Announcement

- Happy hour (optional) vote please via BB!
  - date: TBA (11/21 or 11/28), **19:00-21:00**
  - place: **브롱스(Bronx)** at #206, 2nd floor
  - **free** pizza / chickens / drinks (non-alcohol)
  - only for **AI502/IE408/IE511** students

Previously we learned…

- Inductive Biases in Convolutional Networks
  - Locality Principle
  - Spatial Invariance
- Inductive Biases in Graph Neural Networks
  - Permutation Invariance
- Inductive Biases in Recurrent Neural Networks
  - Sequentiality
  - Temporal Invariance

Attention and Transformer

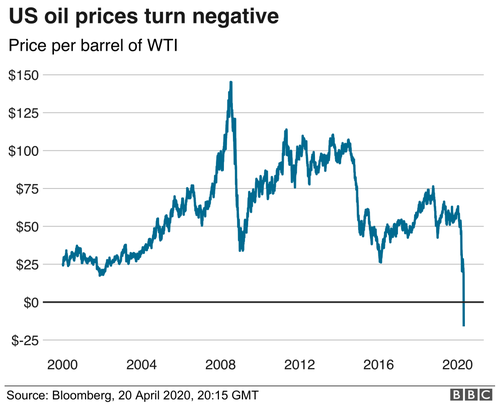
Principles of Deep Learning (AI502/IE408/IE511)

Sungbin Lim (UNIST AIGS & IE)

Contact: ai502deeplearning@gmail.com

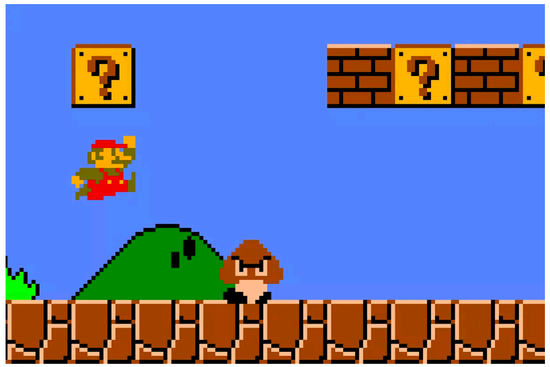
(Review) Understanding sequential data

- $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$: sequential data
  - time-series data
  - text data
  - sequential decision making
  - image fames in video
- Sequential data are **non-i.i.d**.
- Sequence length is not fixed
  - variable length

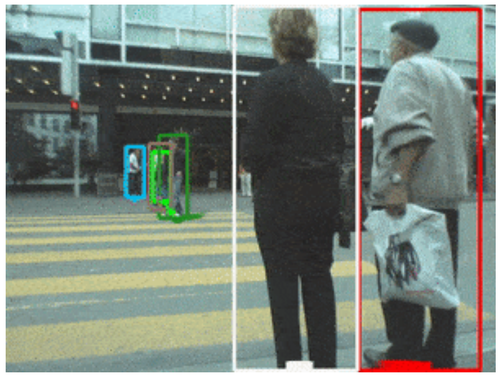


time-series data

text data



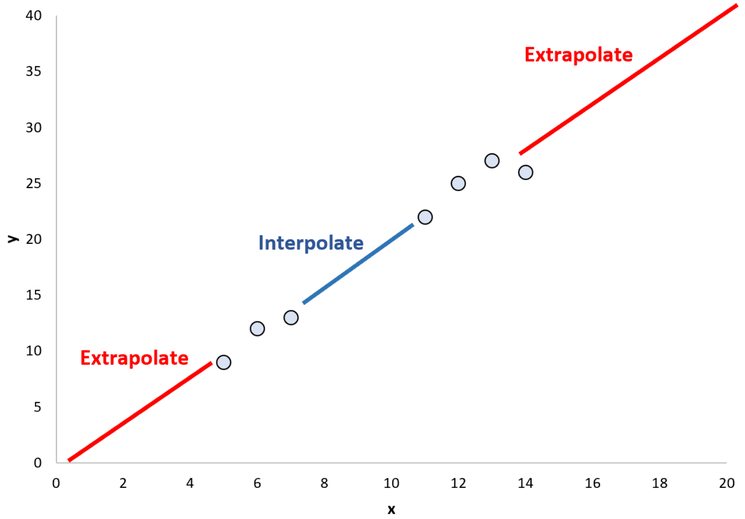
reinforcement learning



video data

(Review) Why sequential patterns are difficult?

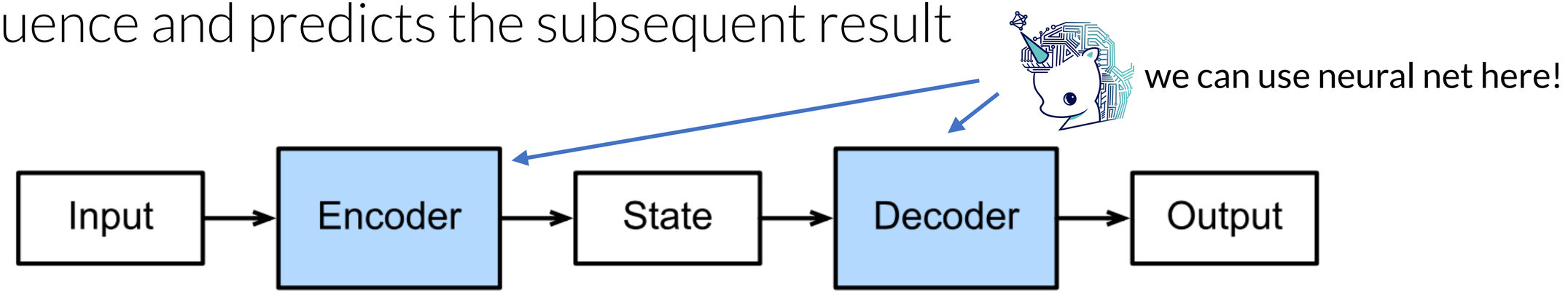
- Sequence length is not fixed
  - variable length → MLP(x), CNN(o)
- Extrapolation vs Interpolation
  - hindsight is easier than foresight (ex. stock price)
- Sequential data are **non-i.i.d**.
  - sequence order is crucial
  - if we permute them, they would make little sense
  - *dog* bites *man* vs *man* bites *dog*



it is possible… huh?

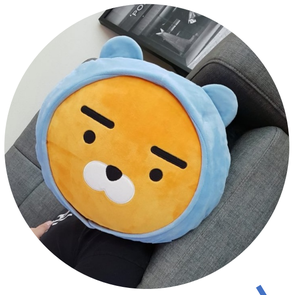
Encoder-Decoder Architecture

- To handle seq-to-seq problems (e.g. machine translation), inputs and outputs are of varying lengths that are unaligned
- Standard approach is an ***encoder-decoder*** architecture
  - **encoder** encodes variable-length sequence as inputs into state
  - **decoder** takes in the encoded state and the context of the target sequence and predicts the subsequent result



Data Compression

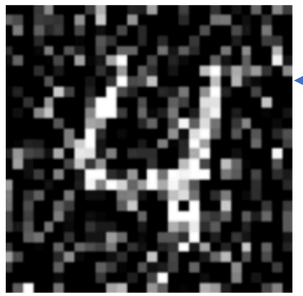
$$\min_{\theta} \|\mathbf{x} - f_\theta(\mathbf{x})\|$$



