

# Dive into Deep Learning for NLP

## 4. Machine Translation and Sequence Sampling

Xingjian Shi

[gluon-nlp.mxnet.io](http://gluon-nlp.mxnet.io)

13:15-14:15	Natural Language Processing and Deep Learning Basics
14:15-14:25	Break
14:25-15:15	Context-free Representations with Word Embeddings
15:15-15:55	Machine Translation and Sequence Sampling
15:55-16:35	Contextual Representations with BERT
16:35-16:45	Break
16:45-17:15	Model Deployment with TVM

# Outline

- Encoder-Decoder
- Attention Mechanism
- Transformer
- Sequence Sampler
- Hands-on Demo

# Encoder- Decoder

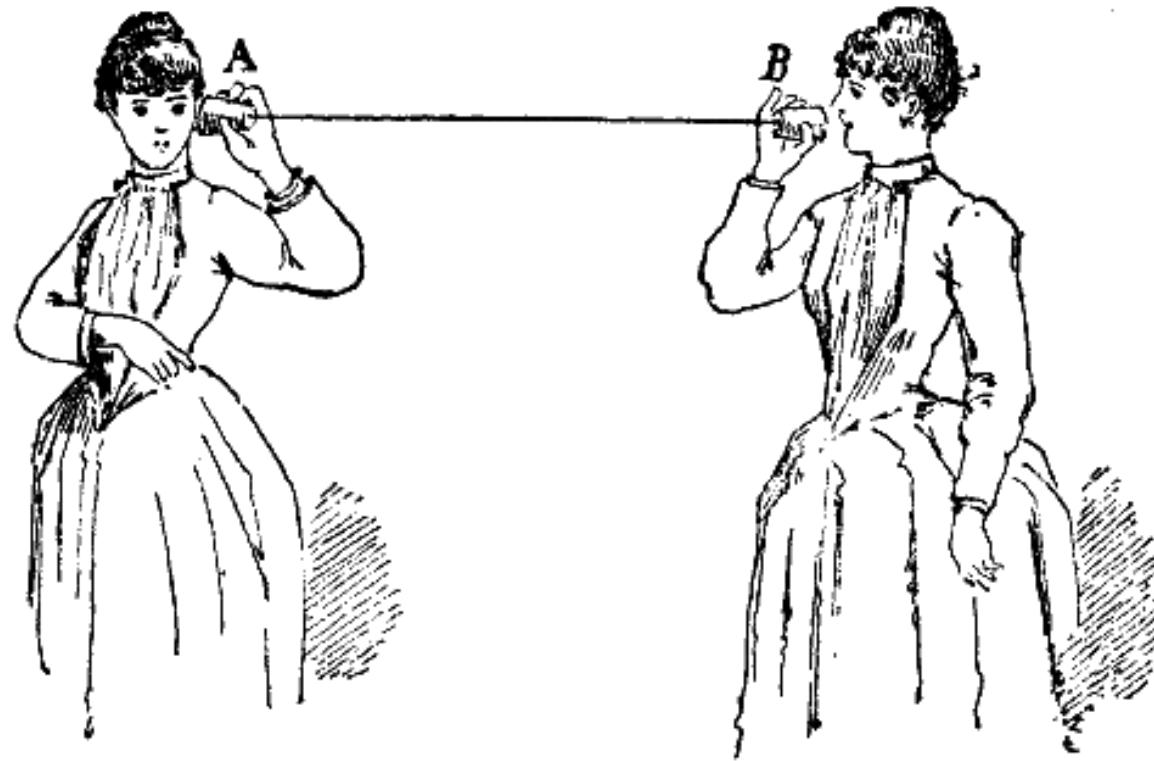
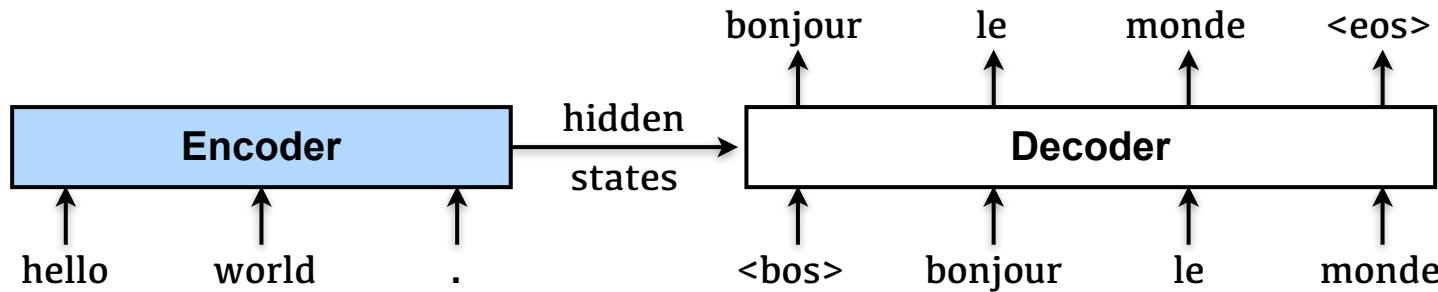


Image Source: [https://en.wikipedia.org/wiki/Tin\\_can\\_telephone](https://en.wikipedia.org/wiki/Tin_can_telephone)

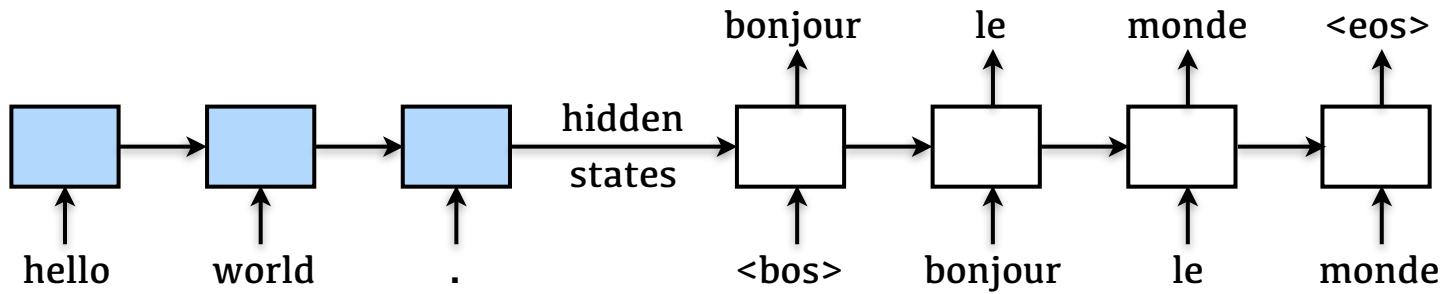
# Encoder-Decoder for Machine Translation



