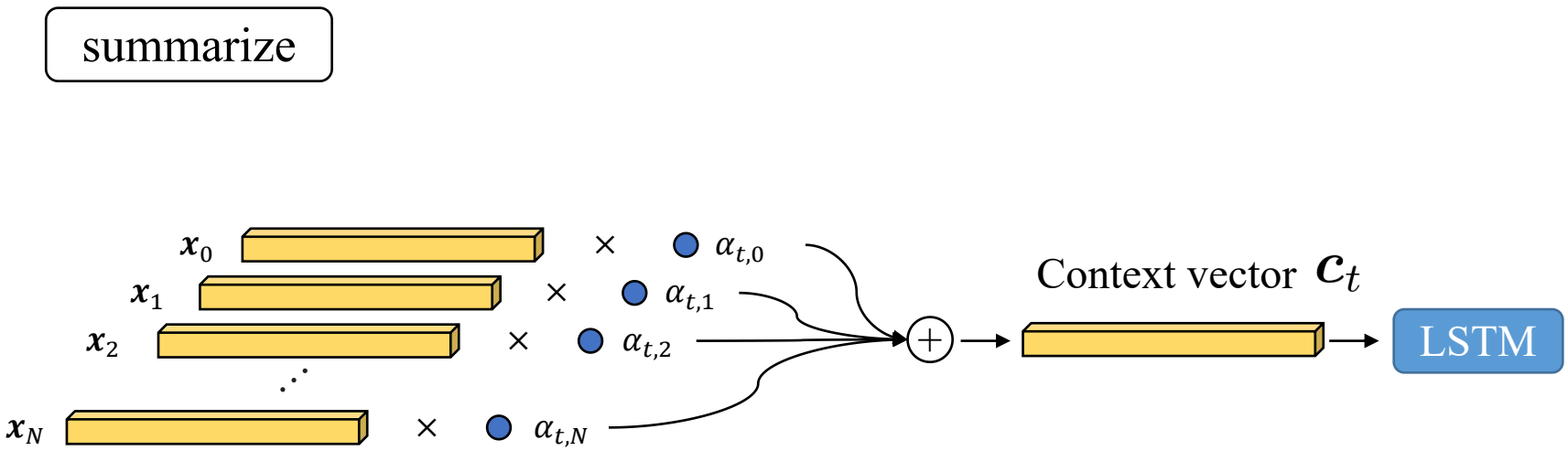
$$e_{t,i} = \boldsymbol{W}^T \tanh(\boldsymbol{W}_x^T \boldsymbol{x}_i + \boldsymbol{W}_h^T \boldsymbol{h}_{t-1}).$$

- Compute the attention weight for $\boldsymbol{x}_i$:

$$\alpha_{t,i} = \frac{\exp(e_{t,i})}{\sum_j \exp(e_{t,j})}$$

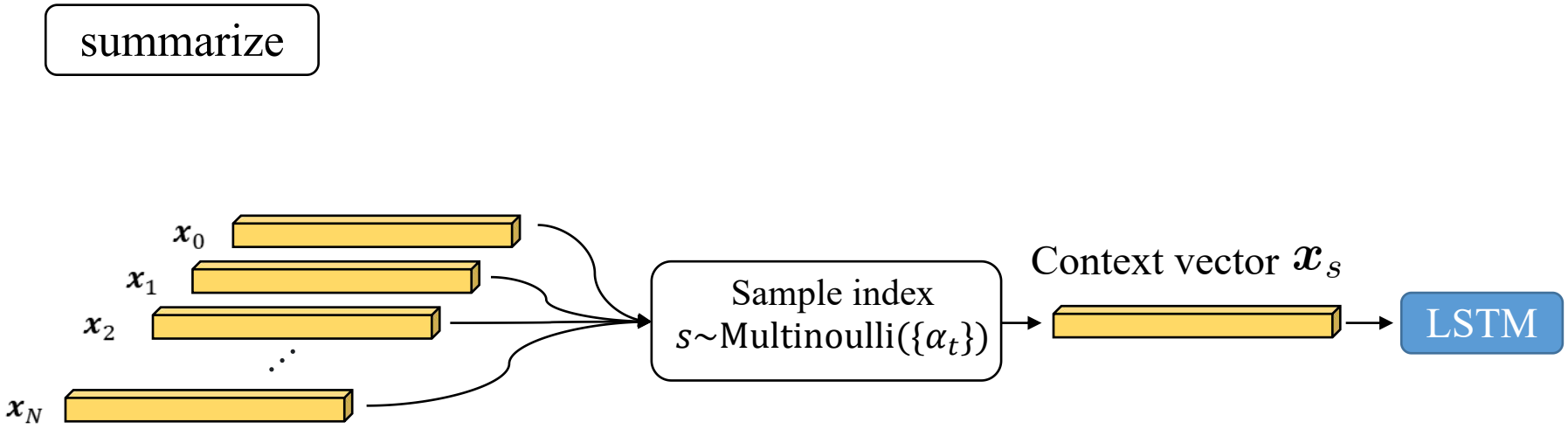
$\boldsymbol{W}, \boldsymbol{W}_h, \boldsymbol{W}_s$ are jointly trained within the whole architecture.

Attention in Image Caption



- Soft attention: summarize all encoder states into a context vector with the attention weights, and feed it into the decoder LSTM.

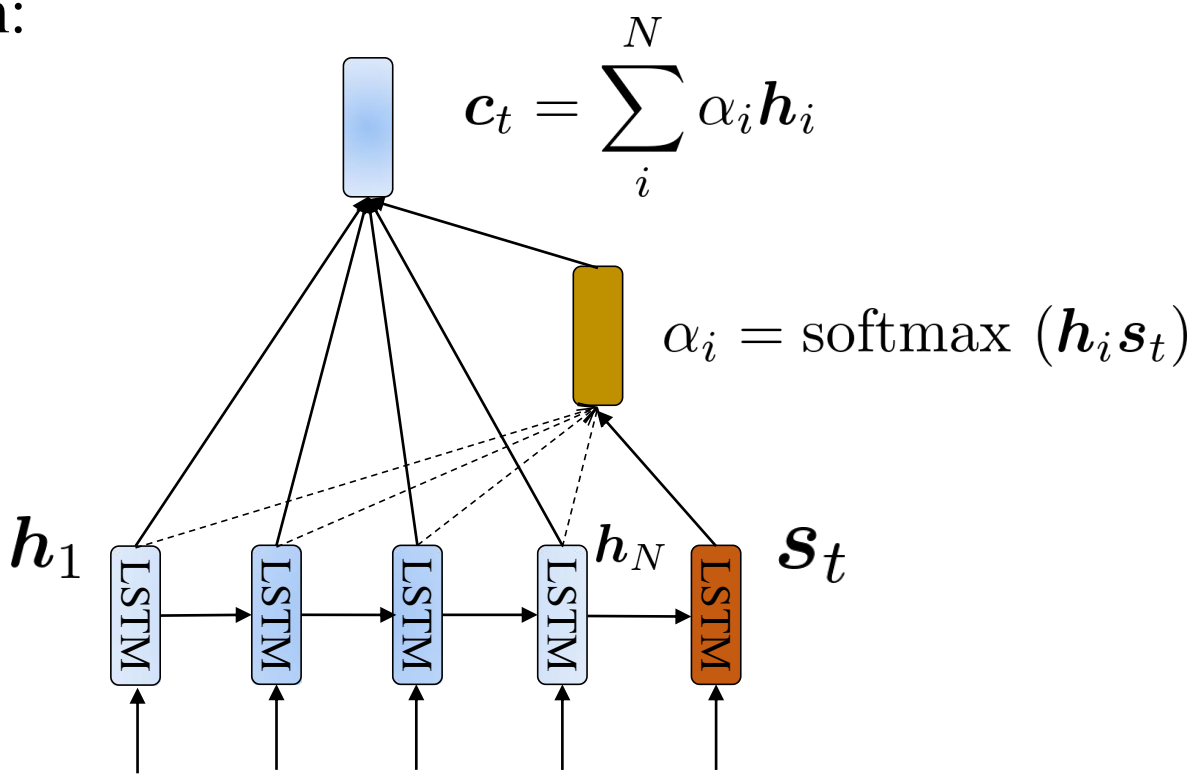
Attention in Image Caption



- Hard attention: parameterize a multinoulli distribution wth the attention weights, and sample an feature vector x_s from $\{x_i\}$according to it.

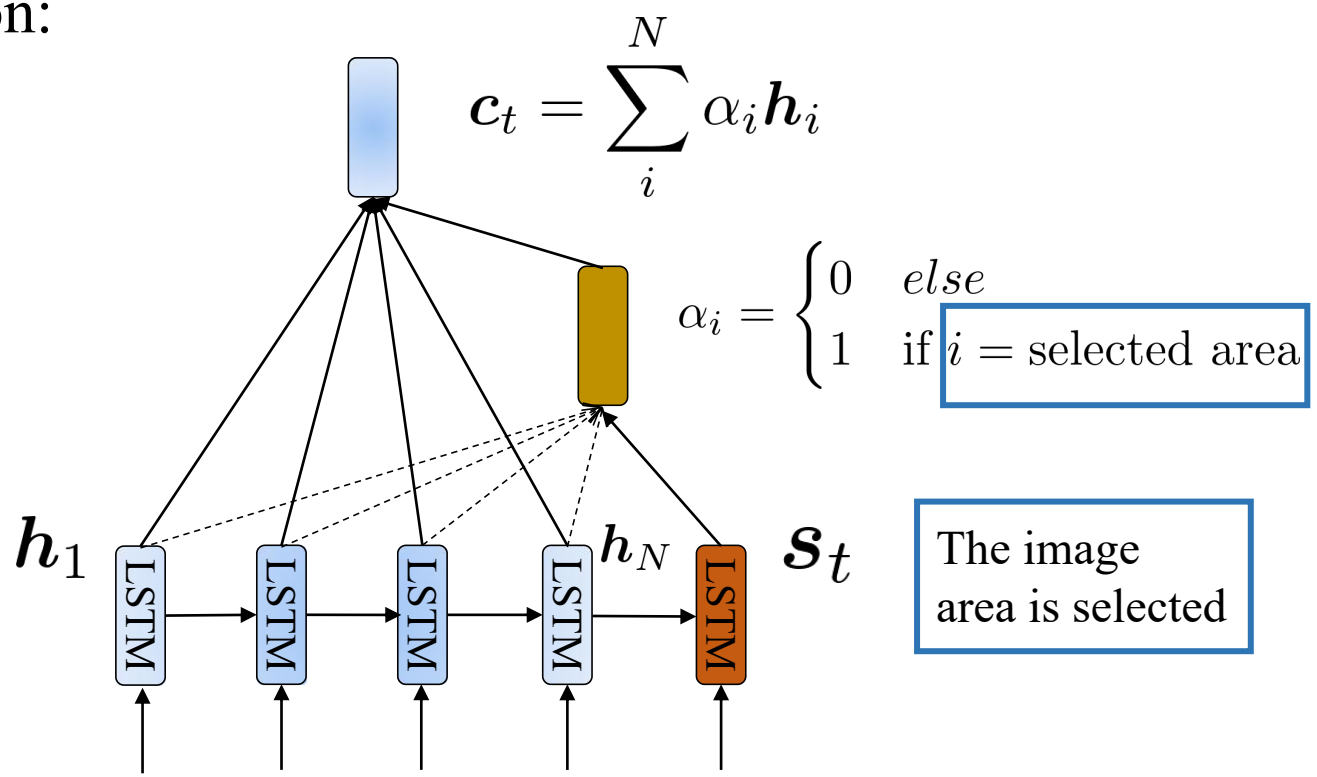
Variants of Attention

Soft attention:



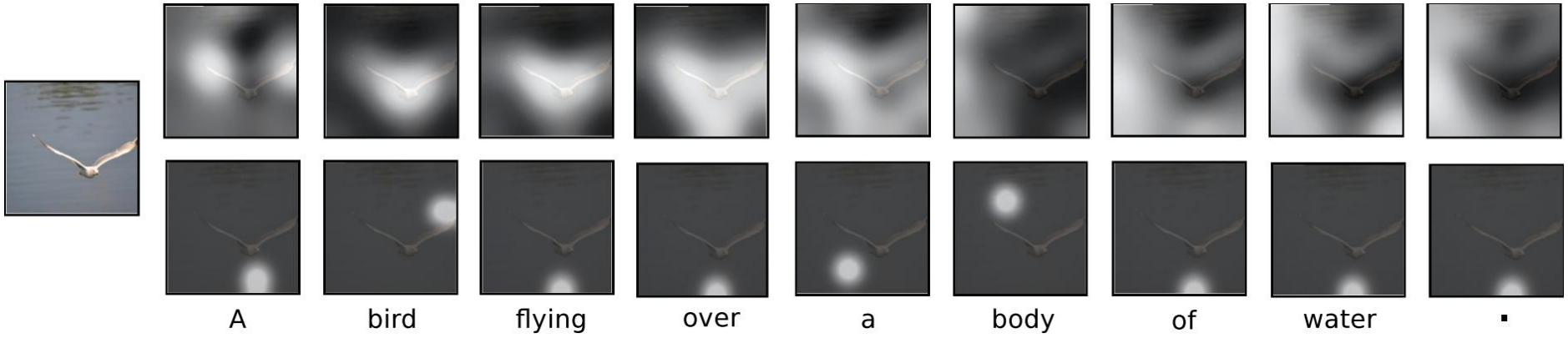
Variants of Attention

Hard attention:

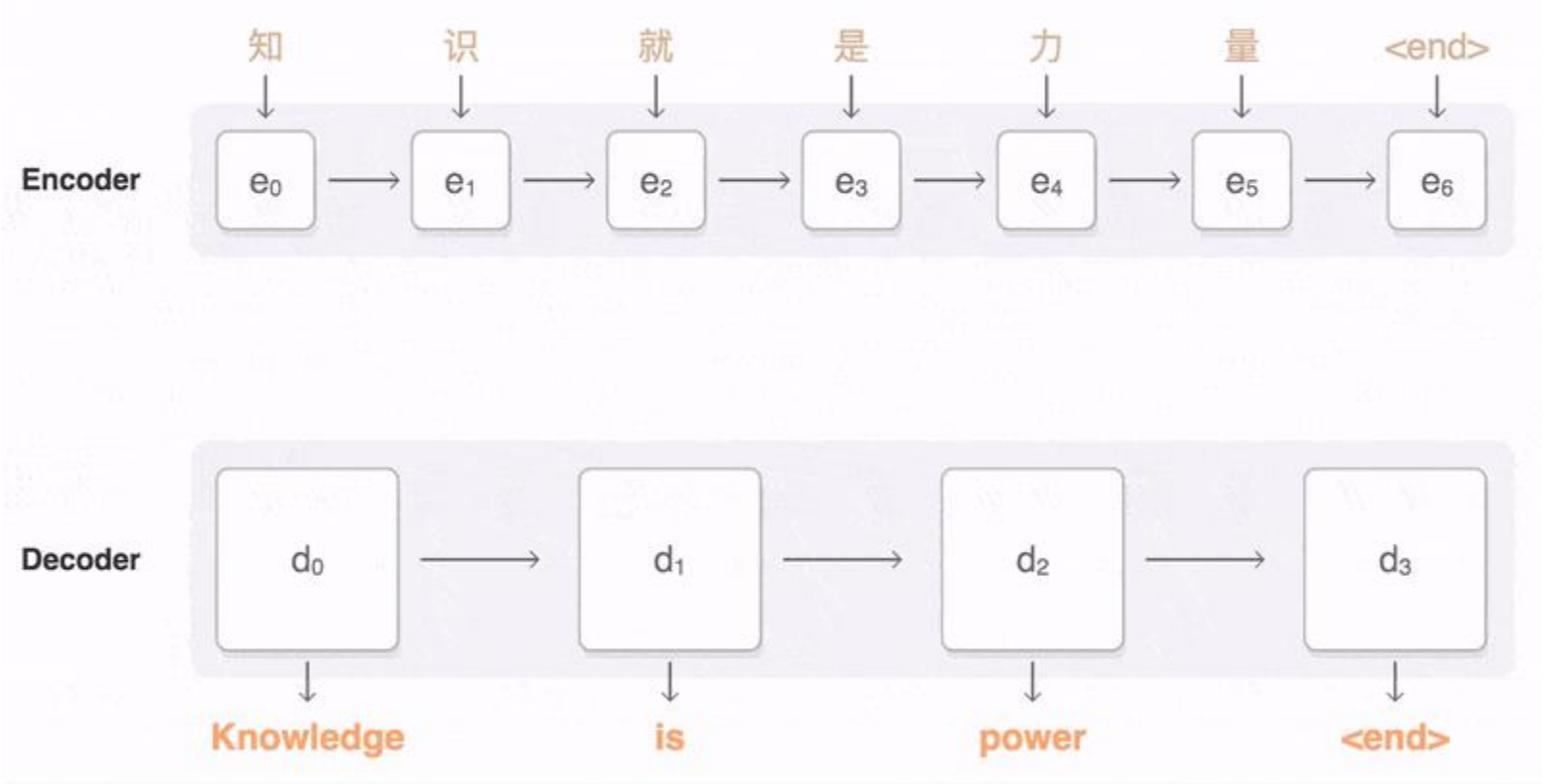


Attention in Image Caption

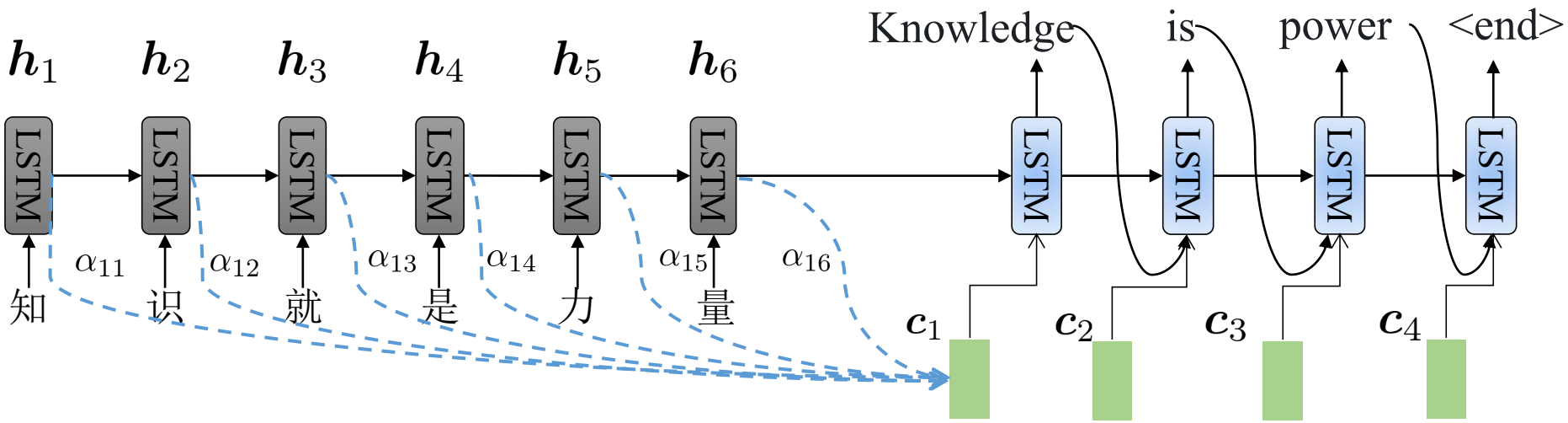
Soft attention vs. Hard attention



Attention in Machine Translation



Attention in Machine Translation



知 识 就 是 力 量 Context vector

$h_1 \times \alpha_{11} + h_2 \times \alpha_{12} + h_3 \times \alpha_{13} + h_4 \times \alpha_{14} + h_5 \times \alpha_{15} + h_6 \times \alpha_{16} = c_1 \rightarrow$ Knowledge

$h_1 \times \alpha_{41} + h_2 \times \alpha_{42} + h_3 \times \alpha_{43} + h_4 \times \alpha_{44} + h_5 \times \alpha_{45} + h_6 \times \alpha_{46} = c_4 \rightarrow$ is

Attention in Machine Translation

How to compute the attention weights α?

