

Deep Learning lecture 9

Sequence Modeling (2)

Yi Wu, IIIS

Spring 2025

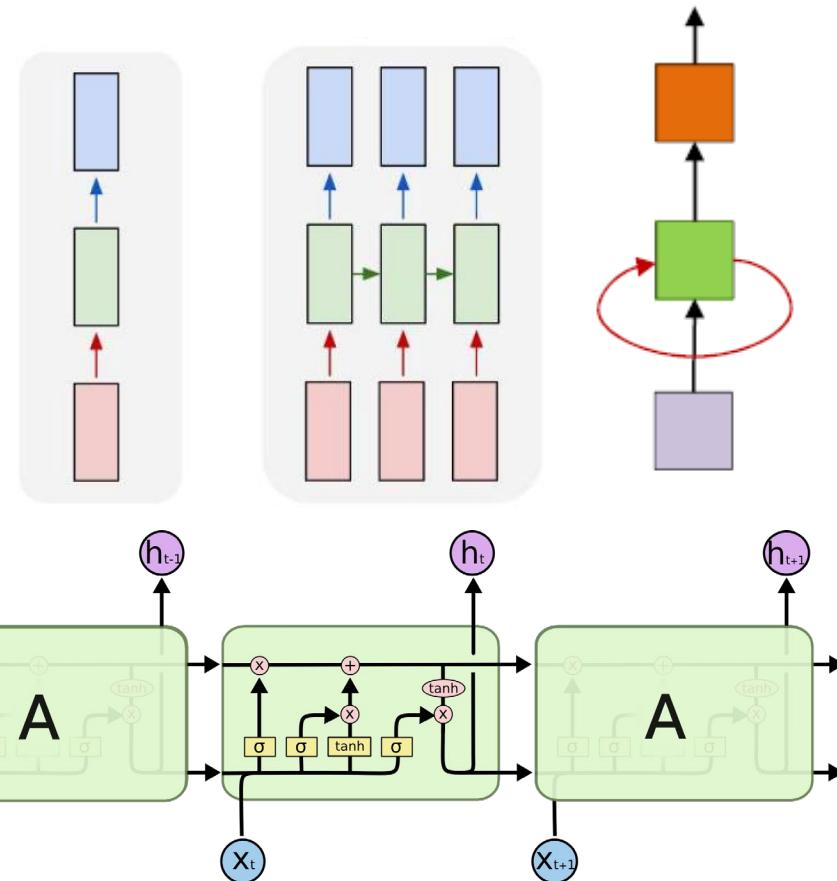
Apr-14

Today's Topic

- Sequence to Sequence Model and Attention Mechanism
- The Transformer Model
- Generation Speedup for Transformer Model

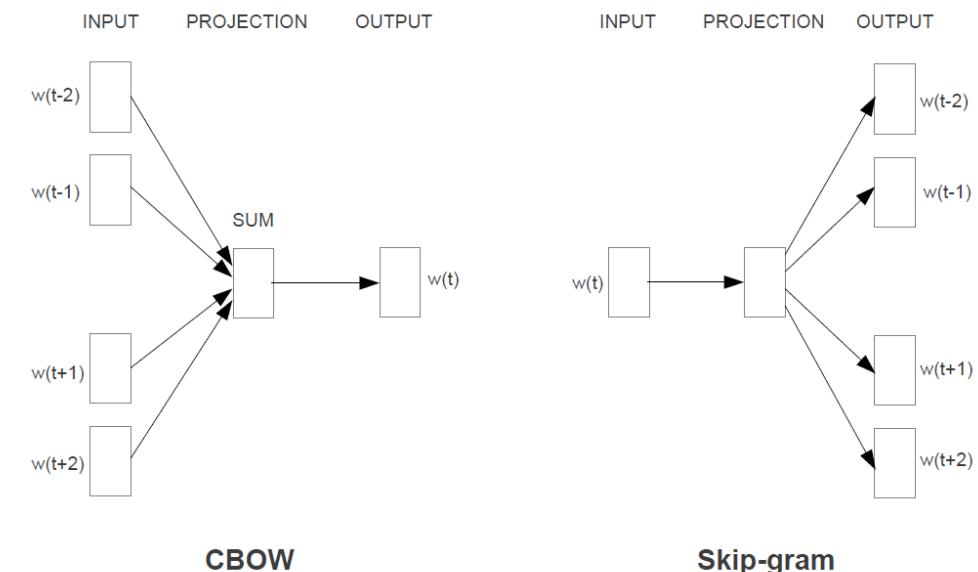
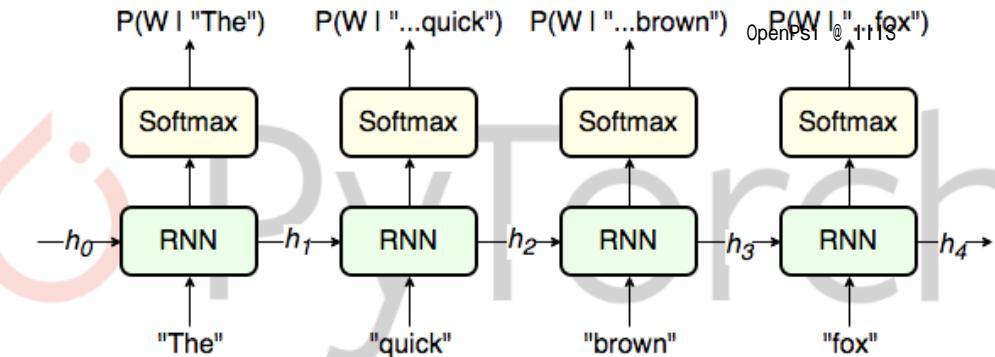
RNN Recap

- Recurrent Neural Network
 - Same MLP network over a sequence (i.e., “loops”)
 - Arbitrarily long sequences → fixed-sized vector
 - Training: backpropagation through time (BPTT)
 - Practical Issues
 - Weights/Gradient explosion and saturation
 - A few tricks for gradient explosion
 - Gradient clipping, truncated BPTT, careful initialization
- Long Short-Term Memory (LSTM) Network
 - A specialized RNN for long-term dependency (~100 timesteps)
 - Key ideas: elementary gates
 - Variants: bidirectional LSTM; Peephole LSTM; GRU; etc



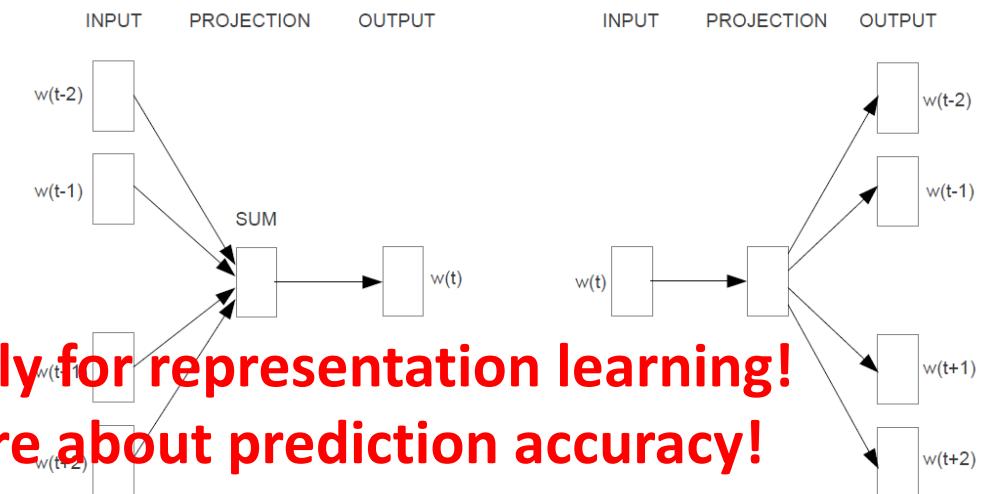
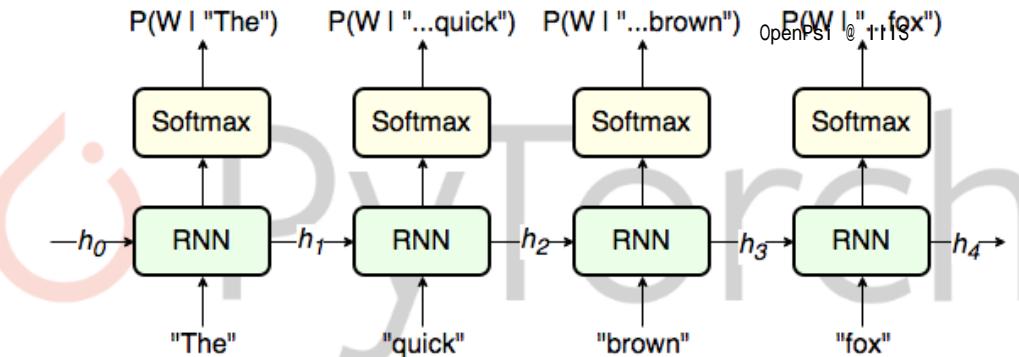
RNN Recap

- Autoregressive Language Model
 - Generative model over texts: $P(X) = \prod_t P(X_t | X_{i < t})$
 - LSTM language model: $Y_t, h_t = LSTM(h_{t-1}, X_t)$; $P(X_t | X_{i < t}) = \text{Softmax}(Y_t)$
 - Word Embedding
 - A distributed representation for word semantics
 - Word2Vec: a tool for word embedding
 - Objective: from context c to predict word w
 - CBOW and Skip-Gram
 - Negative Sampling
 - Multi-class prediction → binary classification
 - D training corpus; V vocabulary
- $$L(W, C) = \sum_{(c,w) \in D} \log \frac{1}{\exp(-w^T c) + 1} + \sum_{c \in D, w \in V} \log \frac{\exp(-\tilde{w}^T c)}{\exp(-\tilde{w}^T c) + 1}$$



RNN Recap

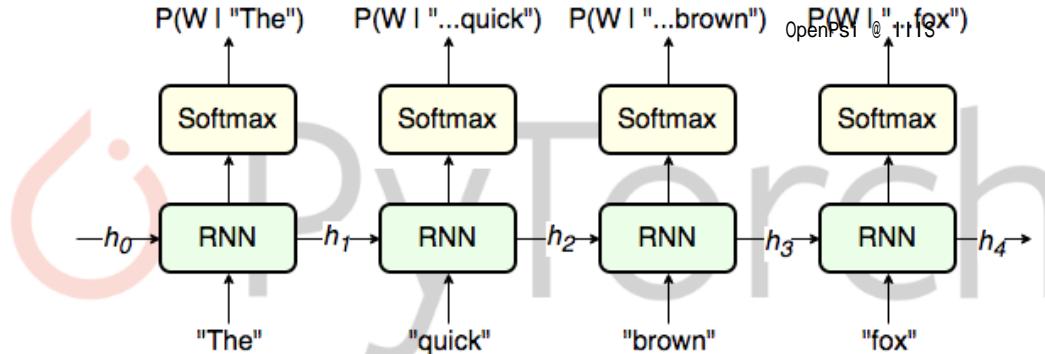
- Autoregressive Language Model
 - Generative model over texts: $P(X) = \prod_t P(X_t | X_{i < t})$
 - LSTM language model: $Y_t, h_t = LSTM(h_{t-1}, X_t); P(X_t | X_{i < t}) = \text{Softmax}(Y_t)$
 - Word Embedding
 - A distributed representation for word semantics
 - Word2Vec: a tool for word embedding
 - Objective: from context c to predict word w
 - CBOW and Skip-Gram
 - Negative Sampling
 - Multi-class prediction → binary classification
 - D training corpus; V vocabulary
- $$L(W, C) = \sum_{(c,w) \in D} \log \frac{1}{\exp(-w^T c) + 1} + \sum_{c \in D, w \in V} \log \frac{\exp(-\tilde{w}^T c)}{\exp(-\tilde{w}^T c) + 1}$$



**Word2Vec is only for representation learning!
It does not care about prediction accuracy!**

RNN Recap

- Autoregressive Language Model
 - Generative model over texts: $P(X) = \prod_t P(X_t | X_{i < t})$
 - LSTM language model: $Y_t, h_t = LSTM(h_{t-1}, X_t); P(X_t | X_{i < t}) = \text{Softmax}(Y_t)$
 - Word Embedding
 - A distributed representation for word semantics
- Word2Vec: a tool for word embedding
- More Techniques
 - Hierarchical softmax
 - Beam search
 - ELMo for contextualized embeddings



Language Model Applications

- Text Classification
 - Supervised learning
- Text Generation
 - $p(X; \theta)$: the probability for X
 - Unconditioned Generation
 - E.g., AI作诗
 - **Conditioned generation?**
 - E.g., Machine translation

深度之梦

在数据的海洋里遨游，
算法如风，吹散迷雾。
神经元闪烁似星辰，
连接着未来的道路。
梯度回溯千重浪，
优化求解万象生。
一行代码塑乾坤，
模型自我去提升。



The screenshot shows a machine translation tool with two input fields. The left field has "检测语言" (Detected language) set to "英语" (English), and the text "Deep learning is a popular area in AI." The right field has "中文 (简体)" (Simplified Chinese) selected, and the translated text is "深度学习是AI的热门领域." Below the translation, the pinyin "Shèndù xuéxí shì AI de rèmén lǐngyù." is shown. At the bottom, there are icons for microphone, speaker, and sharing, along with page navigation symbols.

Machine Translation

- A task of translating a sentence from a source language to the target language

x: *L'homme est né libre, et partout il est dans les fers*



y: *Man is born free, but everywhere he is in chains*

人生而自由，却无往不在枷锁之中。——卢梭《社会契约论》

– Rousseau

Machine Translation

- Before 2014: Statistical Machine Translation
 - Extremely complex systems that require massive human efforts
 - Separately designed components
 - A lot of feature engineering
 - Lots of linguistics domain knowledge and expertise
- Before 2016:
 - Google's commercial translation product is based on statistical machine translation
- What happened in 2014?
 - A borrowed slide from Stanford CS224

2014

(dramatic reenactment)

2014

Neural
Machine
Translation

MT research

(dramatic reenactment)

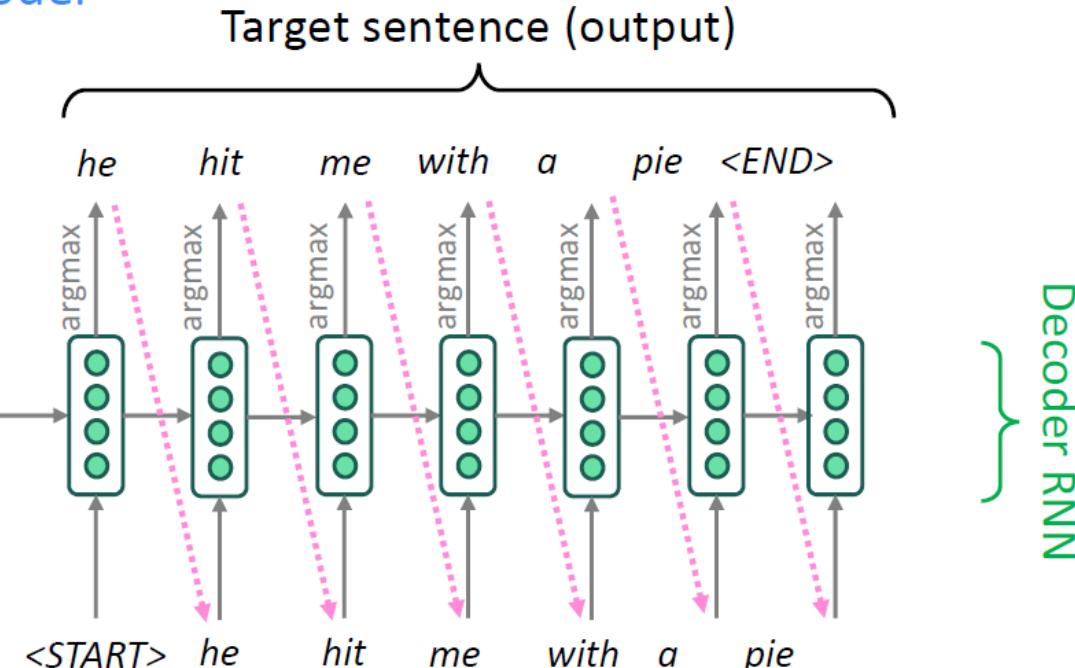
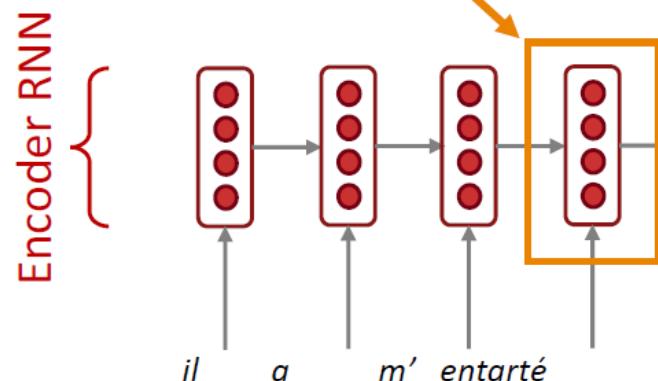
Sequence to Sequence Model

- Neural Machine Translation (NMT)
 - Learning to translate via single end-to-end neural network!
 - Source language X , then $Y = f(X; \theta)$
- Sequence-to-Sequence Model (Seq2Seq, Sutskever et al, NIPS2014)
 - NeurIPS 2024 test-of-time award
 - Two RNNs: f_{enc} and f_{dec} , $X \rightarrow f_{enc} \rightarrow h \rightarrow f_{dec} \rightarrow Y$
 - Encoder f_{enc}
 - It takes in X , and produce the initial hidden state h for decoder
 - We can use bidirectional RNN
 - Decoder f_{dec}
 - It takes in the hidden state h from f_{enc} to generate Y
 - Autoregressive language model

Sequence to Sequence Model

The sequence-to-sequence model

Encoding of the source sentence.
Provides initial hidden state
for Decoder RNN.



Encoder RNN produces
an encoding of the
source sentence.

Decoder RNN is a Language Model that generates target sentence, *conditioned on encoding*.

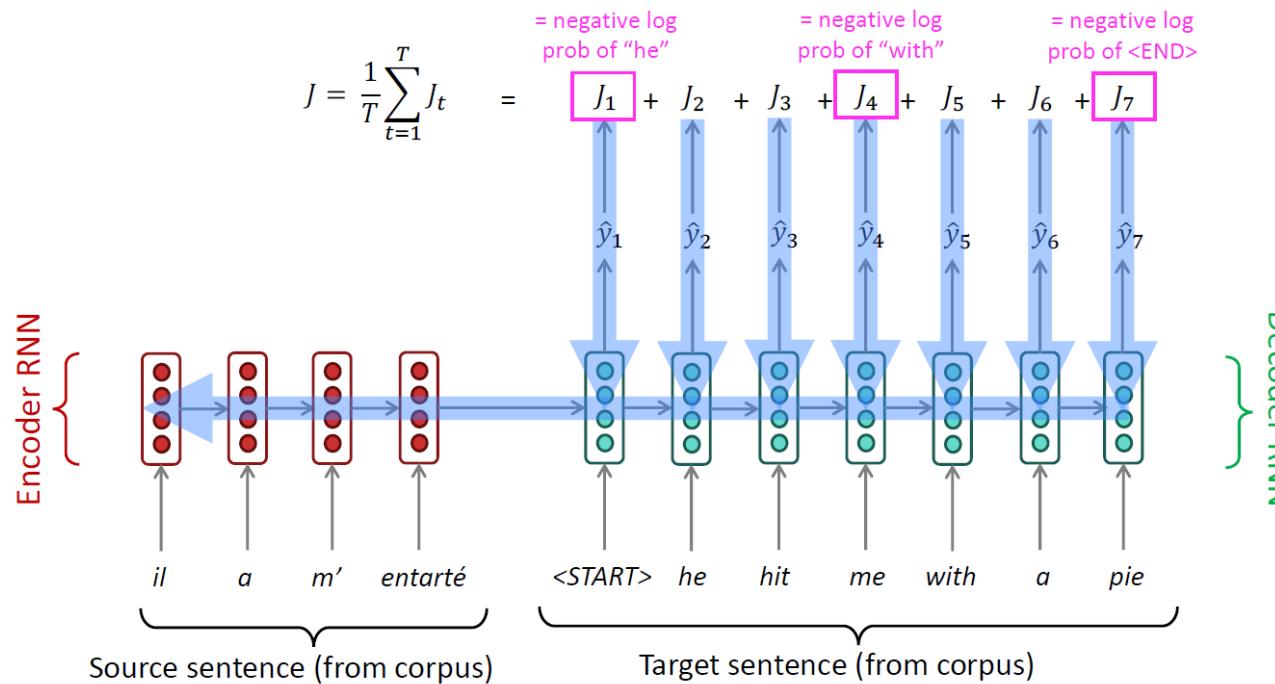
Note: This diagram shows **test time** behavior: decoder output is fed in .as.next step's input

Sequence to Sequence Model

- Seq2Seq is a conditioned language model
 - $h = f_{enc}(X)$ (final hidden state)
 - $Y = f_{dec}(h)$ (a LM that conditions on the initial hidden state h)
- Seq2Seq model is particularly generic for a lot of applications
 - Summarization (摘要) or Captioning (起标题)
 - Article → abstract/caption
 - Dialogue (对话)
 - Previous utterance → next utterance
 - Code generation
 - Natural language → python
 - VAE-based seq2seq model for text generation with latent variables

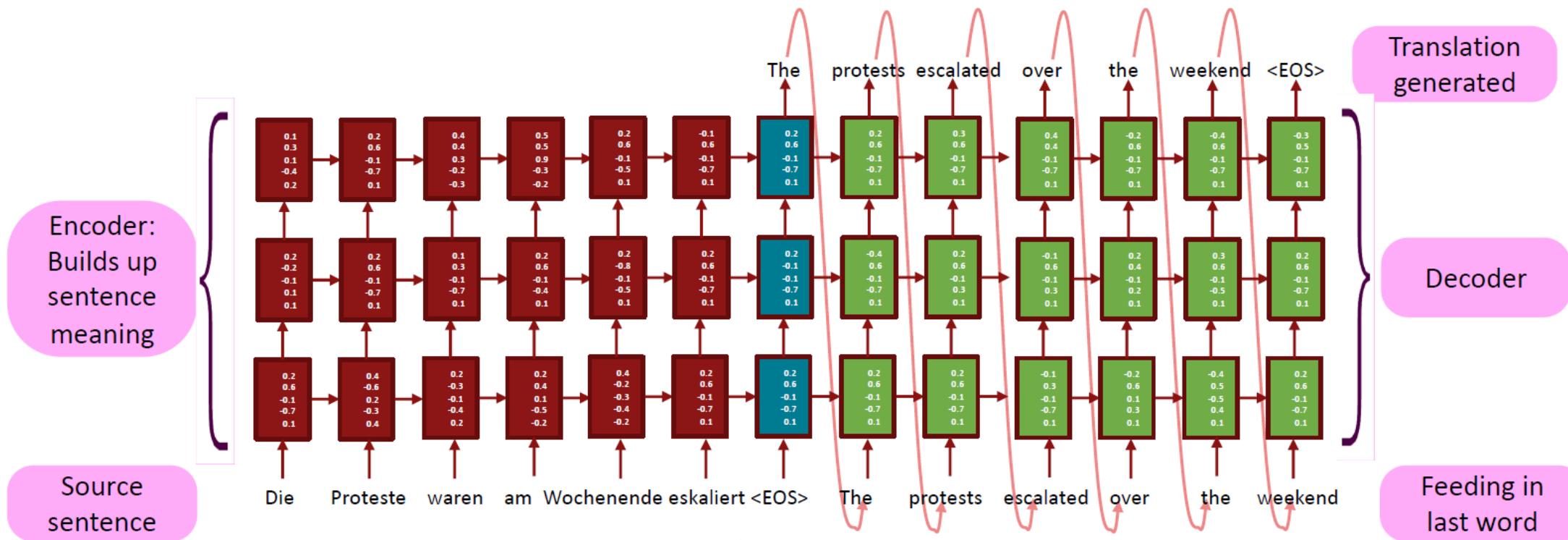
Sequence to Sequence Model

- How to train a seq2seq model?
 - Collect a huge paired dataset and train it end-to-end via BPTT!
 - MLE learning for $P(Y|X) = P(Y|f_{enc}(X))$



