Lec12 Deep Learning
12-4 Recurrent Neural Network and Transformers

Yang Shu

School of Data Science and Engineering

yshu@dase.ecnu.edu.cn

[Acknowledgement: Slides are adapted from Deep Learning Course, Mingsheng Long, THU]

Outline

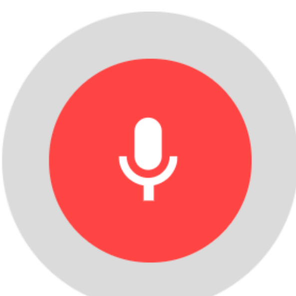
- **Recurrent Neural Network**
- Long Short-Term Memory (LSTM)
- Transformers: Attention is All You Need

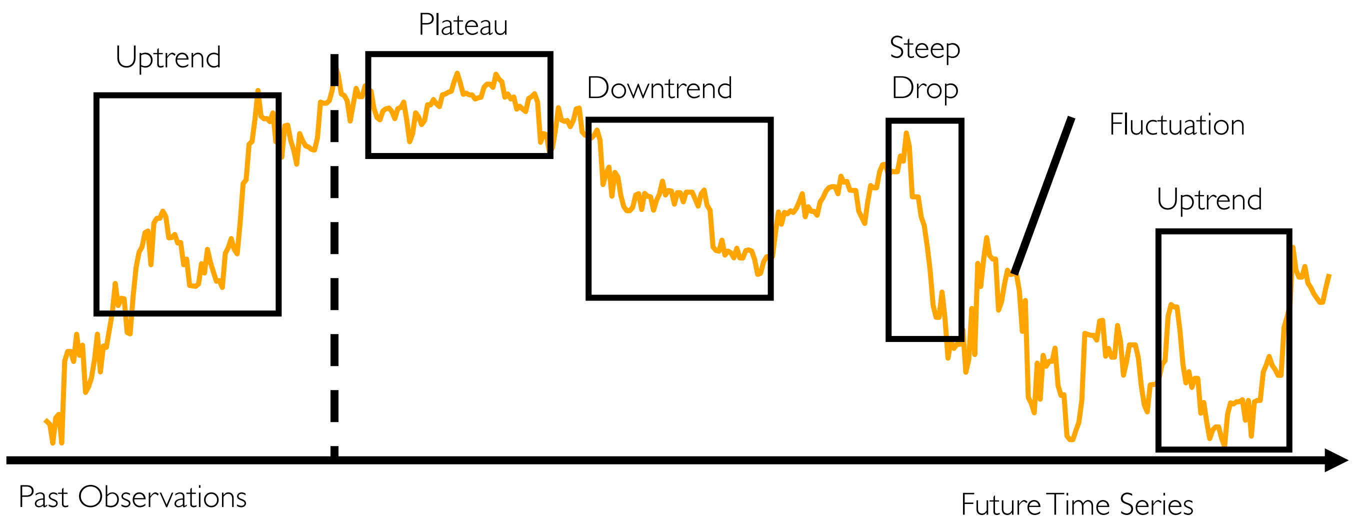
Sequence Modeling

- Modeling sequences for prediction and recognition is ubiquitous
  - Language, time series, video, action trajectories in robotics…
- The key to sequence modeling is capturing the sequential context

boring movie and not great	$\overset{?}{=}$	*great movie and not boring*

Speech





Language Model

- A language model (LM) aims at providing a probability distribution over every word, given all the words before it:

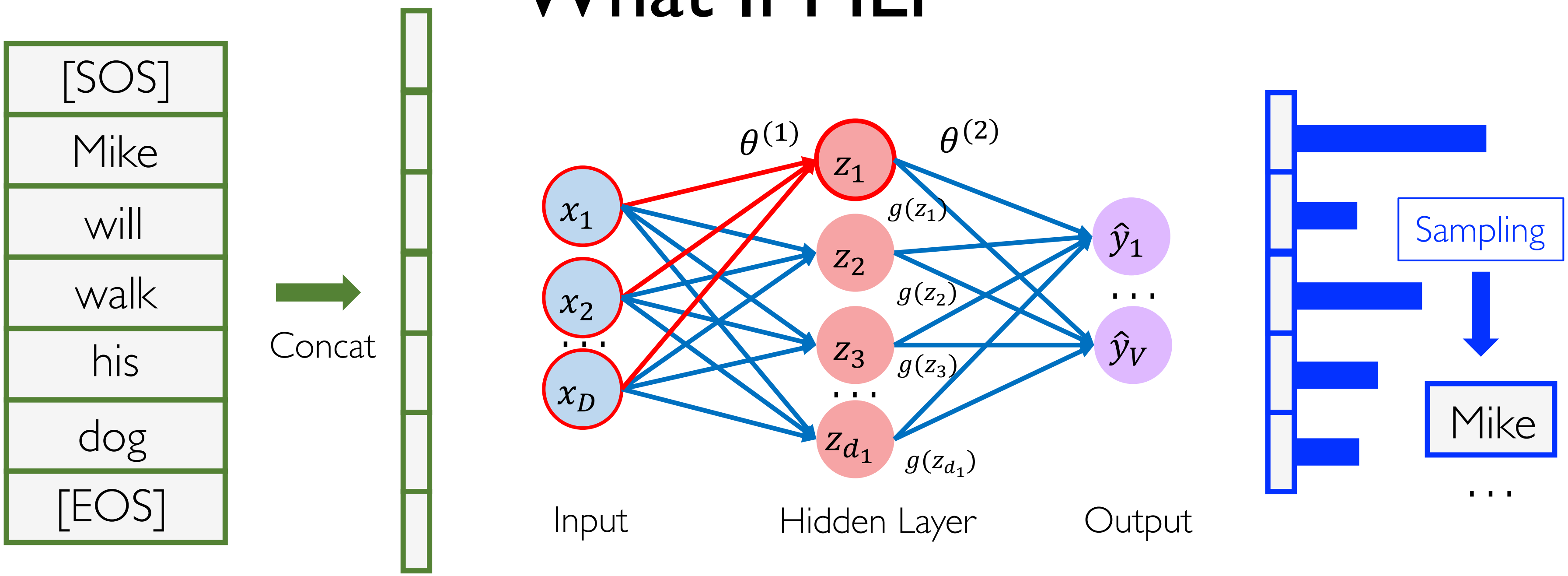
$$P(word_i \mid word_1, \cdots, word_{i-1})$$

- Humans have good sense of what the next word will be:
  - *Deep learning is the study and practice of how we can learn feature representations from large amounts of* $\boxed{}$.

$$P(word_i = \text{“data”} \mid word_1, \cdots, word_{i-1}) = 0.3$$
$$P(word_i = \text{“information”} \mid word_1, \cdots, word_{i-1}) = 0.1$$
$$P(word_i = \text{“hotdogs”} \mid word_1, \cdots, word_{i-1}) = 0.01$$

- How can machines learn useful sequential knowledge from data?

What If MLP



Dimensions
7×1000
Sequence Length
Feature Vector

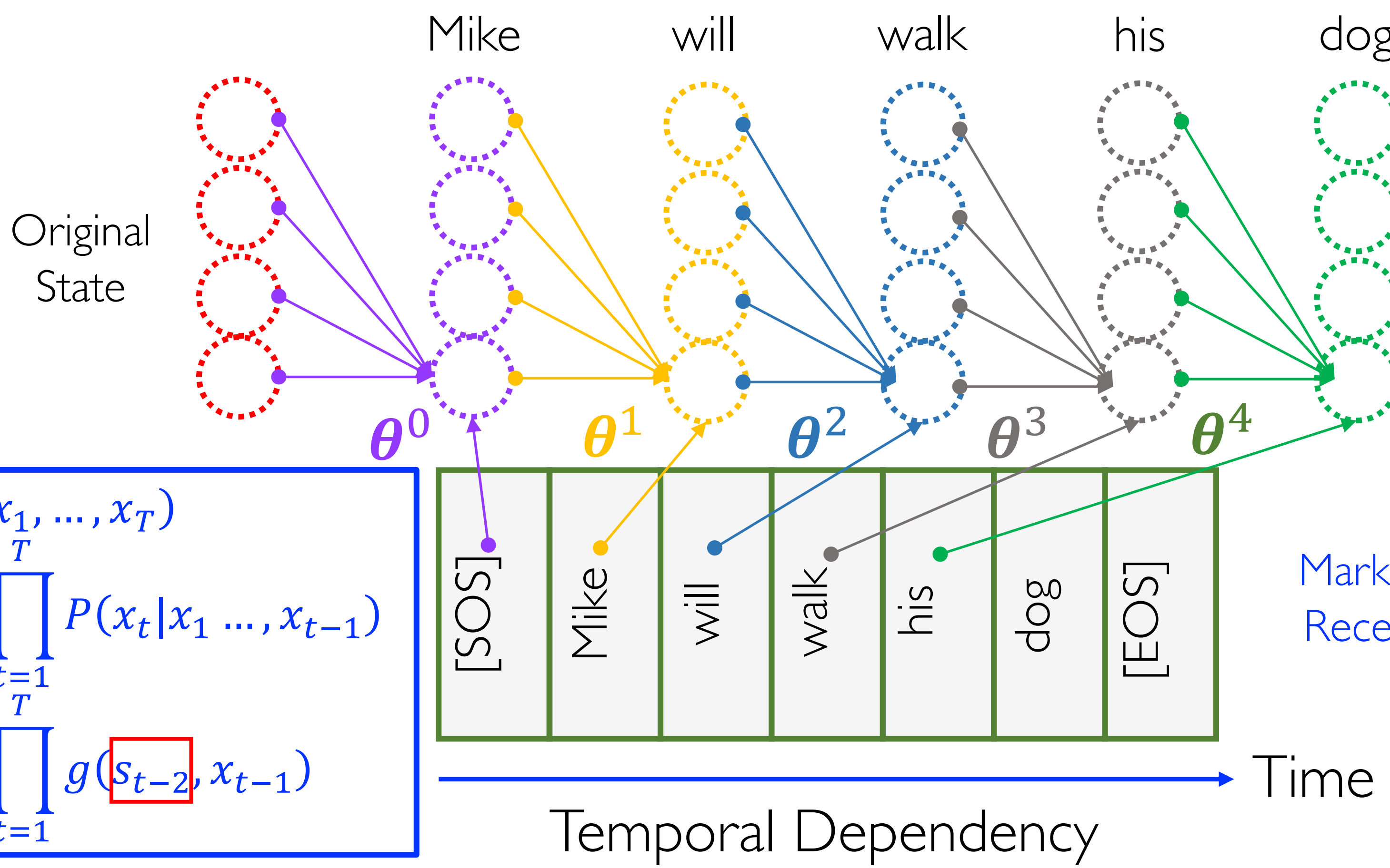
Input
D=7000

k hidden layers
Layer 1: $\alpha \times 7000$
Layer 2: $\beta \times 7000$
Layer 3: $\gamma \times 7000$

Output
V=Vocabulary Size

- Fixed input dimension will limit the length of the input sequence.
- The order of sequence data (temporal dependency) is discarded.

Idea 1: Local Dependency

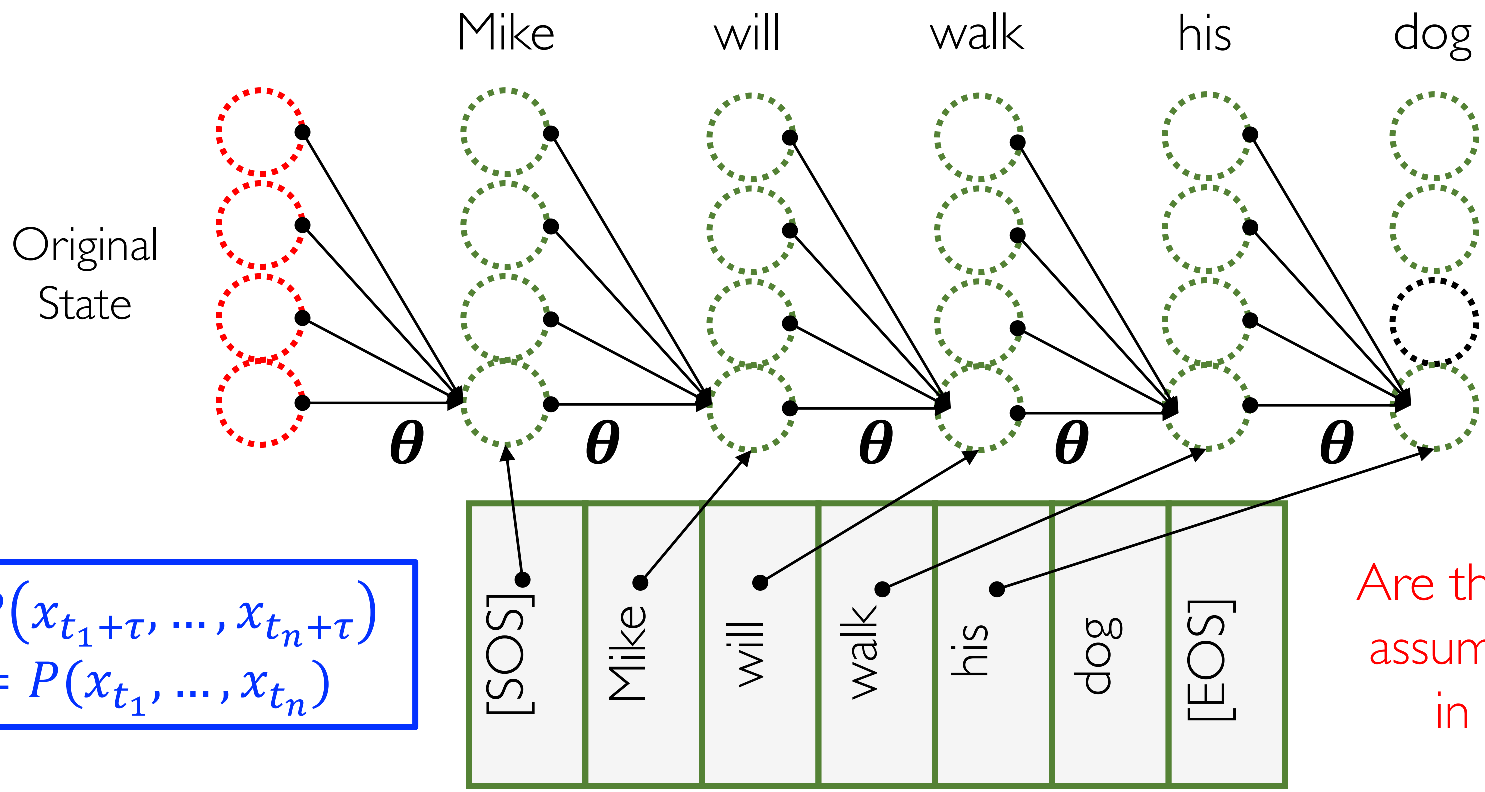


$$
\begin{aligned}
&P(x_1, \dots, x_T) \\
&= \prod_{t=1}^{T} P(x_t | x_1 \dots, x_{t-1}) \\
&= \prod_{t=1}^{T} g(\boxed{s_{t-2}}, x_{t-1})
\end{aligned}
$$

Markov assumption
Receptive field = 2

- **[Local Dependency Assumption]:** The sequential information of all previous timestamps can be encoded into one hidden representation

Idea 2: Parameter Sharing



$$P(x_{t_1+\tau}, \dots, x_{t_n+\tau}) = P(x_{t_1}, \dots, x_{t_n})$$

Are these two key assumptions valid in practice?

- **[Temporal Stationarity Assumption]:** If a feature is useful at time t_1, then it should also be useful for all time stamps t_2.
- Thus can reduce the redundant parameters with parameter sharing.