- Compute the alignment score between $\boldsymbol{h}_i$ and $\boldsymbol{s}_{t-1}$:

$$e_{t,i} = \boldsymbol{W}^{\mathbf{T}} \tanh(\boldsymbol{W}_h^{\mathbf{T}} \boldsymbol{h}_i + \boldsymbol{W}_s^{\mathbf{T}} \boldsymbol{s}_{t-1})$$

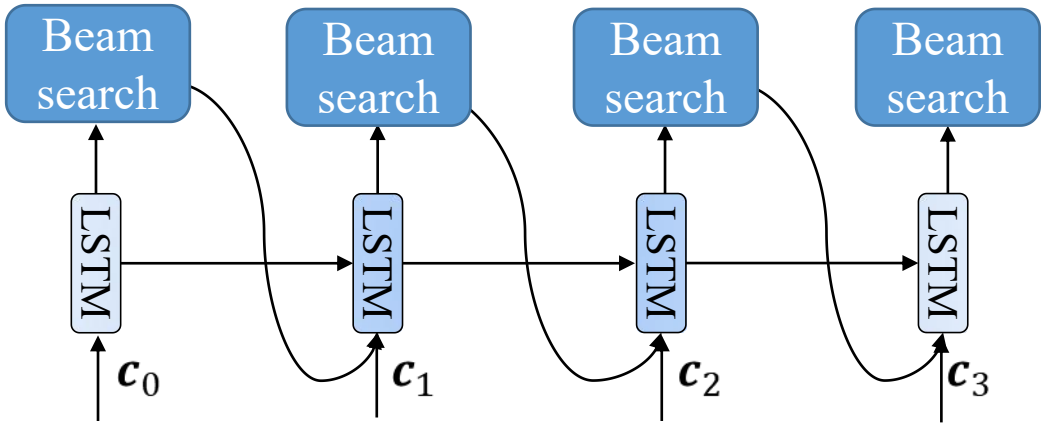
- Compute the attention weight for $\boldsymbol{h}_i$:

$$\alpha_{t,i} = \frac{\exp(e_{t,i})}{\sum_j \exp(e_{t,j})}$$

$\boldsymbol{W}, \boldsymbol{W}_h, \boldsymbol{W}_s$ are jointly trained within the whole architecture.

Attention in Machine Translation

Inference:

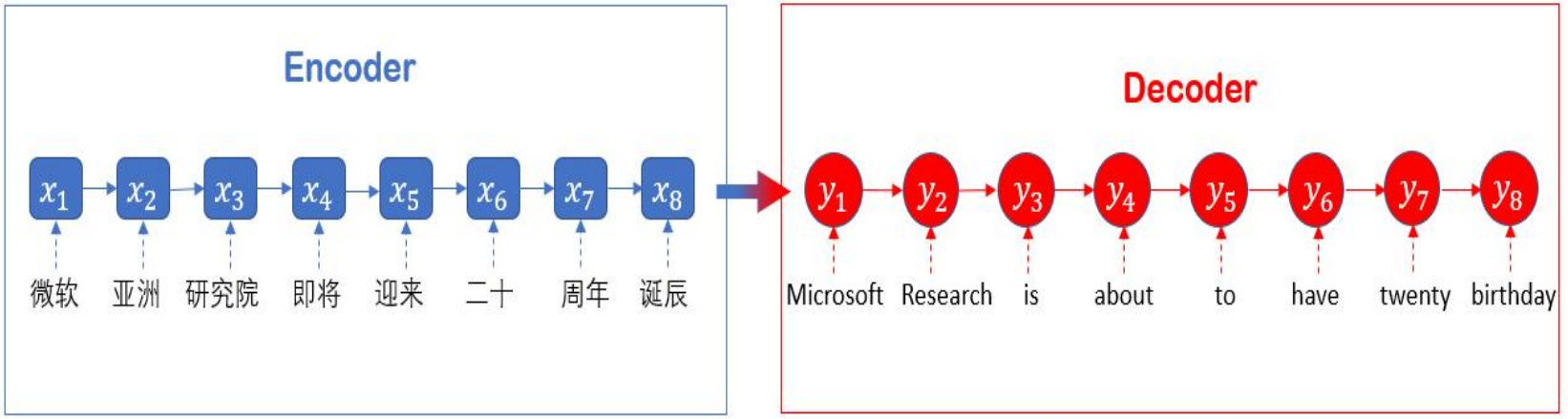


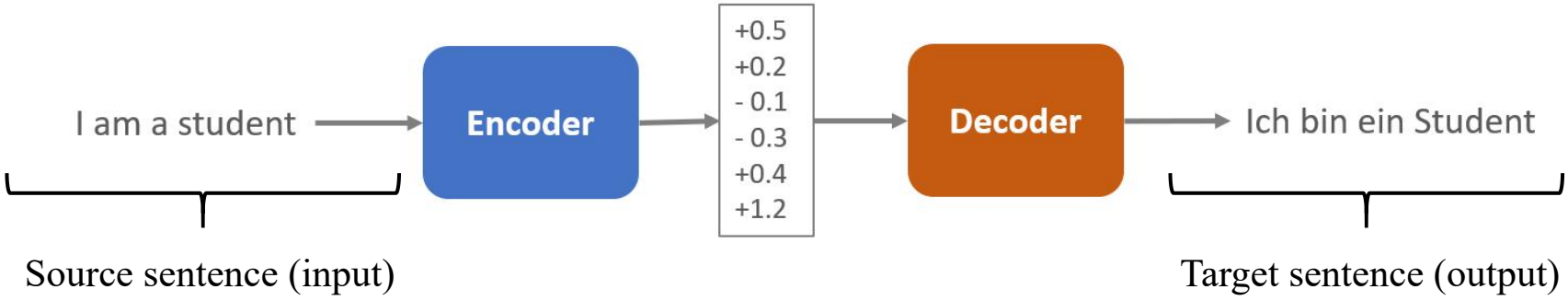
- Beam Search: at every step, select the sequences with top-k probabilities, and feed its last word into the next time step.

Outline

Motivation: Neural Machine Translation (NMT)

- Definition: an approach to machine translation that uses an artificial neural network to predict the likelihood of a sequence of words.





Neural Machine Translation (NMT)

How to predict the likelihood of a sequence of words?

- NMT directly calculates $P(\boldsymbol{y}|\boldsymbol{x})$:

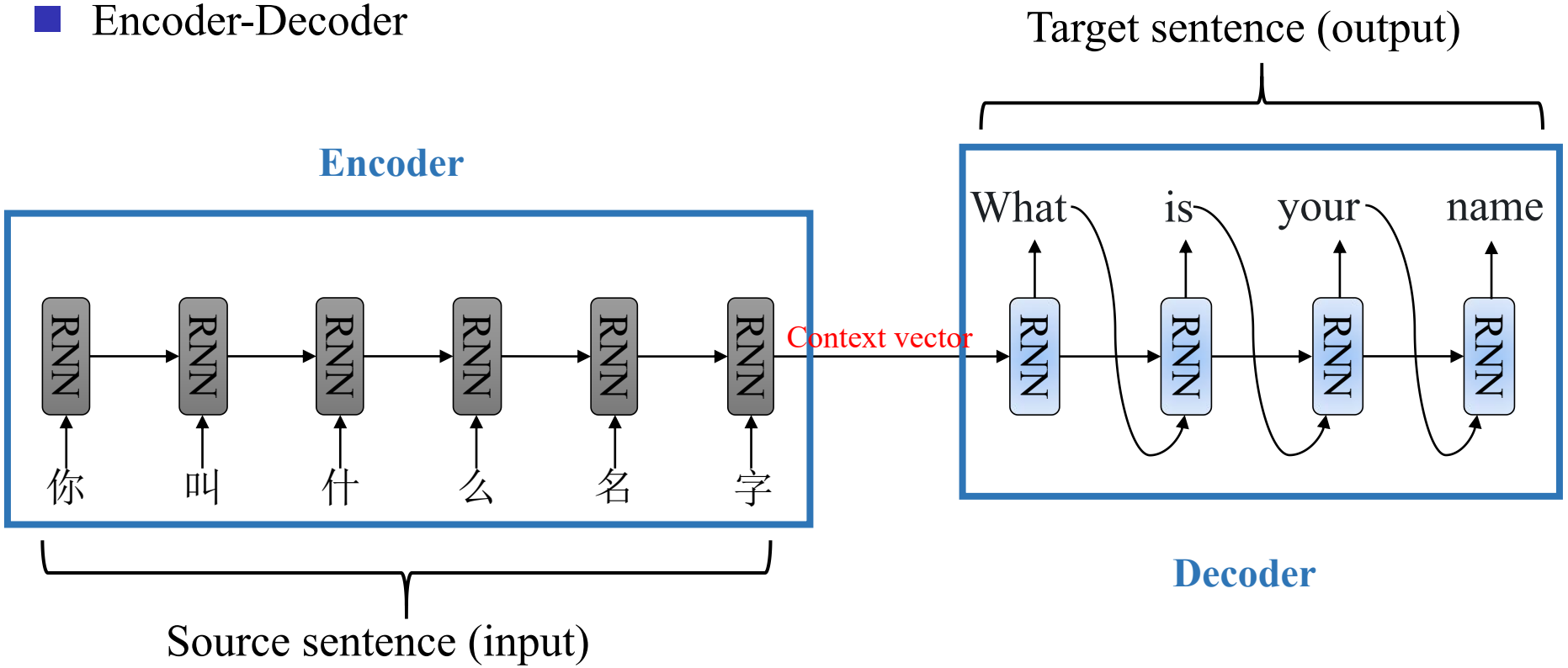
$$P(\boldsymbol{y}|\boldsymbol{x}) = P(\boldsymbol{y}_1|\boldsymbol{x})P(\boldsymbol{y}_2|\boldsymbol{y}_1,\boldsymbol{x})P(\boldsymbol{y}_3|\boldsymbol{y}_1,\boldsymbol{y}_2,\boldsymbol{x})\ldots P(\boldsymbol{y}_T|\boldsymbol{y}_1,\ldots,\boldsymbol{y}_{T-1},\boldsymbol{x})$$

Probability of next target word, given target words so far and souce sentence $\boldsymbol{x}$

- Input: encode of the source sentence $\boldsymbol{x}_1,\ldots,\boldsymbol{x}_T$.
- Output: generate target sentence $\boldsymbol{y}_1,\cdots,\boldsymbol{y}_T$.

Neural Machine Translation (NMT)

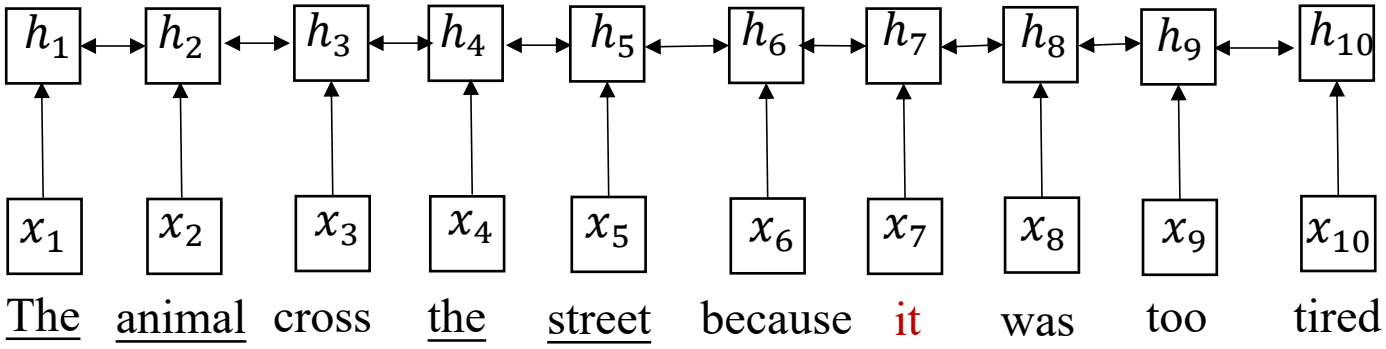
- Encoder-Decoder



- Encoder RNN produces an encoding of the source sentence into a context vector.
- Decoder RNN is a language Model that generates target sentence from the context vector iteratively.

Motivation

RNN: difficult to capture long-distance dependencies due to vanishing gradient

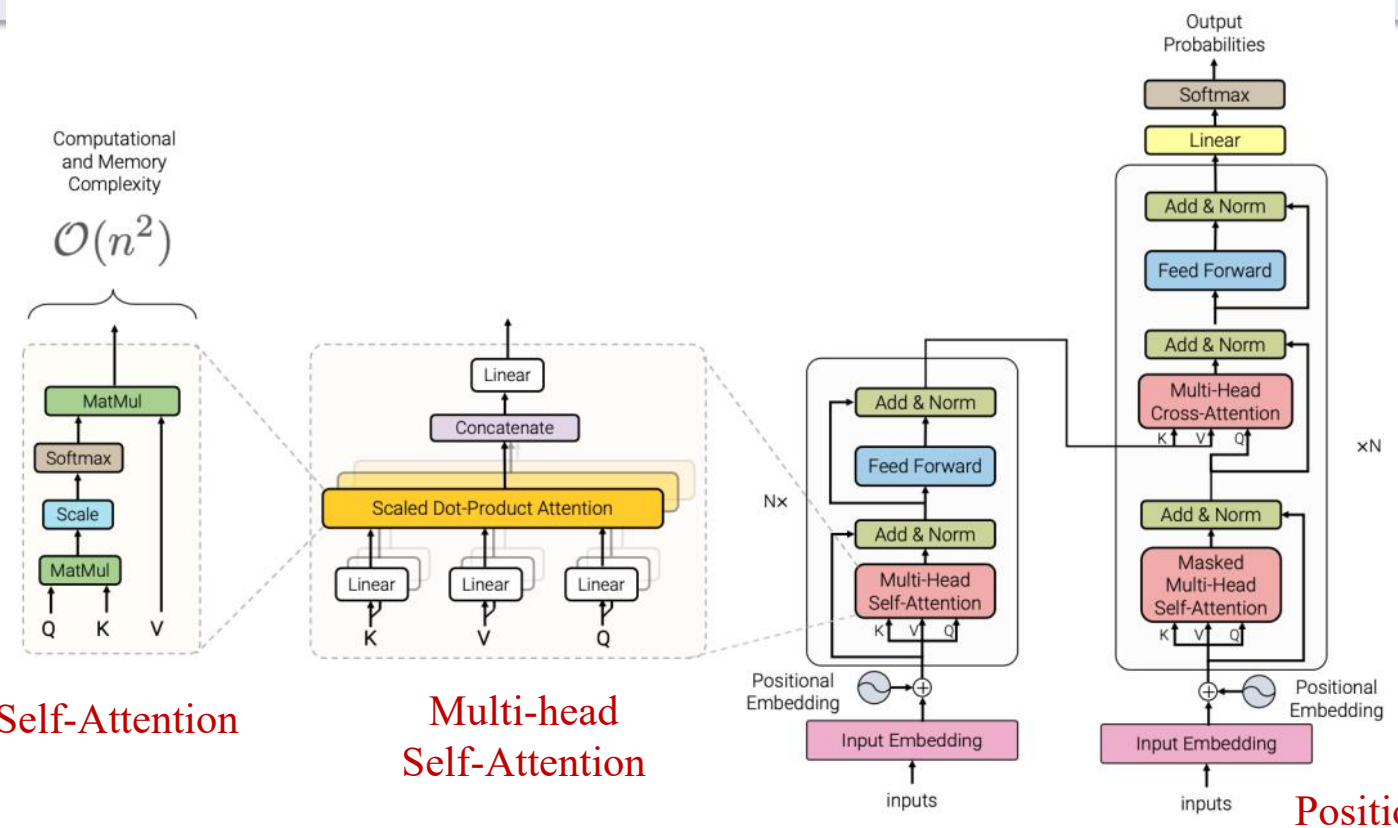


Bidirectional RNN modeling process

Self-Attention in Transformer can solve the long-distance dependence problem by calculating the direct dependence between words.

