FIT3181/5215 Deep Learning

Week 09: Deep Learning for Sequential Data (II): Seq2Seq and Transformers

Lecturer: Trung Le

Email: trunglm@monash.edu

GROUP OF EIGHT AUSTRALIA

Department of Data Science and AI
Faculty of Information Technology, Monash University, Australia

Outline

- Encoder-Decoder models

- Sequence to sequence models
  - Machine Translation and Image Captioning

- Attention mechanism
  - Global attention and Local attention

- Transformer and BERT
  - Self-Attention Mechanism & Multi-Head Self-Attention Mechanism
  - Pre-trained Language Models: BERT

Encoder-Decoder Models

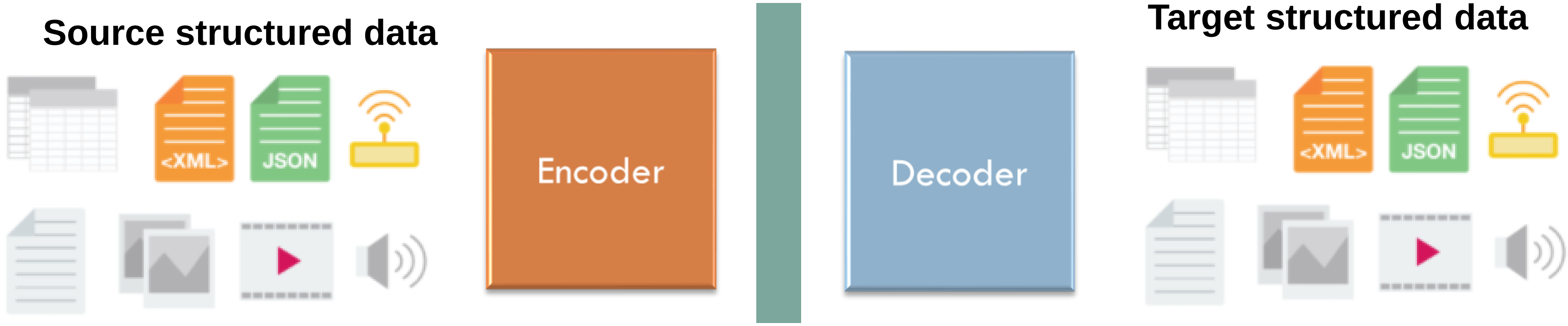
- Image Captioning
- Machine Translation

Encoder-Decoder Models: Application Motivations

Context or **encode** c

Source structured data

Target structured data



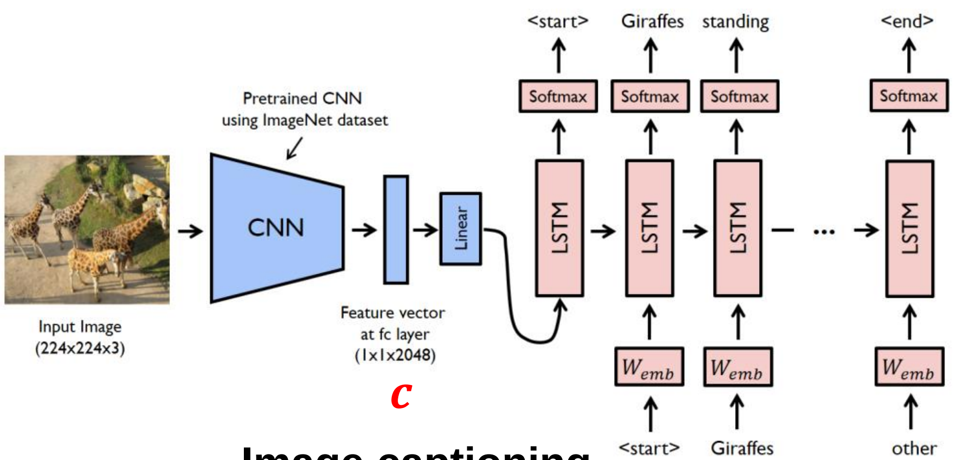
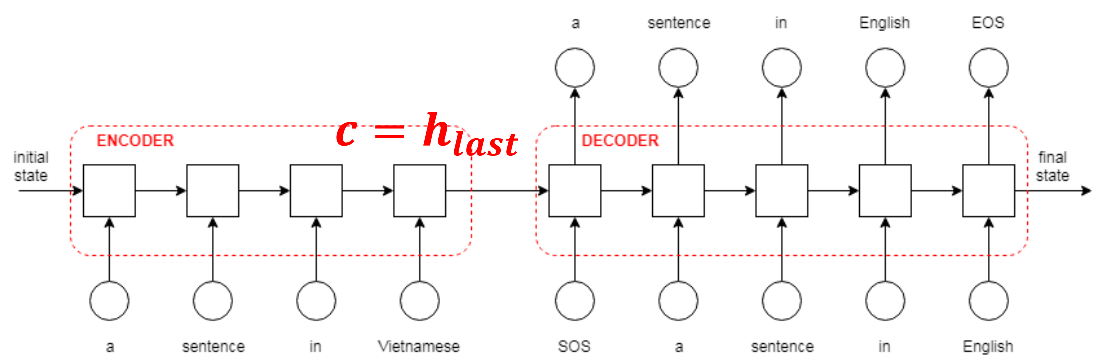


Image captioning



Machine translation

Many more applications

- C++ programs to Java programs, texts to images, question answering (questions to answers), and etc.

Sequence to Sequence Models

Introduction

- Problem statement of seq2seq
  - **Source sequence**: $x = \left(x_1, x_2, \dots, x_{T_x}\right)$ where $x_i \epsilon V_x$
  - **Target sequence**: $y = \left(y_1, y_2, \dots, y_{T_y}\right)$ where $y_i \epsilon V_y$
  - Training set:
    - $\mathfrak{D} = \left\{\left(x^{(1)}, y^{(1)}\right), \left(x^{(2)}, y^{(2)}\right), \dots, \left(x^{(N)}, y^{(N)}\right)\right\} = \left\{\left(x^{(i)}, y^{(i)}\right)\right\}_{i=1}^{N}$
    - S_x: set of **possible source sequences** ($\left\{x^{(i)}\right\} \subseteq S_x$)
    - S_y: set of **possible target sequences** ($\left\{y^{(i)}\right\} \subseteq S_y$)
  - **Task**
    - Learn a function $f: S_x \rightarrow S_y$
- Applications
  - Machine Translation
  - Image Captioning

Machine translation

Sequence to Sequence Learning with Neural Networks

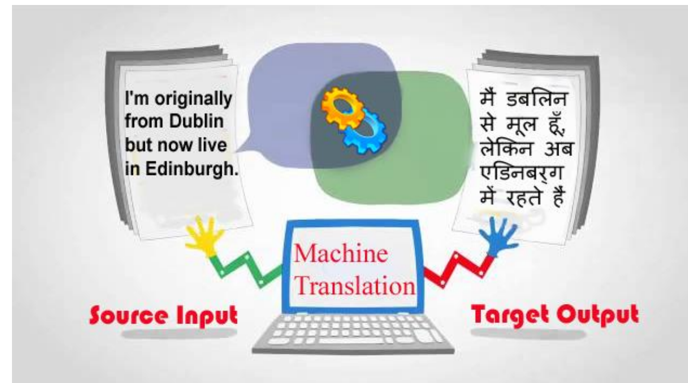
Ilya Sutskever Google ilyasu@google.com

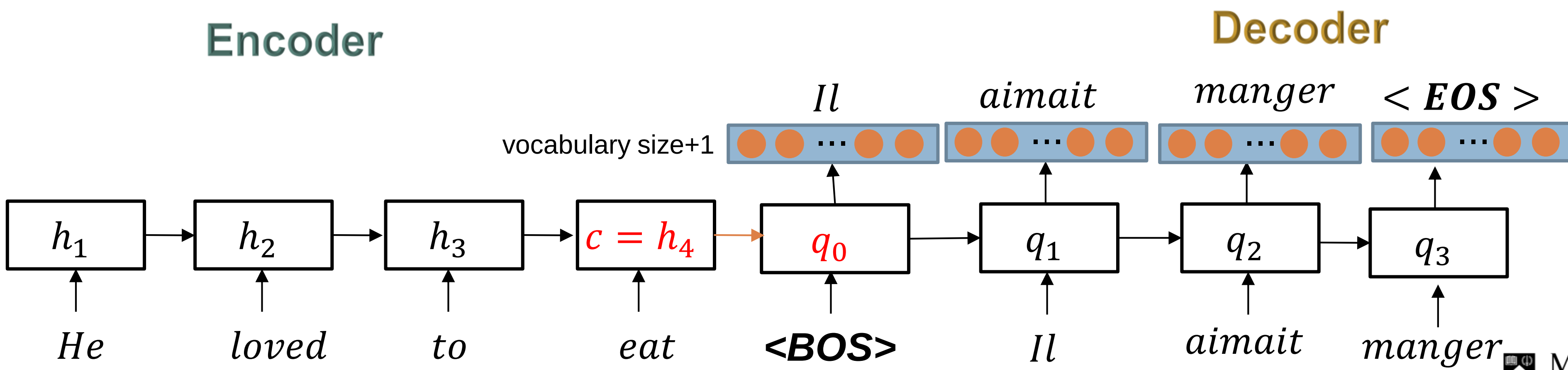
Oriol Vinyals Google vinyals@google.com

Quoc V. Le Google qvl@google.com

Paper: [paper link]

- Translate a sentence in a language to another language
  - **Input**: He loved to eat. (English)
  - **Output**: Il aimait manger. (French)

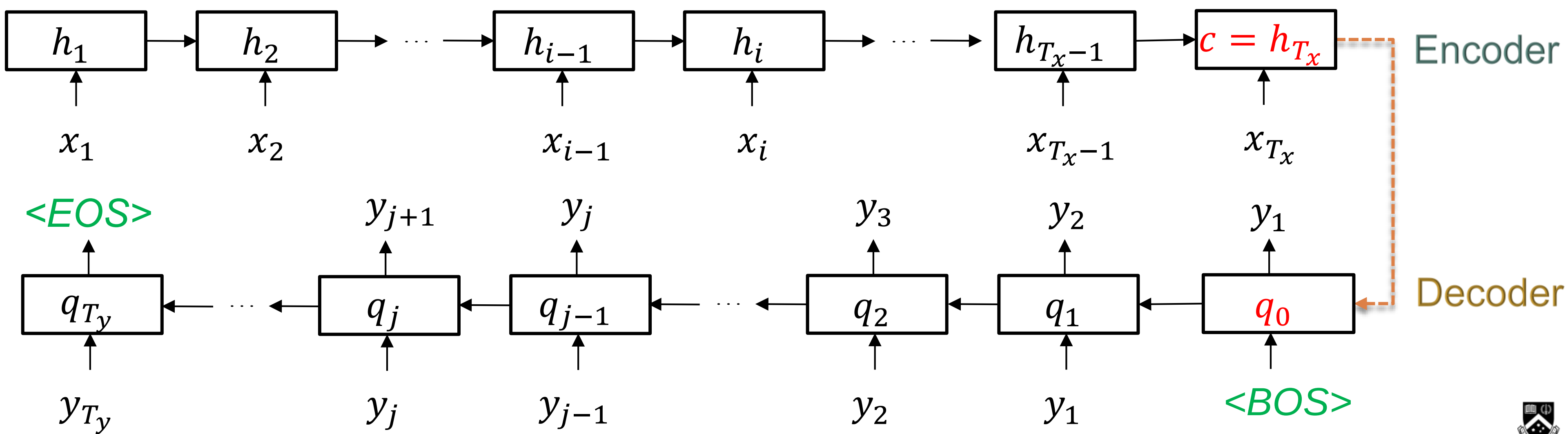




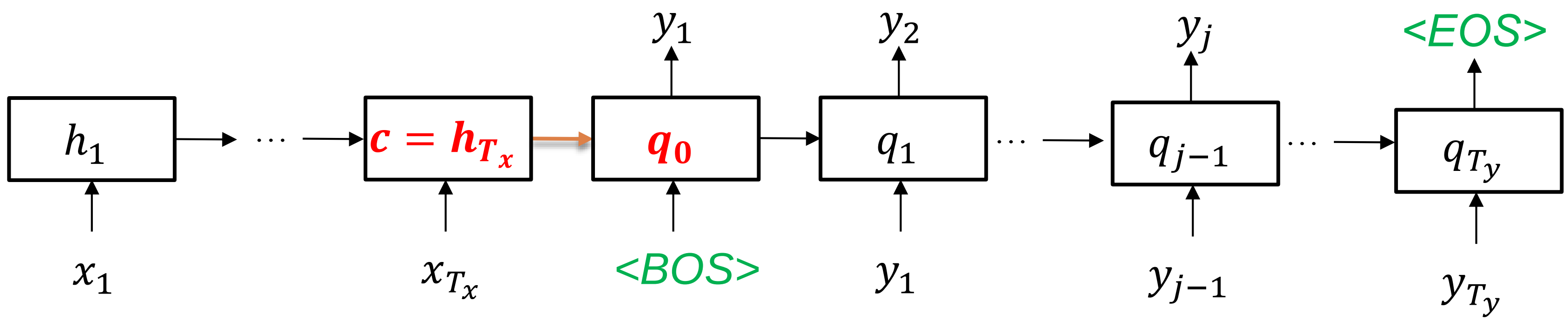
Encoder-decoder model for seq2seq

Fixed context vector

- Encoder
  - Produces the context vector $\boldsymbol{c} = \boldsymbol{h}_{T_x}$ of the input sequence
  - **Context vector c** summarizes input sequence $[\mathbf{x}_1, \dots, \mathbf{x}_{\mathbf{T}_\mathbf{x}}]$.
- Decoder
  - Decodes the encode c to the output sequence
- Special symbols
  - <EOS> signifies the end of a sequence
  - <BOS> signifies the beginning of a sequence



Training of seq2seq



- We need to maximize **the log-likelihood**:

$$\max_{\theta}\left(J(\theta) = \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathfrak{D}} \log P\,(\boldsymbol{y}|\boldsymbol{x},\theta\,)\right)$$

where $\theta = [\theta_e, \theta_d]$ and θ_e, θ_d are encoding and decoding parameters respectively.

- Product rule:

$$P(\boldsymbol{y}|\boldsymbol{x},\theta\,) = P\left(\boldsymbol{y}_{1:T_y} \,\middle|\, \boldsymbol{x}_{1:T_x},\theta\right) = P\left(\boldsymbol{y}_{1:T_y} \,\middle|\, \boldsymbol{c},\theta\right)$$

$$= P(\boldsymbol{y}_1 \mid \boldsymbol{c},\theta\,)P(\boldsymbol{y}_2 \mid \boldsymbol{y}_1,\boldsymbol{c},\theta\,) \ldots P(\boldsymbol{y}_j \mid \boldsymbol{y}_{1:j-1},\boldsymbol{c},\theta\,) \ldots P\left(\boldsymbol{y}_{T_y} \,\middle|\, \boldsymbol{y}_{1:T_y-1},\boldsymbol{c},\theta\right) = \prod_{j=1}^{T_y} P(\boldsymbol{y}_j \mid \boldsymbol{y}_{1:j-1},\boldsymbol{c},\theta\,)$$

$$\log P\,(\boldsymbol{y}|\boldsymbol{x},\theta\,) = \log P\,(\boldsymbol{y}|\boldsymbol{c},\theta) = \sum_{j=1}^{T_y} \log P\,(\boldsymbol{y}_j|\boldsymbol{y}_{1:j-1},\boldsymbol{c},\theta) = \sum_{j=1}^{T_y} \log P\,(\boldsymbol{y}_j|\boldsymbol{q}_{j-1},\boldsymbol{c},\theta)$$

  - We can compute $P(\boldsymbol{y}_j|\boldsymbol{q}_{j-1},\boldsymbol{c}) = g(\boldsymbol{y}_j,\boldsymbol{q}_{j-1},\boldsymbol{c})$ where g is a **nonlinear**, potentially **multi-layered NN** that outputs the probability of y_j.

- Pay attention on how c is used in every step during decoding

Training of seq2seq

- We need to maximize **the log-likelihood**:

$$\max_{\theta}\left(J(\theta) = \sum_{(\boldsymbol{x},\boldsymbol{y})\epsilon\mathfrak{D}} \log P\,(\boldsymbol{y}|\boldsymbol{x},\theta\,)\right)$$

where $\theta = [\theta_e, \theta_d]$ and θ_e, θ_d are encoding and decoding parameters, respectively.

- **Product rule**:

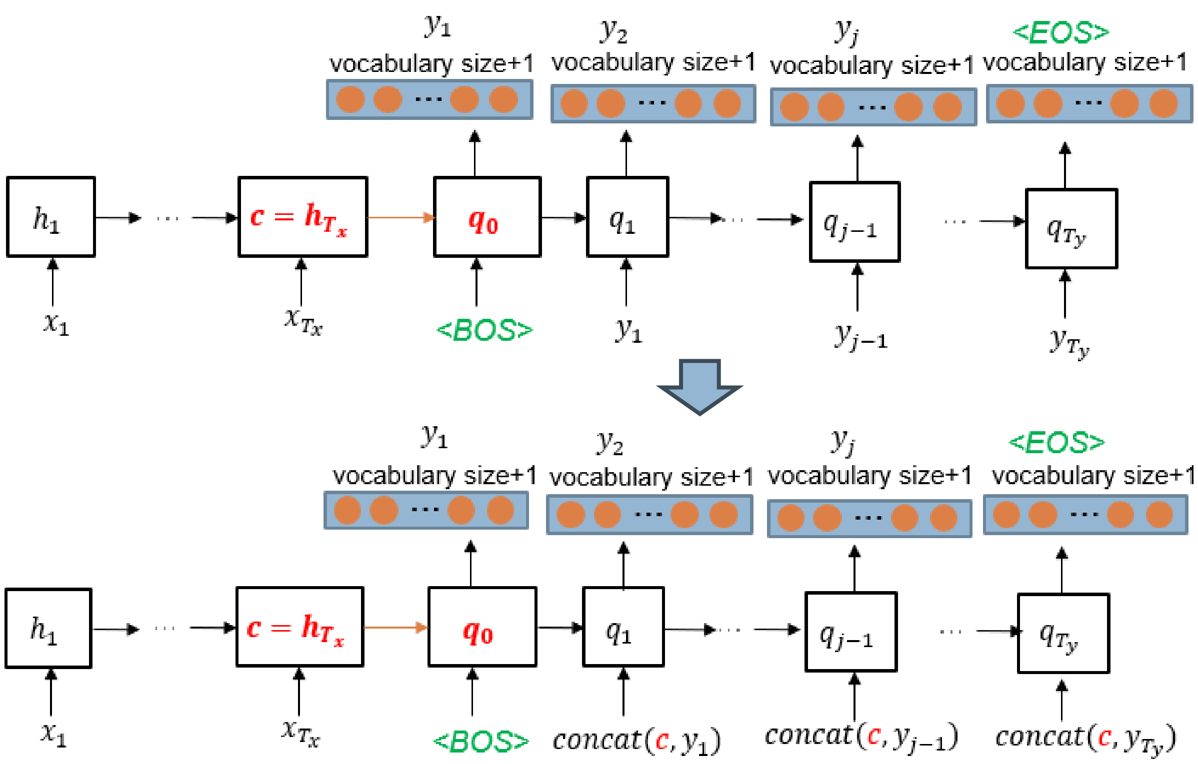
$$\log P\,(\boldsymbol{y}|\boldsymbol{x},\theta\,) = \log P\,(\boldsymbol{y}|\boldsymbol{c},\theta) = \sum_{j=1}^{T_y} \log P\left(\boldsymbol{y}_j|\boldsymbol{y}_{1:j-1},\boldsymbol{c},\theta\right) = \sum_{j=1}^{T_y} \log P\left(\boldsymbol{y}_j|\boldsymbol{q}_{j-1},\boldsymbol{c},\theta\right)$$

