From Shallow to Deep Language Representations

1 Basics · 2 Shallow Models · **3 Transformers** · 4 BERT

KDD'19 Anchorage

Aston Zhang, Haibin Lin, Leonard Lausen, Sheng Zha, Alex Smola

[d2l.ai](d2l.ai) gluon-nlp.mxnet.io

Outline (Transformer)

- Encoder-Decoder Architecture
- Attention
- Transformer Architecture
- Sequence Sampling

Encoder-Decoder

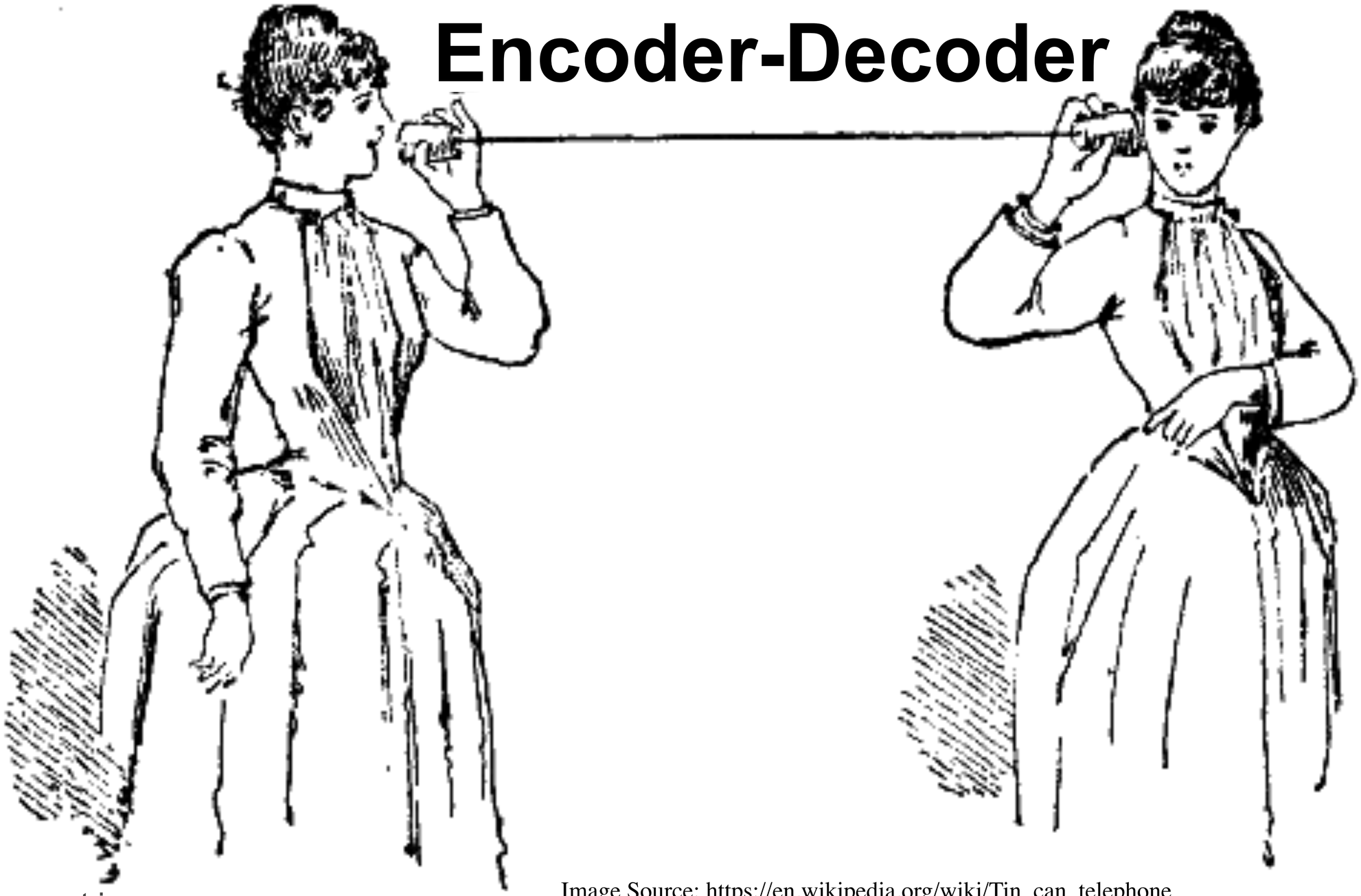
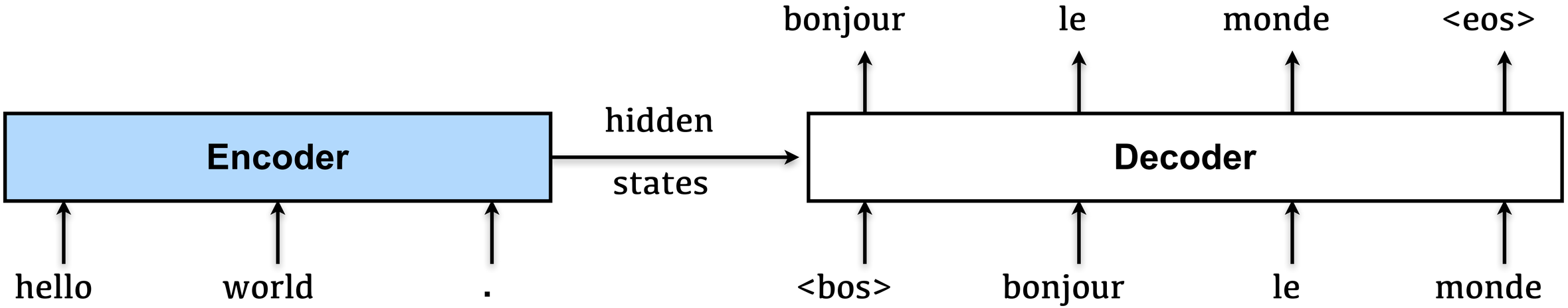


Image Source: https://en.wikipedia.org/wiki/Tin_can_telephone

Encoder-Decoder for Machine Translation

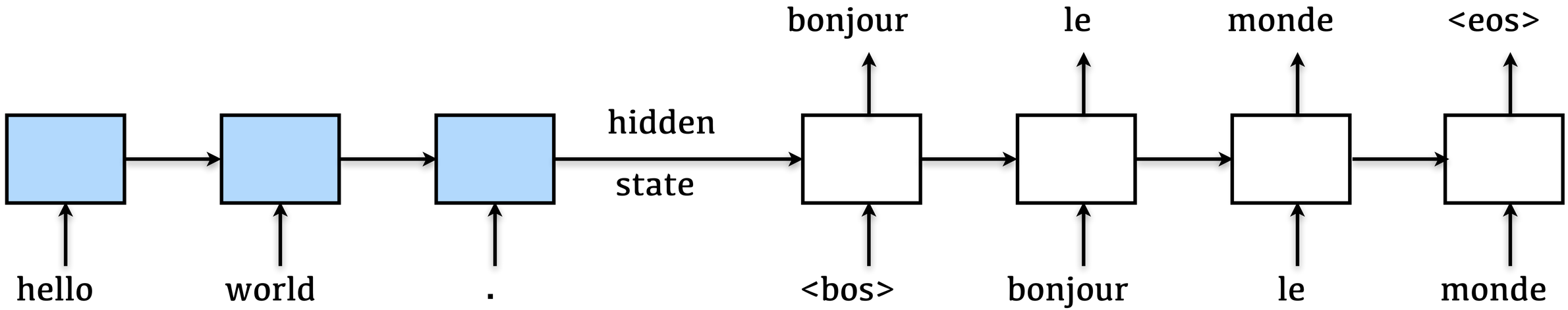
‘le’ = Decoder(‘bonjour’, ‘<bos>’, h)



h = Encoder(‘hello’, ‘world’, ‘.’)

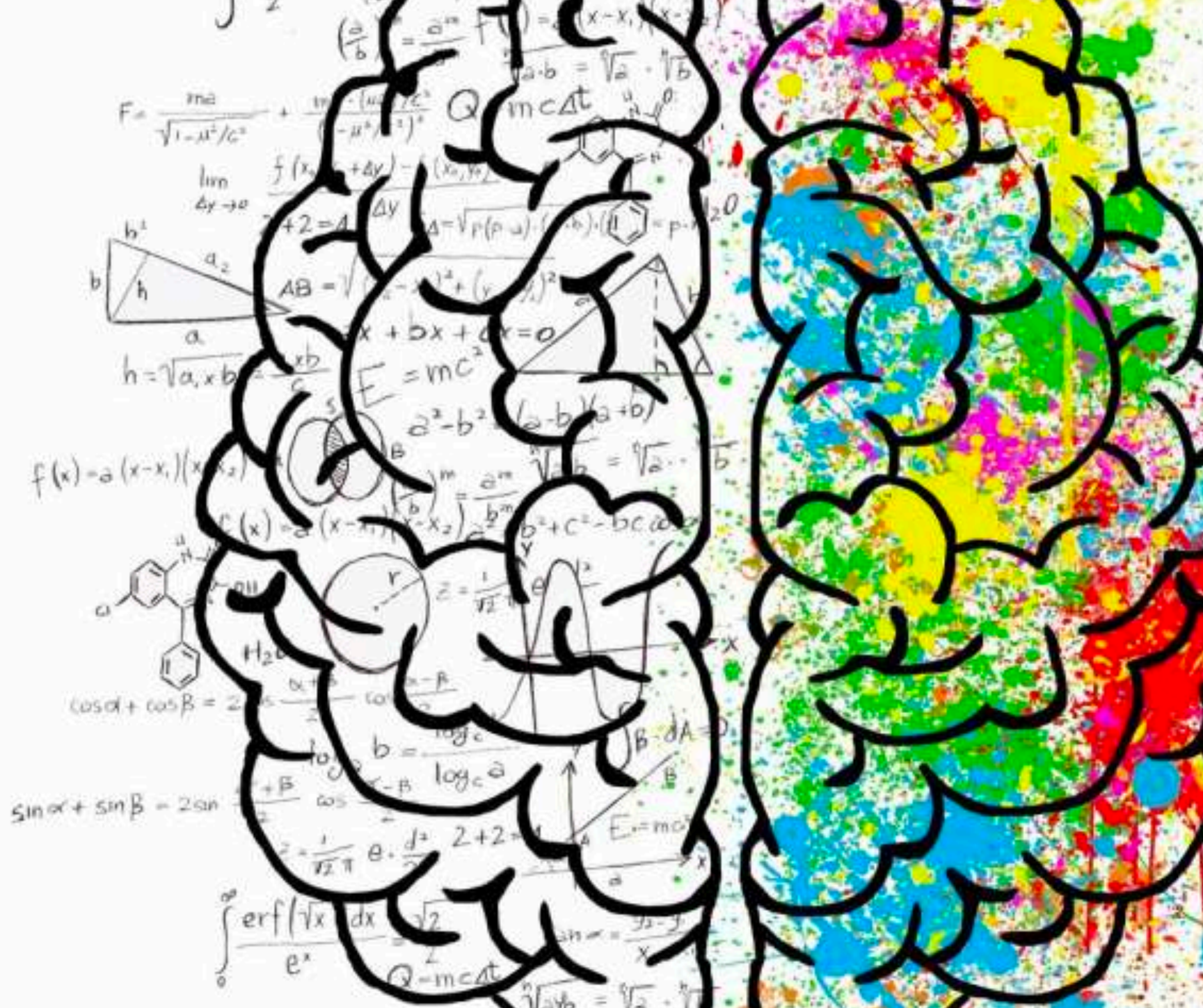
Decoders predict one word at a time.