Transformer

- Transformer has a encoder-decoder architecture similar to the previous RNN models.
- Both encoder and decoder use N stacked self-attention layers.



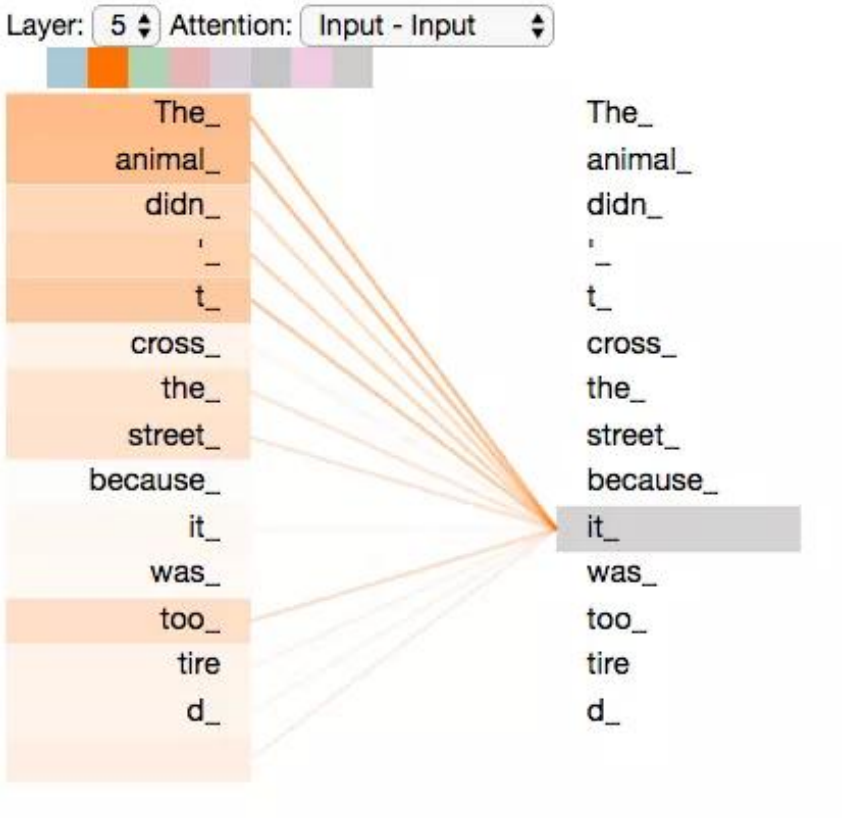
Self-Attention

Multi-head Self-Attention

Positional Encoding

Self-Attention in Transformer

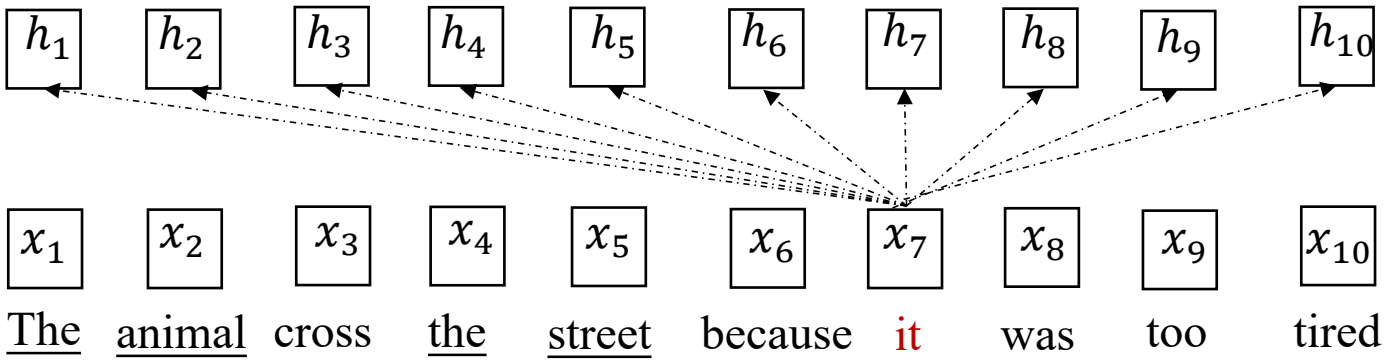
Self-Attention in Transformer can bake the "understanding" of other relevant words into the one we're currently processing.



As we are encoding the word "it", part of the attention mechanism was focusing on "The Animal", and baked a part of its representation into the encoding of "it".

Self-Attention in Transformer

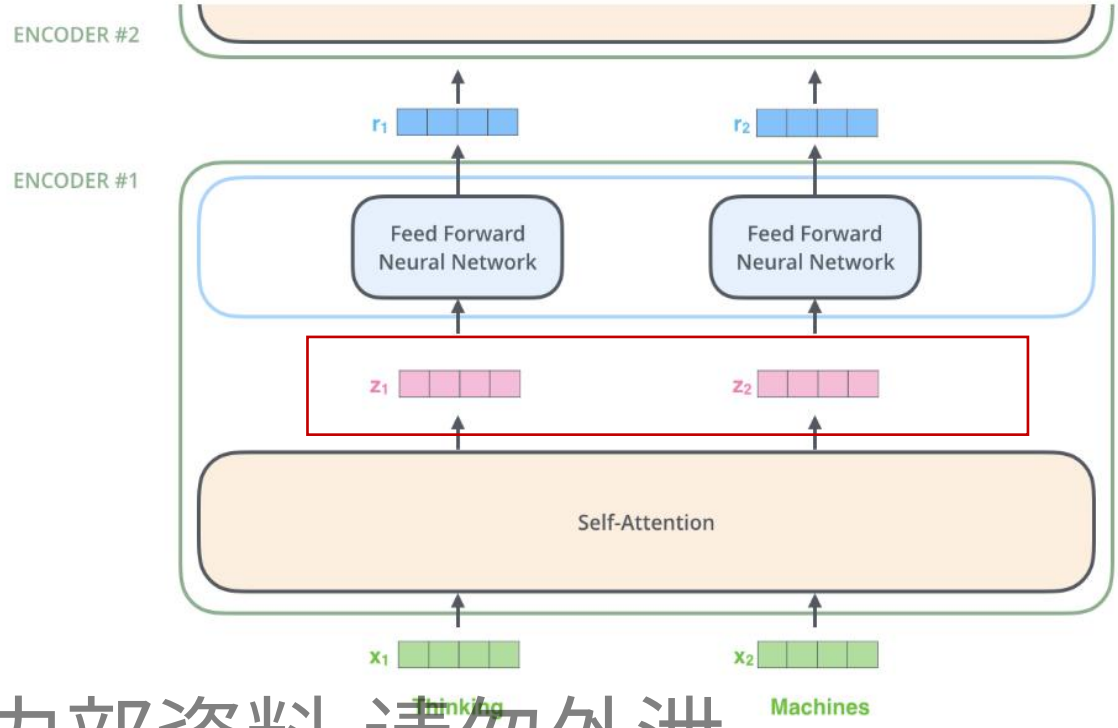
- Self-Attention calculates attention between each word and all words.
- The maximum path length between each word is only 1.
- Self-Attention can capture long-distance dependencies.



Self-Attention modeling process

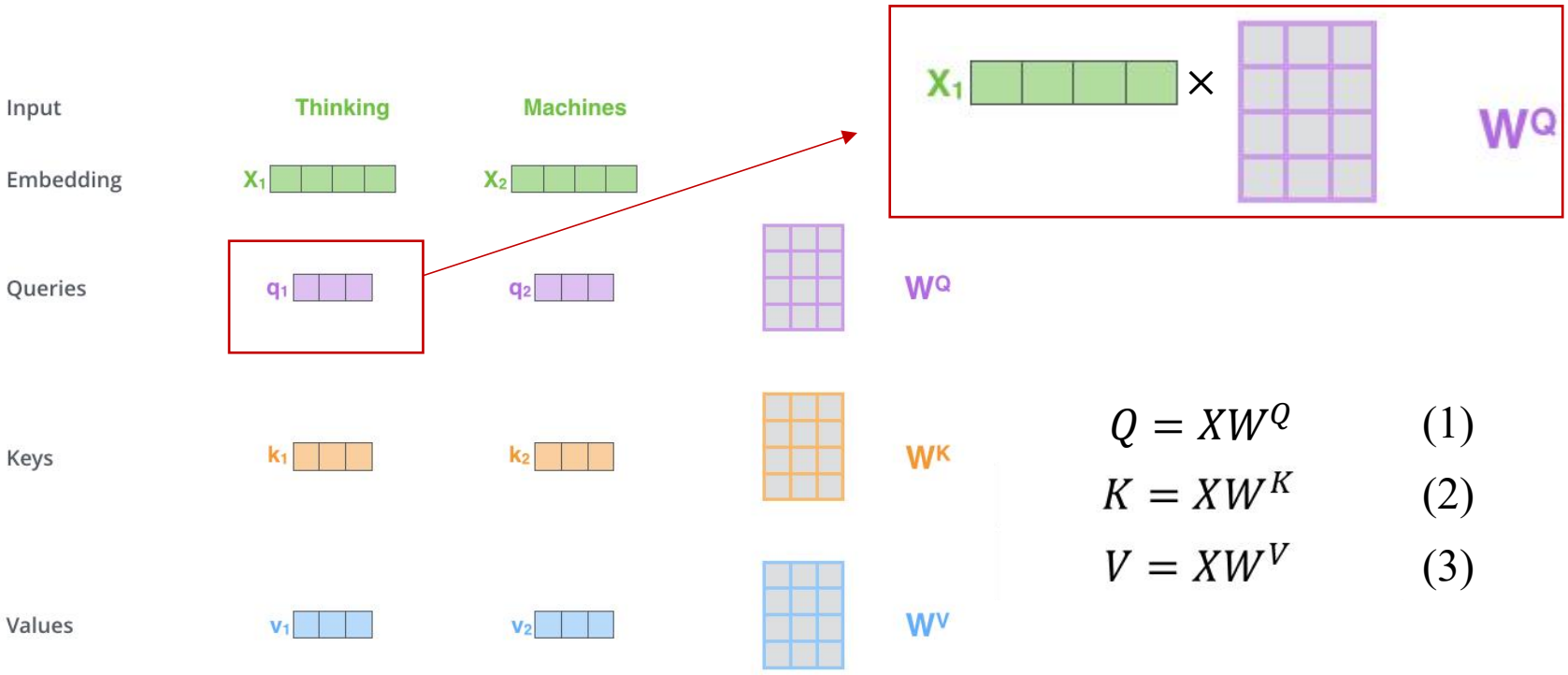
Self-Attention in Transformer

- The encoder's inputs first flow through a self-attention layer.
- It looks at other words in the sentence when it encodes a specific word.



How to get vector z ?

Self-Attention in Transformer



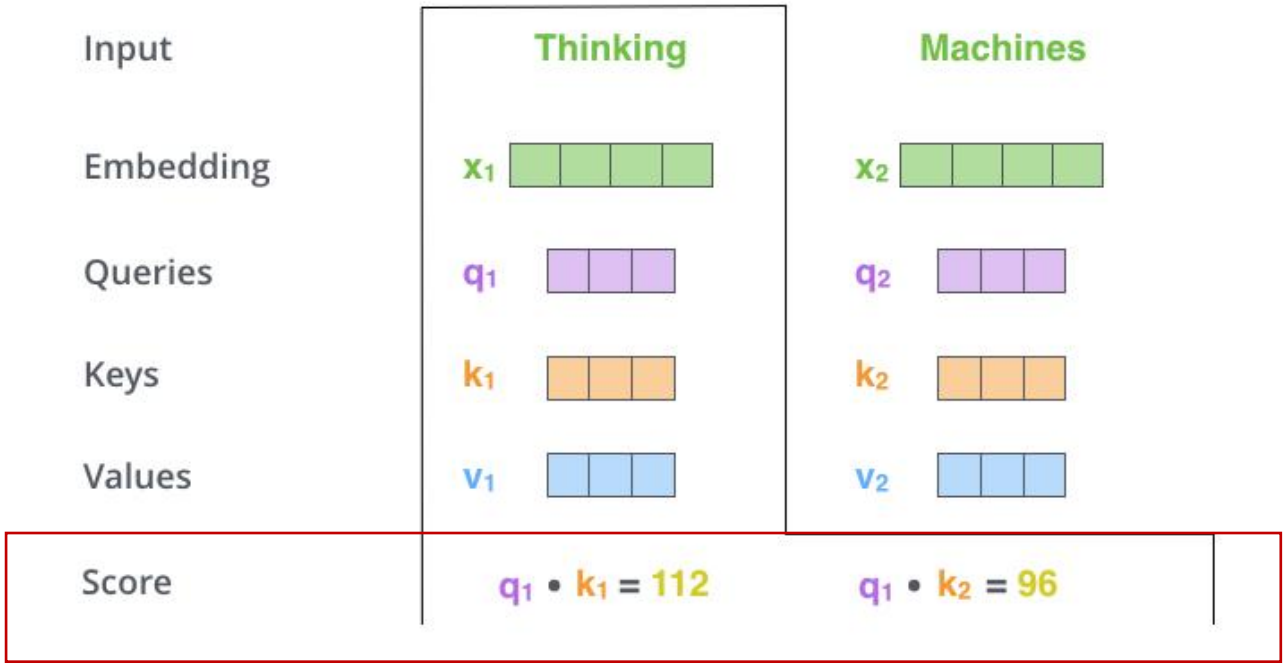
$$Q = XW^Q \quad (1)$$

$$K = XW^K \quad (2)$$

$$V = XW^V \quad (3)$$

Step 1. Create a query vector(q_i), a key vector(k_i), and a value vector(v_1) from each of the encoder's input vectors.

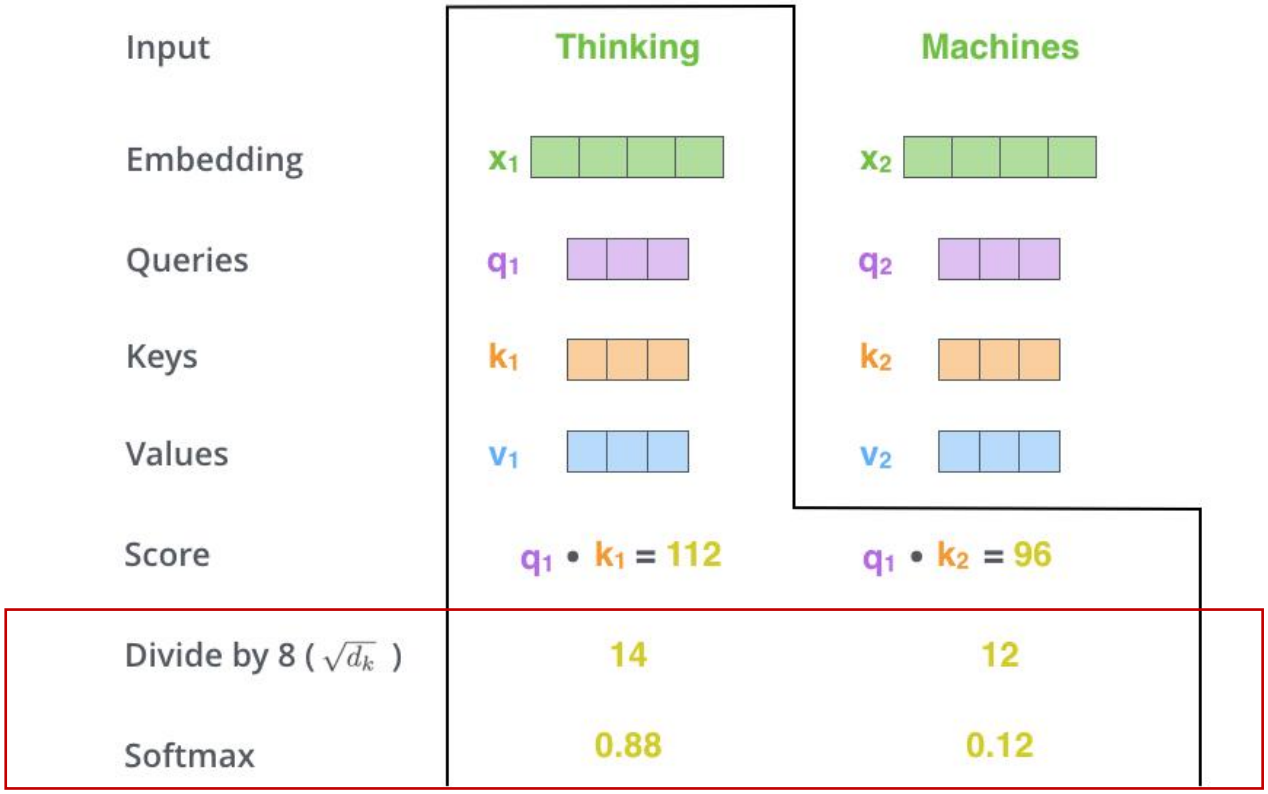
Self-Attention in Transformer



Step 2. Calculate a score between each word.