  - $\boldsymbol{q}_{j-1}$ contains the information of $\boldsymbol{c}$.
  - (1): $P\left(\boldsymbol{y}_j|\boldsymbol{q}_{j-1},\boldsymbol{c},\theta\right) = P\left(\boldsymbol{y}_j|\boldsymbol{q}_{j-1},\theta\right)$

    $q_{j-1} = f(q_{j-2}, y_{j-1})$
  - (2): $P\left(\boldsymbol{y}_j|\boldsymbol{q}_{j-1},\boldsymbol{c},\theta\right) = P\left(\boldsymbol{y}_j|\boldsymbol{q}_{j-1},\theta\right)$

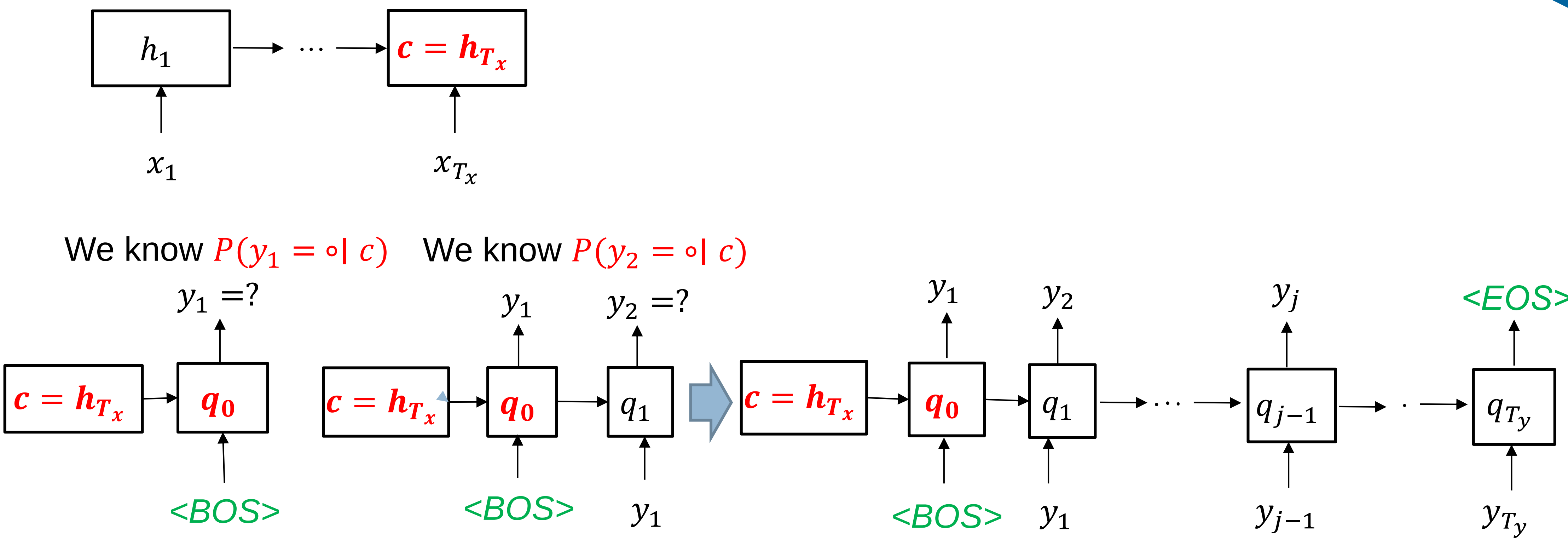
    $q_{j-1} = f(q_{j-2}, concat(c, y_{j-1}))$
  - f is a **memory cell** (e.g., LSTM or GRU)

(1) [Sutskever et al. 2014]



Big drawback: the **context vector** $\boldsymbol{c}$ is **fixed** across timesteps.

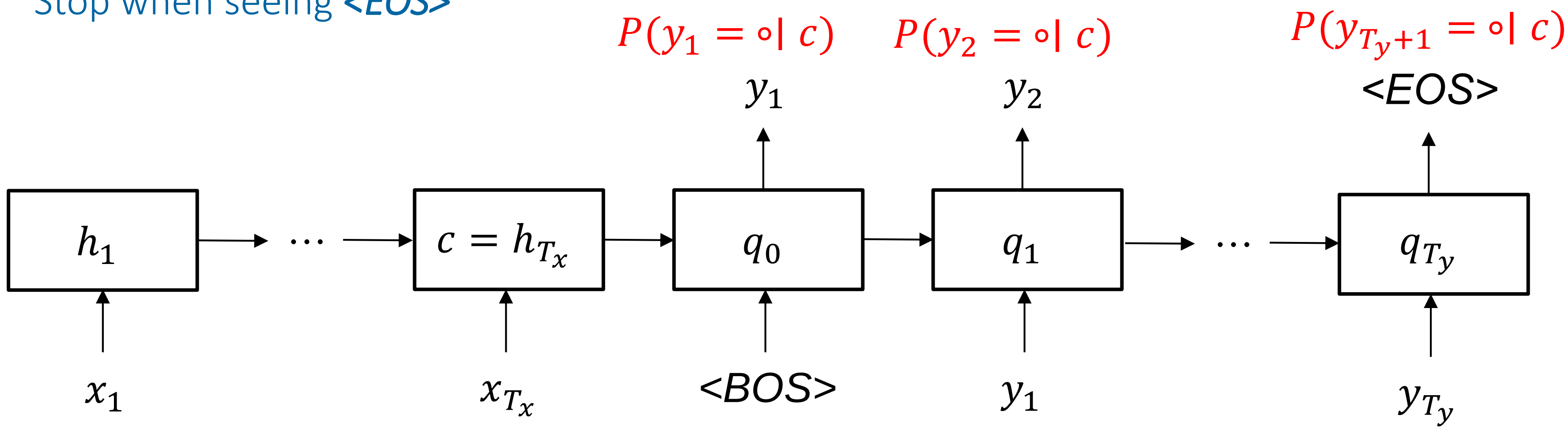
Inference



- Given a **trained model** and **input sequence $\boldsymbol{x}$**
  - We need to infer the corresponding **output sequence $\boldsymbol{y}$**
- Two common strategies
  - **Greedy Decoding** and **Beam Search Decoding**

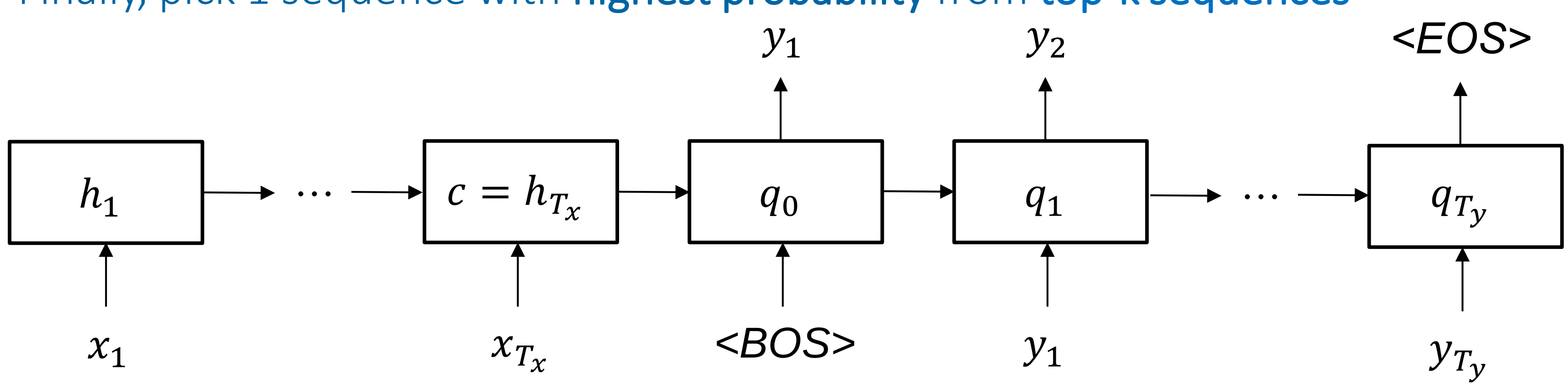
Inference
Greedy decoding

- Greedy Decoding
  - Given **x**, find word $\boldsymbol{y_1}$ with highest probability
  - Given $\boldsymbol{y_1}$ and **x**, find word $\boldsymbol{y_2}$ with highest probability
  - …
  - Stop when seeing ***<EOS>***



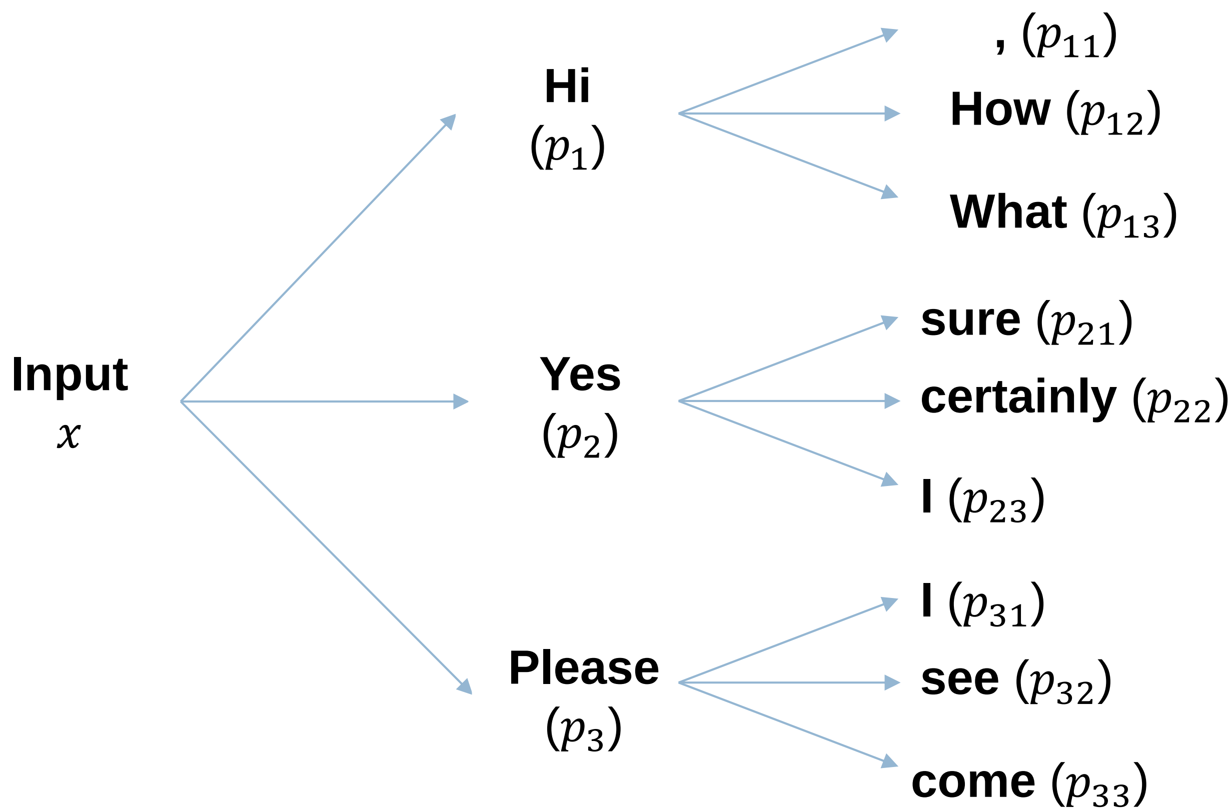
Inference
Beam search

- Beam Search Decoding with **beam width $\boldsymbol{k}$**
    - Given **x**, find **k** candidates for $\boldsymbol{y_1}$ with highest probability
    - Given **x**, for each candidate $\boldsymbol{y_1}$, find **k** candidates for word $\boldsymbol{y_2}$ with highest probability
    - Pick top-**k** sequences $y_1 y_2$ with highest **joint probability**
    - For each $y_1 y_2$, find **k** candidates for word $\boldsymbol{y_3}$ with highest probability
    - Pick top-**k** sequences $y_1 y_2 y_3$ with highest **joint probability**
    - .....
    - Stop when see ***<EOS>*** on each beam
    - Finally, pick 1 sequence with **highest probability** from **top-k sequences**



Inference

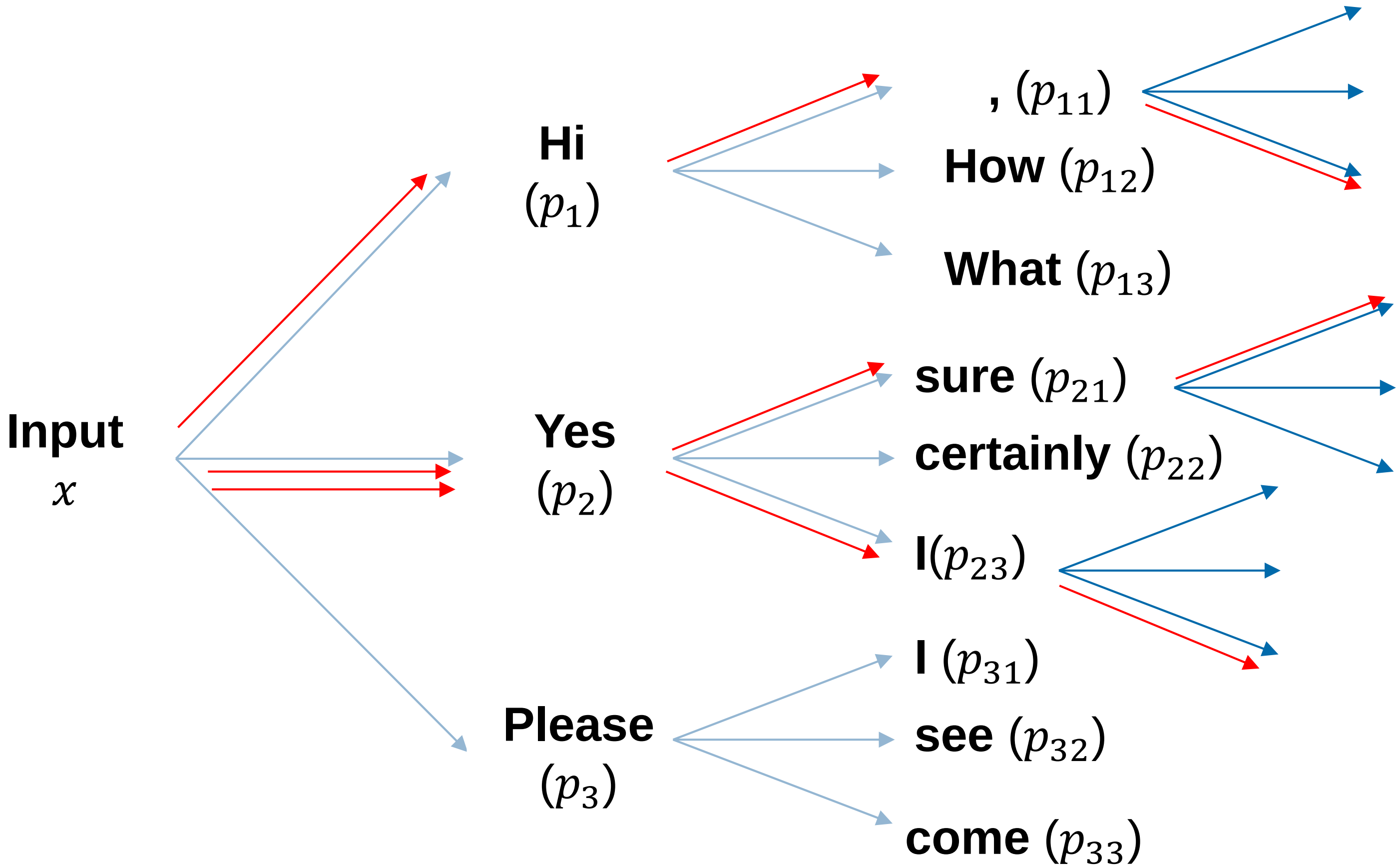
Beam search



Beam width = 3

- We always choose three sentences with **highest joint probabilities**

Inference
Beam search



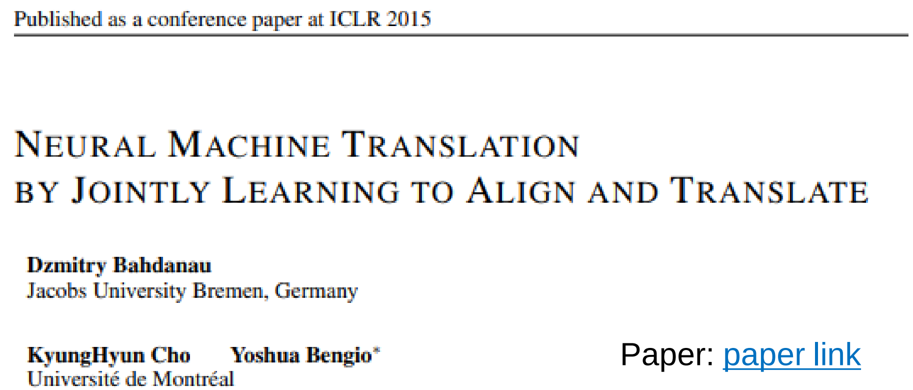
Joint Probability

$p_1 p_{11} \cdots$
$p_1 p_{12} \cdots$
$p_1 p_{13} \cdots$
$p_2 p_{11} \cdots$
$\cdots$

Beam width = 3

- We always choose three sentences with **highest joint probabilities**

Drawback of fixed context



Published as a conference paper at ICLR 2015

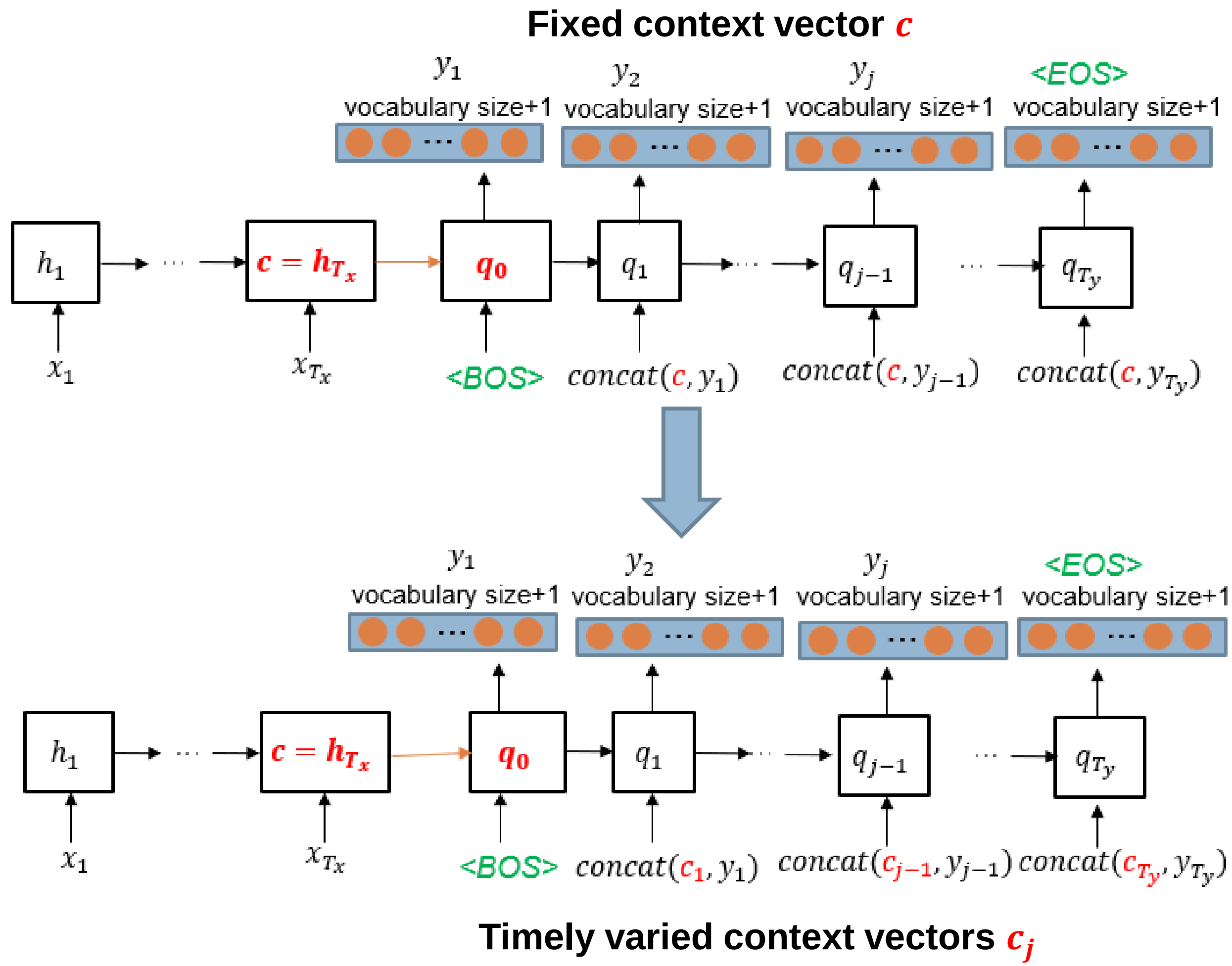
NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