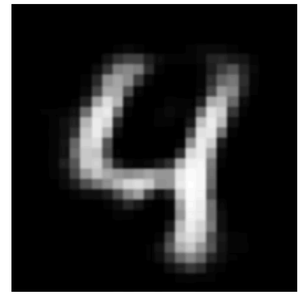
state captures compressed information

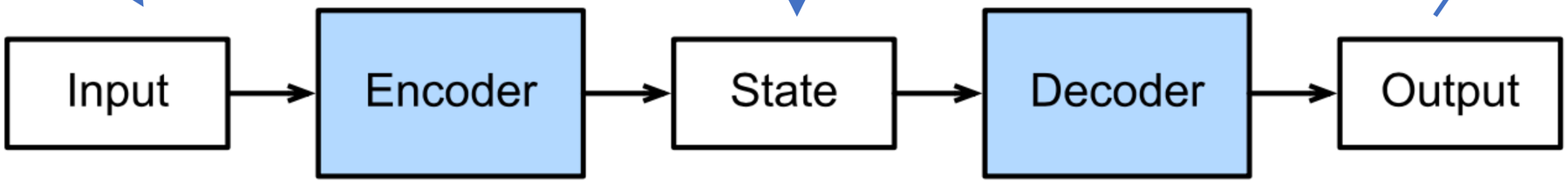
Denoising

$$\min_{\theta} \|\mathbf{x} - f_\theta(\mathbf{x} + \text{noise})\|$$



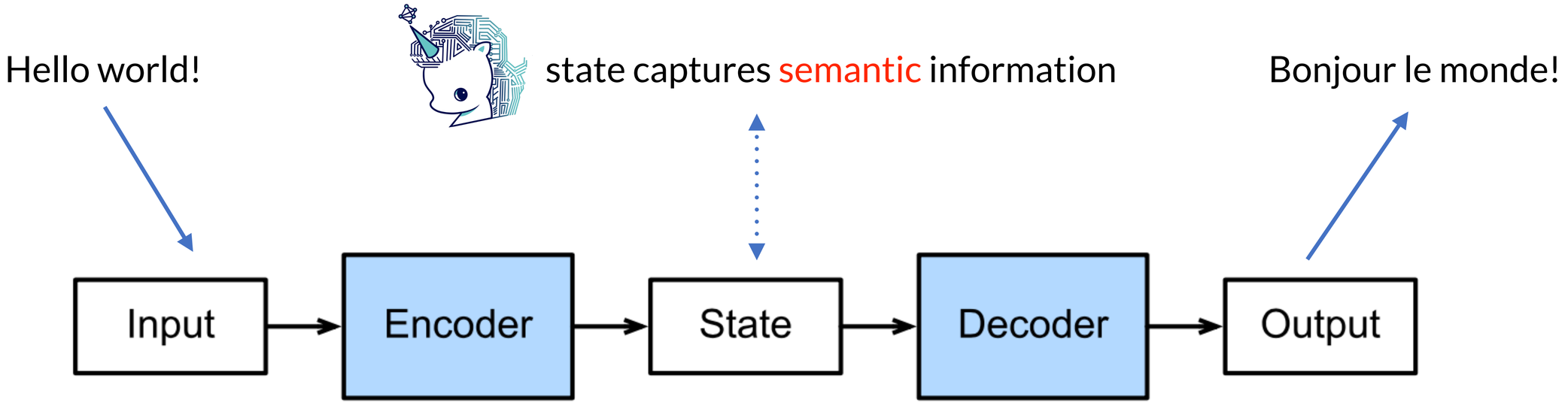
state captures feature information





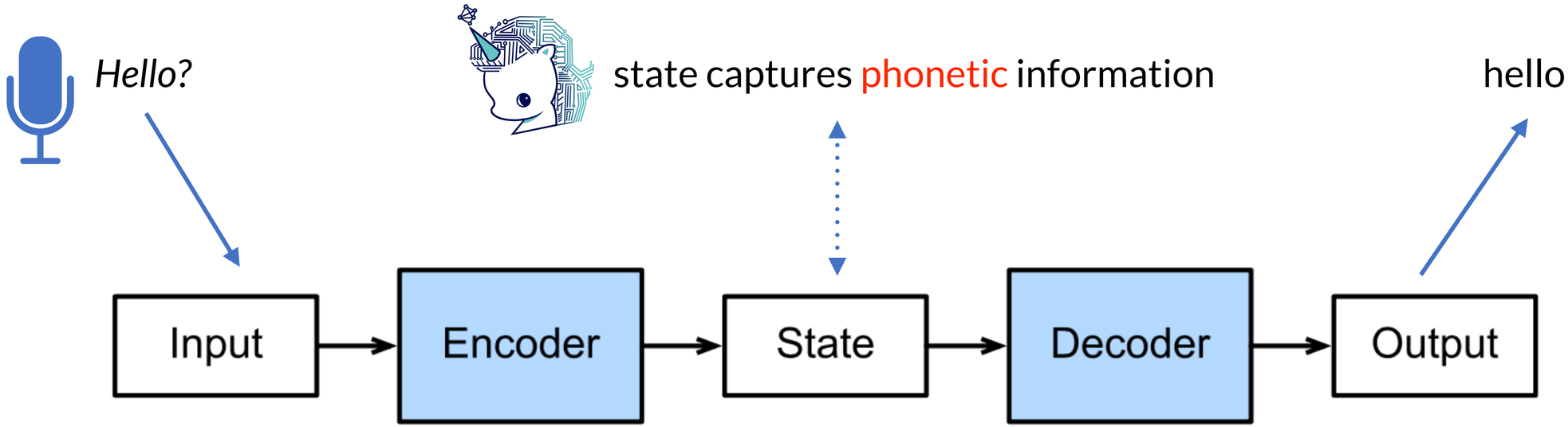
Machine Translation

- In MT, encoder transforms a source sentence into state, and then decoder uses this state to generate the translated target



Speech Recognition

- Similarly, encoder transforms a source voice into state, and then decoder uses this state to generate the sentence



Problem of RNN encoder-decoder?

- RNN (including LSTM, GRU) based encoder-decoder architectures take a **variable-length** sequence as inputs and transform it into a hidden state
  - encoder transforms the hidden states at all time steps into a context

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1}) \quad \mathbf{c} = q(\mathbf{h}_1, \ldots, \mathbf{h}_T)$$

  - decoder transforms the previous hidden state and the context

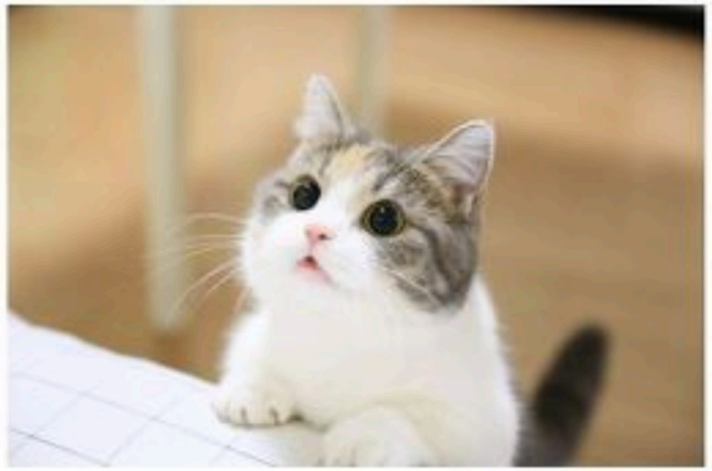
$$\mathbf{s}_\tau = g(y_{\tau-1}, \mathbf{c}, \mathbf{s}_{\tau-1})$$

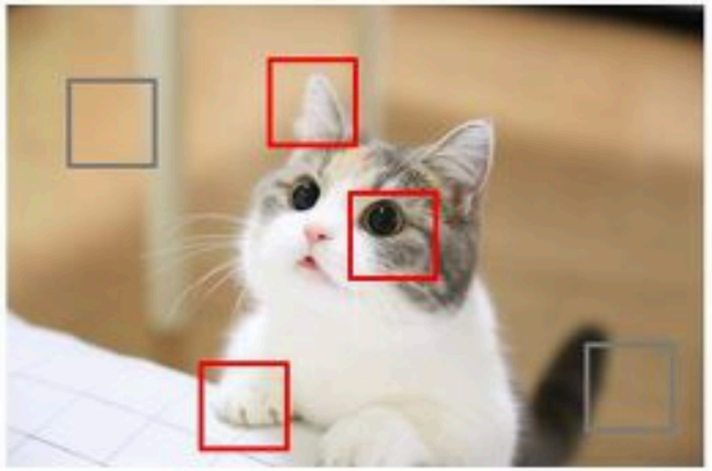
- RNNs compress the entire input into a **fixed-length** representation
  - how can we access encoded inputs without having to compress the entire input into a single fixed-length context representation?

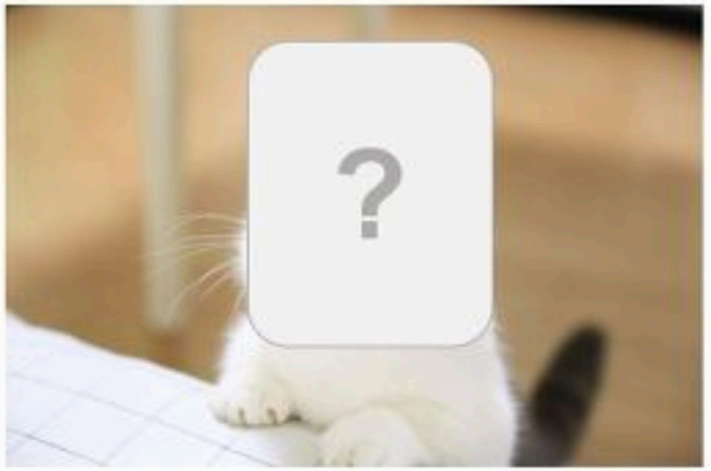
Attention in Cognitive Neuroscience

- Attention enable the human to prioritize the perception in order to deal effectively with others

Task: what is this animal?