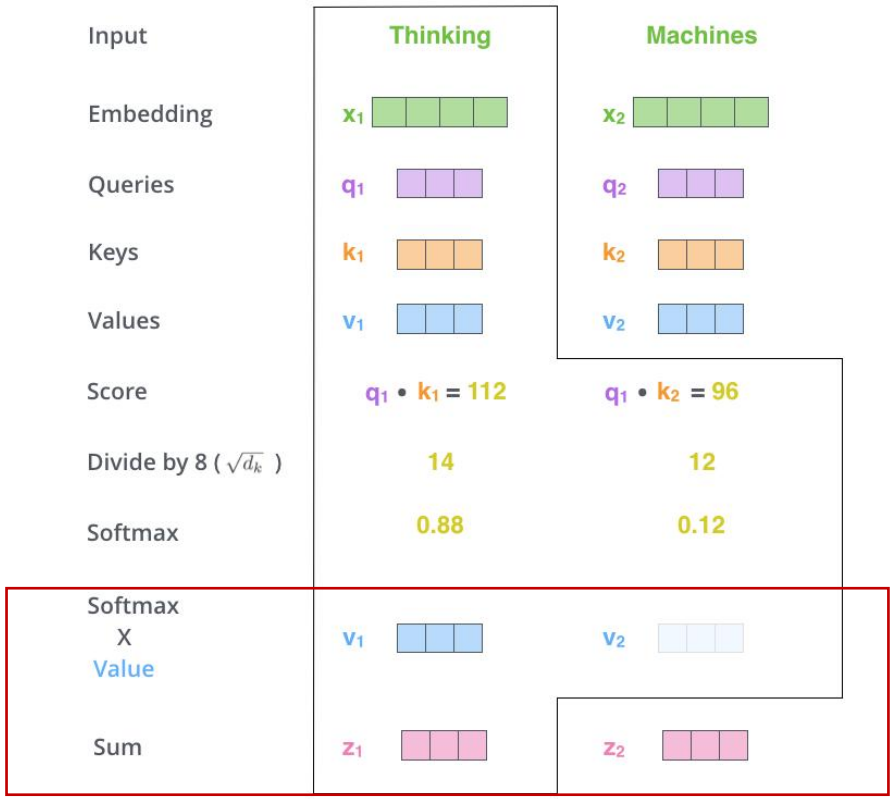
The score determines how much focus to place on other parts of the input sentence as we encode a word at a certain position.

Self-Attention in Transformer



Step 3. Dive the scores by the square root of the dimension of the key vectors.
Step 4. SoftMax normalizes the scores so they're all positive and add up to 1.

Self-Attention in Transformer



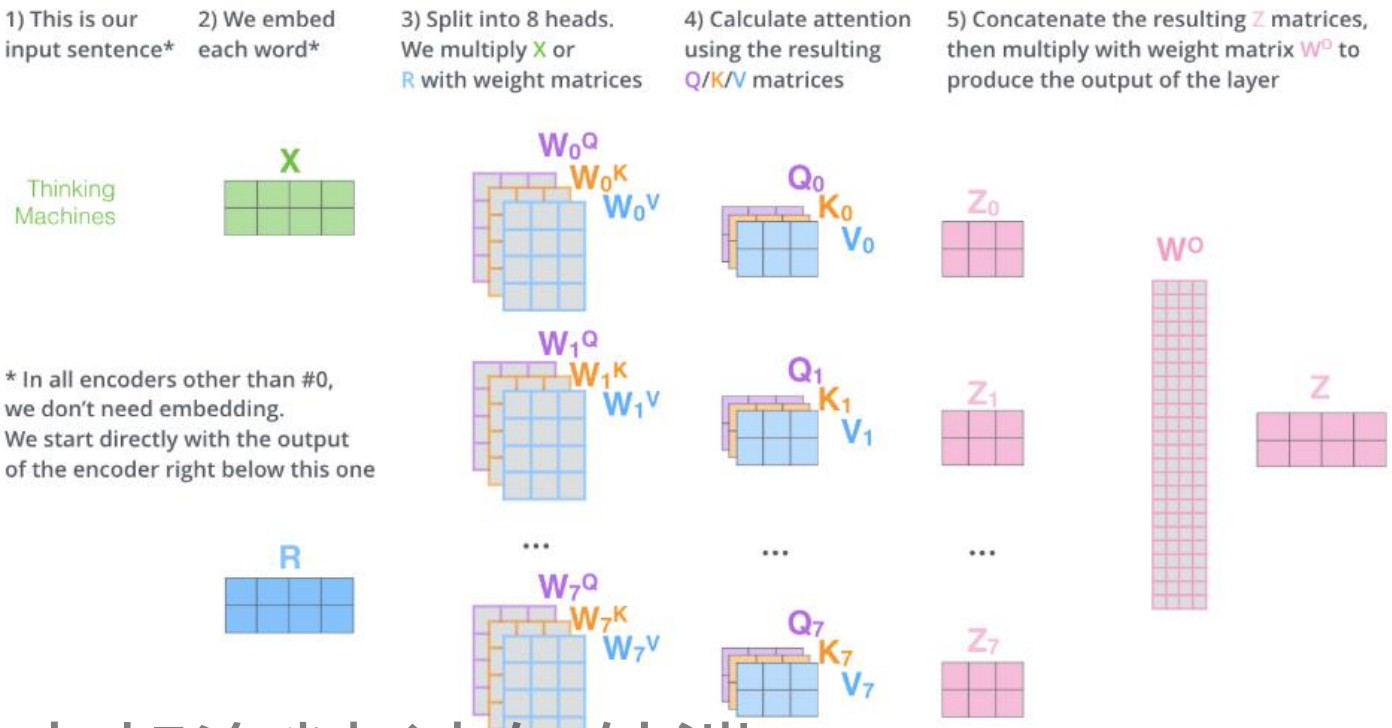
$$Z = softmax\left(\frac{Q \times K^T}{\sqrt{d_k}}\right)V$$

Step 5. Multiply each value vector by the SoftMax score to keep intact the values of the word(s) we want to focus on, and drown-out irrelevant words

Step 6. Sum up the weighted value vectors to produce self-attention outputs(Z_i)

Multi-head Self-Attention

- Expands the model's ability to focus on different positions.
- Gives the attention layer multiple "representation subspaces".



Position Embedding in Transformer

Why we need positional encoding in Transformer?

- A sentence of different word sequence convey a different meaning.
- All building blocks in Transformer such as self-attention is not aware of the word position information.

Why position matters?

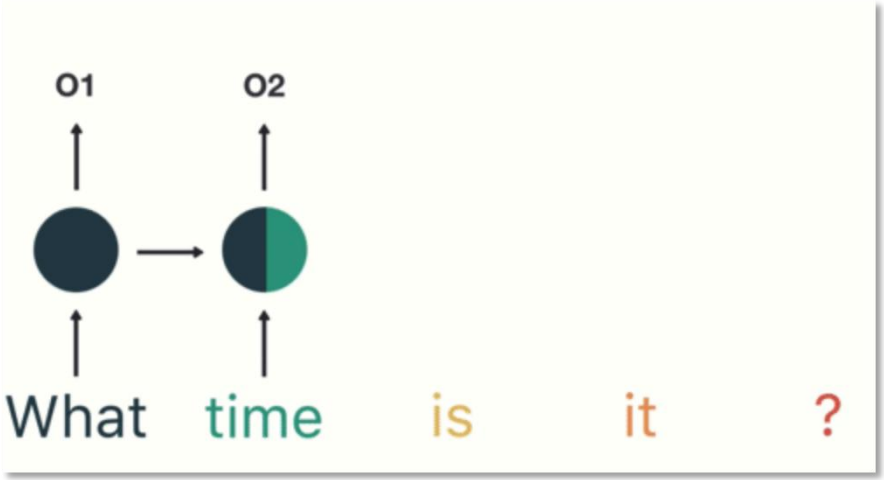
Even though she did not win the award, she was satisfied.

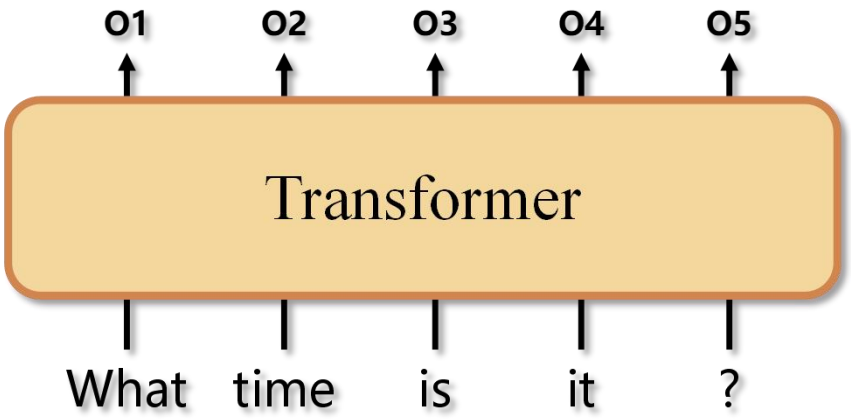
Even though she did win the award, she was not satisfied.

Positional Encoding in Transformer

Think: Why does RNN has positional information?

- The words are inputted one-by-one into the RNN model.





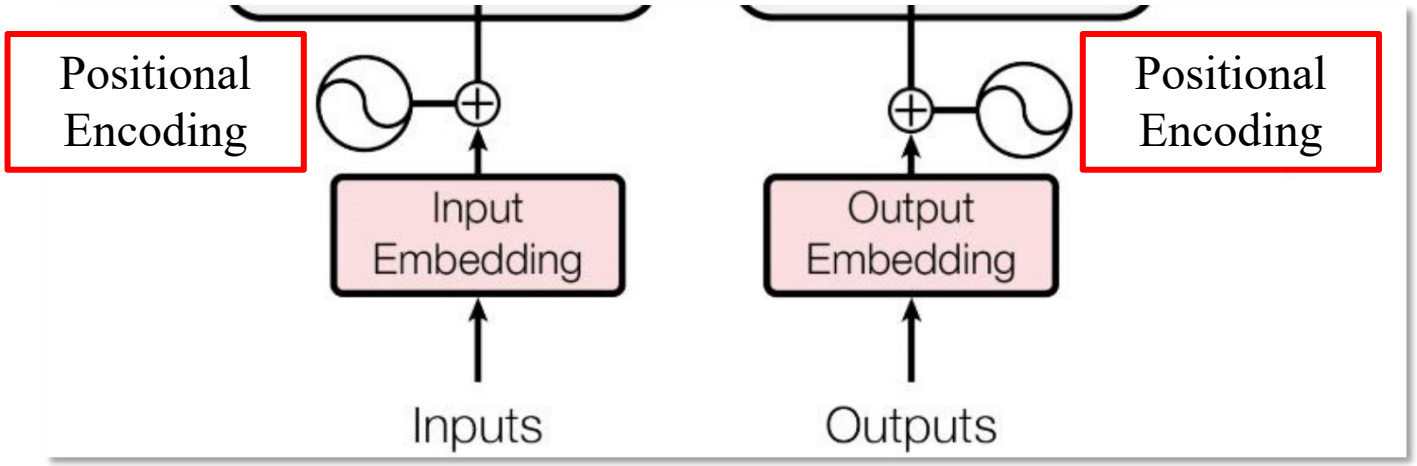
In RNN, one word's representation is based on the previous ones.

In Transformer, all words are inputted and processed simultaneously.

Positional Encoding in Transformer

How to add the position information efficiently into Transformer?

- Add on the input level: **Positional Encoding Layer**.
- Why efficient? Does not modify the Transformer architecture.

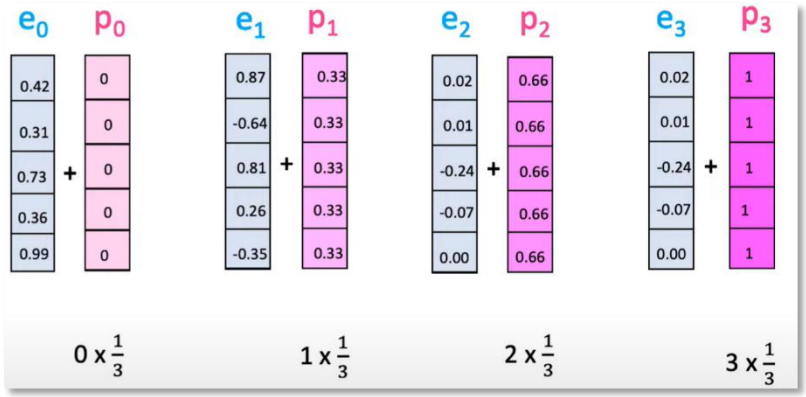


The positional encoding layer encodes the position information and generates position embeddings of inputs.

Positional Encoding in Transformer

Trivial Solutions

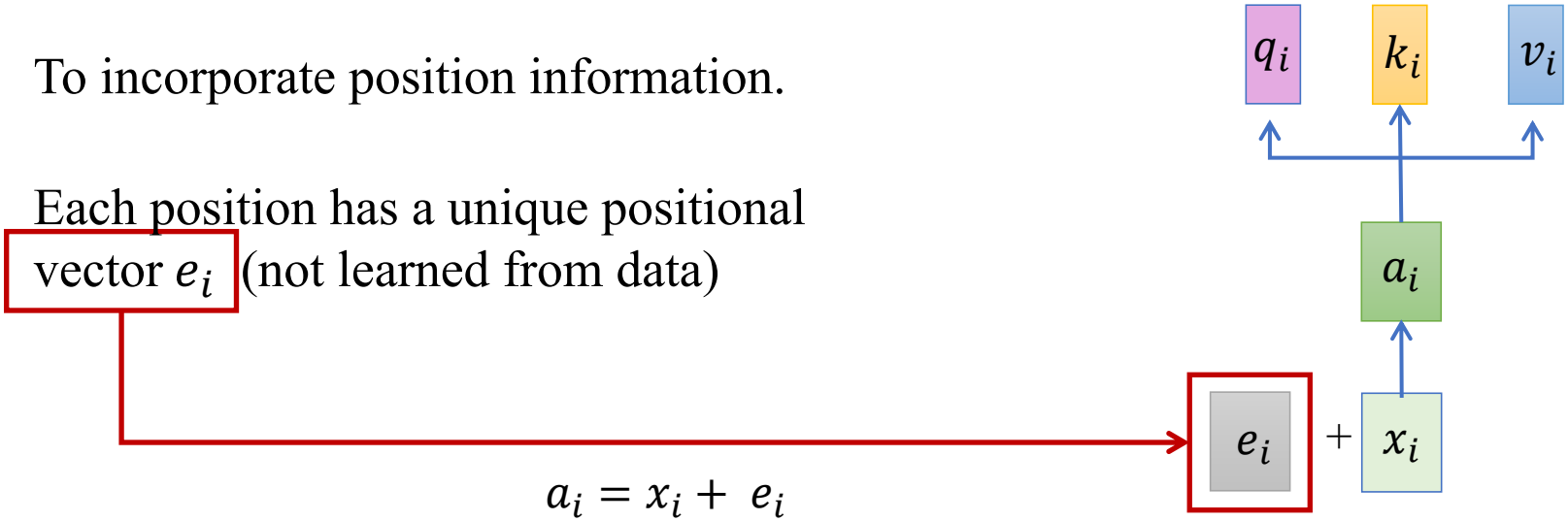
- Increase by word IDs: The words latter in the text have huge embeddings.
- Normalize by sentence length: The same word of the same position in sentences with different lengths does not has the same embedding



Ideally, the position embedding values
(1) Should have **similar scale** in different position
(2) Should **remain the same at a given position** irrespective of the text total length

Positional Encoding in Transformer

- To incorporate position information.
- Each position has a unique positional vector e_i (not learned from data)



$$a_i = x_i + e_i$$

$$e_{i,2j} = \sin(i/10000^{2j/d})$$

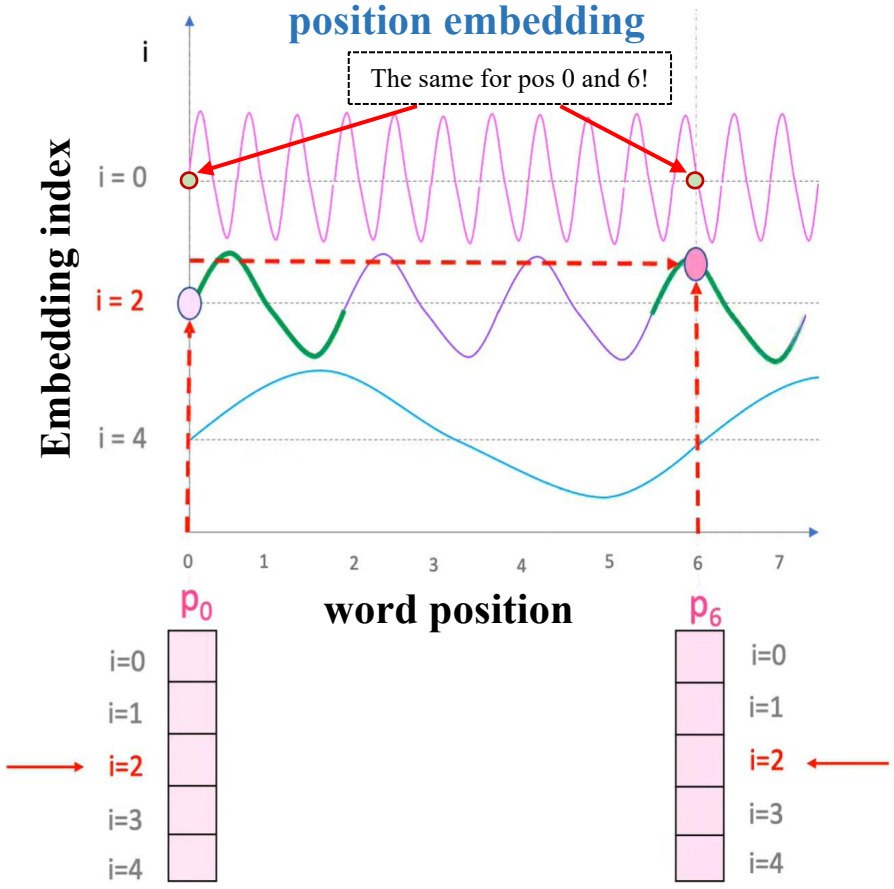
$$e_{i,2j+1} = cos(i/10000^{2j/d})$$

Contain two parts
(1) Sin part for even position
(2) Cos part for odd position

Where i is the position, j is the dimension.
The positional encodings e_i have the same dimension d as the word embedding x_i.