KyungHyun Cho Yoshua Bengio*
Université de Montréal

Paper: paper link

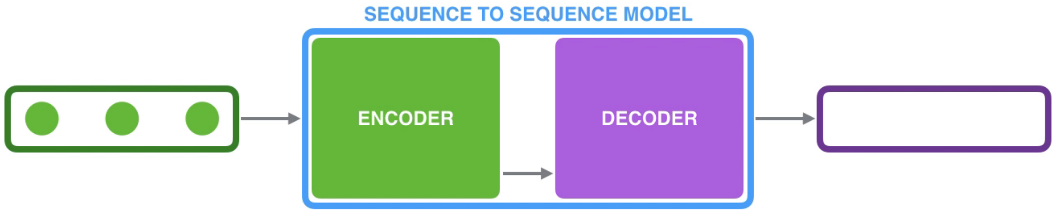
- **Fixed** context vector $\boldsymbol{c}$ is **easily overwhelmed** by long inputs or long outputs.
- At a specific timestep j, some **words** or **items** in the **input sequence** might **possibly contribute more** to the **generation** of **next item** or **word** in the output sequence.
  - I want to see you every day → Je veux te voir chaque jour
  - I want to see **you** every day → Je veux te **?** (voir) ....
- How to **timely adapt** the **context vector** c_j?
  - $c_j = \alpha(h_1, .., h_{T_x}, q_{j-1})$
  - Computed using **attention mechanism**



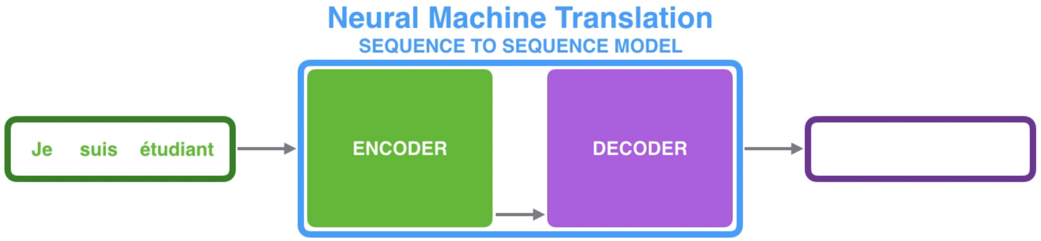
Attention mechanism

Attention mechanism

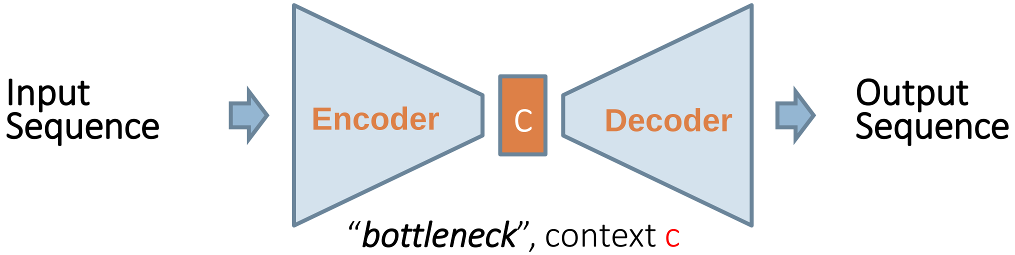
- So far, the input sequence is **summarised** by a **single** context c !



- **Fixed-length context** could be problematic as it is **easily overwhelmed** by long inputs or long outputs
  - The **fixed-length** context **c** might **not be powerful enough** to capture long input sequences



- Some **specific items** in an input sequence might be **more relevant** and **contributing in generating** a given item in output sequence.



- Gratefully acknowledge the excellent visualizations used from Jay Alammar blog at:
  - https://jalammar.github.io/
  - https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Attention mechanism

Published as a conference paper at ICLR 2015

NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

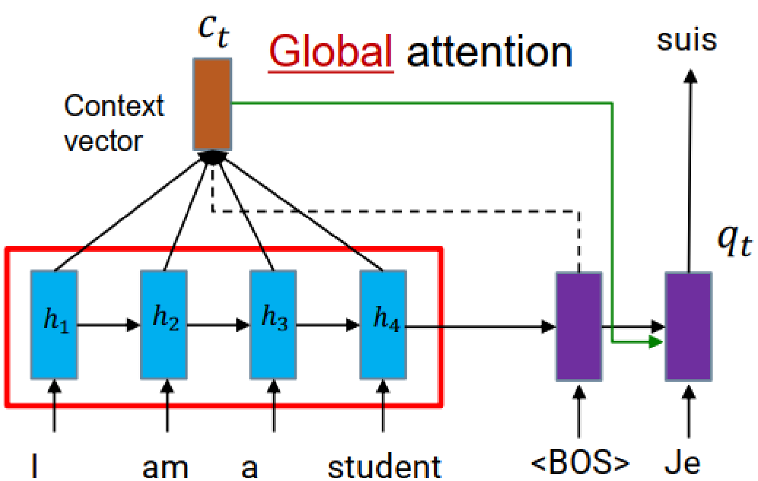
KyungHyun Cho **Yoshua Bengio***
Université de Montréal

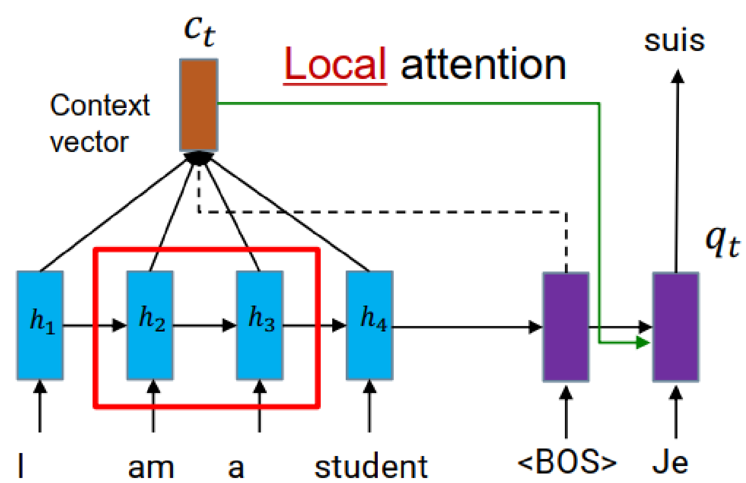
Bahdanau, Cho, Bengio, [Neural Machine Translation by Jointly Learning to Align and Translate], ICLR 2015

Effective Approaches to Attention-based Neural Machine Translation

Minh-Thang Luong **Hieu Pham** **Christopher D. Manning**
Computer Science Department, Stanford University, Stanford, CA 94305
{lmthang,hyhieu,manning}@stanford.edu

Luong, Pham, Manning, [Effective Approach Attention-based Neural Machine Translation], EMNLP, 2015

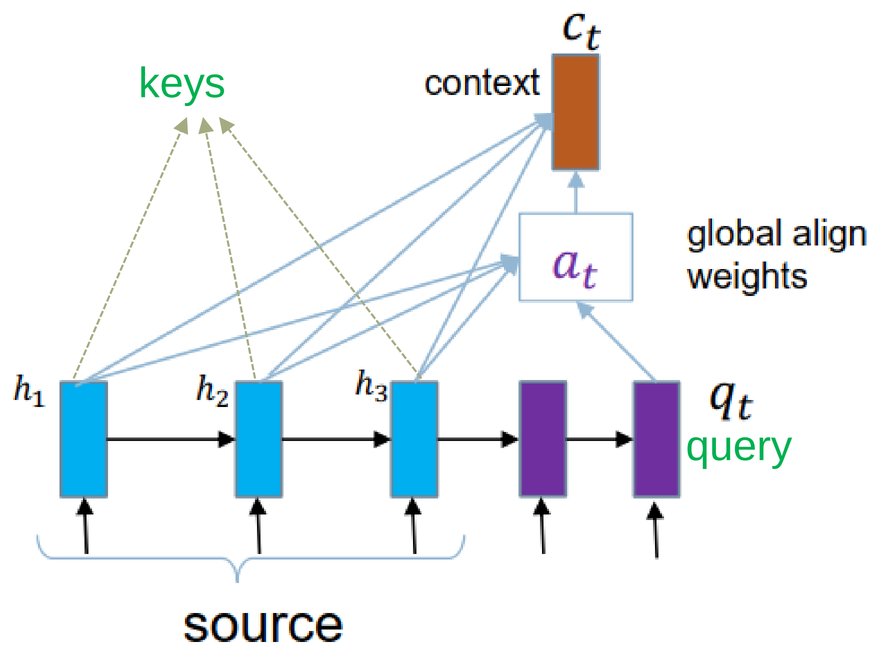




- **Attention mechanism allows the decoding network to refer to the input.**
  - **Global attention**
    - Use **all input hidden states** of the encoder when deriving the context c_t.
  - **Local attention**
    - Use a **selective window of input hidden states** of the encoder when deriving the context c_t.

Global attention

- Main idea
  - Consider **all input hidden states** of the encoder when deriving the context: $c_t = \sum_{s=1}^{3} a_t(s)\, h_s$



- Alignment weights:

$$a_t(s) = \text{align}(q_t, h_s) = \frac{\exp(score(q_t, h_s))}{\sum_{s'} \exp(score(q_t, h_{s'}))}$$

where q_t is current target state, h_s is each source states.

- **Alignment score** function:

$$score(q_t, h_s) = \begin{cases} q_t^{\mathrm{T}} h_s & \textit{dot product} \\ q_t^{\mathrm{T}} W_a h_s & \textit{general metric} \\ v_a^{\mathrm{T}} tanh(W_a[q_t; h_s]) & \textit{concat} \end{cases}$$

Global attention

- Context vector:

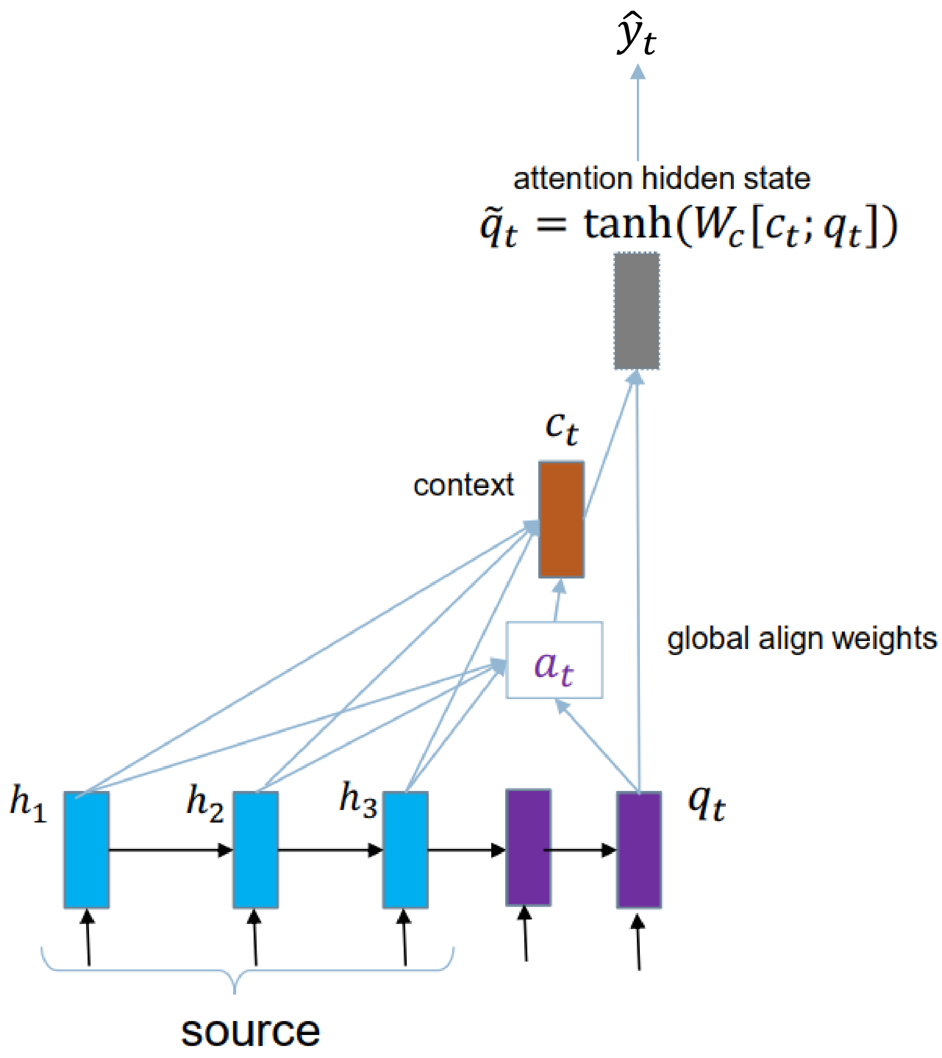
$$c_t = \sum_{s=1}^{3} a_t(s)\, h_s$$

- Attentional hidden state:

$$\tilde{q}_t = tanh(W_c[c_t; q_t])$$

- Predictive distribution:

$$p(y_t|y_{<t}, x) = softmax(W_s\tilde{q}_t)$$



Global attention

Example

1. Convert into alignment weights.

$$a_t(s) = \frac{exp(score(q_t, h_s))}{\sum_{s'} exp(score(q_t, h_{s'}))}$$

2. Build context vector: weighted average

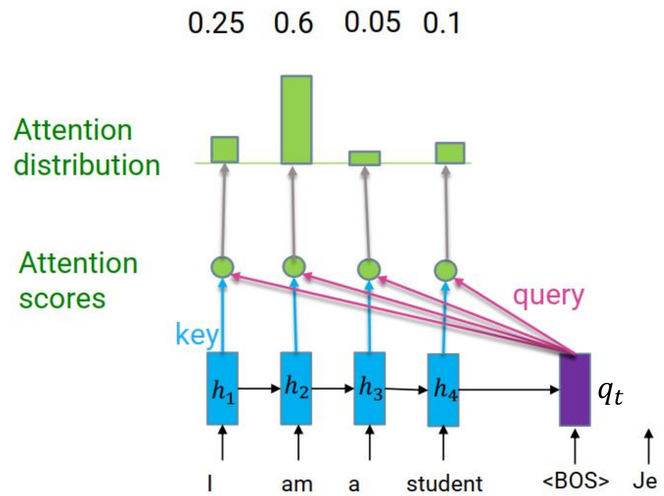
$$c_t = \sum_s a_t(s)\, h_s$$

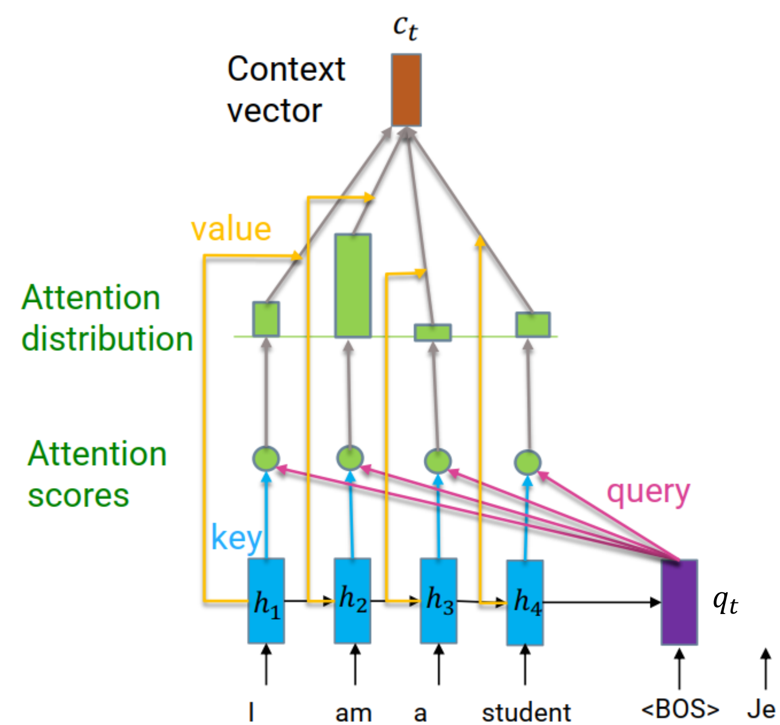
3. Compute the next hidden state.

$$\tilde{q}_t = tanh(W_c[c_t; q_t])$$

4. Predictive distribution:

$$p(y_t|y_{<t}, x) = softmax(W_s \tilde{q}_t)$$



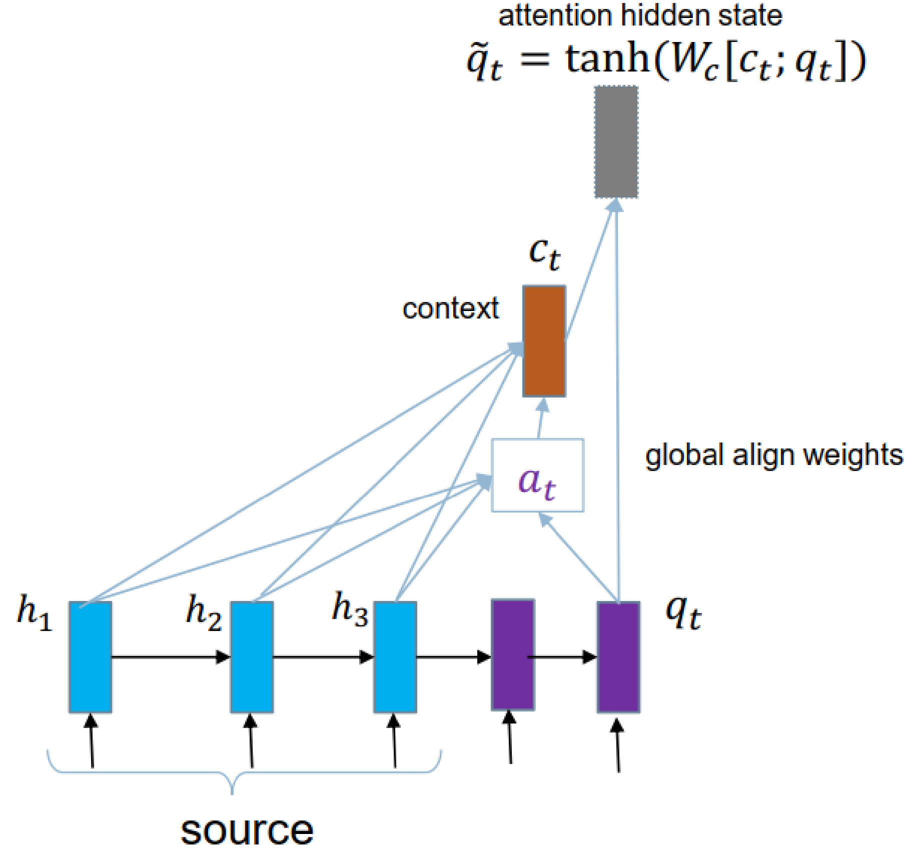


MONASH University

Global attention
Drawback

- **Drawback**: employ **all items** on the source side for deriving each target item.
  - expensive computation
  - impractical to translate longer sequences