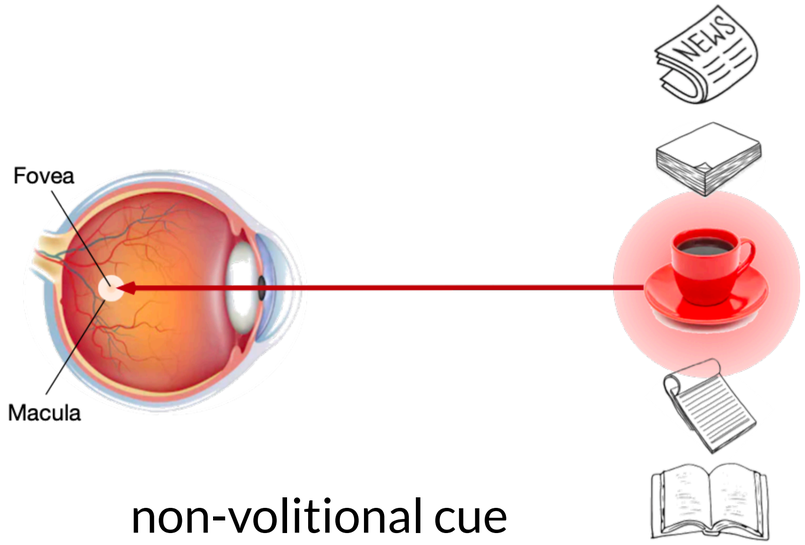




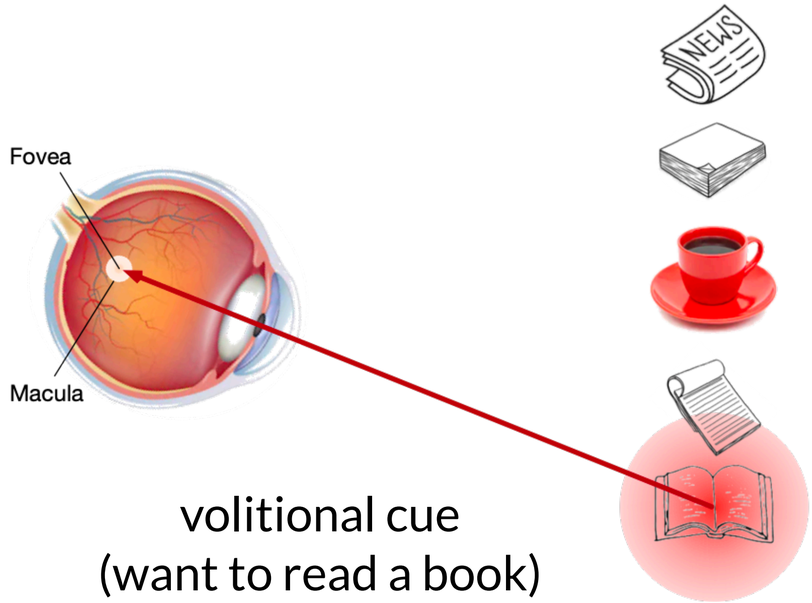


Attention in Cognitive Neuroscience

- Attention enable the human to prioritize the perception in order to deal effectively with others
- In cognitive neuroscience, scientists studied attention since the 1890



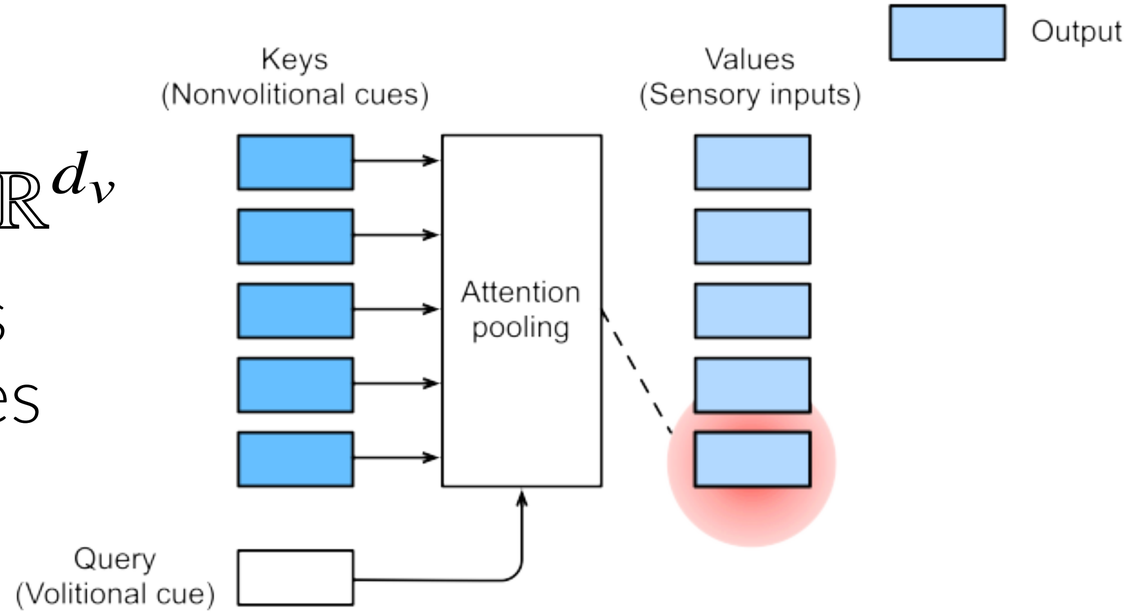
non-volitional cue



volitional cue
(want to read a book)

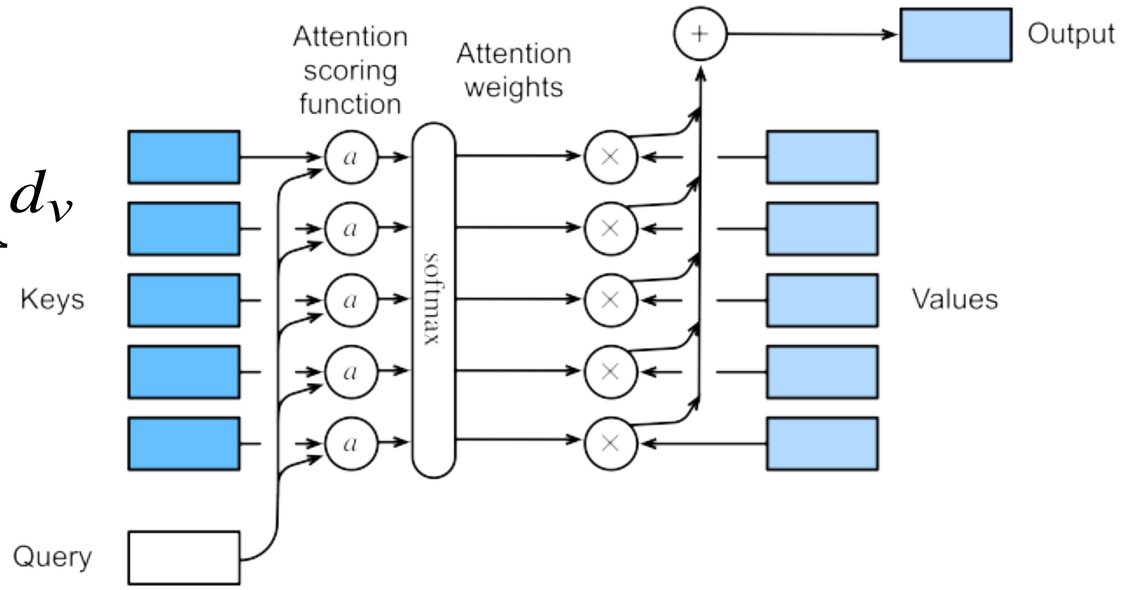
How can we design Attention module?

- Attention can be designed as a ***pooling*** with bias alignment over inputs
  - **keys:** list of cues $\mathbf{k} \in \mathbb{R}^{d_k}$
  - **query:** volitional cue $\mathbf{q} \in \mathbb{R}^{d_q}$
  - **values:** feature representation $\mathbf{v} \in \mathbb{R}^{d_v}$
- Given any query, attention mechanisms bias selection over represented features via ***attention pooling***



How can we design Attention module?

- Attention can be designed as a ***pooling*** with bias alignment over inputs
  - **keys:** list of cues $\mathbf{k} \in \mathbb{R}^{d_k}$
  - **query:** volitional cue $\mathbf{q} \in \mathbb{R}^{d_q}$
  - **values:** feature representation $\mathbf{v} \in \mathbb{R}^{d_v}$
- Given any query, attention mechanisms bias selection over represented features via ***attention pooling***



$$f(\mathbf{q}, \{(\mathbf{k}_i, \mathbf{v}_i)\}_{i=1}^m) = \sum_{i=1}^m \alpha(\mathbf{q}, \mathbf{k}_i)\mathbf{v}_i \qquad \alpha(\mathbf{q}, \mathbf{k}_i) = \mathrm{softmax}(a(\mathbf{q}, \mathbf{k}_i)) = \frac{\exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_{j=1}^m \exp(a(\mathbf{q}, \mathbf{k}_j))}$$

attention weights — attention scoring function

How to compute attention scoring?