$h = \text{Encoder}(\text{'hello'}, \text{'world'}, \text{'.})$

'bonjour', 'le', 'monde', '<eos>' =  $\text{Decoder}(h)$

# RNN as Encoder & Decoder

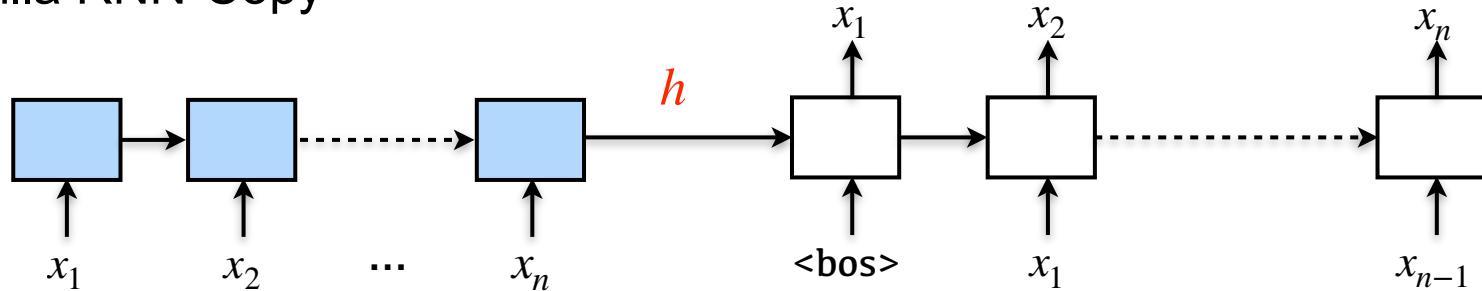


# The Limitation of RNN Encoder-Decoder

- Mental Experiment: Sequence Copy

- $x_1, x_2, \dots, x_n \rightarrow h \rightarrow x_1, x_2, \dots, x_n$

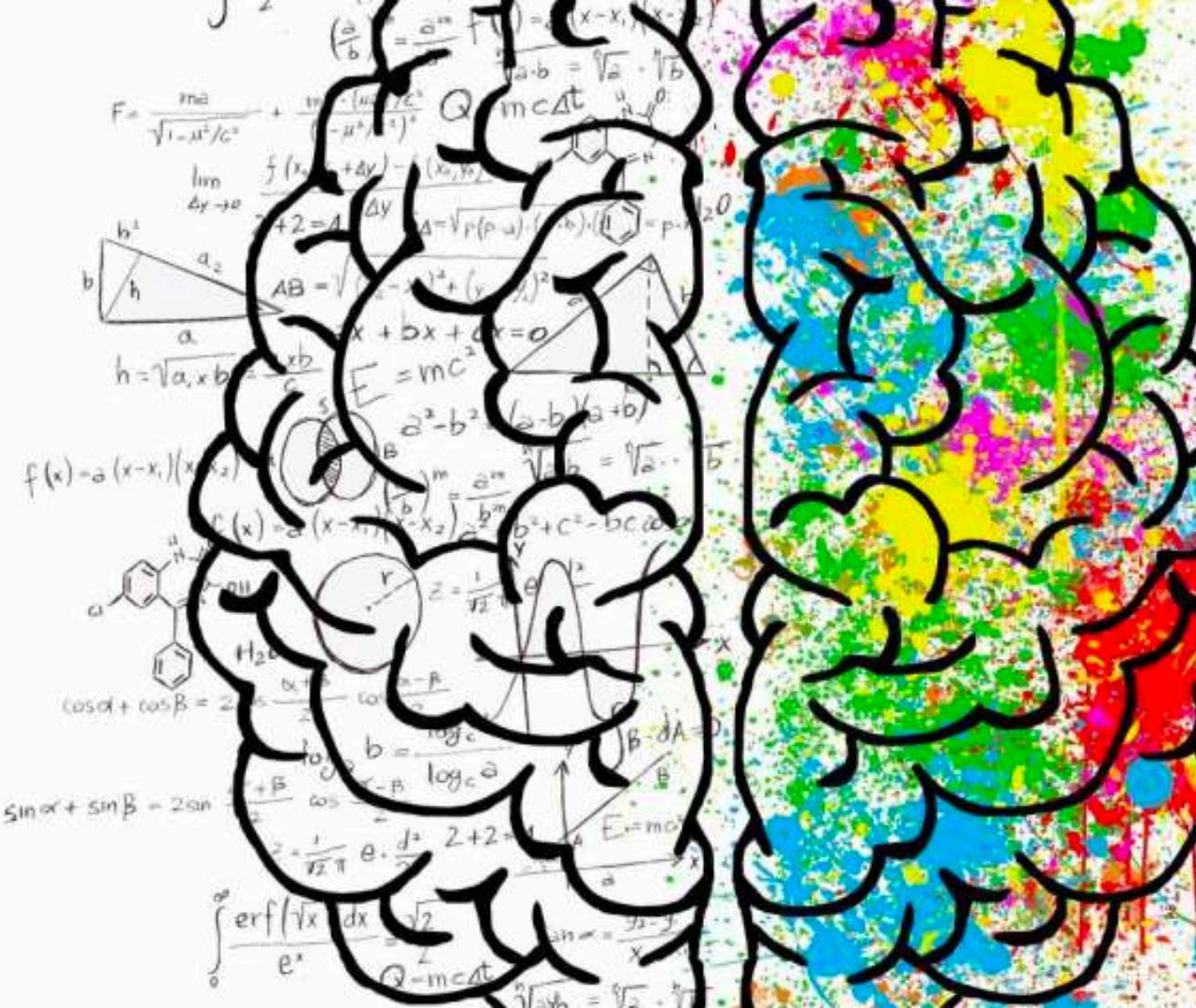
- Vanilla RNN-Copy



- What if the sequence length approaches **infinity**?