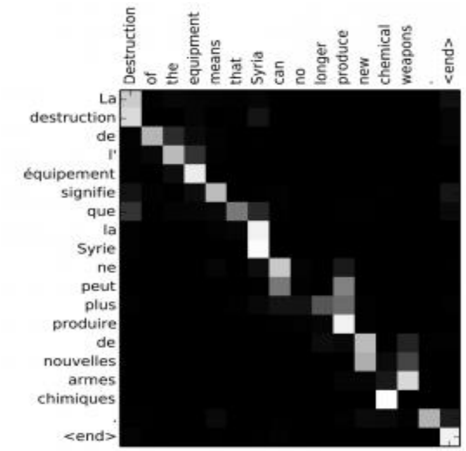
Local attention

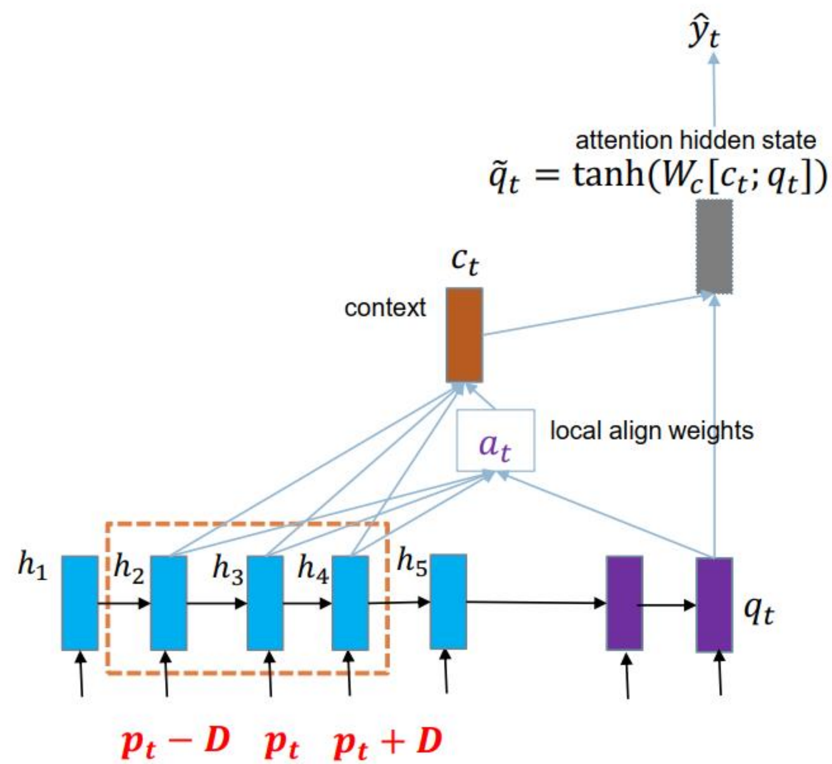


Local attention

- Main idea
  - Selectively focuses on a **small window** of context and is differentiable.

- c_t is then derived as a **weighted average** over the set of **source hidden states within the window** $[\boldsymbol{p_t} - \boldsymbol{D}; \boldsymbol{p_t} + \boldsymbol{D}]$, D is empirically selected.

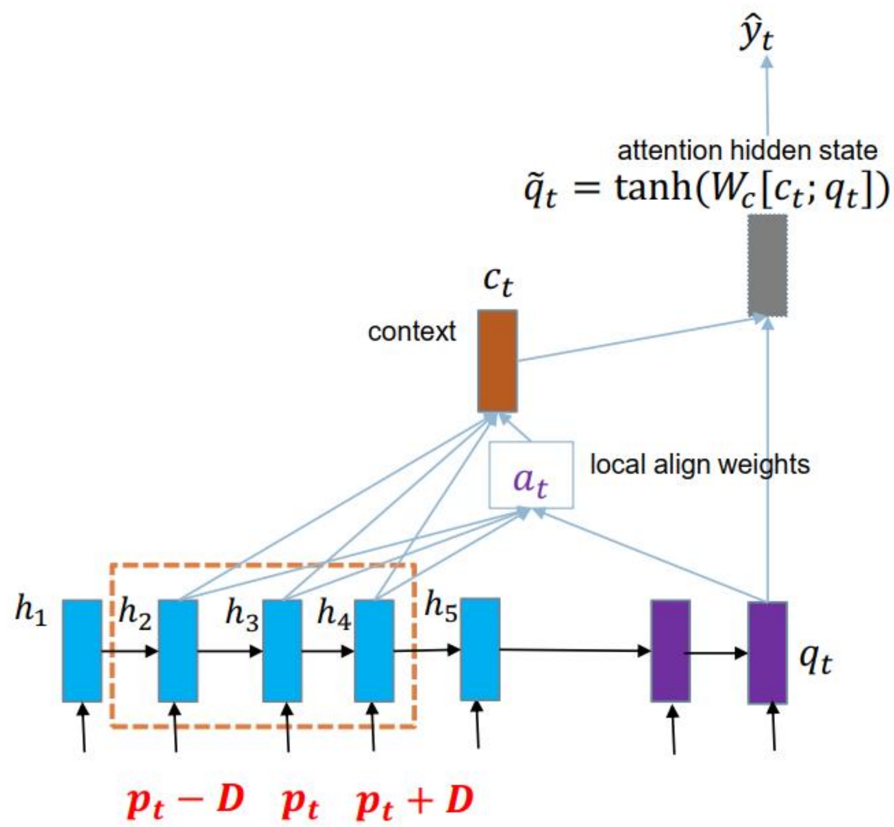




Local attention

- Main idea
  - Selectively focuses on a **small window** of context and is differentiable
- For current target word, predict aligned position p_t
  - $p_t = S \cdot \text{sigmoid}\left(v_p^T \tanh\left(W_p q_t\right)\right)$

  where W_p and v_p are the model parameters, S is the source sentence length, and hence $0 \leq p_t \leq S$
- Alignment weights:
  - $a_t(s) = \text{align}(q_t, h_s)\exp\left(-\frac{(s-p_t)^2}{2\sigma^2}\right)$ with $\sigma = {}^{D}/_{2}$
  - $\text{align}(q_t, h_s) = \frac{\exp(score(q_t, h_s))}{\sum_{s'} \exp(score(q_t, h_{s'}))}$ (**temporary alignment weights**)
- Context is now **computed** as before but **within a windows of size D**, centered at p_t



Other types of attention mechanism

- Self-attention/Multiple Heads/Transformer Networks
  - "Attention is all you need. Vaswani, Ashish, et al. ." *Advances in neural information processing systems,* 2017.
- Pyramidal encoders
  - "Listen Attend and Spell". William Chan, Navdeep Jaitly, Quoc Le, Oriol Vinyals, ICASSP 2015.
- Hierarchical Attention
  - "Learning efficient algorithms with hierarchical attentive memory". Andrychowicz, Marcin, and Karol Kurach, arXiv:1602.03218, 2016.
- Soft/Hard Attention
  - "Show, attend and tell: Neural image caption generation with visual attention". Xu, Kelvin, et al, ICML 2015.


Q&A
When? Why? How? WHAT? Why? When?
WHAT? WHERE? WHEN? How? HOW? WHEN? When? WHERE?
When? WHERE? WHAT? Why? When? WHAT?
WHO? WHEN?
WHERE? What?
Why? HOW? Where? Where? WHERE? When? What? What? WHERE? WHERE? When? HOW?
Why? What? When?
WHEN? WHEN? HOW? When? When? HOW? WHAT? Why?

Transformer and BERT

Question

What does **GPT** stands for in **ChatGPT**?

Transformer and BERT

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Paper: paper link

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin **Ming-Wei Chang** **Kenton Lee** **Kristina Toutanova**
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

Paper: paper link

- Transformer proposed **self-attention** mechanism
- BERT was **developed** based on Transformer
  - BERT currently achieves **SOTA performances** in most tasks of NLP.
- Driving force of generative AI revolution

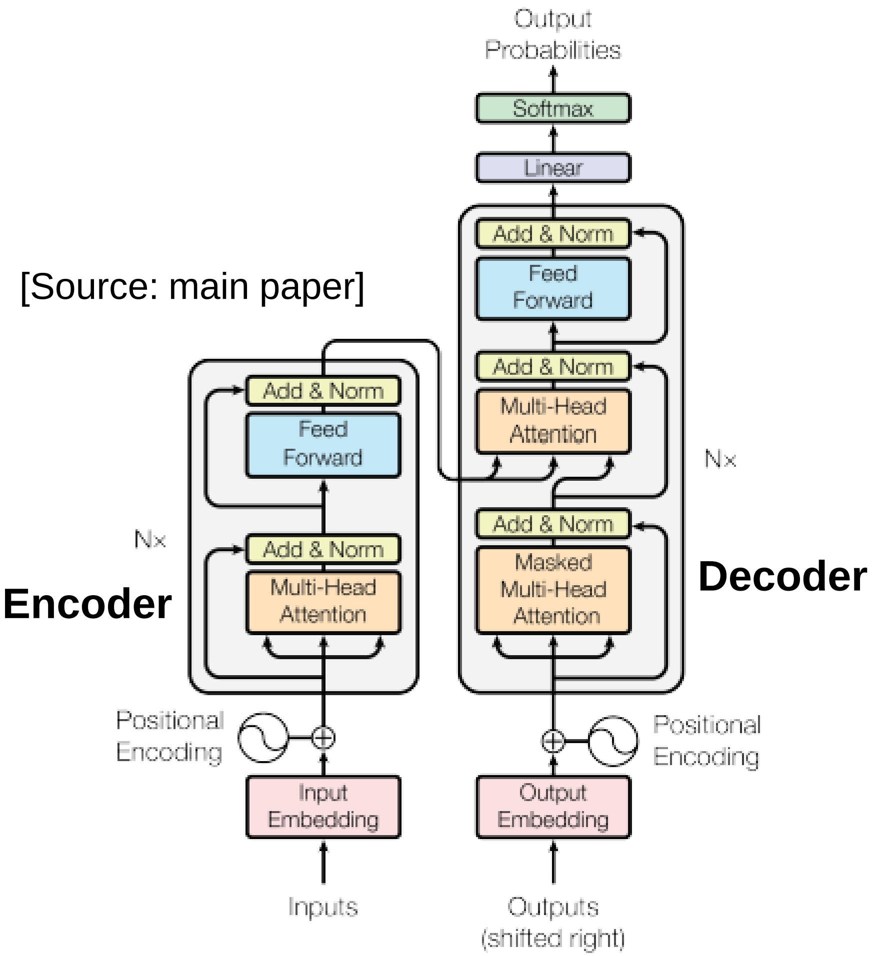
Transformer Motivations

- Problem using RNNs for Seq2Seq models
  - Slow due to sequential nature
  - Poor long-range dependency modelling

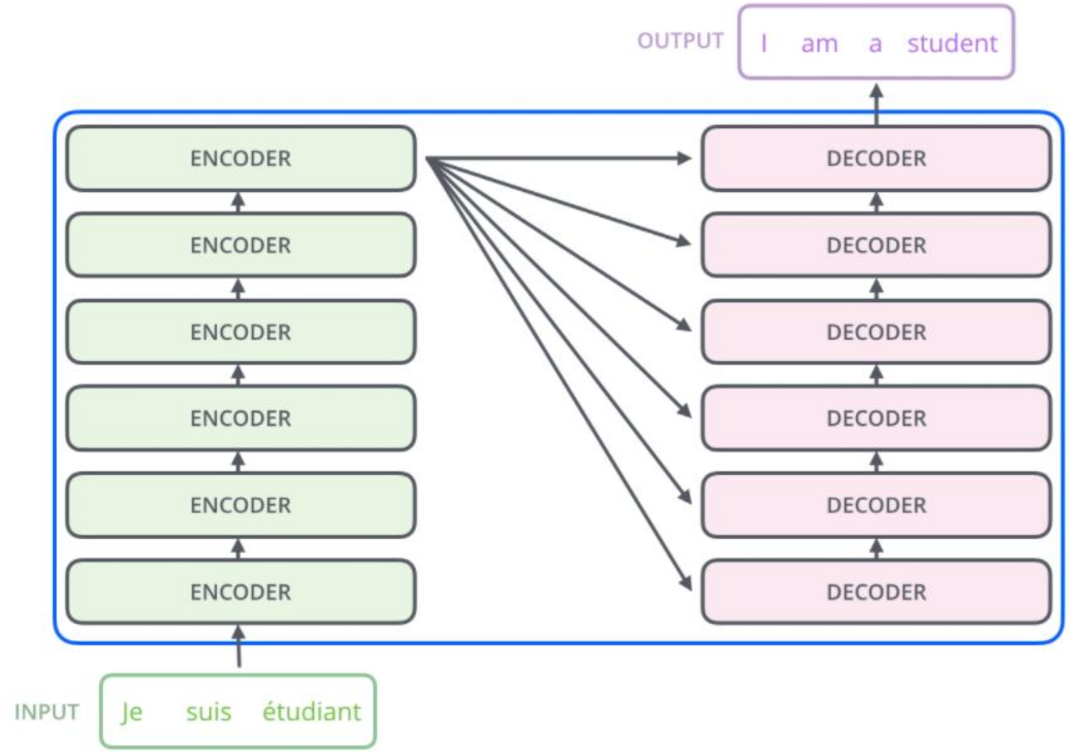
- Transformer
  - Enable **parallel computation**, hence exploit the power of GPU. How?
  - Remove the **dependency** between words
  - But **position/location** matters → resolve by **positional encoding**
  - Extremely good at capturing **long-range dependency**

Transformer – General Architecture

[Source: main paper]



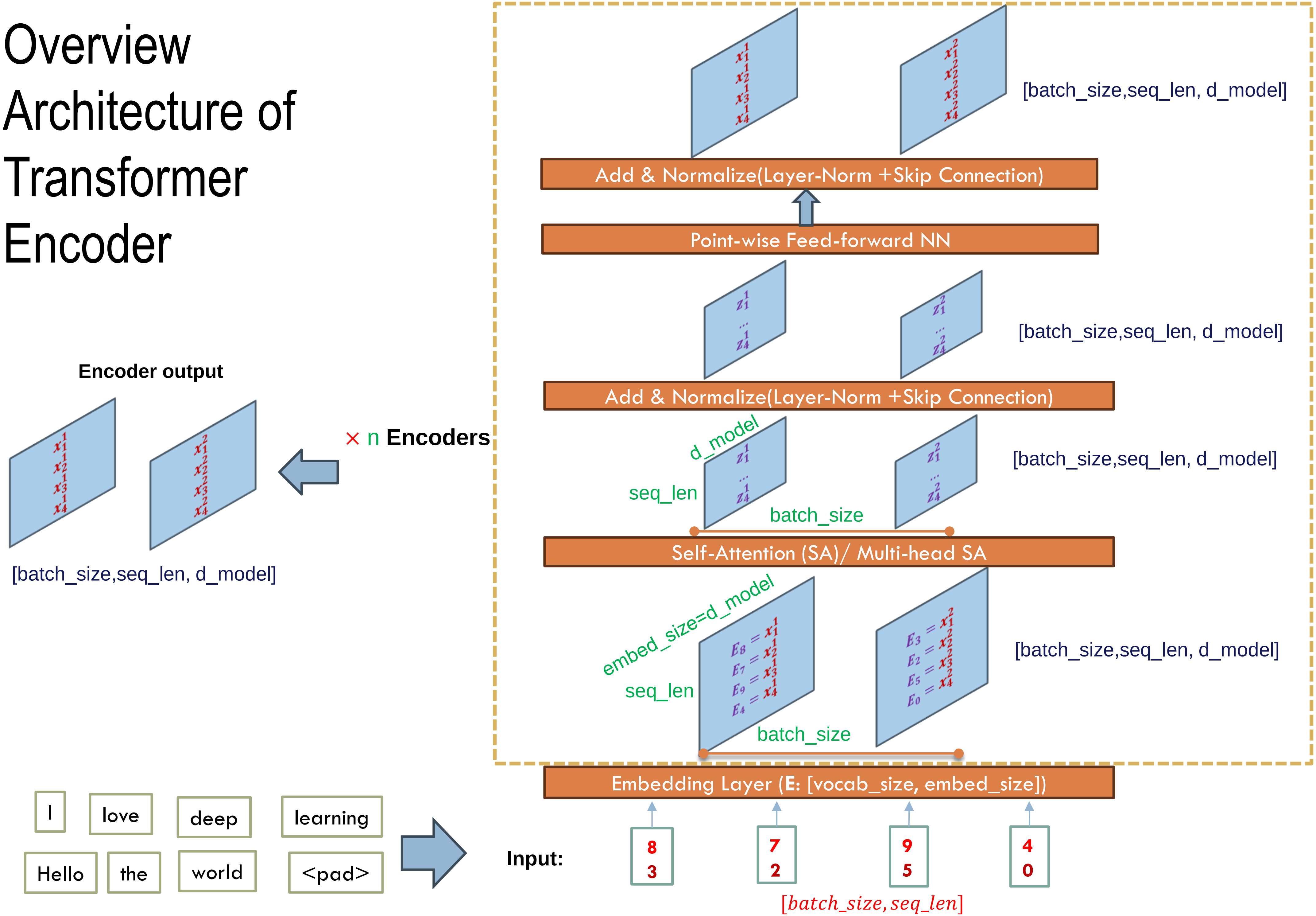
Model architecture



[Source: https://jalammar.github.io/illustrated-transformer/]

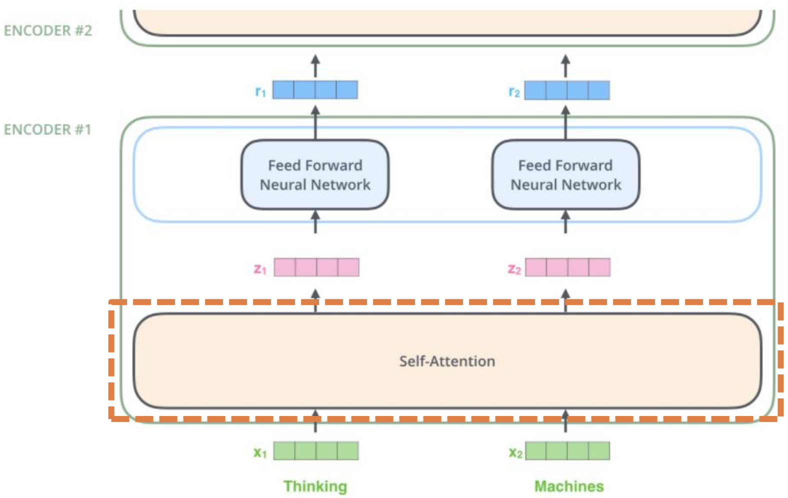
- Some **building blocks**
  - Positional encoding
  - Self-attention and Multi-Head Self-attention Attention
  - Masked Multi-Head Attention
  - Residual add and layer norm