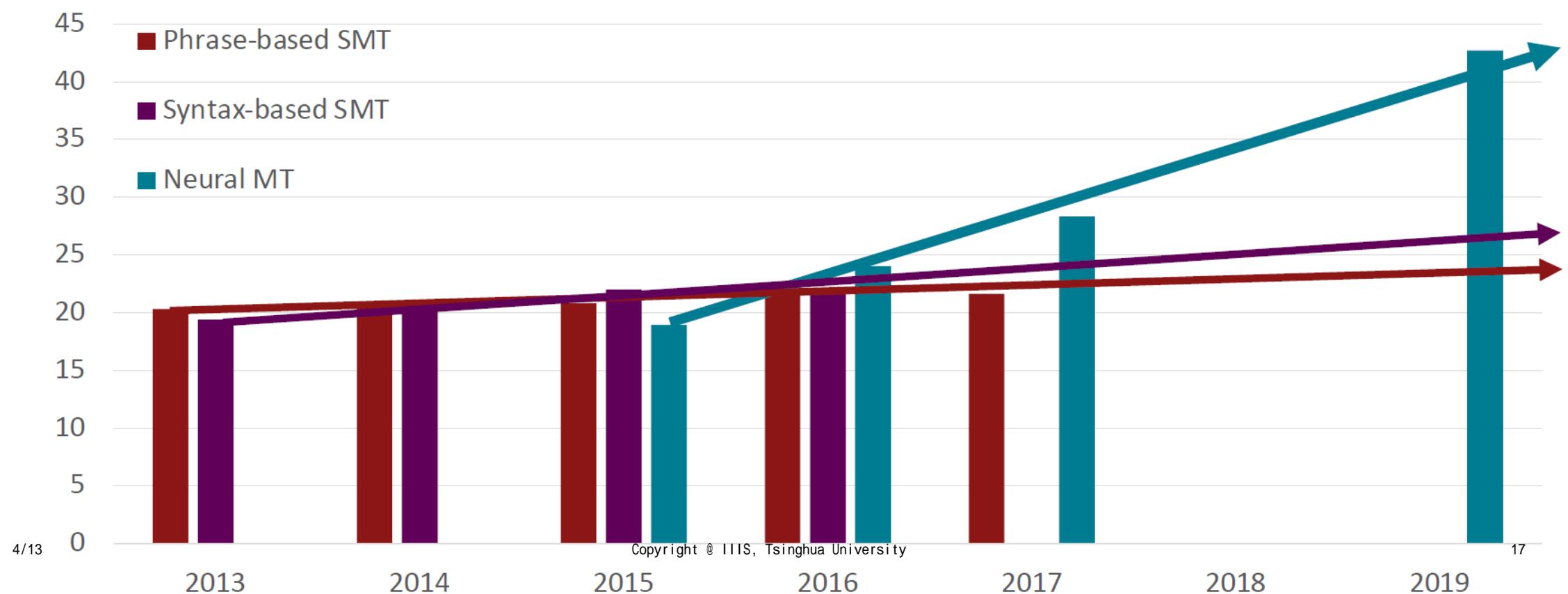
Sequence to Sequence Model

- We can also make the model deeper!
 - Stacked seq2seq model



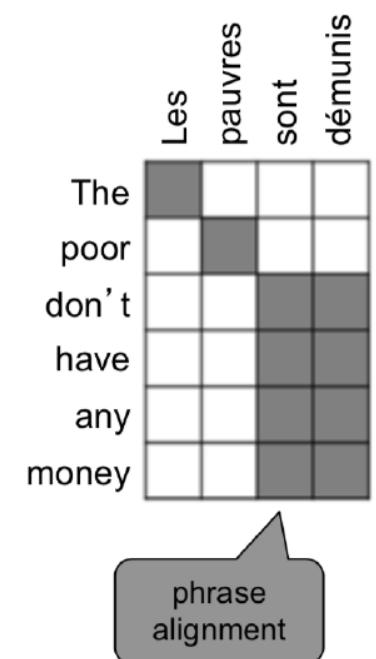
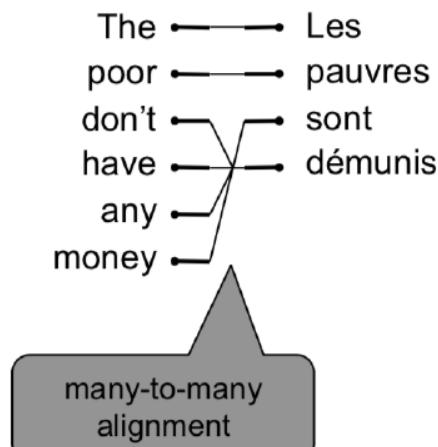
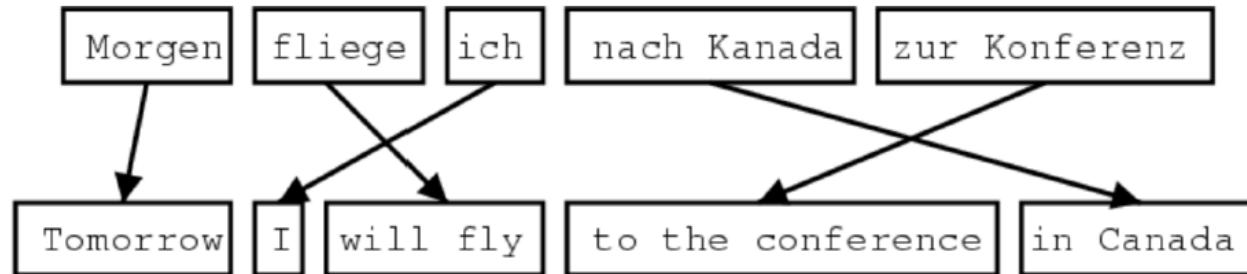
Sequence to Sequence Model

- 2016: Google switch google translate from SMT to NMT
 - Seq2Seq paper has >28.6k citations since 2014



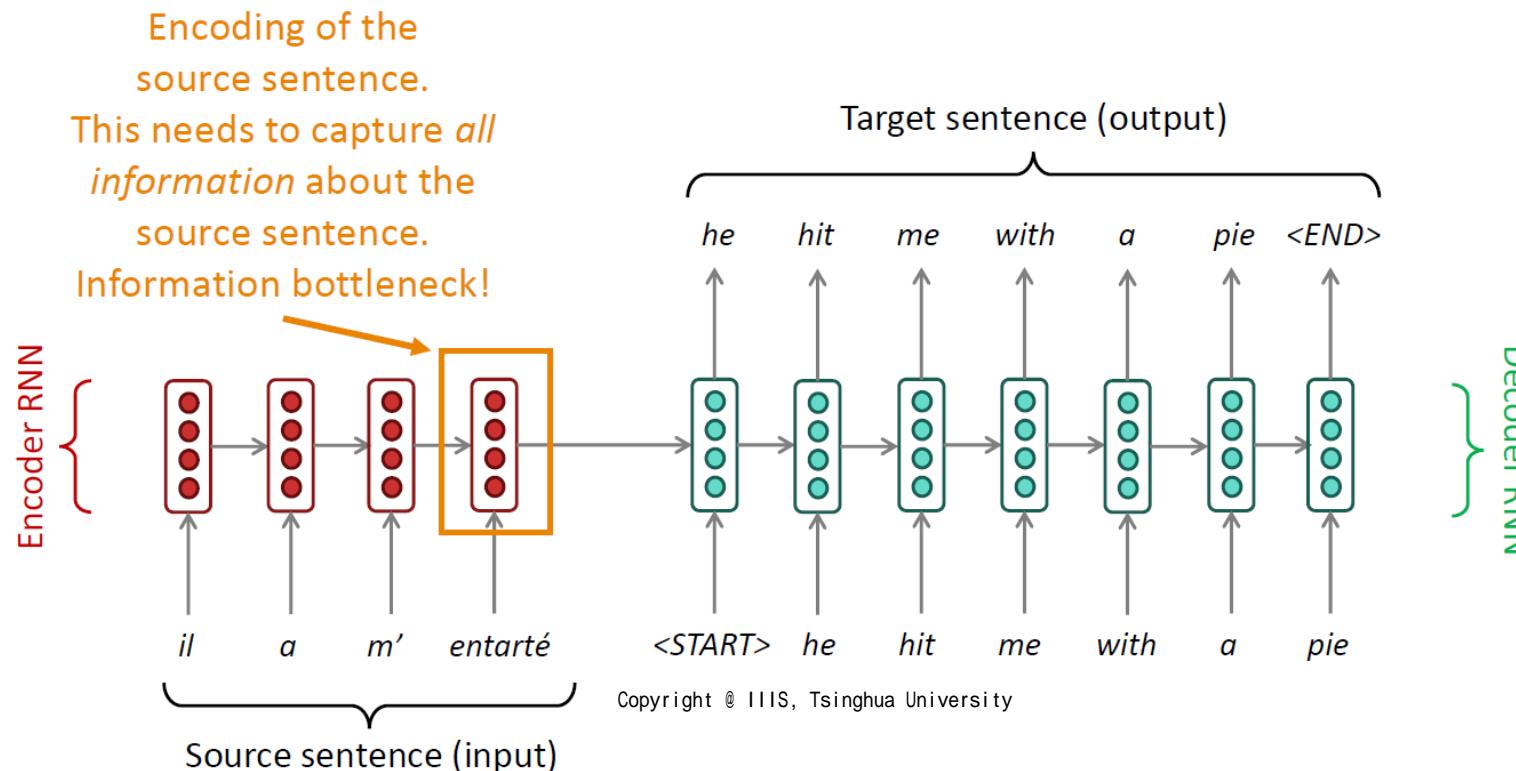
Sequence to Sequence Model

- Issue in the vanilla Seq2Seq model
 - Alignment: the word-level correspondence between X and Y
 - There are complex long-term dependencies



Sequence to Sequence Model

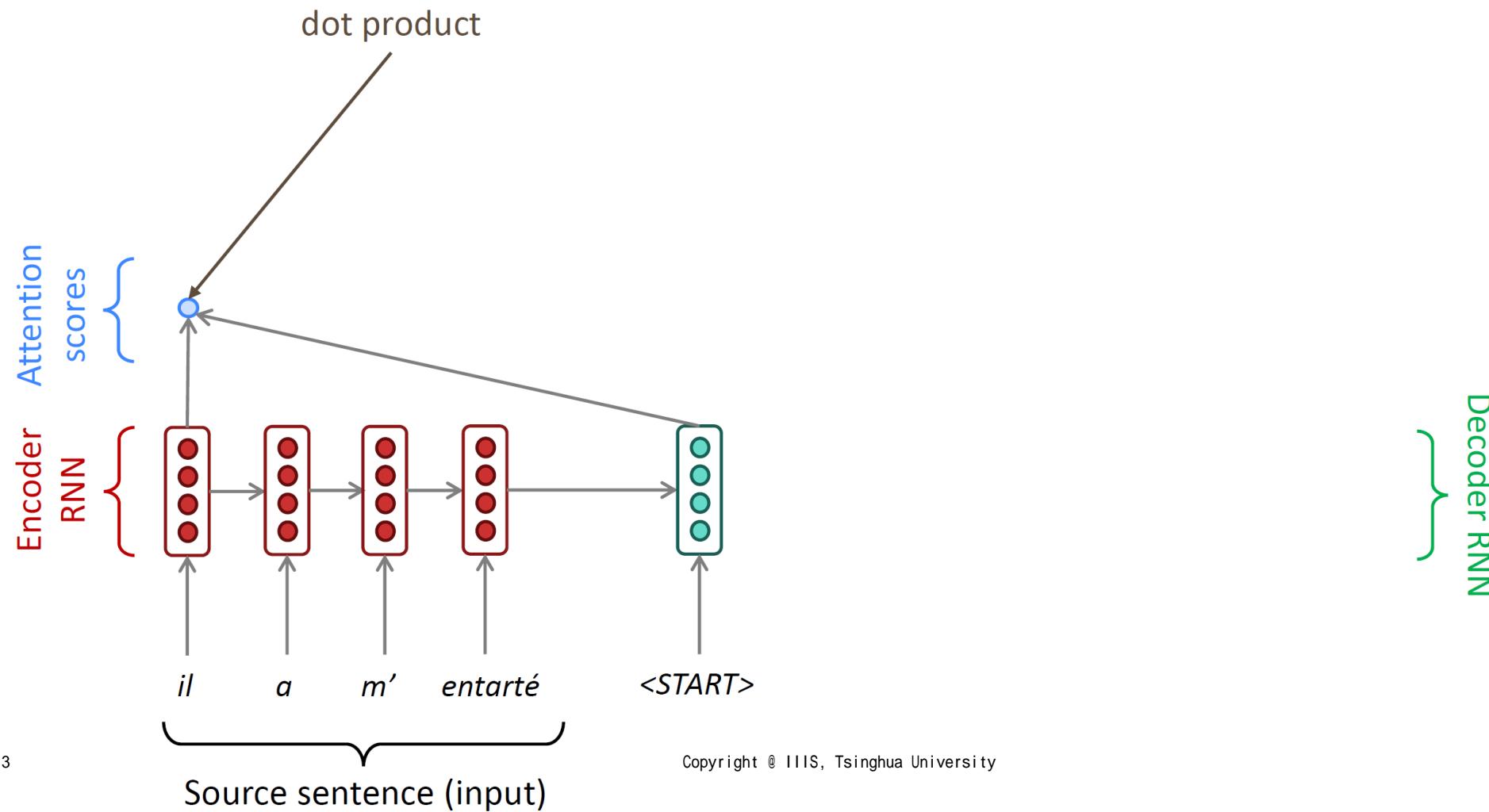
- Issue in the vanilla Seq2Seq model
 - The information bottleneck due to h
 - We want each Y_t to also focus on X_i that it is aligned with



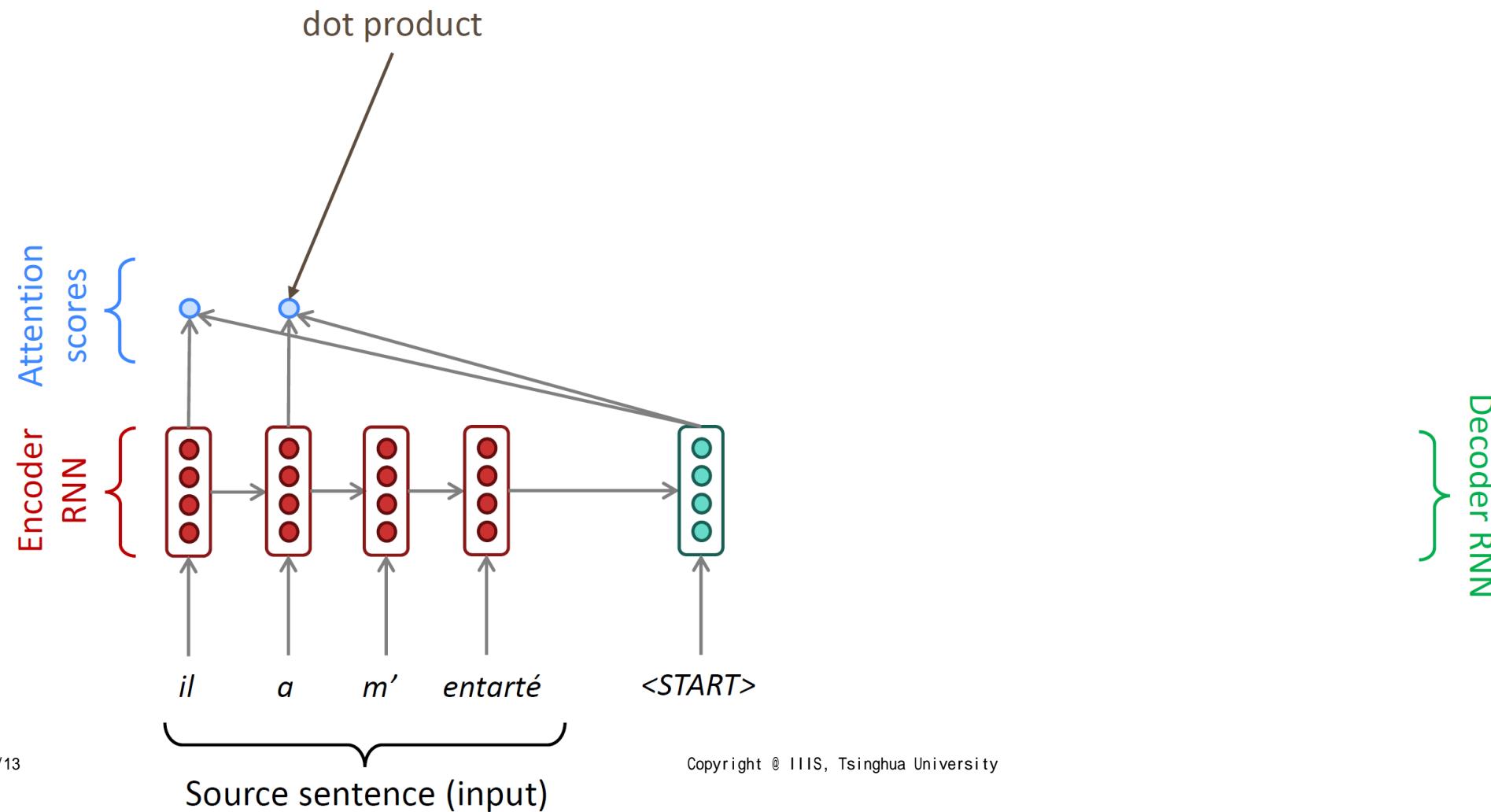
Sequence-to-Sequence with Attention

- NMT by Jointly Learning to Align and Translate
 - Bahdanau, Cho & Bengio, ICLR 2015 (38.5k citation)
 - Core idea:
 - When decoding Y_t , we consider both hidden states and alignments
 - Hidden: h_{t-1} from $Y_{i < t}$, i.e., $h_{t-1} = f_{dec}(Y_{i < t})$
 - Alignment: a direction connection to “key” words from X
 - Which part of X to focus?
 - Learn a softmax weight over X (attention distribution P_{att})
 - $P_{att}(X_i | h_{t-1})$: how much attention you want to put on word X_i
 - attention output $h_{att} = \sum_i f_{enc}(X_i | X_{j < i}) \cdot P_{att}(X_i | h_{t-1})$
 - Use h_{t-1} and h_{att} to compute Y_t
 - Let’s go through the diagram before showing more details

