北京大学
PEKING UNIVERSITY

Seq2Seq, Attention, and Transformers

Kun Yuan

Center for Machine Learning Research @ Peking University

Contents

北京大学 PEKING UNIVERSITY

- Sequence-to-sequence models
- Attention
- Transformer

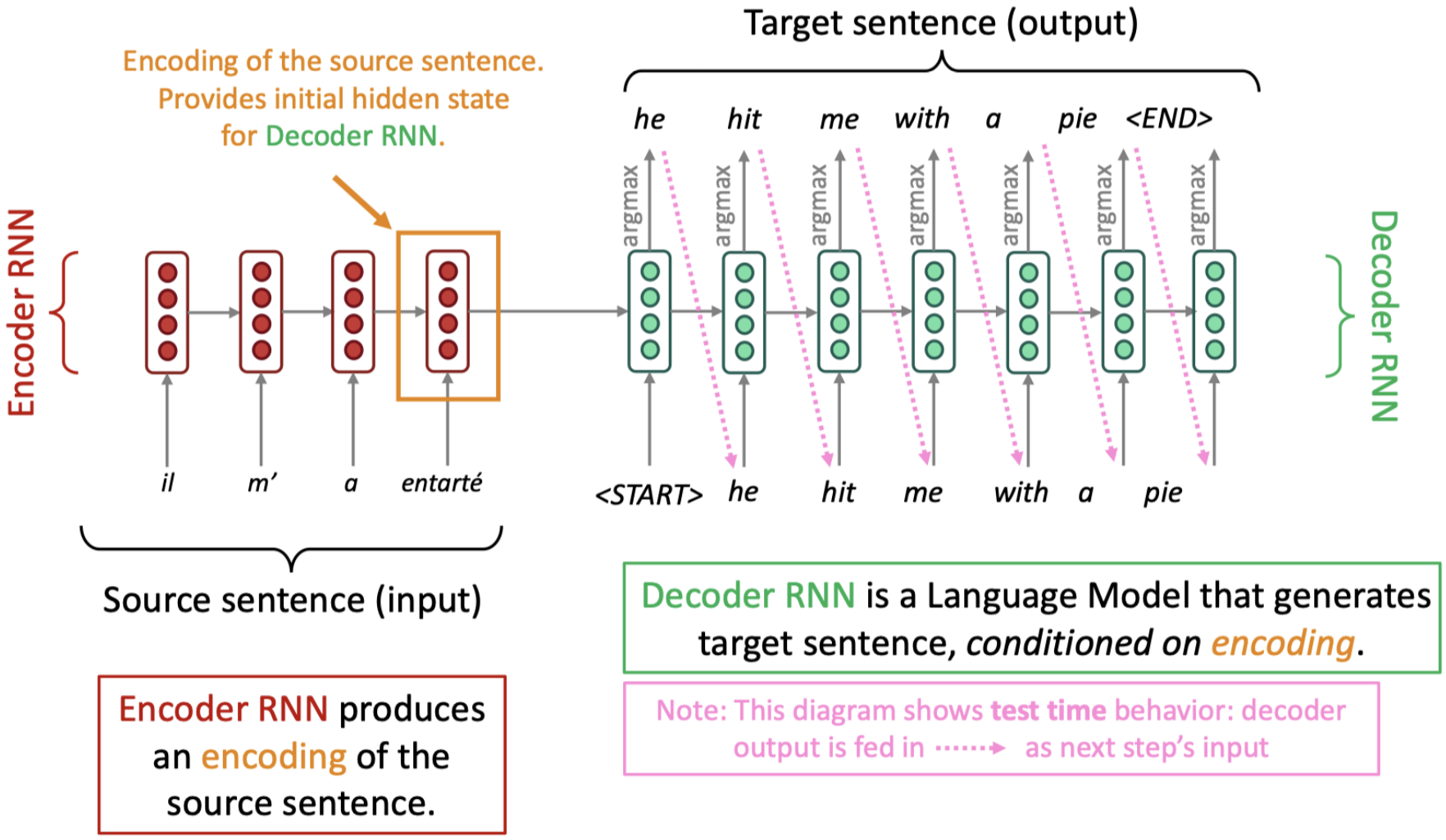
The materials are from a great course [1]

---

[1] Stanford CS224n: Natural Language Processing with Deep Learning

Neural machine translation

- Sequence-to-sequence model



Sequence-to-sequence is versatile!

北京大学 PEKING UNIVERSITY

- The general notion here is an encoder-decoder model
  - One neural network takes input and produces a neural representation
  - Another network produces output based on that neural representation
  - If the input and output are sequences, we call it a seq2seq model

- Sequence-to-sequence is useful for *more than just MT*
- Many NLP tasks can be phrased as sequence-to-sequence:
  - Summarization (long text → short text)
  - Dialogue (previous utterances → next utterance)
  - Parsing (input text → output parse as sequence)
  - Code generation (natural language → Python code)

Neural Machine Translation (NMT)

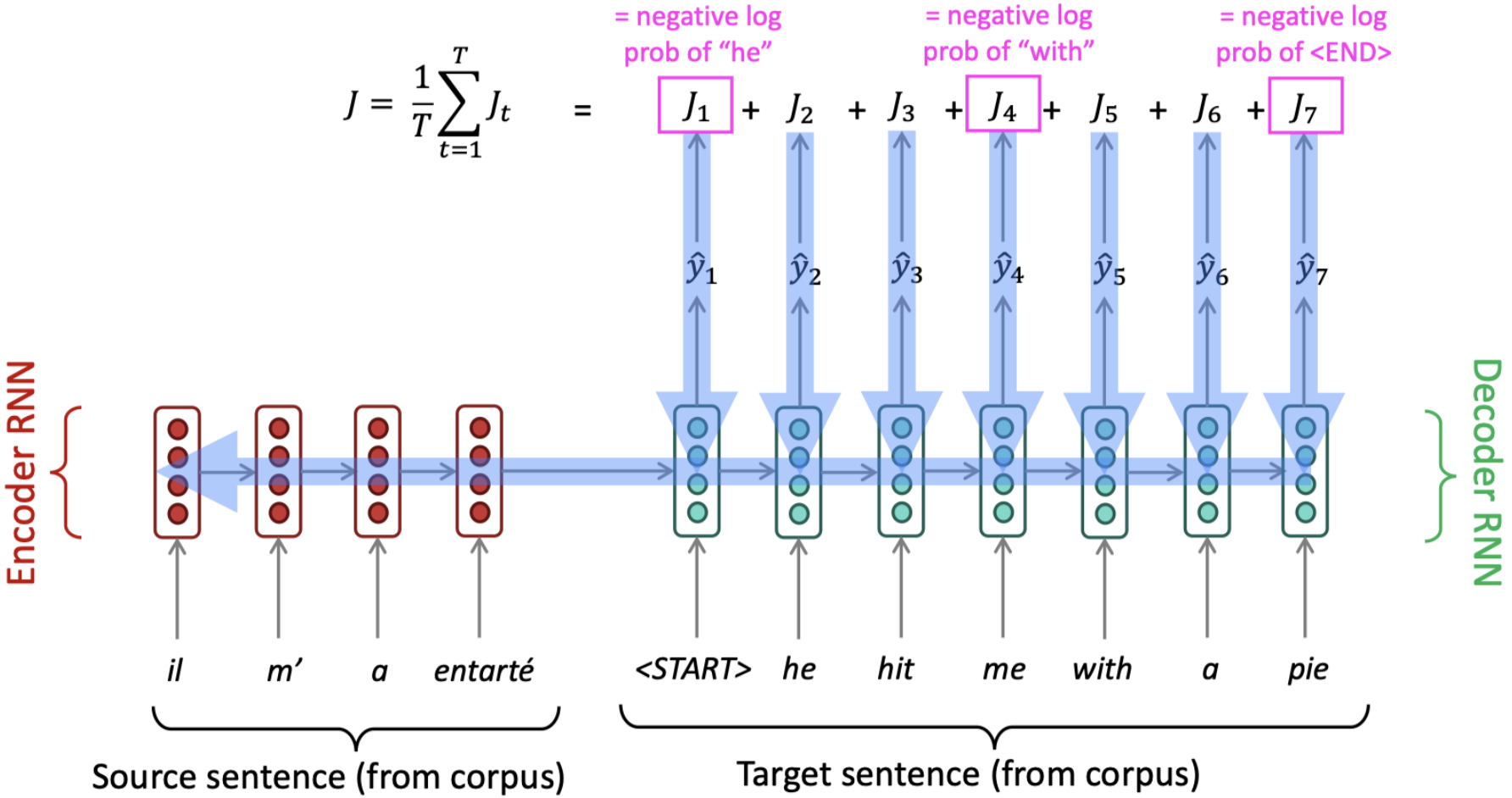
- The sequence-to-sequence model is an example of a **Conditional Language Model**
  - **Language Model** because the decoder is predicting the next word of the target sentence *y*
  - **Conditional** because its predictions are *also* conditioned on the source sentence *x*