RNN as Encoder & Decoder



- Source is encoded in a **fixed** dimensional vector
  - Too detailed for short sequence
  - Not enough storage for long sequences (bad translation)
    Impossible to overcome
- We have the original sequence, can't we just look it up?

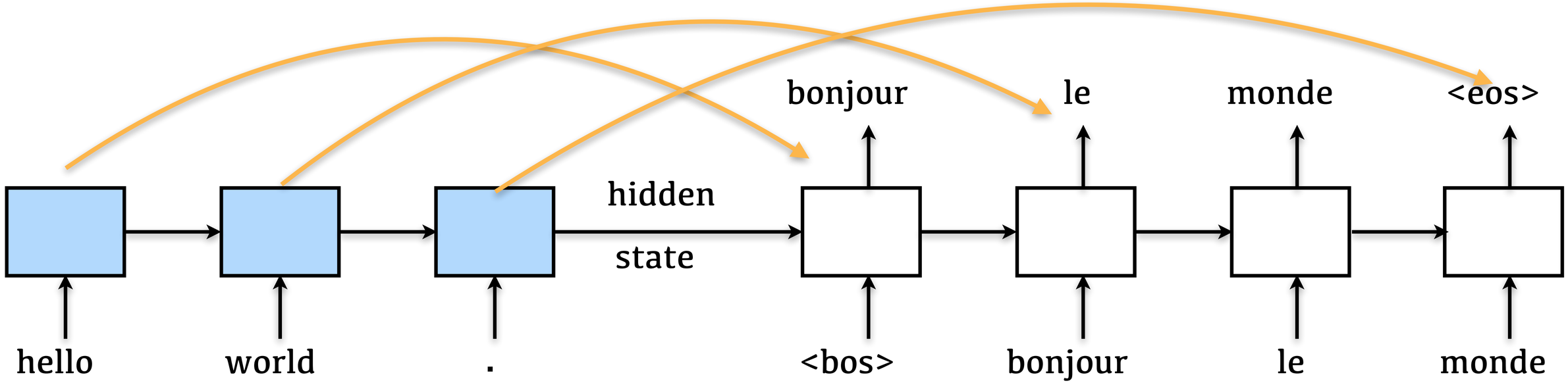
Attention



gluon-nlp.mxnet.io

Motivation

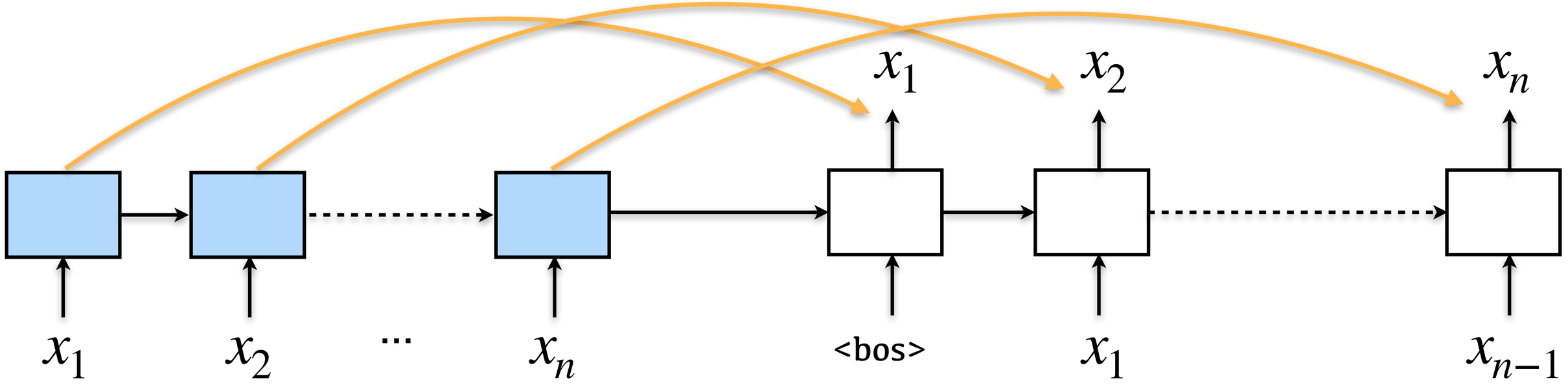
- Output token may be related to specific source tokens



- RNN - use the last encoder state for decoding.
- RNN with attention - Use all states and learn to select

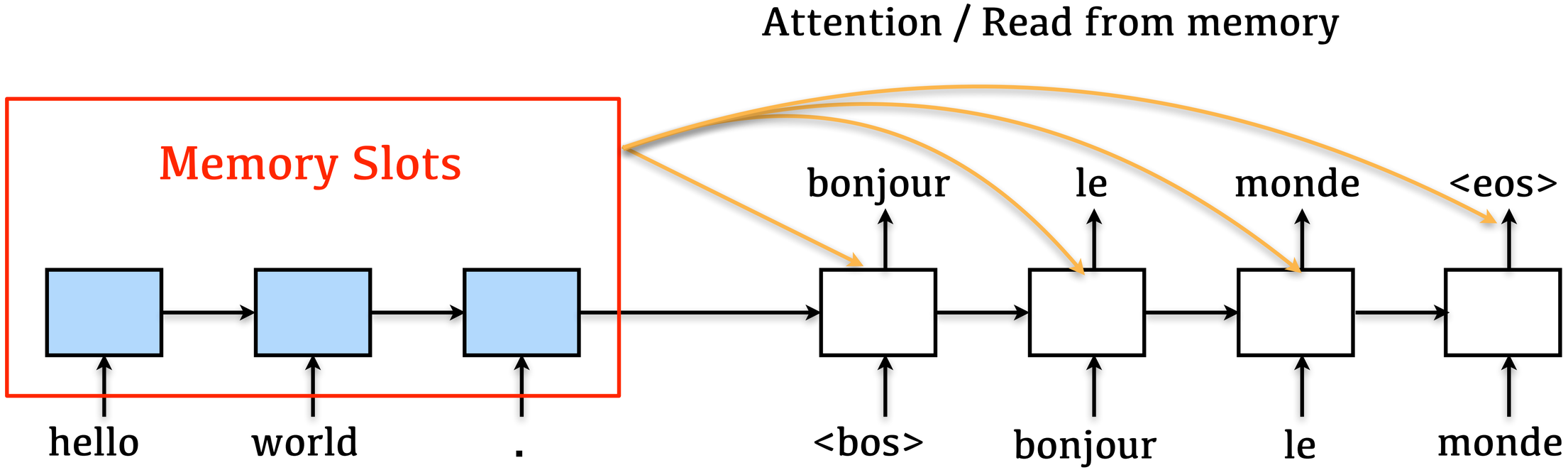
Sequence Copy by Attention

- Sequence Copy with attention is easy
  All states in the encoder are kept in **memory**



$$x_1, x_2, \ldots, x_n \rightarrow h_1, h_2, \ldots, h_n \rightarrow x_1, x_2, \ldots, x_n$$