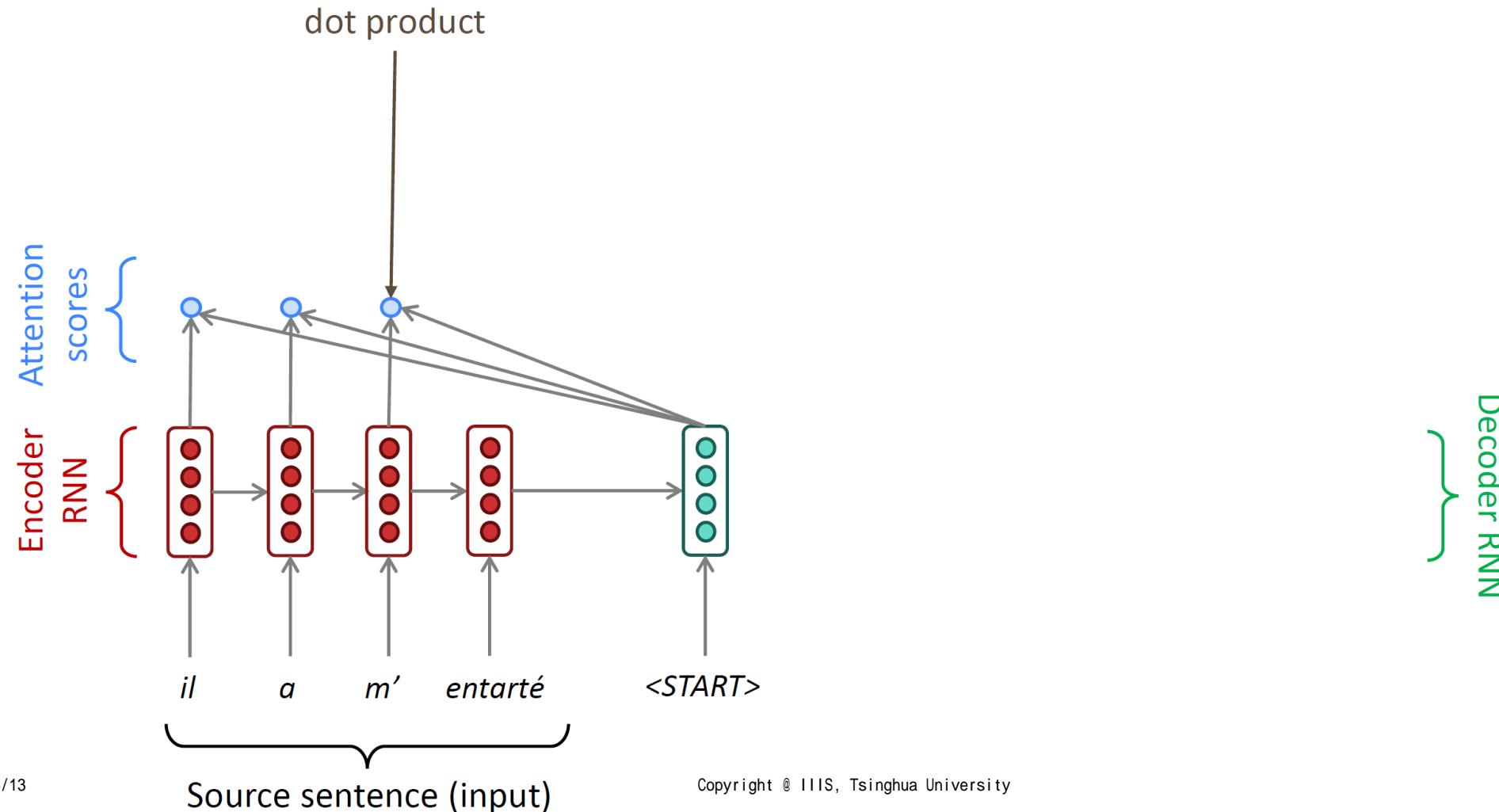
Sequence-to-Sequence with Attention



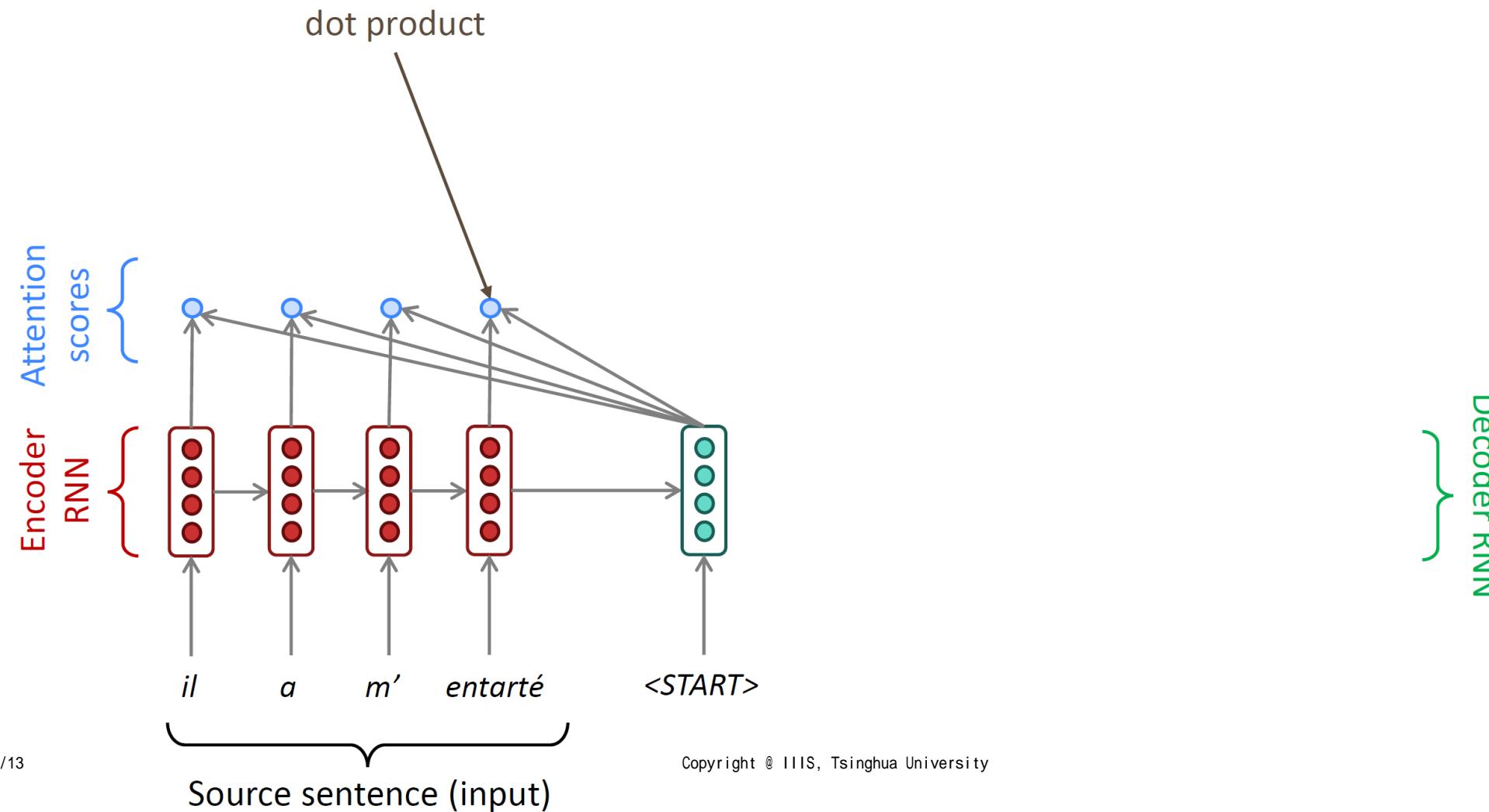
Sequence-to-Sequence with Attention



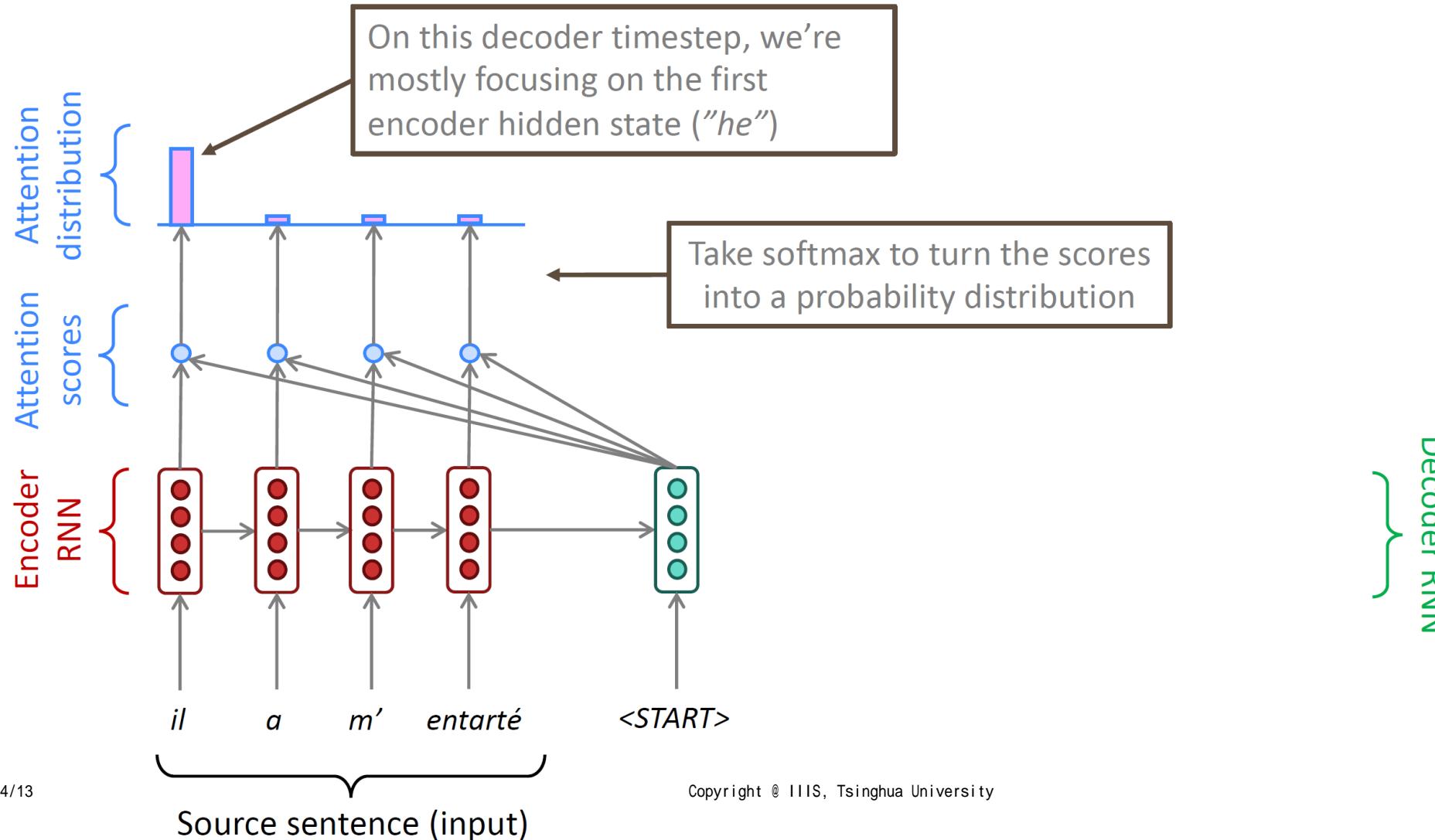
Sequence-to-Sequence with Attention



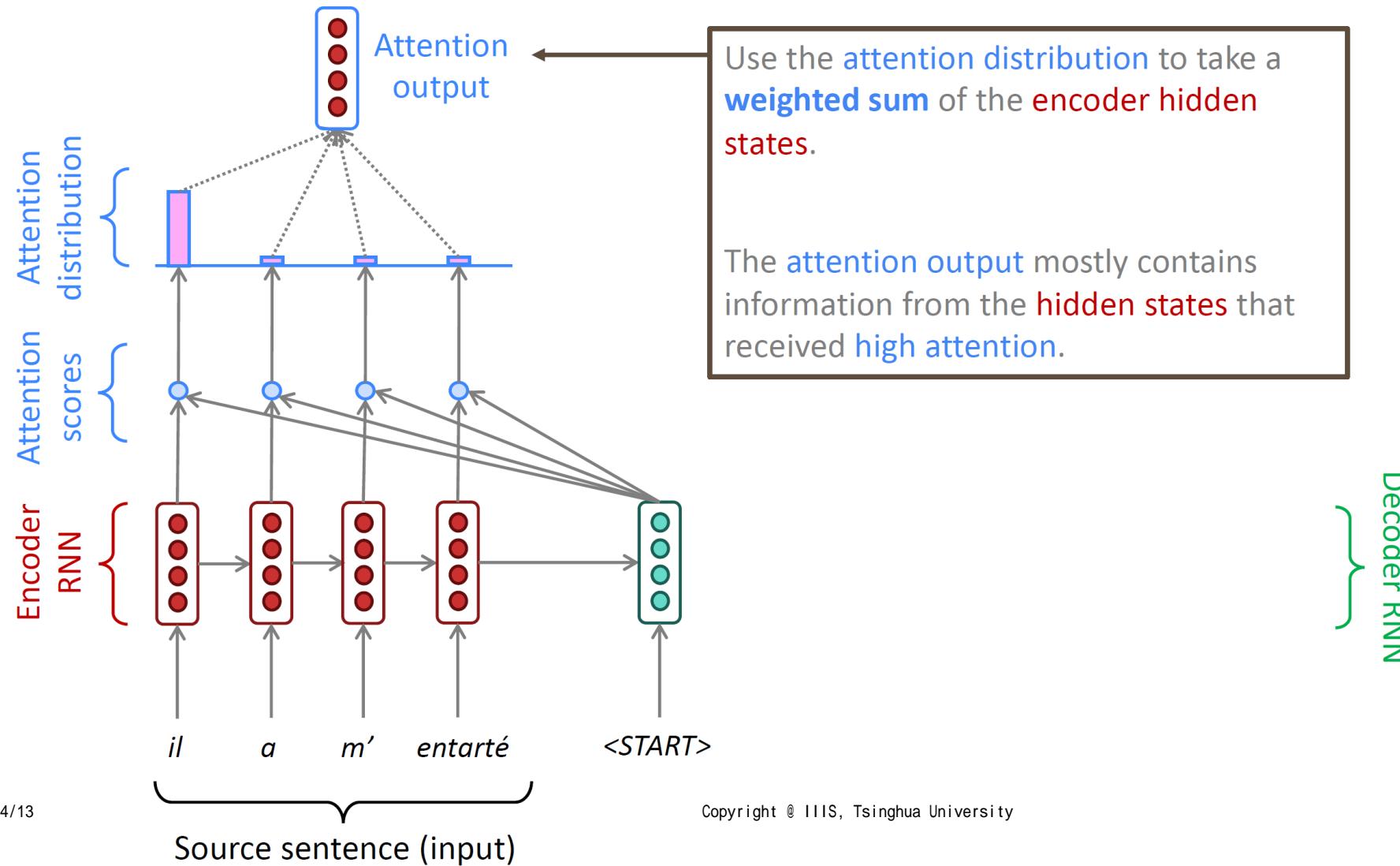
Sequence-to-Sequence with Attention



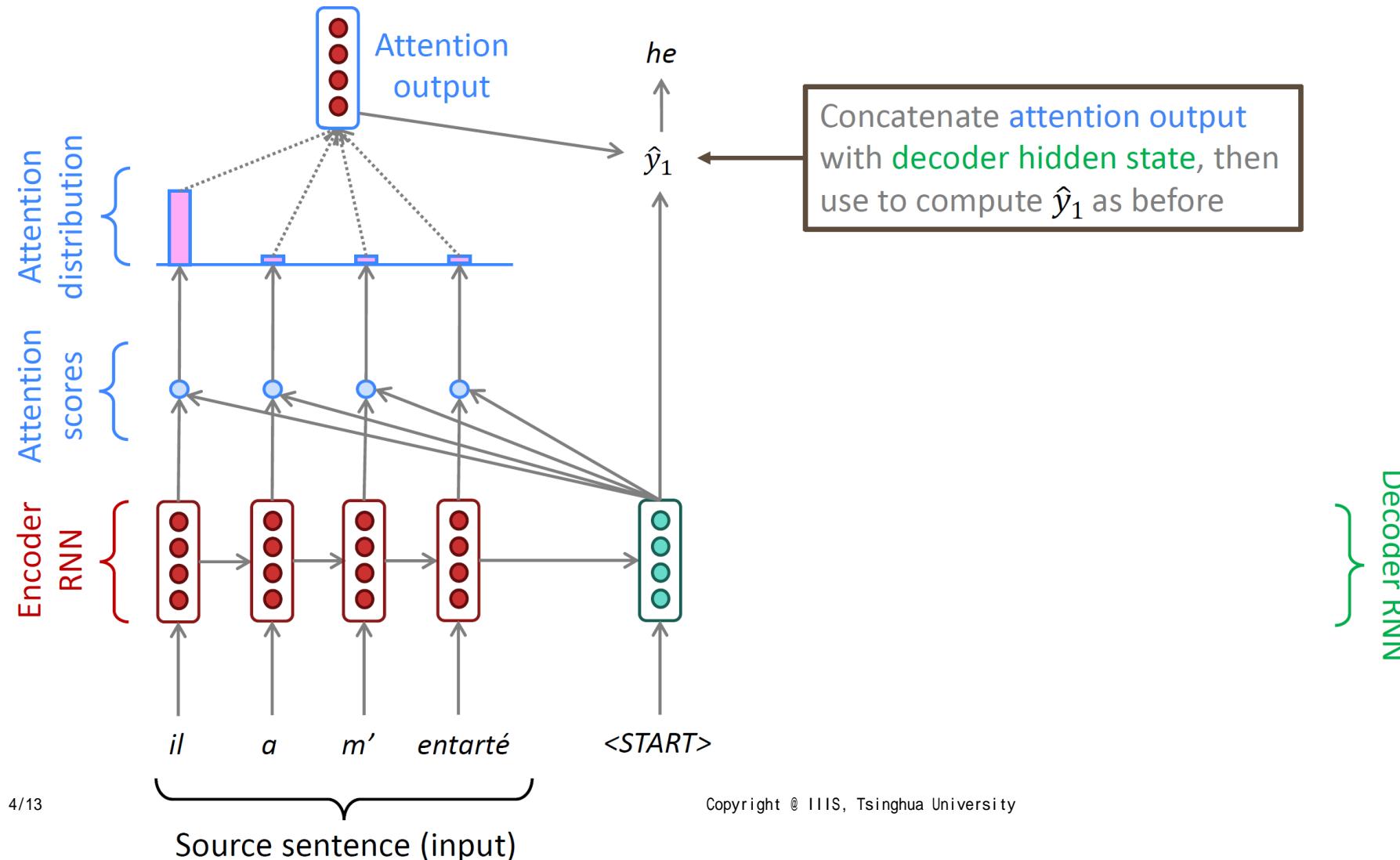
Sequence-to-Sequence with Attention



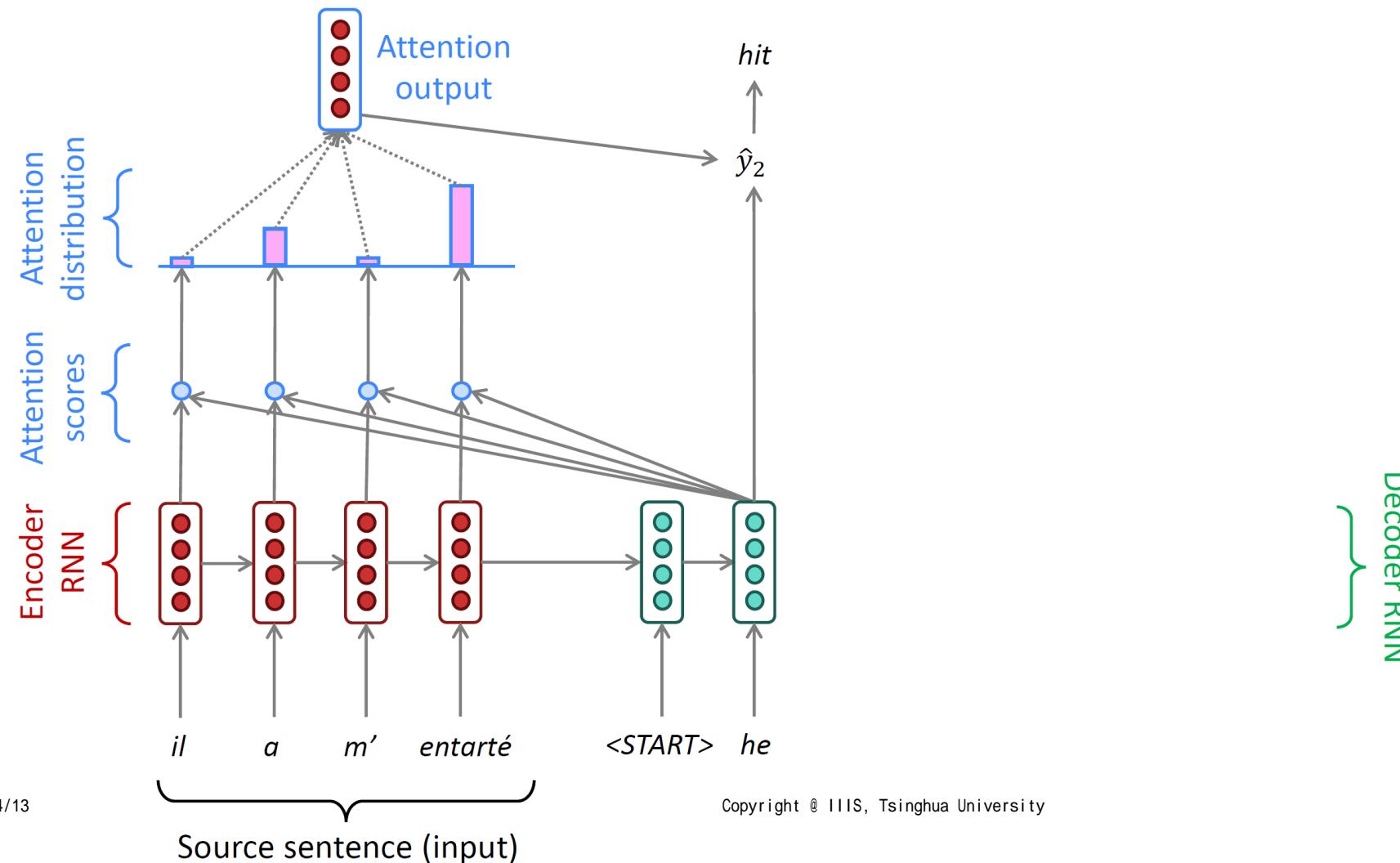
Sequence-to-Sequence with Attention



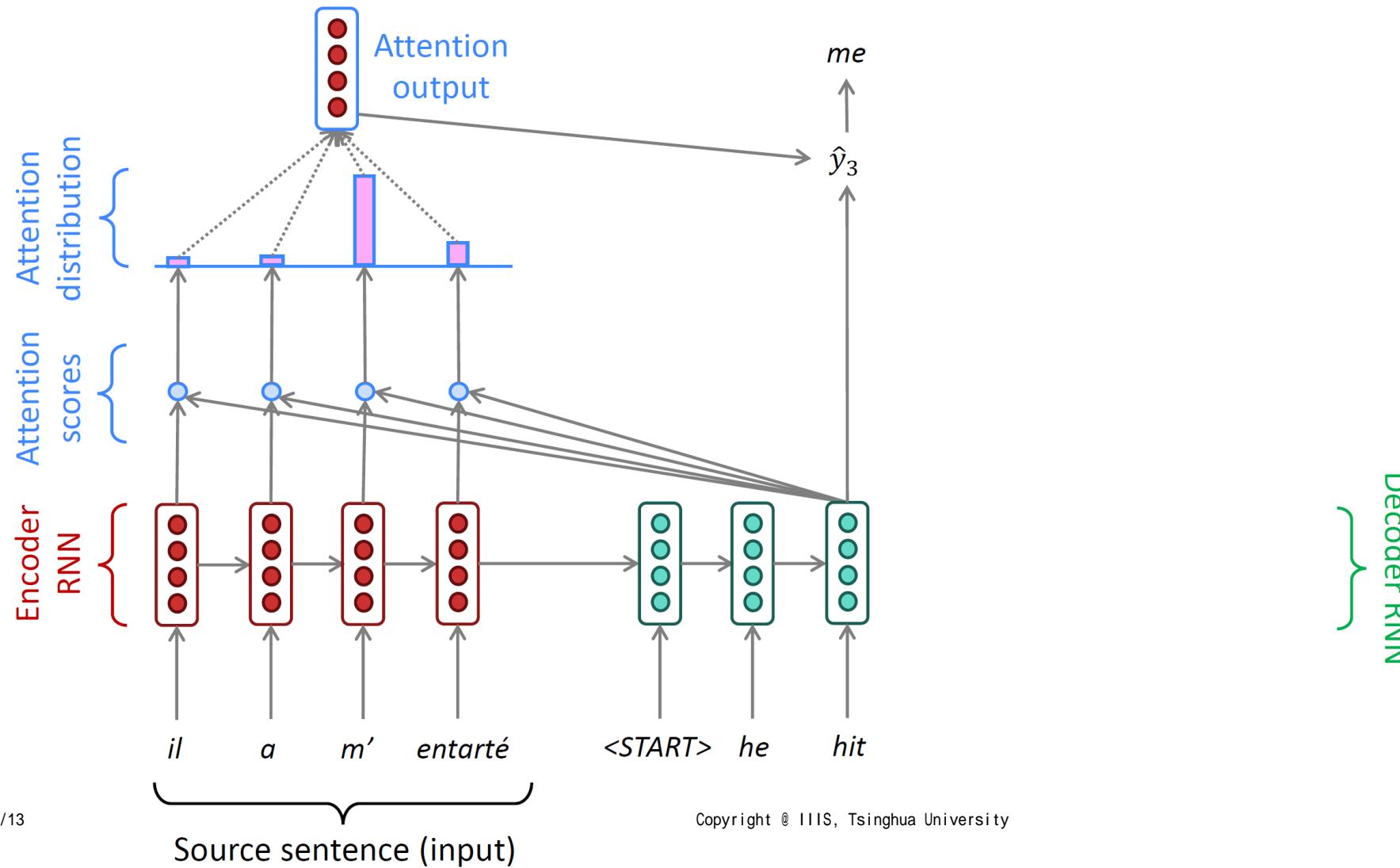
Sequence-to-Sequence with Attention



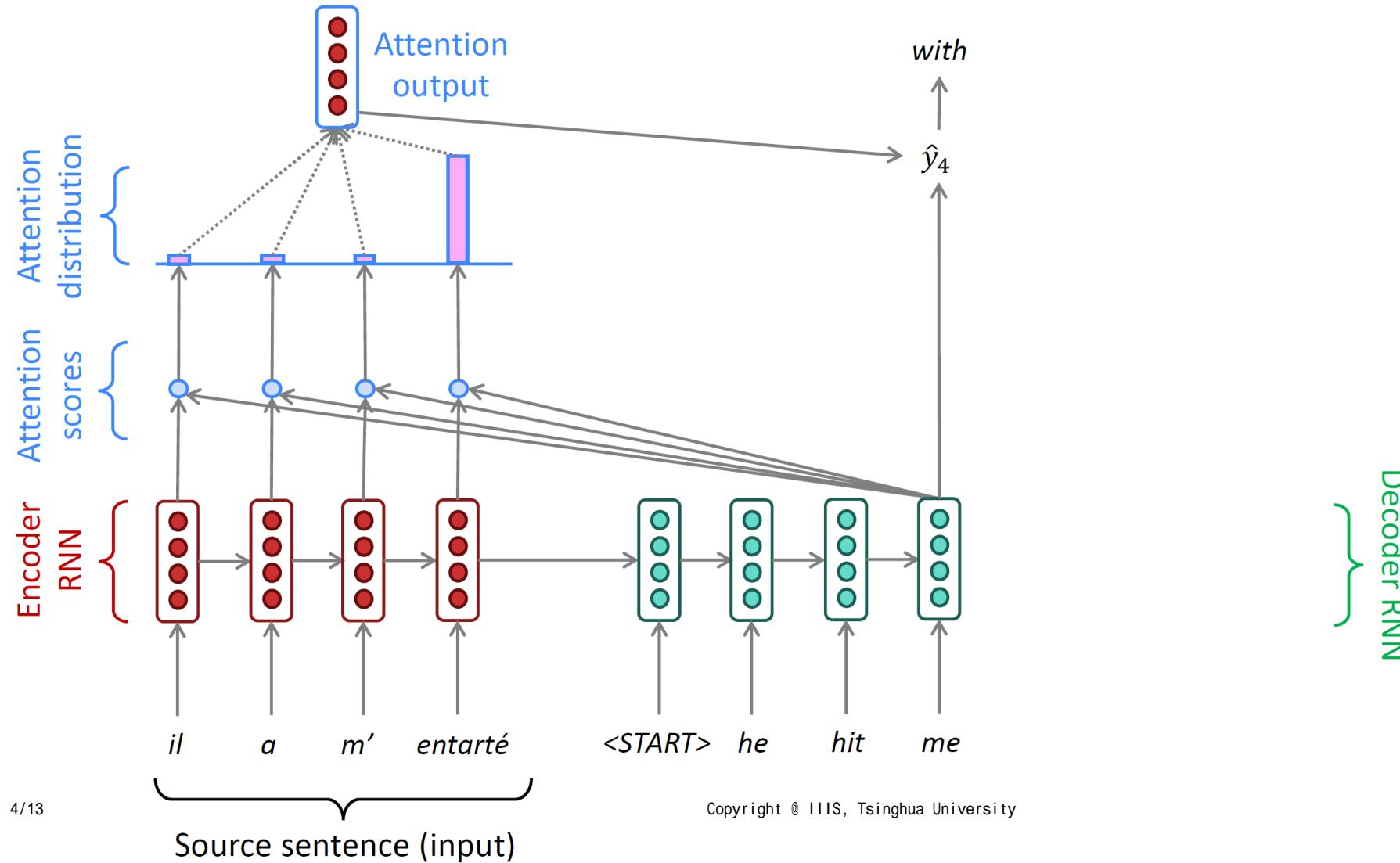
Sequence-to-Sequence with Attention



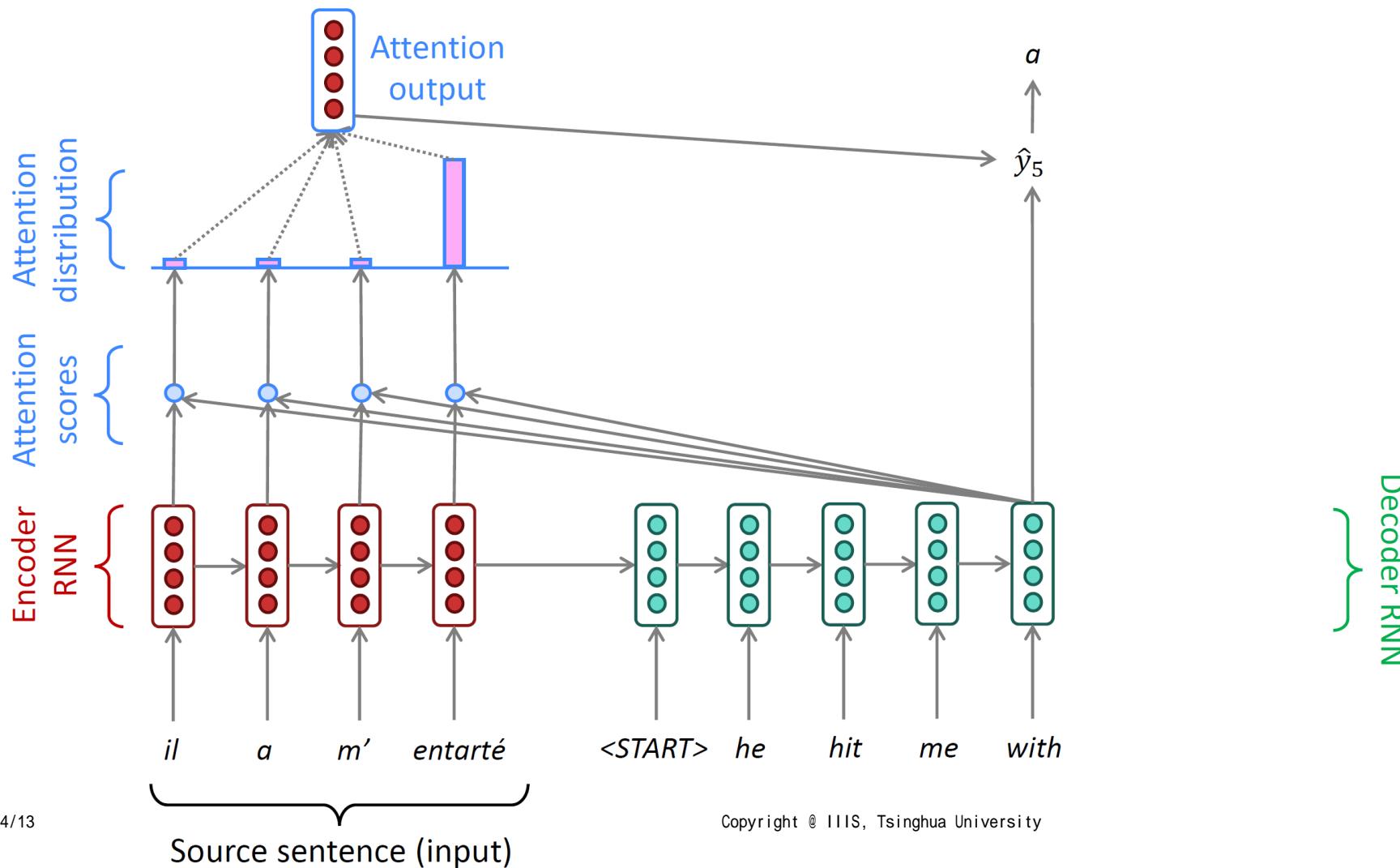
Sequence-to-Sequence with Attention



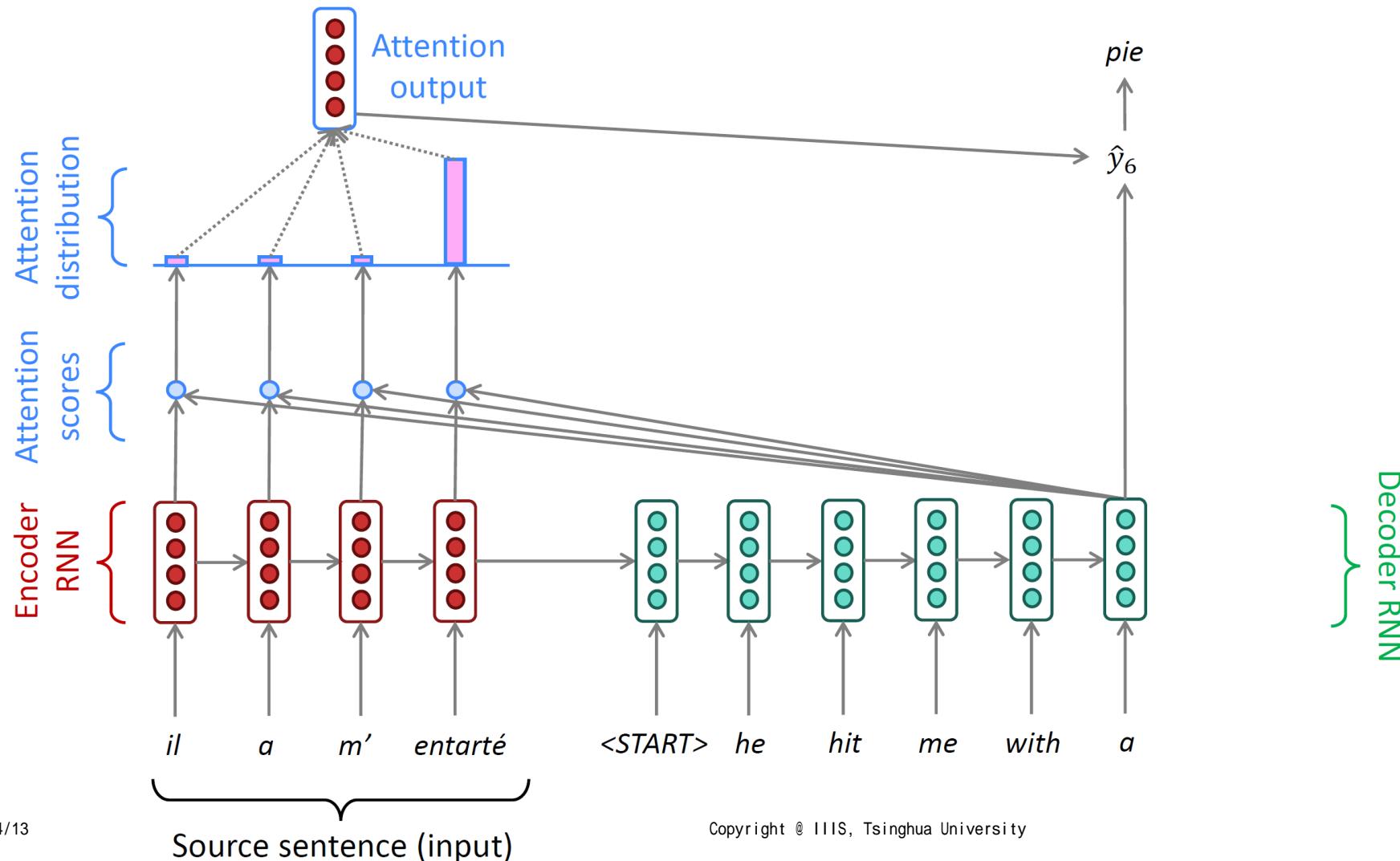
Sequence-to-Sequence with Attention



Sequence-to-Sequence with Attention



Sequence-to-Sequence with Attention



Sequence-to-Sequence with Attention

- Attention in equations

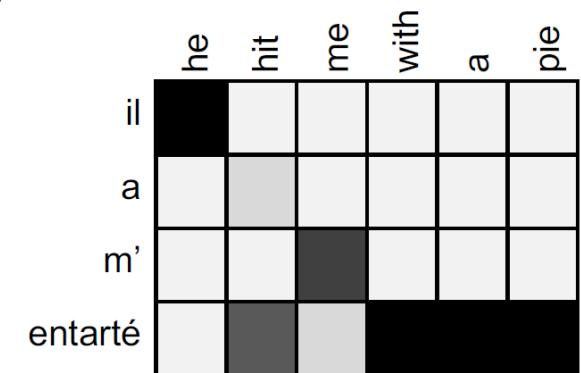
- Input sequence X and encoder f_{enc} and decoder f_{dec}
- $f_{enc}(X)$ produces hidden states $h_1^{enc}, h_2^{enc}, \dots, h_N^{enc}$
- On timestep t , we have decoder hidden state h_t
- Attention score $e_i = h_t^T h_i^{enc}$
- Attention distribution $\alpha_i = P_{att}(X_i) = \text{softmax}(e_i)$
- Attention output

$$h_{att}^{enc} = \sum_i \alpha_i h_i^{enc}$$

- $Y_t \sim g(h_t, h_{att}^{enc}; \theta)$
 - Sample output using both h_t and h_{att}^{enc}

Attention

- Attention is great!
 - It significantly improves NMT!
 - It solves the bottleneck problem and long-term dependency issue
 - Also helps gradient vanishing problem
 - It provides some interpretability
 - We can understand the focus of RNN decoder
- Attention is a general technique
 - Given a set of vector values V_i and a vector query q
 - Attention computes a weighted sum of values depending on q
- Attention can learn a representation of an arbitrary set of vectors $\{v_i\}$ depending on query q



Attention

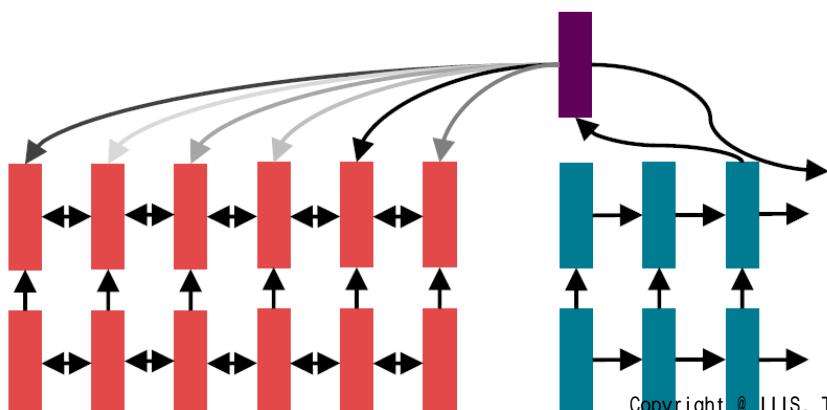
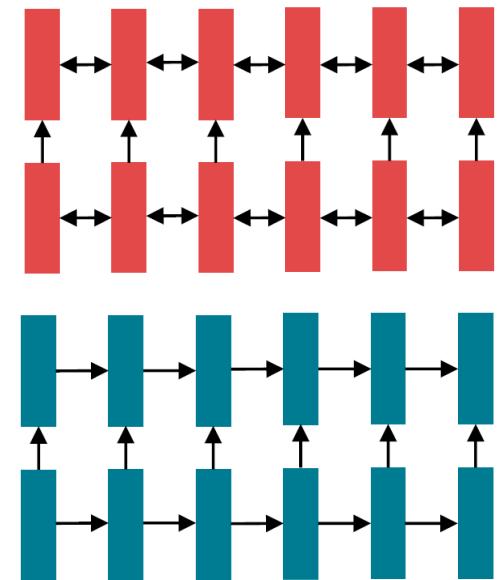
- Attention can learn a representation of an arbitrary set of vectors $\{v_i\}$ depending on query q
 - $\alpha_i = \text{softmax}(f(v_i, q))$ (attention distribution)
 - $v_{att} = \sum_i \alpha_i v_i$ (attention output)
 - Attention is *size-invariant* and *order-invariant*
- More use cases
 - E.g., a representation of a set of points (Pointer network, NIPS2015 & Deep Sets, NIPS2017)
 - E.g., include non-local information in CNN (Non-local network, CVPR18; Self-Attention GAN, ICML 19; BigGAN, ICLR 19)

Attention

- Attention can learn a representation of an arbitrary set of vectors $\{v_i\}$ depending on query q
 - $\alpha_i = \text{softmax}(f(v_i, q))$ (attention distribution)
 - $v_{att} = \sum_i \alpha_i v_i$ (attention output)
- Attention Variants $f(v_i, q)$
 - Multiplicative attention: $f(v_i, q) = q^T W h_i$
 - W is a weight matrix
 - Additive attention: $f(v_i, q) = u^T \tanh(W_1 v_i + W_2 q)$
 - W_1, W_2 are weight matrices
 - u is a weight vector
 - Expressiveness v.s. efficiency

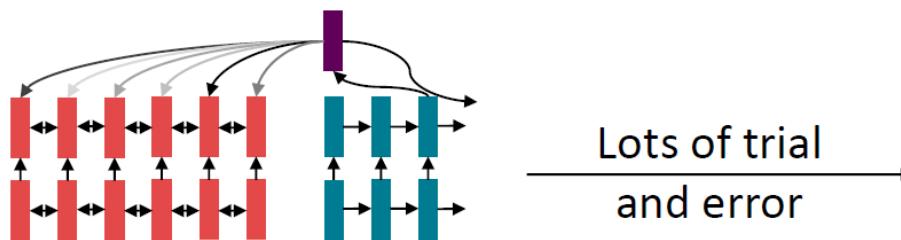
Sequence to Sequence Model with Attention

- Attention-Based Seq2Seq Model
 - Use bidirectional LSTMs as your encoder for input data
 - Use stacked LSTMs as your decoder for output data
 - Use attention for long-term dependencies



Sequence to Sequence Model with Attention

- Attention-Based Seq2Seq Model
 - Use bidirectional LSTMs as your encoder for input data
 - Use stacked LSTMs as your decoder for output data
 - Use attention for long-term dependencies
- Most NLP applications!
 - till 2017 😊

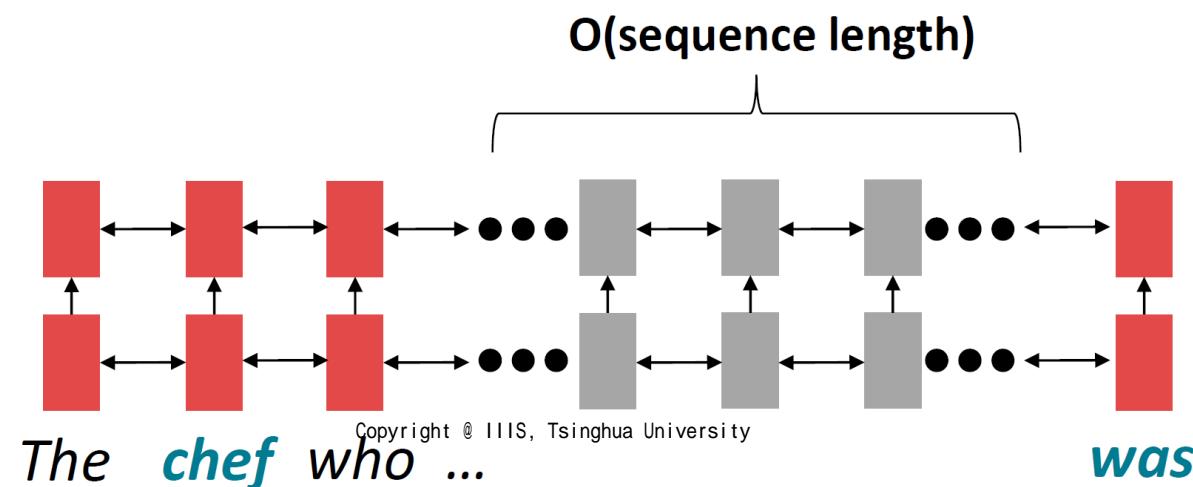


2014-2017ish

Copyright © Tsinghua University

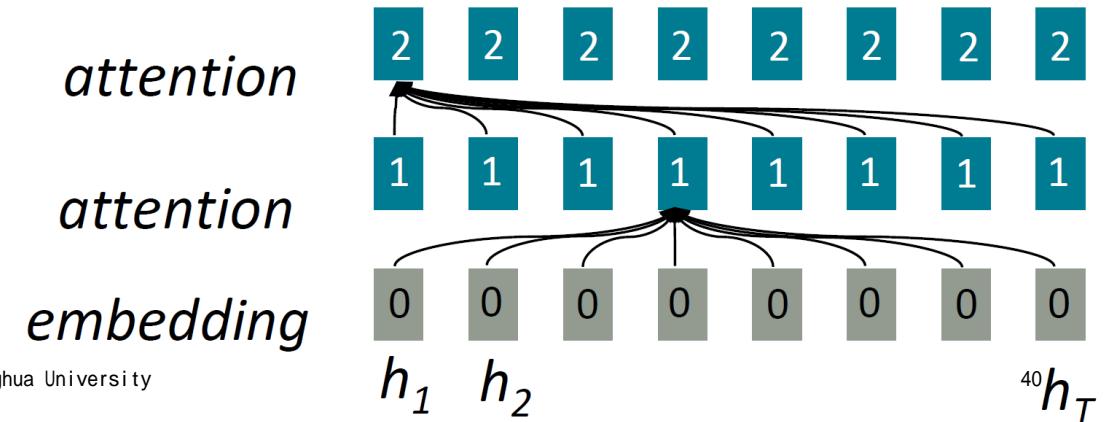
Sequence to Sequence Model with Attention

- Story So Far
 - RNN Models
 - Simple & Generic solution for sequence modeling
 - Issue for long-term dependencies
 - Linear computations for distant words through a single latent state