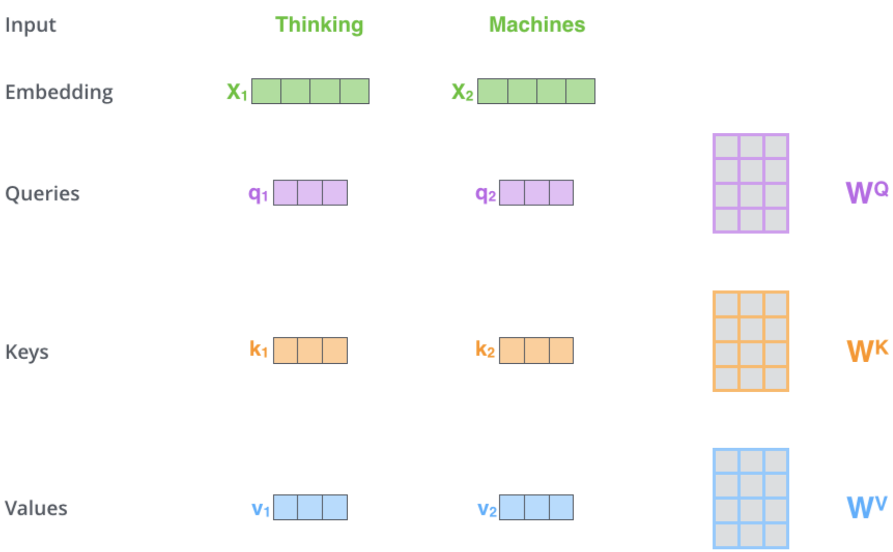
Overview Architecture of Transformer Encoder



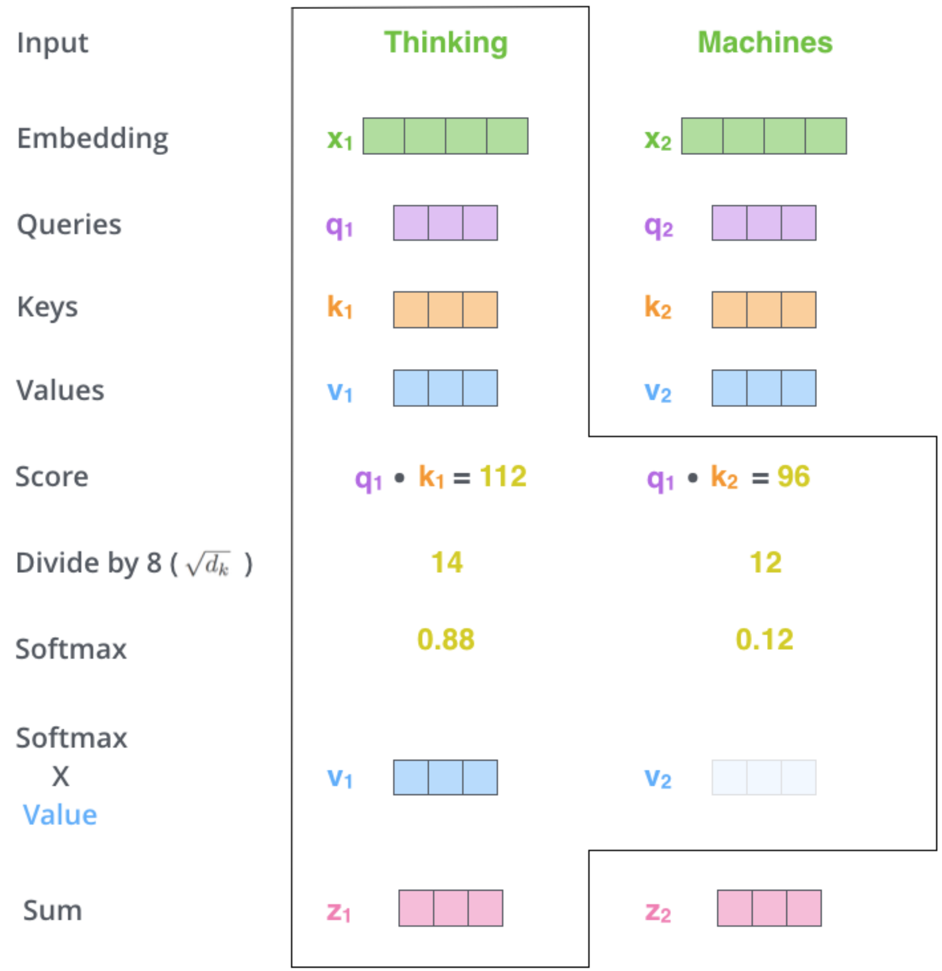
Transformer – Self Attention

[Source: https://jalammar.github.io/illustrated-transformer/]





Multiplying $\boldsymbol{x_1}, \boldsymbol{x_2}$ with $\boldsymbol{W^Q}, \boldsymbol{W^K}, \boldsymbol{W^V}$ to gain **queries**, **keys**, and **values**.



$$z_1 = 0.88 \times v_1 + 0.12 \times v_2$$

- **Self attention** among a sequence of **items/tokens**
  - **Keys** and **queries** for computing self attention weights
  - W^Q, W^K, W^V are **learnable matrices**.

Transformer

Self-Attention

- For each token x_i, calculate its query, key, and value

  $q_i = x_i W^Q, k_i = x_i W^K, v_i = x_i W^V$

- **Matrix/stacked** version ($X = \begin{bmatrix} x_1 \\ \cdots \\ x_L \end{bmatrix}, L = seq\_len$)

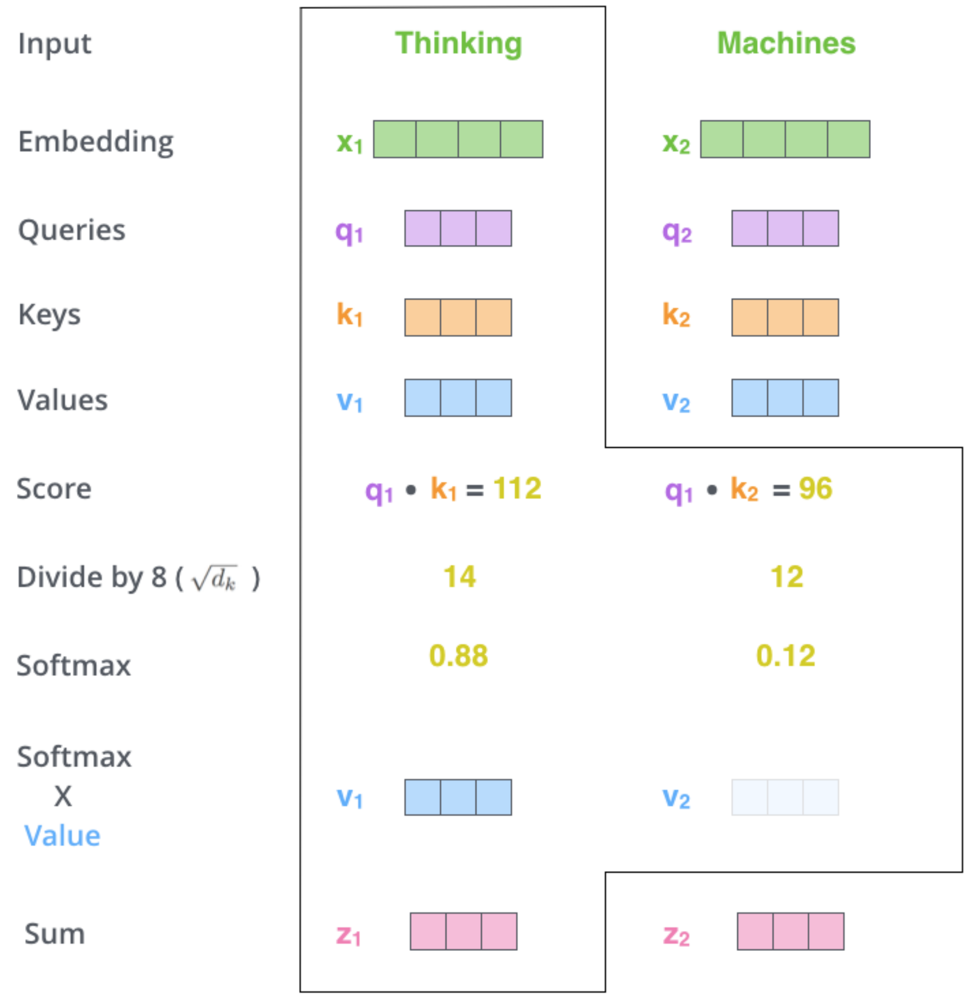
  $Q = XW^Q, K = XW^K, V = XW^V$

- Calculate the **attention probabilities** between query and key
  - Scaled dot-product attention

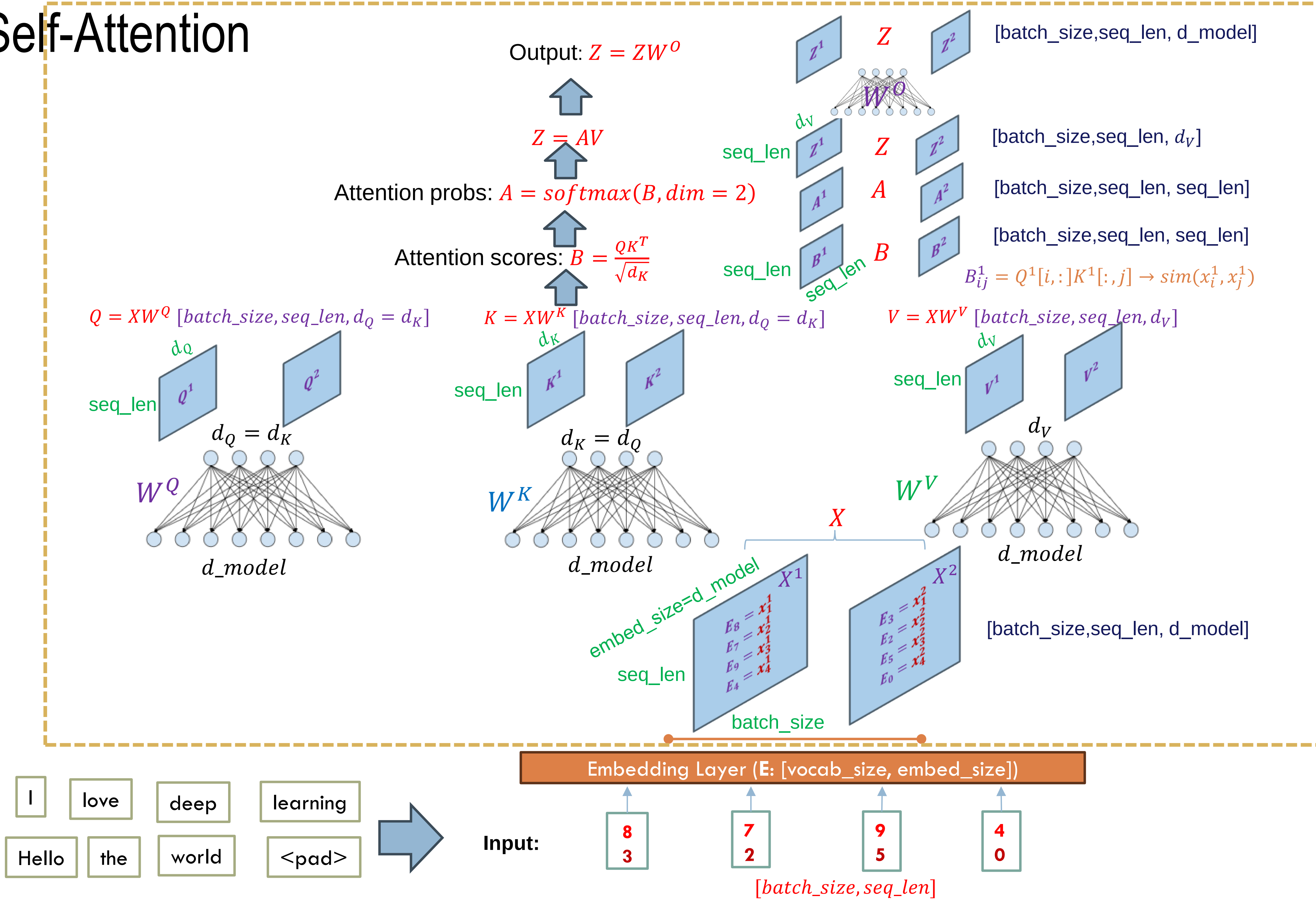
$$A = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

- Take a **weighted sum** of values

$$Z = AV$$



Self-Attention

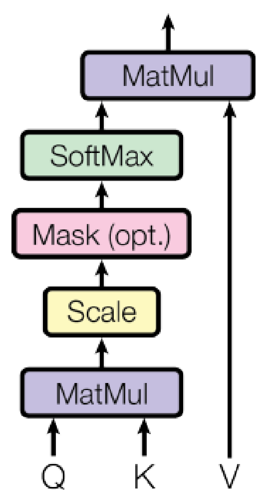


Transformer

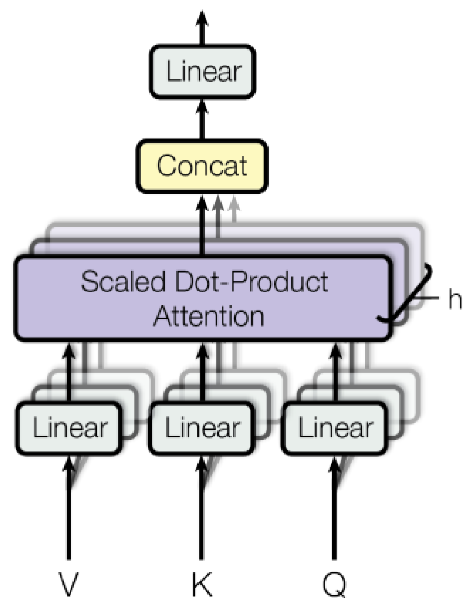
Multi-Head Self-Attention

- Main idea:
  - Perform self-attention multiple times in parallel and combine results
- Multiple attention heads through multiple Q, K, V matrices
- Each attention head performs attention independently
  - Allow attending to parts of the sequence differently

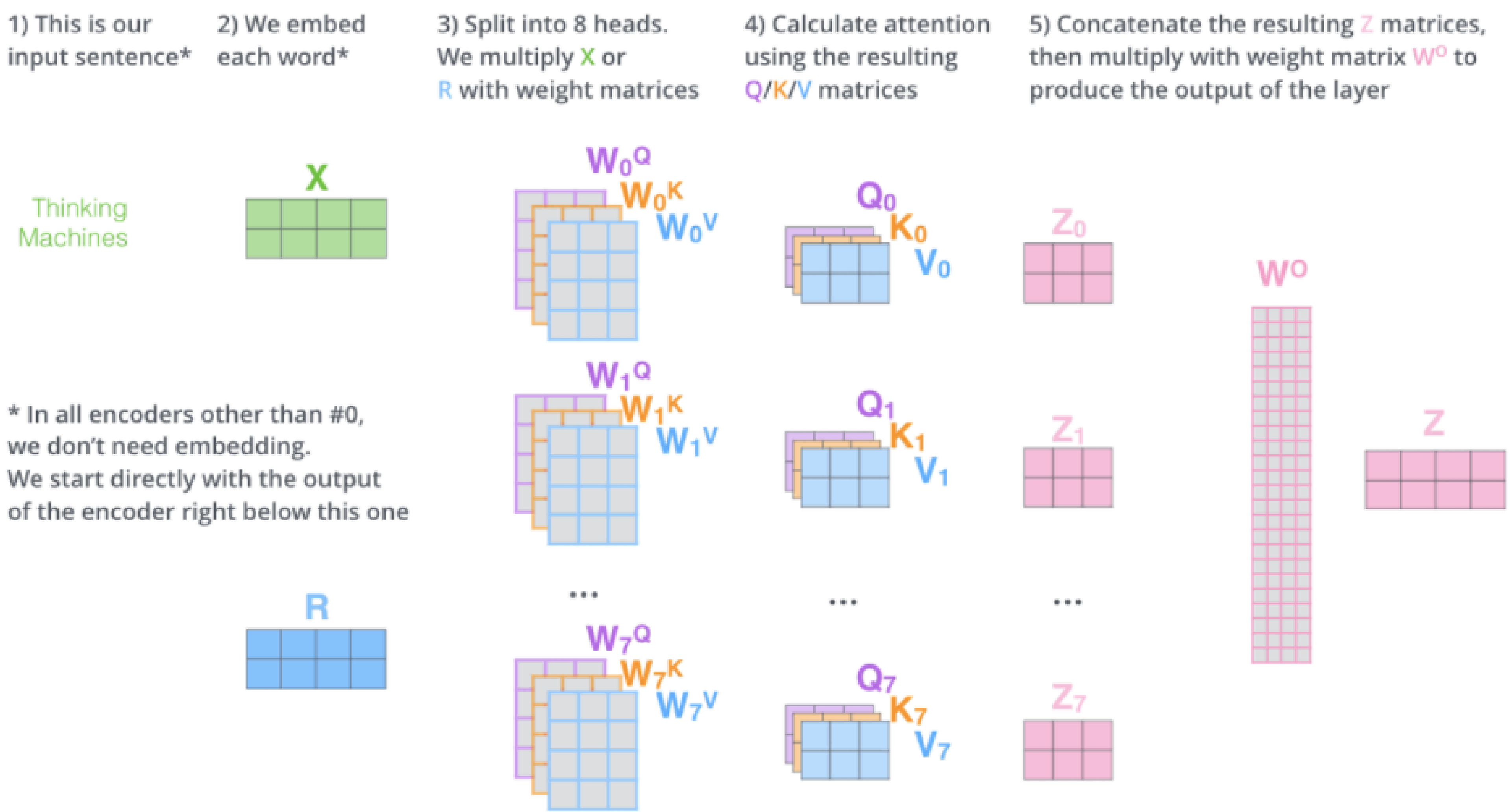
Scaled Dot-Product Attention



Multi-Head Attention



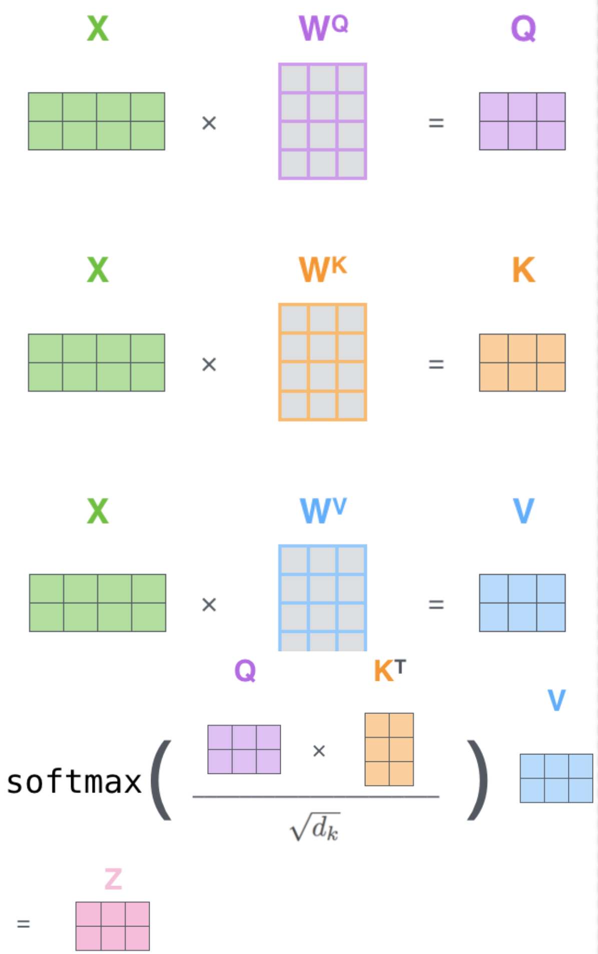
Transformer – Self Attention/Multi-Head Attention