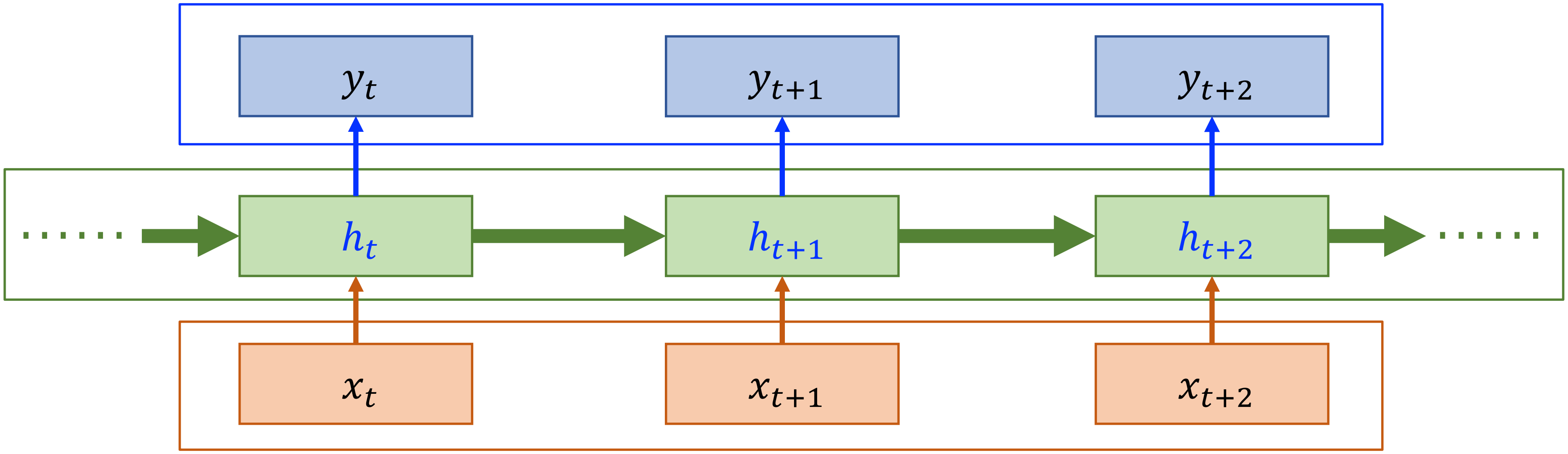
Recurrent Neural Network (RNN)

Prediction | Recognition | …… | Sequence-to-Sequence Prediction

$\{x_t\}$ and $\{y_t\}$ can be different

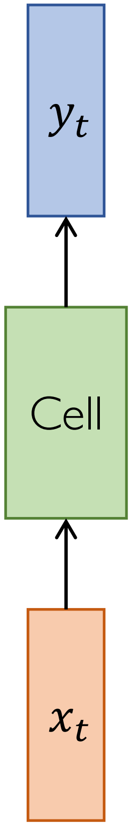
Use different outputs $\{y_t\}|_{t=0}^{T}$ for different tasks

Generate sequential features $\{h_t\}|_{t=0}^{T}$



Process the input sequence $\{x_t\}|_{t=0}^{T}$, where $x_t \in \mathbb{R}^D$ is feature vector

Recurrent Layer



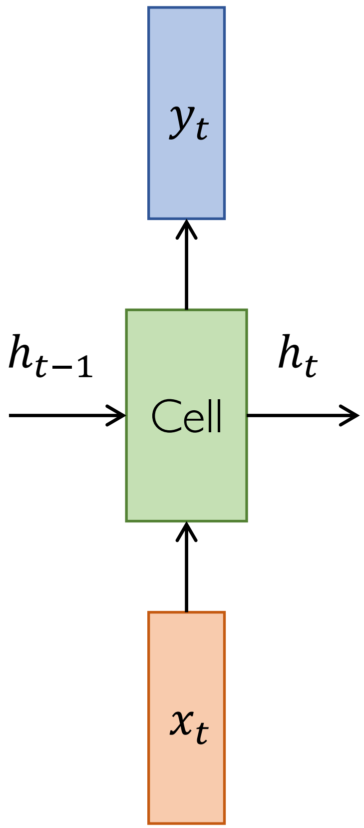
Output

Hidden State

This is a simple MLP.

Input

Recurrent Layer

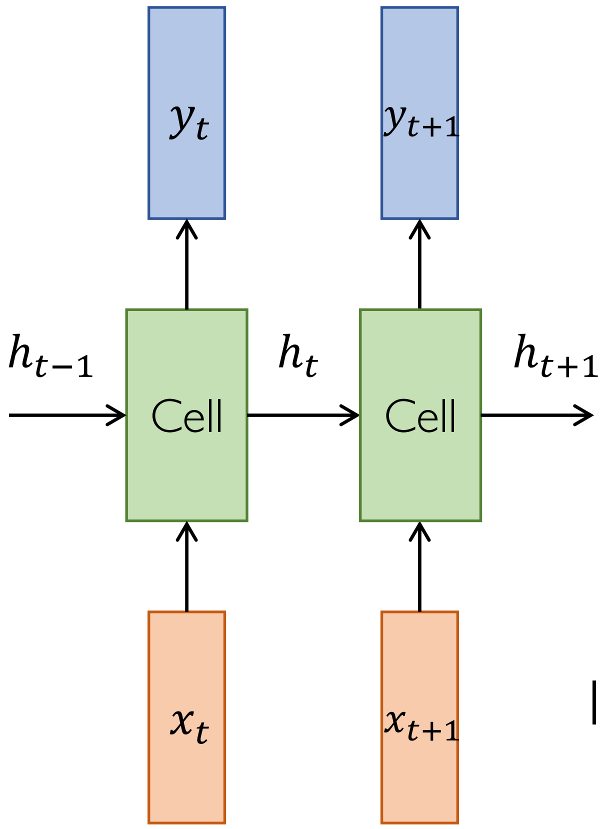


Grows over time…

Hidden State: $\boldsymbol{h_t}$ is used to encode the information from all past timesteps

Implementation of local dependency

Recurrent Layer

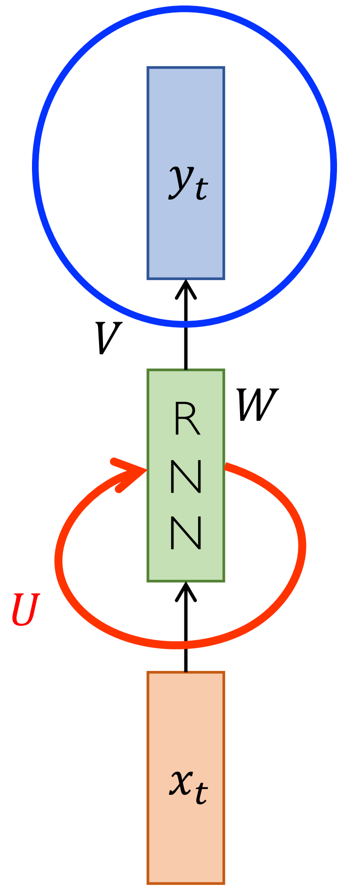


(y_t, y_{t+1}) is a length-2 sequence

Hidden State: $\boldsymbol{h_t}$ is used to encode the information from all past timesteps

Implementation of local dependency

Recurrent Layer



Other activations (ReLU) are applicable

predict a vector at each time step

$$y_t = Vh_t$$

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

input vector at each time step

New state

Old state

some function with parameters W

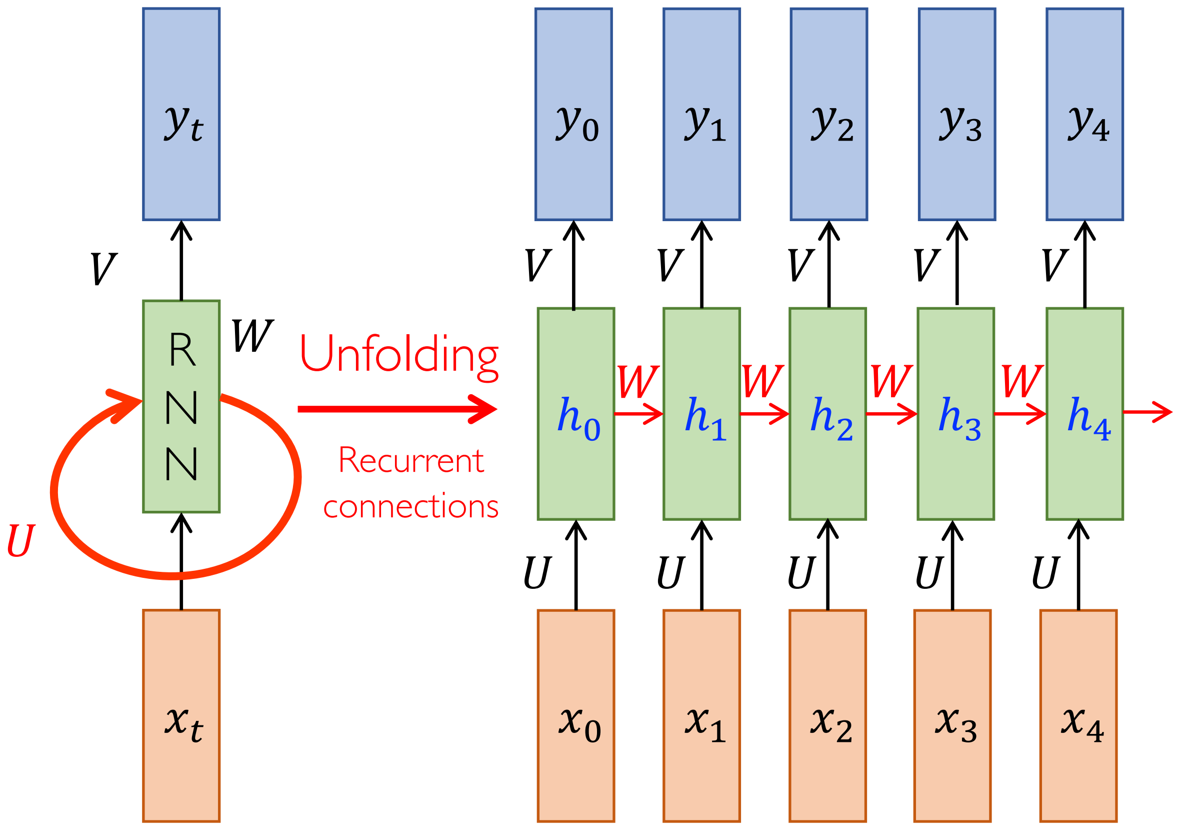
Hidden State: h_t is used to encode the information from all past timesteps

$$h_t = \tanh(\underbrace{Wh_{t-1} + Ux_t}_{})$$

Other activations (ReLU) are applicable

Concat of x_t and h_{t-1}

Unfolding over Time



Note: parameters are not shared over layers

h_t is used to encode the information from all past timesteps

Stationary assumption

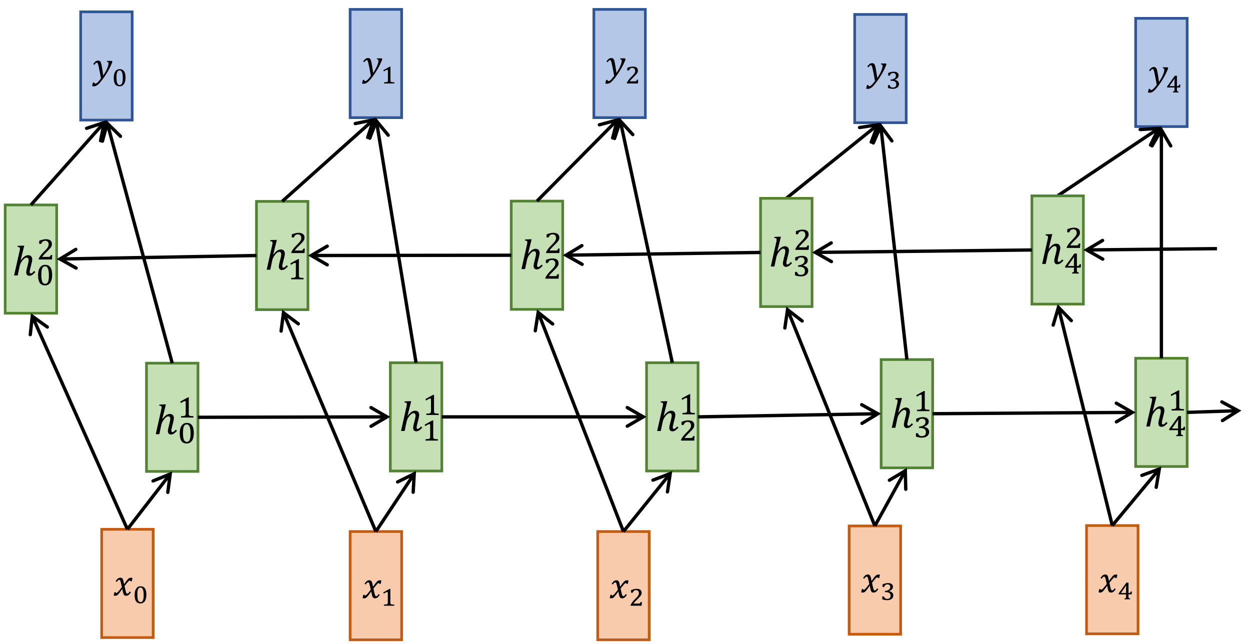
We use temporally shared parameters W, U, V

Bidirectional RNN

- When conditioning on a full input sequence, traversing from both directions can enhance the temporal dependency
- Used in sequence classification, e.g. speech recognition

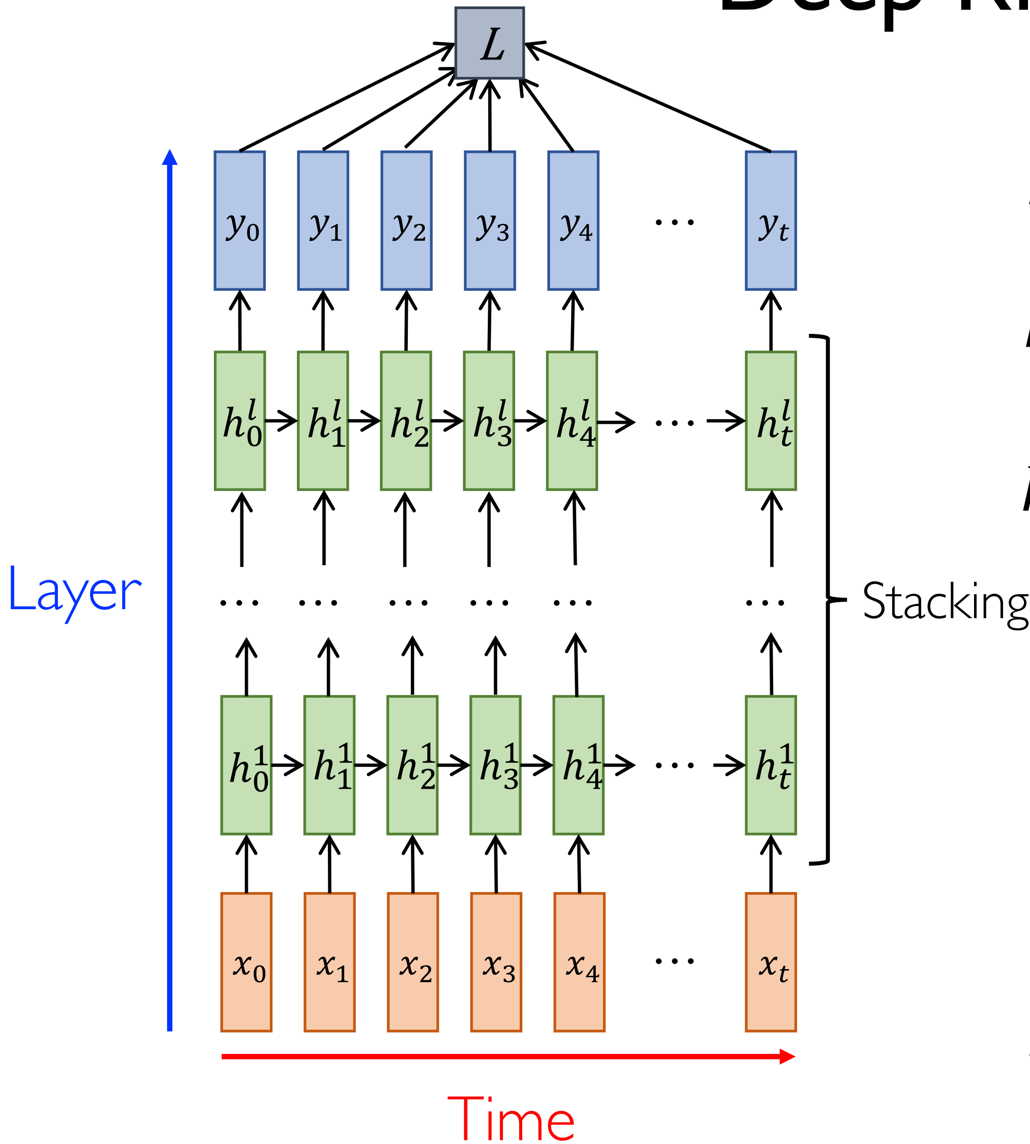
Concat
$[h_t^1, h_t^2]$

Sum
$h_t^1 + h_t^2$



Graves et al. Framewise phoneme classification with bidirectional LSTM. **Neural Networks**, 2005