- NMT directly calculates

$$\underbrace{P(y_T|y_1,\ldots,y_{T-1},x)}$$

Probability of next target word, given target words so far and source sentence *x*

Training a Neural Machine Translation system

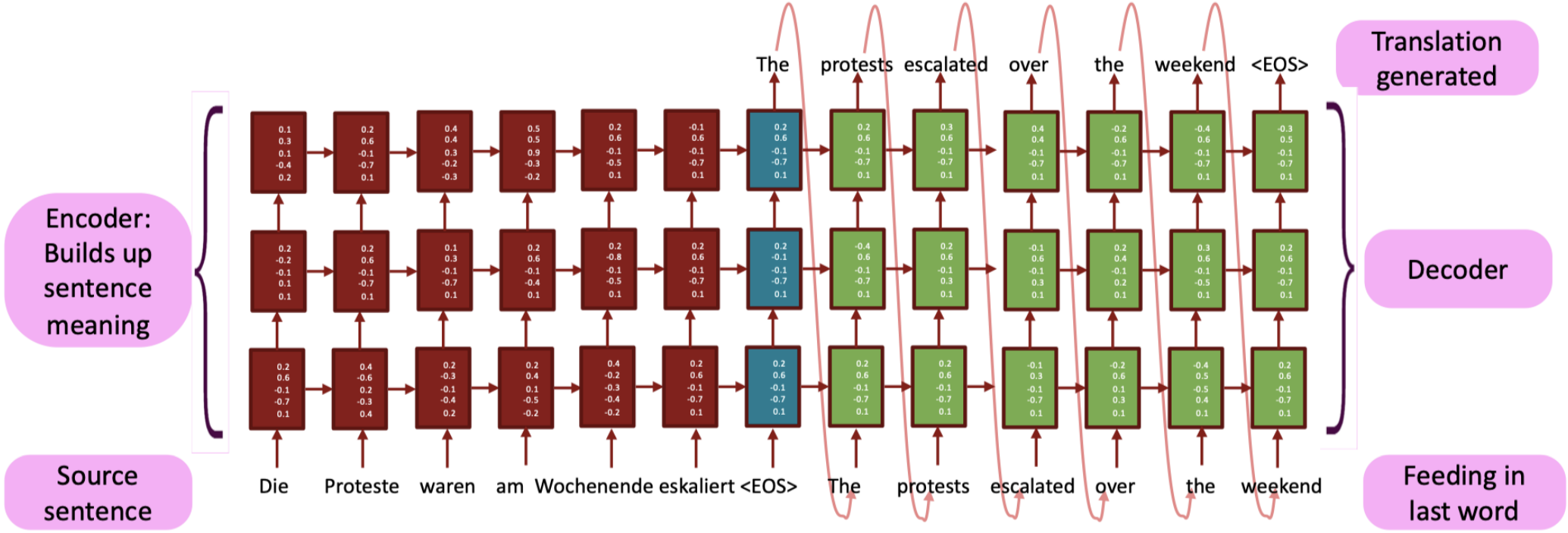


Seq2seq is optimized as a **single system.** Backpropagation operates "*end-to-end*".

Multi-layer deep encoder-decoder machine translation net

[Sutskever et al. 2014; Luong et al. 2015]

The hidden states from RNN layer *i* are the inputs to RNN layer *i*+1



NMT: the first big success story of NLP Deep Learning

北京大学 PEKING UNIVERSITY

Neural Machine Translation went from a fringe research attempt in **2014** to the leading standard method in **2016**

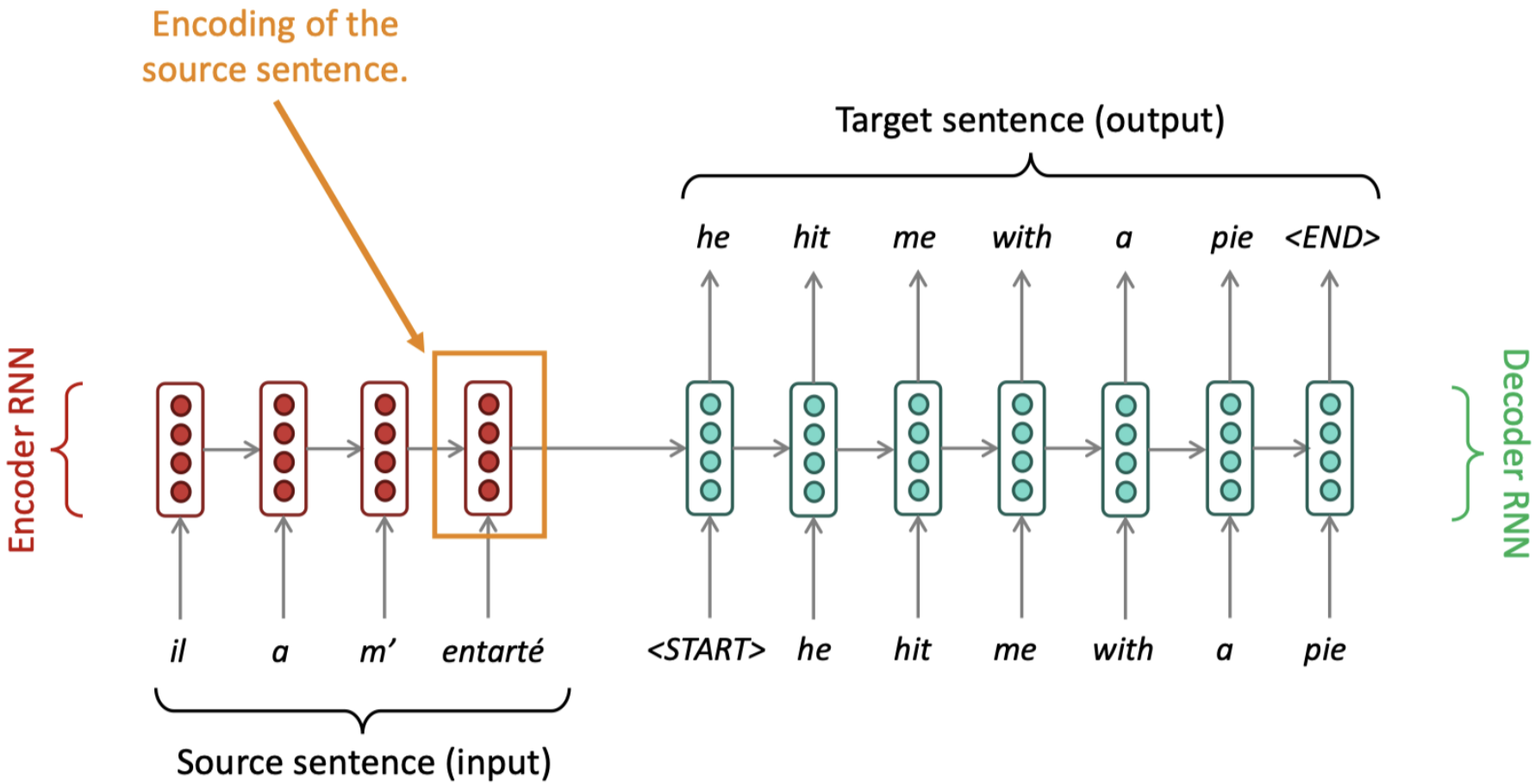
- **2014**: First seq2seq paper published [Sutskever et al. 2014]
- **2016**: Google Translate switches from SMT to NMT – and by 2018 everyone has