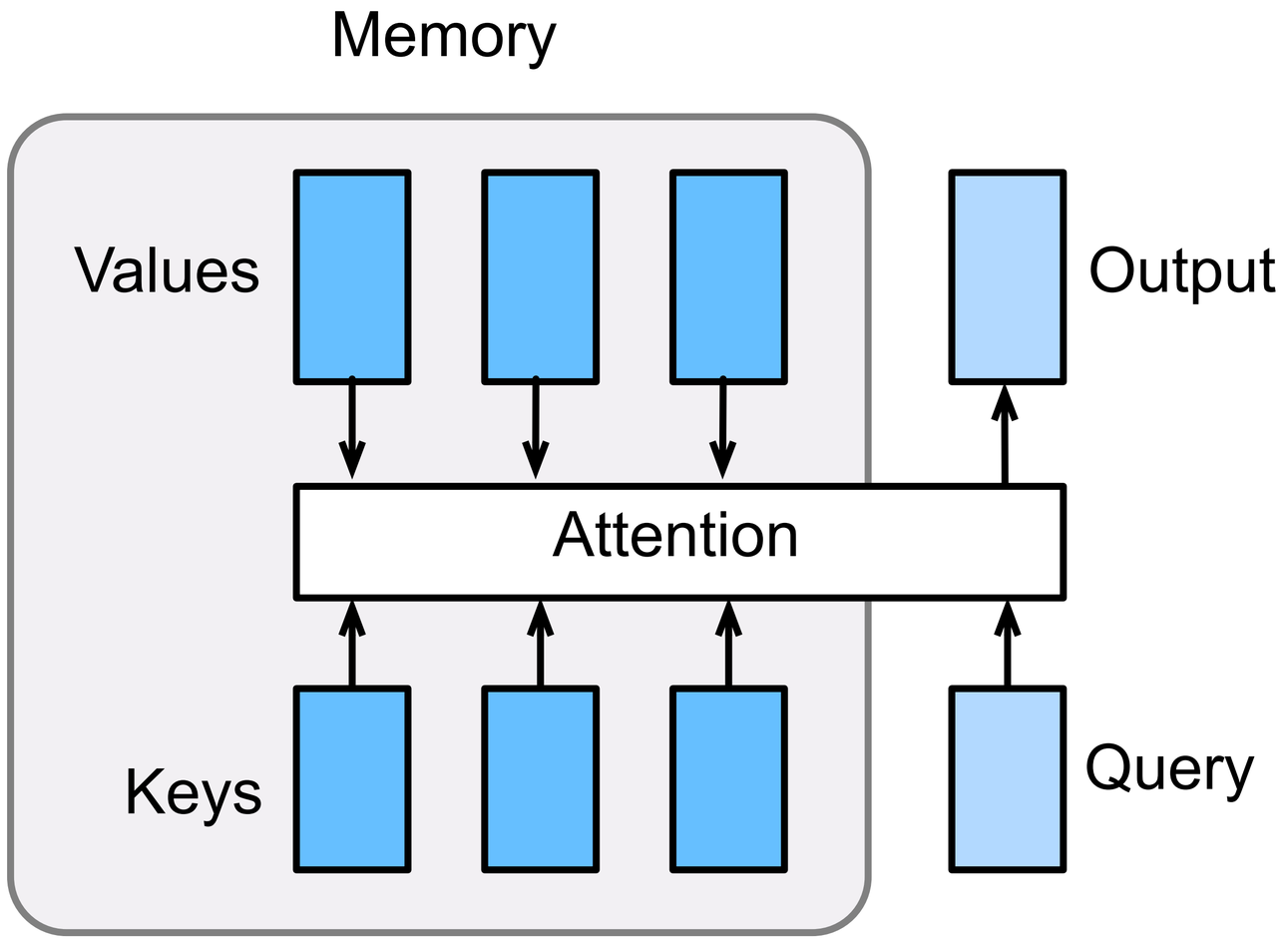
Attention & Memory



- Each state in the RNN can be viewed as memory slots
- Query the memory for related states at every step in the decoding process.

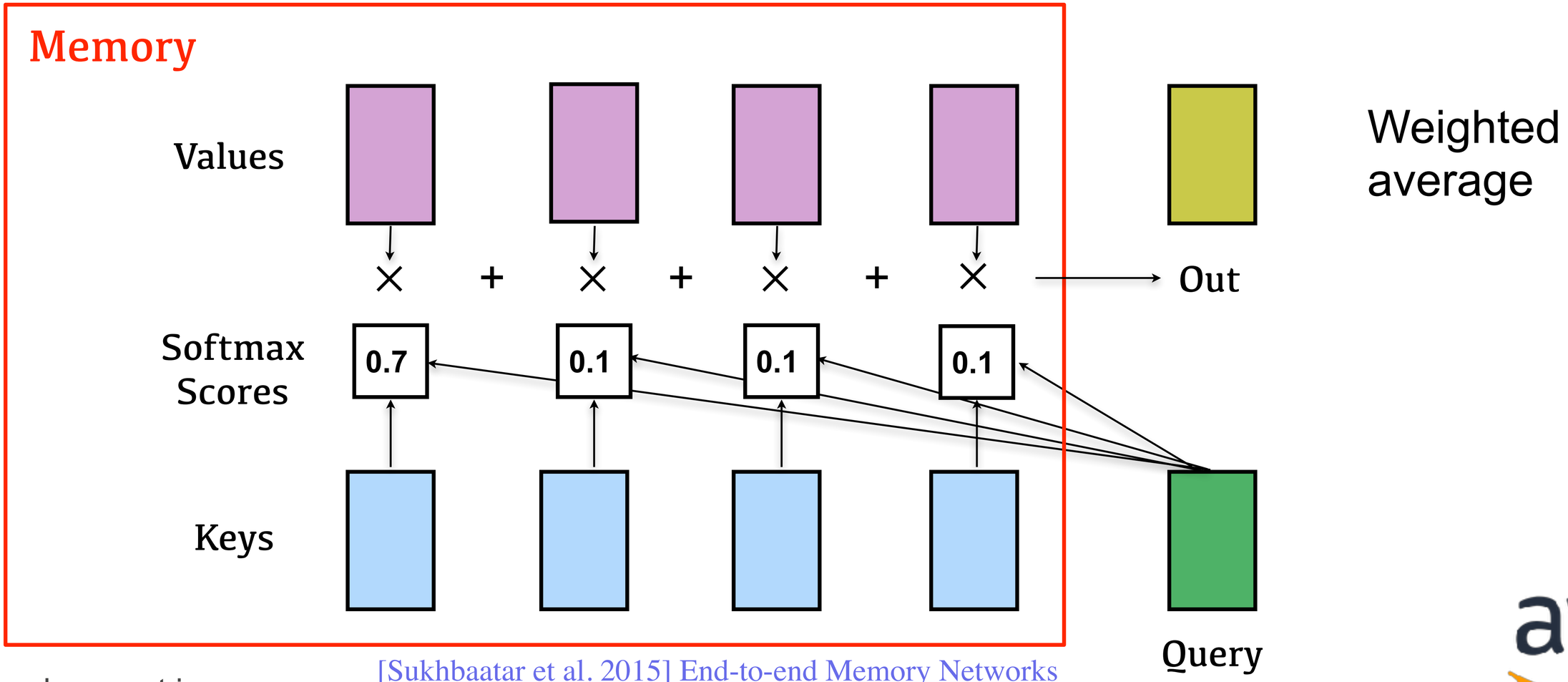
Interpretation — Key-Value Memory Network

- Attention is a key-value memory structure
  - Memory consists of key-value pairs.
  - Fetch values based on respective query-key similarity (weighted average of values).



Interpretation — Key-Value Memory Network

Out = Attention(Query, Key, Value)

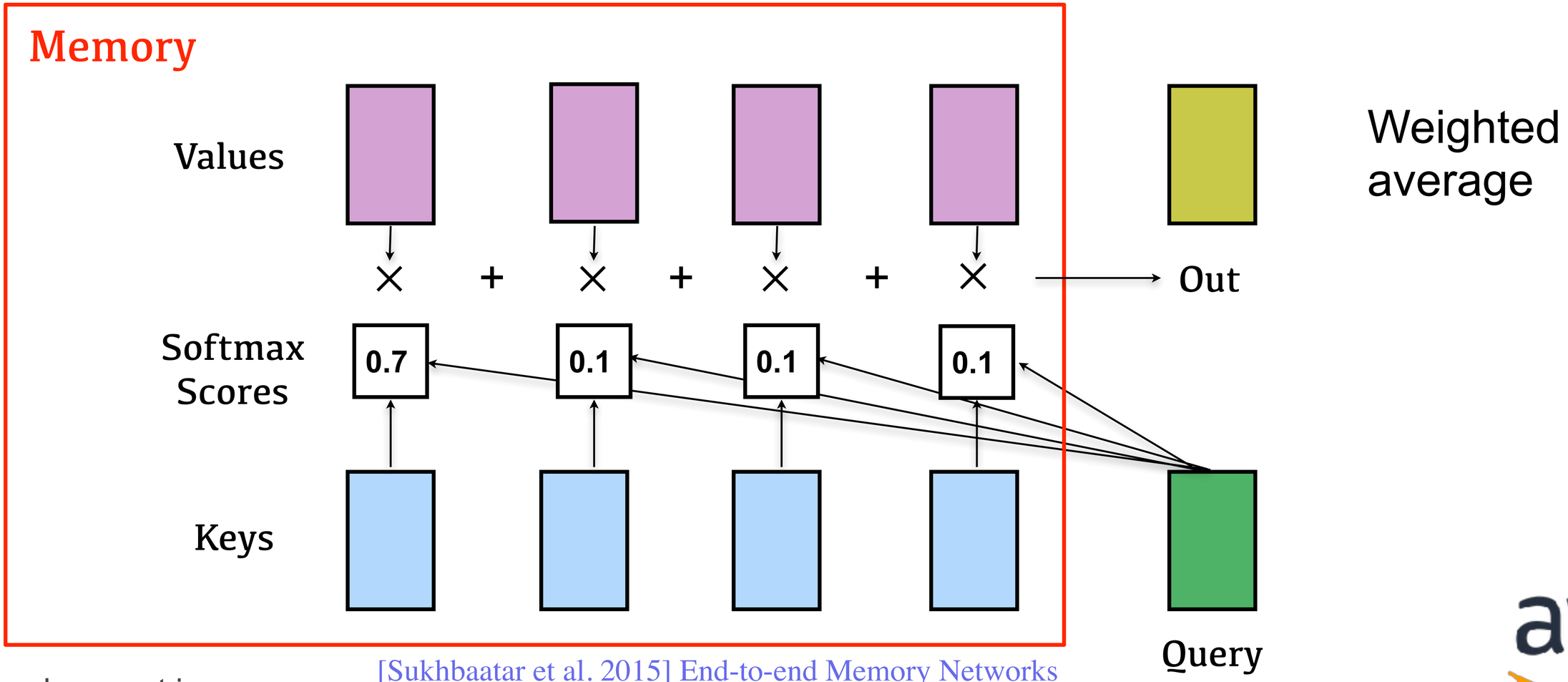


[Sukhbaatar et al. 2015] End-to-end Memory Networks

Interpretation — Key-Value Memory Network

Out = Attention(Query, Key, Value)

Out = Attention(h_i^{dec}, $f(H^{enc})$, $g(H^{enc})$)