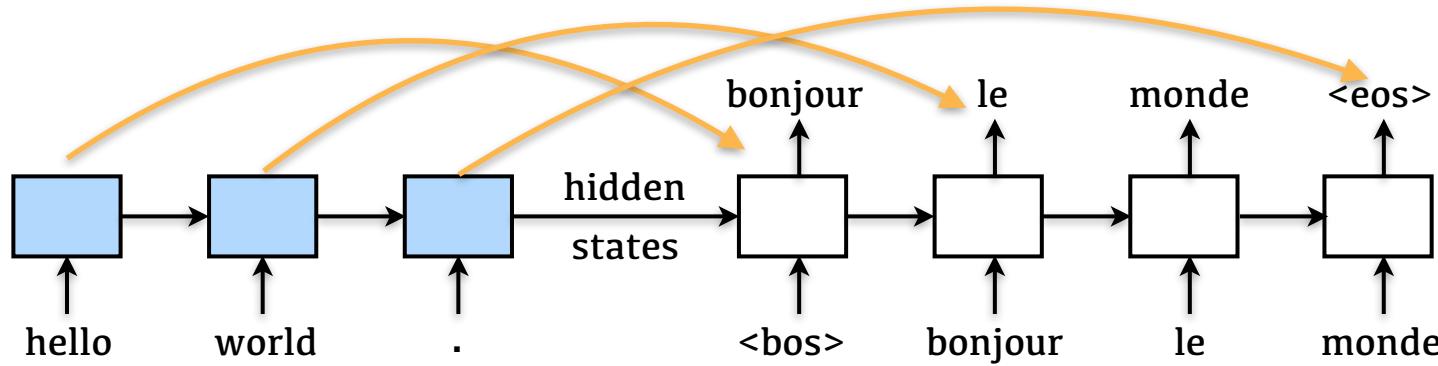
- Encoding to a **fixed-dimensional** vector will have information loss
- **Can never solve the problem!**

# Attention



# Motivation

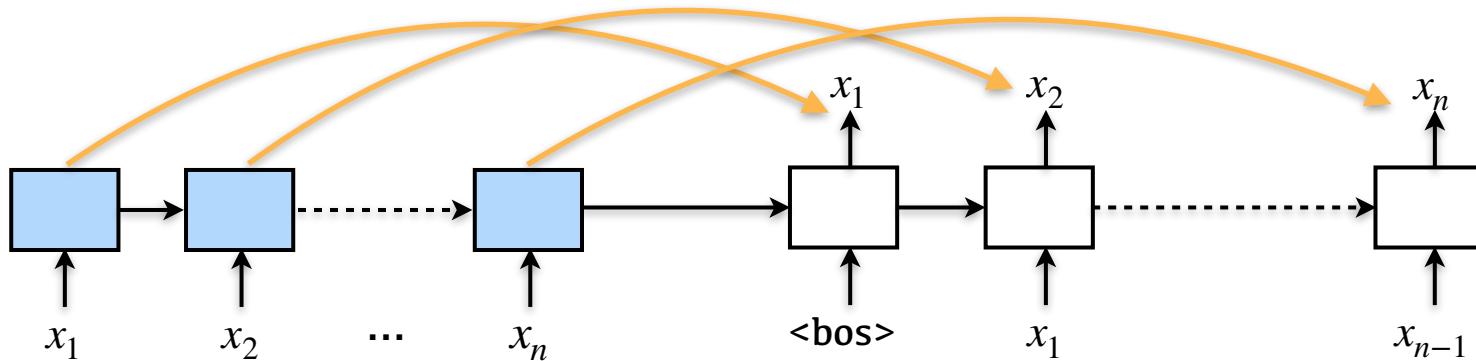
- Each generated token might be related to different source tokens.



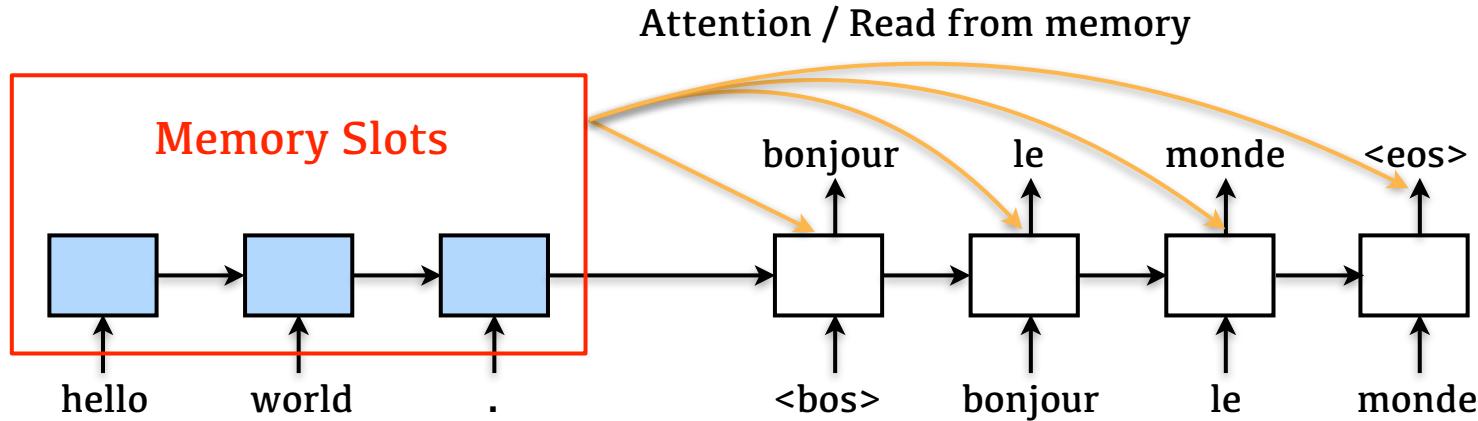
- Instead of just using the last state, **all states** are used in the decoding process.