- Additive pooling $d_k \neq d_q$

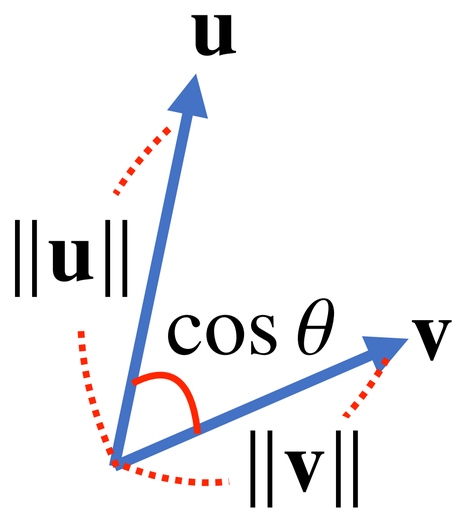
$$a(\mathbf{q}, \mathbf{k}) = \langle \mathbf{w}, \tanh(\mathbf{W}_q \mathbf{q} + \mathbf{W}_k \mathbf{k}) \rangle$$
$$= \langle \mathbf{w}, \tanh([\mathbf{W}_q, \mathbf{W}_k][\mathbf{q}, \mathbf{k}]) \rangle$$

$\mathbf{w} \in \mathbb{R}^{d_h}$, $\mathbf{W}_q \in \mathbb{R}^{d_h \times d_q}$, $\mathbf{W}_k \in \mathbb{R}^{d_h \times d_k}$

concatenation

- Scaled Dot-product $d_k = d_q = d$

$$a(\mathbf{q}, \mathbf{k}) = \frac{\langle \mathbf{q}, \mathbf{k} \rangle}{\sqrt{d}}$$



Here $\langle \cdot, \cdot \rangle$ denotes dot-product which is also called inner product

$$\langle \mathbf{u}, \mathbf{v} \rangle = \sum_{i=1}^{d} u_i v_i = \|\mathbf{u}\| \|\mathbf{v}\| \cos\theta$$

Bahdanau Attention

- Previous RNN encoder-decoder architecture used the same context variable that encodes the entire input sequence at each decoding step

$$\mathbf{s}_\tau = g(y_{\tau-1}, \mathbf{c}, \mathbf{s}_{\tau-1}) \qquad \mathbf{c} = q(\mathbf{h}_1, \ldots, \mathbf{h}_T)$$

Problem of RNN encoder-decoder?

- RNN (including LSTM, GRU) based encoder-decoder architectures take a **variable-length** sequence as inputs and transform it into a hidden state
  - encoder transforms the hidden states at all time steps into a context

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1}) \quad \mathbf{c} = q(\mathbf{h}_1, \ldots, \mathbf{h}_T)$$

  - decoder transforms the previous hidden state and the context

$$\mathbf{s}_\tau = g(y_{\tau-1}, \mathbf{c}, \mathbf{s}_{\tau-1})$$

- RNNs compress the entire input into a **fixed-length** representation
  - how can we access encoded inputs without having to compress the entire input into a single fixed-length representation?

Bahdanau Attention

- Previous RNN encoder-decoder architecture used the same context variable that encodes the entire input sequence at each decoding step

$$\mathbf{s}_\tau = g(y_{\tau-1}, \mathbf{c}_{\tau-1}, \mathbf{s}_{\tau-1}) \quad \mathbf{c}_\tau = \sum_{t=1}^{T} \alpha(\mathbf{s}_{\tau-1}, \mathbf{h}_t)\mathbf{h}_t$$

query key value

Note that **k** and **v** are same in this formula

Problem of RNN encoder-decoder?

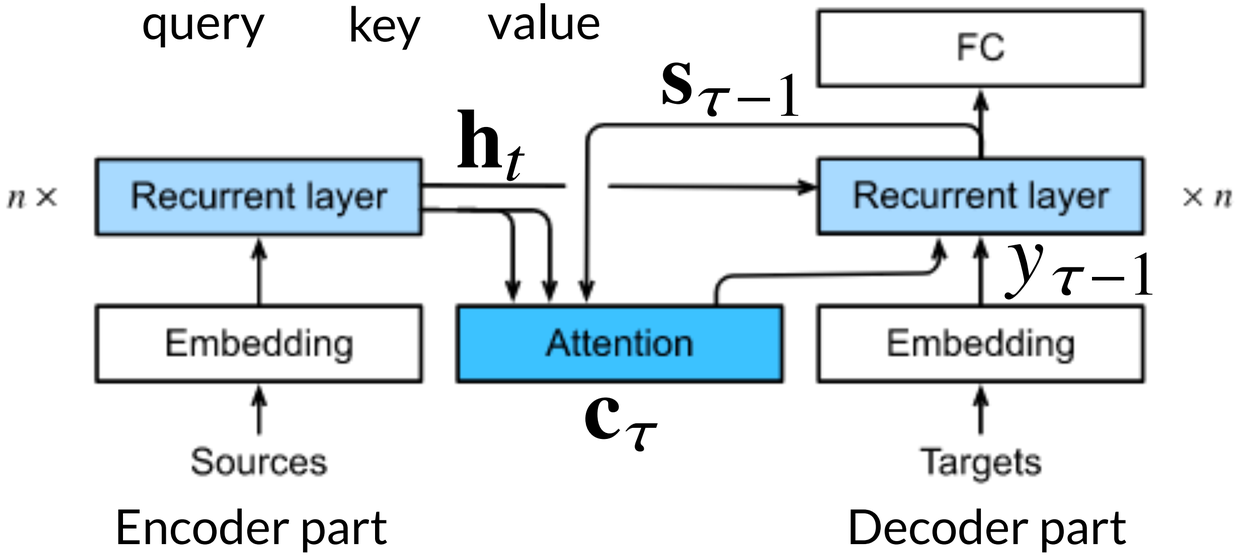
- RNN (including LSTM, GRU) based encoder-decoder architectures take a **variable-length** sequence as inputs and transform it into a hidden state
  - encoder transforms the hidden states at all time steps into a context

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1}) \quad \mathbf{c} = q(\mathbf{h}_1, \ldots, \mathbf{h}_T)$$

  - decoder transforms the previous hidden state and the context

$$\mathbf{s}_\tau = g(y_{\tau-1}, \mathbf{c}, \mathbf{s}_{\tau-1})$$

- RNNs compress the entire input into a **fixed-length** representation
  - how can we access encoded inputs without having to compress the entire input into a single fixed-length representation?



Bahdanau Attention

- Previous RNN encoder-decoder architecture used ~~the same context variable~~ that encodes the entire input sequence at each decoding step

$$\mathbf{s}_\tau = g(y_{\tau-1}, \mathbf{c}_{\tau-1}, \mathbf{s}_{\tau-1}) \quad \mathbf{c}_\tau = \sum_{t=1}^{T} \alpha(\mathbf{s}_{\tau-1}, \mathbf{h}_t)\mathbf{h}_t$$

query key value

We only change the decoder computation

Problem of RNN encoder-decoder?

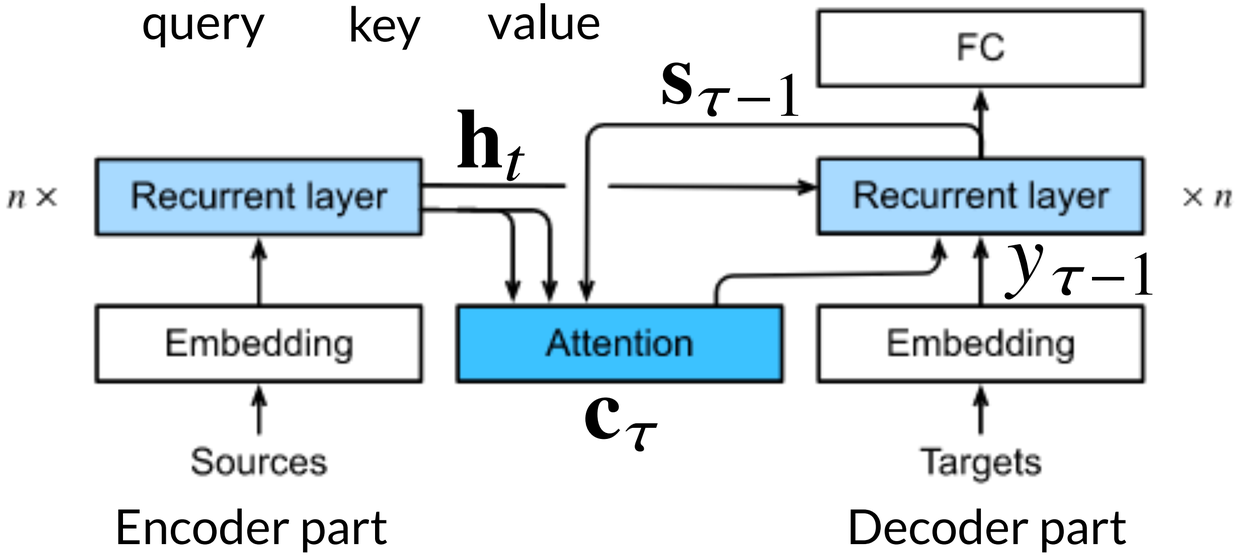
- RNN (including LSTM, GRU) based encoder-decoder architectures take a **variable-length** sequence as inputs and transform it into a hidden state
  - encoder transforms the hidden states at all time steps into a context

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1}) \quad \mathbf{c} = q(\mathbf{h}_1, \ldots, \mathbf{h}_T)$$

  - decoder transforms the previous hidden state and the context

$$\mathbf{s}_\tau = g(y_{\tau-1}, \mathbf{c}, \mathbf{s}_{\tau-1})$$

- RNNs compress the entire input into a **fixed-length** representation
  - how can we access encoded inputs without having to compress the entire input into a single fixed-length representation?