Microsoft SYSTRAN beyond language Google facebook

Baidu 百度 網易 NetEase www.163.com Tencent 腾讯 搜狗搜索

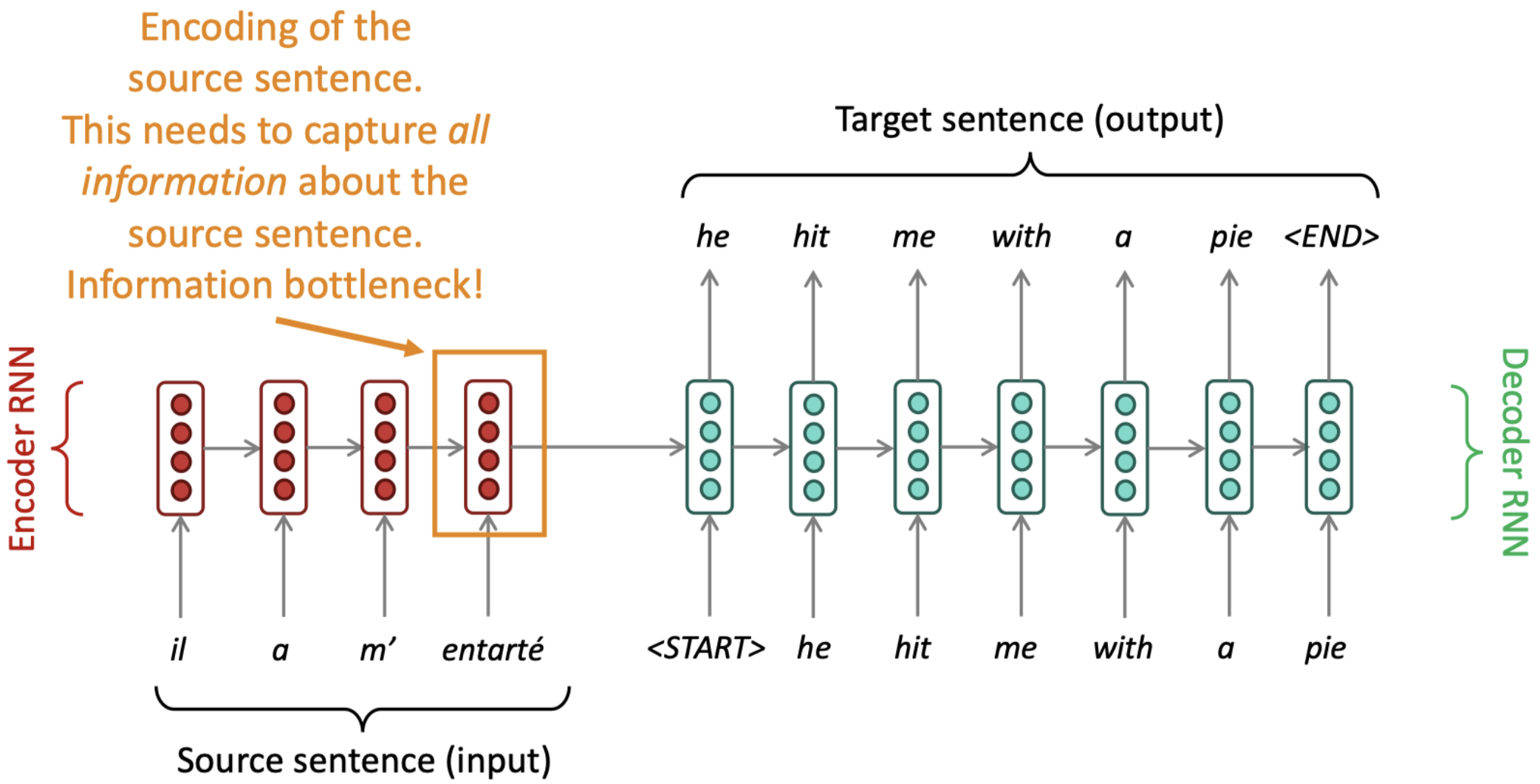
- This is amazing!
  - **SMT** systems, built by hundreds of engineers over many years, outperformed by NMT systems trained by small groups of engineers in a few months

The final piece: the bottleneck problem in RNNs



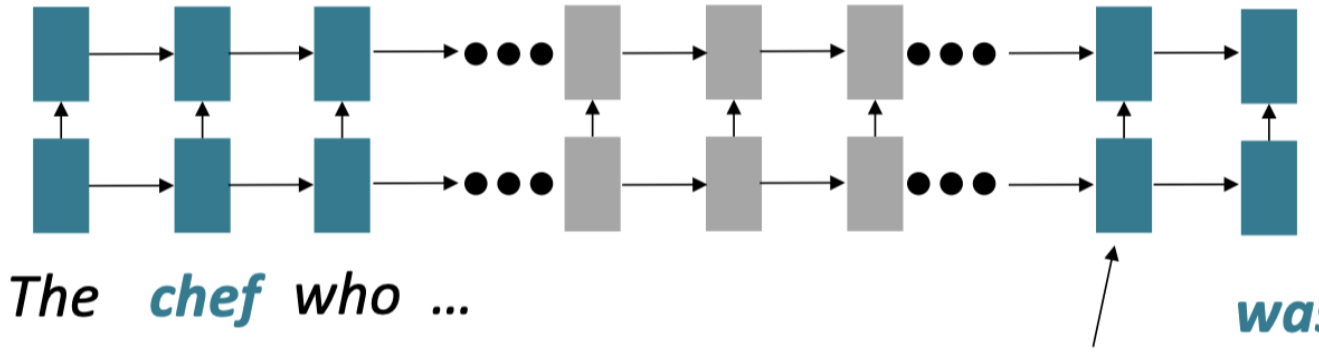
Problems with this architecture?