# Sequence Copy by Attention

- Sequence Copy is possible with attention
  - Why? All states in the encoder are kept — **Memory**
- $x_1, x_2, \dots, x_n \rightarrow h_1, h_2, \dots, h_n \rightarrow x_1, x_2, \dots, x_n$



# Attention & Memory

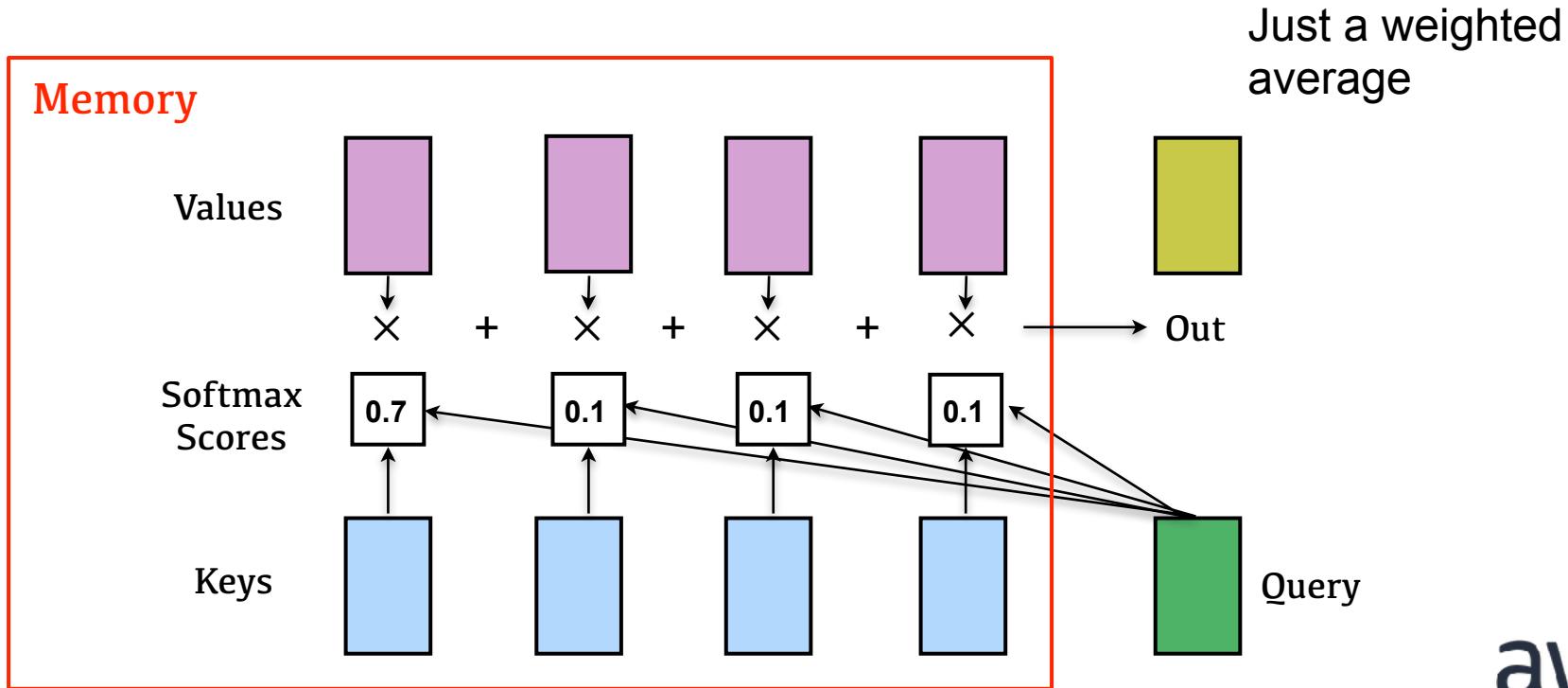


Treat each state in the RNN as **memory slots**

In each decoding step, the network will search the **states** that are related to the **query**.

# Modern Interpretation — Key Value Memory Network

$$\text{Out} = \text{Attention}(\text{Query}, \text{Key}, \text{Value})$$



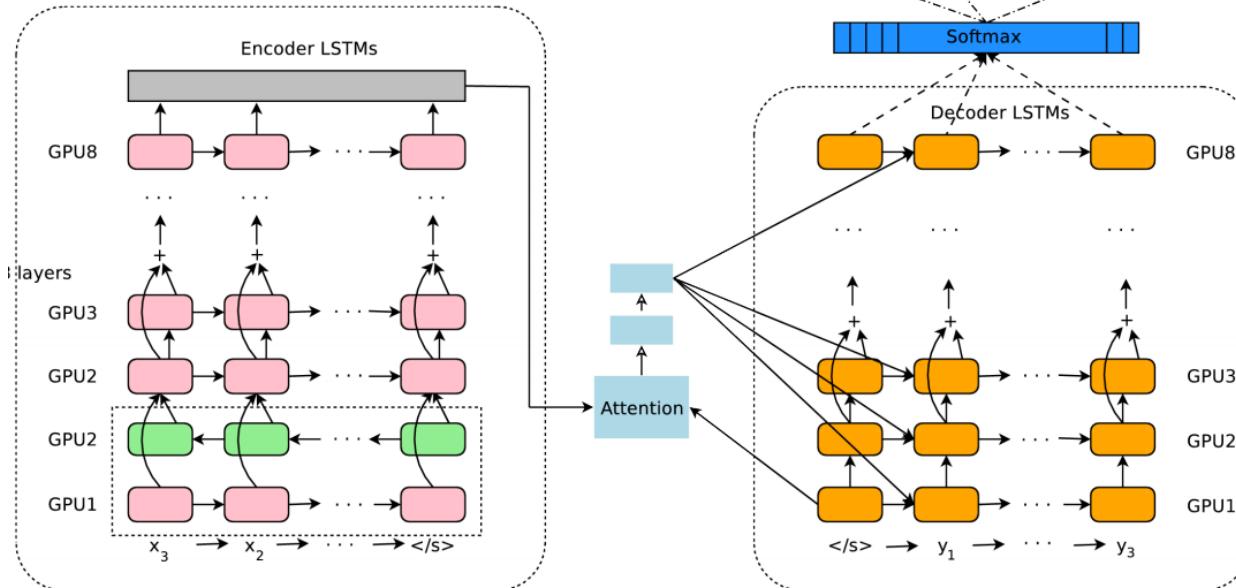
# Implement Different Attention Scores in GluonNLP