 - Lack of parallelization
 - Forced sequential computation (contrast with CNN)



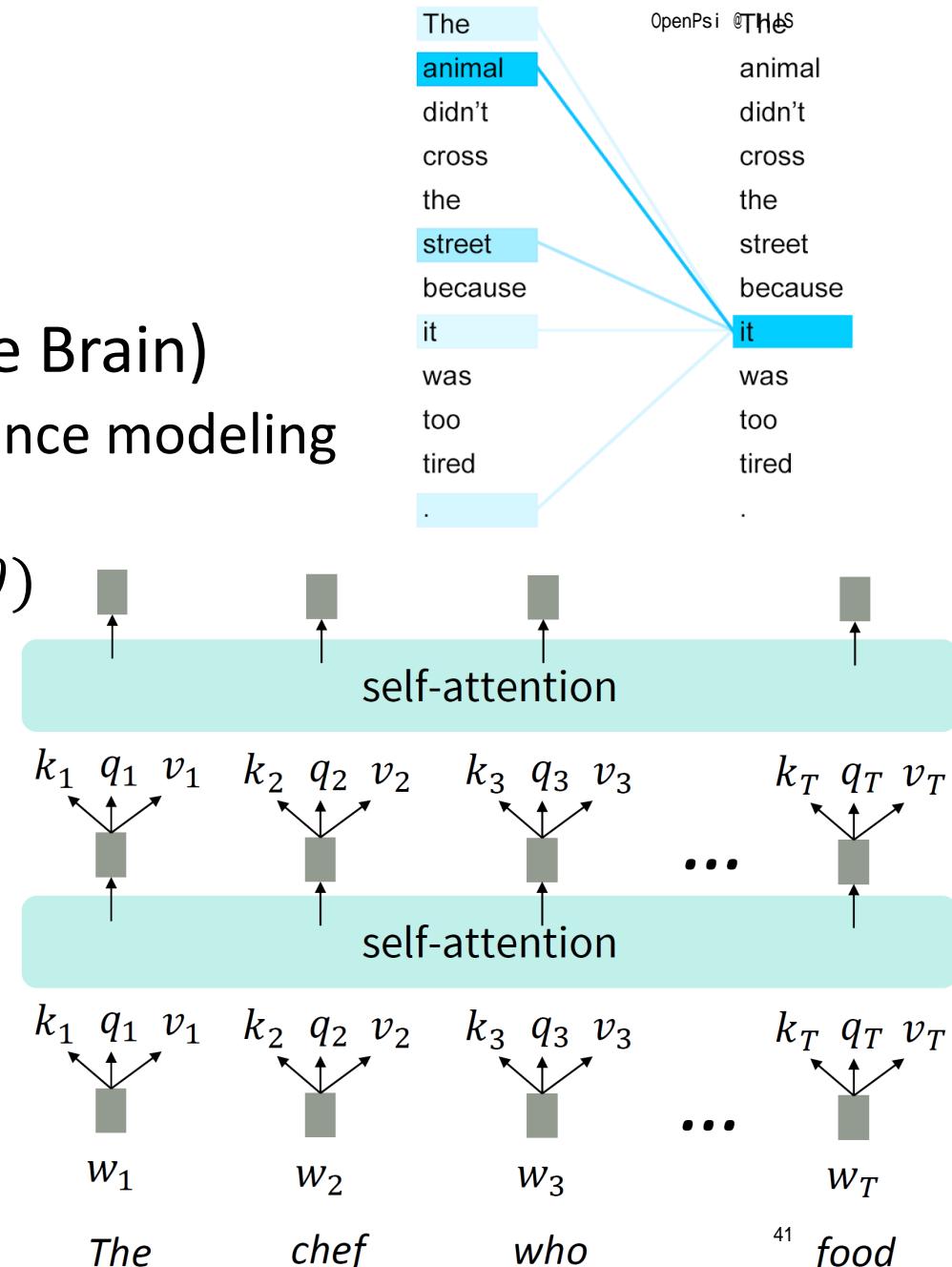
Sequence to Sequence Model with Attention

- Story So Far
 - RNN Models
 - Simple & Generic solution for sequence modeling
 - Issue for long-term dependencies
 - Linear computations for distant words through a single latent state
 - Lack of parallelization
 - Forced sequential computation (contrast with CNN)
 - Attention
 - Direct connection to distant words
 - $O(N)$ computation but perfectly parallel!
 - **Attention is all we need?**



Self-Attention

- Attention is all you need (NIPS2017, Google Brain)
 - A purely attention-base architecture for sequence modeling
 - NO RNN at all
 - Basic component: Self-Attention, $Y = f_{SA}(X; \theta)$
 - Core idea:
 - X_t attends on the entire X sequence
 - Y_t computed from X_t and the attention output
 - Equations for Y_t
 - Key k_t , value v_t , query q_t from X_t
 - $k_t, v_t, q_t = g_1(X_t; \theta)$
 - Attention distribution $\alpha_{t,j} = \text{softmax}(q_t^T k_j)$
 - Attention output $out_t = \sum_j \alpha_{t,j} v_j$
 - $Y_t = g_2(out_t; \theta)$
- Issues of self-attention?



Self-Attention

- Issues of Vanilla Self-Attention
 - Notion of sequence order
 - Attention is order-invariant
 - Lack of non-linearities
 - All the weights are simple linear weighted average
 - Capability of autoregressive modelling
 - In generation tasks, the model cannot “look at the future”
 - E.g., text generation
 - Y_t can only depend on $X_{i < t}$
 - Vanilla self-attention focuses on the entire sequence

Self-Attention

- Issues of Vanilla Self-Attention
 - Notion of sequence order
 - Attention is order-invariant
 - Lack of non-linearities
 - All the weights are simple linear weighted average
 - Capability of autoregressive modelling
 - In generation tasks, the model cannot “look at the future”
 - E.g., text generation
 - Y_t can only depend on $X_{i < t}$
 - Vanilla self-attention focuses on the entire sequence

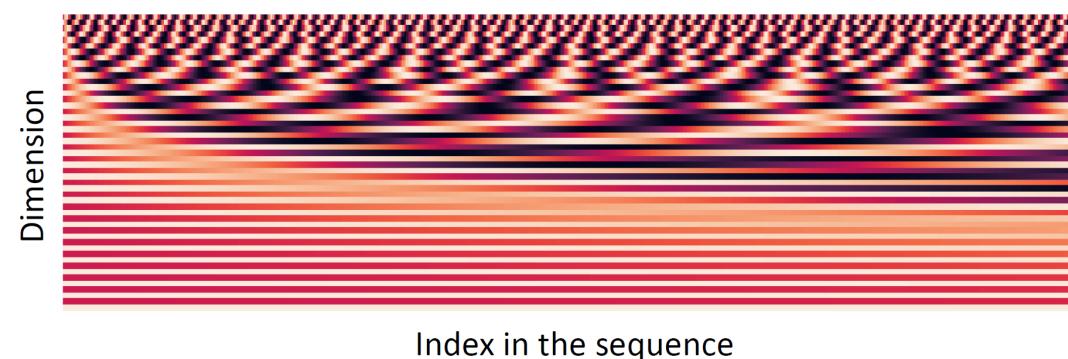
Self-Attention

- Notion of Sequence Ordering
 - Vanilla attention
 - $\tilde{\alpha}_{i,j} = \text{softmax}(\tilde{q}_i^T \tilde{k}_j)$; $\text{out}_i = \sum_j \tilde{\alpha}_{i,j} \tilde{v}_j$
 - $\tilde{k}_t, \tilde{v}_t, \tilde{q}_t = g_1(X_t)$, do not contain position information
 - Idea: position encoding
 - p_i : an embedding vector of position i
 - $k_t, v_t, q_t = g_1([X_t, p_t])$ include position features
 - Practical remark:
 - Additive can be sufficient: $k_t \leftarrow \tilde{k}_t + p_t, q_t \leftarrow \tilde{q}_t + p_t, v_t \leftarrow \tilde{v}_t + p_t$
 - p_t is typically only included in the first layer
 - **How to design p_i ?**
 - Note that the length of a sequence can be long

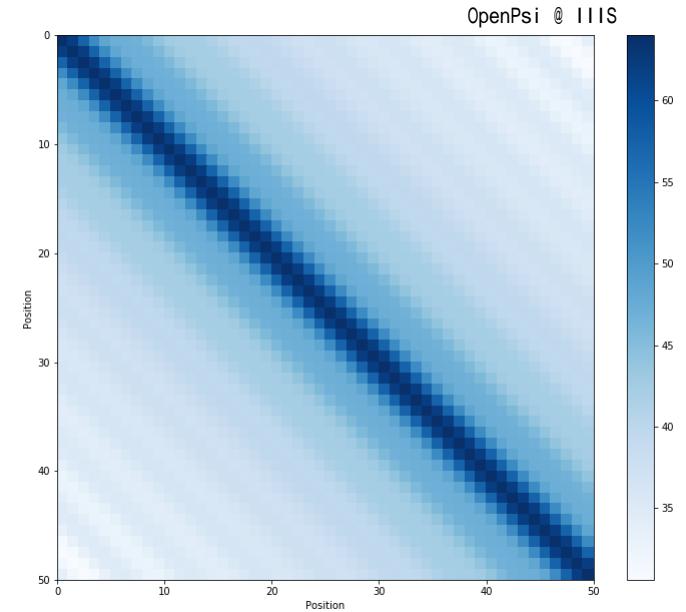
Self-Attention

- Notion of Sequence Ordering
 - Idea: position encoding p_i
 - $\tilde{k}_t, \tilde{v}_t, \tilde{q}_t = g_1(X_t)$, do not contain position information
 - Design of p_t
 - **Sinusoidal position representation**
 - Concatenate sinusoidal functions of varying periods

$$p_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$



- Pros: simple, naturally modelling “relative position”, easily applied to long sequences
- Cons: not learnable; generalization poorly to sequences longer than training data



Heatmap of $p_i^T p_j$

Self-Attention

- Notion of Sequence Ordering
 - Idea: position encoding p_i
 - $\tilde{k}_t, \tilde{v}_t, \tilde{q}_t = g_1(X_t)$, do not contain position information
 - Design of p_t
 - Sinusoidal position representation
 - **Learned absolute representation**
 - Let p_t become a learned parameter vector!
 - Assume maximum length L , learn a matrix $p \in \mathbb{R}^{d \times T}$, p_t is a column of p
 - A popular choice in practice!
 - Pros:
 - Flexible and learnable, more powerful
 - Cons:
 - Assume a fixed maximum length L , does not work at all for length above L