Bahdanau Attention

- Previous RNN encoder-decoder architecture used ~~the same context variable~~ that encodes the entire input sequence at each decoding step

$$\mathbf{s}_\tau = g(y_{\tau-1}, \mathbf{c}_{\tau-1}, \mathbf{s}_{\tau-1}) \quad \mathbf{c}_\tau = \sum_{t=1}^{T} \alpha(\mathbf{s}_{\tau-1}, \mathbf{h}_t)\mathbf{h}_t$$

We only change the decoder computation

query key value

Can't we use attention for encoder?

Problem of RNN encoder-decoder?

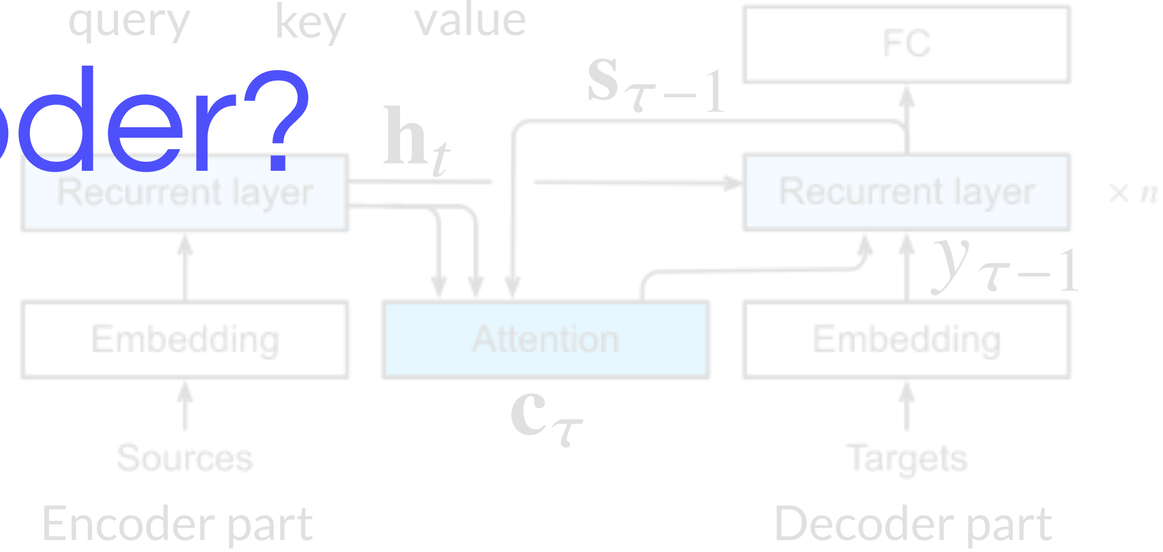
- RNN (including LSTM, GRU) based encoder-decoder architecture take a **variable-length** sequence as inputs and transform it into a hidden state
  - encoder transforms the hidden states at all time steps into a context

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1}) \quad \mathbf{c} = q(\mathbf{h}_1, \ldots, \mathbf{h}_T)$$

  - decoder transforms the previous hidden state and the context

$$\mathbf{s}_\tau = g(y_{\tau-1}, \mathbf{c}, \mathbf{s}_{\tau-1})$$

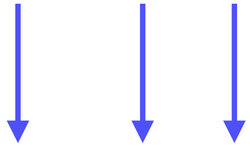
- RNNs compress the entire input into a **fixed-length** representation
  - how can we access encoded inputs without having to compress the entire input into a single fixed-length representation?



Self-Attention

- We can feed a sequence of tokens into attention pooling so that the same set of tokens act as queries, keys, and values. This is called **self-attention**.

$$f(\mathbf{q}, \{(\mathbf{k}_i, \mathbf{v}_i)\}_{i=1}^{m}) = \sum_{i=1}^{m} \alpha(\mathbf{q}, \mathbf{k}_i)\mathbf{v}_i$$



$$f(\mathbf{x}, \{(\mathbf{x}_i, \mathbf{x}_i)\}_{i=1}^{n}) = \sum_{i=1}^{n} \alpha(\mathbf{x}, \mathbf{x}_i)\mathbf{x}_i$$

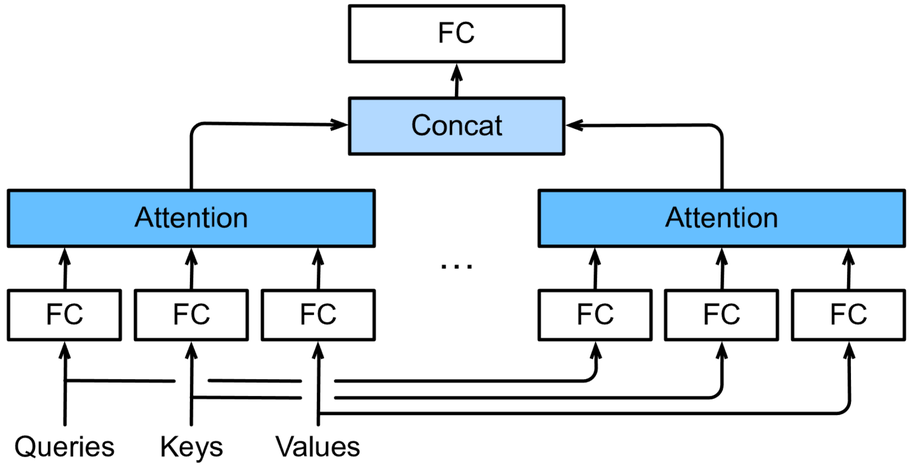
Self-Attention

Self-attention with MHA benefits from parallel computation

- We can feed a sequence of tokens into attention pooling so that the same set of tokens act as queries, keys, and values. This is called **self-attention**.

$$f(\mathbf{q}, \{(\mathbf{k}_i, \mathbf{v}_i)\}_{i=1}^{m}) = \sum_{i=1}^{m} \alpha(\mathbf{q}, \mathbf{k}_i)\mathbf{v}_i$$

$$f(\mathbf{x}, \{(\mathbf{x}_i, \mathbf{x}_i)\}_{i=1}^{n}) = \sum_{i=1}^{n} \alpha(\mathbf{x}, \mathbf{x}_i)\mathbf{x}_i$$



$$\mathbf{h}_m = f(\mathbf{W}_m^{(q)}\mathbf{x}, \{\mathbf{W}_m^{(k)}\mathbf{x}_i, \mathbf{W}_m^{(v)}\mathbf{x}_i\}_{i=1}^{n}) = \sum_{i=1}^{n} \alpha(\mathbf{W}_m^{(q)}\mathbf{x}, \mathbf{W}_m^{(k)}\mathbf{x}_i)\mathbf{W}_m^{(v)}\mathbf{x}_i$$

(m-th head of MHA)

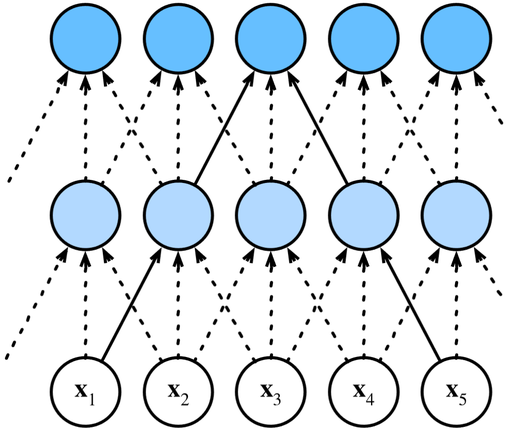
- We use **M**ulti-**H**ead **A**ttention to combine input information differently

Comparing CNN, RNN, and Self-Attention

- We can compare self-attention with in terms of computational complexity (**CC**), sequential operations (**SO**), maximum path lengths (**MPL**)

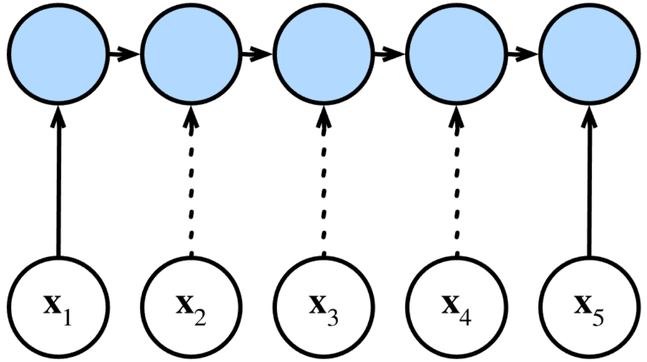
Note that high SO prevents parallel computation while MPL degrades modeling long-range dependency within the sequence