- Query:  $q$ , Keys:  $k_{1:n}$

[gluonnlp.model.attention\\_cell](#)

Attention Score Before Softmax	GluonNLP
$s_i = \mathbf{q}^T \mathbf{k}$	DotProductAttentionCell
$s_i = W_o \tanh(W_i[\mathbf{q} \parallel \mathbf{k}] + b)$	MLPAttentionCell

# Google Neural Machine Translation



Encoder: Bidirectional  
LSTM + Residual

Decoder: LSTM +  
Residual + Attention

[Wu et al., 2016] [Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation](#)

Residual:  $X + f(X)$

Implementation  
available in GluonNLP

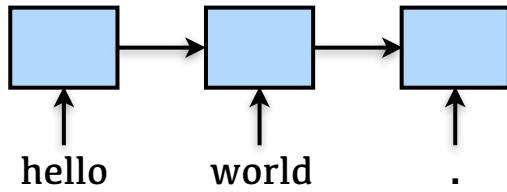
[scripts/machine\\_translation](#)

# Transformer



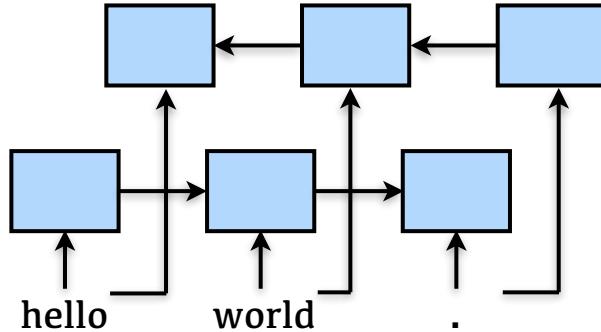
# RNN is slow in the encoding phase

RNN



Sequential, Hard to make parallel

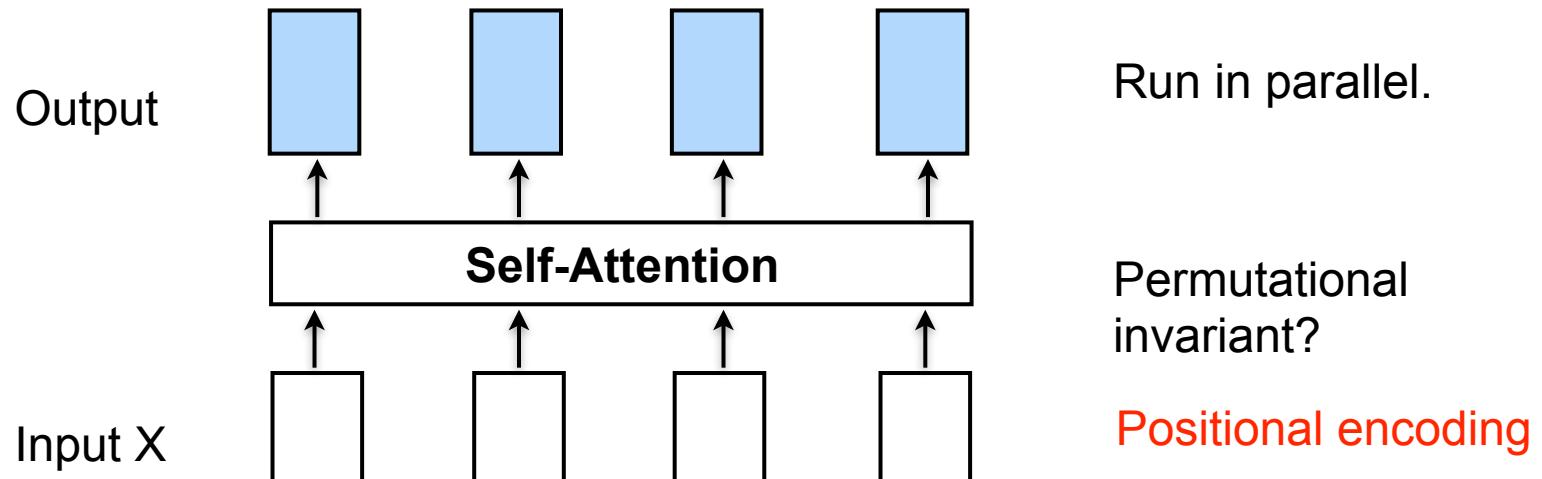
Bidirectional  
RNN



# Self-Attention

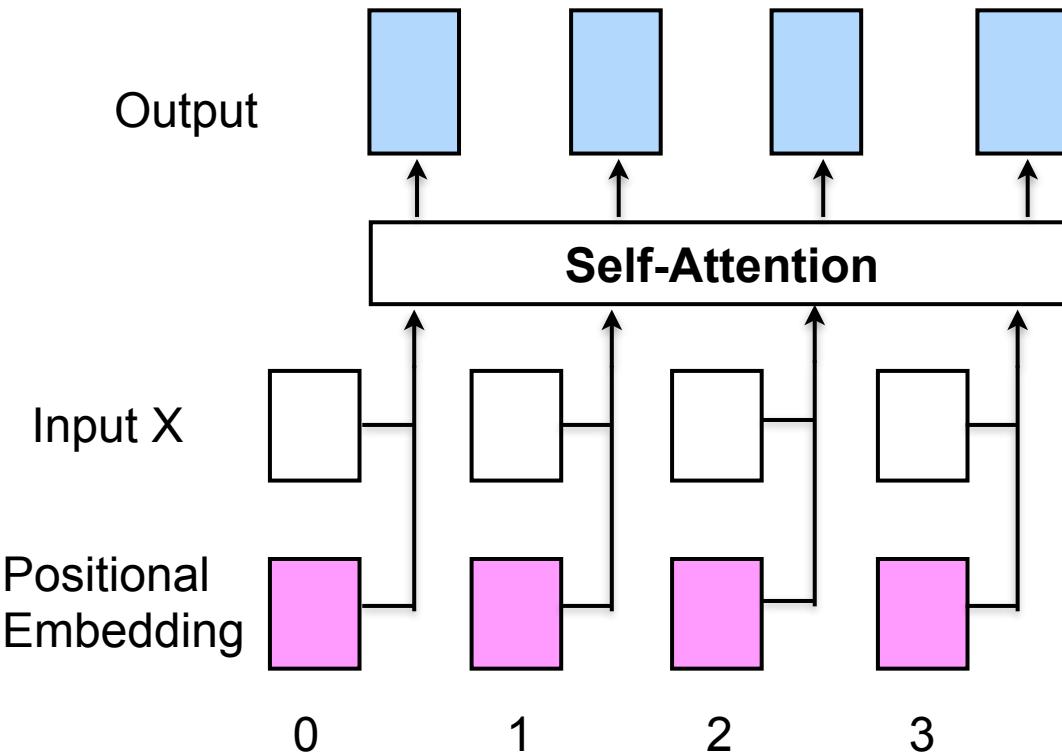
$\text{Out} = \text{Attention}(\underline{\text{Query}}, \text{Key}, \text{Value})$