Deep RNN



$$\hat{y}_t = \mathrm{softmax}(V h_t^l)$$

$$h_t^1 = \tanh(W_1 h_{t-1}^1 + U_1 x_t)$$

$$\cdots\cdots$$

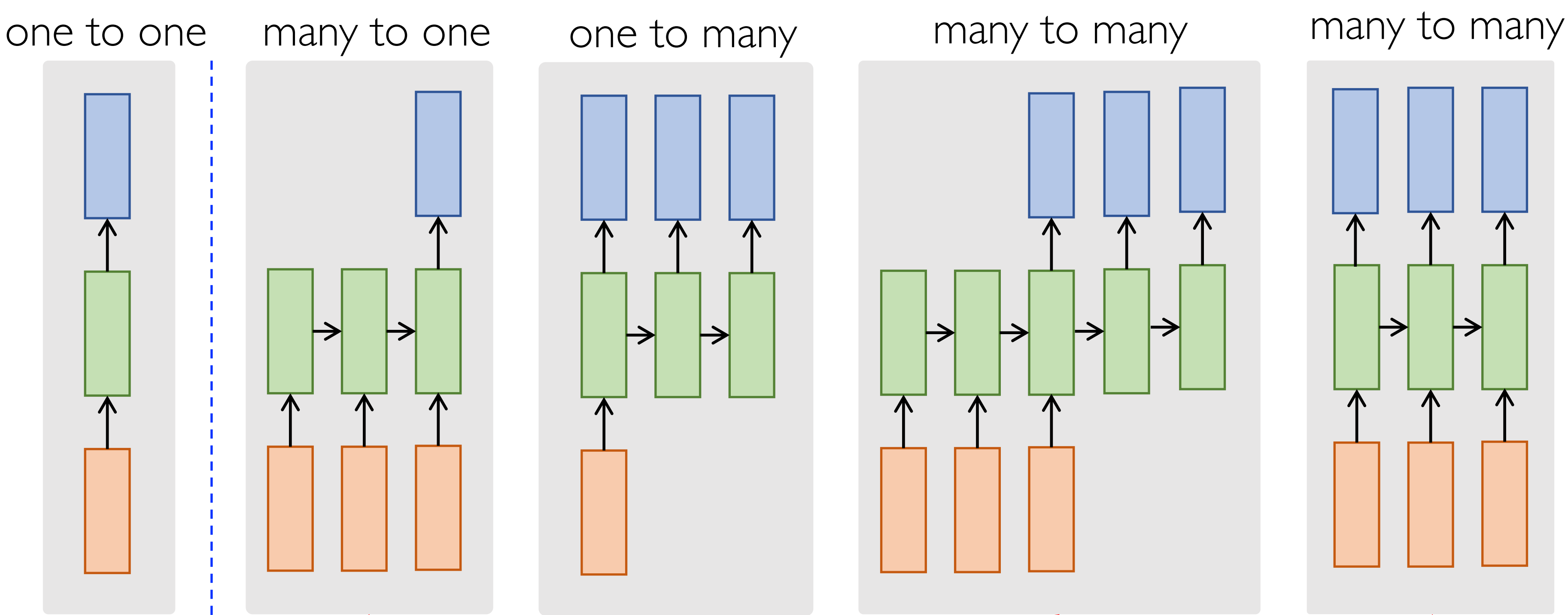
$$h_t^l = \tanh(W_l h_{t-1}^l + U_l h_t^{l-1})$$

Temporally shared parameters W, U, V (no superscript t)

Loss function (cross-entropy):

$$L = -\frac{1}{T} \sum_{t=1}^{T} \sum_{c=1}^{C} y_{t,c} \log(\hat{y}_{t,c})$$

Architectures



MLP

1

Sentiment Classification
Sentence → Sentiment

Image Captioning
Image → Sentence

2

3

Machine Translation
Sentence (en) → Sentence (zh)
(heterogeneous)

Language Model
Sentence → Sentence
(homogeneous)

0

Sequence to Sequence

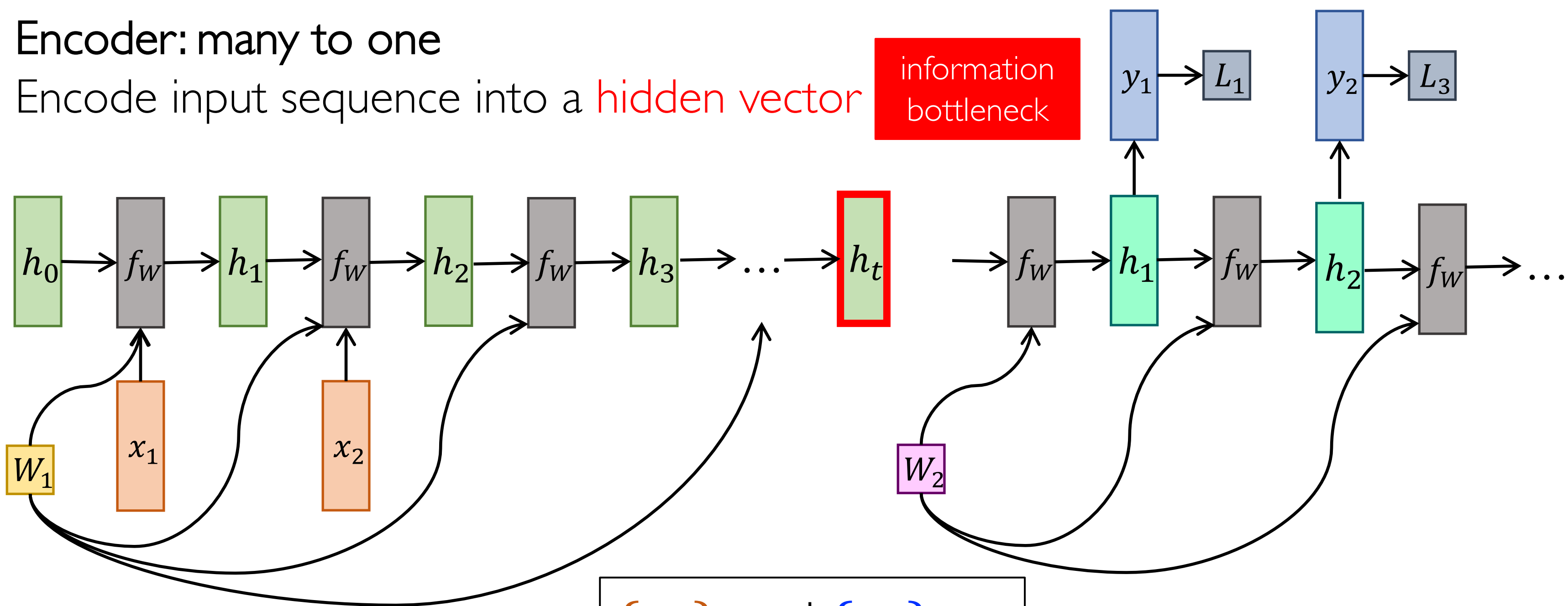
Many-to-One + One-to-Many

Decoder: one to many

Produce output sequence from the hidden vector

Encoder: many to one

Encode input sequence into a hidden vector



$\{x_t\}$ and $\{y_t\}$ can be different

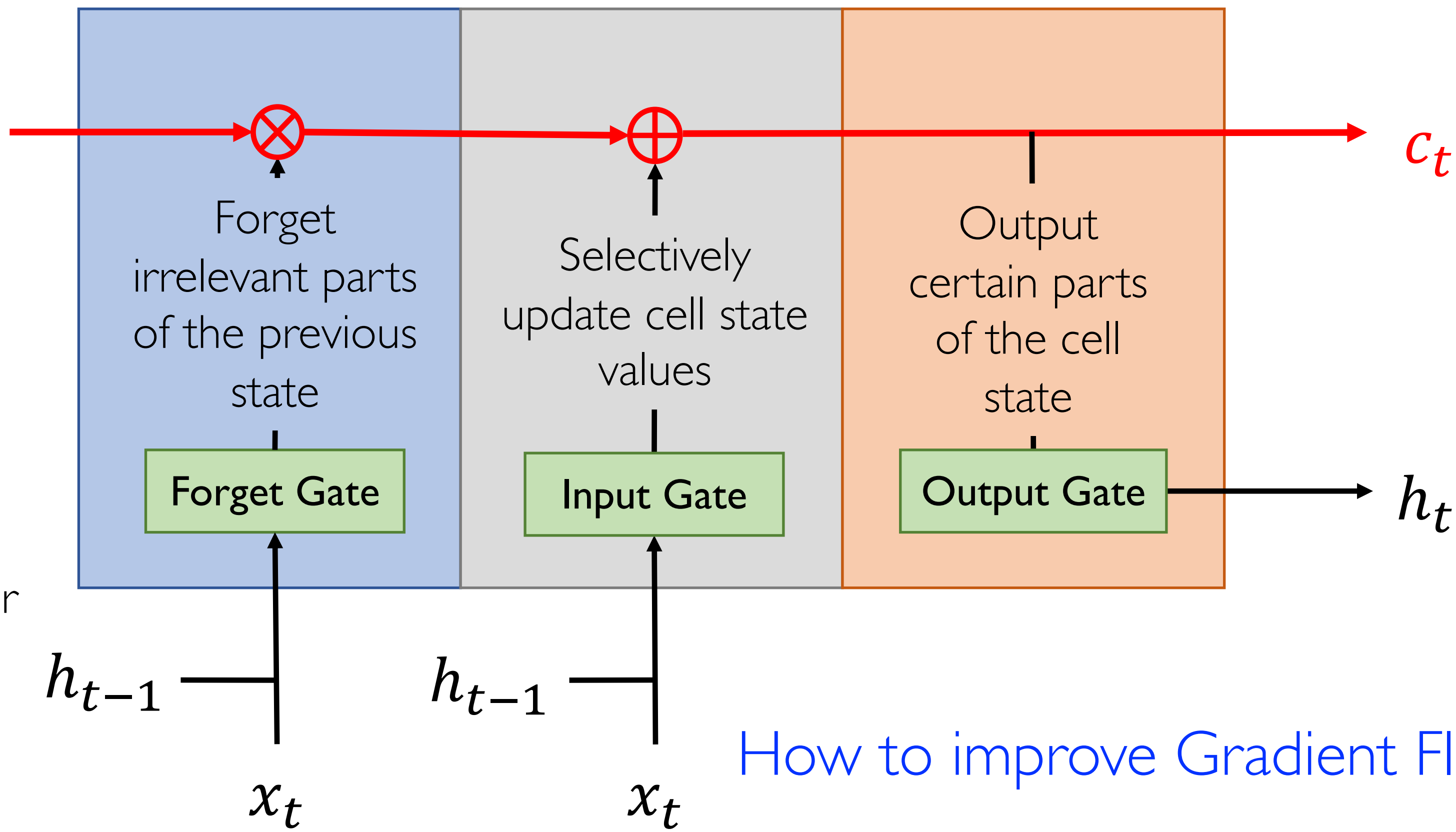
Outline

- Recurrent Neural Network
- **Long Short-Term Memory (LSTM)**
- Transformers: Attention is All You Need

Long Short-Term Memory (LSTM)

Difficulty of learning is caused by interrupted gradient flow in RNNs

J. Schmidhuber

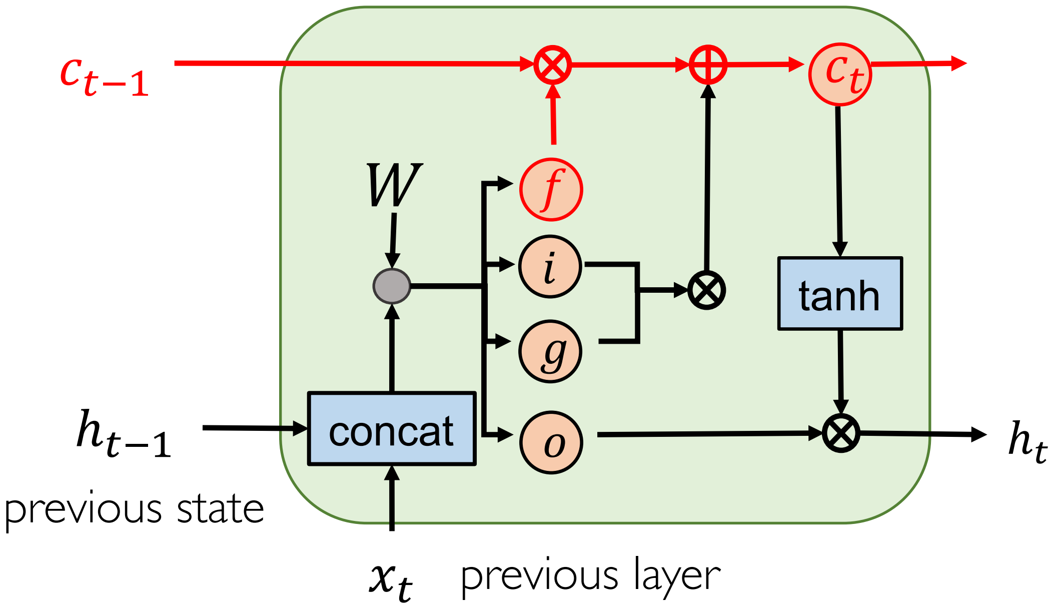


How to improve Gradient Flow?

Hochreiter and J. Schmidhuber. Long short-term memory, 1995

Long Short-Term Memory (LSTM)

Gradient Flow Highway



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \begin{bmatrix} W_i \\ W_f \\ W_o \\ W_g \end{bmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

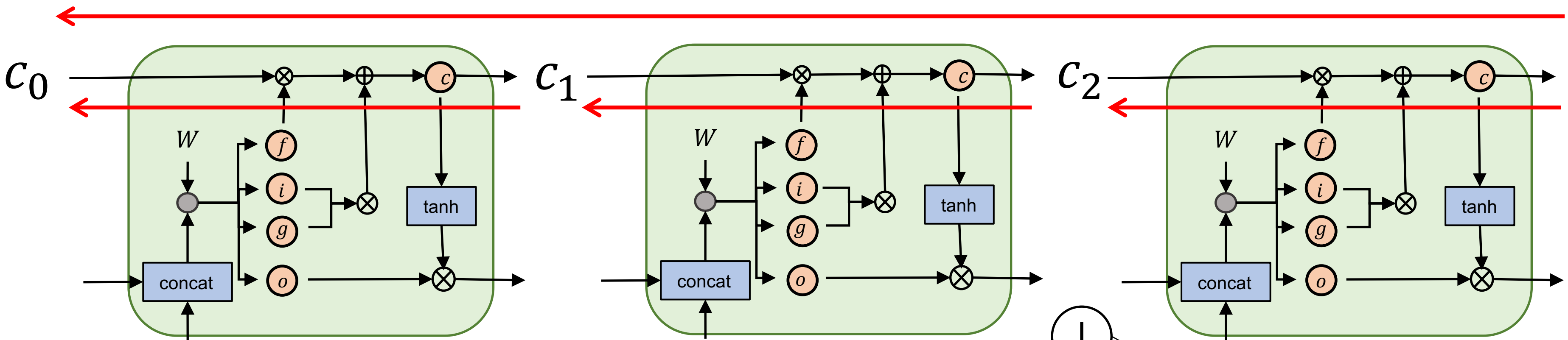
f: **Forget gate**, Whether to erase cell
i: **Input gate**, whether to write to cell
g: **Gate gate**, How much to write to cell
o: **Output gate**, How much to reveal cell

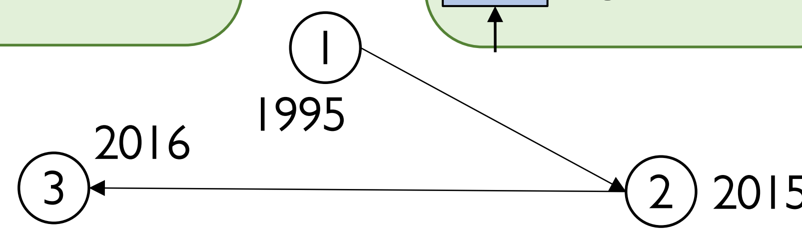
Hochreiter and J. Schmidhuber. Long short-term memory, 1995

LSTM: Gradient Flow

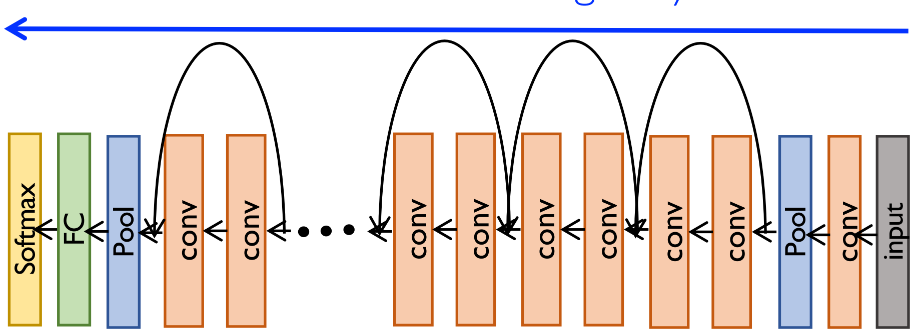
Gradient Flow Highway

Backpropagate from c_t to c_{t-1} via only element-wise multiplication, no matrix multiply.