Self-Attention

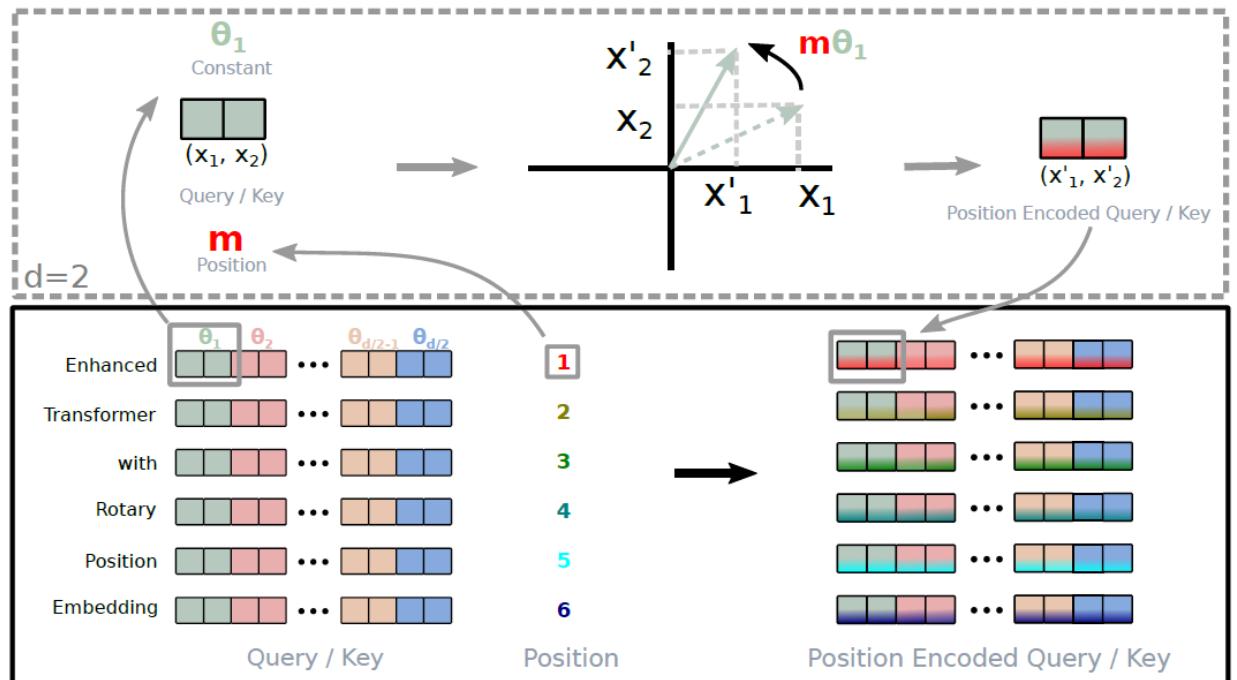
- Notion of Sequence Ordering
 - Idea: position encoding p_i
 - $\tilde{k}_t, \tilde{v}_t, \tilde{q}_t = g_1(X_t)$, do not contain position information
 - Design of p_t
 - Sinusoidal position representation
 - Learned absolute representation
 - **Relative position representation (ACL2018, Google)**
 - When computing attention, relative distance is important!
 - $\alpha_{i,j} = \text{softmax}(q_i^T(k_j + p_{[i-j]}))$
 - $out_i = \sum_j \alpha_{i,j}(v_j + p_{[i-j]})$
 - Bounded relative distance $p_t = p_{\max(-k, \min(k, t))}$
 - Truncate $t < -k$ to k and $t > k$ to k
 - Pros: learned representation and extrapolate well; More powerful.
 - Cons: computation overhead (refer to the paper for implementation tricks)

Self-Attention

- Notion of Sequence Ordering
 - Idea: position encoding p_i
 - $\tilde{k}_t, \tilde{v}_t, \tilde{q}_t = g_1(X_t)$, do not contain p
 - Design of p_t
 - Sinusoidal position representation
 - Learned absolute representation
 - Relative position representation
 - Rotary position embedding (RoPE, Roformer, 2021)**
 - Relative position but factored computation

$$R_{\Theta,m}^d = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix}$$

Copyright © IIS, Tsinghua University
4/13



$$\Theta = \{\theta_i = 10000^{-2(i-1)/d}, i \in [1, 2, \dots, d/2]\}$$

$$q_m^\top k_n = (R_{\Theta,m}^d W_q x_m)^\top (R_{\Theta,n}^d W_k x_n) = x_m^\top W_q R_{\Theta,n-m}^d W_k x_n$$

$$R_{\Theta,n-m}^d = (R_{\Theta,m}^d)^\top R_{\Theta,n}^d$$

Remark:

- Compatible with any dimension and length
- Fast computation
- Effective in practice

Self-Attention

- Issues of Vanilla Self-Attention
 - Notion of sequence order
 - Solution: position encoding
 - Lack of non-linearities
 - All the weights are simple linear weighted average
 - Capability of autoregressive modelling
 - In generation tasks, the model cannot “look at the future”
 - E.g., text generation
 - Y_t can only depend on $X_{i < t}$
 - Vanilla self-attention focuses on the entire sequence

Self-Attention

- Issues of Vanilla Self-Attention
 - Notion of sequence order
 - Solution: position encoding
 - **Lack of non-linearities**
 - All the weights are simple linear weighted average
 - Capability of autoregressive modelling
 - In generation tasks, the model cannot “look at the future”
 - E.g., text generation
 - Y_t can only depend on $X_{i < t}$
 - Vanilla self-attention focuses on the entire sequence

Self-Attention

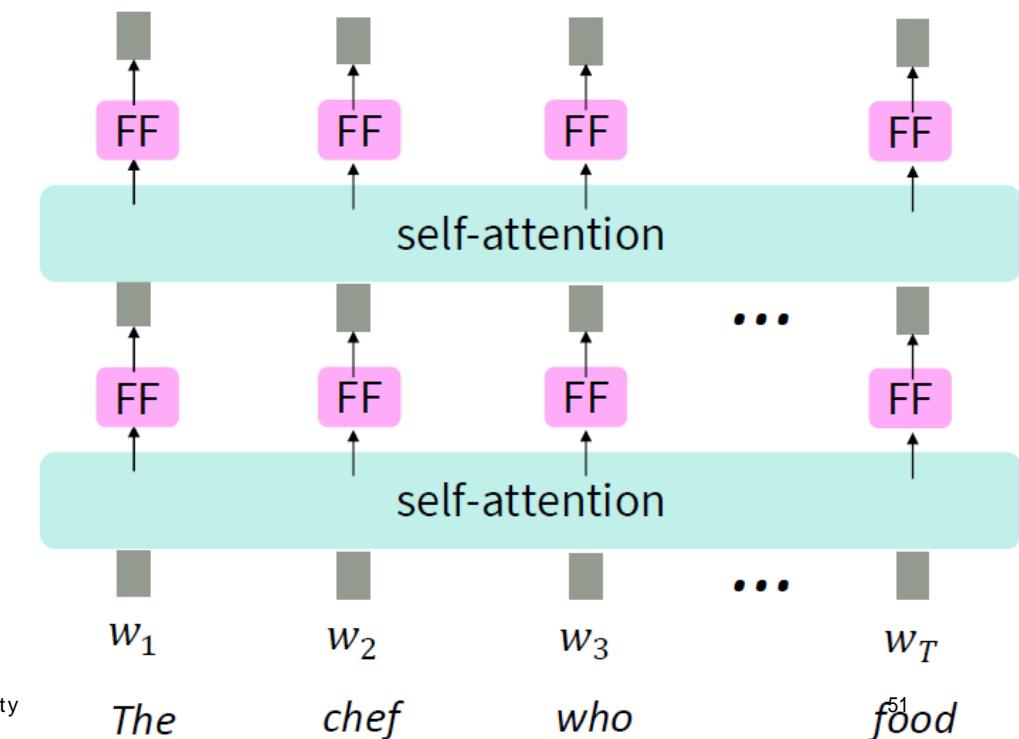
- Combine nonlinearities in self-attention

- Vanilla self-attention

- No element-wise activation functions (e.g., ReLU, tanh)
- Only weighted average and softmax operators
 - Essentially linear transformations of inputs

- Easy fix:

- Add an MLP to process out_i
- $m_i = MLP(out_i)$
- $= W_2 \cdot \text{ReLU}(W_1 \cdot out_i + b_1) + b_2$
- Remark
 - we do not put activation layer before softmax



Self-Attention

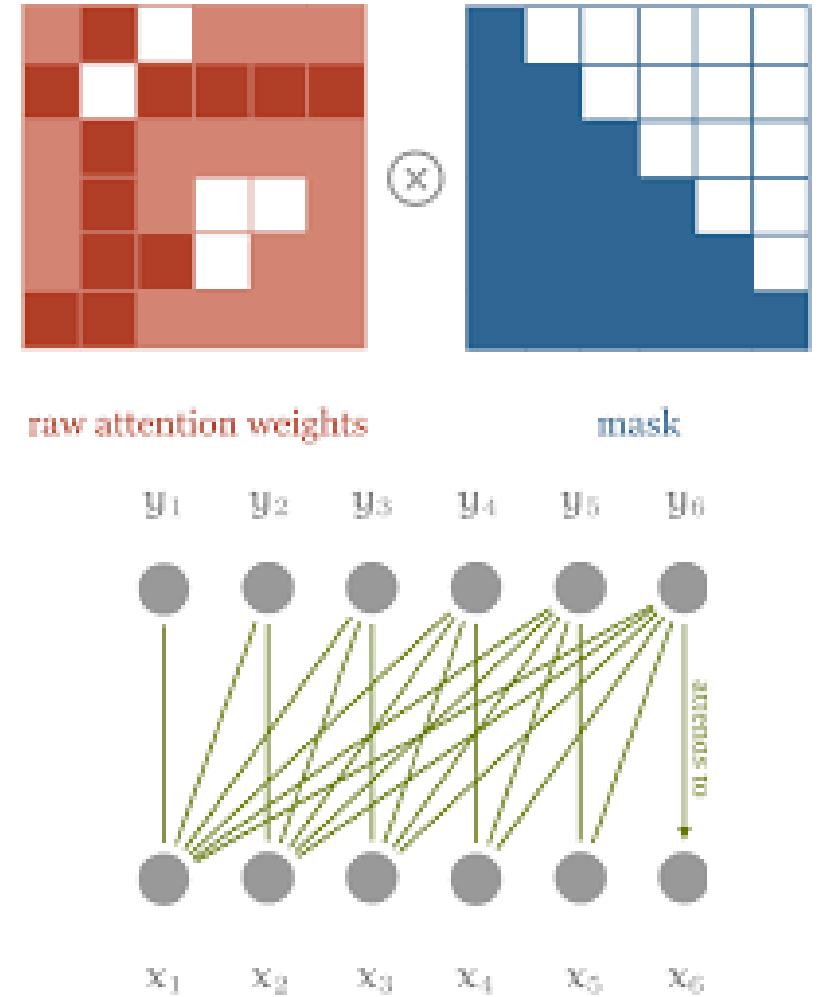
- Issues of Vanilla Self-Attention
 - Notion of sequence order
 - Solution: position encoding
 - **Lack of non-linearities**
 - Solution: post-processing MLP layer
 - Capability of autoregressive modelling
 - In generation tasks, the model cannot “look at the future”
 - E.g., text generation
 - Y_t can only depend on $X_{i < t}$
 - Vanilla self-attention focuses on the entire sequence

Self-Attention

- Issues of Vanilla Self-Attention
 - Notion of sequence order
 - Solution: position encoding
 - Lack of non-linearities
 - Solution: post-processing MLP layer
 - **Capability of autoregressive modelling**
 - In generation tasks, the model cannot “look at the future”
 - E.g., text generation
 - Y_t can only depend on $X_{i < t}$
 - Vanilla self-attention focuses on the entire sequence

Self-Attention

- Autoregressive Modeling
 - In language mode decoder: model $P(Y_t | X_{i < t})$
 - out_t cannot look at future $X_{i > t}$
 - Naïve solution:
 - For each t , a varying for-loop only iterating over $i \leq t$
 - Varying for-loop for each t , parallelization unfriendly
 - Masked Attention
 - Compute $e_{i,j} = q_i^T k_j$ as usual (perfect parallel)
 - Mask out $e_{i>j}$ by setting $e_{i>j} = -\infty$ (perfect parallel)
 - $e \odot (1 - M) \leftarrow -\infty$; M is a fixed 0/1 mask matrix
 - Then compute $\alpha_i = \text{softmax}(e_i)$ (perfect parallel)
 - Remark:
 - $M = 1$ for full self-attention
 - Set M for arbitrary dependency ordering



Self-Attention

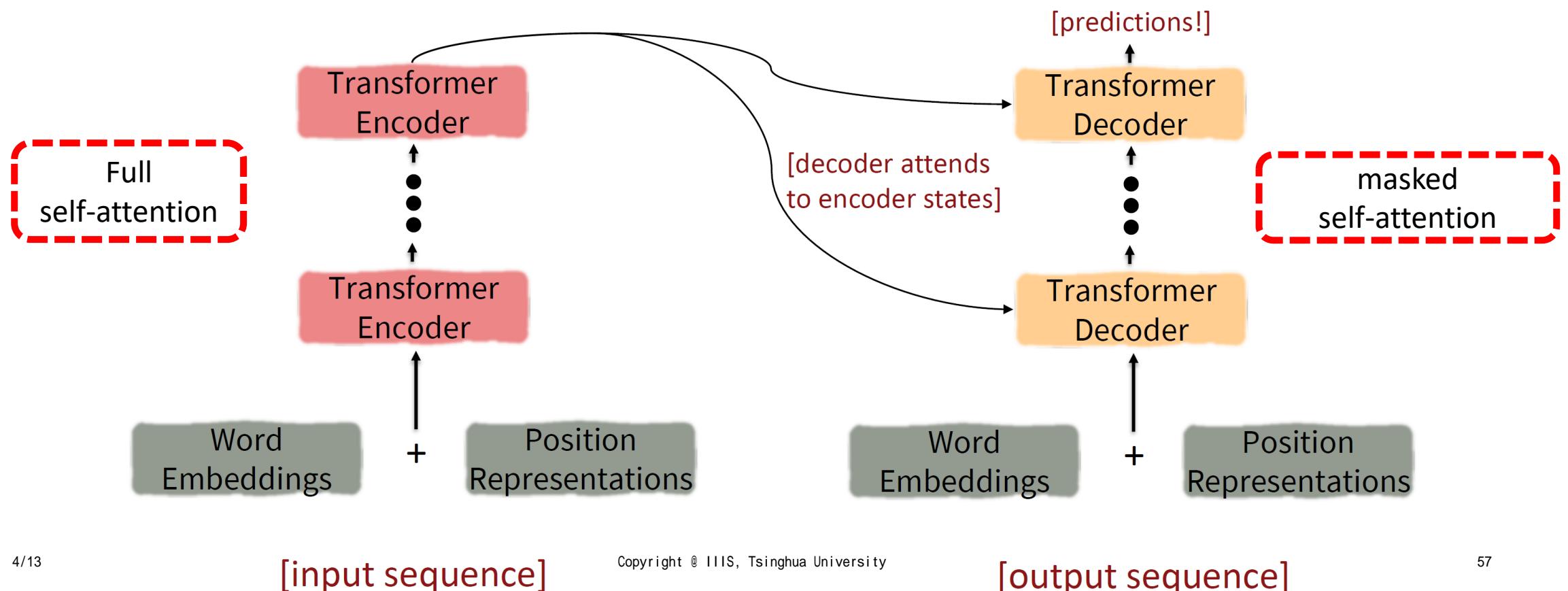
- Issues of Vanilla Self-Attention
 - Notion of sequence order
 - Solution: position encoding
 - Lack of non-linearities
 - Solution: post-processing MLP layer
 - **Capability of autoregressive modelling**
 - Solution: masked self-attention

Self-Attention

- Issues of Vanilla Self-Attention
 - Notion of sequence order
 - Solution: position encoding
 - Lack of non-linearities
 - Solution: post-processing MLP layer
 - Capability of autoregressive modelling
 - Solution: masked self-attention
- Basic building block for the famous “Transformer” model!
 - Attention is all you need (NIPS2017, Vaswani et al, Google)
 - Self-attention + **a few more other enhancements!**
 - A milestone: first pure attention-based model for effective sequence modeling
 - *Originally proposed for NMT; Soon dominates general sequence modeling problems*

Transformer Model

- Transformer-based sequence-to-sequence modeling



Transformer Model

- Transformer-based sequence-to-sequence modeling
 - Basic building blocks: masked self-attention
 - Enhancements
 - **Key-query-value attention**
 - Obtain q_t, v_t, k_t from X_t
 - $q_t = W^q X_t; v_t = W^v X_t; k_t = W^k X_t$ (position encoding omitted)
 - W^q, W^v, W^k are learnable weight matrices
 - $\alpha_{i,j} = \text{softmax}(q_i^T k_j); out_i = \sum_j \alpha_{i,j} v_j$
 - Intuition: key, query, and value can focus on different parts of input

All pairs of attention scores!

$$\begin{aligned}
 & XQ \quad K^T X^T \quad = \quad XQK^T X^T \in \mathbb{R}^{T \times T} \\
 & \text{softmax} \left(\begin{matrix} XQK^T X^T \\ XV \end{matrix} \right) \quad = \quad \text{output} \in \mathbb{R}^{T \times d}
 \end{aligned}$$

Transformer Model

- Transformer-based sequence-to-sequence modeling
 - Basic building blocks: masked self-attention
 - Enhancements
 - Key-query-value attention
 - **Multi-headed attention**
 - Standard attention → single-headed attention
 - $X_t \in \mathbb{R}^d, Q, K, V \in \mathbb{R}^{d \times d}$
 - We only “look at” a single position j with high $\alpha_{i,j}$
 - What if we want to look at different j for different reasons?
 - Idea: define h separate attention heads
 - h different attention distributions, keys, values and queries
 - $Q^l, K^l, V^l \in \mathbb{R}^{d \times \frac{d}{h}}$, for $1 \leq l \leq h$
 - $\alpha_{i,j}^l = \text{softmax}(q_i^{lT} k_j^l); out_i^l = \sum_j \alpha_{i,j}^l v_j^l$

#Params Unchanged!

Single-head attention
(just the query matrix)

$$X \quad Q = XQ$$

Multi-head attention
(just two heads here)

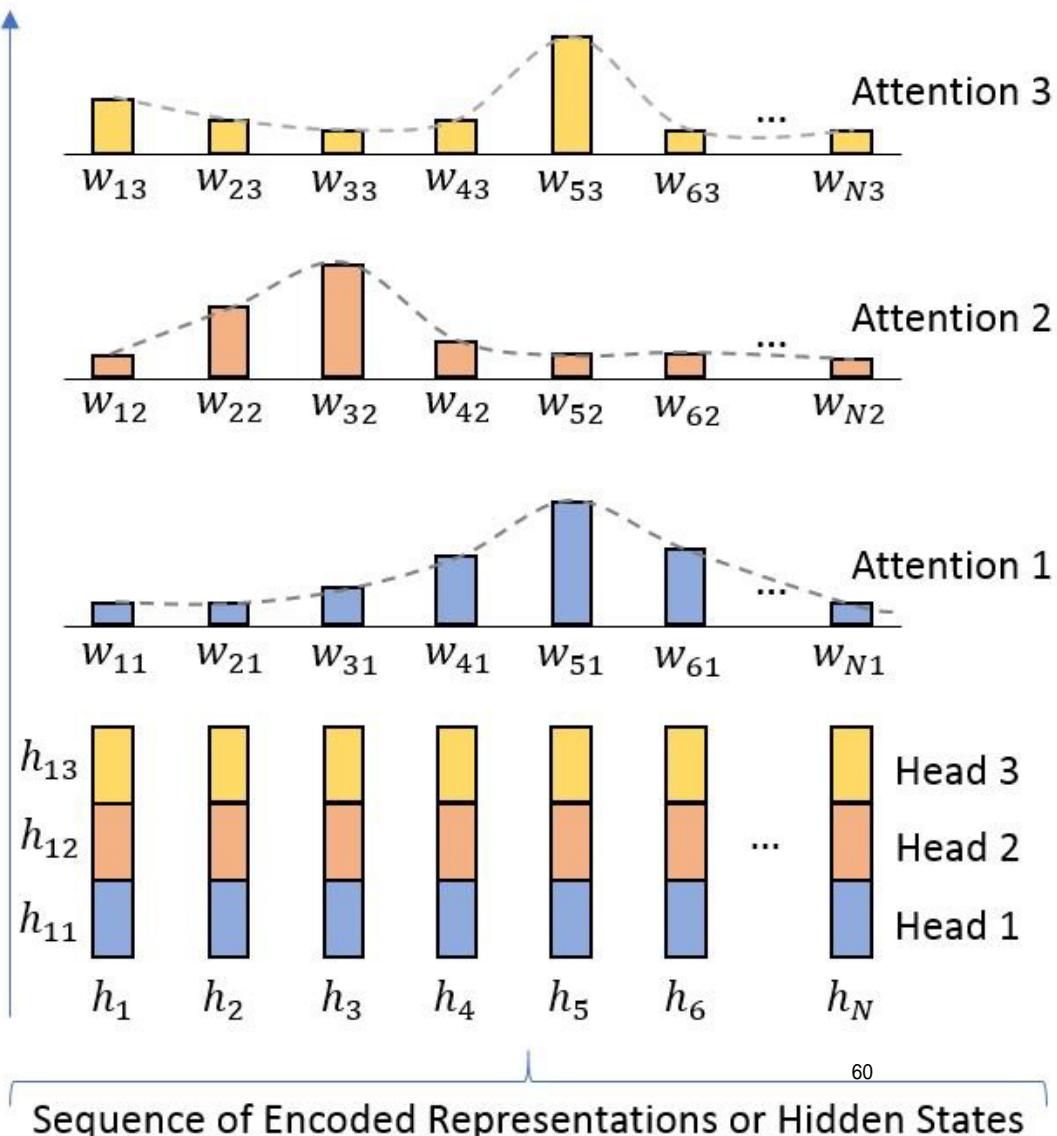
$$X \quad Q_1 \quad Q_2 = XQ_1 \quad XQ_2$$

Transformer Model

- Transformer-based sequence-to-sequence
 - Basic building blocks: masked self-attention
 - Enhancements
 - Key-query-value attention
 - **Multi-headed attention**
 - Standard attention → single-headed attention
 - $X_t \in \mathbb{R}^d, Q, K, V \in \mathbb{R}^{d \times d}$
 - We only “look at” a single position j with high probability
 - What if we want to look at different j for different i ?
 - Idea: define h separate attention heads
 - h different attention distributions, keys, values
 - $Q^l, K^l, V^l \in \mathbb{R}^{d \times \frac{d}{h}}$, for $1 \leq l \leq h$
 - $\alpha_{i,j}^l = \text{softmax}(q_i^T k_j^l); \text{out}_i^l = \sum_j \alpha_{i,j}^l v_j^l$