Self-Attention:  $\text{Attention}(W_q X, W_k X, W_v X)$



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

# Positional Encoding



Many options:

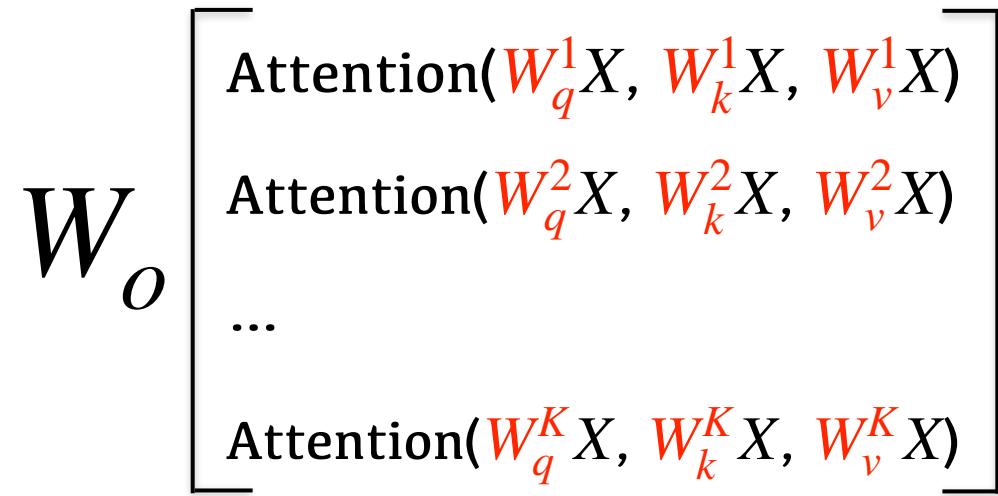
- Learn the positional embeddings
- Interpolating a Sinusoidal function

# Multi-head Attention

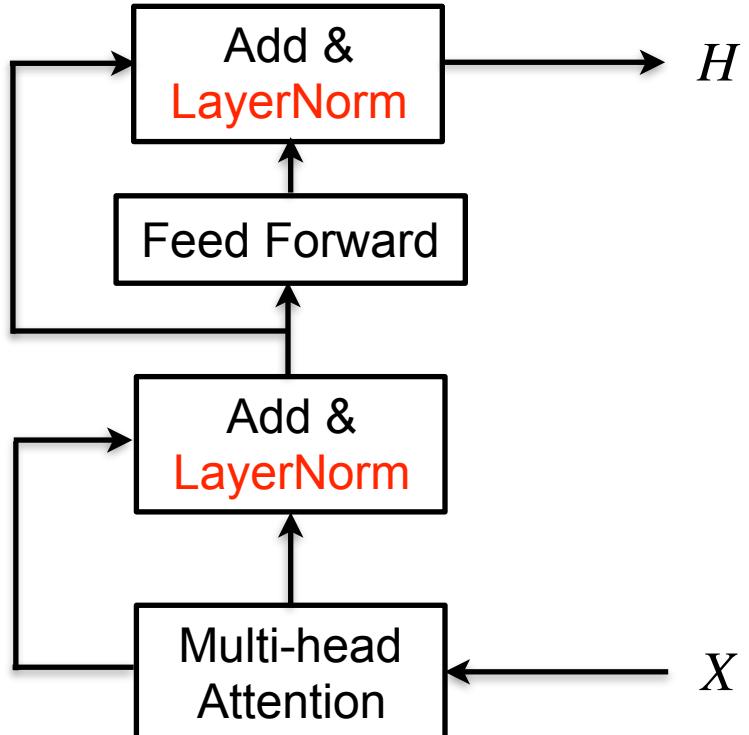
One-head: Similarity measurement in one subspace

Multi-head: **Multiple subspaces**

Multi-head Attention:



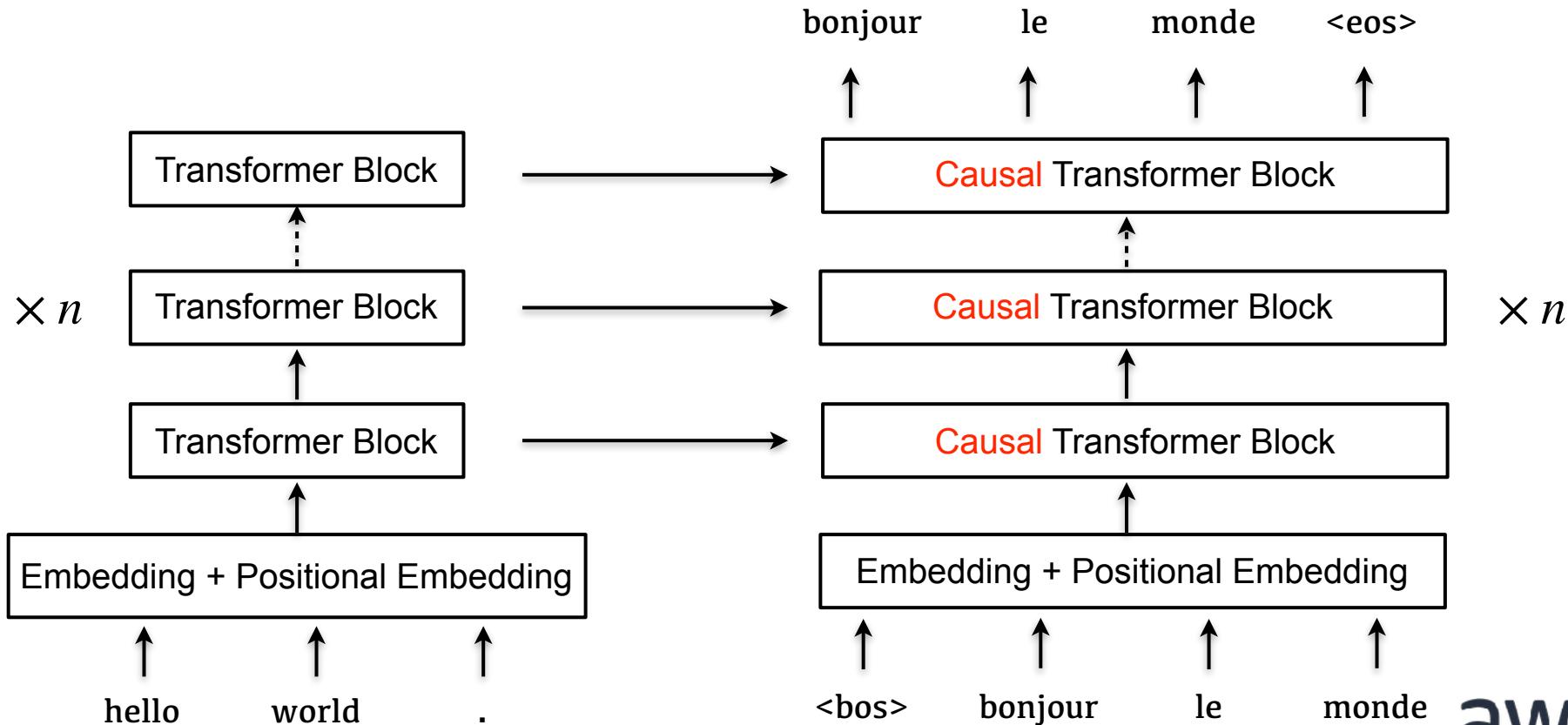
# Transformer Block = Multi-head Attention + Residual + Layer Normalization



$X.\text{shape} = (\#\text{Batch}, \#\text{Channels})$

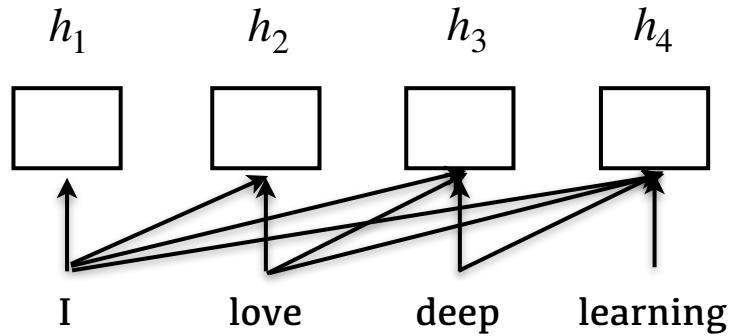
$$\text{LayerNorm: } \frac{X - X.\text{mean}(\text{axis}=-1)}{X.\text{std}(\text{axis}=-1)}$$

# Transformer Architecture for Machine Translation



# Causal Attention

- Never attend to the future. Avoid information leak.



# Sequence Sampling



# How to Predict with a Machine Translation Model?

- Decoder returns the probability of single-step ahead predictions:  $p(y_i | y_{1:i-1}, x_{1:n})$
- One-step ahead — Multi-step ahead
  - $\arg \max_{y_{1:m}} \log p(y_{1:m} | x_{1:n}) = \arg \max_{y_{1:m}} \sum_{i=1}^m \log p(y_i | y_{1:i-1}, x_{1:n})$

# Possible Algorithm for Inference

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

# Possible Algorithm for Inference

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

$O(|V|T)$  Accumulative Error

# Possible Algorithm for Inference

$$\arg \max_{y_{1:m}} \log p(y_{1:m} | x_{1:n}) \xrightarrow{\text{Greedy}} \arg \max_{y_i} \log p(y_i | y_{1:i-1}, x_{1:n})$$