Positional Encoding in Transformer

- A deep look of encoding function (sin part)



Position Embedding is a 2D matrix

- Axis *i*: embedding index (0, 1, …, 128)
- Axis *pos*: word position (0, 1, …, ∞)

Why two axes?

(1) The axis *pos* is required to distinguish different word position.

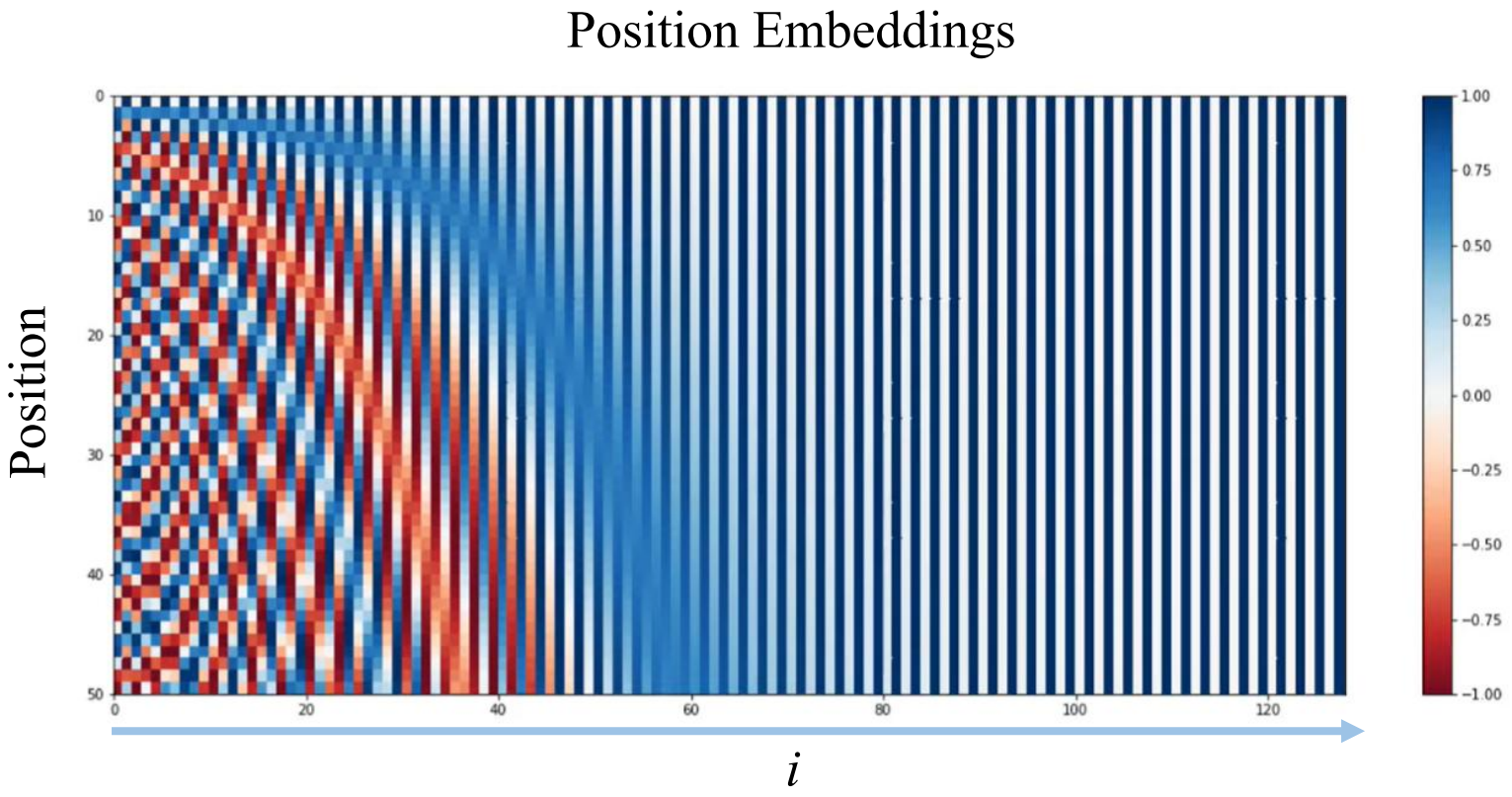
(2) Then, why we need the axis *i* ?

Answer: Sin is periodic function!

- At pos 0 and 6, position embedding values are the same at index 0.
- At pos 0 and 6, position embedding values are differ at index 2.

Positional Encoding in Transformer

- Visualization of $PE_{(pos,2i)} = \sin(\frac{pos}{10000^{\frac{2i}{d}}})$



Positional Encoding in Transformer

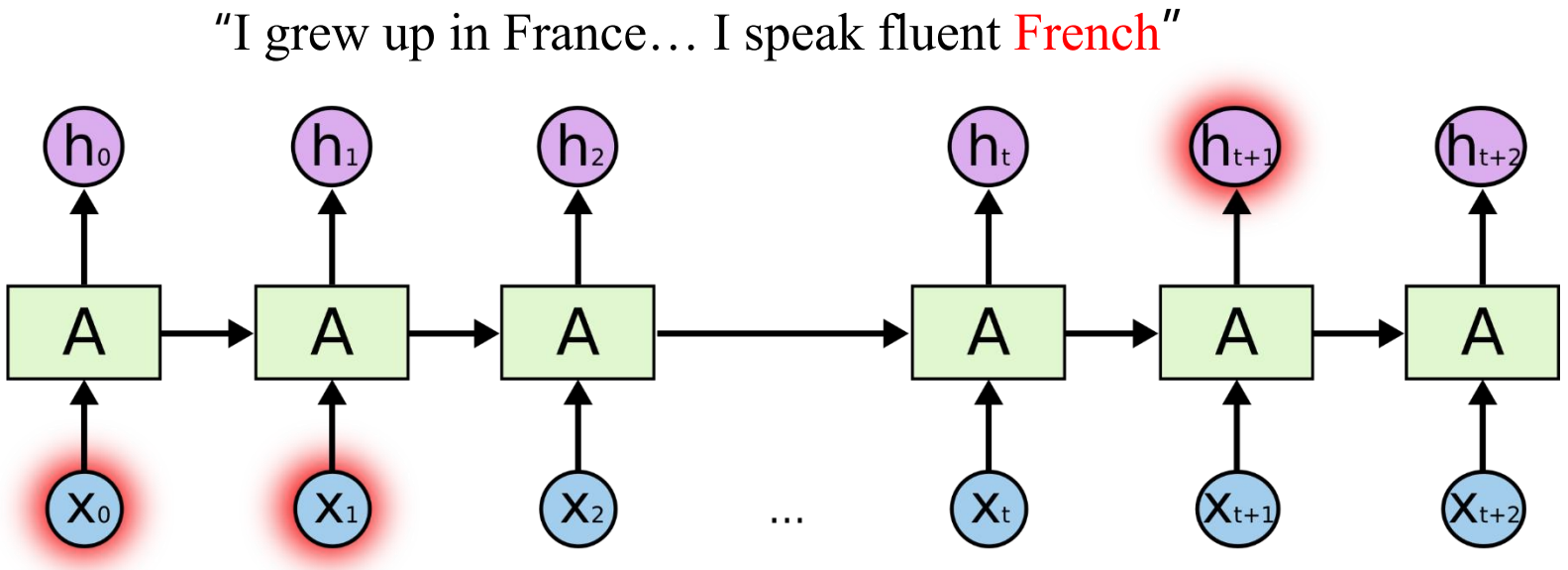
- Implementation of positional encoding layer

```
class PositionalEncoding(nn.Module):
    def __init__(self, emb_size: int, dropout, maxlen: int = 5000):
        super(PositionalEncoding, self).__init__()
        den = torch.exp(- torch.arange(0, emb_size, 2) * math.log(10000) / emb_size)
        pos = torch.arange(0, maxlen).reshape(maxlen, 1)
        pos_embedding = torch.zeros((maxlen, emb_size))
        pos_embedding[:, 0::2] = torch.sin(pos * den)
        pos_embedding[:, 1::2] = torch.cos(pos * den)
        pos_embedding = pos_embedding.unsqueeze(-2)

        self.dropout = nn.Dropout(dropout)
        self.register_buffer('pos_embedding', pos_embedding)

    def forward(self, token_embedding: Tensor):
        return self.dropout(token_embedding +
                            self.pos_embedding[:token_embedding.size(0),:])
```

Outline

Long-Term Dependency

- Long distance between previous context and present location.

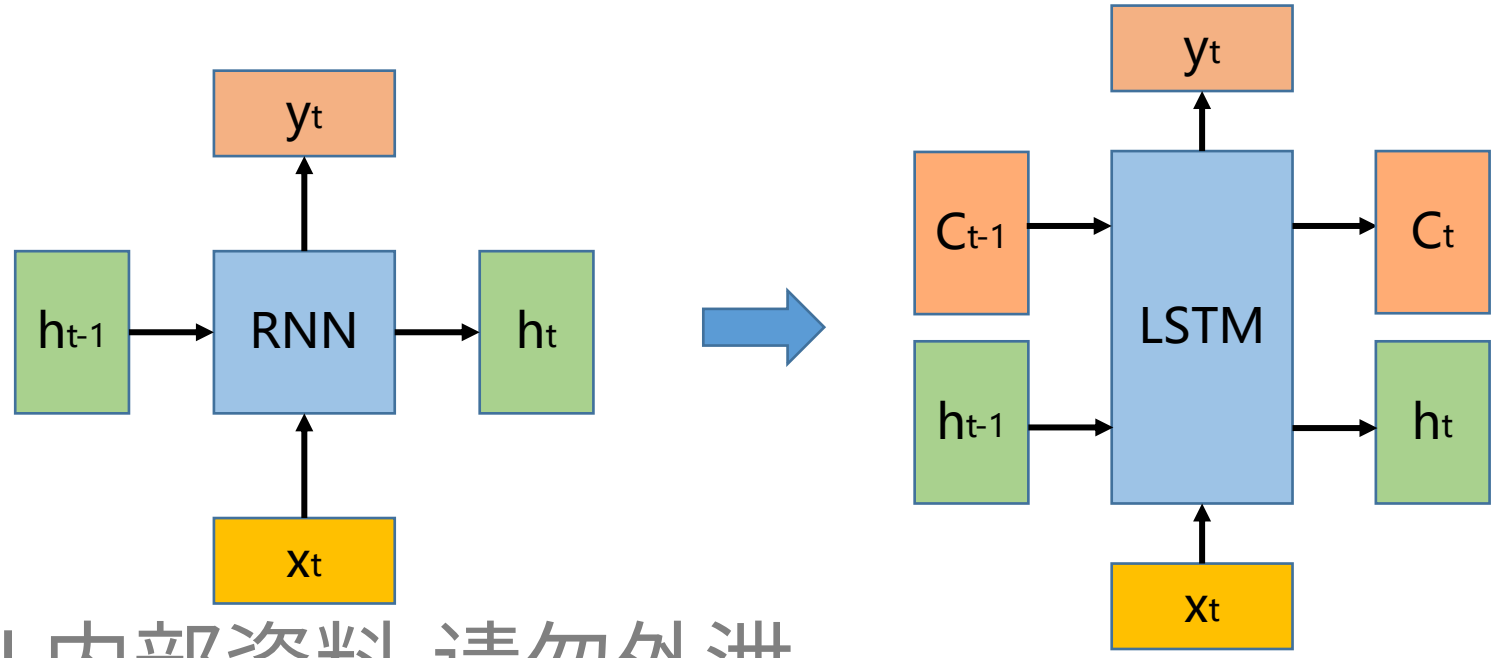


- RNN is unable to connect to information too far ahead.

Long Short-Term Memory Model

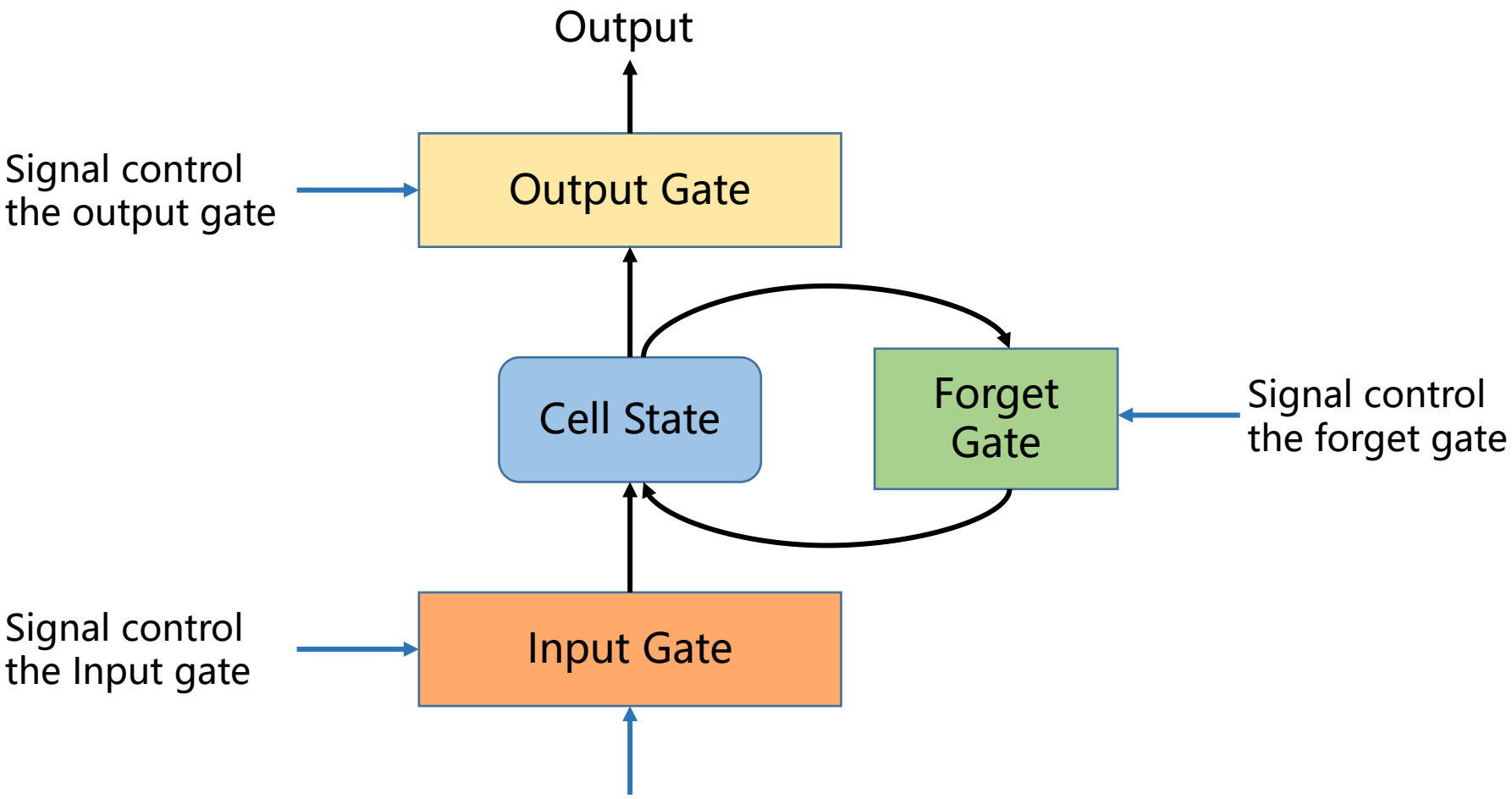
- Long Short-term Memory Models (LSTMs) make dependencies cross long distance easier to learn.

- LSTM has two states $h_t(hidden\ state)$ and $C_t(cell\ state)$, while RNN only has one state h_t.