The final piece: the bottleneck problem in RNNs



Seq2Seq bottleneck I: Linear interaction distance

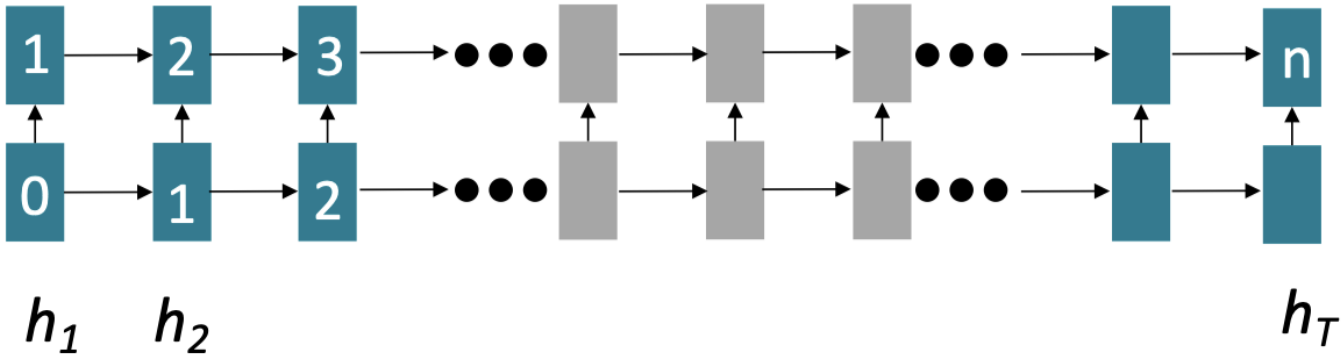
- **O(sequence length)** steps for distant word pairs to interact means:
  - Hard to learn long-distance dependencies (because gradient problems!)
  - Linear order of words is "baked in"; we already know linear order isn't the right way to think about sentences…



Info of ***chef*** has gone through O(sequence length) many layers!

Issues with recurrent models: Lack of parallelizability

- Forward and backward passes have **O(sequence length)** unparallelizable operations
  - GPUs can perform a bunch of independent computations at once!
  - But future RNN hidden states can't be computed in full before past RNN hidden states have been computed
  - Inhibits training on very large datasets!



Numbers indicate min # of steps before a state can be computed

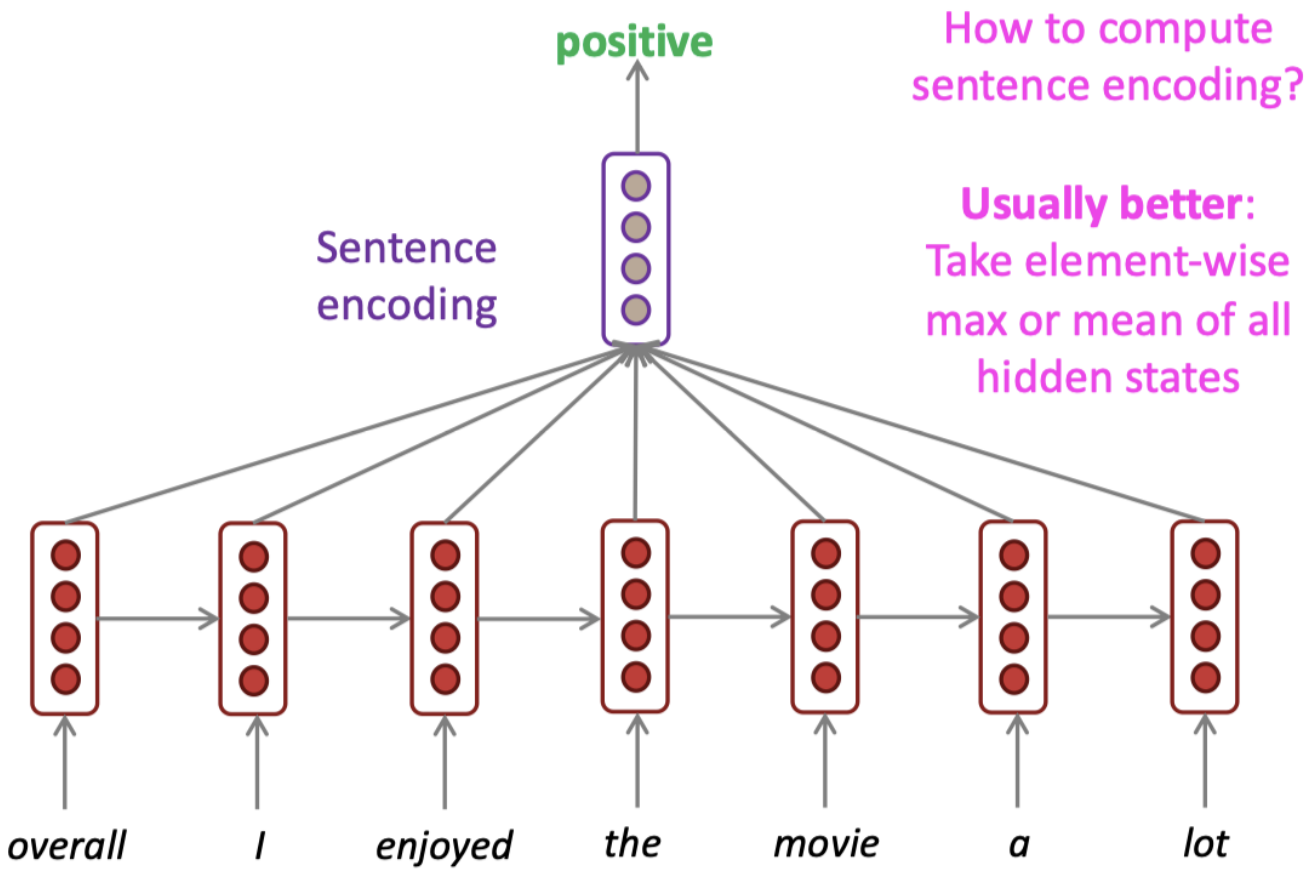
Attention

北京大学 PEKING UNIVERSITY

- **Attention** provides a solution to the bottleneck problem.