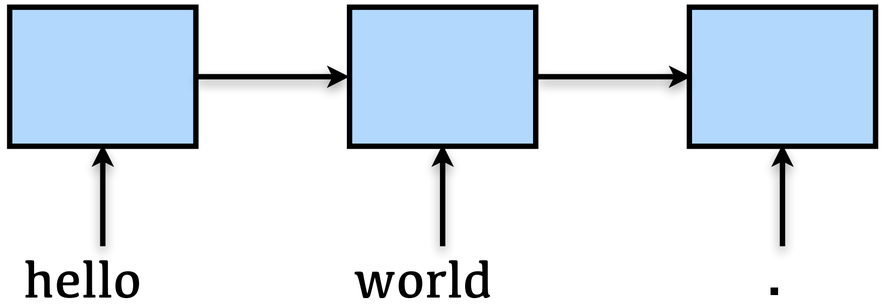


[Sukhbaatar et al. 2015] End-to-end Memory Networks


TRANS
FORMERS

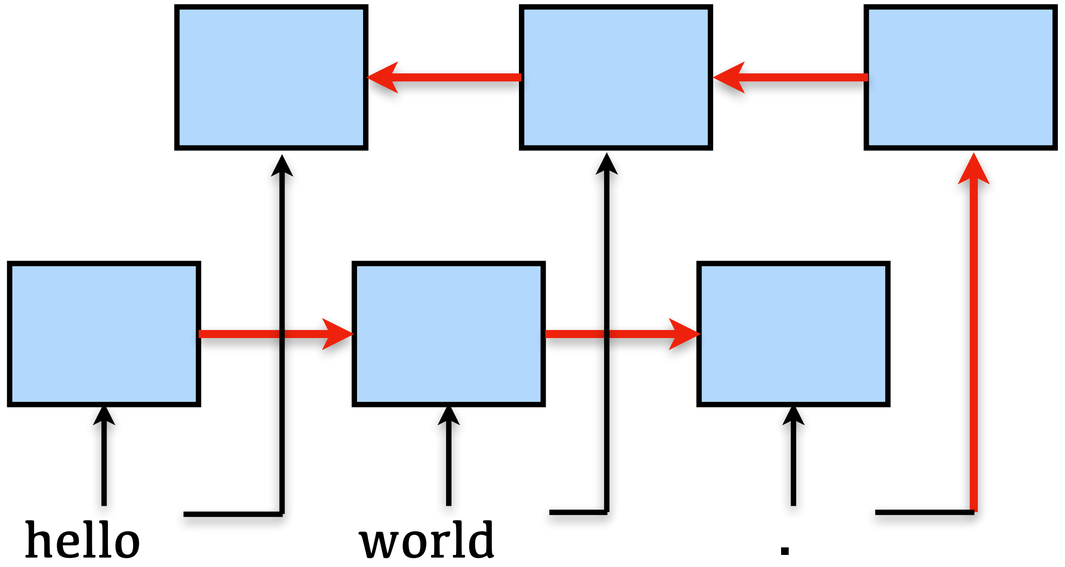
Recurrence is expensive

RNN



Sequential dependencies are bad for parallel computation

Bidirectional RNN



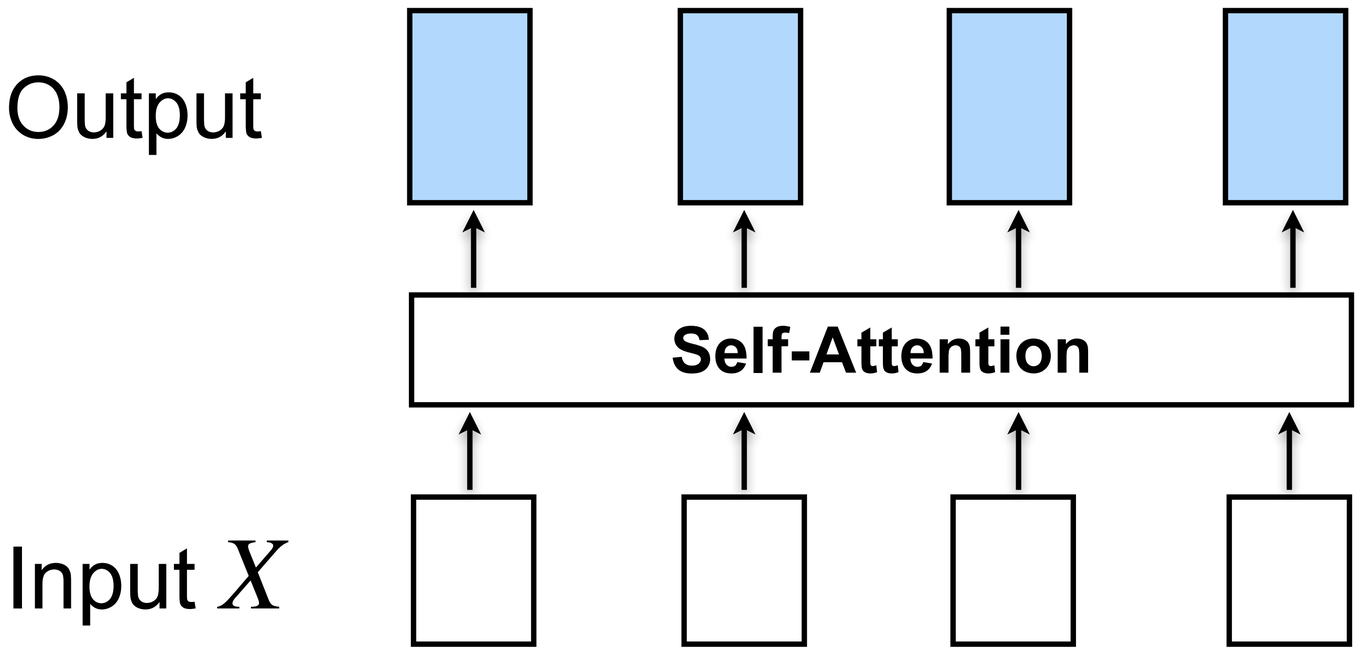
gluon-nlp.mxnet.io

aws

(Self) Attention is all you need

$$\text{out} = \text{attention}(\text{query}, \text{key}, \text{value})$$
$$= \text{attention}(W_q X, W_k X, W_v X)$$

self-attention

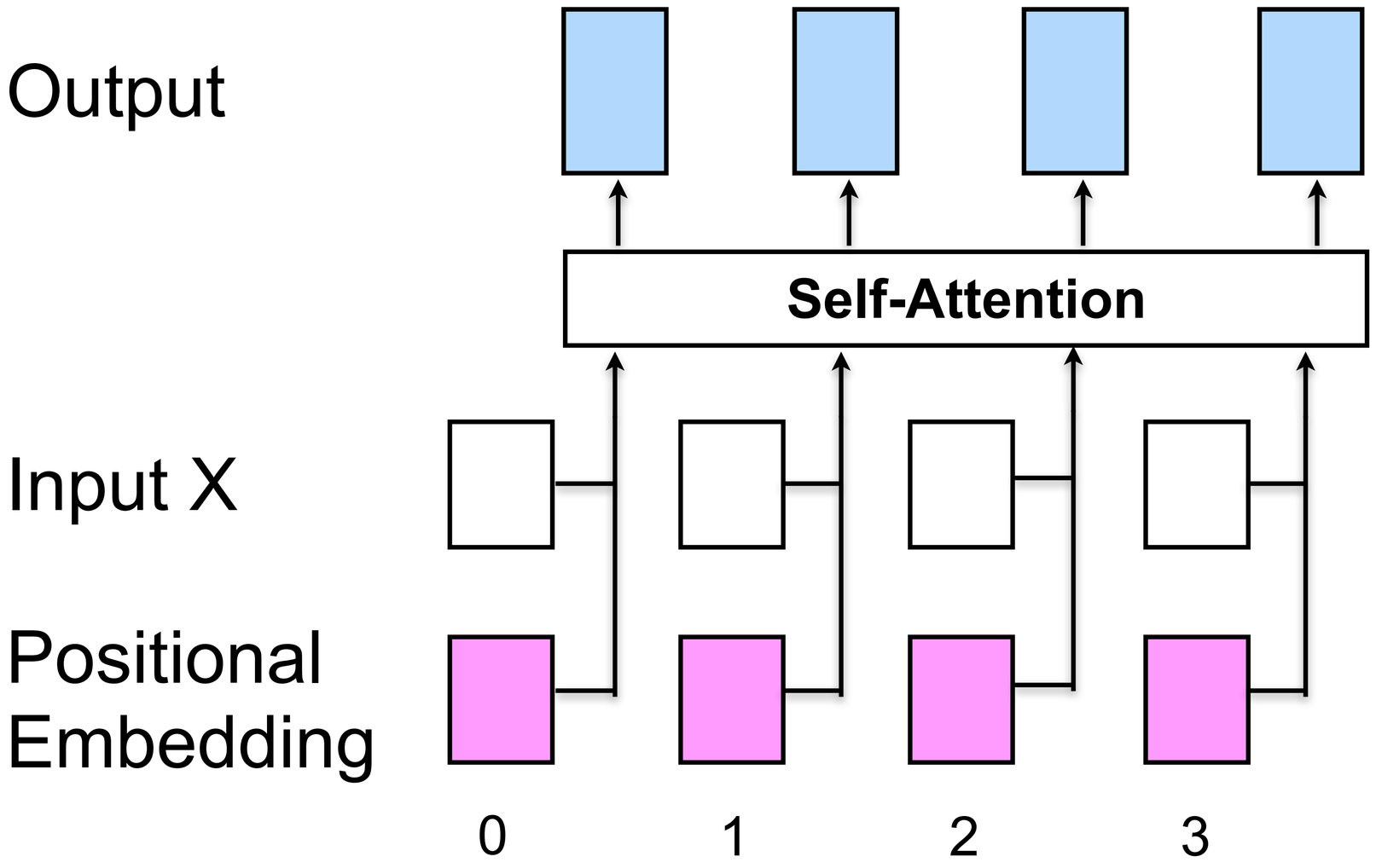


Permutational invariant?
Positional encoding

Attention in the encoder.
Run in parallel.