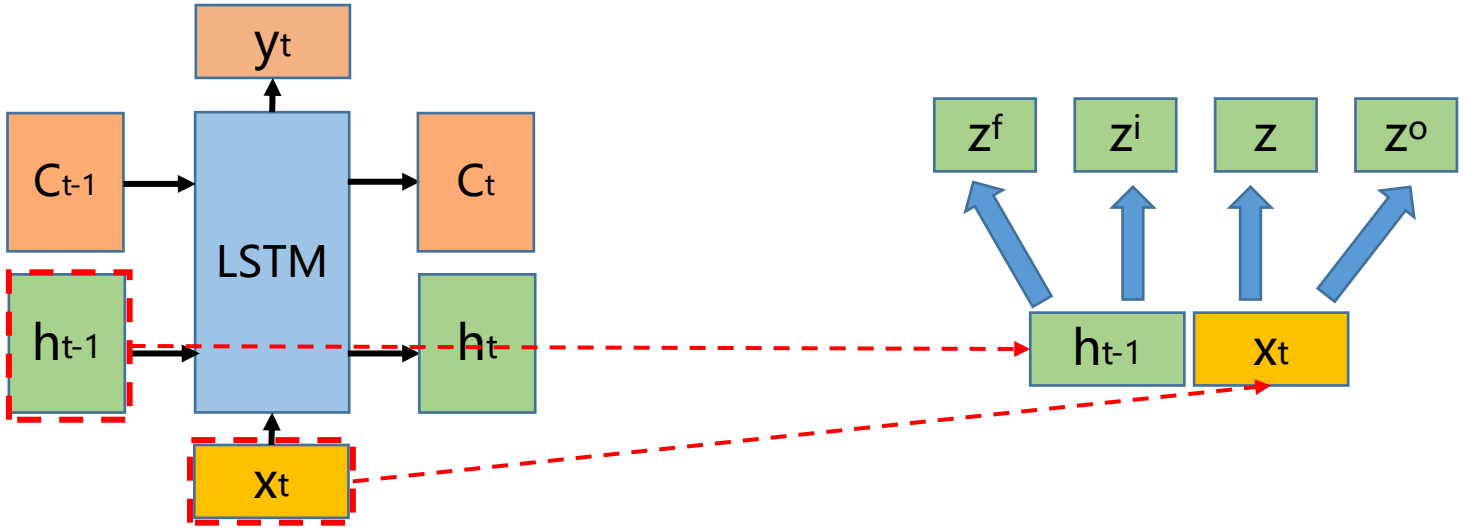
Long Short-Term Memory Model

- Basic structure of LSTMs



Long Short-Term Memory Model

- Three gates of LSTM



Input gate: $\mathbf{z^i} = \sigma(W_i\,[h_{t-1}, x_t] + b_i)$

Forget gate: $\mathbf{z^f} = \sigma(W_f\,[h_{t-1}, x_t] + b_f)$

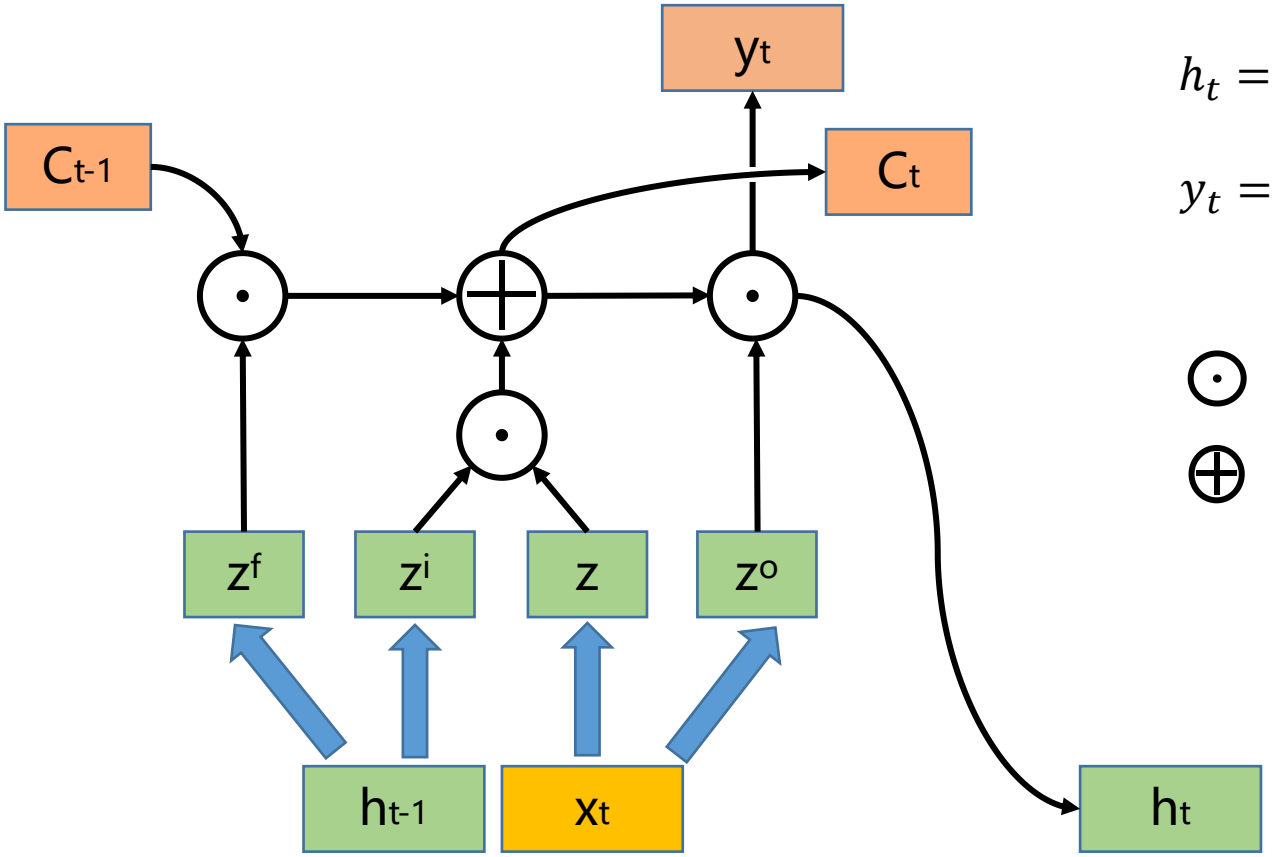
Output gate: $\mathbf{z^o} = \sigma(W_o\,[h_{t-1}, x_t] + b_o)$

Input activation: $\mathbf{z} = \tanh(W[h_{t-1}, x_t] + b)$

- Apply sigmoid function σ and z^i, z^f, $z^o \in (0,1)$, simulate the gates' open and close.
- Apply tanh activation function to get z.

Long Short-Term Memory Model

- How the gates work



$$C_t = z^f \odot C_{t-1} + z^i \odot z$$

$$h_t = z^o \odot \tanh(C_t)$$

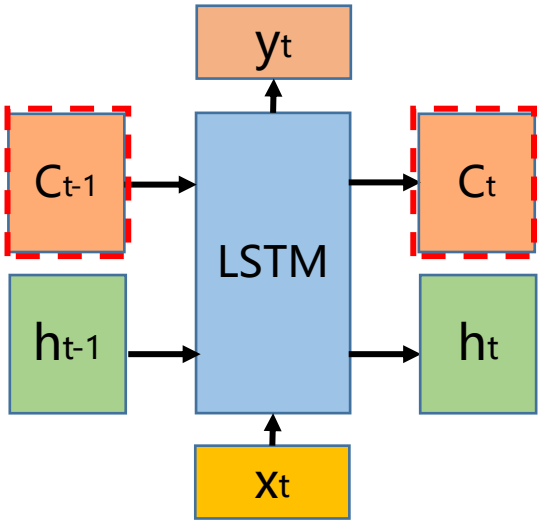
$$y_t = \sigma(W' h_t)$$

$\odot$ Hadamard Product

$\oplus$ Matrix Addition

Long Short-Term Memory Model

- How LSTMs handle long-term dependencies



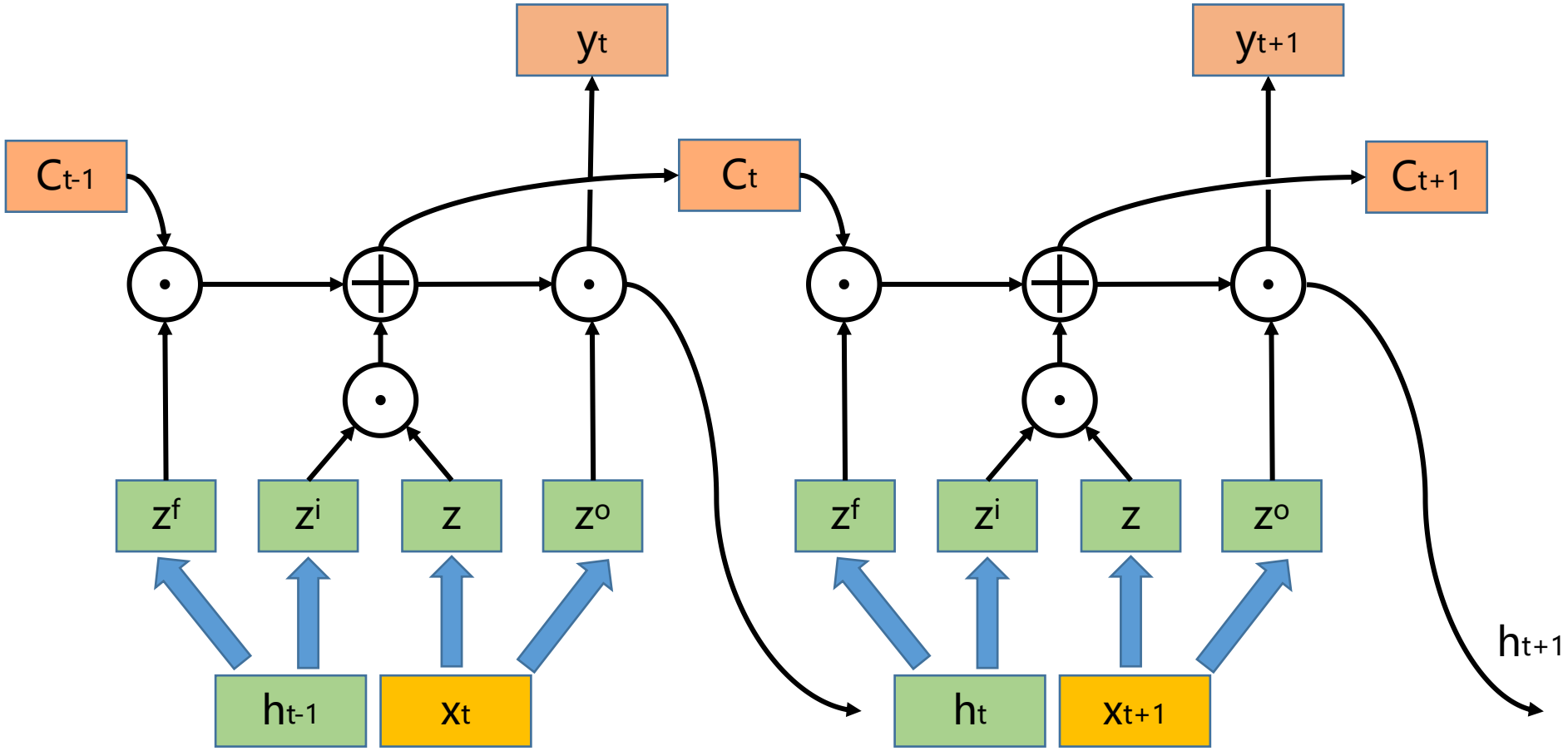
$$C_t = z^f \odot C_{t-1} + z^i \odot z$$

With the forget gate z^f and input gate z^i, cell state C_t can selectively store the information.

- By introducing gates and cell state C_t, LSTMs are able to keep previous information for long time and deal with long-term dependencies.

Long Short-Term Memory Model

- Forward Propagation Through Time (BPTT)



Forward Propagation: Prediction for Networks

- Given input a seqence of words $x_1, \cdots, x_T$, the forward propagation of a LSTM cell is to predict the output y_1 of x_1 .

- Detailed prediction process

for t from 1 to T do

$\mathbf{z^i} = \sigma(W_i\,[h_{t-1}, x_t] + b_i)$,

$\mathbf{z^f} = \sigma(W_f\,[h_{t-1}, x_t] + b_f)$,

$\mathbf{z^o} = \sigma(W_o[h_{t-1}, x_t] + b_o)$,

$\mathbf{z} = \tanh(W[h_{t-1}, x_t] + b)$

$\boldsymbol{C_t} = z^f \odot C_{t-1} + z^i \odot z$

$\boldsymbol{h_t} = z^o \odot \tanh(C_t)$

$\boldsymbol{y_t} = \sigma(W' h_t)$

end for

σ: sigmoid function

tanh: hyperbolic tangent function

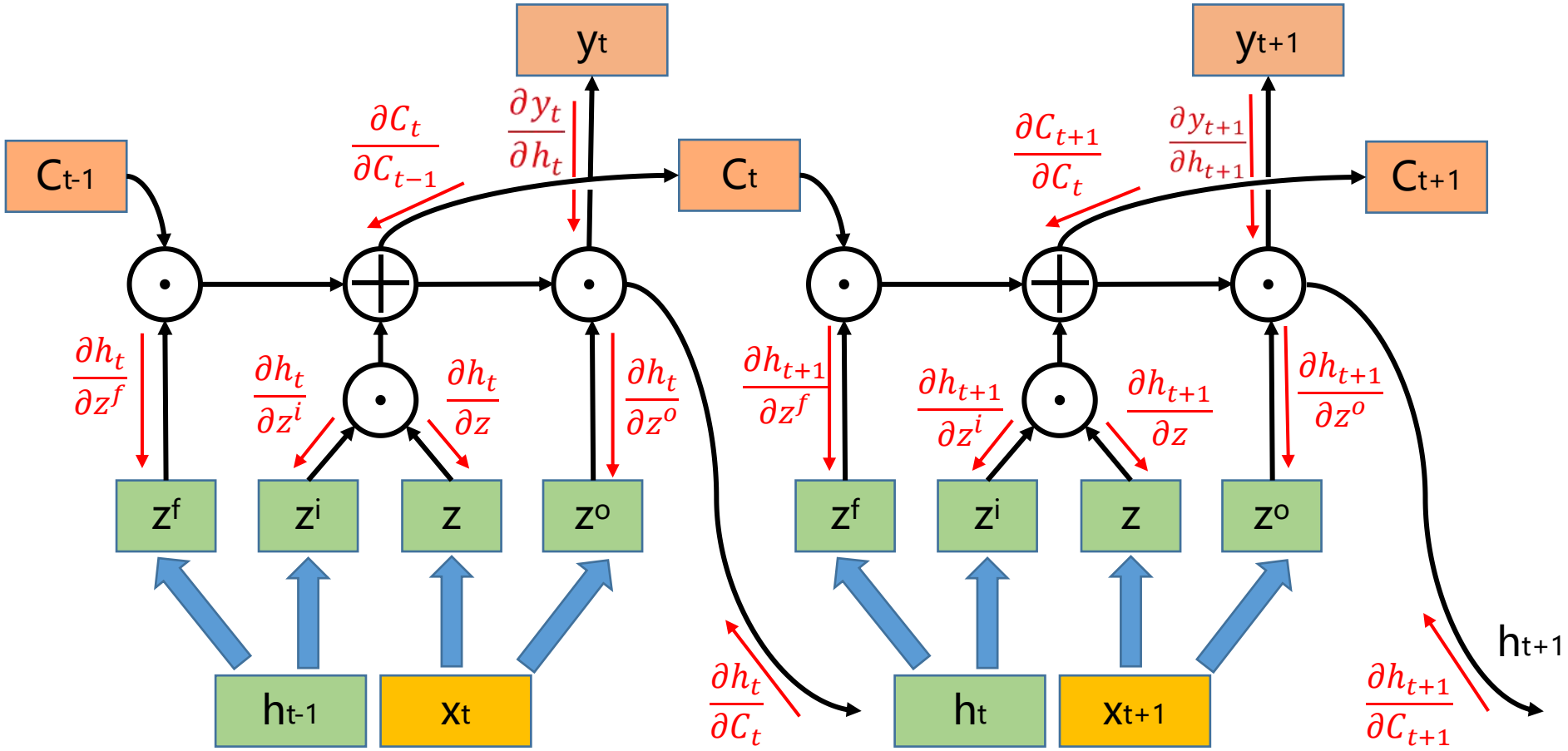
- Total loss: $L(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \sum_{t=1}^{T} L(\boldsymbol{y}_t, \hat{\boldsymbol{y}}_t)$

$\hat{\boldsymbol{y}}_t$: ground-truth label

$\boldsymbol{y}_t$: predicted label

Long Short-Term Memory Model

- Backward Propagation Through Time (BPTT)



Backpropagation of Neural Networks

- Let us resort to Chain Rule to compute the gradient.

$$\frac{\partial L_3(y,\hat{y})}{\partial W'} = \frac{\partial L_3(y,\hat{y})}{\partial y_3}\frac{\partial y_3}{\partial W'}$$

Use $L_3(\boldsymbol{y},\hat{\boldsymbol{y}})$ as an example

$$\frac{\partial L_3(y,\hat{y})}{\partial W} = \sum_{k=0}^{3}\frac{\partial L_3(y,\hat{y})}{\partial y_3}\frac{\partial y_3}{\partial h_3}\frac{\partial h_3}{\partial h_k}\frac{\partial h_k}{\partial C_k}\frac{\partial C_k}{\partial z}\frac{\partial z}{\partial W}$$

$$\frac{\partial L_3(y,\hat{y})}{\partial W_i} = \sum_{k=0}^{3}\frac{\partial L_3(y,\hat{y})}{\partial y_3}\frac{\partial y_3}{\partial h_3}\frac{\partial h_3}{\partial h_k}\frac{\partial h_k}{\partial C_k}\frac{\partial C_k}{\partial z^i}\frac{\partial z^i}{\partial W_i}$$

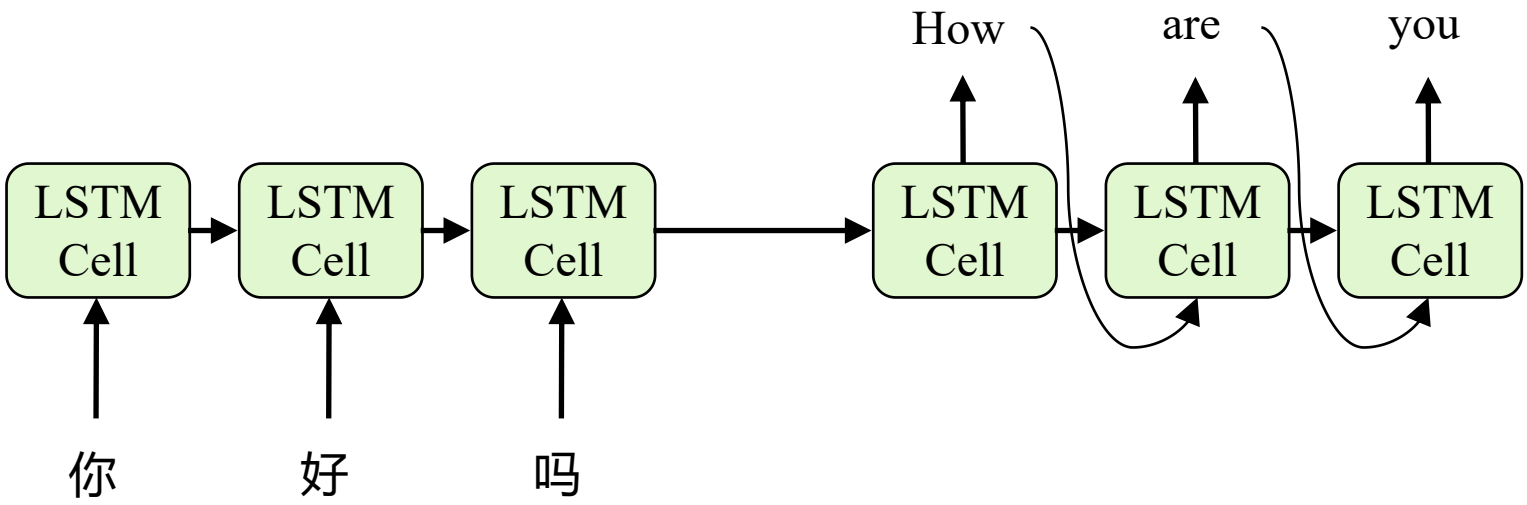
$$\frac{\partial L_3(y,\hat{y})}{\partial W_f} = \sum_{k=0}^{3}\frac{\partial L_3(y,\hat{y})}{\partial y_3}\frac{\partial y_3}{\partial h_3}\frac{\partial h_3}{\partial h_k}\frac{\partial h_k}{\partial C_k}\frac{\partial C_k}{\partial z^f}\frac{\partial z^f}{\partial W_f}$$

$$\frac{\partial L_3(y,\hat{y})}{\partial W_o} = \sum_{k=0}^{3}\frac{\partial L_3(y,\hat{y})}{\partial y_3}\frac{\partial y_3}{\partial h_3}\frac{\partial h_3}{\partial h_k}\frac{\partial h_k}{\partial C_k}\frac{\partial C_k}{\partial z^o}\frac{\partial z^o}{\partial W_o}$$

Application

What can LSTM do?