- **Core idea**: on each step of the decoder, use *direct connection to the encoder* to *focus on a particular part* of the source sequence

The starting point: mean-pooling for RNNs

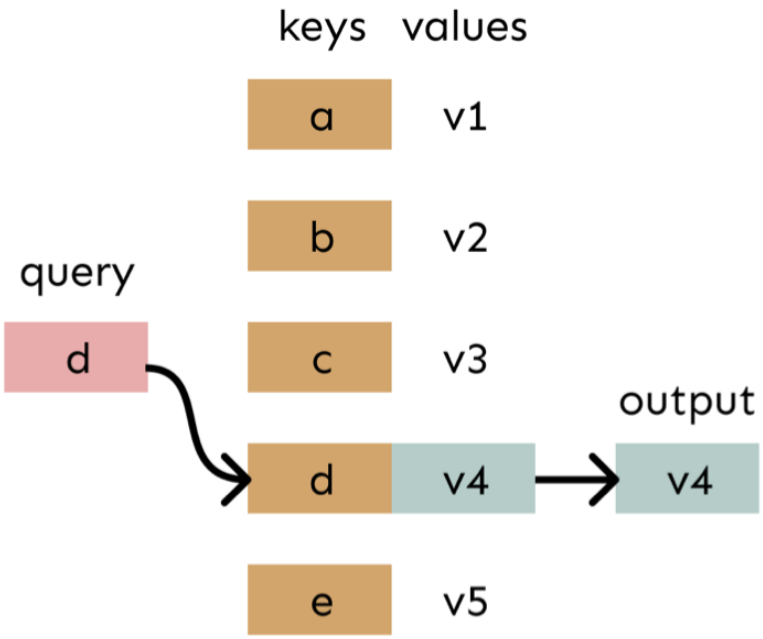


- Starting point: a *very* basic way of 'passing information from the encoder' is to *average*

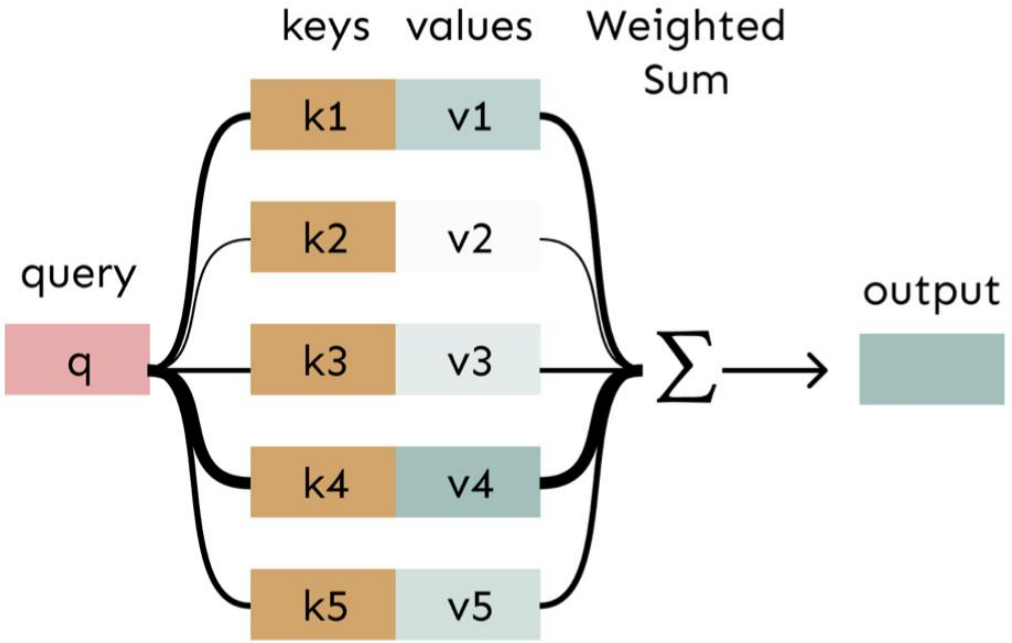
Attention is weighted averaging

Attention is just a **weighted** average – this is very powerful if the weights are learned!

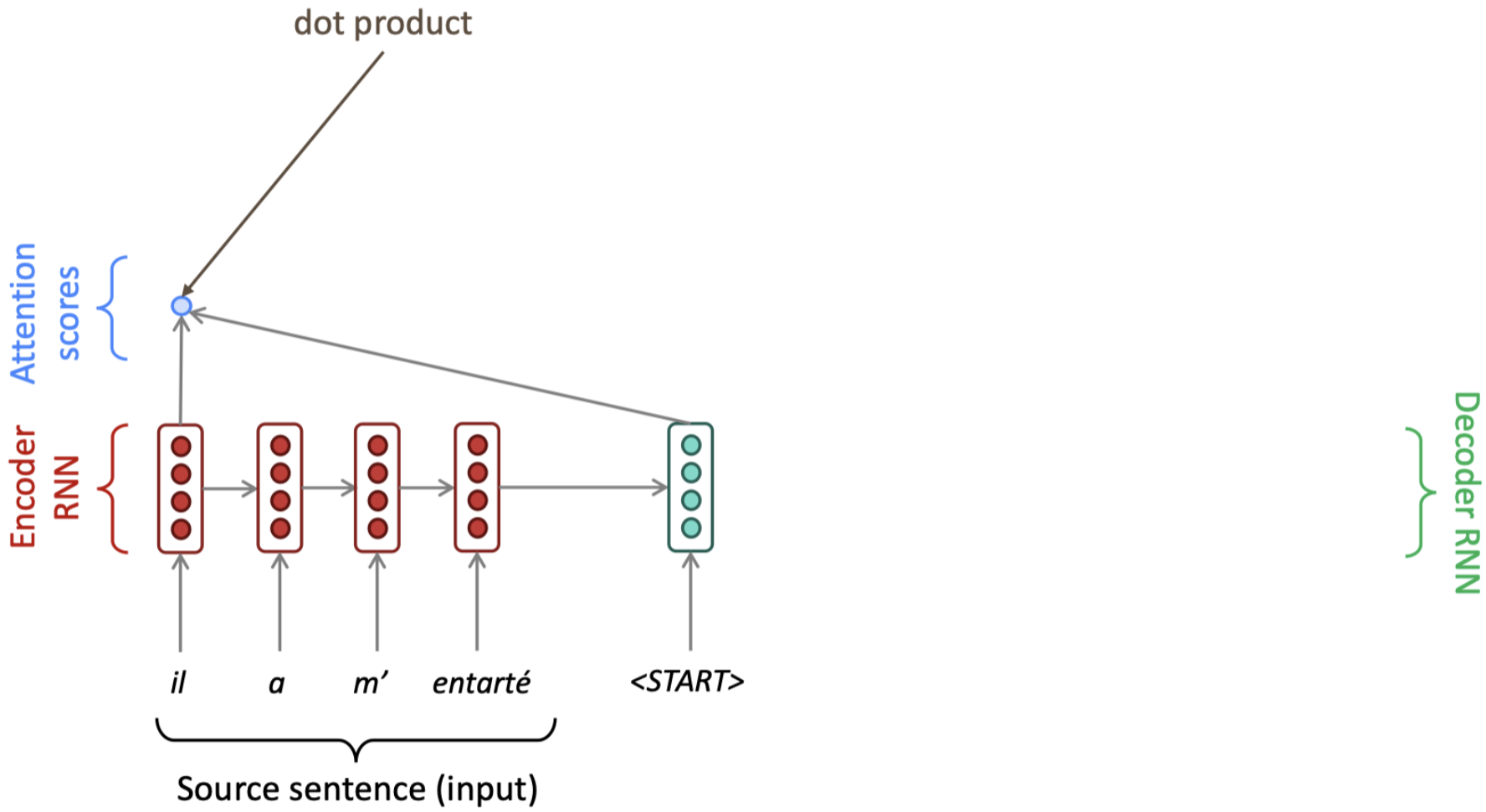
In a **lookup table**, we have a table of **keys** that map to **values**. The **query** matches one of the keys, returning its value.