ResNet motivated from Highway Network



Highway Network motivated from LSTM.

$$g = T(x, W_T)$$

$$y = g \odot H(x, W_h) + (1 - g) \odot x$$

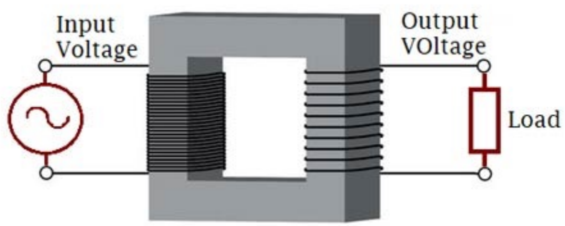
Schmidhuber et al. Highway Networks. NIPS 2015

Outline

- Recurrent Neural Network
- Long Short-Term Memory (LSTM)
- **Transformers: Attention is All You Need**

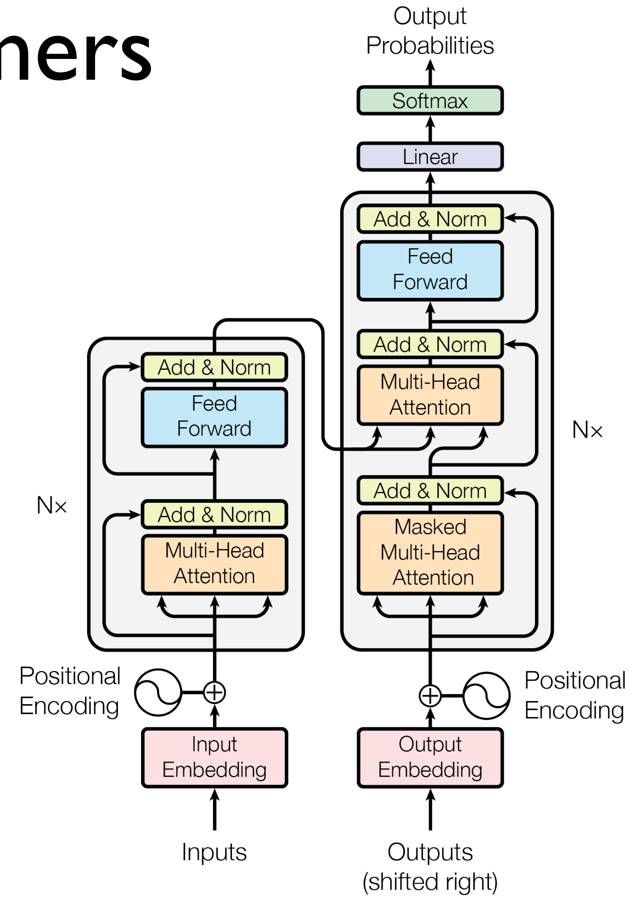
Transformers

变形金刚



Transformer

变压器



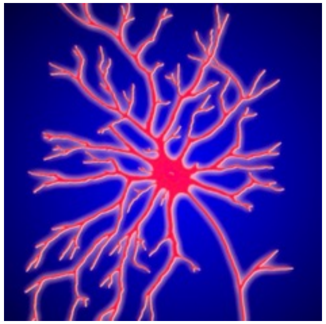
变换器

A neural network

Based on the attention mechanism.

Vaswani et al. Attention is all you need. NIPS 2017.

Human Attention



- Attention is your **brain function** that allocates cognitive processing resources to <u>focus on information or stimuli</u>.
- **Sustained Attention**
  - Focus on one specific task for a continuous amount of time.
- **Selective Attention**
  - Select from many factors or stimuli and to focus on only the one that you want while filtering out other distractions.
- **Alternating Attention**
  - Switch your focus back and forth between tasks.
- **Divided Attention**
  - Process more responses to different demands simultaneously.

Petersen et al. The Attention System of the Human Brain: 20 Years After. 2012.

Attention in Deep Learning

- Generally referred to as the attention mechanism
  - Introduced to machine learning by Y. Bengio in 2014
  - Incorporates the notion of relevance by allowing the model to dynamically pay attention to only certain parts of the input that help in performing the task at hand effectively.

pork belly = delicious . || scallops? || I don't even like scallops, and these were a-m-a-z-i-n-g . || fun and tasty cocktails. || next time I in Phoenix, I will go back here. || Highly recommend.

Temporal Attention

Spatial Attention

Bahdanau et al. Neural Machine Translation by Jointly Learning to Align and Translate. ICLR 2015.

Attention

- Compute importance:

$$\alpha_{ij} \propto \exp\left(a(s_{i-1}, h_j)\right)$$

- Aggregate by importance:

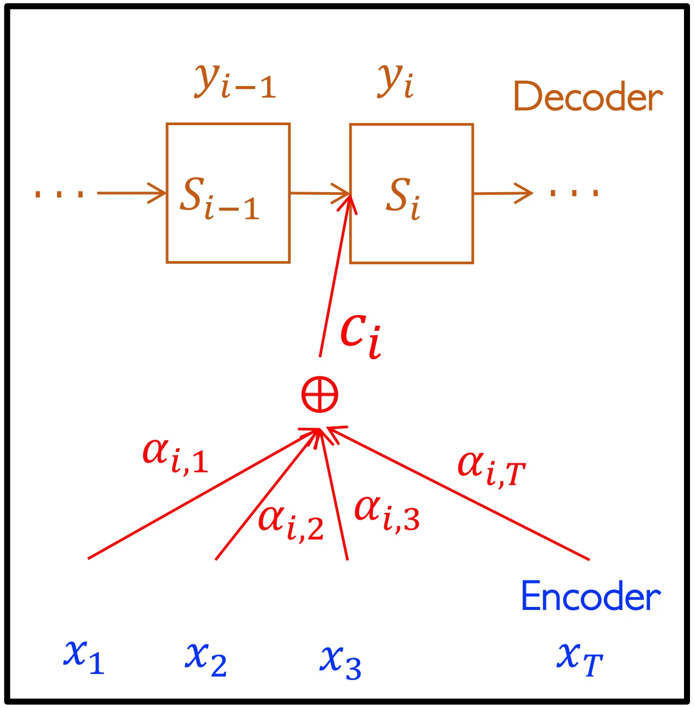
$$c_i = \sum_{j=1}^{T} \alpha_{ij} h_j$$

- Represent elements in Attention by:

- Key k
- Query q
- Value v

$$w_{ij} = a(q_{i-1}, k_j)$$

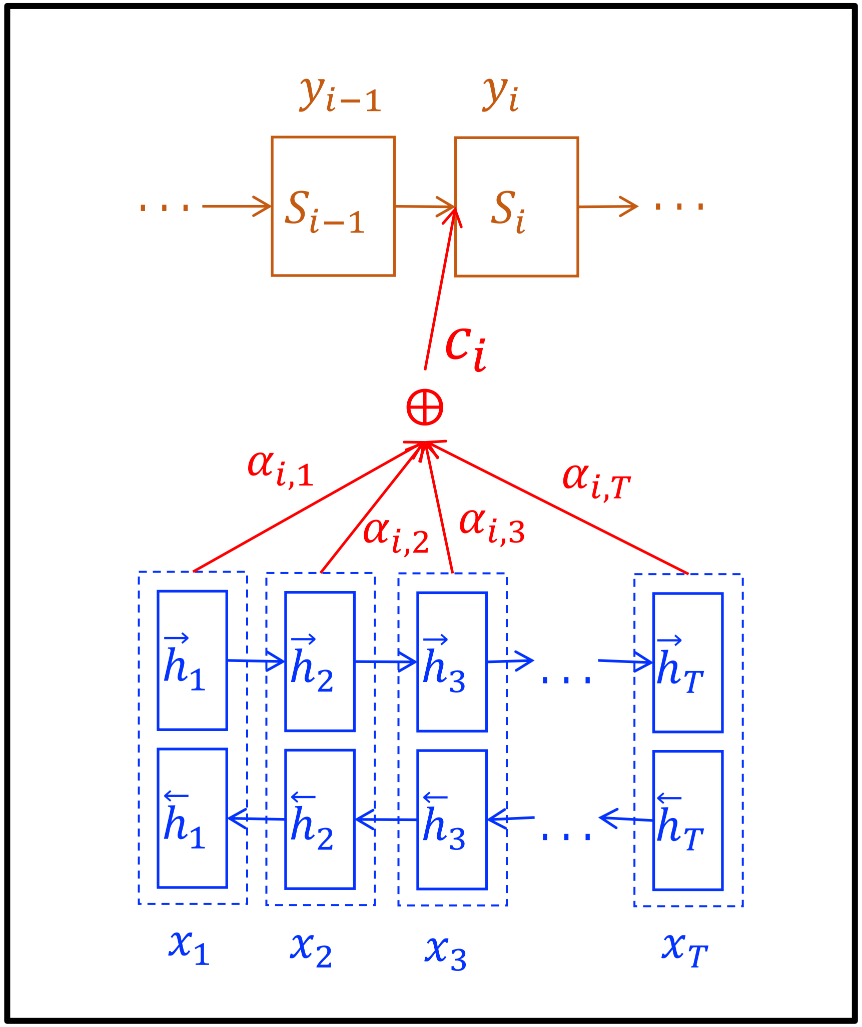
$$c_i = \sum_{j=1}^{T} w_{ij} v_j$$



- Importance is computed by a parameterized model on q and k.

RNN with Attention

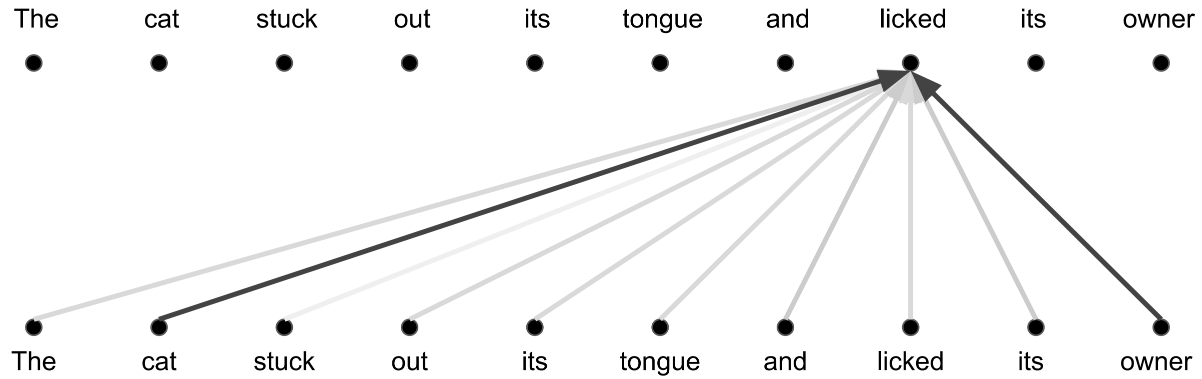
- RNN with Attention works well for long sequences.
- But it is wasteful.
  - Complex model.
  - Too many RNNs.
    - » Large complexity and memory requirement.
  - Sequential computation inhibits parallelization.
- Can we design a simpler model?



Self-Attention

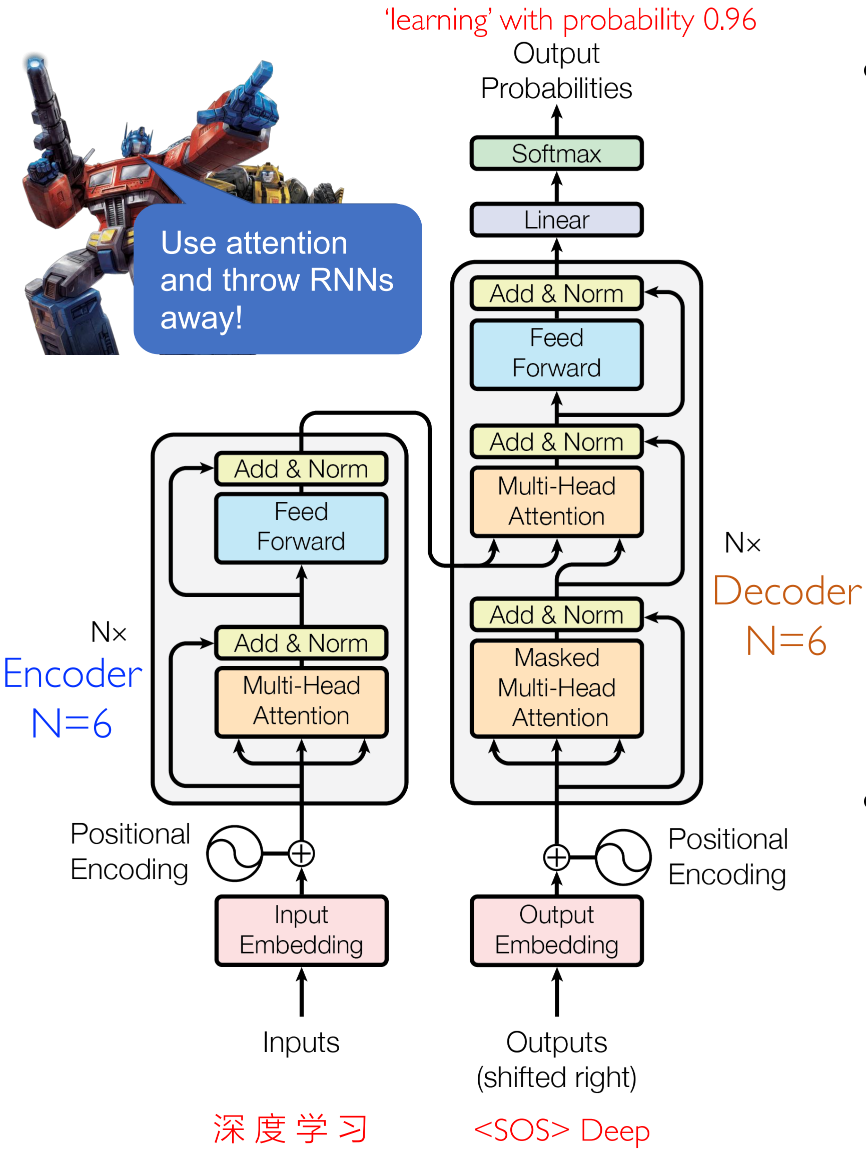
- In RNN, we need attention modules.
- But with attention, we even don't need RNN!

- Self-Attention:
  - Query $Q = [q_1, \dots, q_n]$, Key $K = [k_1, \dots, k_n]$, Value $V = [v_1, \dots, v_k]$:
  - Q, K, V are from the same sequence 😄
    - » Compute weights with each others in the whole sequence.



Vaswani et al. Attention is all you need. NIPS 2017.

Transformer



- Main components:
  1. Scaled Dot-Product Attention
  2. (Masked) Multi-Head Attention
  3. Position-wise FFN
  4. Residual Connections
  5. Layer Normalization
  6. Positional encoding
- Encoder-decoder architecture:
  1. Encoder
  2. Decoder with Masking
  3. Encoder-Decoder Attention

Scaled Dot-Product Attention

- Recall relevance $e_{ij} = v_a^{\mathrm{T}} \tanh\left(W_a s_{i-1} + U_a h_j\right)$: too complex!
- Given query q (to match others) and key k (to be matched):
  - Bilinear

$$a(q,k) = q^T W k \longrightarrow \text{Parameter-heavy}$$

  - Dot Product

$$a(q,k) = q^T k \longrightarrow \text{Dimension-sensitive}$$

  - Scaled Dot-Product

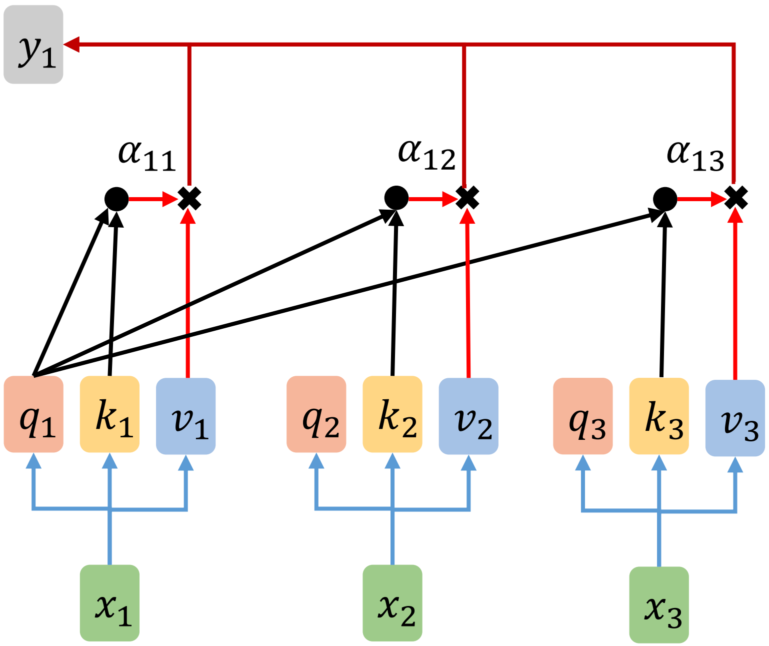
$$a(q,k) = q^T k / \sqrt{d_k} \longrightarrow \text{Dimension of } q \text{ and } k$$

  - For independent q_i, k_i with mean 0 and variance 1:
  - Dot-product $q^T k = \sum_{i=1}^{d_k} q_i k_i$ has mean 0 and variance d_k.
    - » Will not cause large values going into the saturation of Softmax