Utterance Level Representation

$$\underline{c} = [c_1 \ c_2 \ c_3]$$

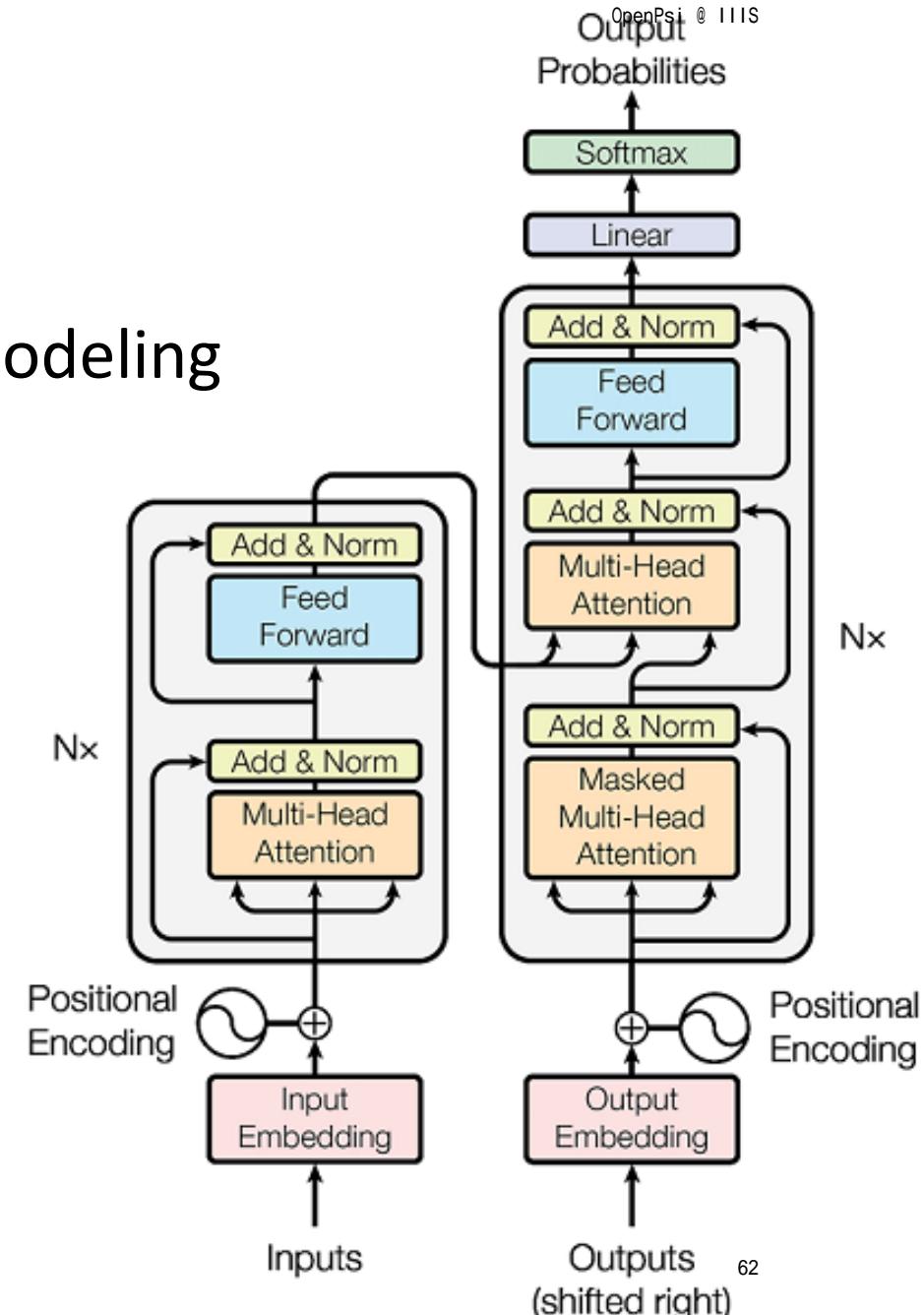


Transformer Model

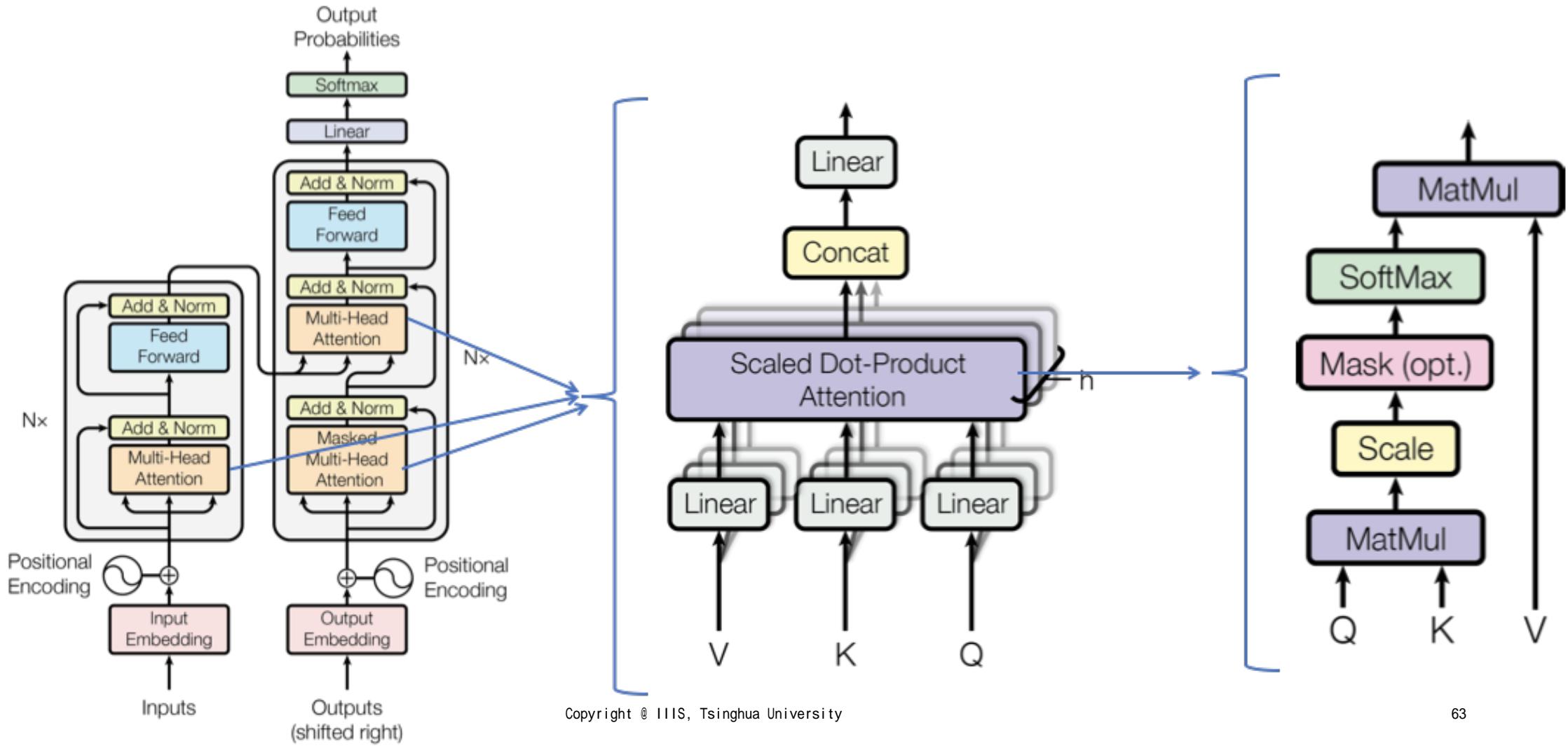
- Transformer-based sequence-to-sequence modeling
 - Basic building blocks: masked self-attention
 - Enhancements
 - Key-query-value attention
 - Multi-headed attention
 - **Architecture modifications**
 - Residual connection
 - Layer normalization
 - $out_t = \text{LN}(f_{SA}(X_t, M) + X_t); m_t = \text{LN}(\text{MLP}(out_t) + out_t)$
 - Scaled dot product
 - Intuition: when dimension d becomes large, $q^T k$ can be large
 - Issue: input to softmax can be large and make gradient small
 - $\alpha_{i,j}^l = \text{softmax}\left(\frac{q_i^l T k_j^l}{\sqrt{d/h}}\right)$

Transformer Model

- Transformer-based sequence-to-sequence modeling
 - Basic building blocks: masked self-attention
 - Position encoding
 - Post-processing MLP
 - Attention mask
 - Enhancements
 - Key-query-value attention
 - Multi-headed attention
 - Architecture modifications
 - Residual connection
 - Layer normalization
 - Scaled dot product



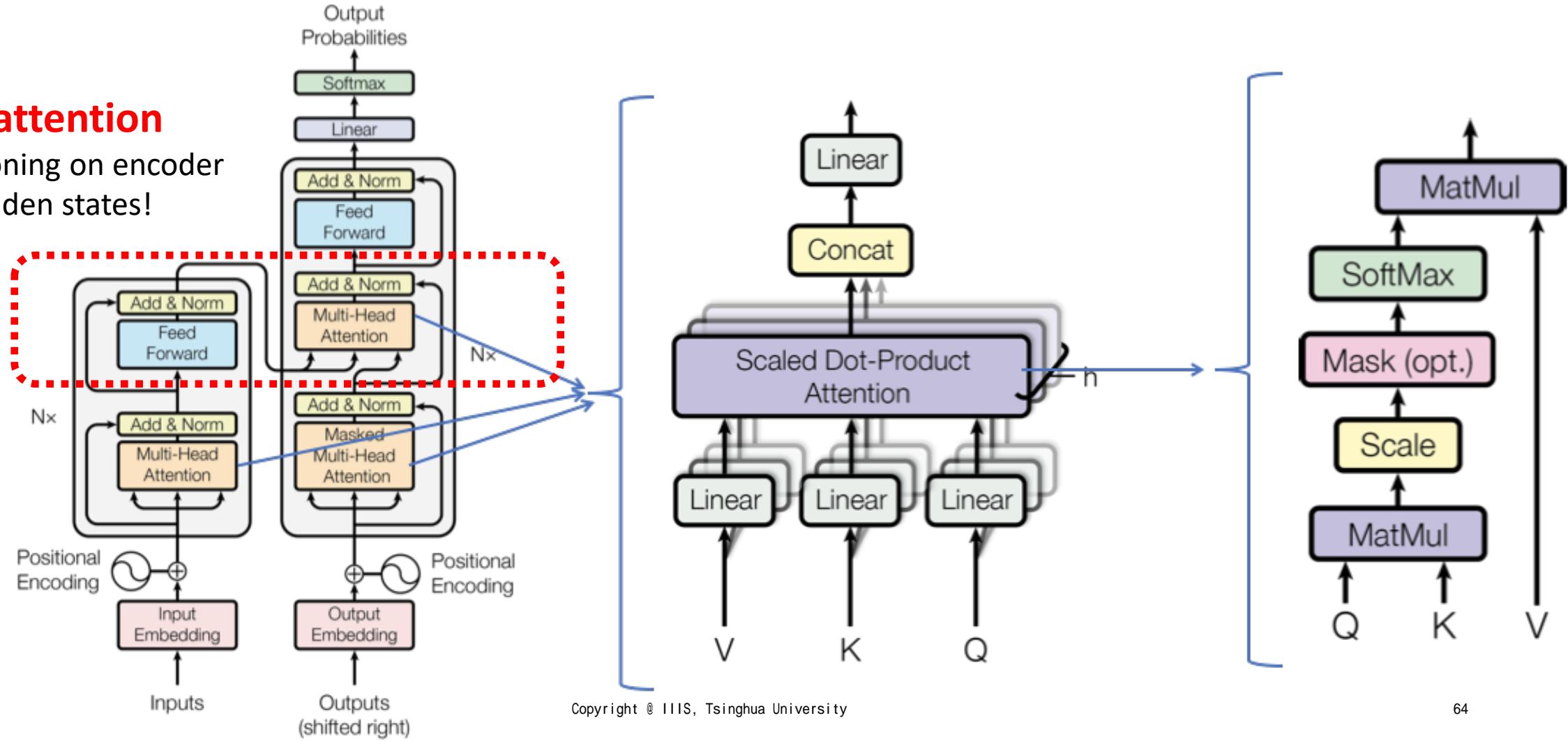
Transformer-Based Seq2Seq Model



Transformer-Based Seq2Seq Model

Cross-attention

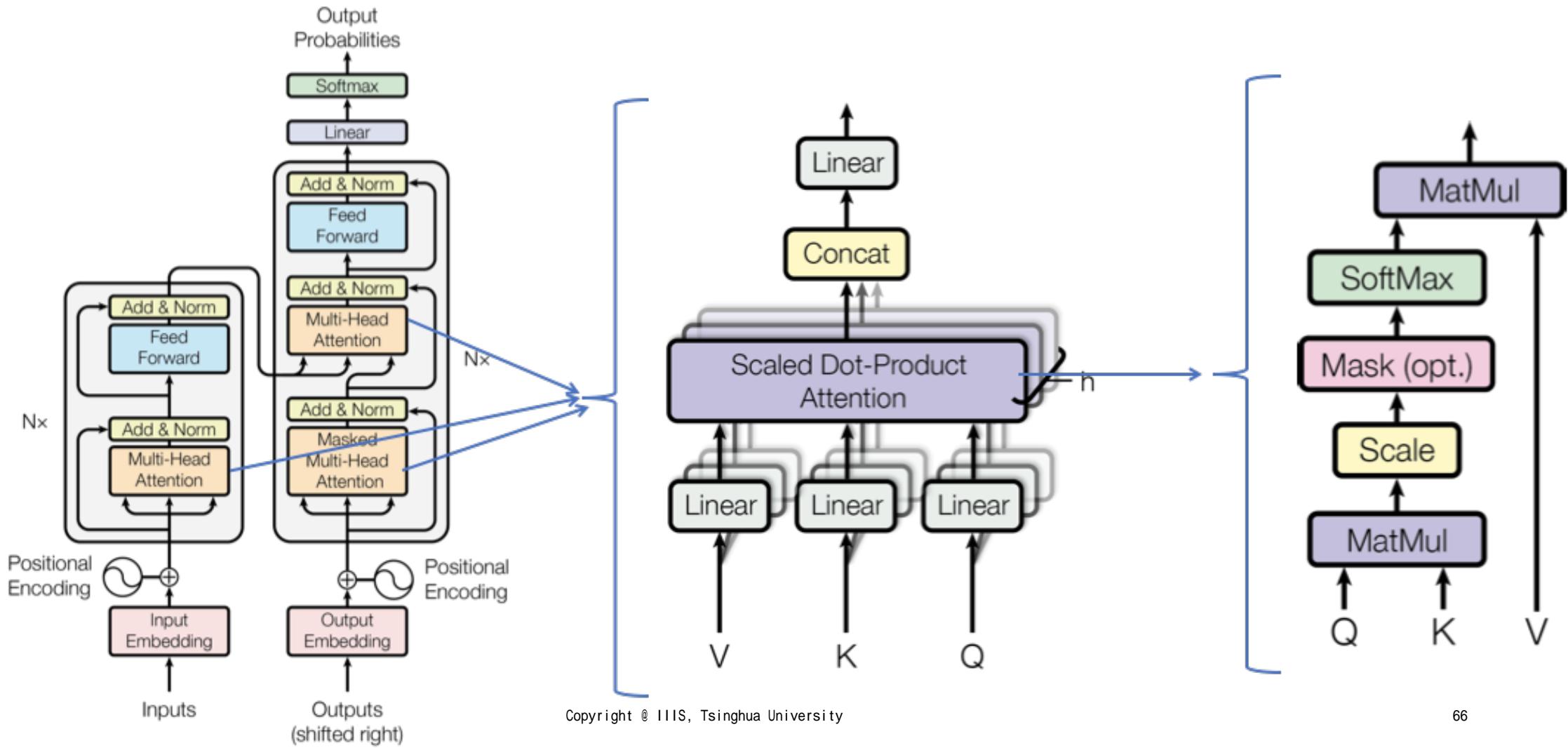
Conditioning on encoder hidden states!



Transformer-Based Seq2Seq Model

- Cross-attention
 - The conditioning part of transformer
 - Decoder can generate texts conditioning on the input sequence
 - Just like standard attention in RNN seq2seq model
 - z_t : decoder SA module inputs
 - h_t : encoder output hidden states
 - For each decoder out_t , we attend on encoder hiddens
 - Query from decoder: $q_t = W^q z_t$
 - Key and value from encoder: $k_j = W^k h_j; v_j = W^v h_j$
 - $\alpha_{ij} = \text{softmax}\left(\frac{q_i^T k_j}{\sqrt{d}}\right); out_i = LN\left(\sum_j \alpha_{ij} v_j + z_i\right)$
 - Many practical variants can be implemented

Transformer-Based Seq2Seq Model



Transformer-Based Seq2Seq Model

- Machine translation with transformer (NIPS2017, Google)

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	41.29	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1		$3.3 \cdot 10^{18}$
Transformer (big)	28.4	41.8		$2.3 \cdot 10^{19}$

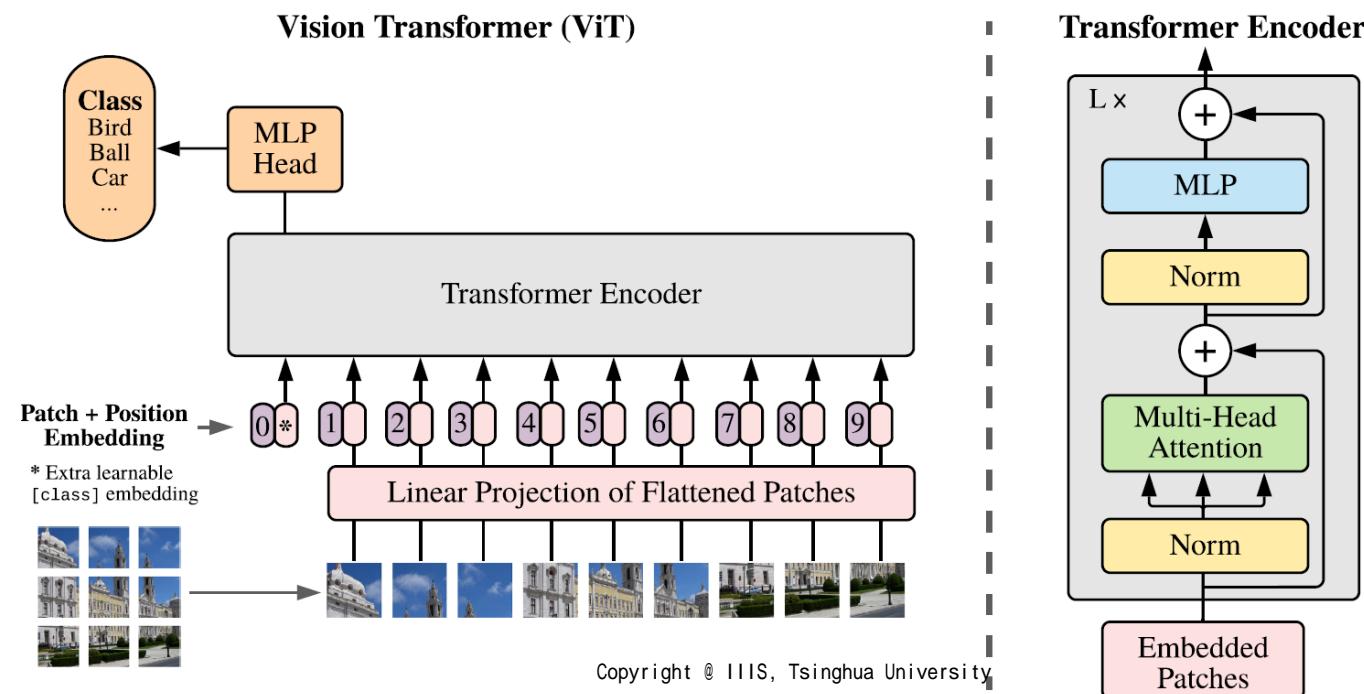
Transformer-Based Seq2Seq Model

- Generating Wikipedia by summarizing long sequences (ICLR2018, Google)
 - Document generation

Model	Test perplexity	ROUGE-L
<i>seq2seq-attention, L = 500</i>	5.04952	12.7
<i>Transformer-ED, L = 500</i>	2.46645	34.2
<i>Transformer-D, L = 4000</i>	2.22216	33.6
<i>Transformer-DMCA, no MoE-layer, L = 11000</i>	2.05159	36.2
<i>Transformer-DMCA, MoE-128, L = 11000</i>	1.92871	37.9
<i>Transformer-DMCA, MoE-256, L = 7500</i>	1.90325	38.8

Transformer Model for Images

- Vision Transformer (ViT, Google Brain, ICLR 2021, 33.4k citation)
 - *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*
 - Decompose an image to 16x16 patches and then apply transformer encoder



Transformer Model for Images

- Vision Transformer (ViT, Google Brain, ICLR 2021, 33.4k citation)
 - *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*
 - Decompose an image to 16x16 patches and then apply transformer encoder

	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21k (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	88.55 ± 0.04	87.76 ± 0.03	85.30 ± 0.02	87.54 ± 0.02	88.4/88.5*
ImageNet ReaL	90.72 ± 0.05	90.54 ± 0.03	88.62 ± 0.05	90.54	90.55
CIFAR-10	99.50 ± 0.06	99.42 ± 0.03	99.15 ± 0.03	99.37 ± 0.06	—
CIFAR-100	94.55 ± 0.04	93.90 ± 0.05	93.25 ± 0.05	93.51 ± 0.08	—
Oxford-IIIT Pets	97.56 ± 0.03	97.32 ± 0.11	94.67 ± 0.15	96.62 ± 0.23	—
Oxford Flowers-102	99.68 ± 0.02	99.74 ± 0.00	99.61 ± 0.02	99.63 ± 0.03	—
VTAB (19 tasks)	77.63 ± 0.23	76.28 ± 0.46	72.72 ± 0.21	76.29 ± 1.70	—
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k

Transformer Model for Images

- Swin Transformer (MSRA, CVPR 2021 best paper)
 - Build hierarchical feature maps at different resolution
 - Self-attention only within each block (linear computation for image size)
 - Shifted block partitions to encode information between blocks

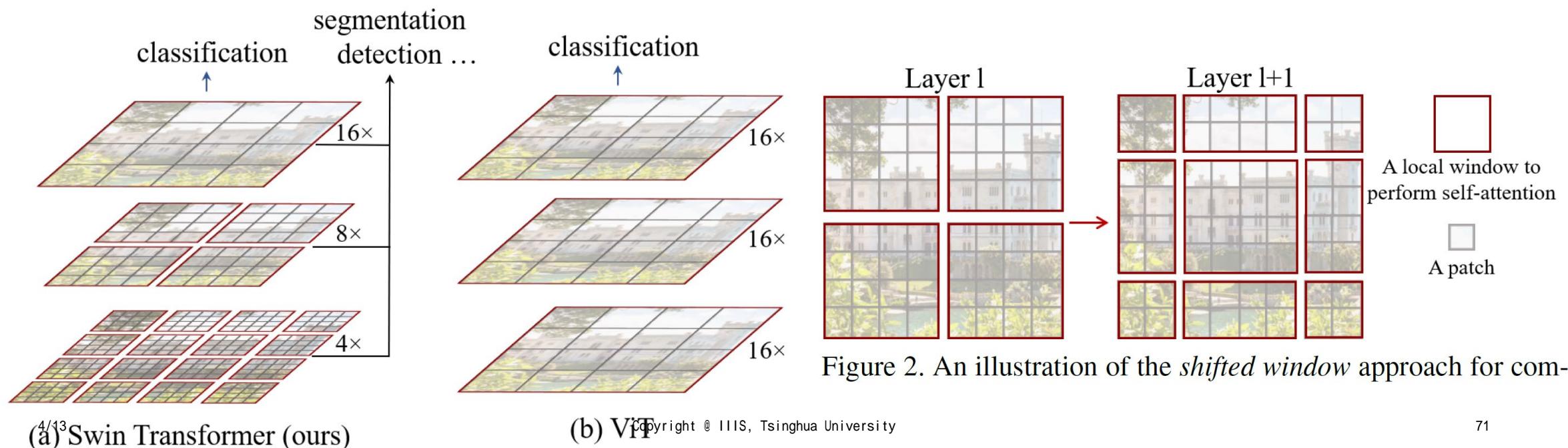


Figure 2. An illustration of the *shifted window* approach for com-

Transformer Model for Images

- Swin Transformer (MSRA, CVPR 2021 best paper)
 - Build hierarchical feature maps at different resolution
 - Self-attention only within each block (linear computation for image size)
 - Shifted block partitions to encode information between blocks

Method	mini-val		test-dev		#param.	FLOPs
	AP ^{box}	AP ^{mask}	AP ^{box}	AP ^{mask}		
RepPointsV2* [12]	-	-	52.1	-	-	-
GCNet* [7]	51.8	44.7	52.3	45.4	-	1041G
RelationNet++* [13]	-	-	52.7	-	-	-
SpineNet-190 [21]	52.6	-	52.8	-	164M	1885G
ResNeSt-200* [78]	52.5	-	53.3	47.1	-	-
EfficientDet-D7 [59]	54.4	-	55.1	-	77M	410G
DetectoRS* [46]	-	-	55.7	48.5	-	-
YOLOv4 P7* [4]	-	-	55.8	-	-	-
Copy-paste [26]	55.9	47.2	56.0	47.4	185M	1440G
X101-64 (HTC++)	52.3	46.0	-	-	155M	1033G
Swin-B (HTC++)	56.4	49.1	-	-	160M	1043G
Swin-L (HTC++)	57.1	49.5	57.7	50.2	284M	1470G
Swin-L (HTC++)*	58.0	50.4	58.7	51.1	284M	-

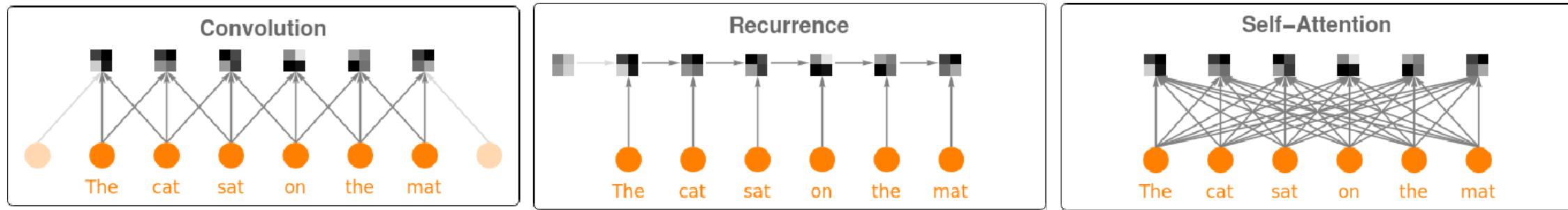
Table 2. Results on COCO object detection and instance segmentation. [†]denotes that additional deconvolution layers are used to produce hierarchical feature maps. * indicates multi-scale testing.

ADE20K Method	Backbone	val mIoU	test score	#param.	FLOPs	FPS
DANet [23]	ResNet-101	45.2	-	69M	1119G	15.2
DLab.v3+ [11]	ResNet-101	44.1	-	63M	1021G	16.0
ACNet [24]	ResNet-101	45.9	38.5	-	-	-
DNL [71]	ResNet-101	46.0	56.2	69M	1249G	14.8
OCRNet [73]	ResNet-101	45.3	56.0	56M	923G	19.3
UperNet [69]	ResNet-101	44.9	-	86M	1029G	20.1
OCRNet [73]	HRNet-w48	45.7	-	71M	664G	12.5
DLab.v3+ [11]	ResNeSt-101	46.9	55.1	66M	1051G	11.9
DLab.v3+ [11]	ResNeSt-200	48.4	-	88M	1381G	8.1
SETR [81]	T-Large [‡]	50.3	61.7	308M	-	-
UperNet	DeiT-S [†]	44.0	-	52M	1099G	16.2
UperNet	Swin-T	46.1	-	60M	945G	18.5
UperNet	Swin-S	49.3	-	81M	1038G	15.2
UperNet	Swin-B [‡]	51.6	-	121M	1841G	8.7
UperNet	Swin-L [‡]	53.5	62.8	234M	3230G	6.2

Table 3. Results of semantic segmentation on the ADE20K val and test set. [†] indicates additional deconvolution layers are used to produce hierarchical feature maps. [‡] indicates that the model is pre-trained on ImageNet-22K.