[Vaswani et al., 2017] Attention is all you need

Positional Encoding

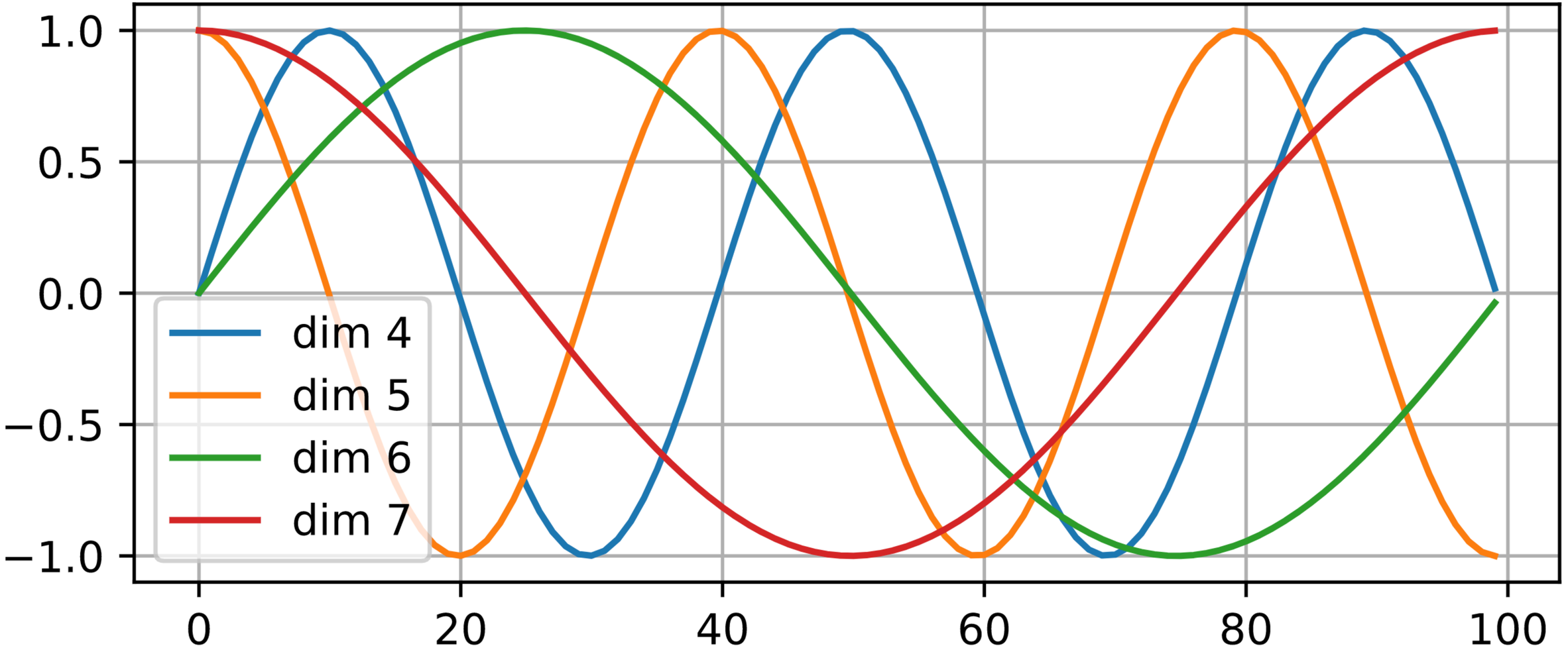


Options

- Learn positional embeddings
- Use Fourier features

gluon-nlp.mxnet.io

Fourier Features



gluon-nlp.mxnet.io

aws

Multi-head Attention

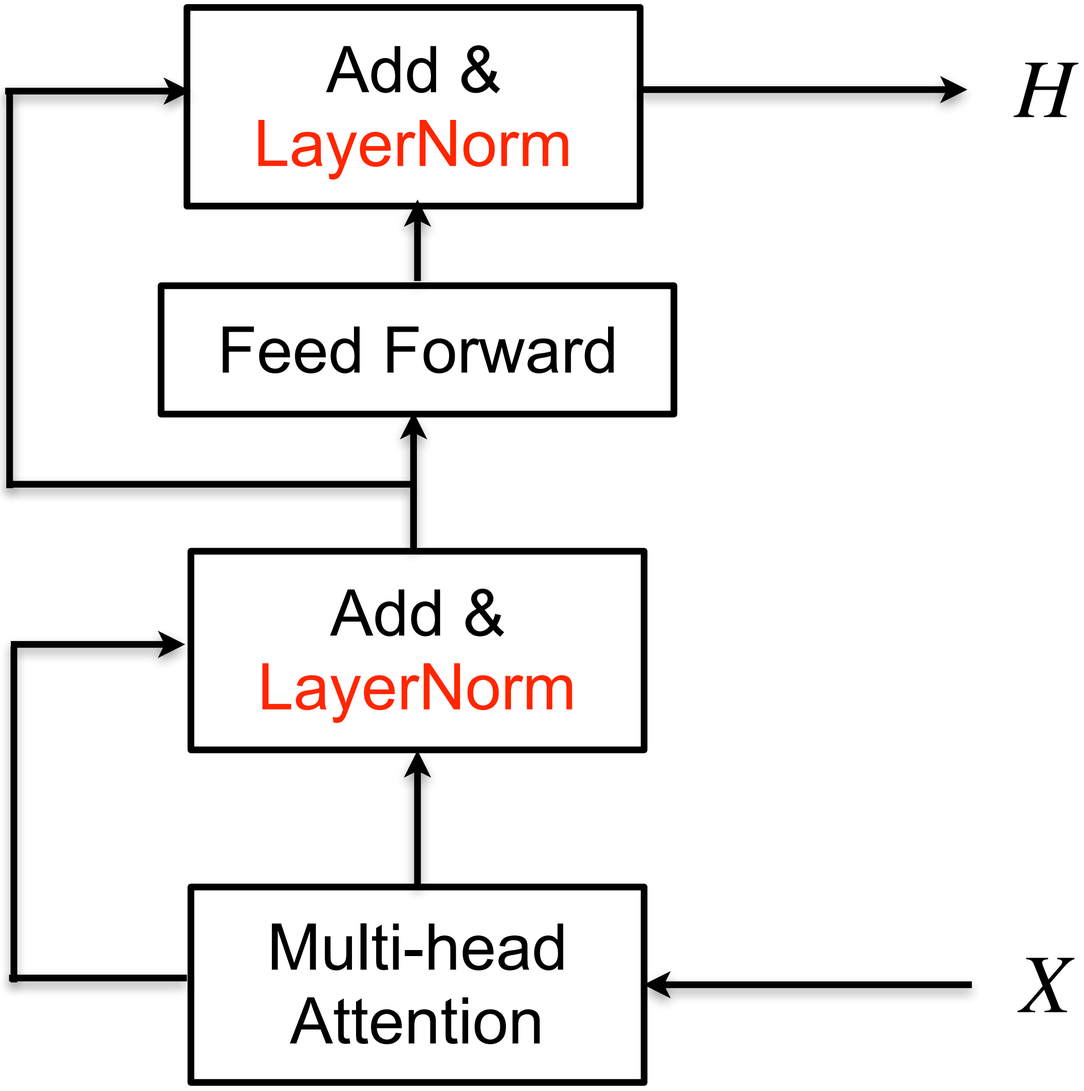
- Increase representation power of attention
  - One head - similarity in one subspace WX
  - Multiple heads - multiple subspaces

$$W_o \begin{bmatrix} \text{Attention}(W_q^1 X,\ W_k^1 X,\ W_v^1 X) \\ \text{Attention}(W_q^2 X,\ W_k^2 X,\ W_v^2 X) \\ \dots \\ \text{Attention}(W_q^K X,\ W_k^K X,\ W_v^K X) \end{bmatrix}$$

multi-head attention

gluon-nlp.mxnet.io

Transformer Block



- Multi-head Attention
- Residual Network

$$\mathbf{x} \rightarrow \mathbf{x} + f(\mathbf{x})$$

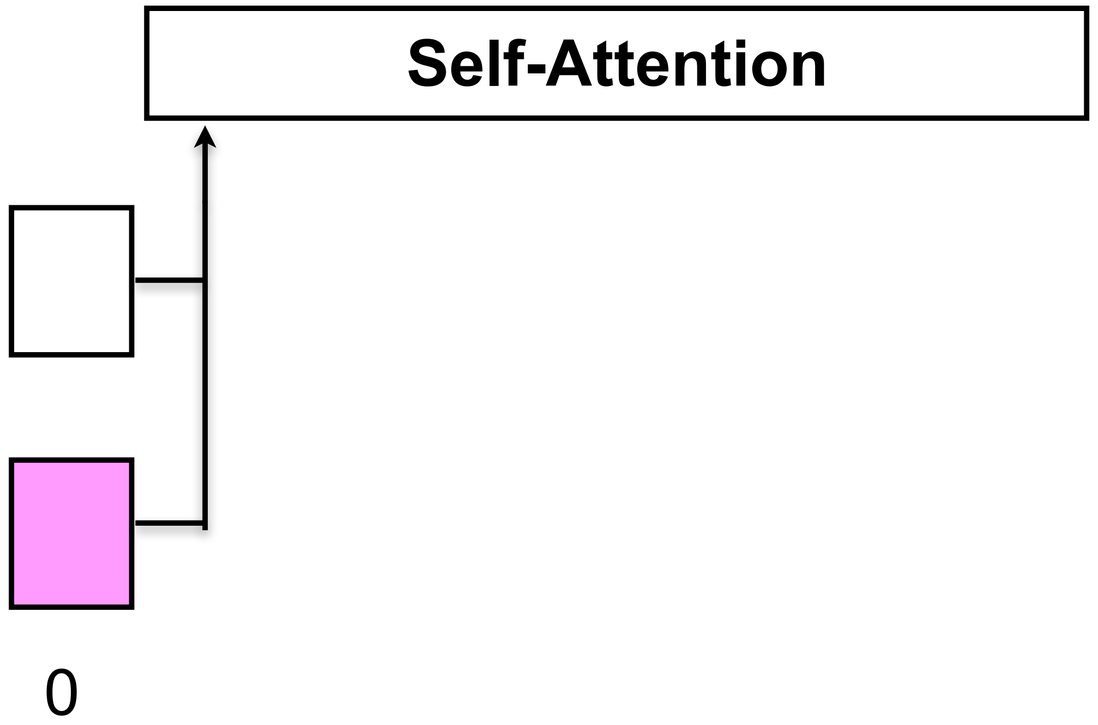
- Layer Normalization

$$\mathbf{x} \rightarrow \alpha \cdot \frac{\mathbf{x} - \mu[\mathbf{x}]}{\mathrm{std}[\mathbf{x}]} + \beta$$