Predict a sequence:



- Neural machine translation.
- Encoder-decoder structure.

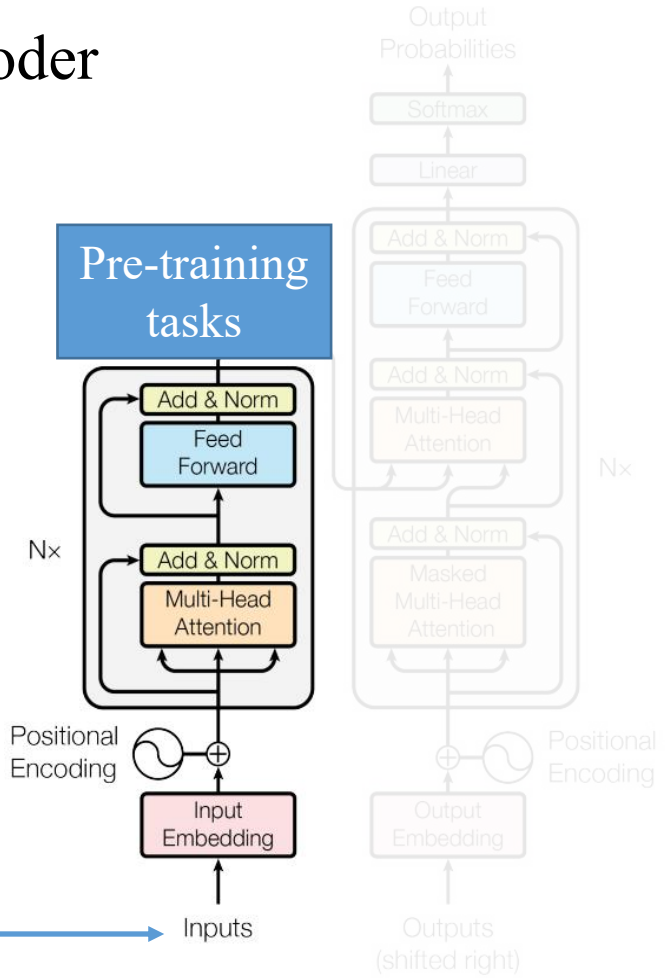
Outline

BERT

BERT = Pre-trained Transformer Encoder

The same as Transformers:

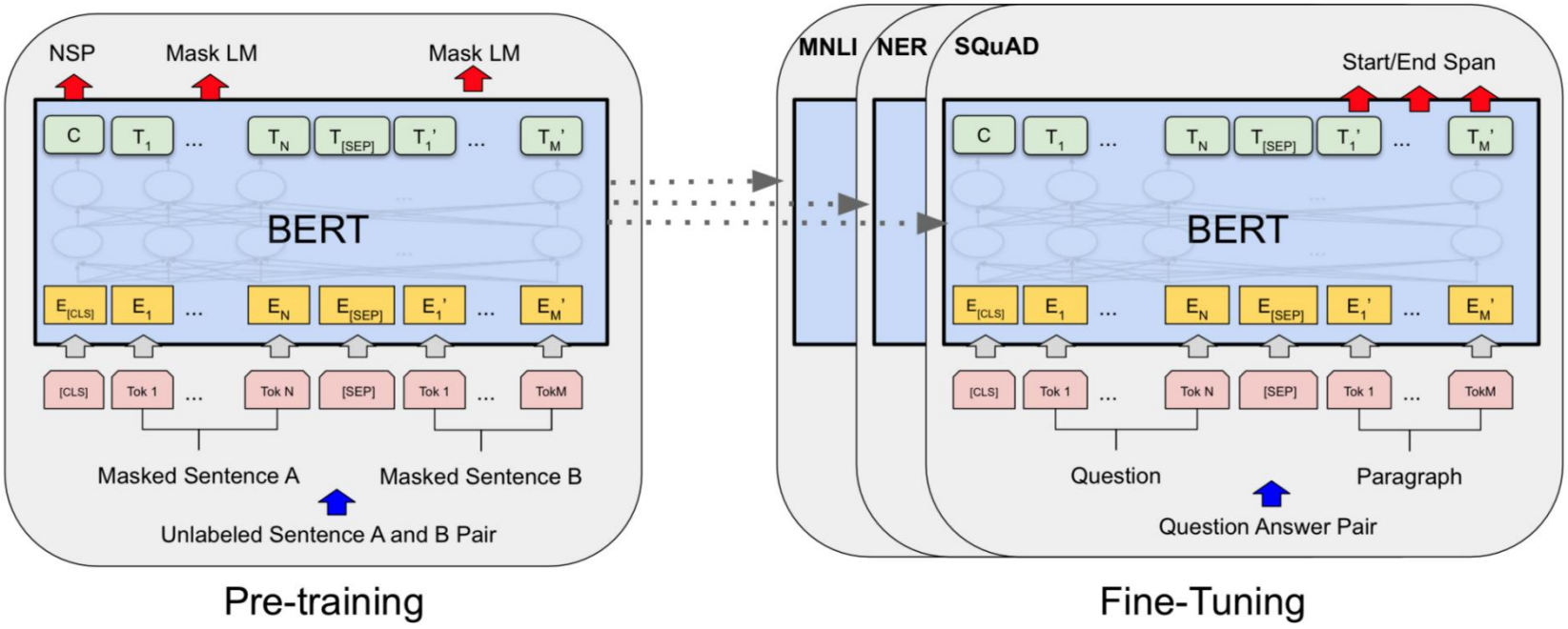
- Multi-head self attention
- Feed-forward layers
- Layer normalization and residual connections
- Input Embedding



large amount of text without annotation →

BERT

Pre-training, then fine-tuning on various downstream tasks.



Task #1: **Masked Language Model** (cloze task)

How to best utilize the **bi-directional** architecture of Transformers using **pre-training**?

- Mask out k% of the input words, then predict the masked words.
- The prediction is conditioned on context from both direction.

BERT always use k=15%

The dog [MASK] across the street because it was [MASK] tired.

↓ didn't

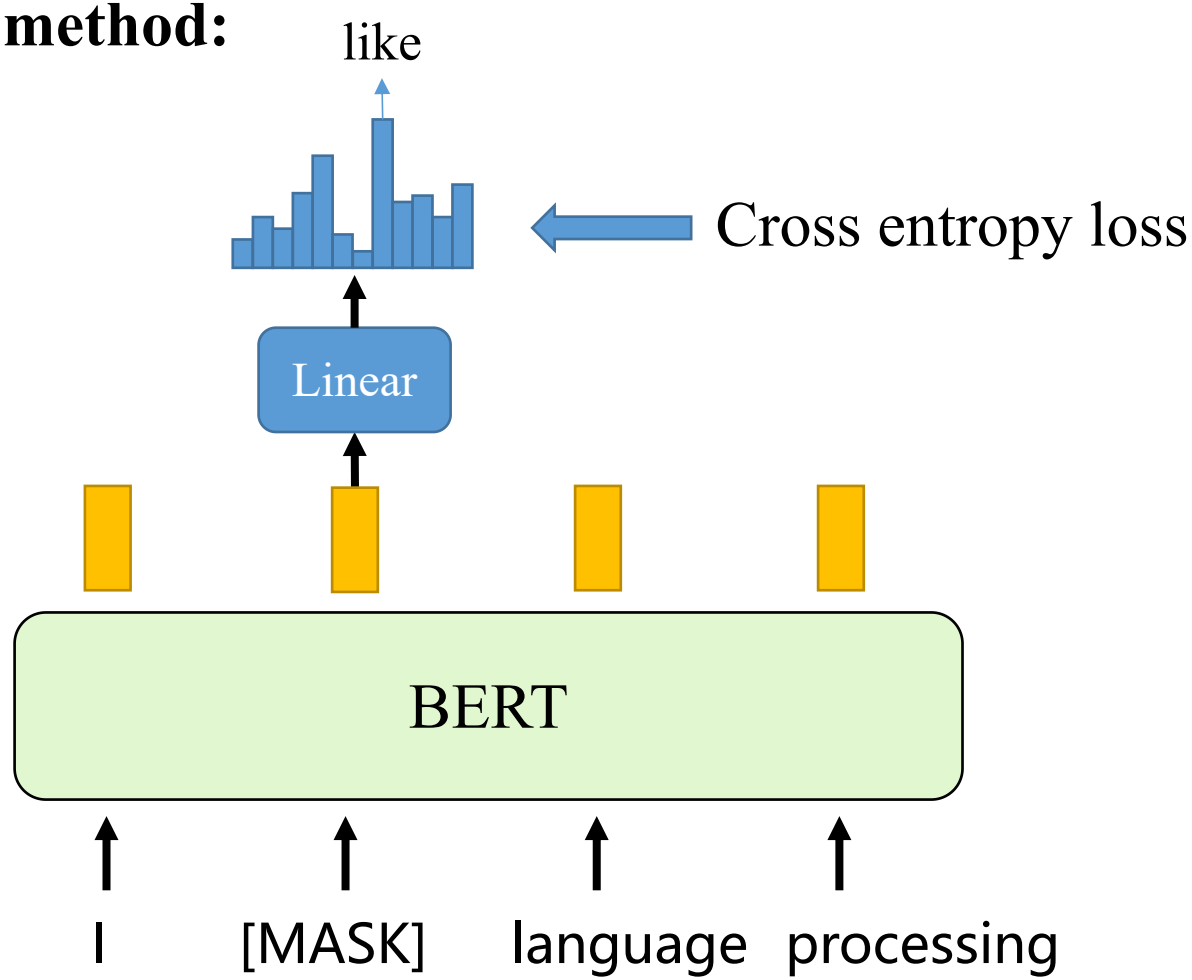
↓ too

Too little masking: too expensive to train

Too much masking: not enough context

Task #1: **Masked Language Model** (cloze task)

Training method:



Task #1: **Masked Language Model** (cloze task)

- **Problem**: [MASK] does not appear during fine-tuning.
- **Solution**: The training data generator randomly chooses 15% of the token positions for prediction.

1. 80% of the time, replace with [MASK]

   my dog is cute → my dog is [MASK]

2. 10% of the time, replace random word

   my dog is cute → my dog is apple

3. 10% of the time, keep same

   my dog is cute → my dog is cute

Task #2: **Next Sentence Prediction**

Motivation: understanding the relationship between two sentences is important to some tasks, e.g.:

- Question Answering
- Natural Language Inference

Choose two sentence A, B:

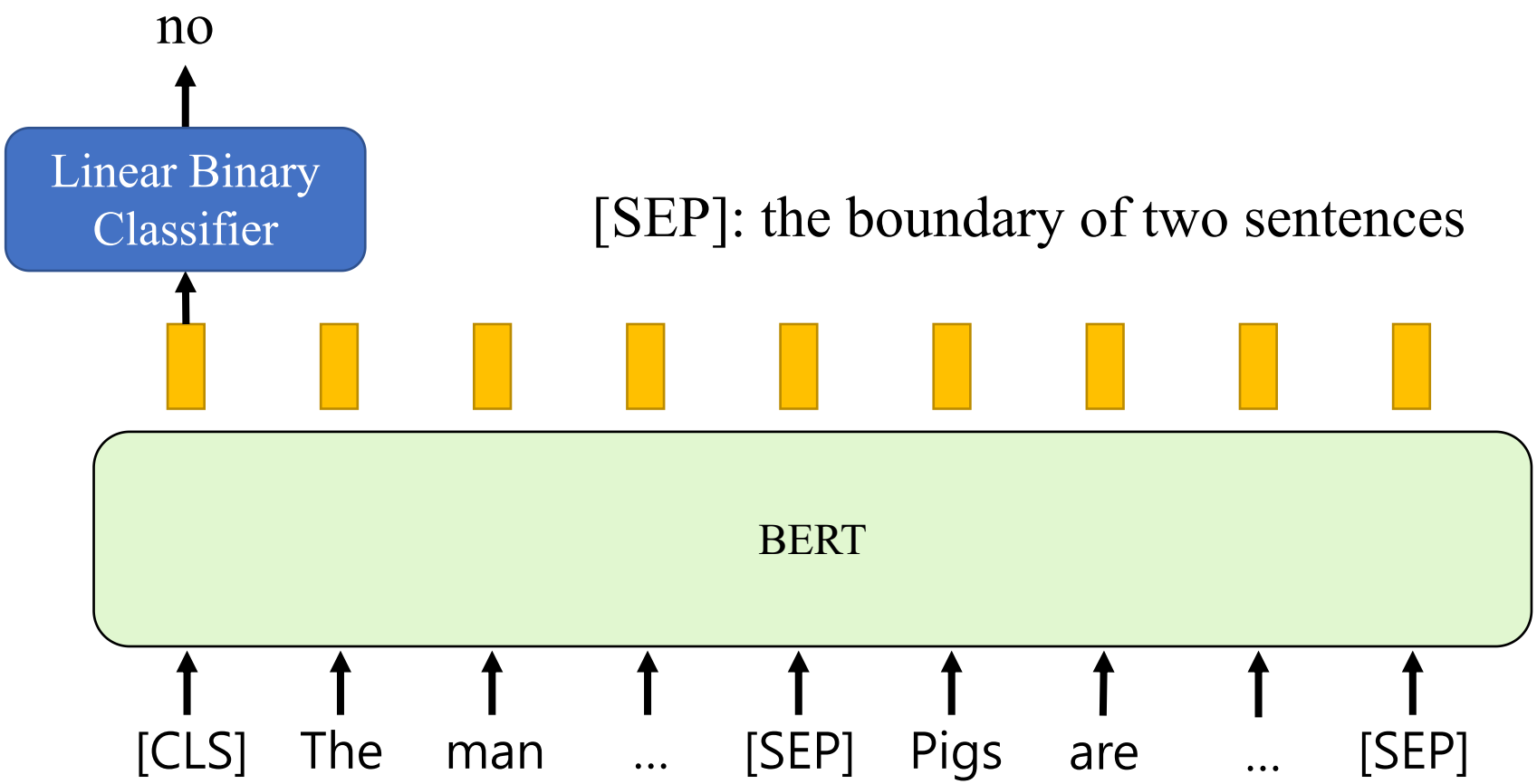
- 50% of the time, B is the actual next sentence that follows A.
- 50% of the time, B is a random sentence from the corpus.
- Let neural networks predict which is the case.