Convolution



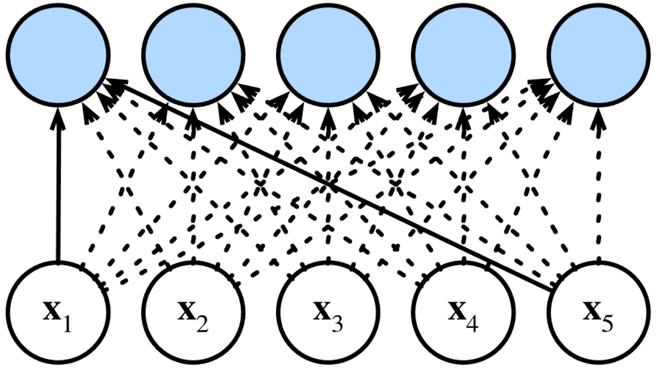
CC: $\mathcal{O}(knd^2)$, SO: $\mathcal{O}(1)$, MPL: $\mathcal{O}(n/k)$

RNN



CC: $\mathcal{O}(d^2)$, SO: $\mathcal{O}(nd^2)$, MPL: $\mathcal{O}(n)$

Self-attention



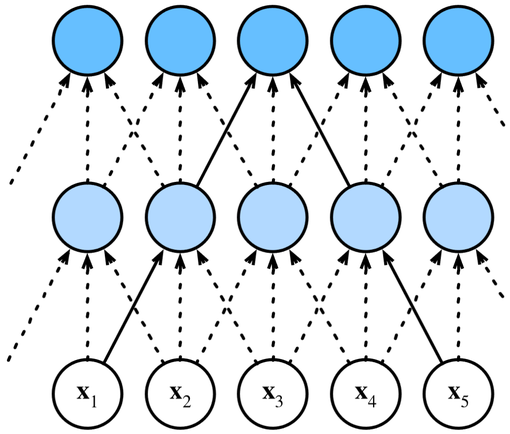
CC: $\mathcal{O}(n^2d)$, SO: $\mathcal{O}(1)$, MPL: $\mathcal{O}(1)$

Comparing CNN, RNN, and Self-Attention

- We can compare self-attention with in terms of computational complexity (**CC**), sequential operations (**SO**), maximum path lengths (**MPL**)

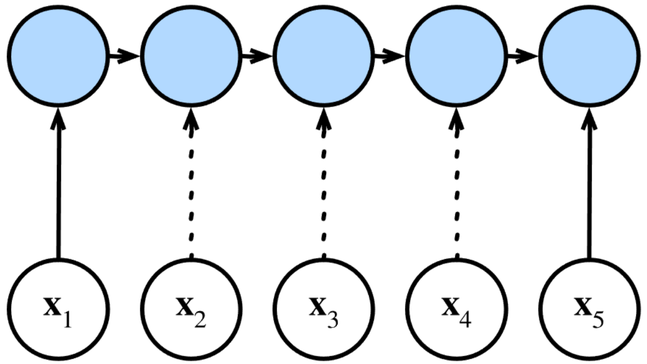
Self-Attention benefits from parallel computation and sequential modeling but suffers computational complexity when the input sequence is too long

Convolution



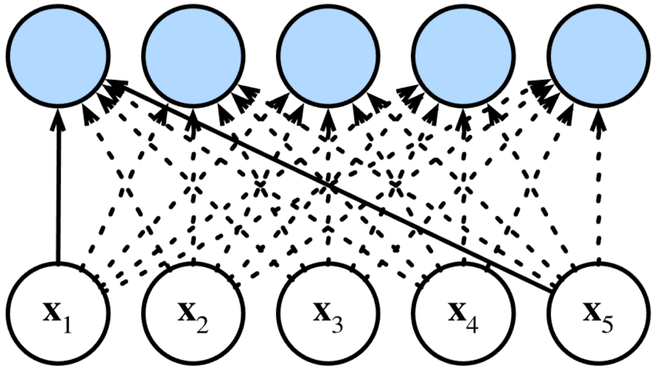
CC: $\mathcal{O}(knd^2)$, SO: $\mathcal{O}(1)$, MPL: $\mathcal{O}(n/k)$

RNN



CC: $\mathcal{O}(d^2)$, SO: $\mathcal{O}(nd^2)$, MPL: $\mathcal{O}(n)$

Self-attention



CC: $\mathcal{O}(n^2d)$, SO: $\mathcal{O}(1)$, MPL: $\mathcal{O}(1)$

Positional Encoding

- Self-attention is permutation invariant → no sequence order information!
  - to inject positional information, we use **positional encoding**

$$f(\mathbf{x}, \{(\mathbf{x}_i, \mathbf{x}_i)\}_{i=1}^n) = \sum_{i=1}^n \alpha(\mathbf{x}, \mathbf{x}_i)\mathbf{x}_i$$

permutation

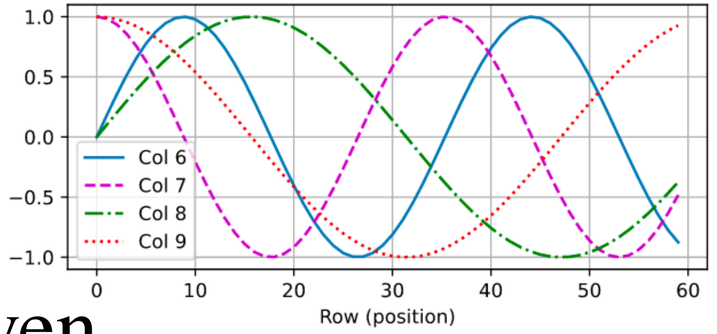
$$f(\mathbf{x}, \{(\mathbf{x}_{\pi(i)}, \mathbf{x}_{\pi(i)})_{i=1}^n\}) = \sum_{i=1}^n \alpha(\mathbf{x}, \mathbf{x}_{\pi(i)})\mathbf{x}_{\pi(i)}$$

Self-attention is permutation-invariant
since softmax is permutation-equivariant

Positional Encoding

- Self-attention is permutation invariant → no sequence order information!
  - to inject positional information, we use **positional encoding**

$$f(\mathbf{P}(\mathbf{x}), \{(\mathbf{P}(\mathbf{x}_i), \mathbf{P}(\mathbf{x}_i))\}_{i=1}^{n}) = \sum_{i=1}^{n} \alpha(\mathbf{P}(\mathbf{x}), \mathbf{P}(\mathbf{x}_i))\mathbf{P}(\mathbf{x}_i)$$

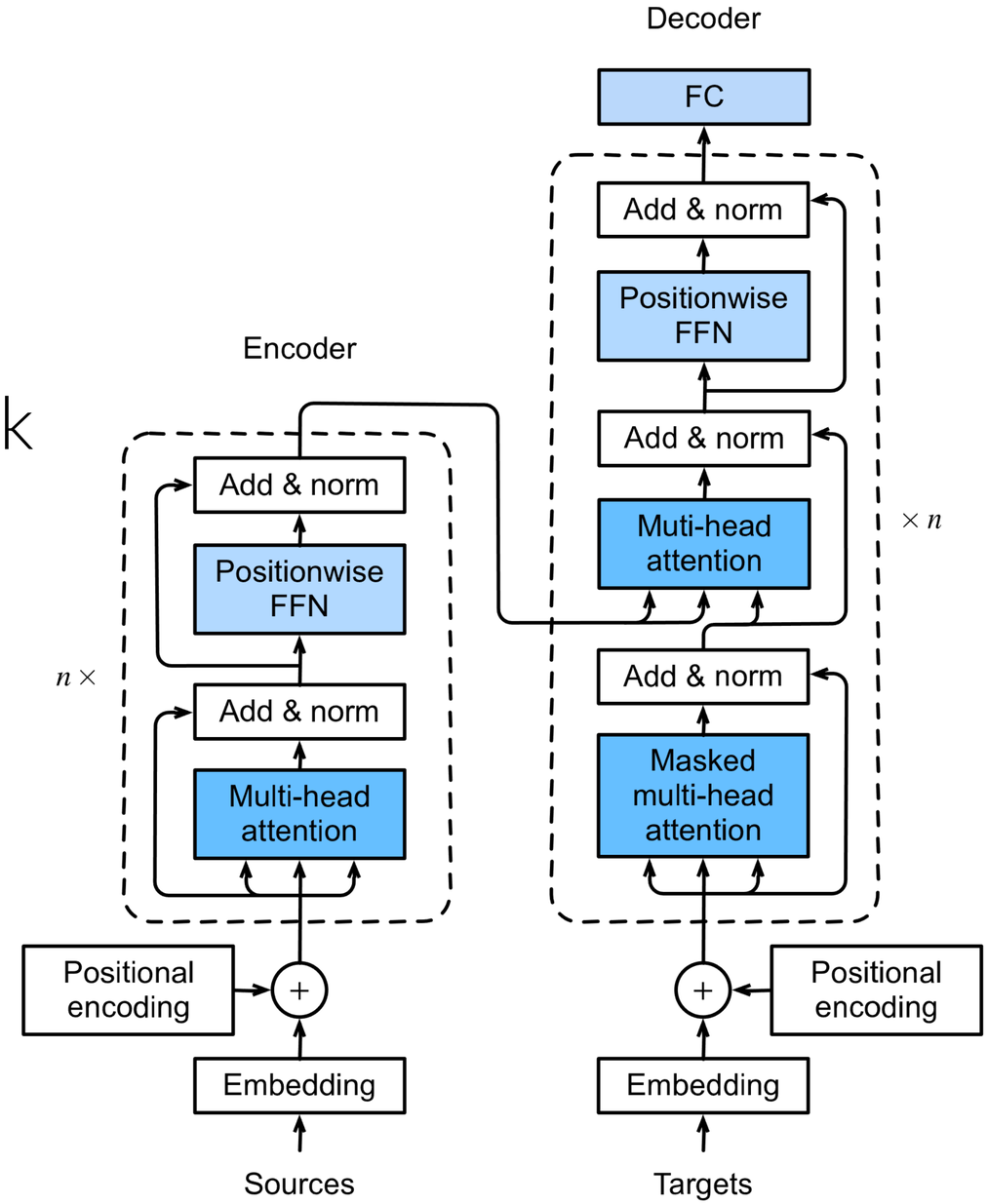


$$\underset{\text{positional encoding}}{\mathbf{x}_i \rightarrow \mathbf{P}(\mathbf{x}_i)} = \begin{cases} x_j + \sin\left(\dfrac{i}{10000^{j/d}}\right) & j : \text{even} \\ x_j + \cos\left(\dfrac{i}{10000^{(j-1)/d}}\right) & j : \text{odd} \end{cases}$$