$O(|V|T)$  Accumulative Error

Dynamic  
programming

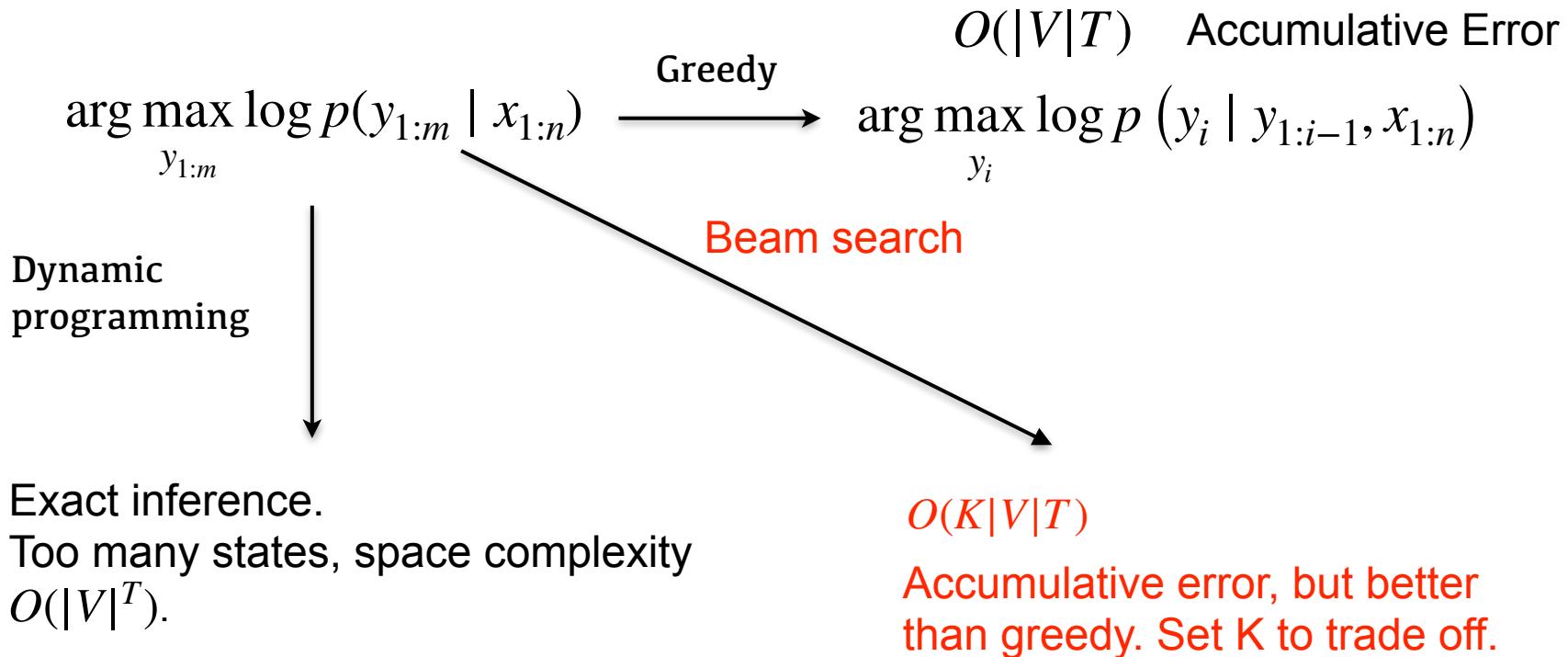


Exact inference.

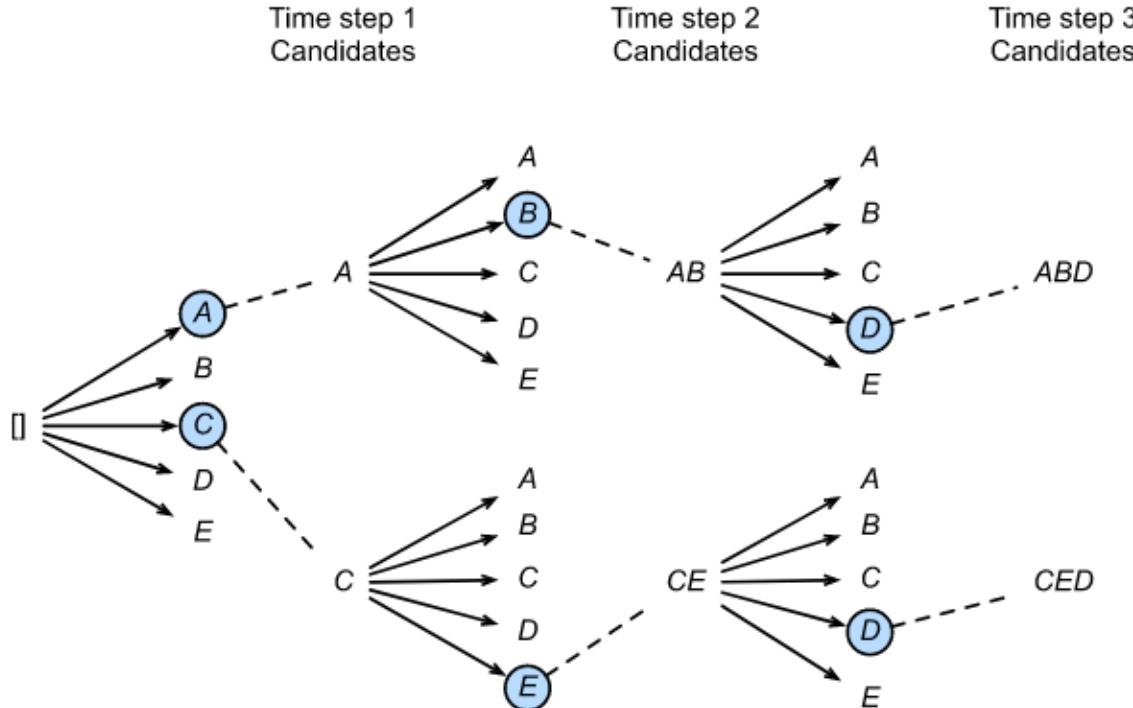
Too many states, space complexity

$O(|V|^T)$ .

# Possible Algorithm for Inference



# Beam Search for Sequence Sampling



Beam Size = 2,  $|V| = 5$

# Length Penalty

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

# Length Penalty

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

↓

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

# Length Penalty

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

↓

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

How to solve? Penalize shorter sequences.

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

# Length Penalty

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

↓

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

How to solve? Penalize shorter sequences.

$$\arg \max_{y_{1:m}} \frac{\log p(y_{1:m} \mid x_{1:n})}{lp(m)} \quad \text{Log-likelihood Score}$$

# Length Penalty

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

↓

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

How to solve? Penalize shorter sequences.

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

Log-likelihood ScoreLength penalty function

# Length Penalty

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

↓

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

How to solve? Penalize shorter sequences.

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

Log-likelihood ScoreLength penalty function

$$lp(m) = \frac{(K+m)^\alpha}{(K+1)^\alpha}$$

# Length Penalty

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

↓

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

How to solve? Penalize shorter sequences.

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

Log-likelihood ScoreLength penalty function

$$lp(m) = \frac{(K+m)^\alpha}{(K+1)^\alpha}$$

E.g., K = 5, alpha = 0.6

# How to evaluate the prediction?

- Bilingual Evaluation Understudy (BLEU) Score
- Based on a modified version of n-gram precision
- Not go to the details, rely on [sacreBLEU](#) to evaluate

[\[Papineni et al., 2002\] BLEU: a Method for Automatic Evaluation of Machine Translation](#)

# Demo: Machine Translation

04\_machine\_translation/transformer.ipynb