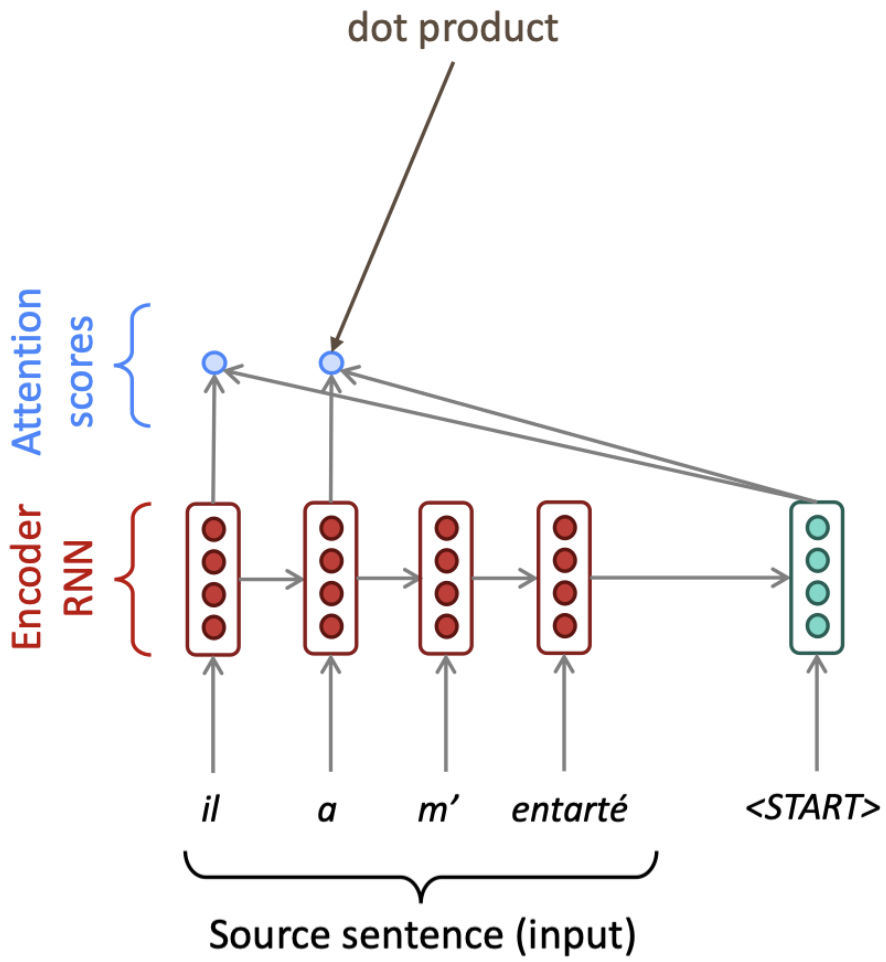
In **attention**, the **query** matches all **keys** *softly*, to a weight between 0 and 1. The keys' **values** are multiplied by the weights and summed.