Or you can design a different type of positional encoding depending on your machine learning tasks → *very active* research area!

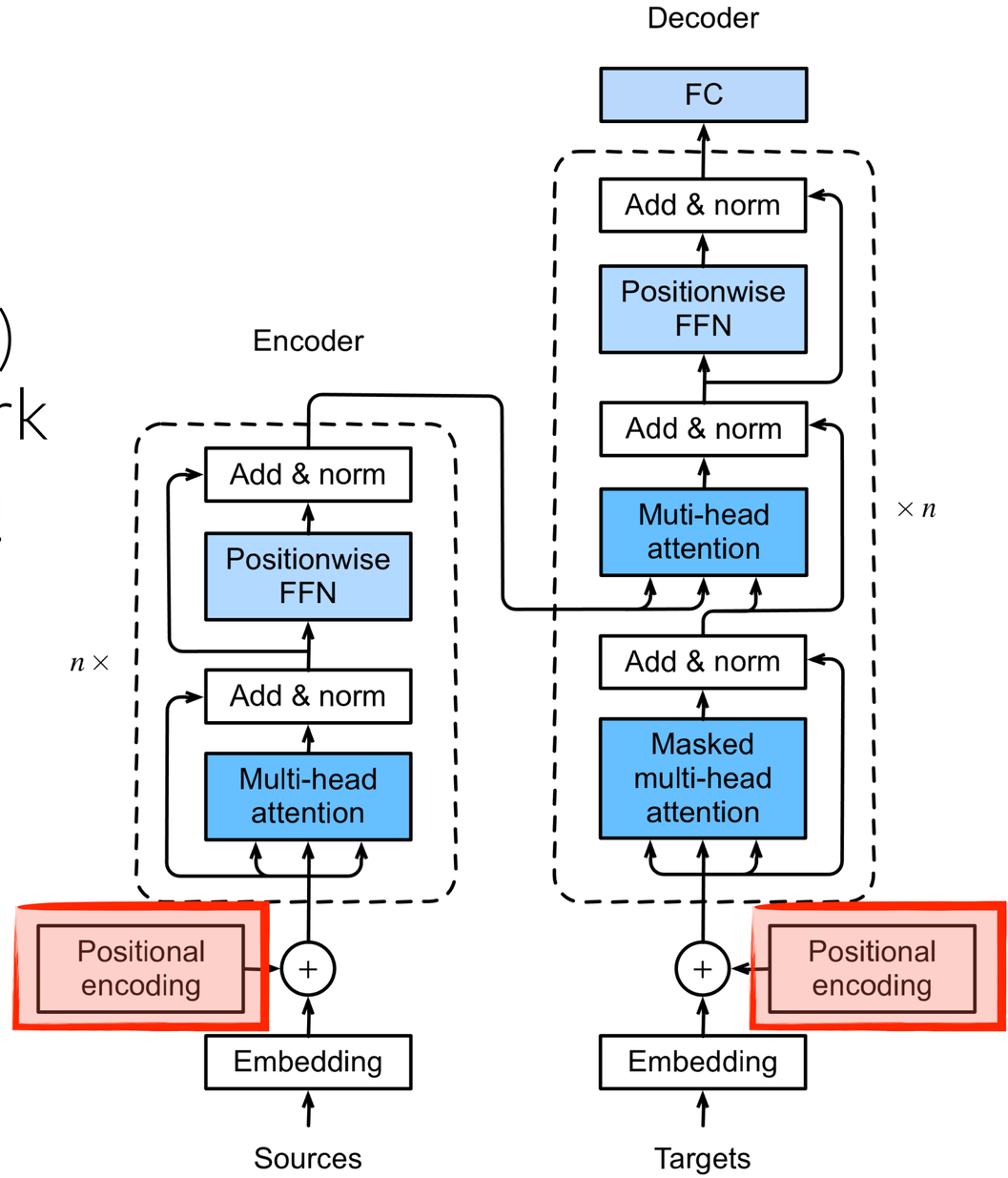
Transformer

- Transformer (Vaswani et al., NeurIPS 2017) uses attention modules for encoder network
  - neither encoder nor decoder has RNNs!



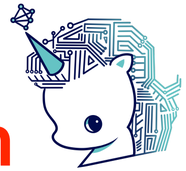
Transformer

- Transformer (Vaswani et al., NeurIPS 2017) uses attention modules for encoder network
  - neither encoder nor decoder has RNNs!
- Transformer has four components
  - **positional encoding**
  - multi-head self-attention
  - residual connection + layer norm
  - positionwise feedforward network

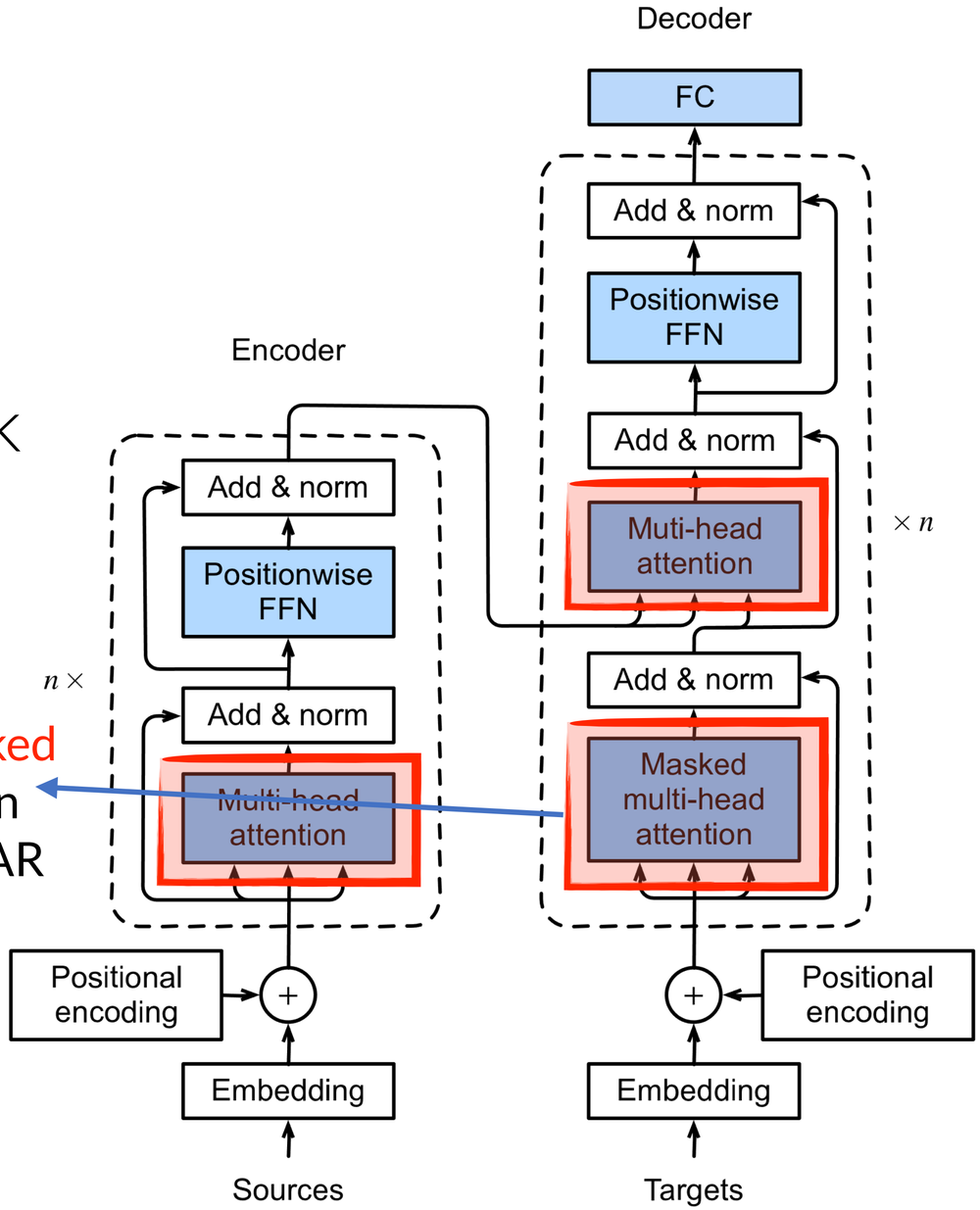


Transformer

- Transformer (Vaswani et al., NeurIPS 2017) uses attention modules for encoder network
  - neither encoder nor decoder has RNNs!
- Transformer has four components
  - positional encoding
  - **multi-head self-attention**
  - residual connection + layer norm
  - positionwise feedforward network