Comparisons

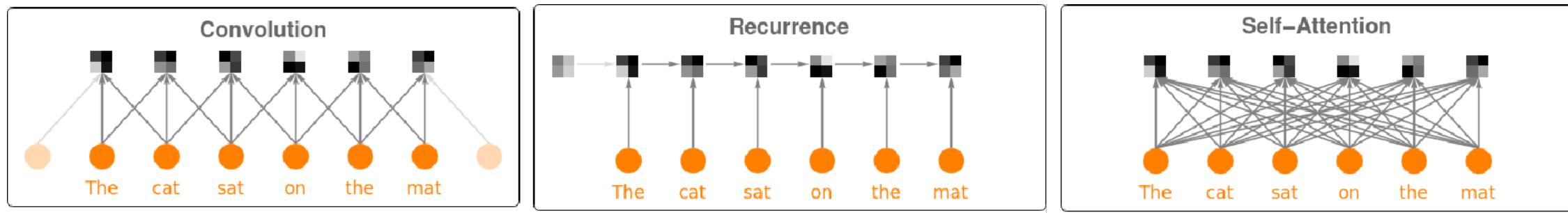
- CNN v.s. LSTM v.s. Transformer



Context length L	Temporal Convolution	RNN	Transformer
Layers			

Comparisons

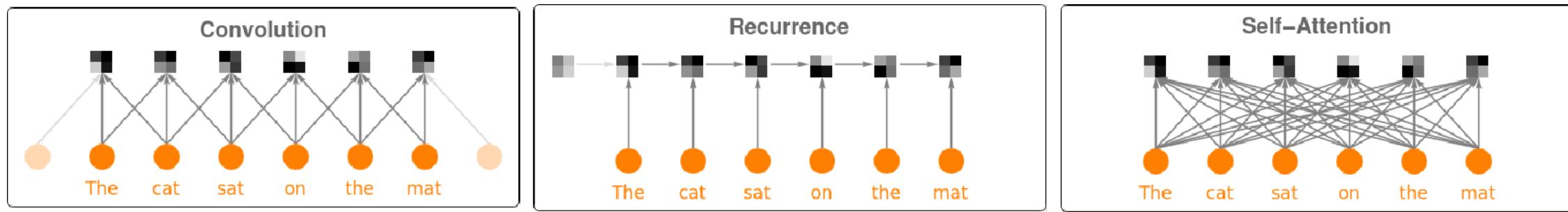
- CNN v.s. LSTM v.s. Transformer



Context length L	Temporal Convolution	RNN	Transformer
Layers	$O(\log L)$	$O(1)$	$O(1)$
Generation			
Inference			

Comparisons

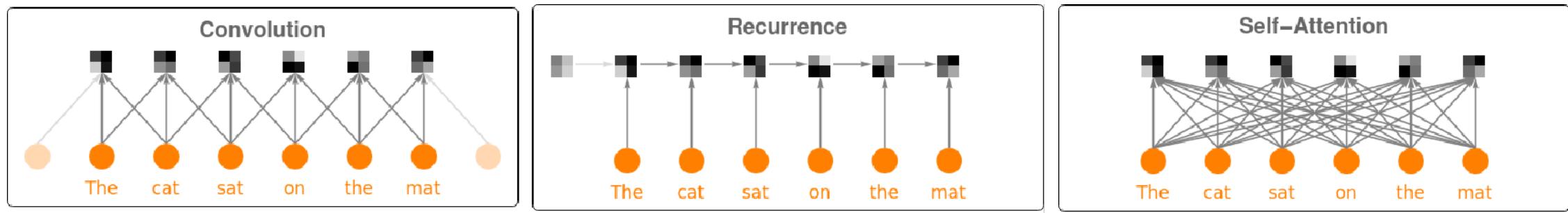
- CNN v.s. LSTM v.s. Transformer



Context length L	Temporal Convolution	RNN	Transformer
Layers	$O(\log L)$	$O(1)$	$O(1)$
Generation	$O(L \log L)$	$O(L)$	$O(L^2)$
Inference			

Comparisons

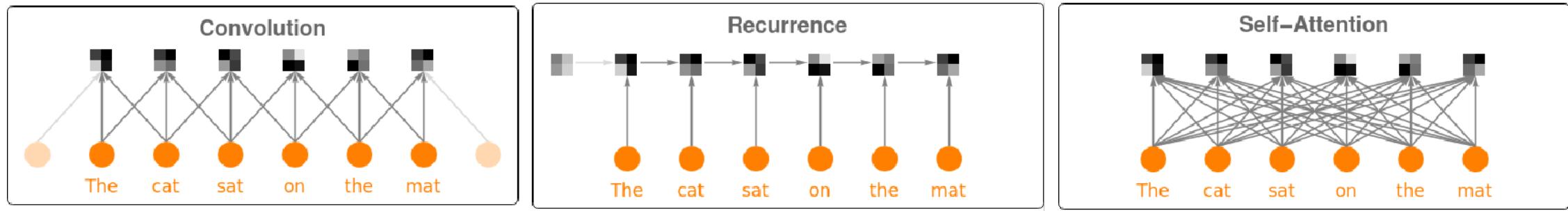
- CNN v.s. LSTM v.s. Transformer



Context length L	Temporal Convolution	RNN	Transformer
Layers	$O(\log L)$	$O(1)$	$O(1)$
Generation	$O(L \log L)$	$O(L)$	$O(L^2)$
Inference	$O(\log L)$	$O(L)$	$O(1)$

Comparisons

- CNN v.s. LSTM v.s. Transformer



Context length L	Temporal Convolution	RNN	Transformer
Layers	$O(\log L)$	$O(1)$	$O(1)$
Generation	$O(L \log L)$	$O(L)$	$O(L^2)$
Inference	$O(\log L)$	$O(L)$	$O(1)$

Quadratic!!

Can we speedup transformer generation?

Speed up Transformers

- Quadratic generation cost
 - $O(L^2)$ for length L : sequential generation $O(L) \times$ attention $O(L)$
 - What if we want to model sequence length of, say, $L > 10^4$

Speed up Transformers

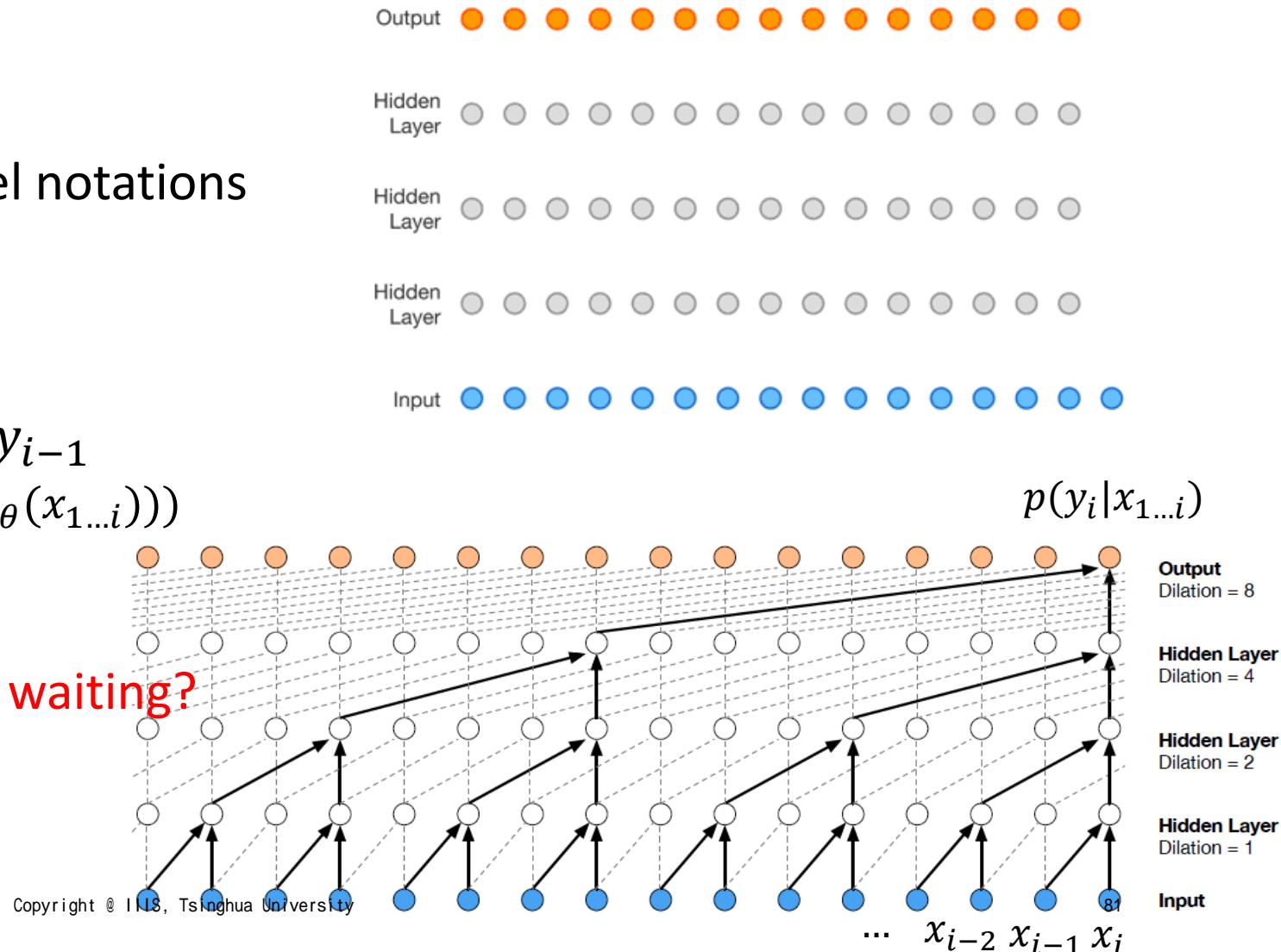
- Quadratic generation cost
 - $O(L^2)$ for length L : sequential generation $O(L) \times$ attention $O(L)$
- Make **attention** faster/better
 - Large-scale training: transformer-XL; XL-net (Zhilin Yang, et al, Google, 2020)
 - Projection tricks: Linformer (Facebook AI, $O(n)$ computation, 2020)
 - Math tricks: Performer (Google, $O(n)$ computation, 2020)
 - Sparse interactions:
 - Big Bird (Google, 2020), Multi-head Latent Attention (DeepSeek, 2024) <https://planetbanatt.net/articles/mla.html>
 - Fast and memory-efficient attention:
 - Flash Attention (Tri Dao, et al, 2022) and Ring Attention (Hao Liu, et al, 2023)
 - System engines for fast generation: vLLM (Berkeley) and SGLang (xAI & UCLA)
 - Even Parallel/Contextualized RNN (make RNN great again):
 - RWKV RNN (Open-Source, 2023) & Mamba (Gu, Albert and Tri Dao, 2023)
 - Reduce attention flatten issue when length grows: Scalable-Softmax (2025)

Speed up Transformers

- Quadratic generation cost
 - $O(L^2)$ for length L : sequential generation $O(L) \times$ attention $O(L)$
- Remark:
 - Ideally, attention can be computed in parallel given unbounded computation and memory bandwidth
- Can we accelerate autoregressive generation?
 - This is the key bottleneck for language model generation, which cannot be accelerated by hardware improvement

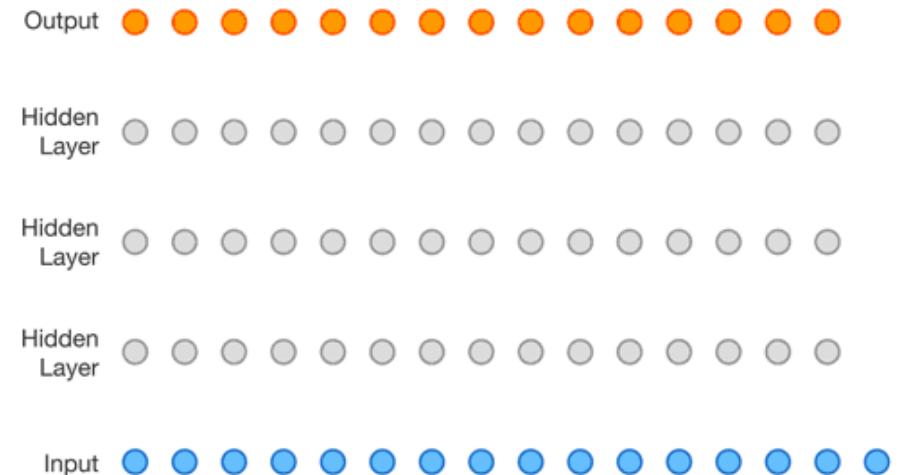
Beyond Autoregressive Generation

- WaveNet (Recap)
 - Let's use the language model notations
 - Output $y_1 \dots y_i$
 - Input $x_1 \dots x_i$ ($x_i = y_{i-1}$)
 - $p(y) = \prod_i p(y_i|x_1 \dots i)$
 - y_i can only be computed after y_{i-1}
 - $p(y_i) = N(\mu_\theta(x_1 \dots i), \exp^2(\alpha_\theta(x_1 \dots i)))$
 - $x_i \leftarrow y_{i-1}$
 - y_{i-1} is part of input of y_i
 - **Can we compute y_i without waiting?**



Beyond Autoregressive Generation

- WaveNet (Recap)
 - Let's use the language model notations
 - Output $y_1 \dots y_i$
 - Input $x_1 \dots x_i$ ($x_i = y_{i-1}$)
 - $p(y) = \prod_i p(y_i|x_1 \dots i)$
 - Reparameterization trick
 - $z_i \sim N(0,1)$
 - $y_i \leftarrow \mu_\theta(x_1 \dots i) + z_i \cdot \exp(\alpha_\theta(x_1 \dots i))$
 - $x_i \leftarrow y_{i-1}$



Beyond Autoregressive Generation

- WaveNet (Recap)

- Let's use the language model notations

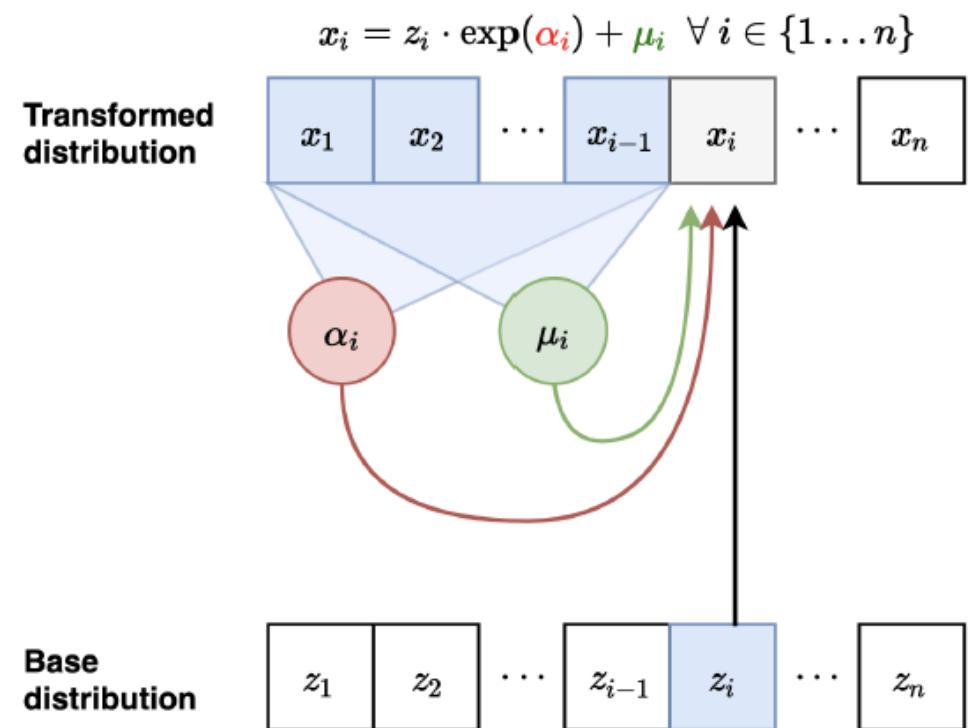
- Output $y_1 \dots y_i$
- Input $x_1 \dots x_i$ ($x_i = y_{i-1}$)
- $p(y) = \prod_i p(y_i|x_1 \dots i)$

- Reparameterization trick

- $z_i \sim N(0,1)$
- $y_i \leftarrow \mu_\theta(x_{1 \dots i}) + z_i \cdot \exp(\alpha_\theta(x_{1 \dots i}))$
- $x_i \leftarrow y_{i-1}$

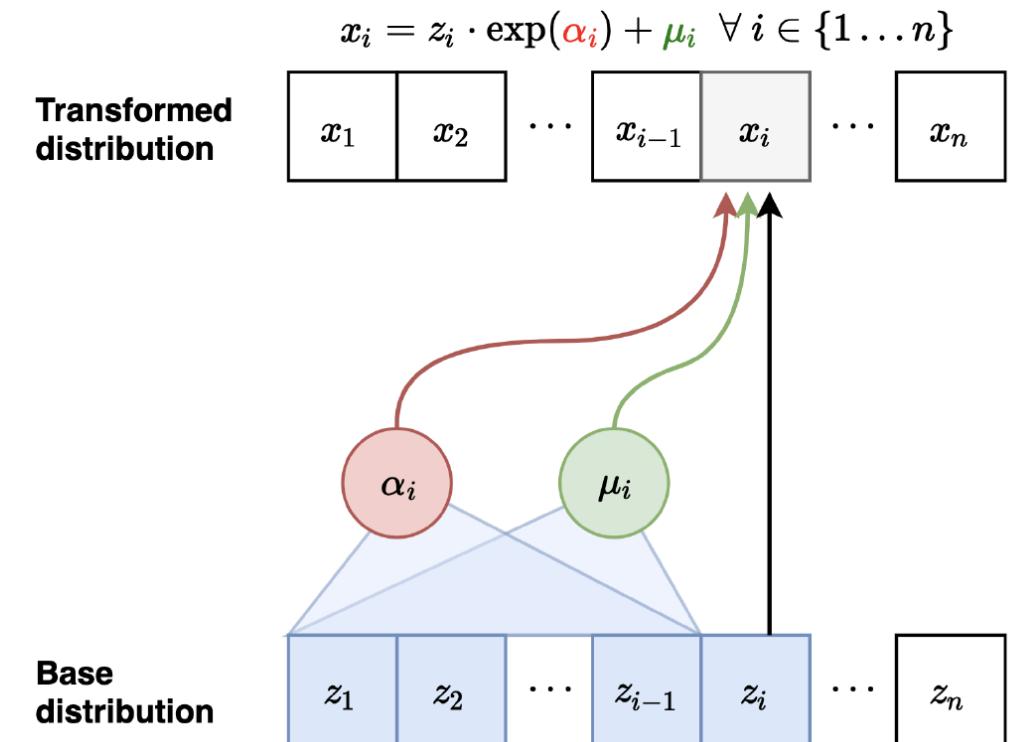
- x_i can be written as a function of $z_{1 \dots i}$ without compromising the representation power!

- Each x is corresponding to a unique z ! (your homework)



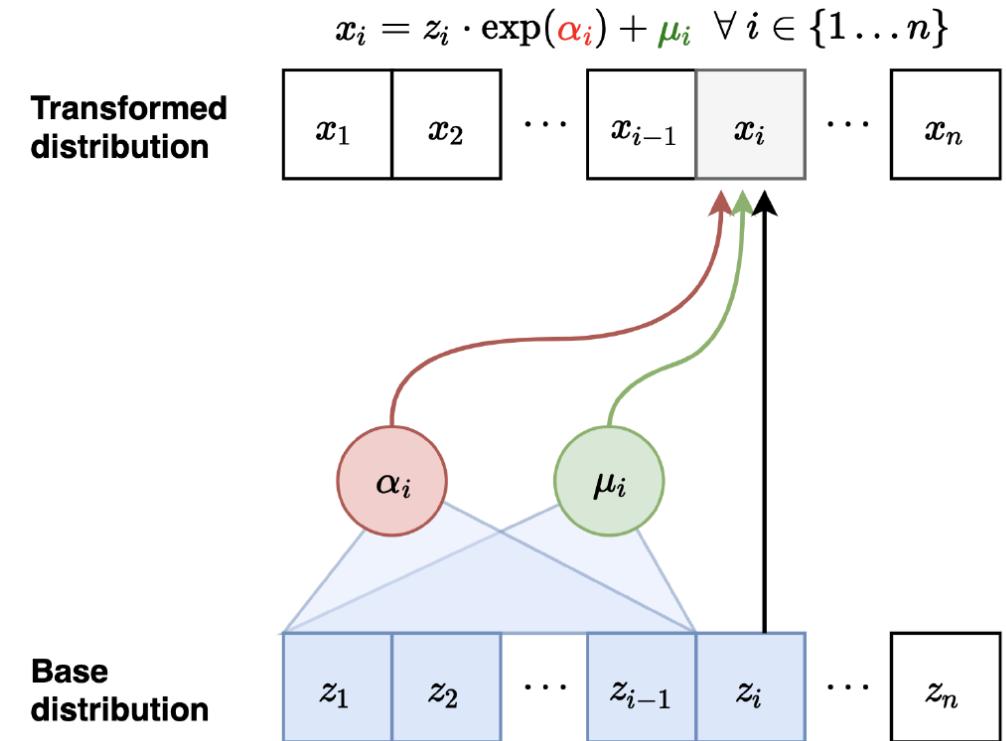
Beyond Autoregressive Generation

- Parallel WaveNet (DeepMind, ICML 2018)
 - Sequential modeling notation (ignore y)
 - Sequence tokens x & latent variable z
 - $p(x) = \prod_i p(x_i|x_1\dots i)$
 - Reparameterization trick
 - $z_i \sim N(0,1)$
 - $x_i \leftarrow \mu_\theta(z_{1\dots i-1}) + z_i \cdot \exp(\alpha_\theta(z_{1\dots i-1}))$
 - Parallel generation
 - First generate z and then x



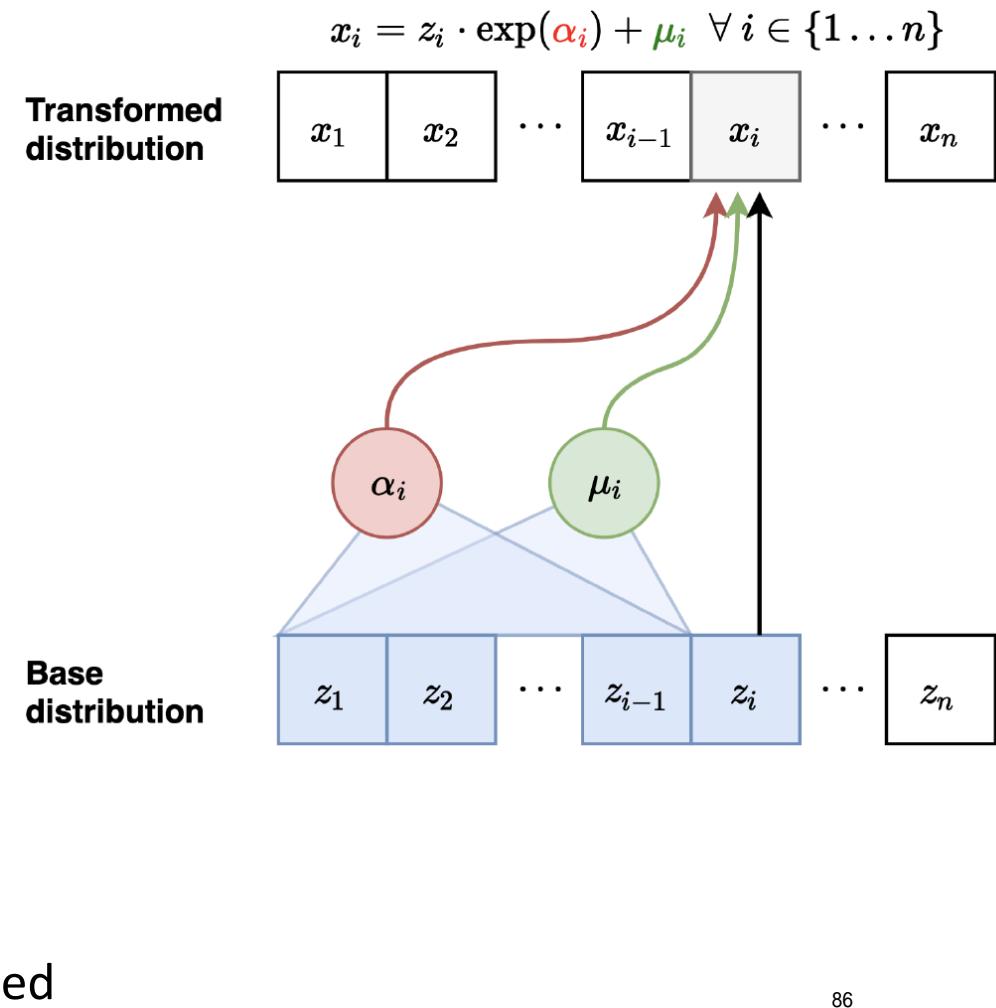
Beyond Autoregressive Generation

- Parallel WaveNet (DeepMind, ICML 2018)
 - Sequential modeling notation (ignore y)
 - Sequence tokens x & latent variable z
 - $p(x) = \prod_i p(x_i|x_1...i)$
 - Reparameterization trick
 - $z_i \sim N(0,1)$
 - $x_i \leftarrow \mu_\theta(z_{1...i-1}) + z_i \cdot \exp(\alpha_\theta(z_{1...i-1}))$
 - Parallel generation
 - First generate z and then x
 - **What about inference?**
 - Given x , how to compute $p(x)$ for MLE training?