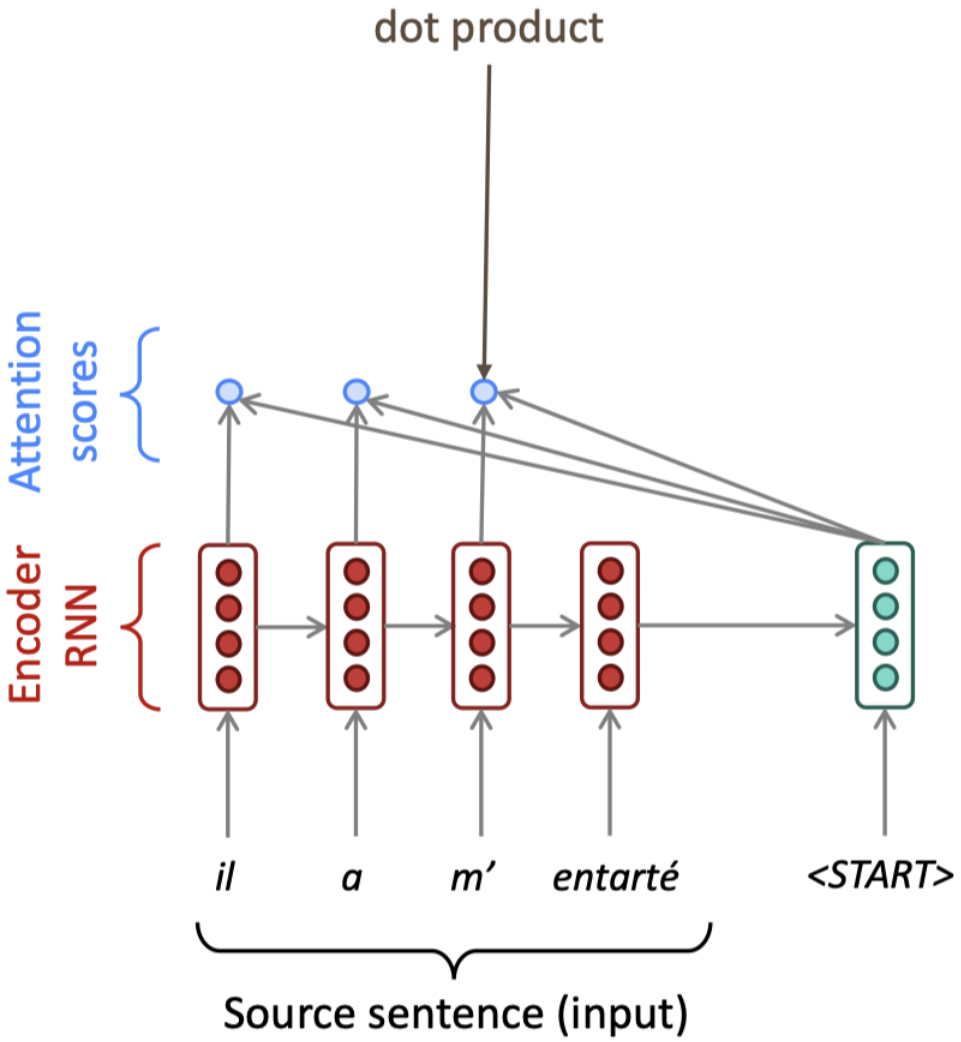
Sequence-to-sequence with attention



Sequence-to-sequence with attention

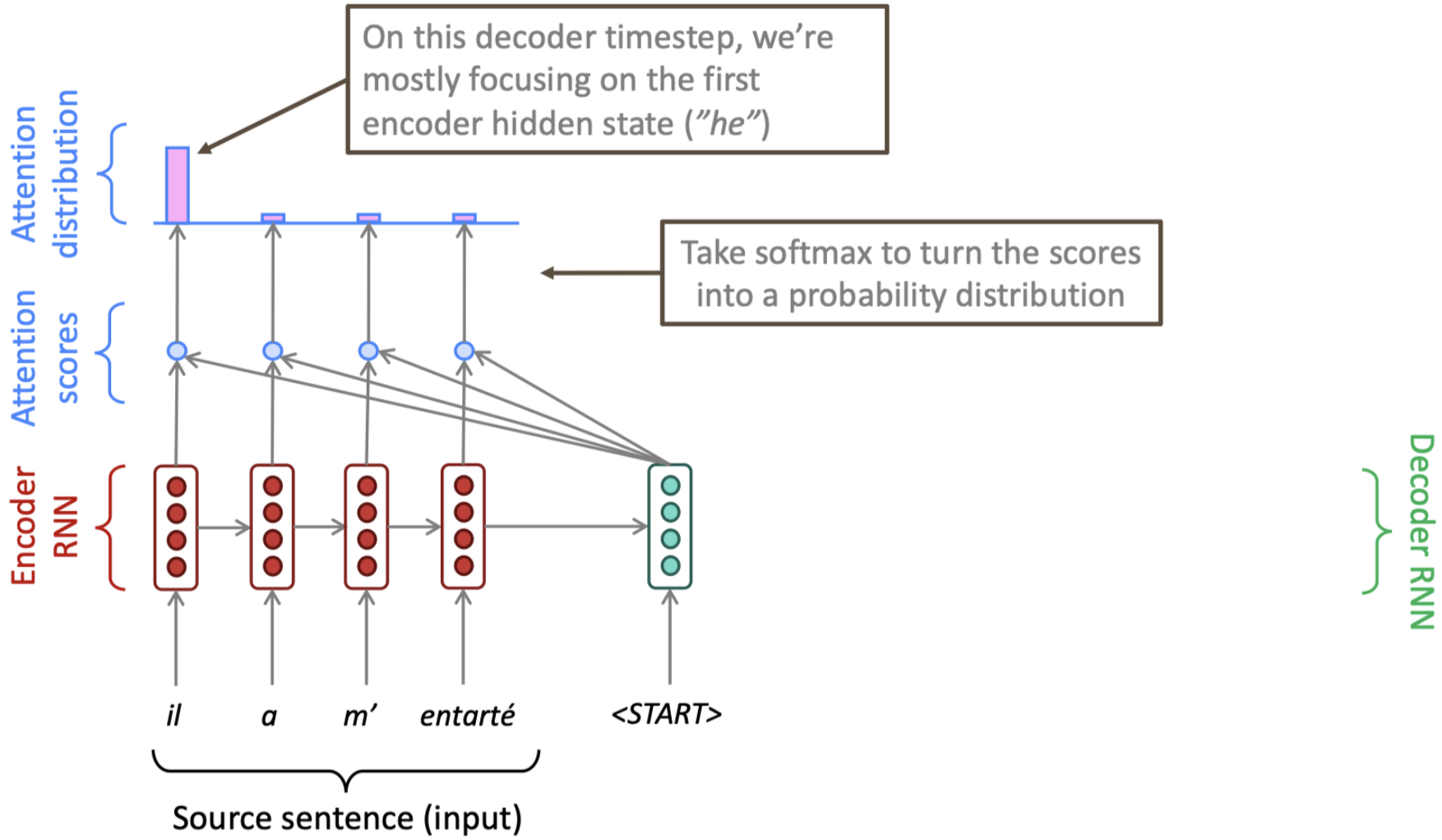


Sequence-to-sequence with attention