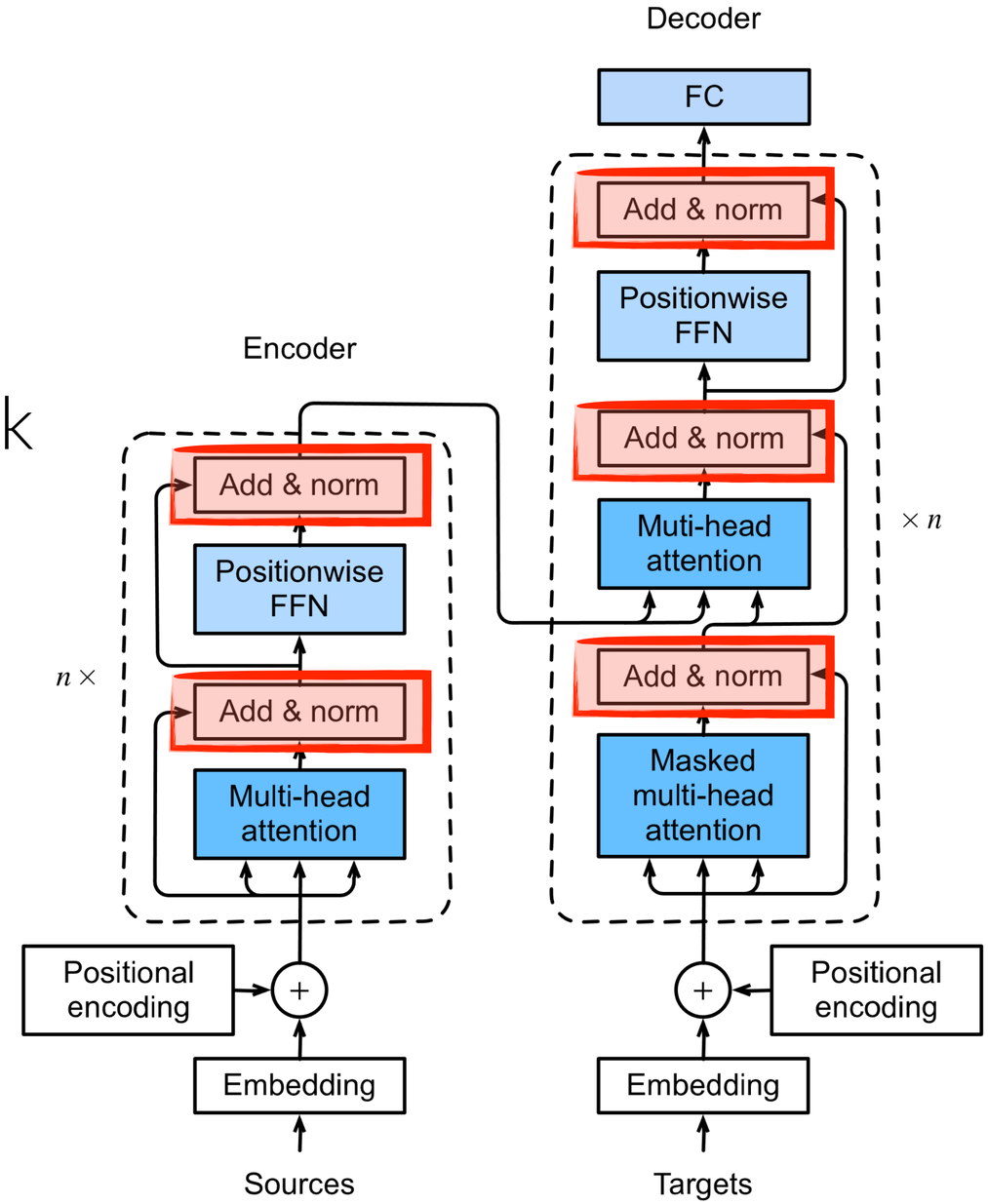


We use masked self-attention to perserve AR



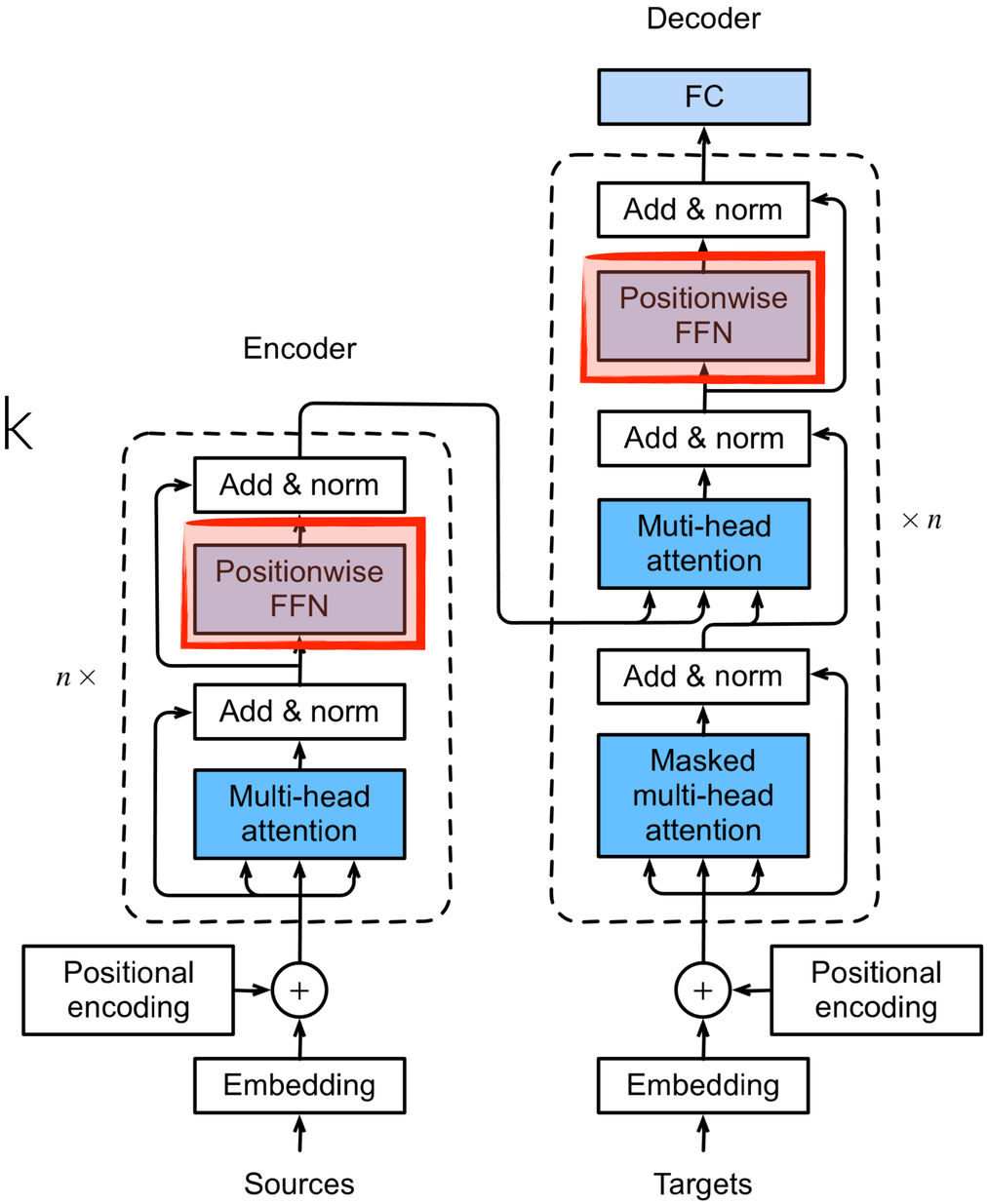
Transformer

- Transformer (Vaswani et al., NeurIPS 2017) uses attention modules for encoder network
  - neither encoder nor decoder has RNNs!
- Transformer has four components
  - positional encoding
  - multi-head self-attention
  - **residual connection + layer norm**
  - positionwise feedforward network



Transformer

- Transformer (Vaswani et al., NeurIPS 2017) uses attention modules for encoder network
  - neither encoder nor decoder has RNNs!
- Transformer has four components
  - positional encoding
  - multi-head self-attention
  - residual connection + layer norm
  - **positionwise feedforward network**



Q&A