gluon-nlp.mxnet.io

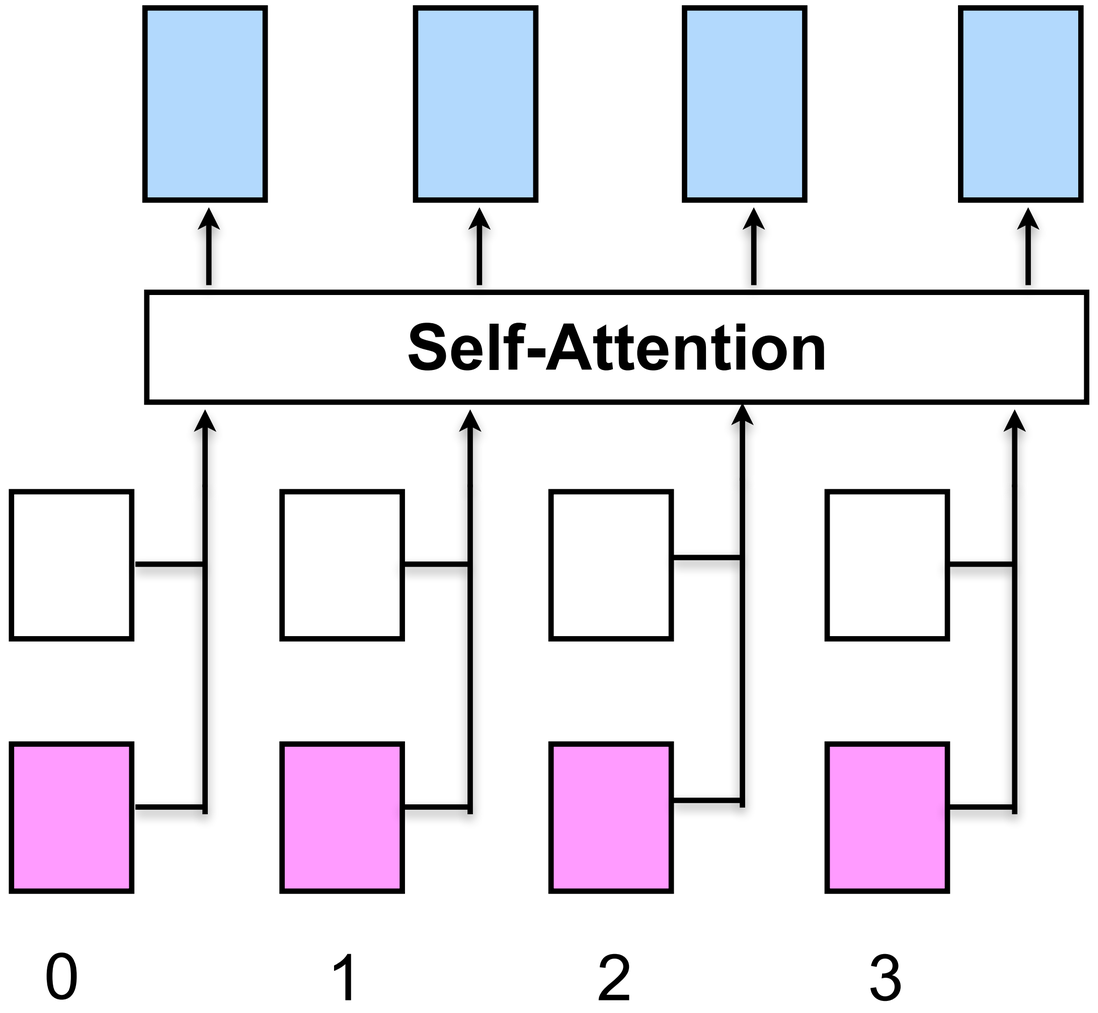
Causal Transformer

- Decoding needs to rely on data generated so far
- Block out all ‘future’ data (to avoid information leak)

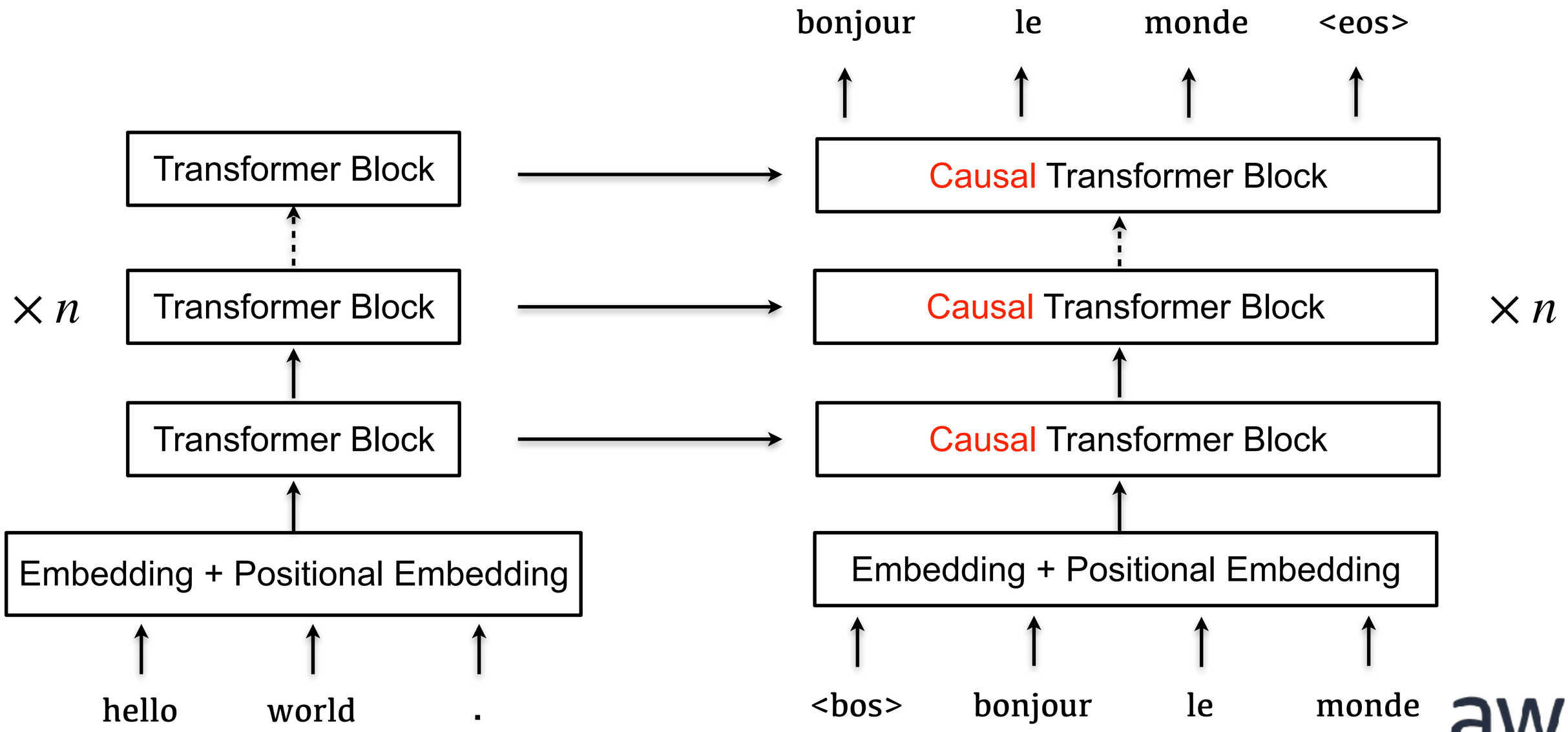


Causal Transformer

- Decoding needs to rely on data generated so far
- Block out all ‘future’ data (to avoid information leak)