① Scaled Dot-Product Attention

$$y_1 = \sum_i \alpha_{1i} v_i \qquad \alpha_{1i} = \frac{\exp(q_1 k_i)}{\sum_j \exp(q_1 k_j)}$$

$$\text{Attention}(Q,K,V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

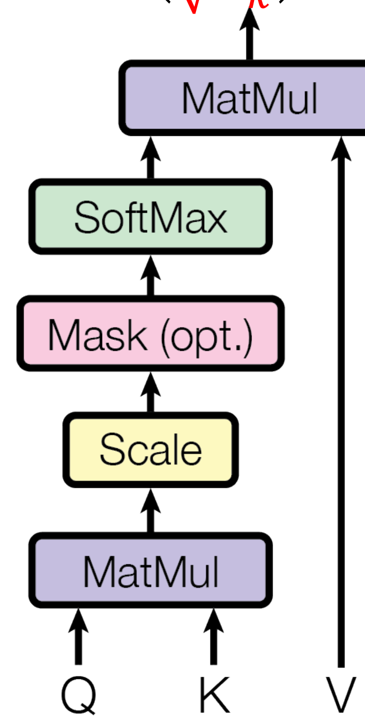


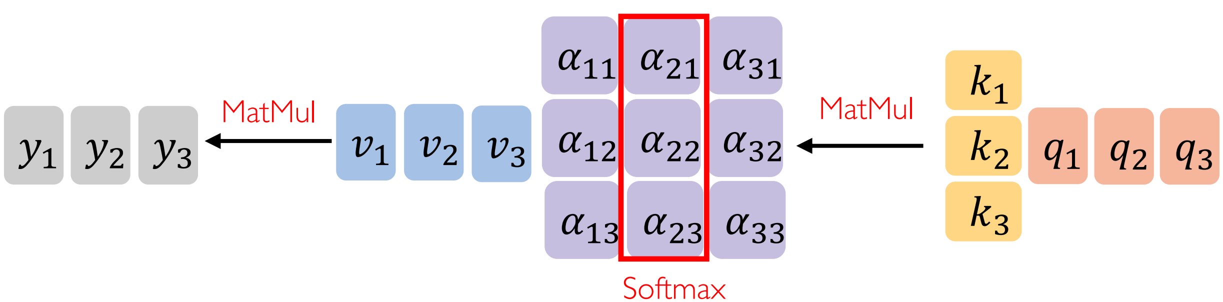
$$\underbrace{q_1\ q_2\ q_3}_{Q} = W^q\ x_1\ x_2\ x_3$$

$$\underbrace{k_1\ k_2\ k_3}_{K} = W^k\ x_1\ x_2\ x_3$$

$$\underbrace{v_1\ v_2\ v_3}_{V} = W^v\ x_1\ x_2\ x_3$$

$$q_i = W^q x_i$$
$$k_i = W^k x_i$$
$$v_i = W^v x_i$$



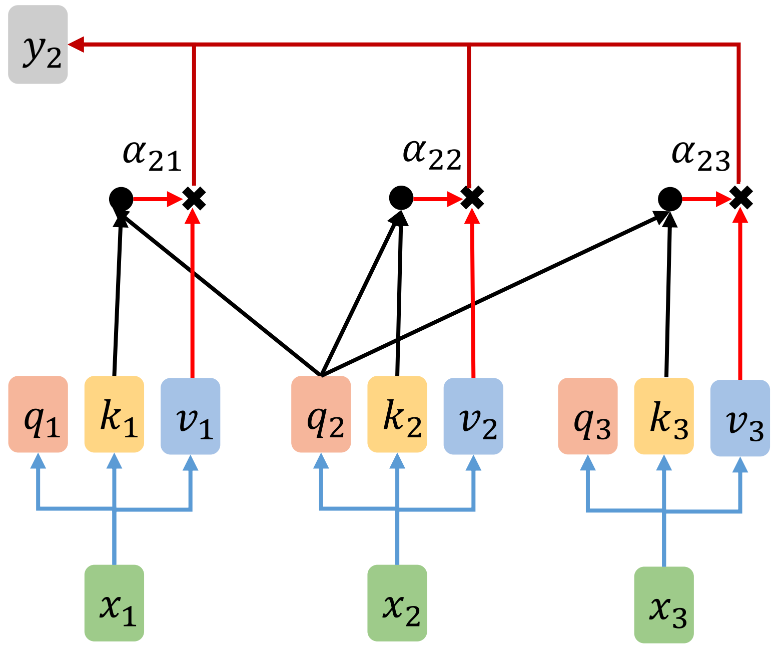


Ignore $\sqrt{d_k}$ for simplicity

① Scaled Dot-Product Attention

$$y_1 = \sum_i \alpha_{1i} v_i \qquad \alpha_{1i} = \frac{\exp(q_1 k_i)}{\sum_j \exp(q_1 k_j)}$$

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V$$

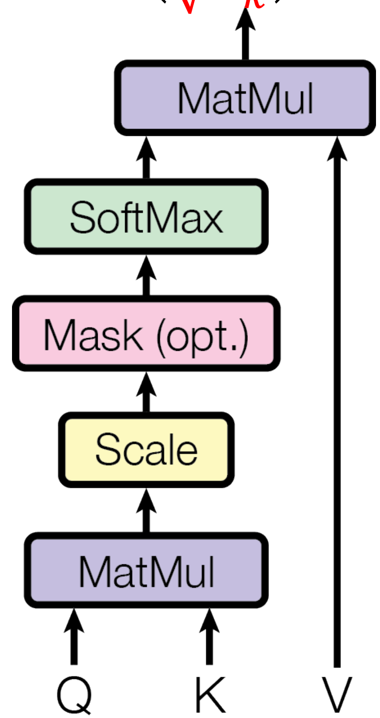


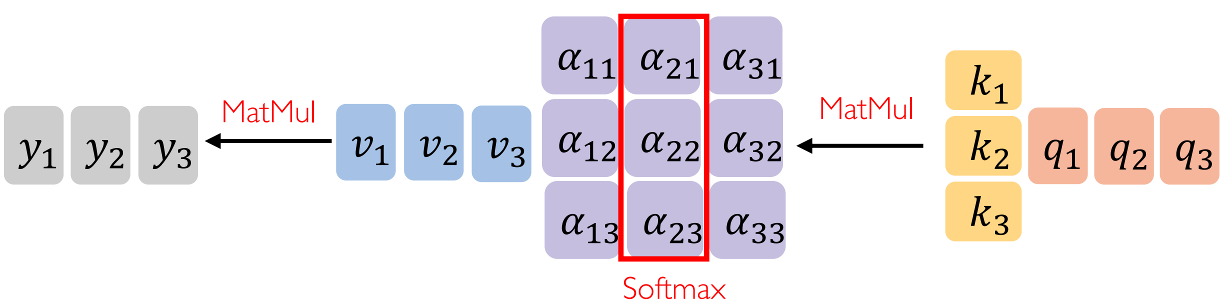
$$\underbrace{q_1\ q_2\ q_3}_{Q} = W^q\ x_1\ x_2\ x_3$$

$$\underbrace{k_1\ k_2\ k_3}_{K} = W^k\ x_1\ x_2\ x_3$$

$$\underbrace{v_1\ v_2\ v_3}_{V} = W^v\ x_1\ x_2\ x_3$$

$$q_i = W^q x_i$$
$$k_i = W^k x_i$$
$$v_i = W^v x_i$$



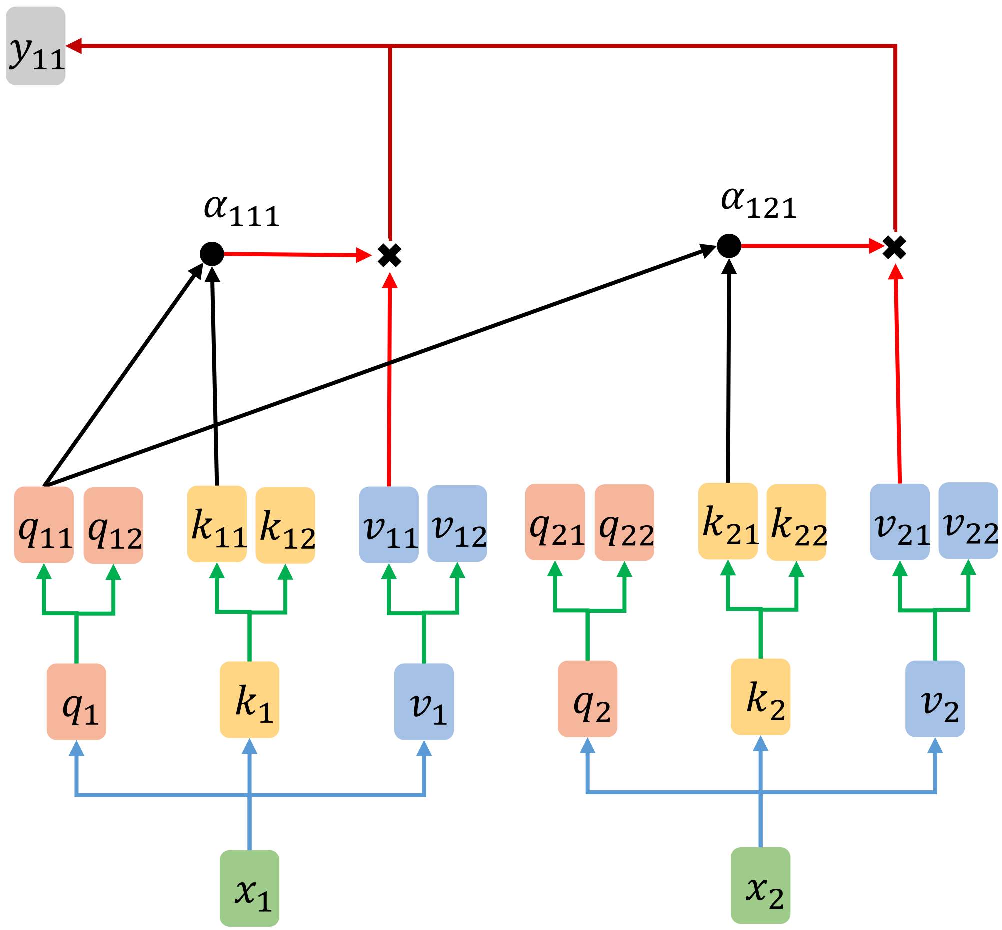


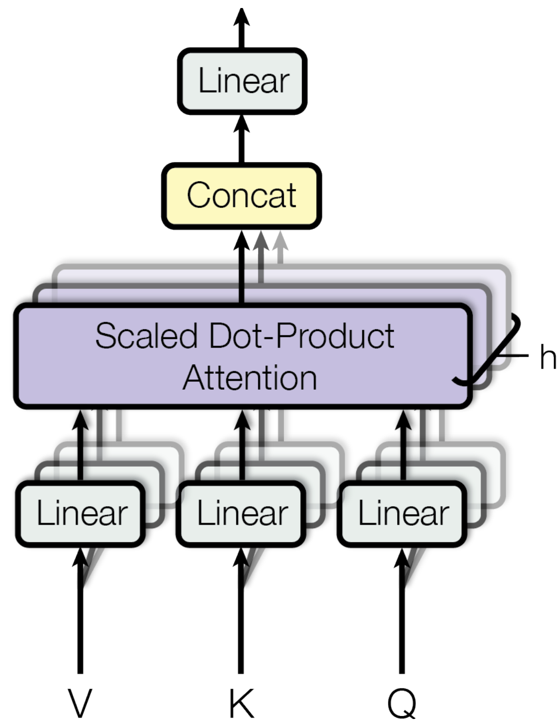
Ignore $\sqrt{d_k}$ for simplicity

② Multi-Head Attention

$$\mathrm{MultiHead}(Q,K,V) = \mathrm{Concat}(\mathrm{head}_1,\ldots,\mathrm{head}_h)W^O$$

$$\mathrm{head}_i = \mathrm{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right)$$





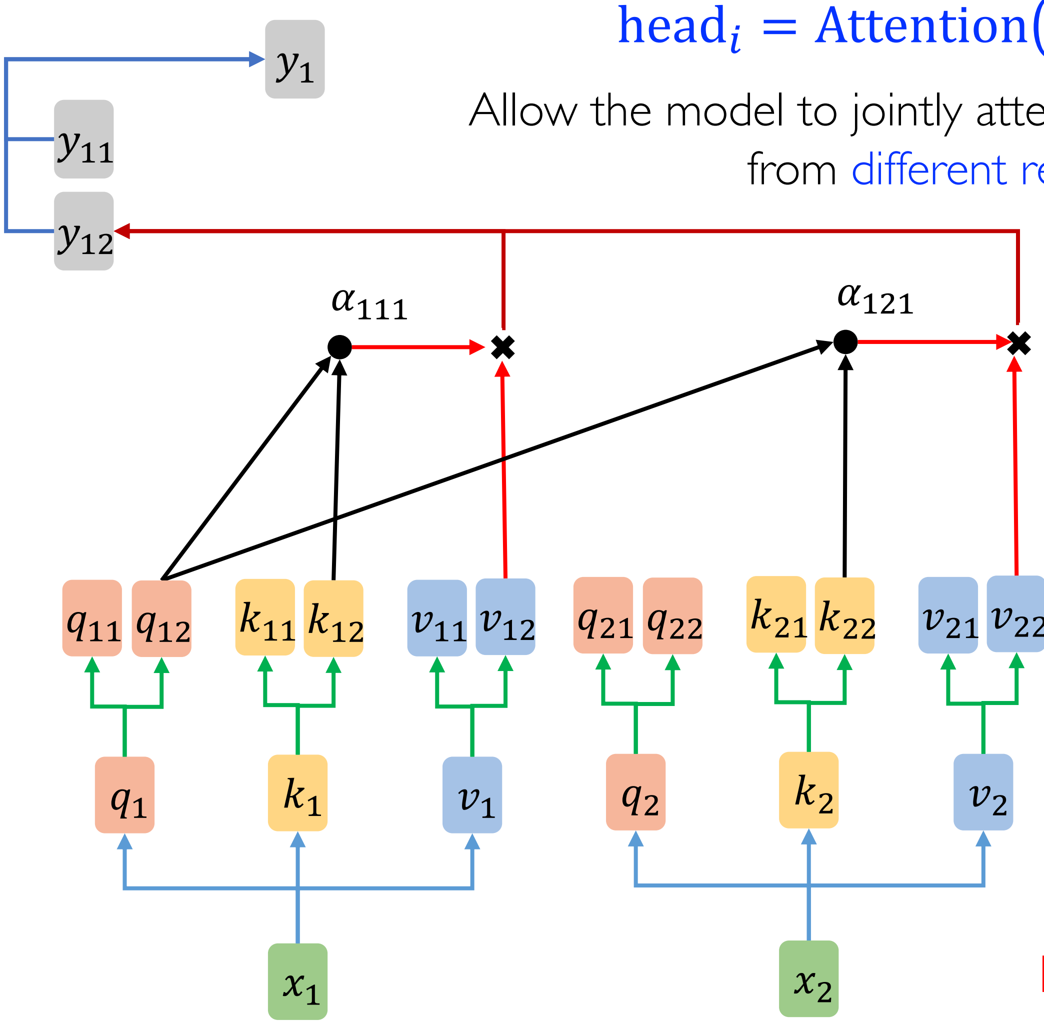
Increase expressiveness

② Multi-Head Attention

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

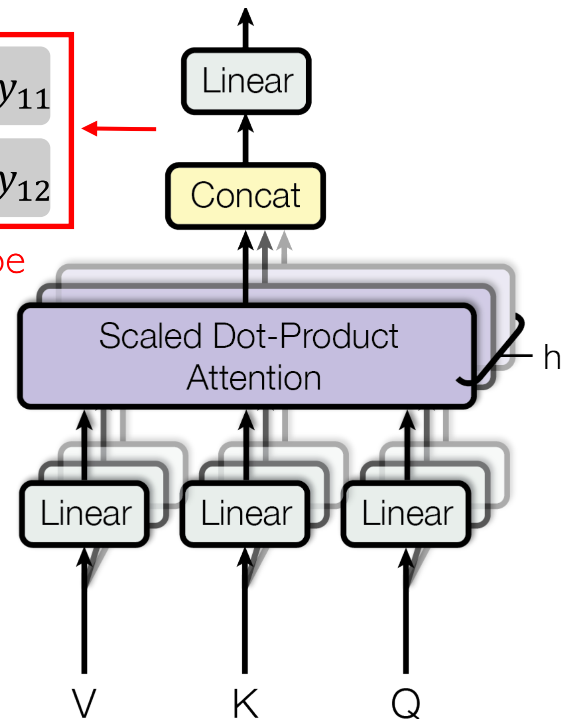
$$\text{head}_i = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right)$$

Allow the model to jointly attend to information at different positions from different representation subspaces.