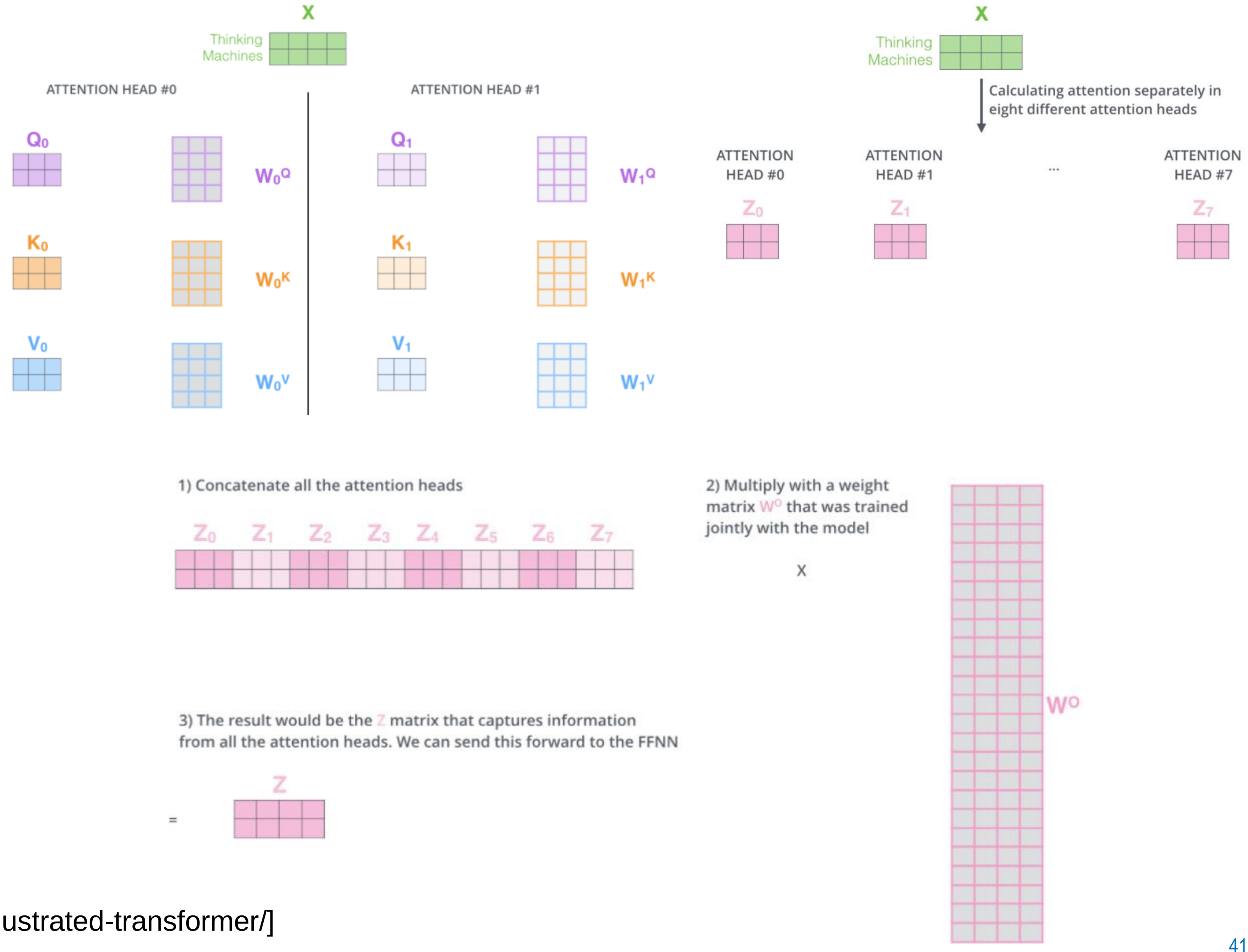
[Source: https://jalammar.github.io/illustrated-transformer/]

Transformer – Self Attention/Multi-Head Attention

Matrix calculation for One-Head

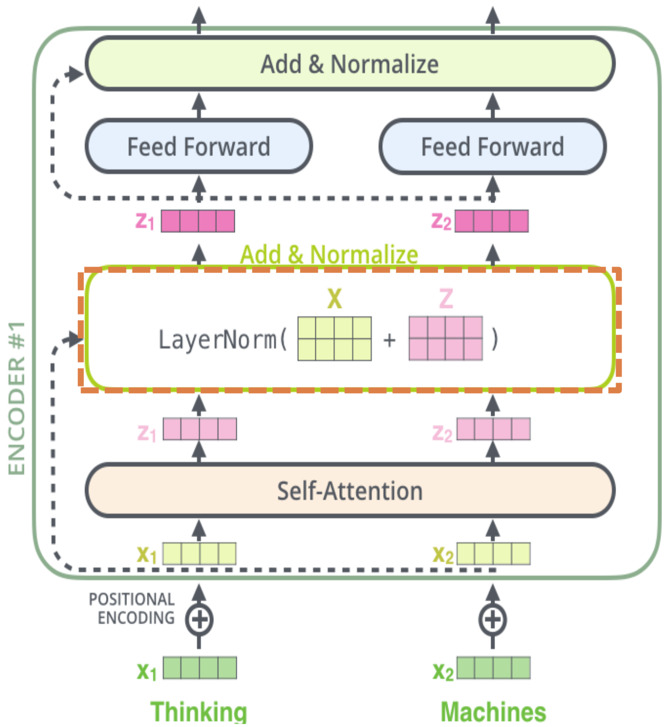


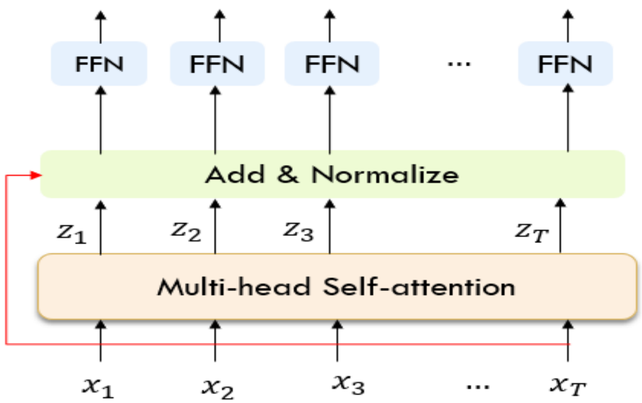
Matrix calculation for Multi-Head

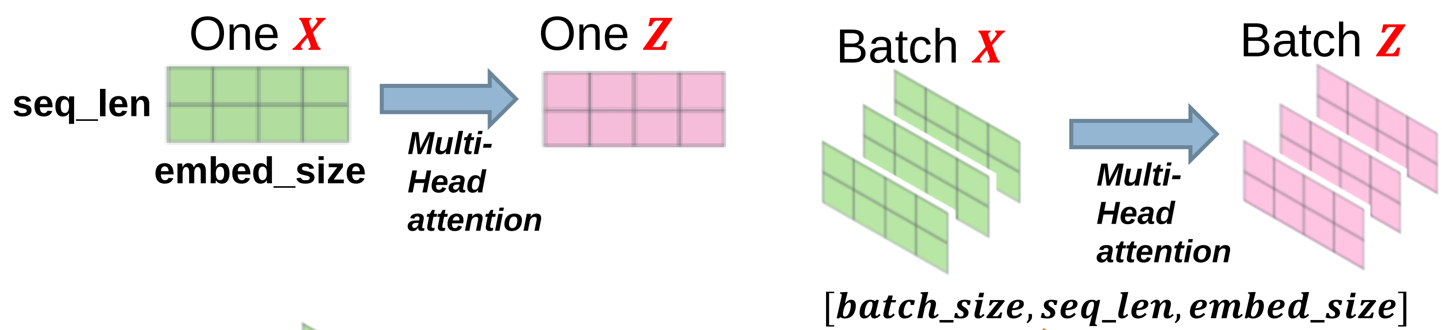


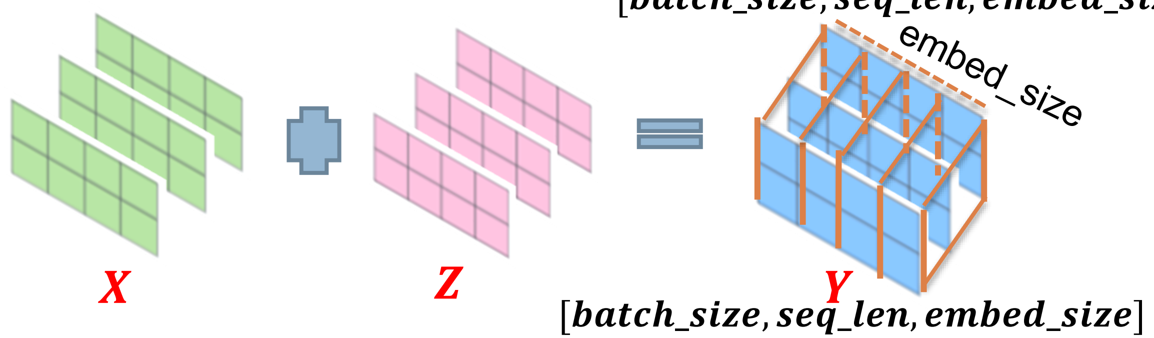
[Source: https://jalammar.github.io/illustrated-transformer/]

Transformer – Residual Connection/Layer Norm





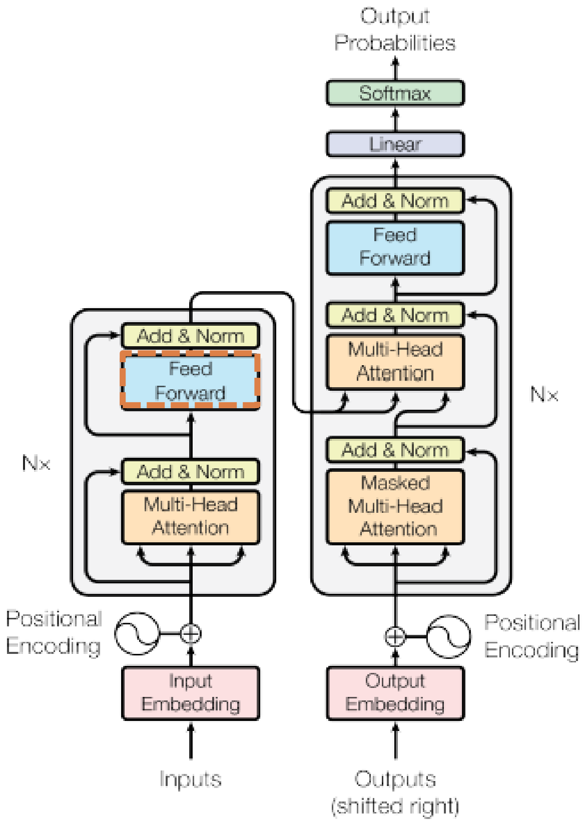


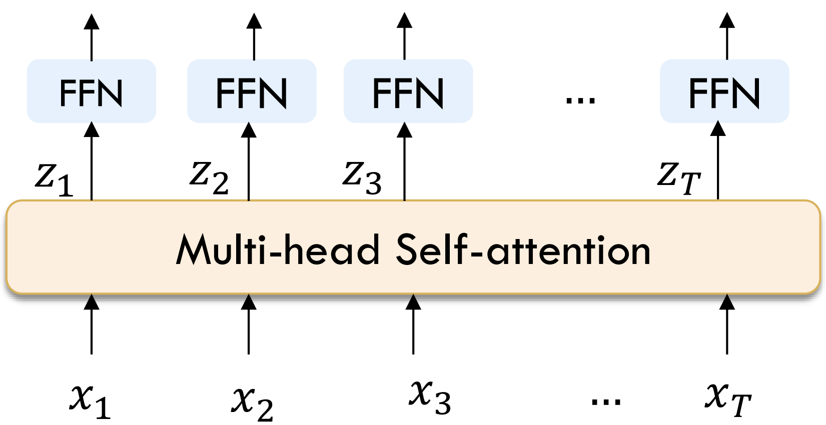


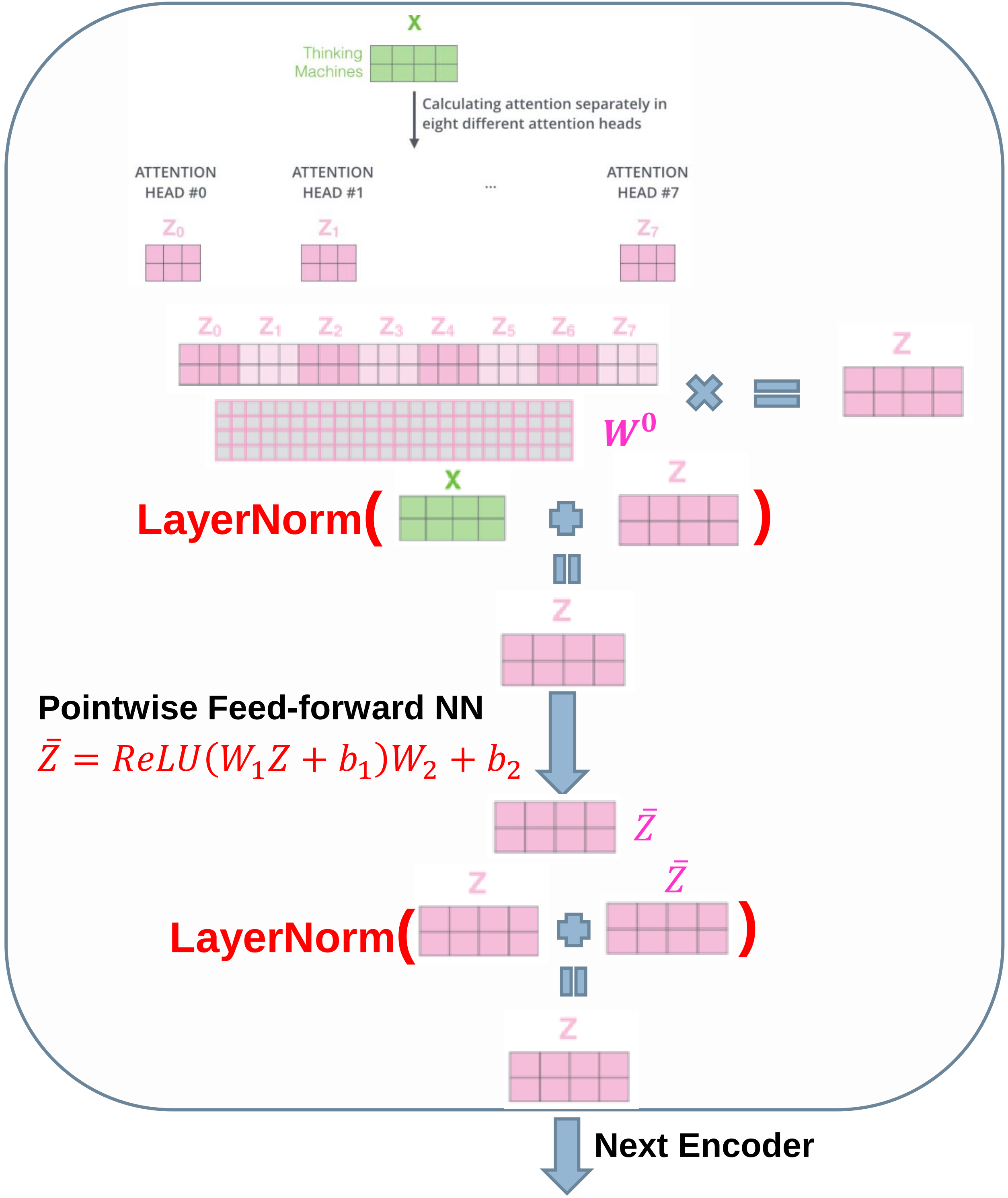
$$Y_m = mean(Y, axis = embed\_size, keep\_dims = True)$$
$$Y_\sigma = std(Y, axis = embed\_size, keep\_dims = True)$$
$$LayerNorm(Y) = \gamma \frac{Y - Y_m}{Y_\sigma} + \beta$$

- **Layer normalization** (LN) is the same as **batch normalization** except that LN **normalizes** across the **feature dimension**.
  - **Batch normalization** is **usually empirically less effective** than **layer normalization** in natural language processing tasks, whose inputs are often variable-length sequences.

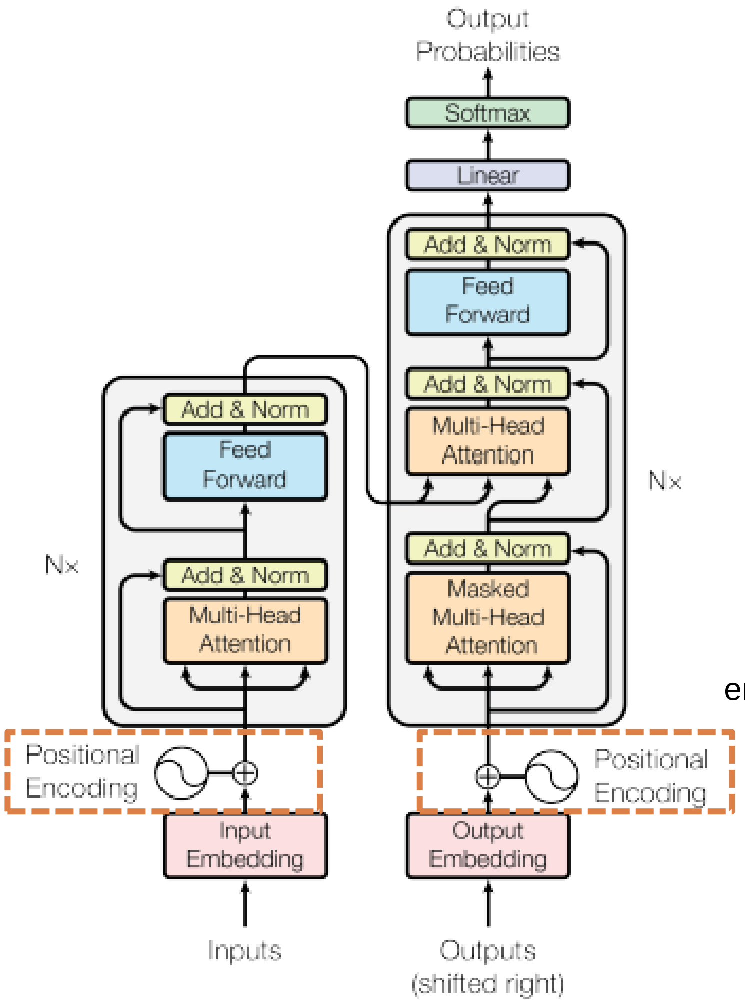
Transformer- Pointwise Feed-forward NN





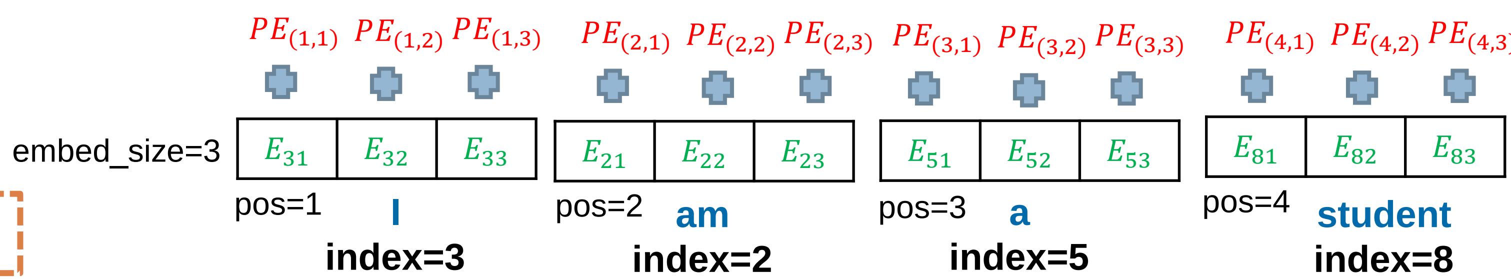


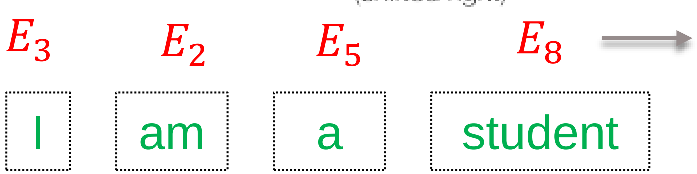
Transformer – Positional Encoding



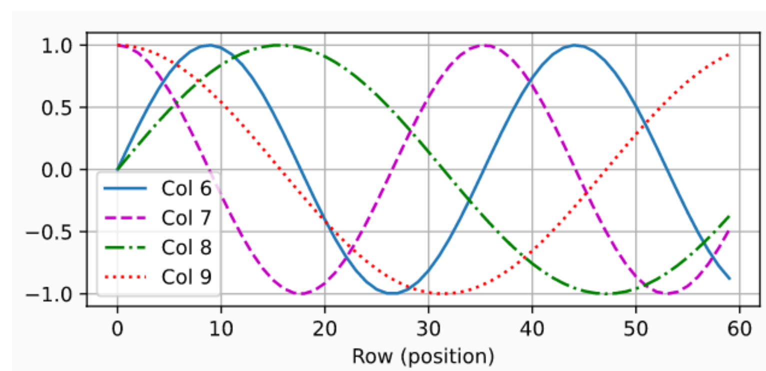
- Use to capture **position** of a word in a sentence
  - Transformer has **no recurrence** and **no convolution**.
  - For the model to make use of the **order of the sequence**, some information about the **relative or absolute position** of the tokens in the sequence **needs to be injected**

$$PE_{(pos,2i)} = \sin(pos/1000^{2i/embed\_size})$$
$$PE_{(pos,2i+1)} = \cos(pos/1000^{2i/embed\_size})$$