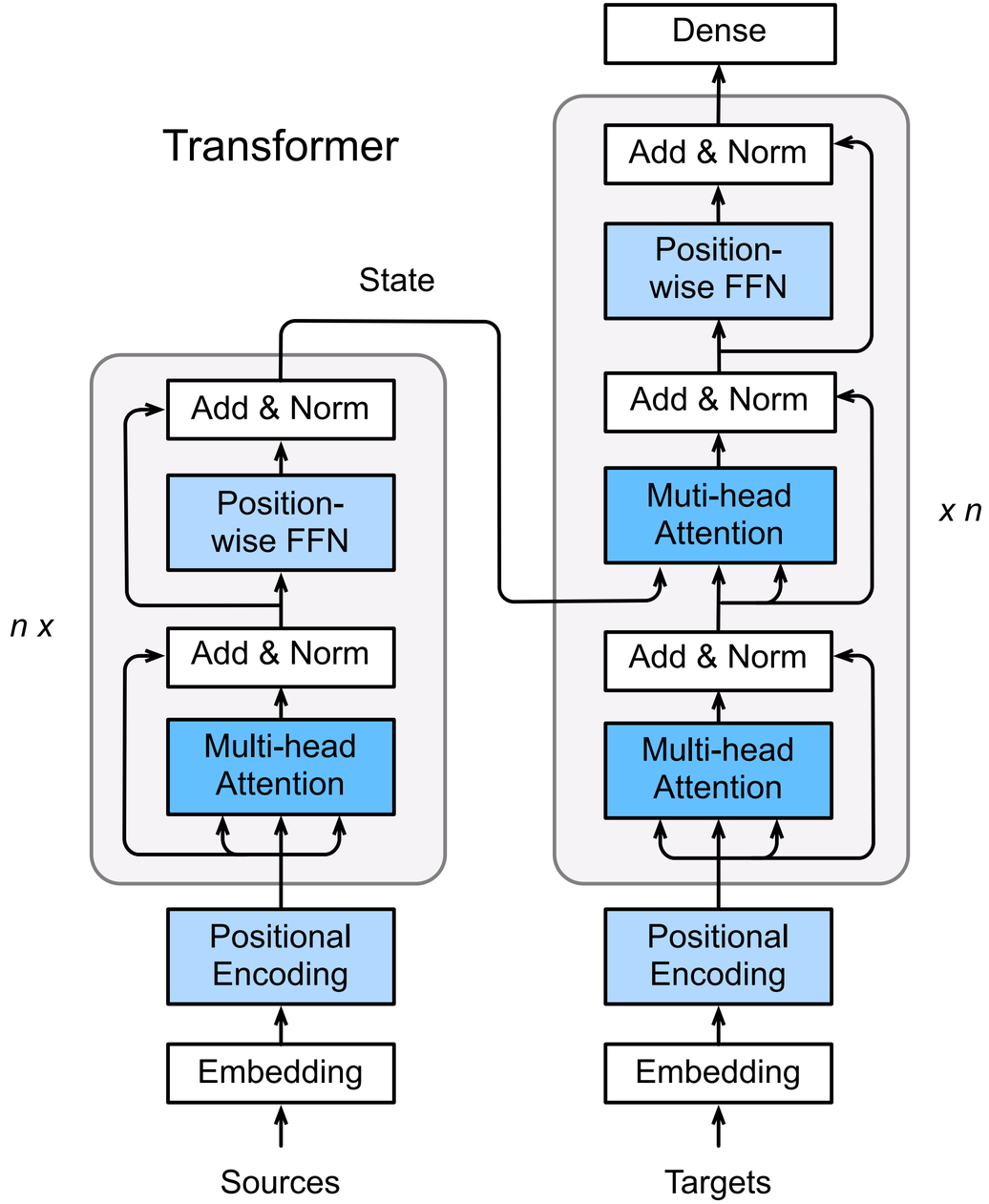


Transformer Architecture for Machine Translation



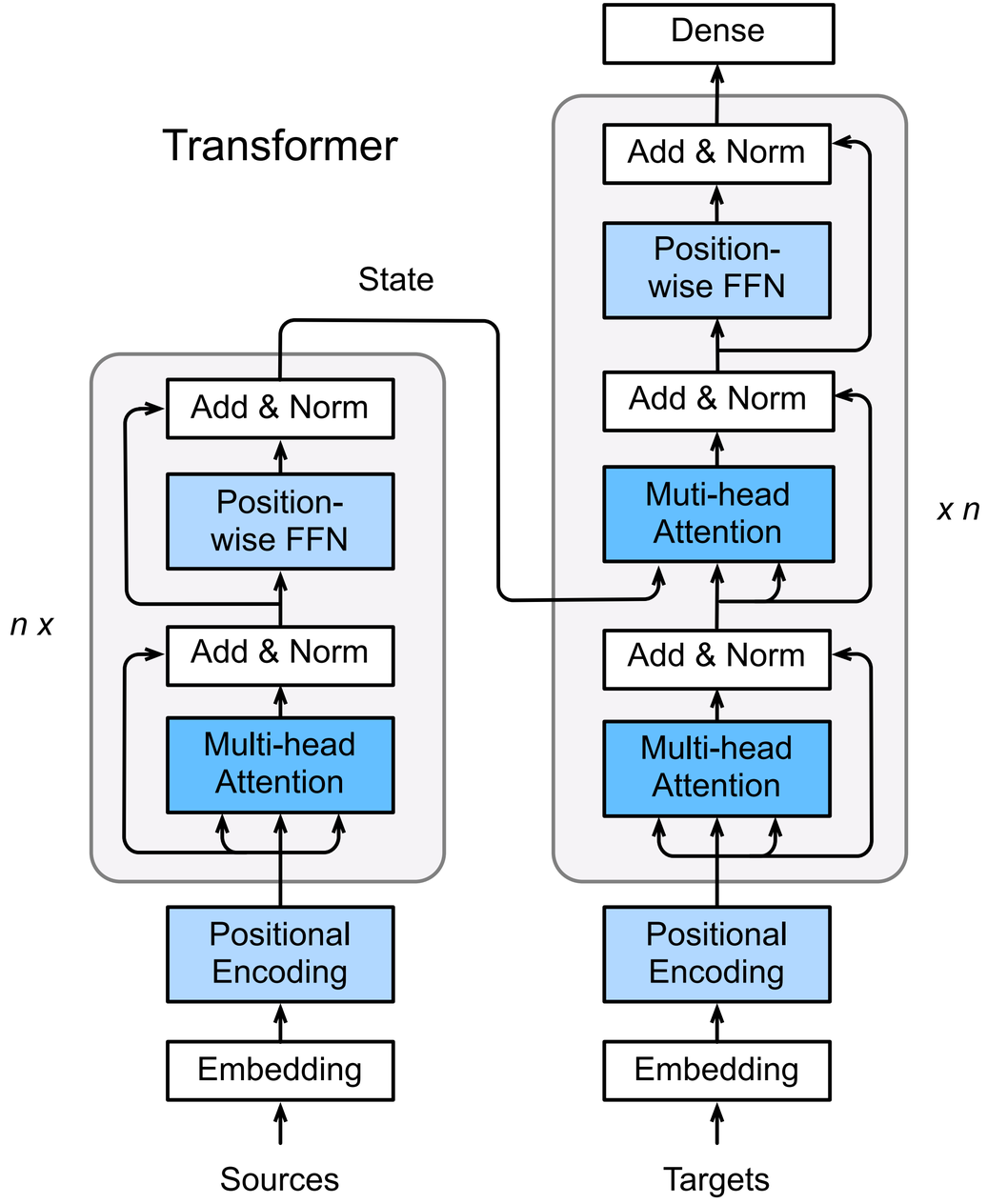
gluon-nlp.mxnet.io

Transformer Encoder-Decoder Architecture



Transformer Encoder-Decoder Architecture

Transformer block

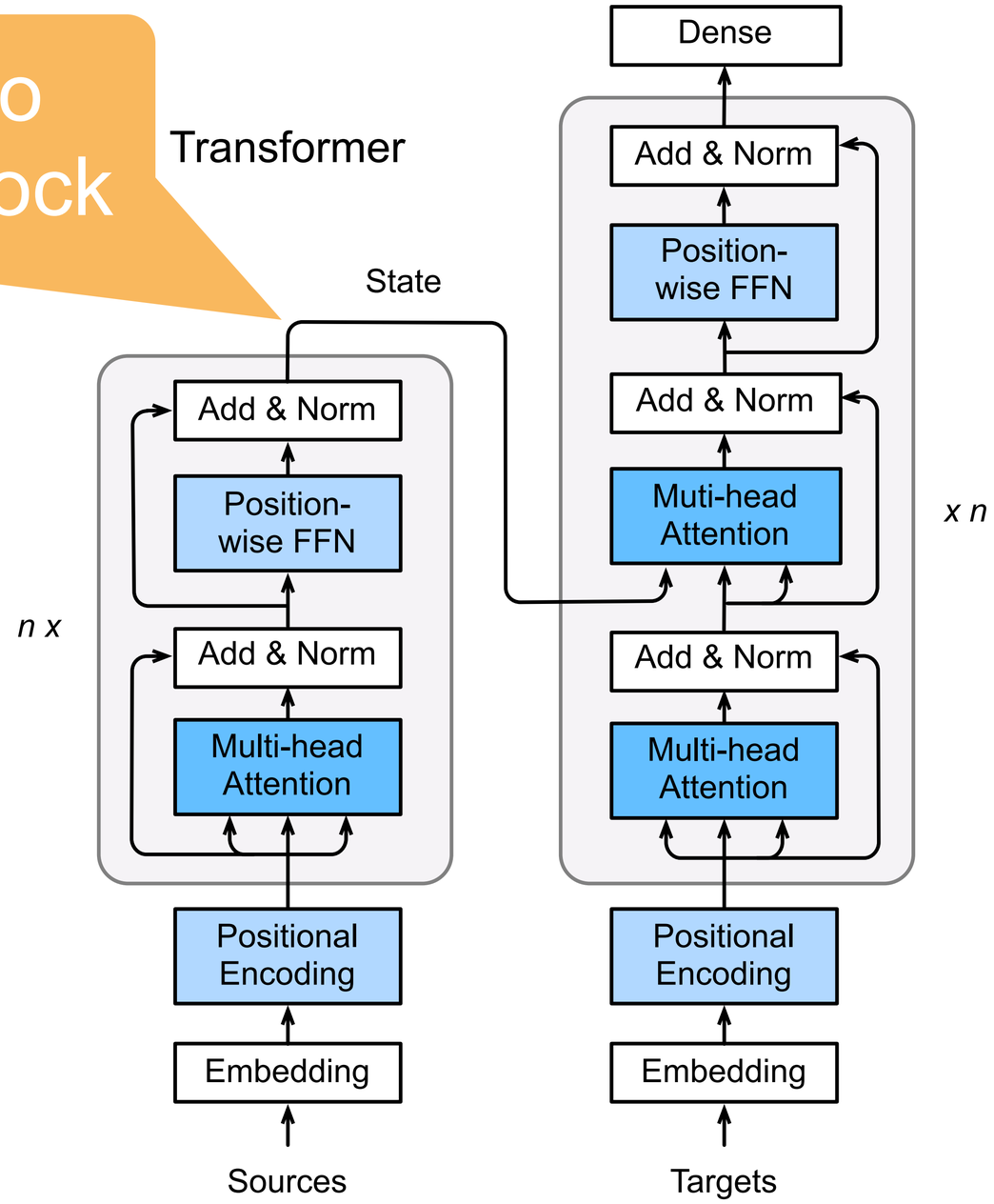


gluon-nlp.mxnet.io

Transformer Encoder-Decoder Architecture

Pass to every block

Transformer block



gluon-nlp.mxnet.io

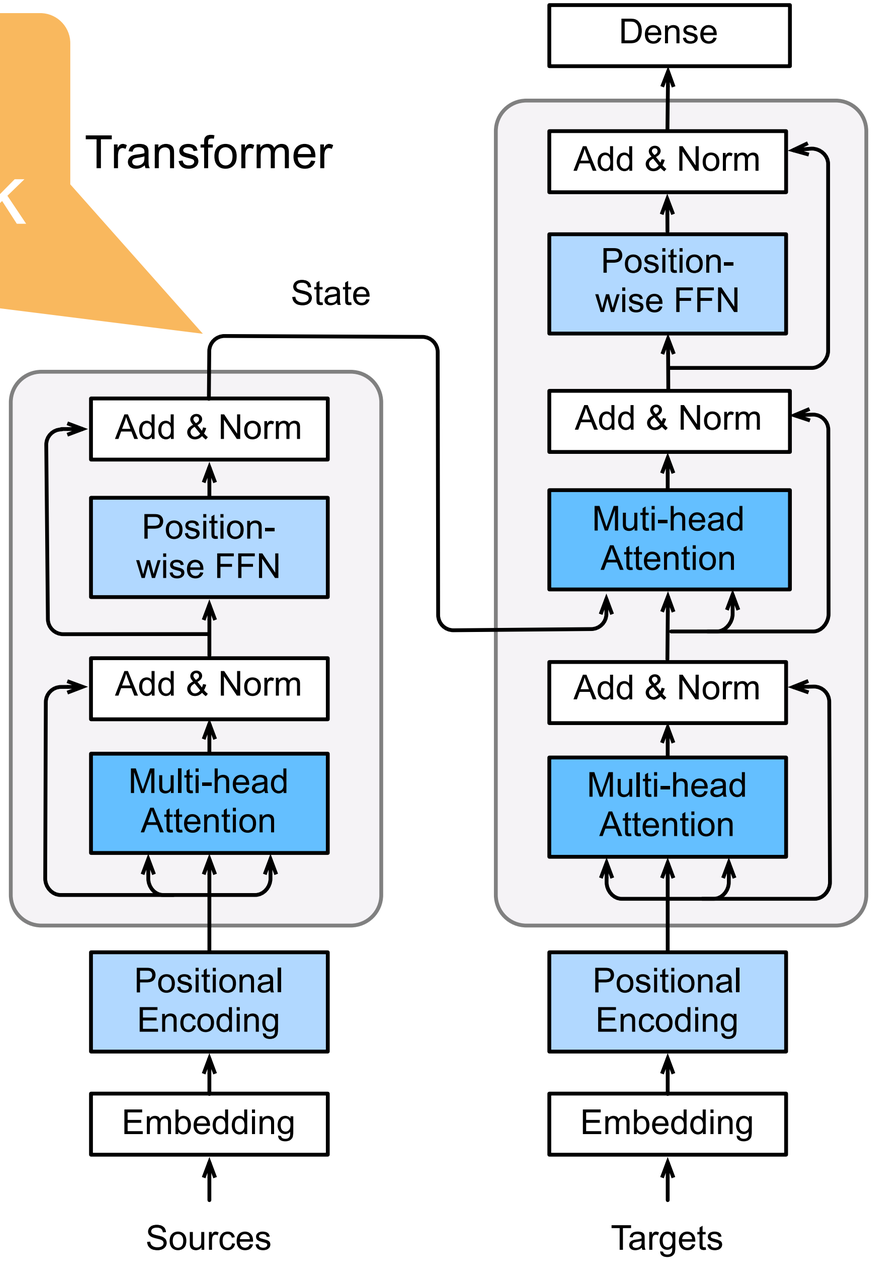
aws

Transformer Encoder-Decoder Architecture

Pass to every block

Transformer block

Sequential information




Sequence
Sampling

How to Predict with a Machine Translation Model?