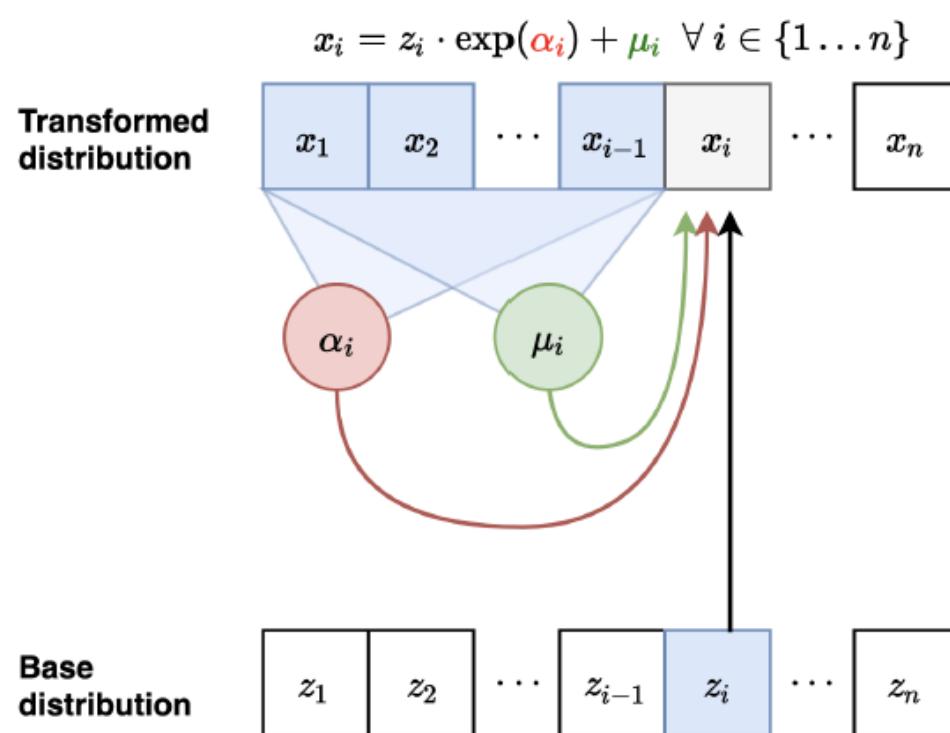
Beyond Autoregressive Generation

- Parallel WaveNet (DeepMind, ICML 2018)
 - Sequential modeling notation (ignore y)
 - Sequence tokens x & latent variable z
 - $p(x) = \prod_i p(x_i|x_1...i)$
 - Reparameterization trick
 - $z_i \sim N(0,1)$
 - $x_i \leftarrow \mu_\theta(z_{1...i-1}) + z_i \cdot \exp(\alpha_\theta(z_{1...i-1}))$
 - Parallel generation
 - First generate z and then x
 - **Sequential inference $O(L)$**
 - x_i is a function of $z_1 \dots z_i$
 - $z_i \leftarrow (x_i - \mu_i) / \exp(\alpha_i)$
 - z_i can only be recovered after z_1, z_2, \dots, z_{i-1} is computed



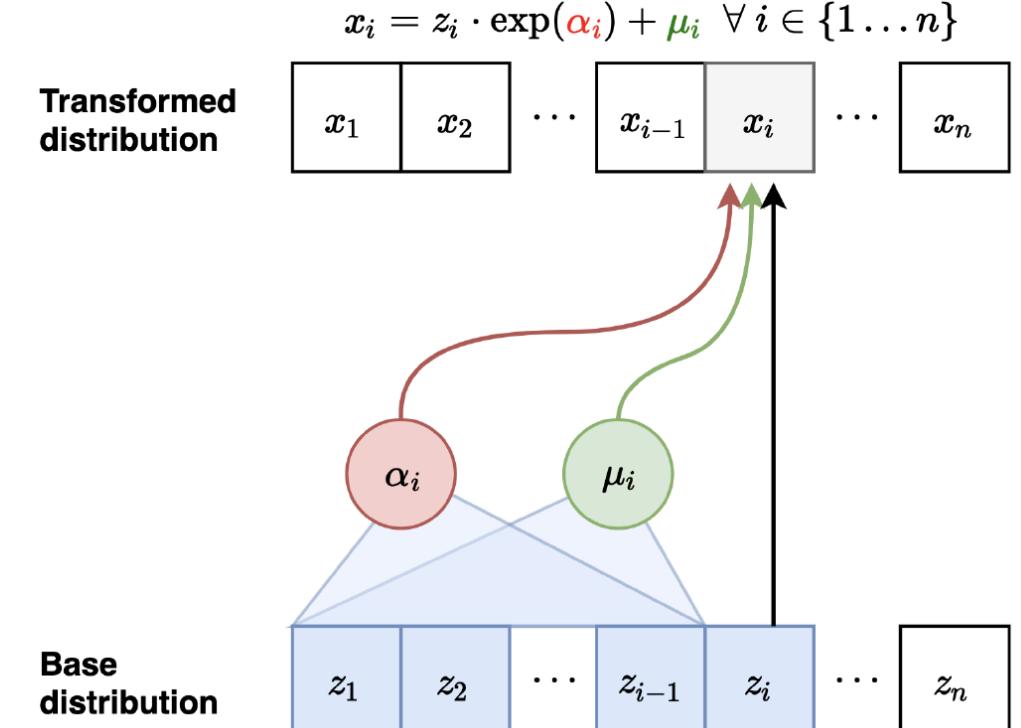
Beyond Autoregressive Generation

- Parallel WaveNet (DeepMind, ICML 2018)



Standard WaveNet

- Autoregressive generation $O(L \log L)$
- Parallel inference $O(\log L)$

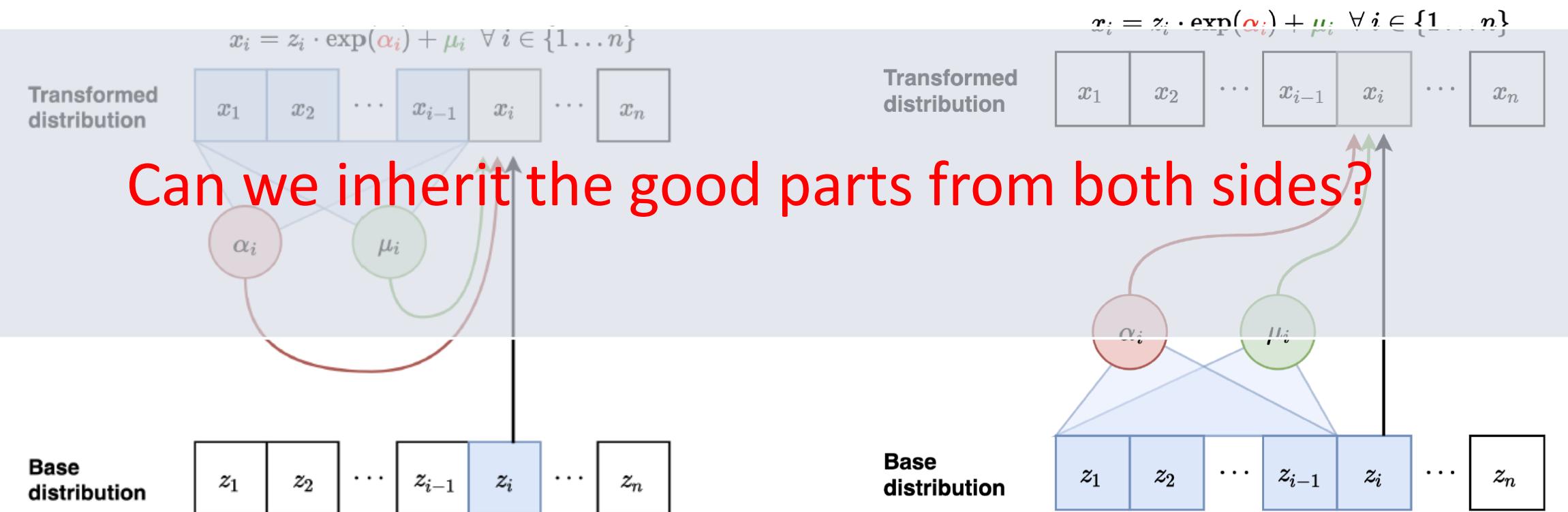


Parallel WaveNet

- Parallel generation $O(\log L)$
- Autoregressive inference $O(L \log^8 L)$

Beyond Autoregressive Generation

- Parallel WaveNet (DeepMind, ICML 2018)



Standard WaveNet

- Autoregressive generation $O(L \log L)$
- Parallel inference $O(\log L)$

Parallel WaveNet

- Parallel generation $O(\log L)$
- Autoregressive inference $O(L \log^{88} L)$

Beyond Autoregressive Generation

- Parallel WaveNet (DeepMind, ICML 2018)
 - Key facts
 - Standard WaveNet is fast for training (inference is fast)
 - Parallel WaveNet is fast for serving (generation is fast)
 - Distillation by teacher-student framework!
 - Teacher: $p_T(x_i|x_{<i})$ a standard WaveNet for training on massive data
 - Student: $p_S(x_i|z_{<i})$ a parallel WaveNet for serving
 - p_S is trained by distillation from p_T
 - i.e., minimize the KL-difference between $p_S(x)$ and $p_T(x)$
 - Algorithm Sketch
 - Step 1: Train teacher $p_T(x_i|x_{<i})$ network and fix it
 - Step 2: Minimize the KL-difference $KL(p_S||p_T)$
 - Finally we use $p_S(x)$ for fast sampling

Beyond Autoregressive Generation

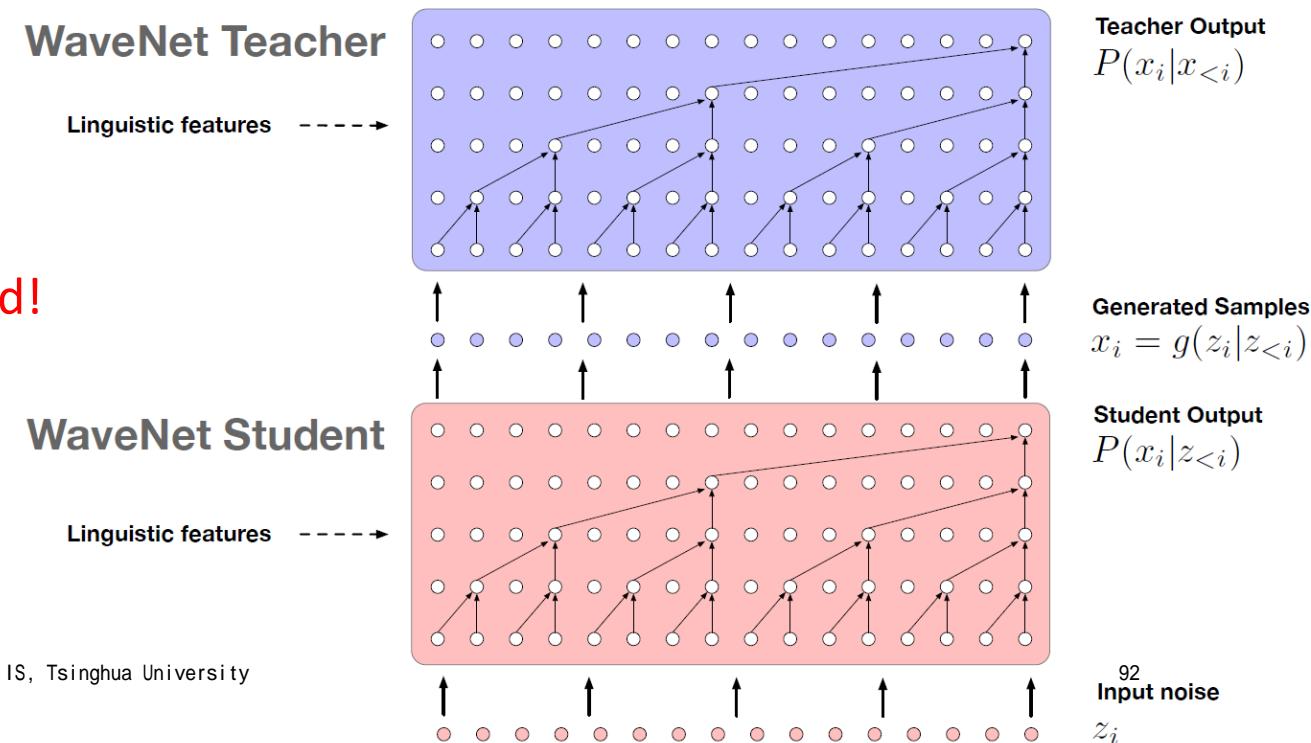
- Parallel WaveNet (DeepMind, ICML 2018)
 - Key facts
 - Standard WaveNet is fast for training (inference is fast)
 - Parallel WaveNet is fast for serving (generation is fast)
 - Distillation by teacher-student framework!
 - Teacher: $p_T(x_i|x_{<i})$ a standard WaveNet for training on massive data
 - Student: $p_S(x_i|z_{<i})$ a parallel WaveNet for serving
 - p_S is trained by distillation from p_T
 - i.e., minimize the KL-difference between $p_S(x)$ and $p_T(x)$
 - Algorithm Sketch
 - Step 1: Train teacher $p_T(x_i|x_{<i})$ network and fix it
 - Step 2: Minimize the **KL-difference** $KL(p_S||p_T)$ Pay attention to the order!
 - Finally we use $p_S(x)$ for fast sampling

Beyond Autoregressive Generation

- Parallel WaveNet (DeepMind, ICML 2018)
 - Teacher-Student Learning
 - Pretrain $p_T(x)$ and then train $p_S(x)$ by imitation learning
 - Distance measure for two distributions
 - KL divergence: $KL(p||q) = \mathbb{E}_{x \sim p} \left[\log \frac{p(x)}{q(x)} \right]$
 - Distillation (Imitation learning)
$$L(\theta) = KL(p_S || p_T) = \mathbb{E}_{x \sim p_S} [\log p_S(x; \theta) - \log p_T(x)]$$
 - Monte Carlo estimates for the expectation
 - **Key: sample from the student network!**
 - Sample $z \sim N(0, I)$, generate $x \sim p_S(x|z)$ (parallel)
 - Evaluate $p_S(x|z)$ (parallel since z is known)
 - Evaluate $p_T(x)$ (parallel since $p_T(x)$ is a standard autoregressive model)

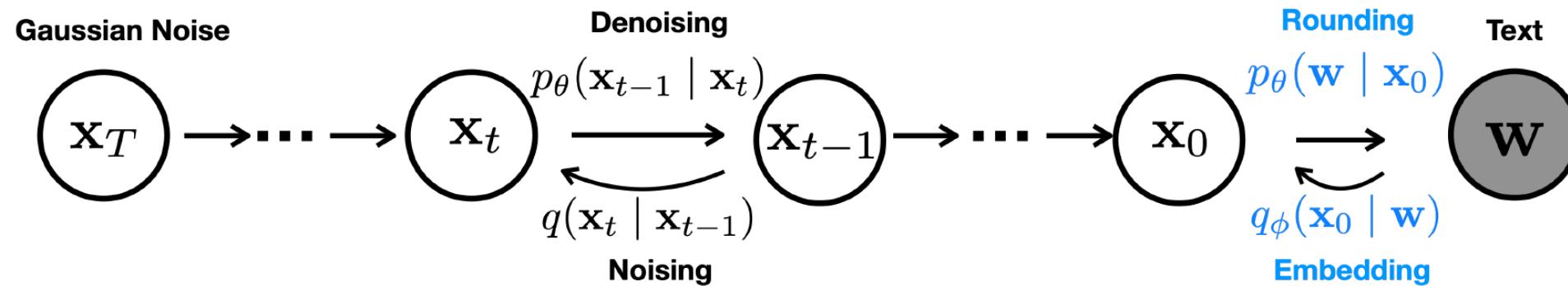
Beyond Autoregressive Generation

- Parallel WaveNet (DeepMind, ICML 2018)
 - Teacher-Student Learning
 - Pretrain $p_T(x)$ and then train $p_S(x)$ by imitation learning
 - Speedup
 - 20x faster than real-time
 - 1000x faster than WaveNet
 - Google production
 - Remark
 - A reparameterization trick is assumed!
- What about language model?
 - Output are discrete tokens
 - No parameterization available



Beyond Autoregressive Generation

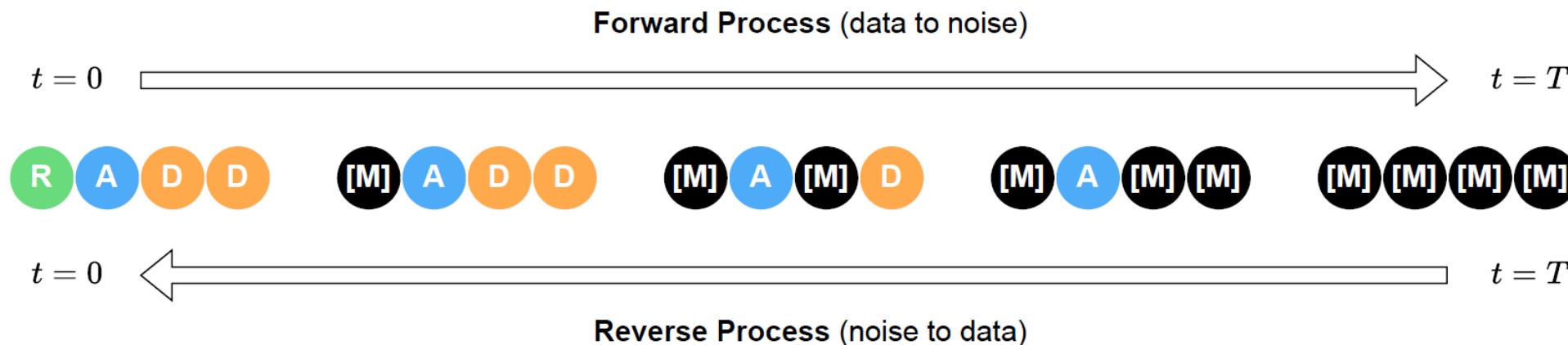
- Diffusion-based language model
 - Key idea: use a diffusion model to generate all tokens at once!
 - Idea#1: treat embeddings of tokens as images



- Pros: we can directly apply all techniques from diffusion models
- Cons: **length issue**, extremely high dimensions, poor generation quality

Beyond Autoregressive Generation

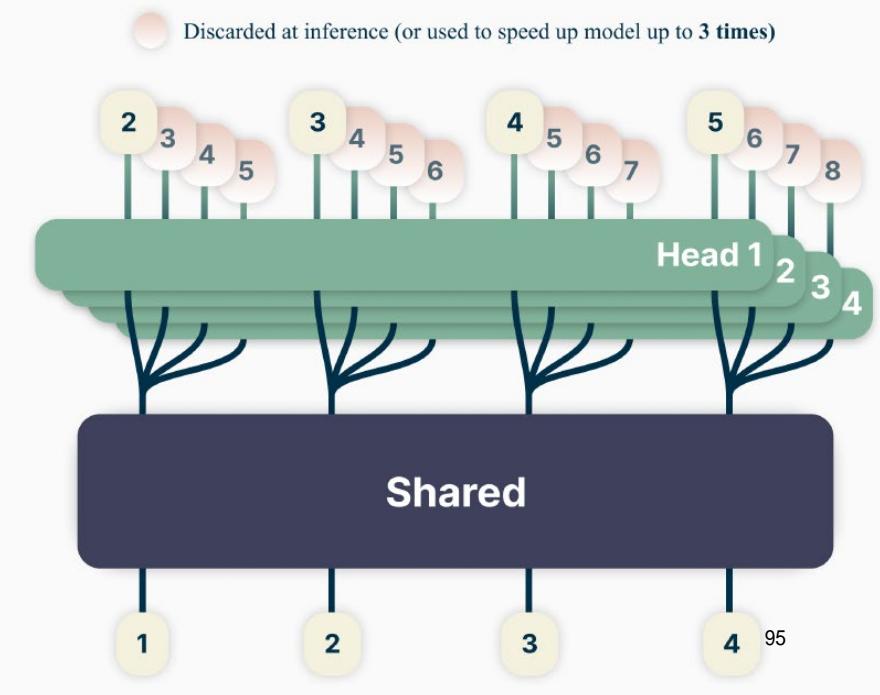
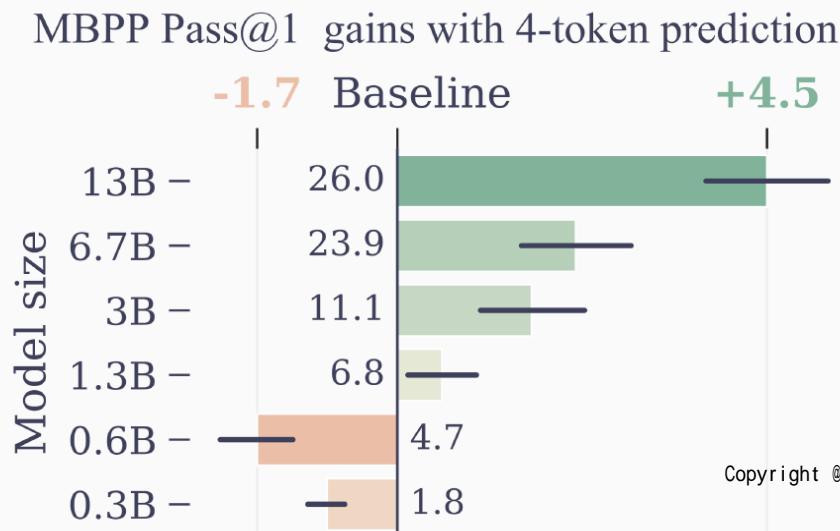
- Diffusion-based language model
 - Key idea: use a diffusion model to generate all tokens at once!
 - Idea#1: treat embeddings of tokens as images
 - Idea#2: define the denoising process over token masks



- Pros: reduce denoising operator to standard token prediction, reasonable quality
- Cons: **length issue**, no denoising acceleration anymore
- Any simpler method for joint token predictions?

Beyond Autoregressive Generation

- Multi-Token Prediction (ICML 2024)
 - Key insight $p(x_{1..L}) \approx \prod_i p(x_L)$
 - When the sequence length L is really short, we can break the sequential dependency
 - Predict K tokens jointly in parallel
 - $p(x_{i-K+1} \dots | x_{1..i-K}) \approx \prod_{j=1}^K p(x_{i-j+1} | x_{1..i-K})$
 - Small K & larger models higher gains



Beyond Autoregressive Generation

- Multi-Token Prediction (ICML 2024)
 - Key insight $p(x_{1..L}) \approx \prod_i p(x_L)$
 - When the sequence length L is really short, we can break the sequential dependency
 - Predict K tokens jointly in parallel
 - $p(x_{i-K+1} \dots i | x_{1..i-K}) \approx \prod_{j=1}^K p(x_{i-j+1} | x_{1..i-K})$
 - Understanding multi-token prediction
 - Using a **faster but worse model** (independent model) to approximate the **target distribution** (full language model)
 - **Other choice of fast sampling model?**
 - E.g., an LSTM, or a just smaller model

Beyond Autoregressive Generation

- Speculative Decoding (ICML 2023)
 - Key facts
 - A general language model $p(x)$ is fast at evaluation but slow at generation
 - A sampling model $q(x)$ is fast at generation but at low quality
 - Goal: adaptively use $q(x)$ to generate at **easy cases**
 - How to define **easy** cases?

Beyond Autoregressive Generation

- Speculative Decoding (ICML 2023)
 - Key facts
 - A general language model $p(x)$ is fast at evaluation but slow at generation
 - A sampling model $q(x)$ is fast at generation but at low quality
 - Goal: adaptively use $q(x)$ to generate at **easy cases**
 - MCMC Sampling!
 - We treat $q(x)$ as a proposal distribution
 - Given a partial prefix $x_{1\dots i}$, run q to generate next K tokens x'
 - Evaluate $p(x'|x)$ and $q(x'|x)$, accept x' with prob. $\min\left(1, \frac{p(x'|x)}{q(x'|x)}\right)$ (parallel)
 - If rejection, re-sample $x' \propto \max(0, p(x'|x) - q(x'|x))$ (autoregressive)
 - In practice, we can run speculative sampling for multiple K

Beyond Autoregressive Generation

[INST]Write a poem for my three year old[/INST]

[INST]Write a poem for my three year old[/INST]

Faster Transformer Generation

- Reparameterization and distillation
 - Pros: fast training and inference
 - Cons: only works for continuous values
- Diffusion-based language model
 - High complexity for generation and still low generation quality
- Multi-token prediction
 - Trade generation quality for speed
- Speculative decoding
 - Most general approach to speed up generation
 - Can be applied to any trained language model without modification

Summary

- Language Model & Sequence to Sequence Model
 - Fundamental ideas and methods for sequence modeling/tasks
- Attention Mechanism
 - So far the most successful idea for sequence data in deep learning
 - A scale/order-invariant representation
 - Transformer: a fully attention-based model for sequence data
- Speedup Transformers
 - Generation is the key bottleneck for transformer models
 - Acceleration by faster attention
 - Acceleration by non-autoregressive generation
 - Model changes: distillation, diffusion, multi-token prediction
 - Sampling methods: speculative decoding

Thanks