No information of word order

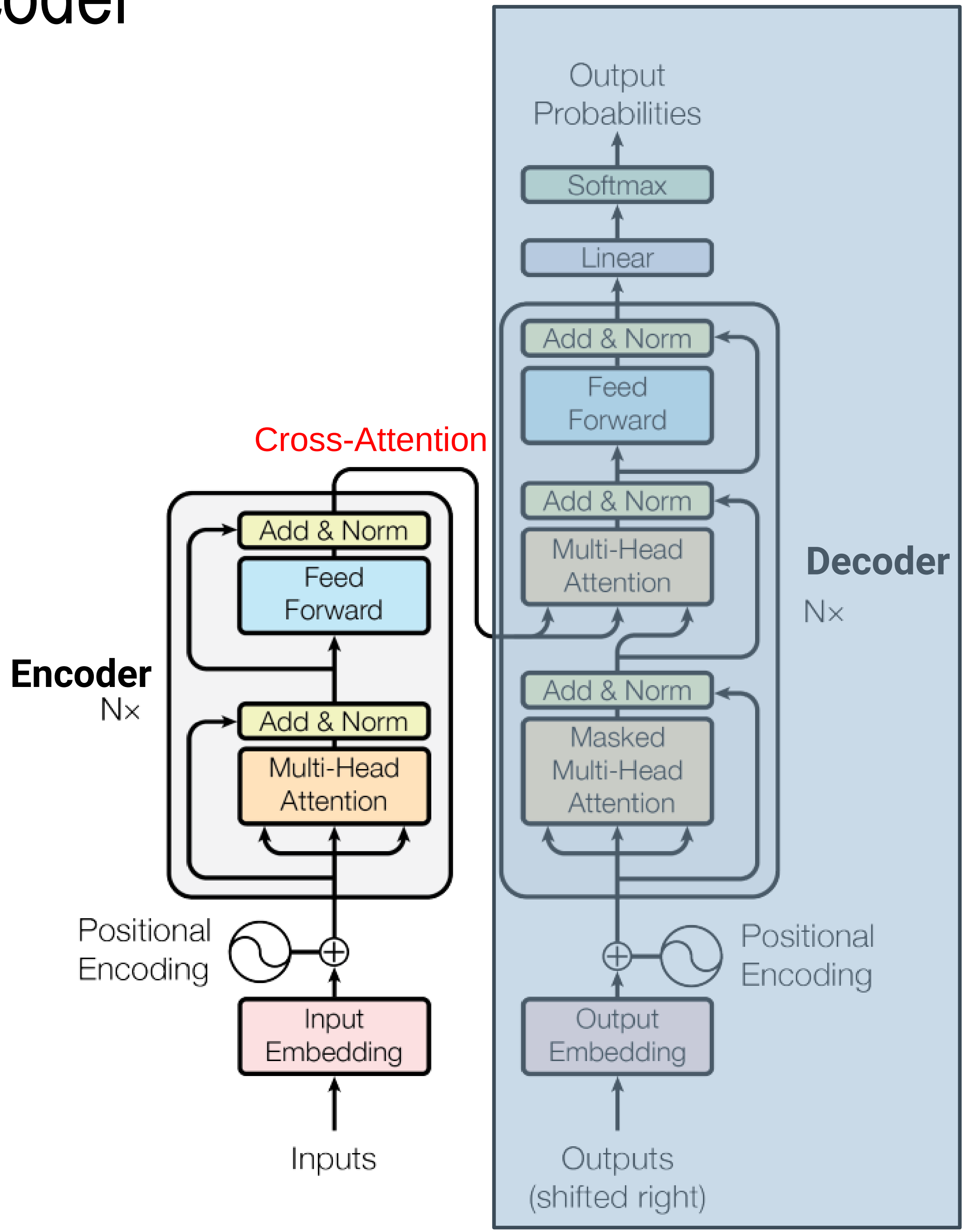


All words or **word embeddings** are **inputted simultaneously** to the Transformer

[Source: Dive into DL]

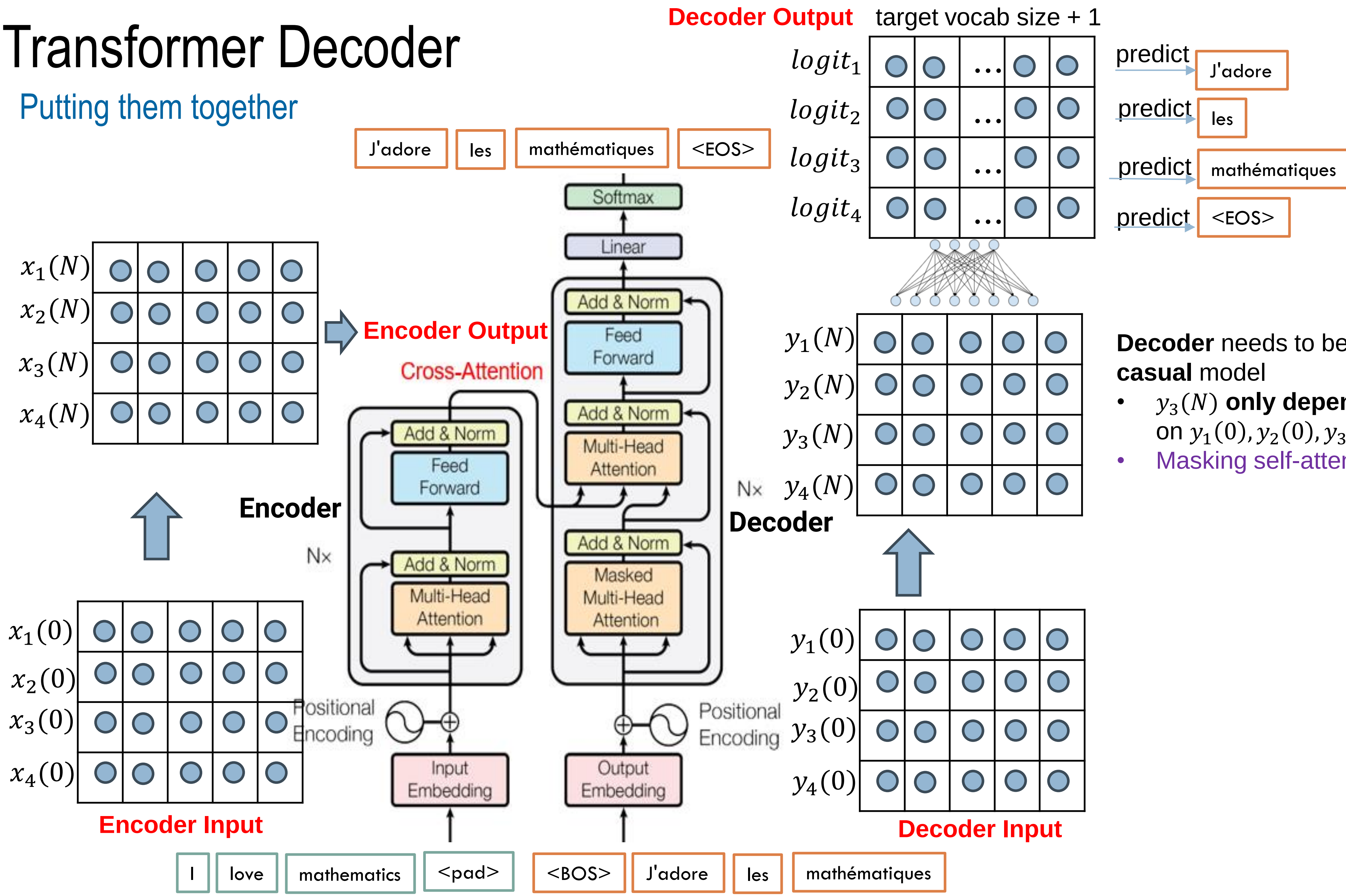
Transformer Decoder

Putting them together



Transformer Decoder

Putting them together

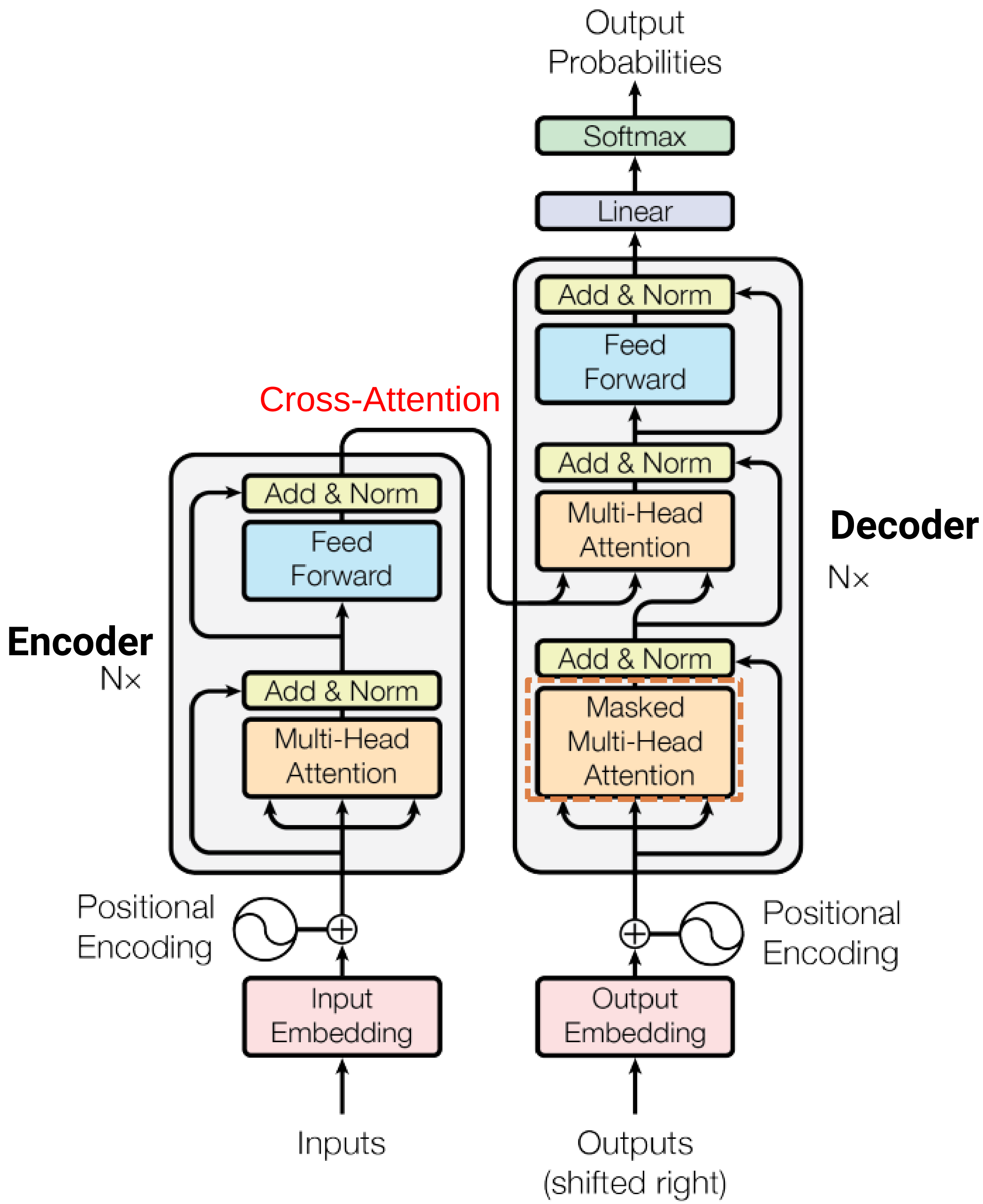


Decoder needs to be a **casual** model

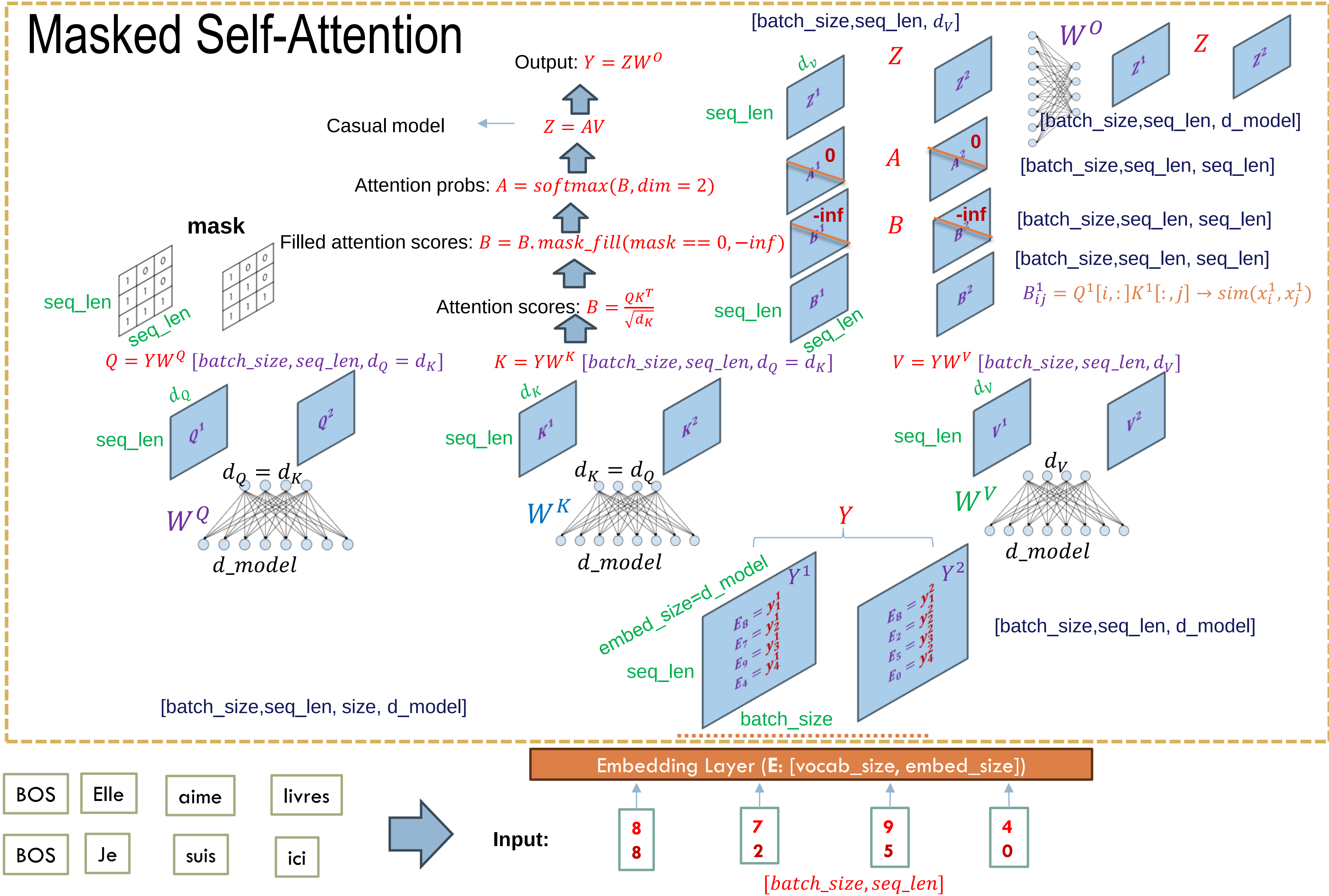
- $y_3(N)$ **only depends** on $y_1(0), y_2(0), y_3(0)$
- Masking self-attention