- Decoder returns single-step prediction $p\left(y_i \mid y_{1:i-1}, x_{1:n}\right)$
  - Easy to find one step ahead
  - Multiple steps ahead might not be one-step optimal due to dependence of terms

$$\arg\max_{y_{1:m}} \log p(y_{1:m} \mid x_{1:n})$$

$$= \arg\max_{y_{1:m}} \sum_{i=1}^{m} \log p(y_i \mid y_{1:i-1}, x_{1:n})$$

Search-based Algorithms for Inference

$$\arg\max_{y_{1:m}} \log p(y_{1:m} \mid x_{1:n})$$

gluon-nlp.mxnet.io

Search-based Algorithms for Inference

$$\arg\max_{y_{1:m}} \log p(y_{1:m} \mid x_{1:n}) \xrightarrow{\text{greedy}} \arg\max_{y_i} \log p\left(y_i \mid y_{1:i-1}, x_{1:n}\right)$$

$O(|V| \cdot T)$ cumulative error

gluon-nlp.mxnet.io

Search-based Algorithms for Inference

$$\arg\max_{y_{1:m}} \log p(y_{1:m} \mid x_{1:n})$$

greedy →

$$\arg\max_{y_i} \log p\left(y_i \mid y_{1:i-1}, x_{1:n}\right)$$

$O(|V| \cdot T)$ cumulative error

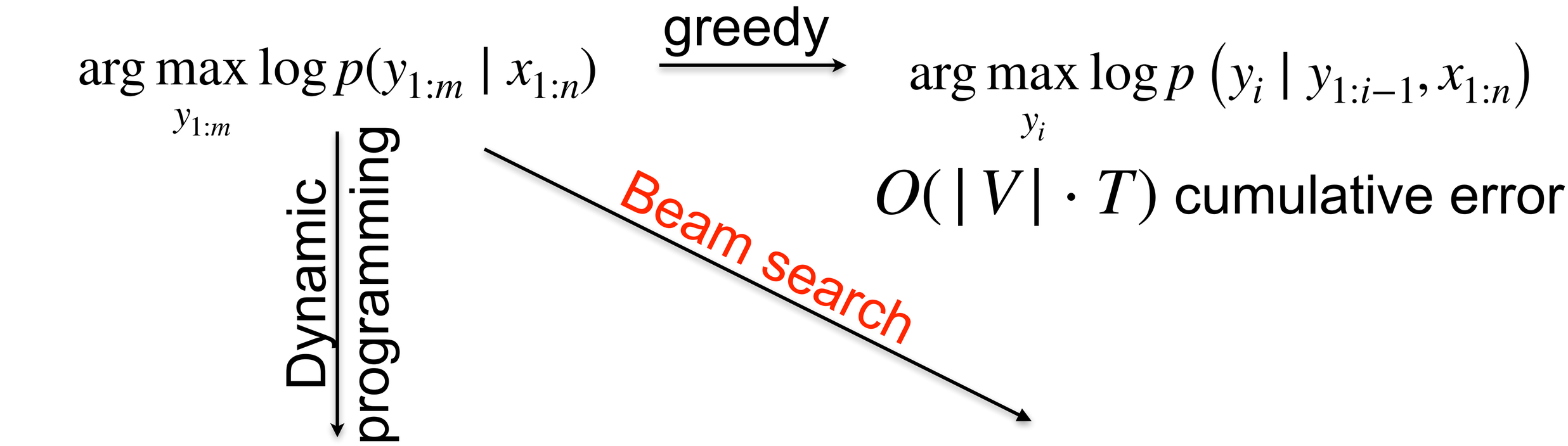
Dynamic programming ↓

Exact inference.

Too many states

space complexity $O(|V|^T)$

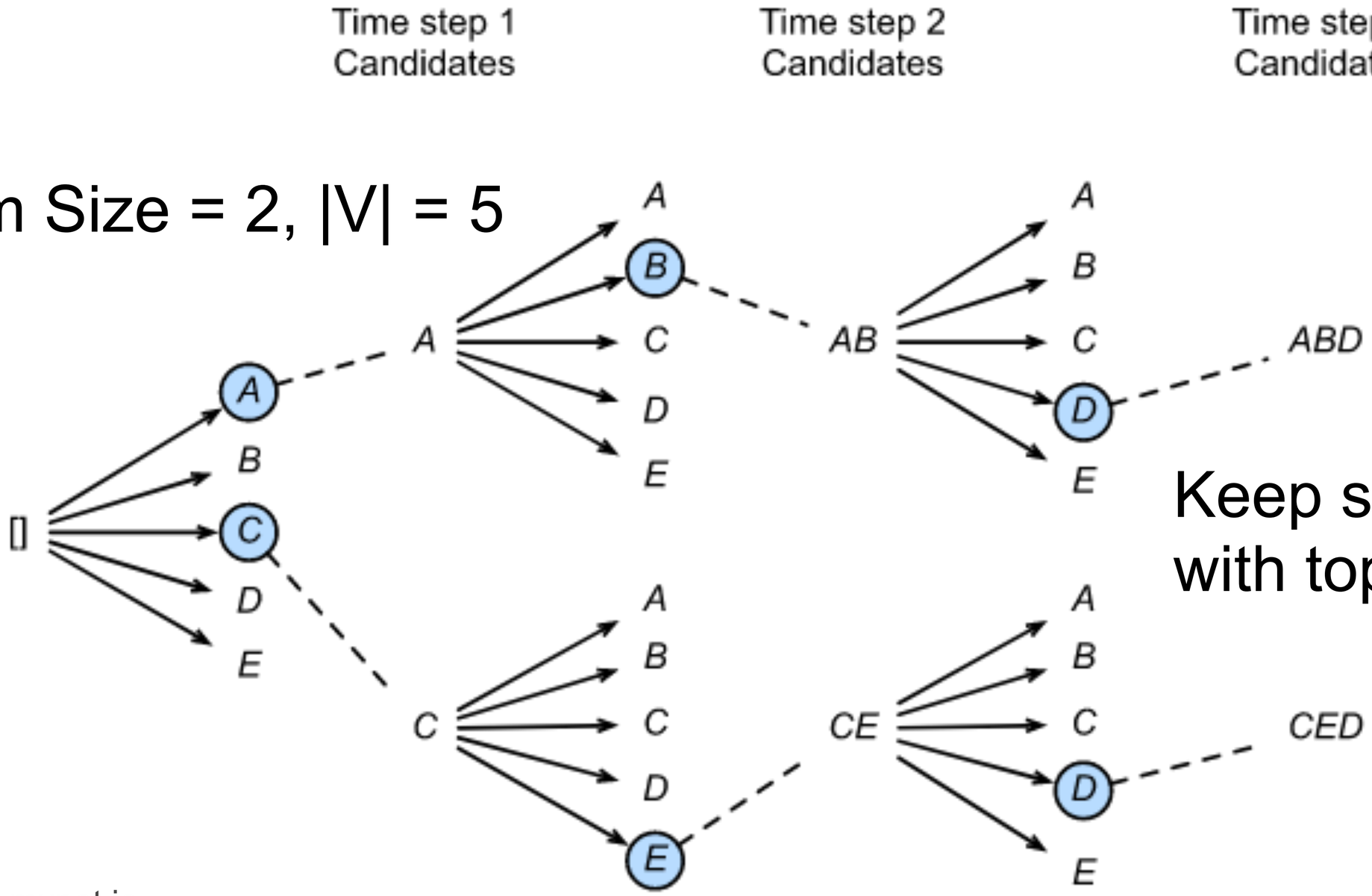
Search-based Algorithms for Inference



Exact inference.
Too many states
space complexity $O(|V|^T)$

Runtime $O(K \cdot |V| \cdot T)$
Cumulative error still linear but better than greedy. K as trade-off.

Beam Search for Sequence Sampling



Keep sequences with top K scores.

gluon-nlp.mxnet.io

Length Penalty

$\log p(y_i \mid y_{1:i-1}, x_{1:n})$ is always negative

Length Penalty

$$\log p(y_i \mid y_{1:i-1}, x_{1:n}) \text{ is always negative}$$

$$\downarrow$$

$$\arg\max_{y_{1:m}} \sum_{i=1}^{m} \log p(y_i \mid y_{1:i-1}, x_{1:n}) \text{ prefers shorter sequences!}$$

Length Penalty

$$\log p(y_i \mid y_{1:i-1}, x_{1:n})$$ is always negative

$$\downarrow$$

$$\arg\max_{y_{1:m}} \sum_{i=1}^{m} \log p(y_i \mid y_{1:i-1}, x_{1:n})$$ prefers shorter sequences!

Solution - penalize shorter sequences by length penalty.

$$\operatorname{argmax}_{y_{1:m}} \frac{\log p(y_{1:m} \mid x_{1:n})}{\mathrm{lp}(m)} \text{ where } \mathrm{lp}(m) = \frac{(K+m)^{\alpha}}{(K+1)^{\alpha}}$$

Length Penalty

$\log p(y_i \mid y_{1:i-1}, x_{1:n})$ is always negative

$$\downarrow$$

$$\arg\max_{y_{1:m}} \sum_{i=1}^{m} \log p(y_i \mid y_{1:i-1}, x_{1:n})$$ prefers shorter sequences!

Solution - penalize shorter sequences by length penalty.

$$\operatorname{argmax}_{y_{1:m}} \frac{\log p(y_{1:m} \mid x_{1:n})}{\mathrm{lp}(m)} \text{ where } \mathrm{lp}(m) = \frac{(K+m)^{\alpha}}{(K+1)^{\alpha}}$$

E.g. $K = 5$ and $\alpha = 0.6$