$$\boxed{y_1 = W^0 \begin{bmatrix} y_{11} \\ y_{12} \end{bmatrix}}$$

Maintain shape



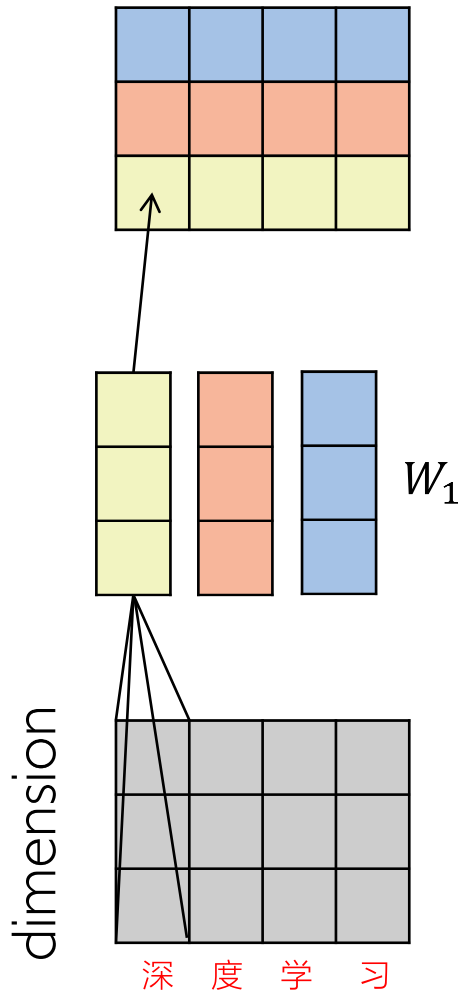
Highly parallelizable, but high memory

③ Position-wise FFN

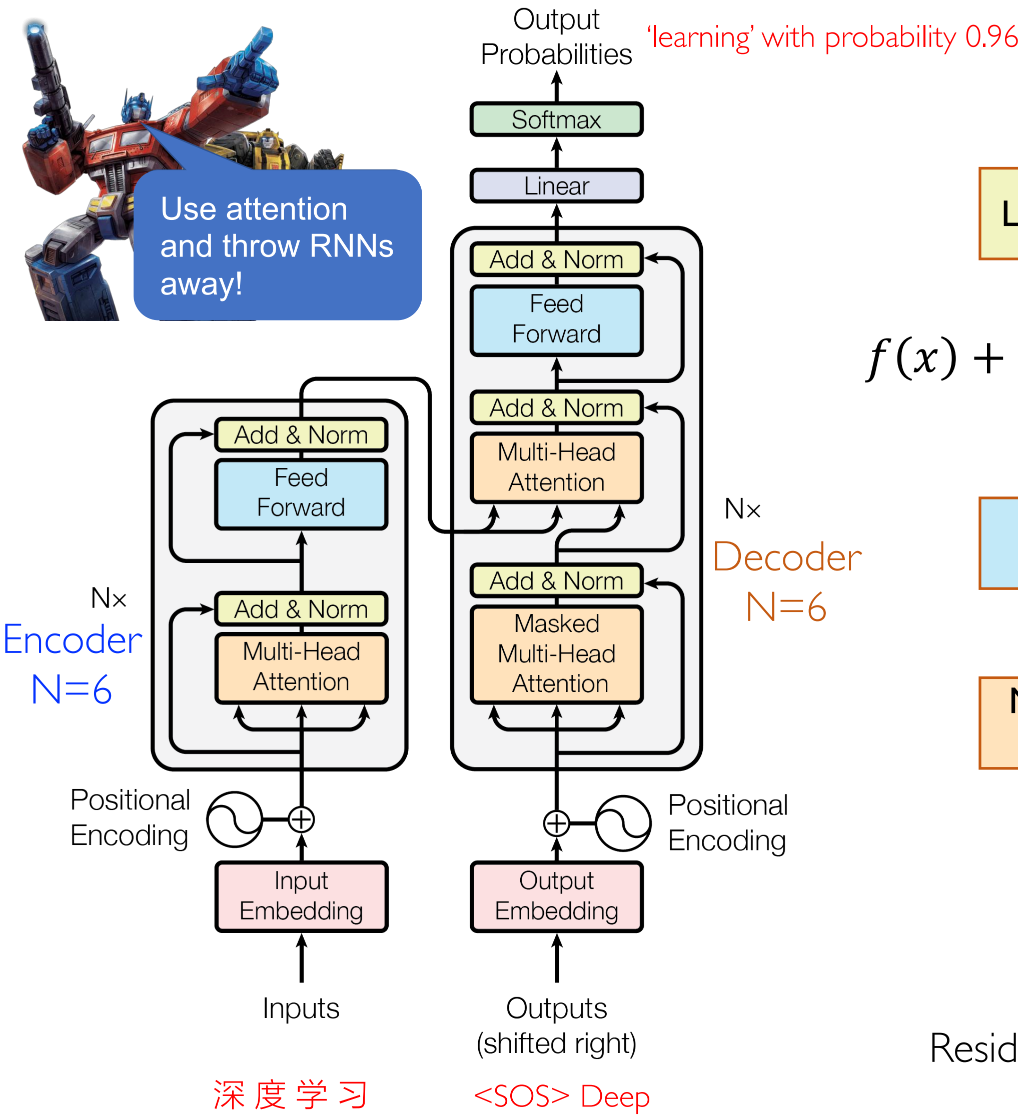
- Apply a two-layer position-wise MLP
  - Consisting of two linear transformations with a ReLU activation in between

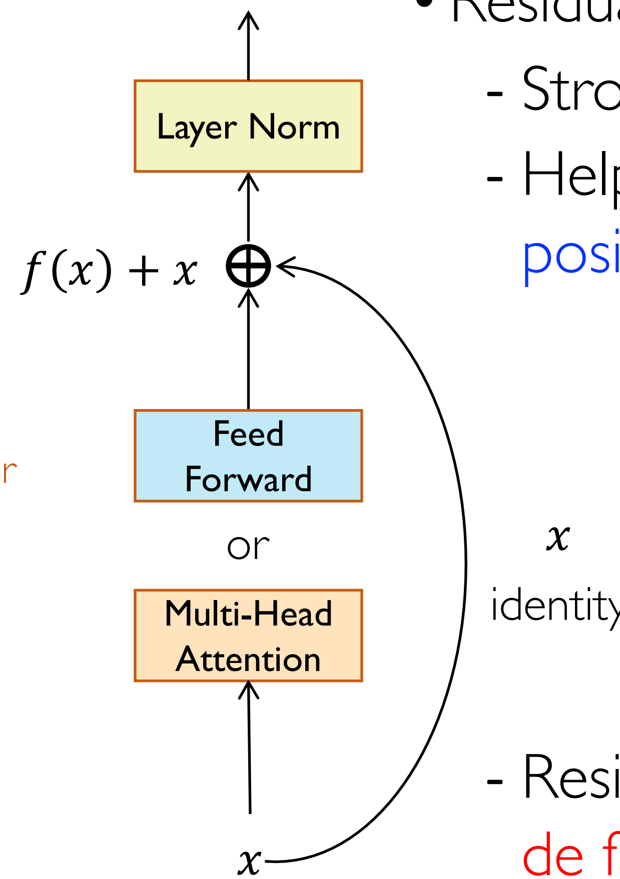
    $$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

  - Huge number of parameters.
- Equivalent to apply **1D** convolution layers
  - Recall 1×1 convolution
  - Parameter are sharing among all positions
  - Applied to sequences with arbitrary length



④ Residual Connection

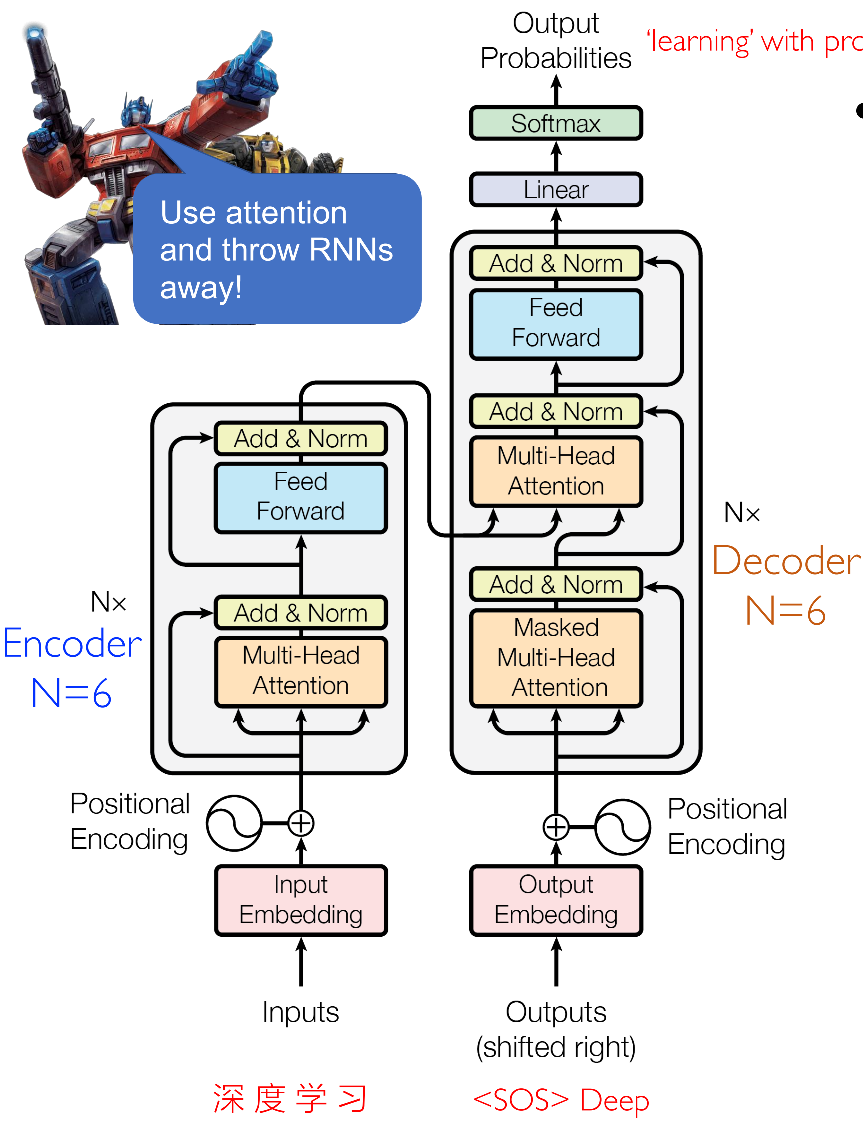




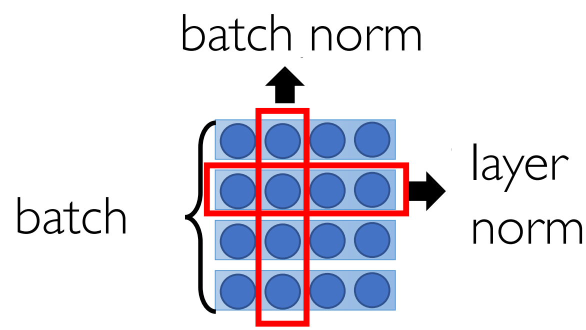
Residual Connections

- Residual connection:
  - Stronger gradients
  - Help remember positional encoding
  - Residual has been de facto design everywhere!

⑤ Layer Normalization



- Layer normalization:
  - Center embeddings around origin, which helps attention layers
  - No train-inference mismatch: Same computation throughout

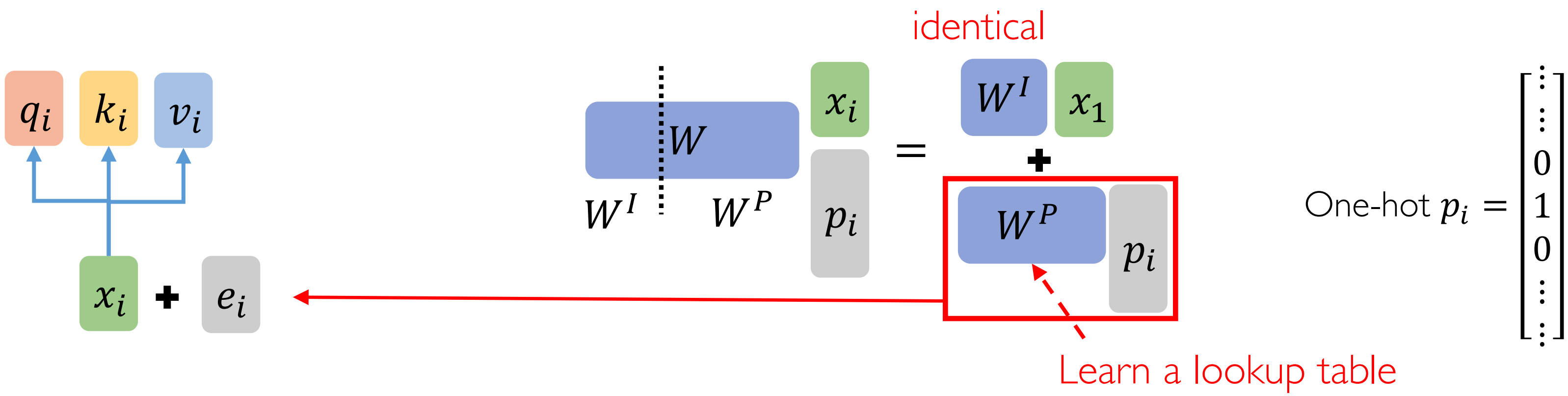


  - DeepNet: Scaling Transformers to 1000 Layers. arXiv 2022. 

⑥ Positional Encoding

- No position information in self-attention 😔

Inject some information about the relative or absolute position of the tokens in the sequence

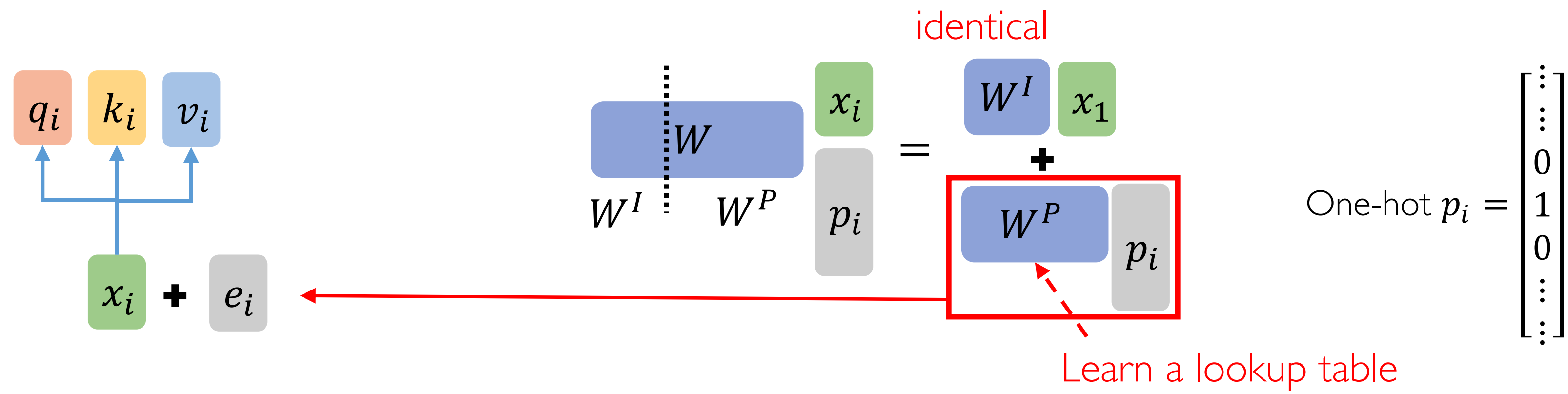


- Should output a unique and deterministic encoding for each position.
- Distance between any two positions should be consistent across sequences.
- Should generalize to longer sentences easily with bounded values.