Transformer Decoder

Putting them together

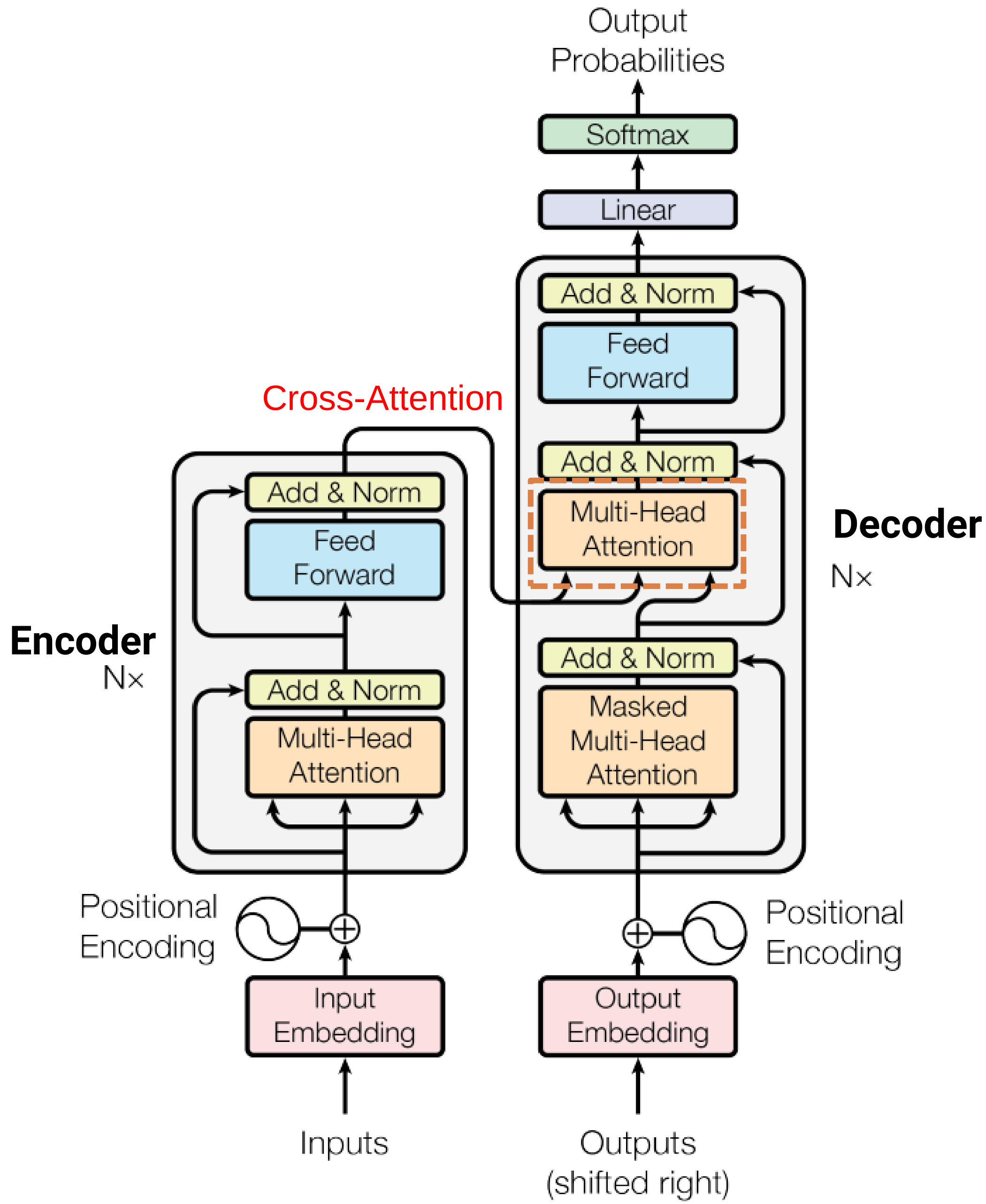


Masked Self-Attention

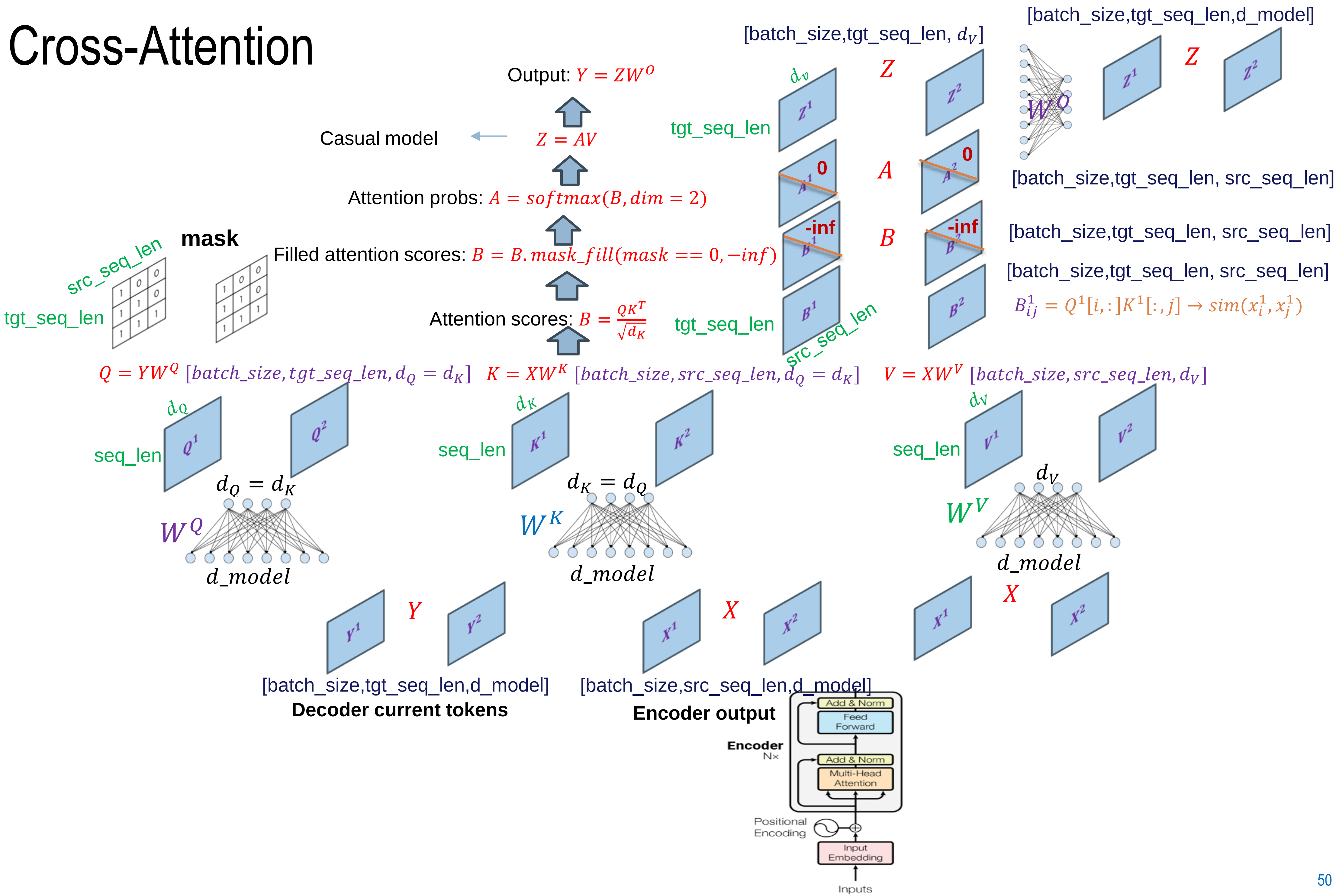


Transformer Decoder

Putting them together

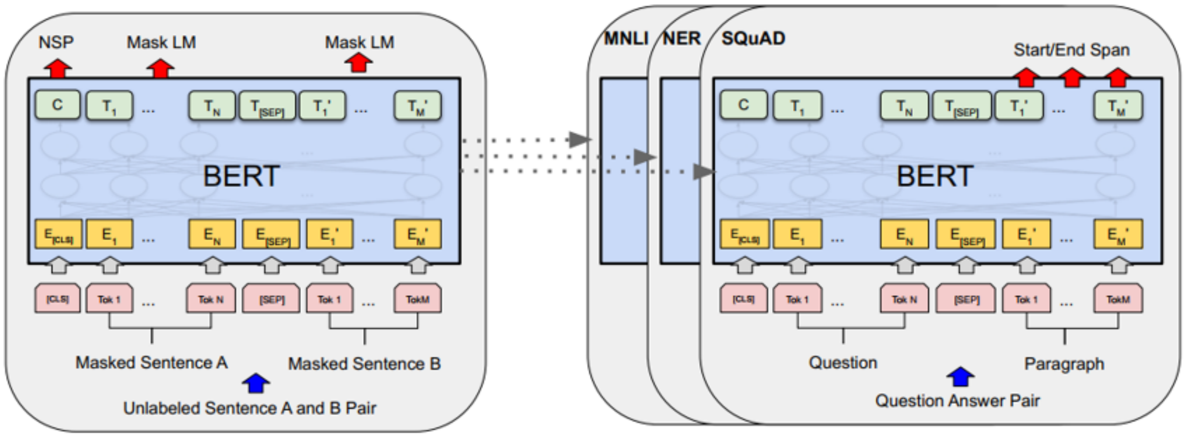


Cross-Attention



Pretrained Language Models

Bidirectional Encoder Representations from Transformers (BERT)



Pre-training

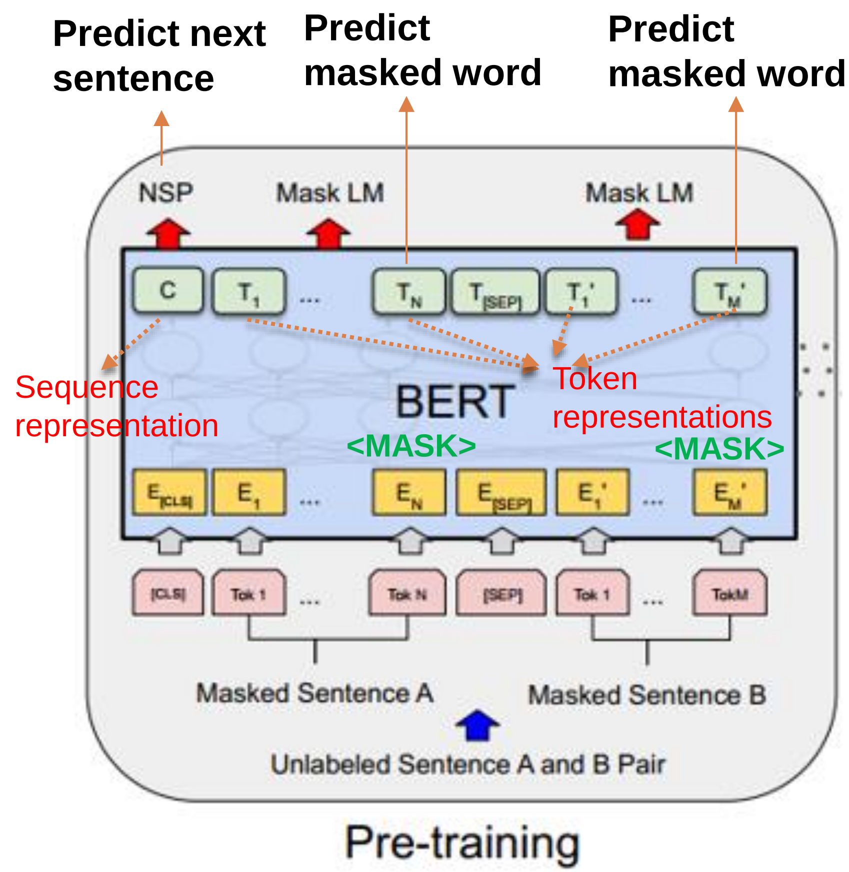
Fine-Tuning

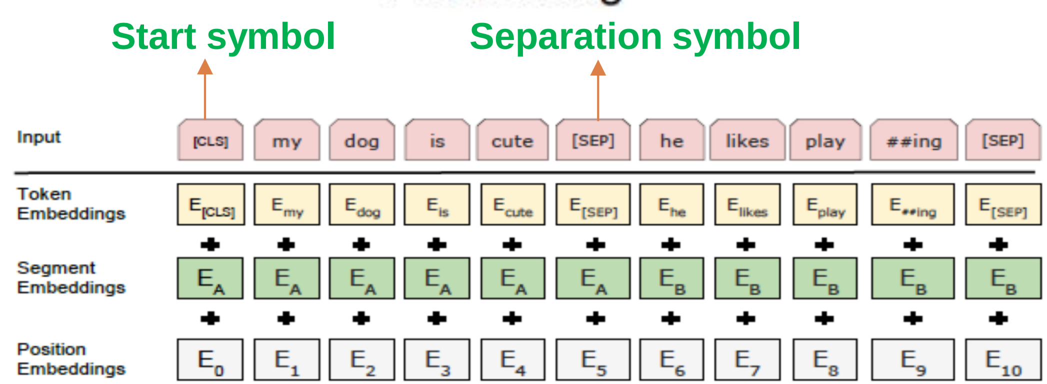
- Developed based on the architecture of **Transformer**
- **Two steps** in the framework
  - **Pretrain** and **fine tuning**
- **Pretrain**
  - The model is trained on unlabeled data over different pre-training tasks
  - **Masked word prediction** and **next sentence prediction**
- **Fine tuning**
  - The **model for a downstream task** is first initialized with the pre-trained parameters, and all of the parameters are fine-tuned using labeled data from the downstream tasks.
  - Some downstream tasks, for example, **sequence-level classification**, **predicting the answer text span in the passage**, and **sentence-pair completion**

BERT - Pretrain

- **Task 1**: *Masked words prediction*
  - **15%** of the words are **masked** at **random**
  - **Not all tokens were masked** in the same way. Given a masked word, it happens
    - With an **80%** of chance, this word is replaced by the **<MASK>** token
    - With a **10%** of chance, this word is replaced by a **random** word
    - With a **10%** of change, this word is **left intact**
  - Predict the **indices of masked words** on top of representations of those words.

- **Task 2**: *Next sentence prediction*
  - When choosing the sentences A and B for each pretraining example, **50% of the time** B is the **actual next sentence** following A (labeled as **IsNext**), and **50% of the time** it is a **random sentence** from the corpus (labeled as **NotNext**).
  - The **final hidden vector** of the special **[CLS]** token as $C \in \mathbb{R}^H$ is used to predict two labels: **IsNext** and **NotNext.**

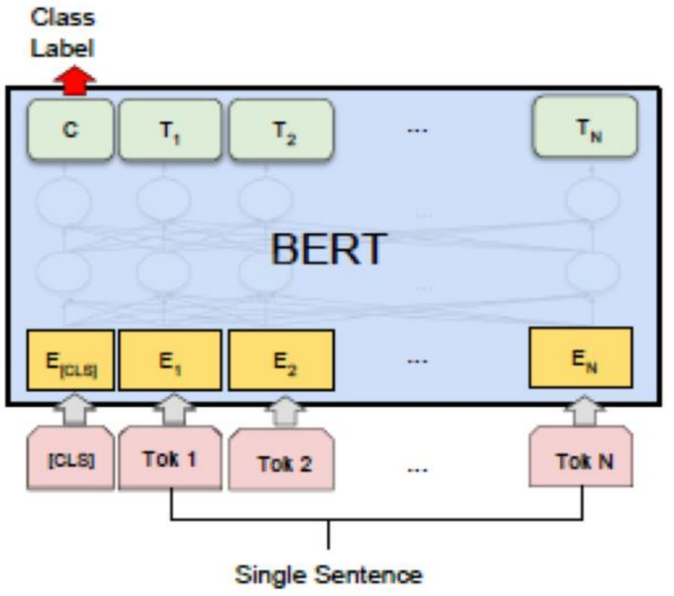




BERT – Fine Tuning

Sequence level classification

- Input a **single sentence** to pretrain BERT.
- Observe the representations $C, T_1, \ldots, T_N \in \mathbb{R}^h$
  - $C \in \mathbb{R}^h$ **aggregates** the information of tokens/words $1,2,\ldots,N$, hence can be viewed as a **sentence representation**.
- On top of the **sentence representation** C, we predict sentence label
  - $W \in \mathbb{R}^{M\times h}$ is an **additional weight matrix** where M is the number of classes.
  - Prediction probabilities $P = softmax(WC)$
- Fine tuning $\boldsymbol{W}$ and **pretrain BERT parameters**.



BERT – Fine Tuning

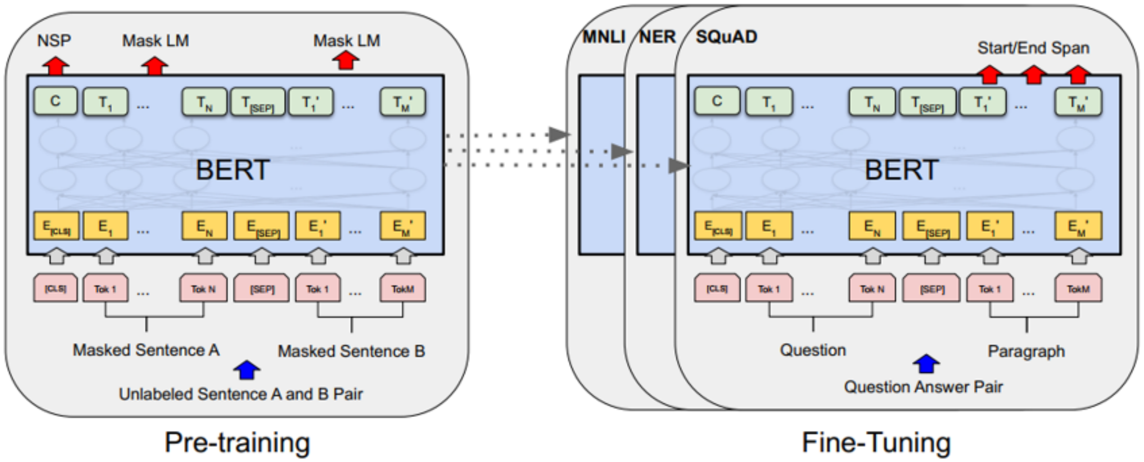
Predicting the answer text span

- **Task description**
  - Input Question:

    Where do water droplets collide with ice crystals to form precipitation?
  - Input Paragraph:

    .... Precipitation forms as smaller droplets coalesce via collision with other raindrops or ice crystals within a cloud. ...
  - Output Answer:

    within a cloud



- Represent the **input question** and **input paragraph** as a **single packed sequence**
- The **question** uses the **A embedding** and the **paragraph** uses the **B embedding**
- **New parameters** to be learned in fine-tuning are **start vector** $S \in \mathbb{R}^h$ and **end vector** $E \in \mathbb{R}^h$
- The probability of **word *i*** being **the start** of the answer span
  - $P_i = \frac{\exp\{S^T T_i\}}{\sum_{j \in input\ paragraph} \exp\{S^T T_j\}}$
- The probability **word *i*** being **the start** and **word *j*** being **the end** of the answer span
  - $P_{i,j} = \frac{\exp\{S^T T_i + E^T T_j\}}{\sum_{(i',j'):j'>i'} \exp\{S^T T_{i'} + E^T T_{j'}\}}$
- Train $\boldsymbol{S}, \boldsymbol{E}$ and **pretrain BERT parameters** by maximizing log-likelihood.

Why does pretraining-then-finetuning work?

Model	Model Size (Parameters)	Training Data (# Tokens)	Notes
GPT-3 (175B)	175 billion	300 billion tokens	Trained on a diverse dataset including Common Crawl, Wikipedia, and various books
GPT-4	Unknown (estimated 1 trillion)	Trained on hundreds of billions of tokens	Exact size not disclosed, improved multimodal capabilities, and larger context window
BERT (Base)	110 million	3.3 billion tokens	Pretrained on BooksCorpus and English Wikipedia
BERT (Large)	340 million	3.3 billion tokens	Same data as BERT Base, but with more layers and parameters
T5 (Base)	220 million	1 trillion tokens	Trained on the Colossal Clean Crawled Corpus (C4)
T5 (Large)	770 million	1 trillion tokens	Same dataset as T5 Base, scaled up
T5 (XXL)	11 billion	1 trillion tokens	Same dataset as T5 Base, scaled to a massive number of parameters
PaLM	540 billion	780 billion tokens	Trained on a multilingual dataset, including data from the web, books, Wikipedia
LLaMA (7B)	7 billion	1.0 trillion tokens	Trained on publicly available datasets and academic sources
LLaMA 2 (13B)	13 billion	2.0 trillion tokens	Enhanced version of LLaMA with more training tokens and improved training architecture
LLaMA 2 (70B)	70 billion	2.0 trillion tokens	Largest version of LLaMA 2 with significant improvements in token efficiency
Chinchilla	70 billion	1.4 trillion tokens	Developed by DeepMind, trained on more tokens with optimal compute scaling laws
Gopher	280 billion	300 billion tokens	Trained on diverse sources such as books, articles, and websites
BLOOM (176B)	176 billion	366 billion tokens	Trained on multilingual data, including a variety of web sources and datasets
Claude 2	70 billion	Unknown (hundreds of billions estimated)	Built by Anthropic, focused on safety and interpretability
Mistral (7B)	7 billion	2.0 trillion tokens	A dense model trained for efficiency and performance in NLP tasks

1. Transformer architecture
2. Large scale training data
3. Large number of parameters

Further Reading Recommendation

- Seq2Seq
  - https://jalammar.github.io/
  - https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/
  - https://trungtran.io/2019/02/27/neural-machine-translation-with-tensorflow-model-creation/
- Transformer
  - https://ai.plainenglish.io/transformers-visual-guide-153af370693f
  - https://jalammar.github.io/illustrated-transformer/
- BERT
  - https://jalammar.github.io/illustrated-bert/
  - https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270
- ViT
  - https://arxiv.org/pdf/2010.11929.pdf
- Swin
  - https://arxiv.org/pdf/2103.14030.pdf
- Dive into Deep Learning: Chapter 10

Summary

- Encoder-Decoder models
- Sequence to sequence models
- Attention mechanism
  - Motivation
  - Global attention
  - Local attention
- Transformer and BERT

Thanks for your attention!

Question time