Sentence A = The man went to the store.
Sentence B = He bought a glass of milk.
Label = IsNextSentence

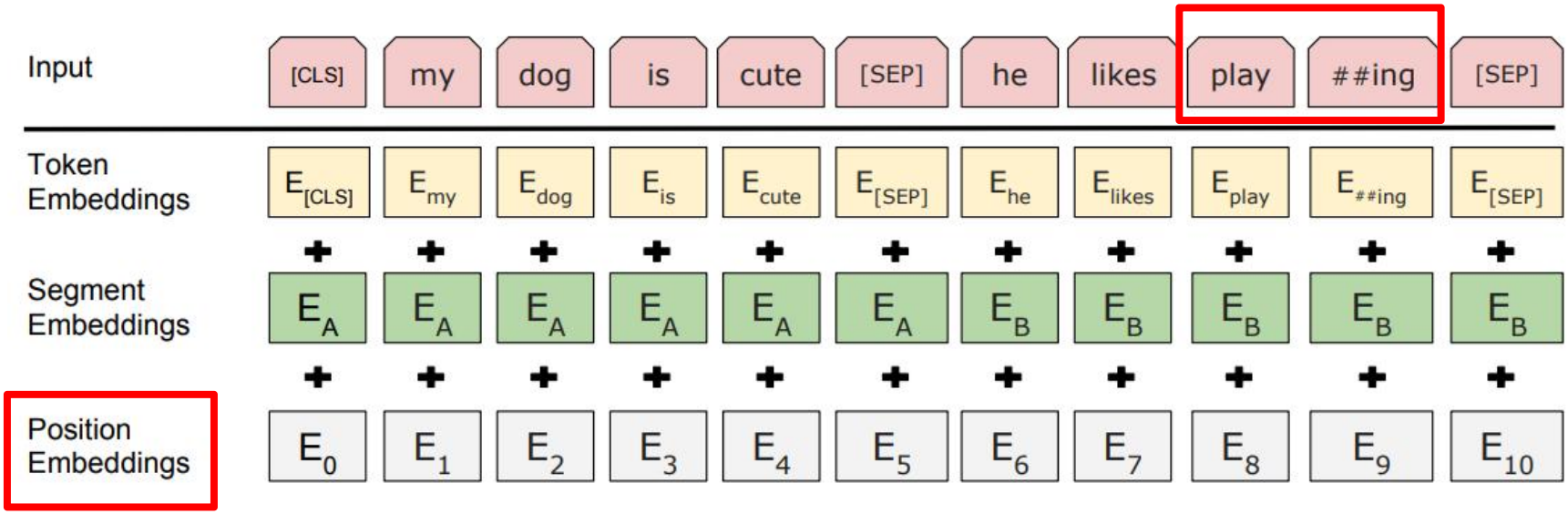
Sentence A = The man went to the store.
Sentence B = Pigs are flightless.
Label = NotNextSentence

Task #2: Next Sentence Prediction

Training method:

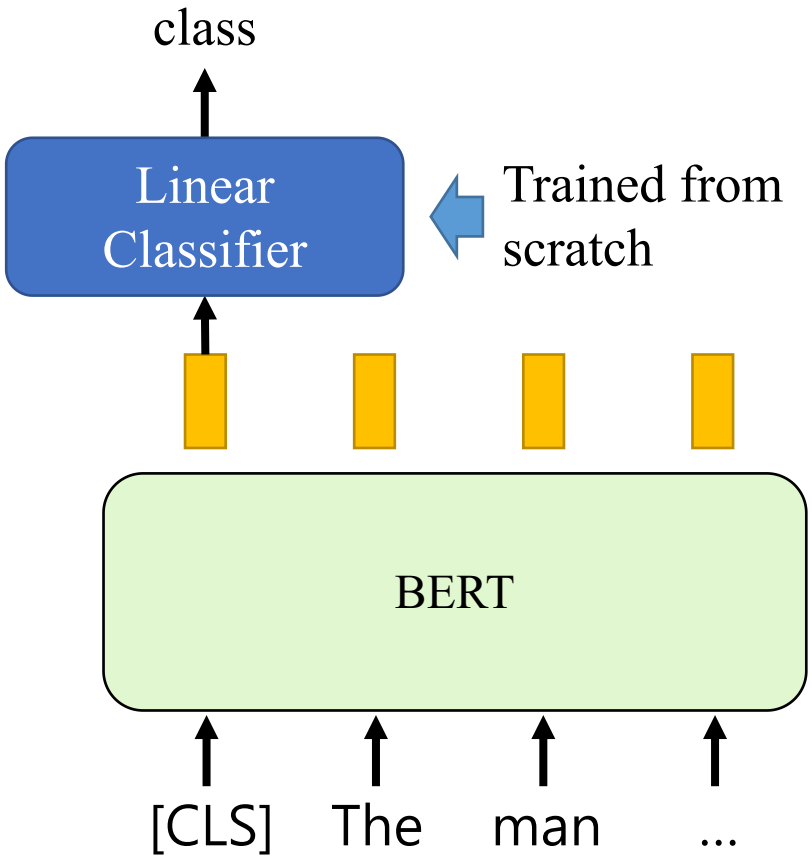


BERT



- Use WordPiece embeddings with a 30k token vocabulary.
- Segment embeddings are used to differentiate sentences.
- Position embeddings are different from Transformer.

Down-stream task #1: **Sentence classification**

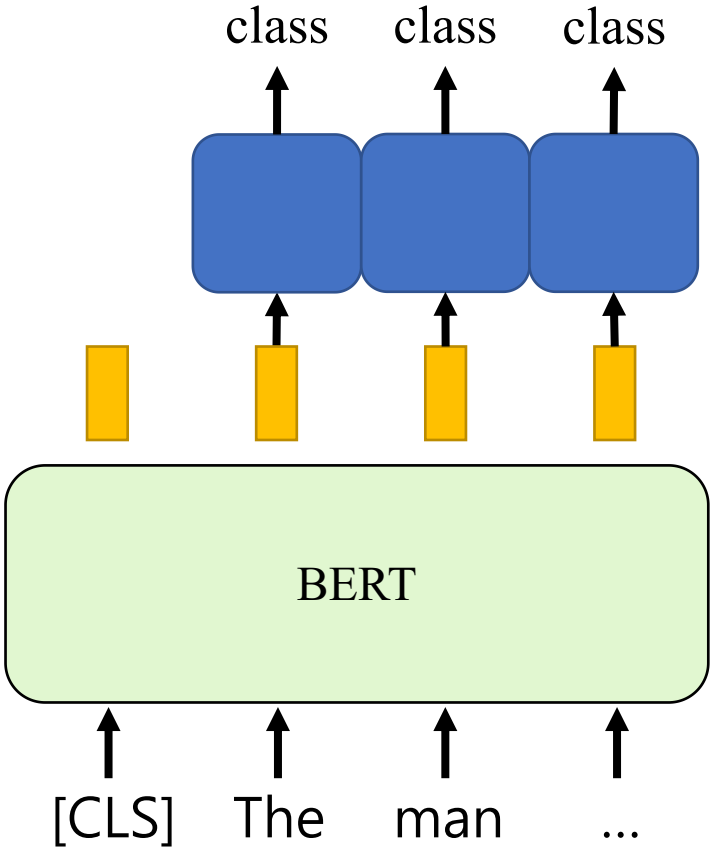


Input: A sentence or a pair of sentence
Output: class

Example: MNLI (Multi-Genre Natural Language Inference)

- **Fiction**: The Old One always comforted Jack, except today.
- **Hypothesis**: Jack knew the Old One very well.
- **Label**: neutral
  (neutral / contradiction / entailment)

Down-stream task #2: **Sentence tagging**



Input: A sentence
Output: class of each token

Example: NER (Named Entity Recognition)

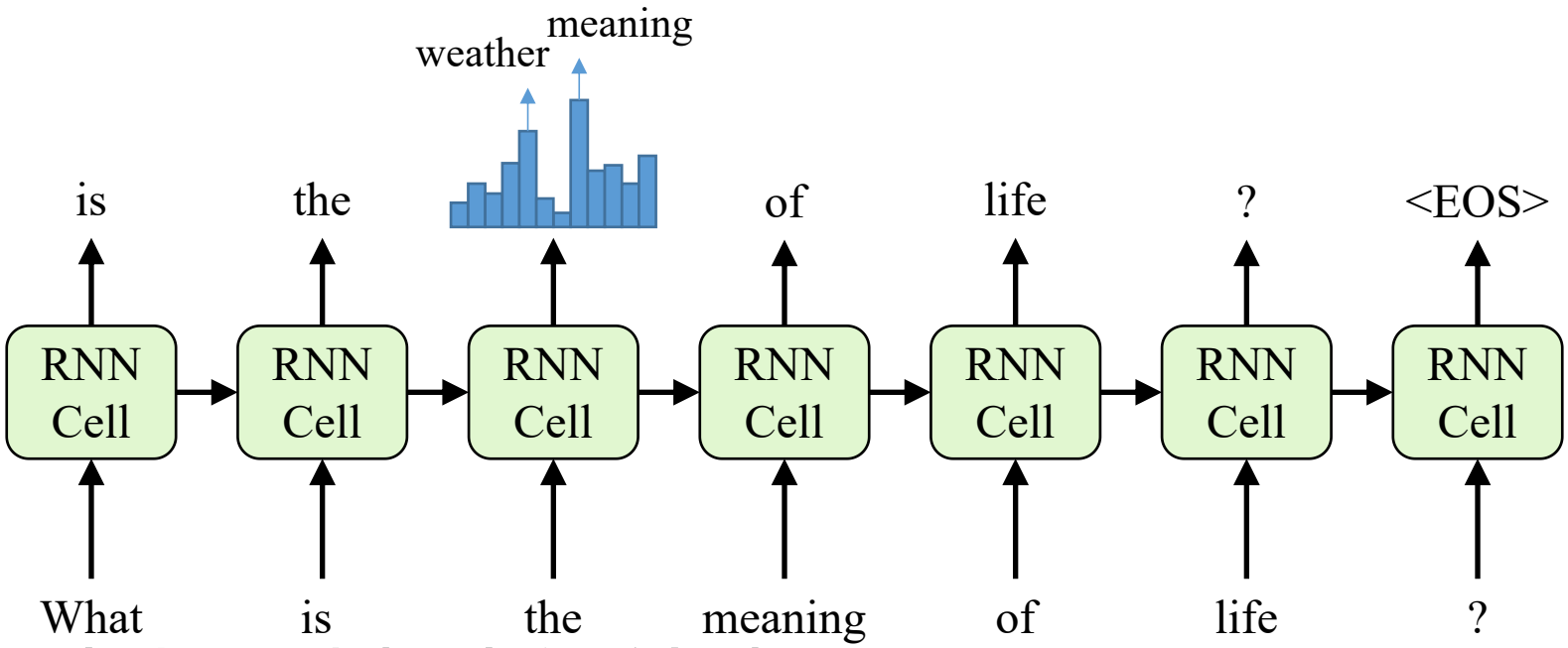
U.N. official Ekeus heads for Baghdad .
ORG - PER - - LOC -

Outline

Vanilla RNN

Vanilla Recurrent Neural Network:

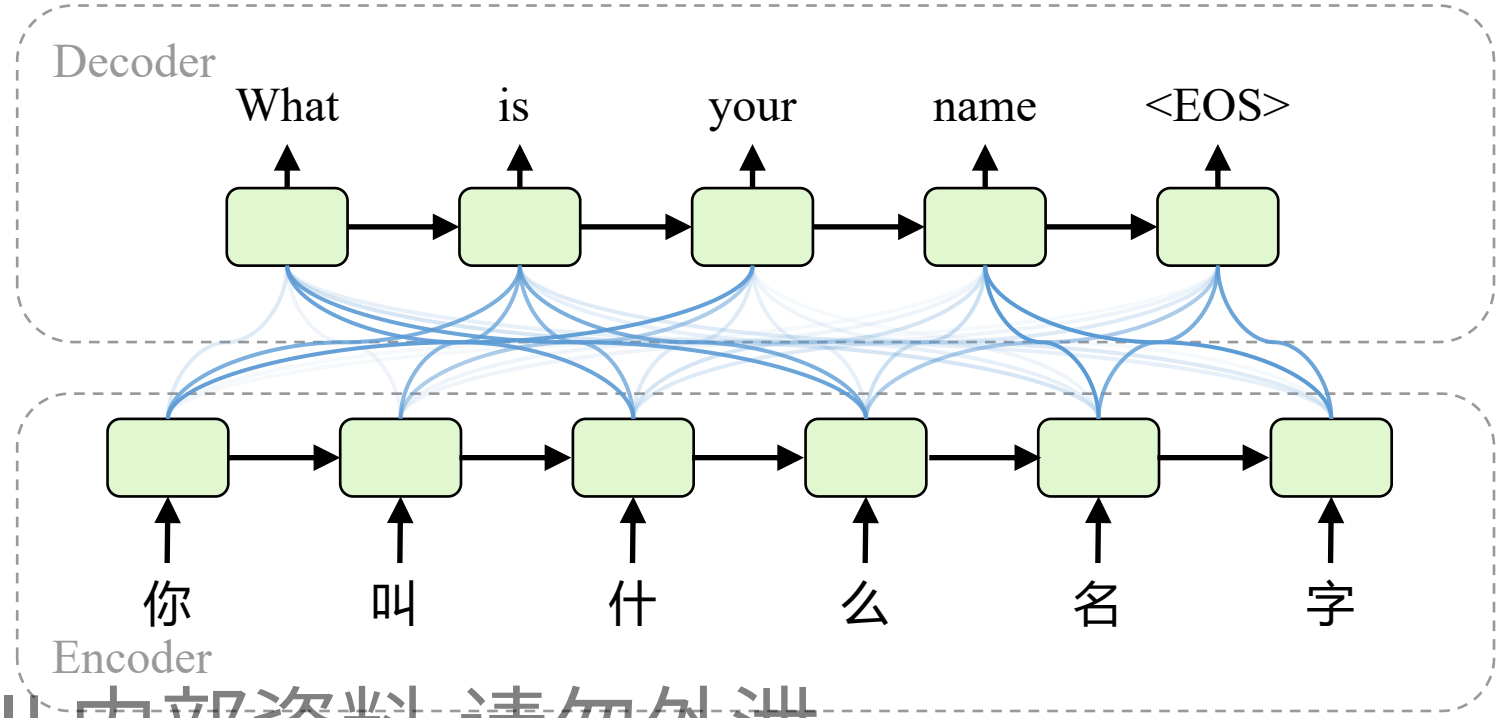
- Introduce **hidden state**, which remembers information about a sequence.
- Use **variable-length** sequences as input/output.
- Example application: Language Model (LM)



Attention

Attention in encoder-decoder

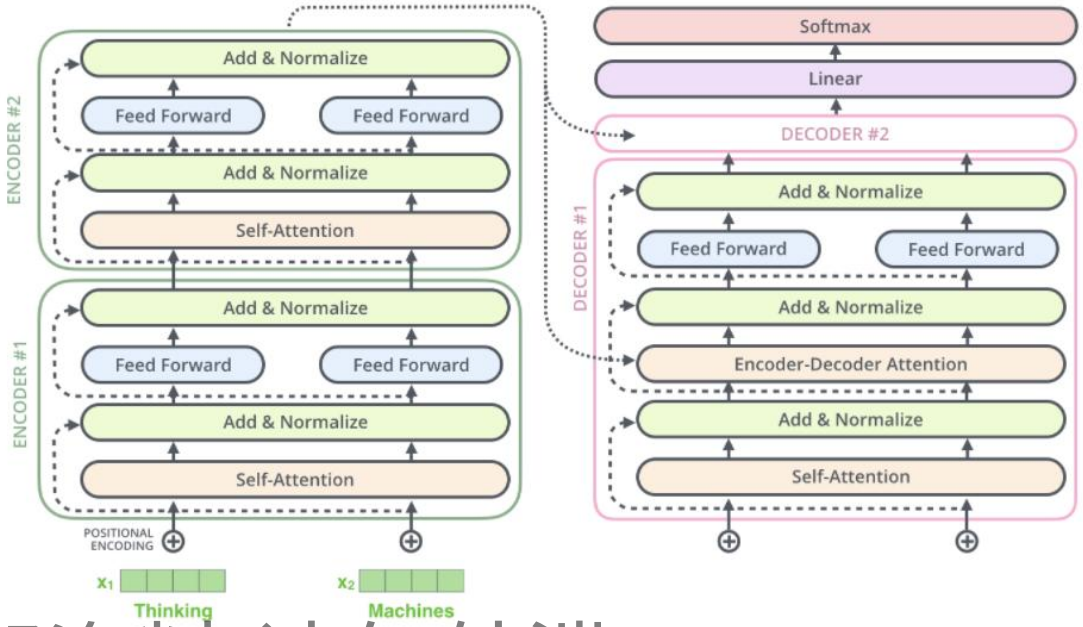
- Imitate human attention mechanism.
- Avoid encoding all information in a fixed-length vector.



Transformer

Attention is all you need: **Transformer**

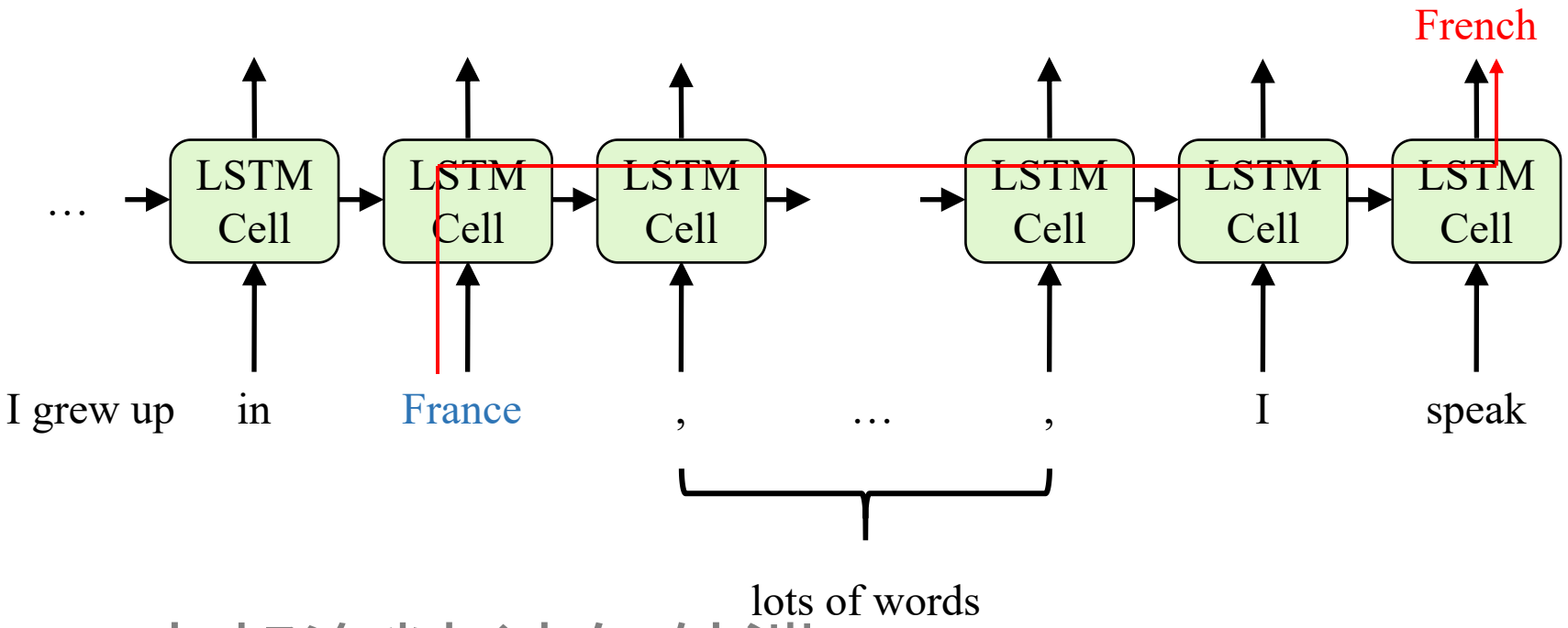
- A new simple network architecture based solely on attention mechanisms.
- The self-attention solves the problem of long distance dependence.



LSTM

Long Short-Term Memory

- Take good care of the long-term dependencies.



BERT

Bidirectional Encoder Representation from Transformers (BERT):

- Pre-trained Transformer encoders
- One pre-trained model, transferred to 11 tasks, got state-of-the-art result.