⑥ Positional Encoding

- No position information in self-attention 😔

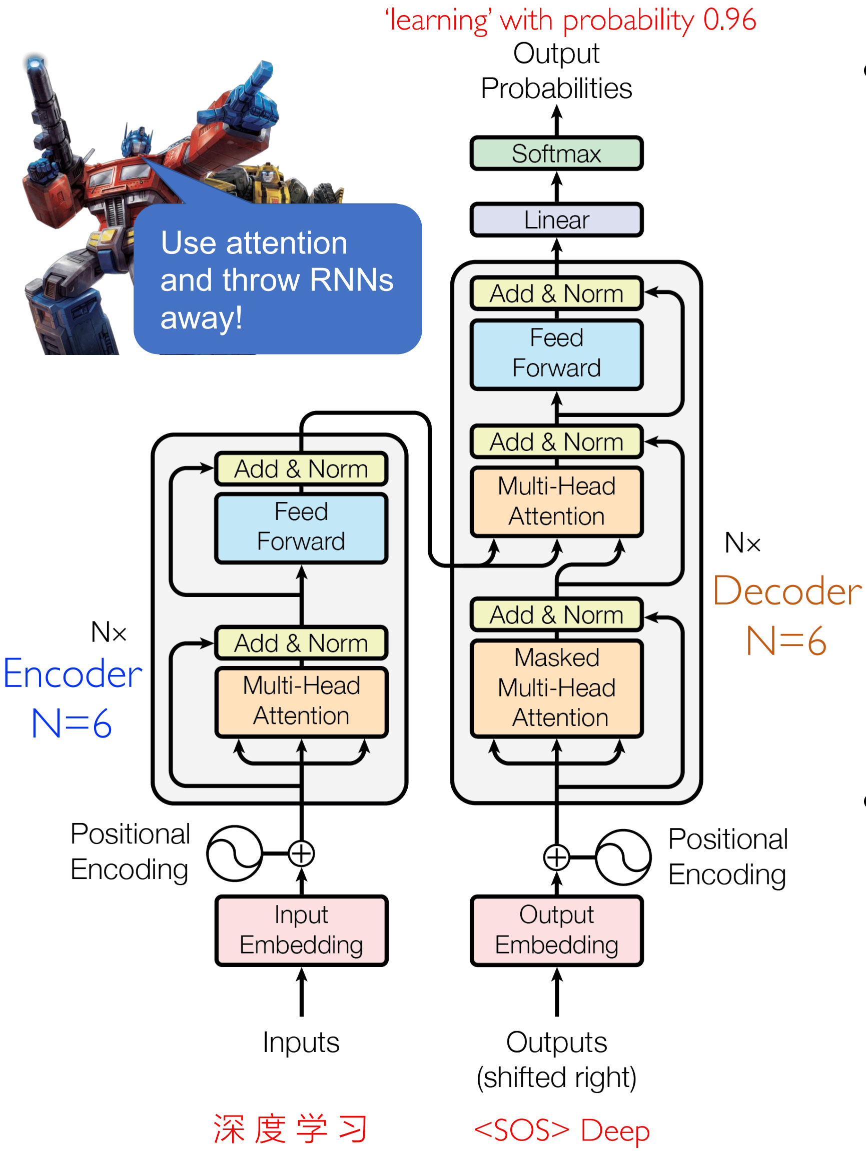
Inject some information about the relative or absolute position of the tokens in the sequence



- Each position i has a unique positional vector e_i yielded by fixed function

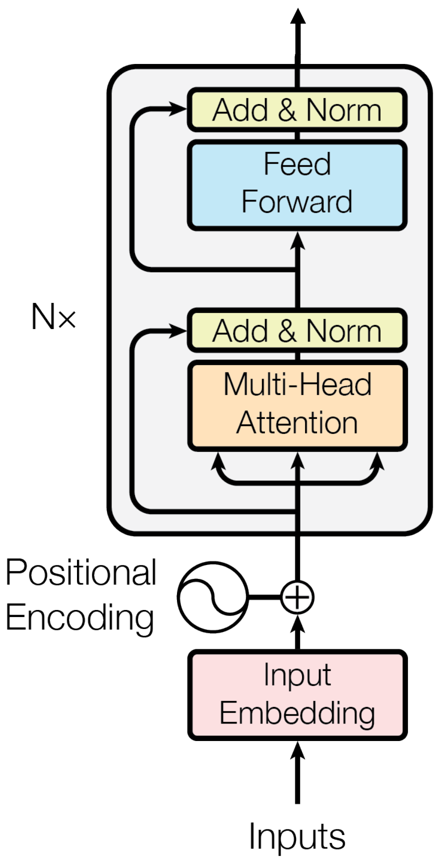
$$e_i(2j) = \sin\left(\frac{i}{10000^{\frac{2j}{d_{\text{model}}}}}\right), e_i(2j+1) = \cos\left(\frac{i}{10000^{\frac{2j}{d_{\text{model}}}}}\right)$$

Transformer



- Main components:
  ① Scaled Dot-Product Attention
  ② (Masked) Multi-Head Attention
  ③ Position-wise FFN
  ④ Residual Connections
  ⑤ Layer Normalization
  ⑥ Positional encoding
- Encoder-decoder architecture:
  ① Encoder
  ② Decoder with Masking
  ③ Encoder-Decoder Attention

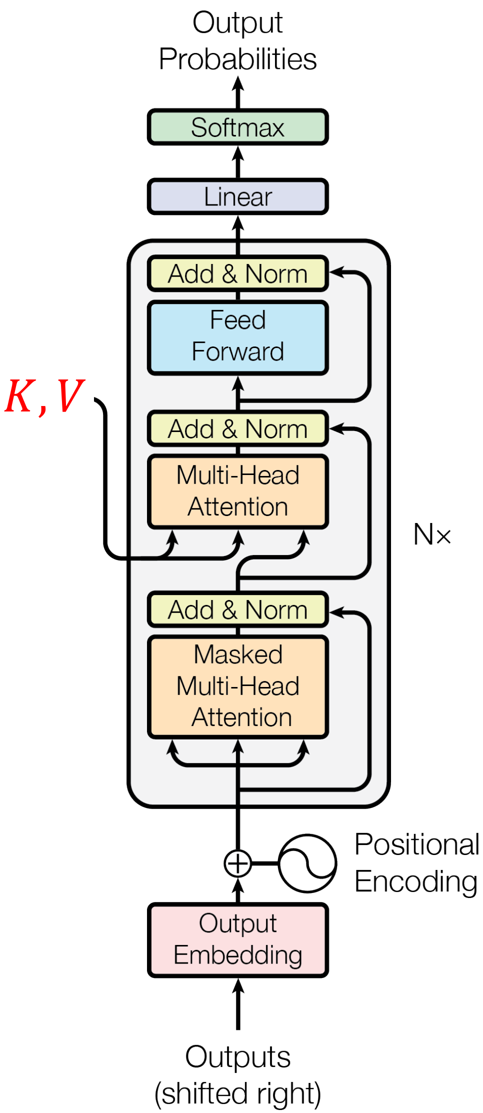
① Encoder



Encoder

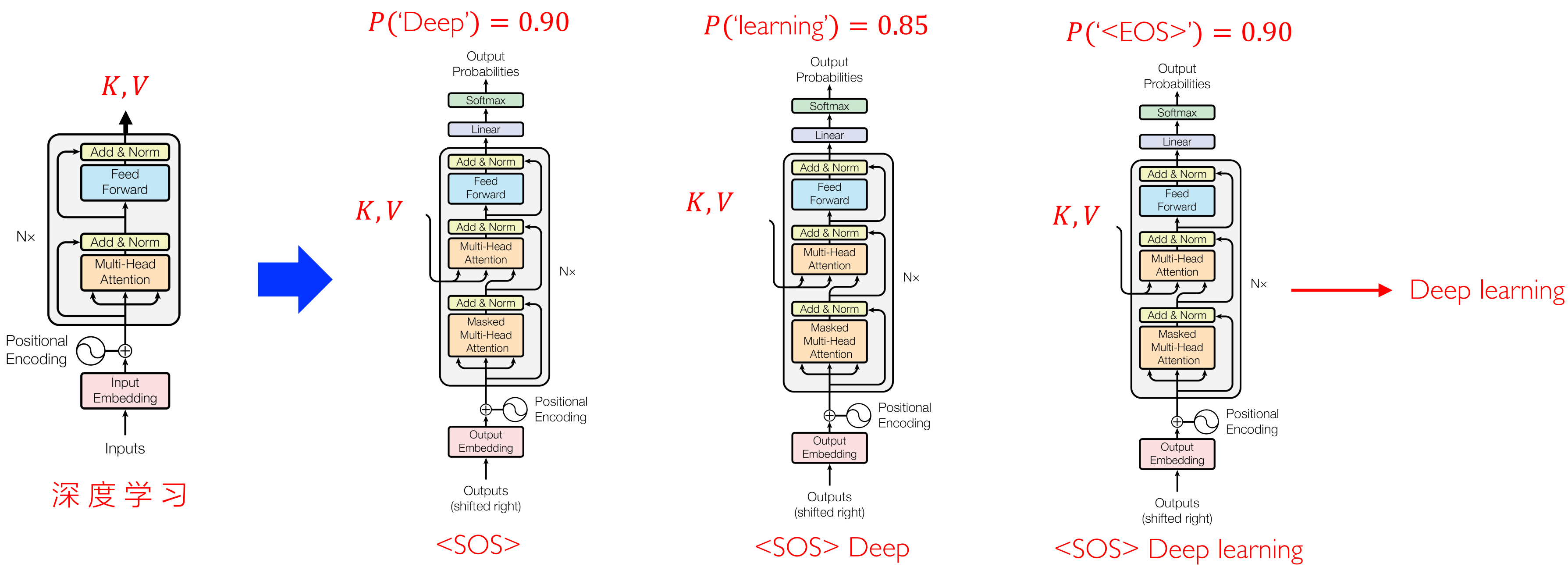
- The encoder stacks N encoder blocks
  - ($N = 6$ in original paper)
  - Multi-head self-attention
  - Position-wise FFN
  - Positional encoding at the bottoms of encoder
- Each block maintains shape
  - (#vectors, vector length)
- Weaknesses:
  - Complexity $O(n^2)$, n is sequence length
  - Difficult to train with huge parameters
  - Many heads are unimportant and can be pruned with little impact on performance

② Decoder



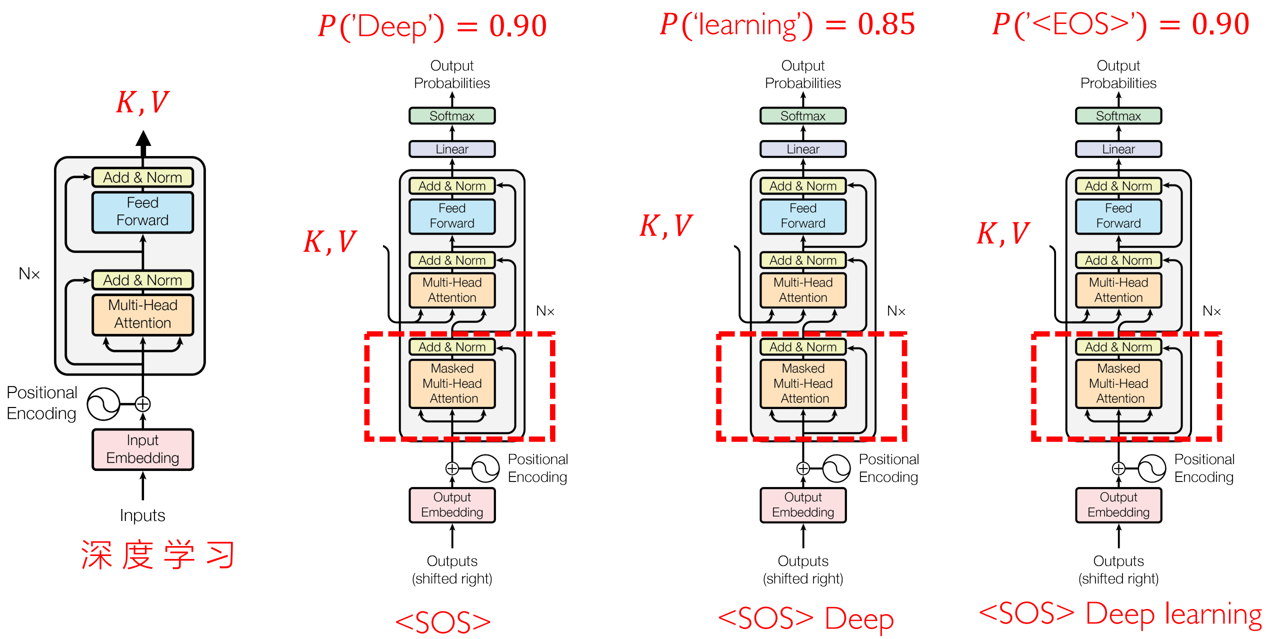
- The encoder stacks N decoder blocks
  - ($N = 6$ in original paper)
  - Multi-head self-attention
    - » K and V in the multi-head self-attention are from the encoder outputs
  - Masked multi-head self-attention
  - Position-wise FFN
  - Positional encoding at the bottoms of decoder
- Each block maintains shape
  - (#vectors, vector length)
- How does decoder work at inference and training?

② Decoder: Inference



- **Autoregressive (repeat until end of sequence):**
  - Input the translated words of target sequence
  - Compute next word probability
  - Sample next word (can use beam search to improve readability)

② Decoder: Training



('深 度 学 习', 'Deep learning')

$$\underbrace{\sum_{i=1}^{\text{len}_y} -\log P\ (y_i | y_{1:i-1}, x_{1:\text{len}_x})}$$

In parallel?

- Training with data pairs (original sequence, target sequence)
- **Autoregressive (repeat until end of sequence):**
  - Input the translated words of target sequence
  - Compute next word probability
  - Minimize cross-entropy loss: maximize probability of target word

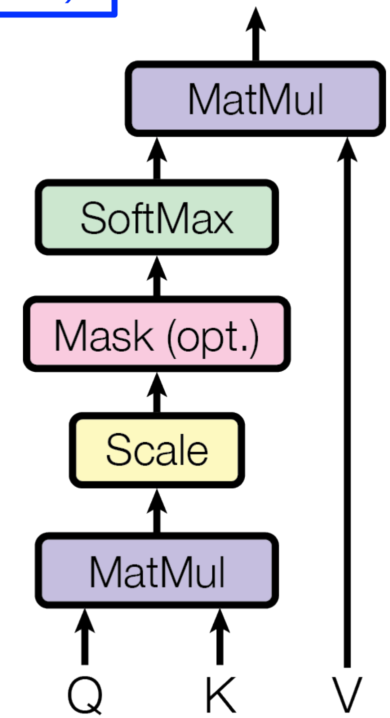
Inefficient

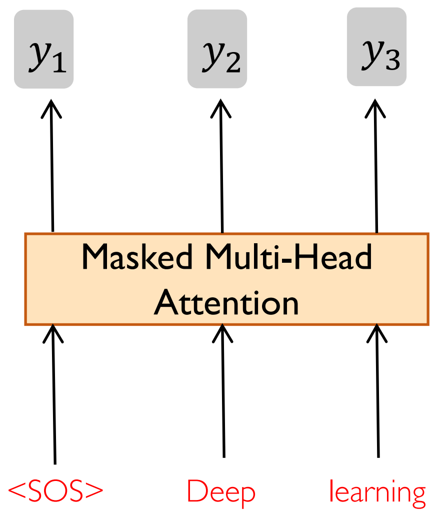
② Masked Multi-Head Self-Attention

Mask is used in the decoder to keep autoregressive (causal)

Train self-attention blocks in parallel,
and avoid the model cheating

$$q_i = W^q x_i$$
$$k_i = W^k x_i$$
$$v_i = W^v x_i$$



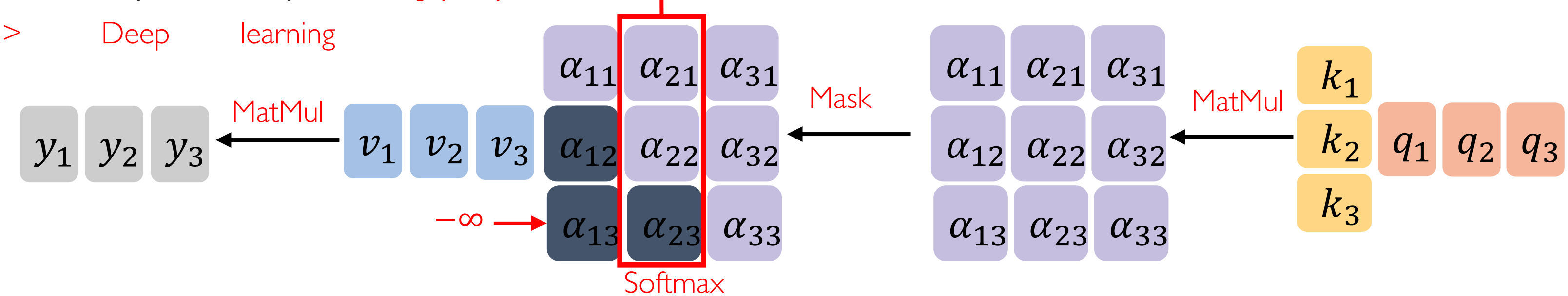


$$y_1 = \sum_i \alpha_{1i} v_i \qquad \alpha_{1i} = \frac{\exp(q_1 k_i)}{\sum_j \exp(q_1 k_j)}$$

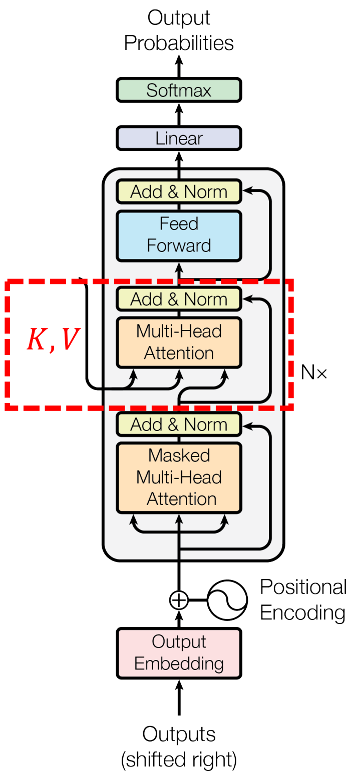
$$\alpha_i = \begin{bmatrix} \exp(\alpha_{21}) \\ \exp(\alpha_{22}) \\ 0 \end{bmatrix} \frac{1}{\exp(\alpha_{21}) + \exp(\alpha_{22}) + 0}$$

$\exp(-\infty)$

y_2 should only depend on x_1, x_2



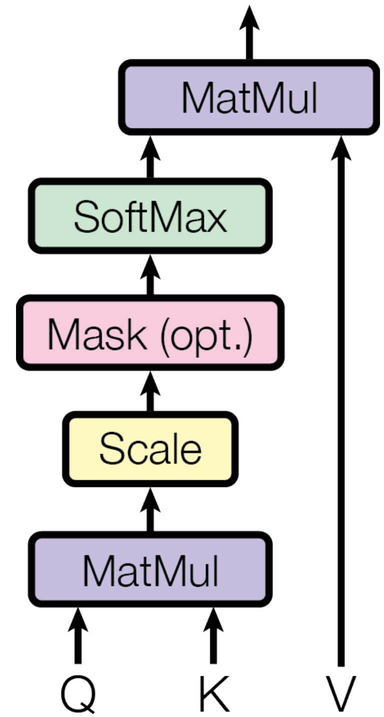
③ Encoder-Decoder Attention

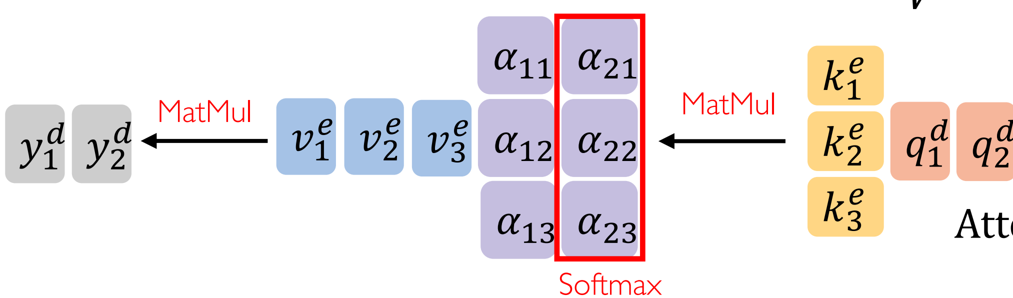


- Encoder-decoder self-attention takes two inputs:
  - $X^e: d^e \times \mathrm{len}_e$ (encoder outputs)
  - $X^d: d^d \times \mathrm{len}_d$ (decoder inputs)

$$q_i^d = W^q x_i^d$$
$$k_i^e = W^k x_i^e$$
$$v_i^e = W^v x_i^e$$

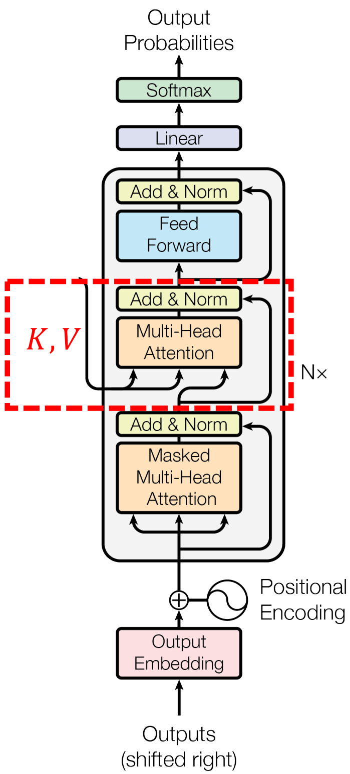
$$\underbrace{q_1^d \; q_2^d}_{Q^d} = W^q \; x_1^d \; x_2^d$$
$$\underbrace{k_1^e \; k_2^e \; k_3^e}_{K^e} = W^k \; x_1^e \; x_2^e \; x_3^e$$
$$\underbrace{v_1^e \; v_2^e \; v_3^e}_{V^e} = W^v \; x_1^e \; x_2^e \; x_3^e$$



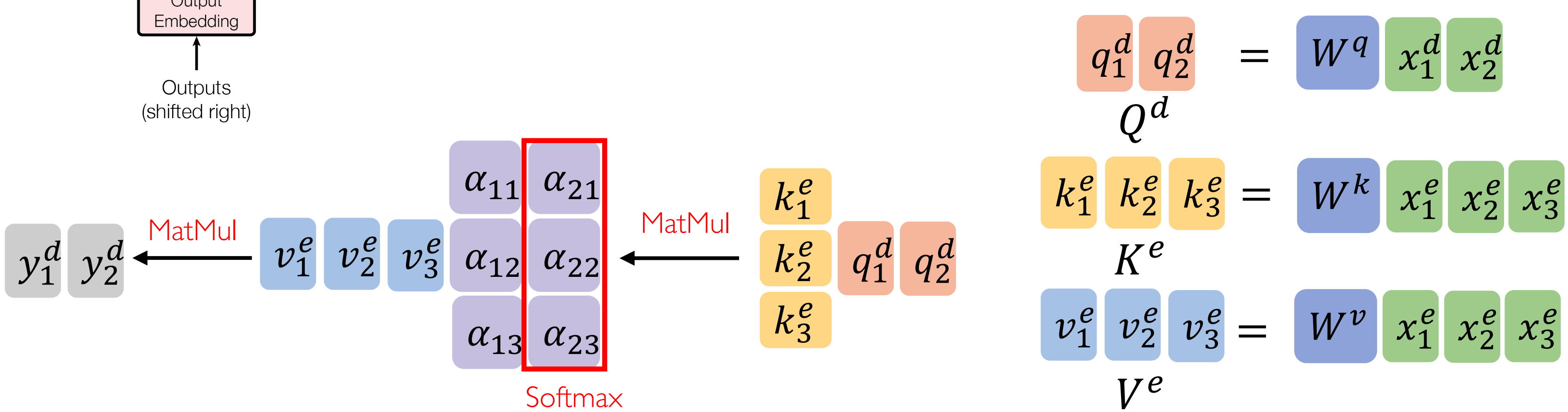


$$\mathrm{Attention}(Q^d, K^e, V^e) = \mathrm{Softmax}\left(\frac{Q^d {K^e}^T}{\sqrt{d_k}}\right) V^e$$

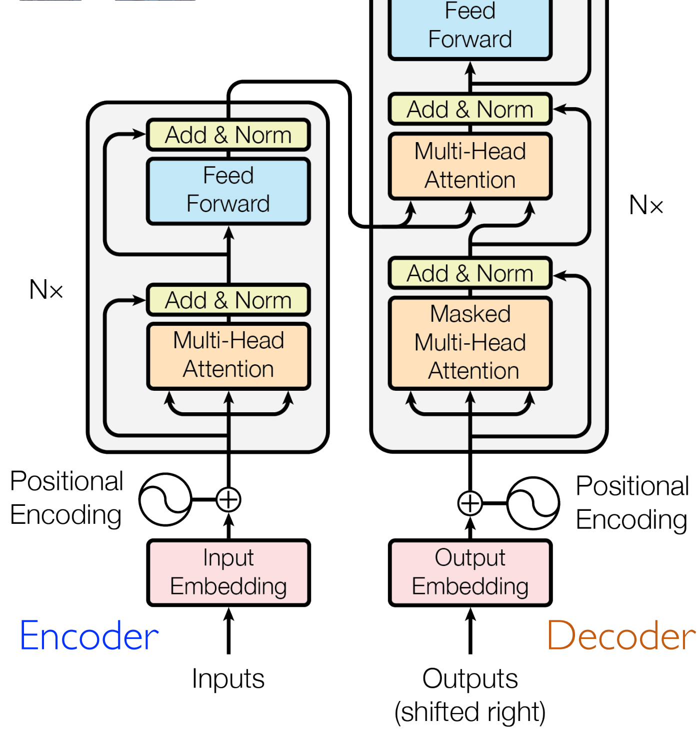
③ Encoder-Decoder Attention



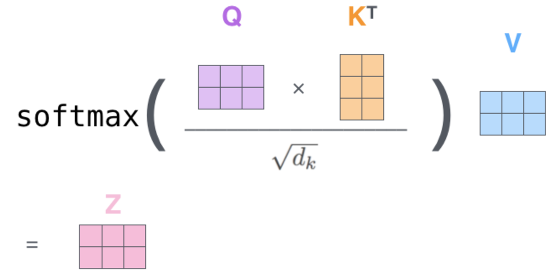
- Difference to standard self-attention:
  - Queries are computed by decoder X^d
  - Keys and values are computed by encoder X^e
  - Rectangle matrix multiplication
- No masking
- Output y_i depends on x_i^d and X^e
- The length of outputs is len_d

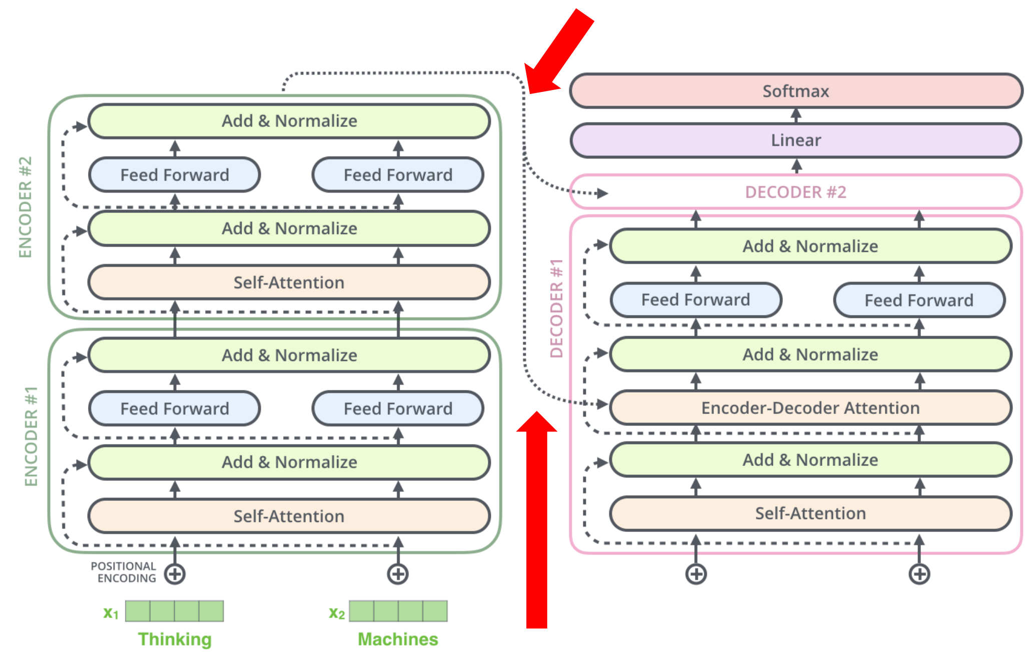


Transformer: All in One



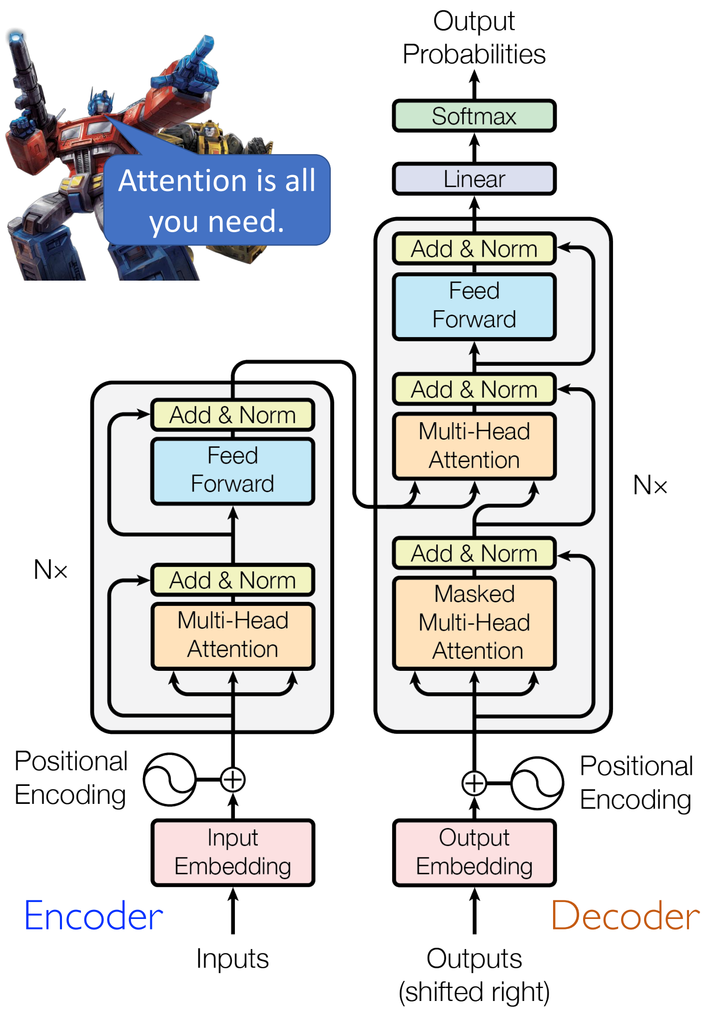
Self-Attention



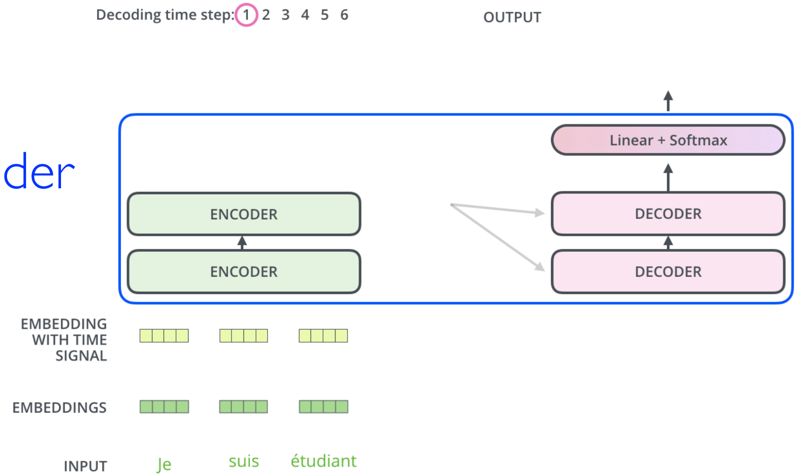


Vaswani et al. Attention is all you need. NIPS *2017.*

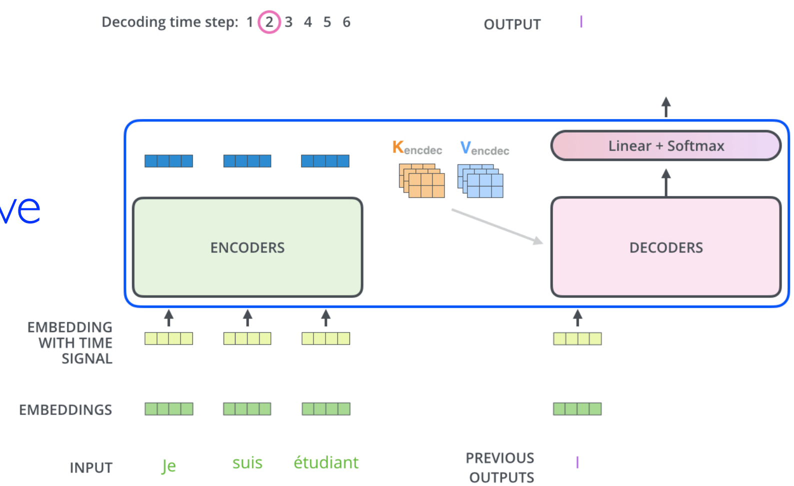
Transformer: All in One



Encoder-decoder attention



Decoder autoregressive prediction



Vaswani et al. Attention is all you need. **NIPS** *2017.*

RNN vs. Transformer

- RNN
  - **Strength:** Powerful at modeling sequences (Turing complete).
  - **Drawback:** Their sequential nature makes it quite slow to train.
  - Fail on large-scale language understanding tasks like translation.
- Transformer
  - **Strength:** Process in parallel thus much faster to train.
  - **Drawback:** Fails on smaller and more structured language understanding tasks, or even simple algorithmic tasks such as copying a string while RNNs perform well.
  - Requires a lot of memory: $n \times m$ alignment and attention scalers need to be calculated and stored for a single self-attention head.

Inductive Biases

- Convolutional networks
  - Locality (restricted window in grids)
  - Translation invariance (shared kernel across spatial positions)
- Recurrent networks
  - Locality (Markovian structure)
  - Temporal invariance (shared function across timesteps)
- Transformer
  - No structural prior (prone to overfitting in small-scale data)
  - Permutation equivariance (requires position representations to encode sequences)