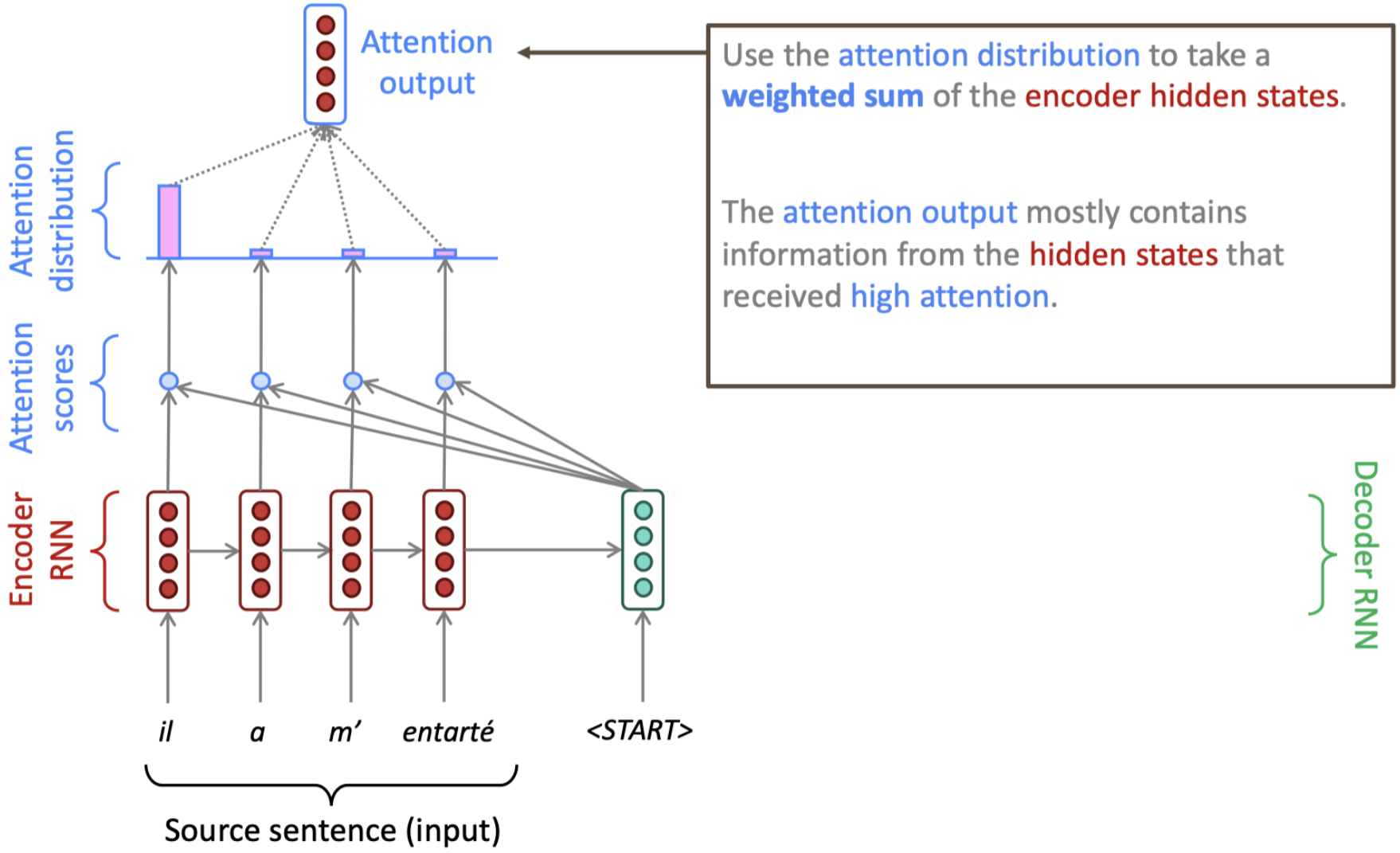


Decoder RNN

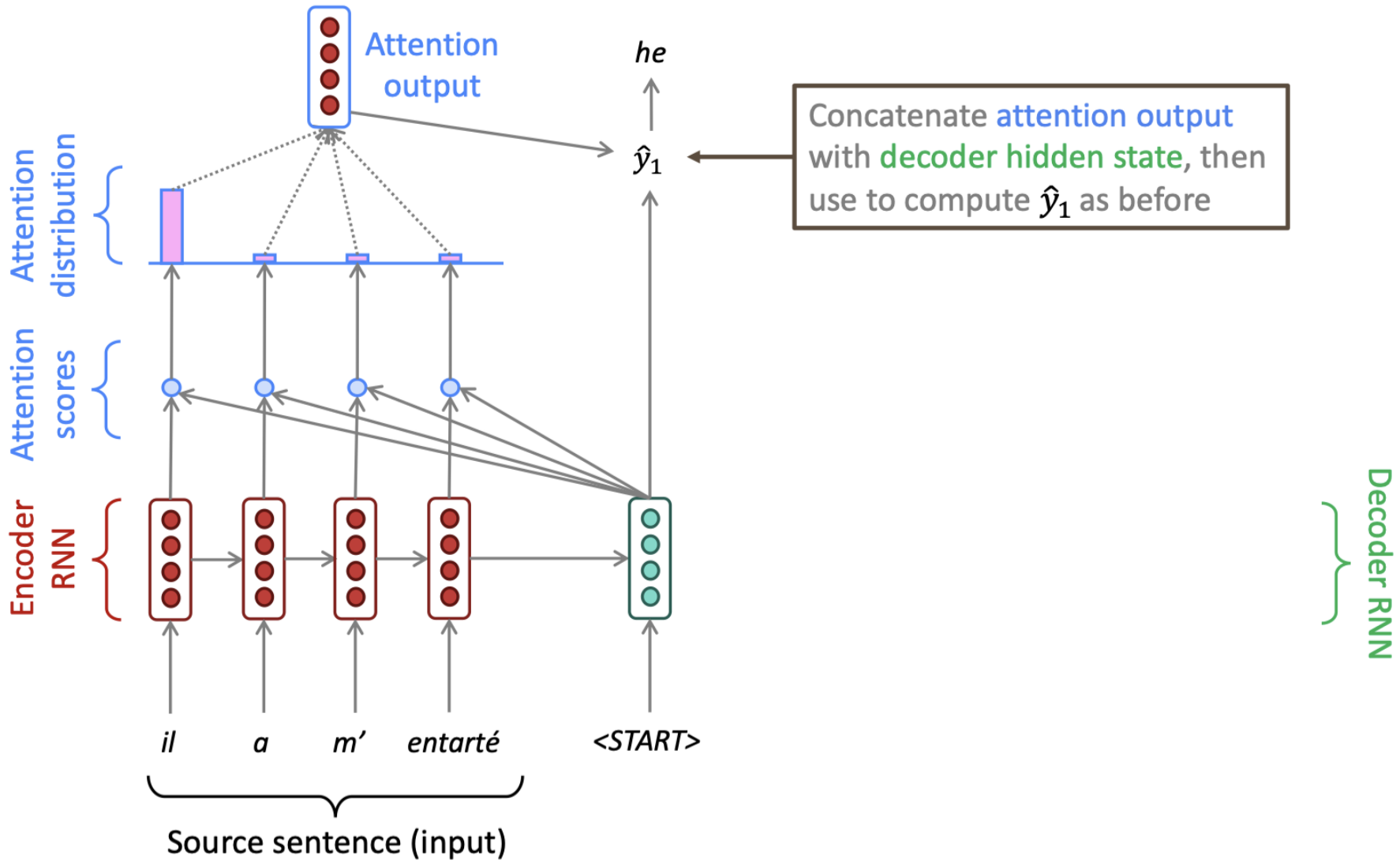
Sequence-to-sequence with attention



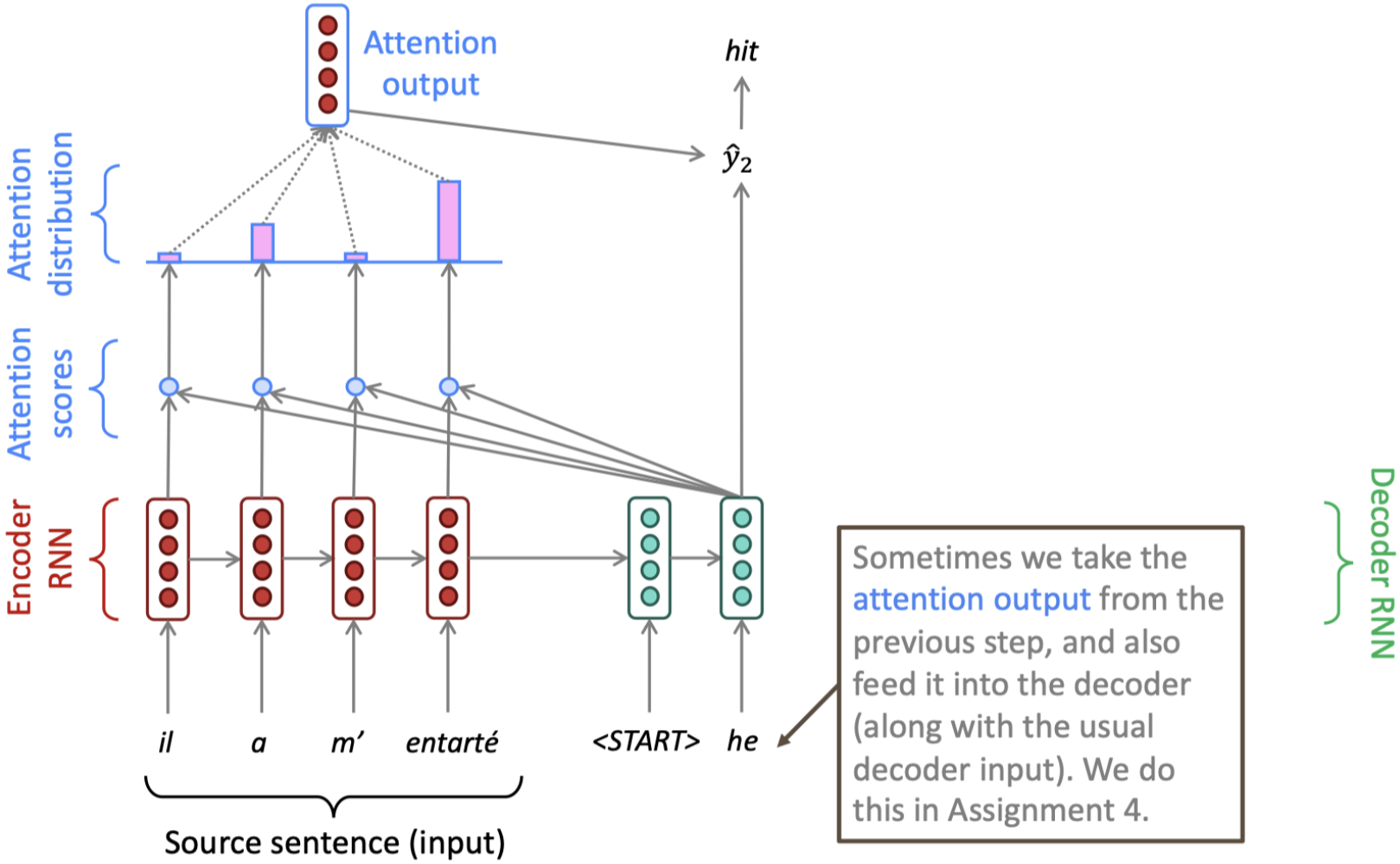
Sequence-to-sequence with attention



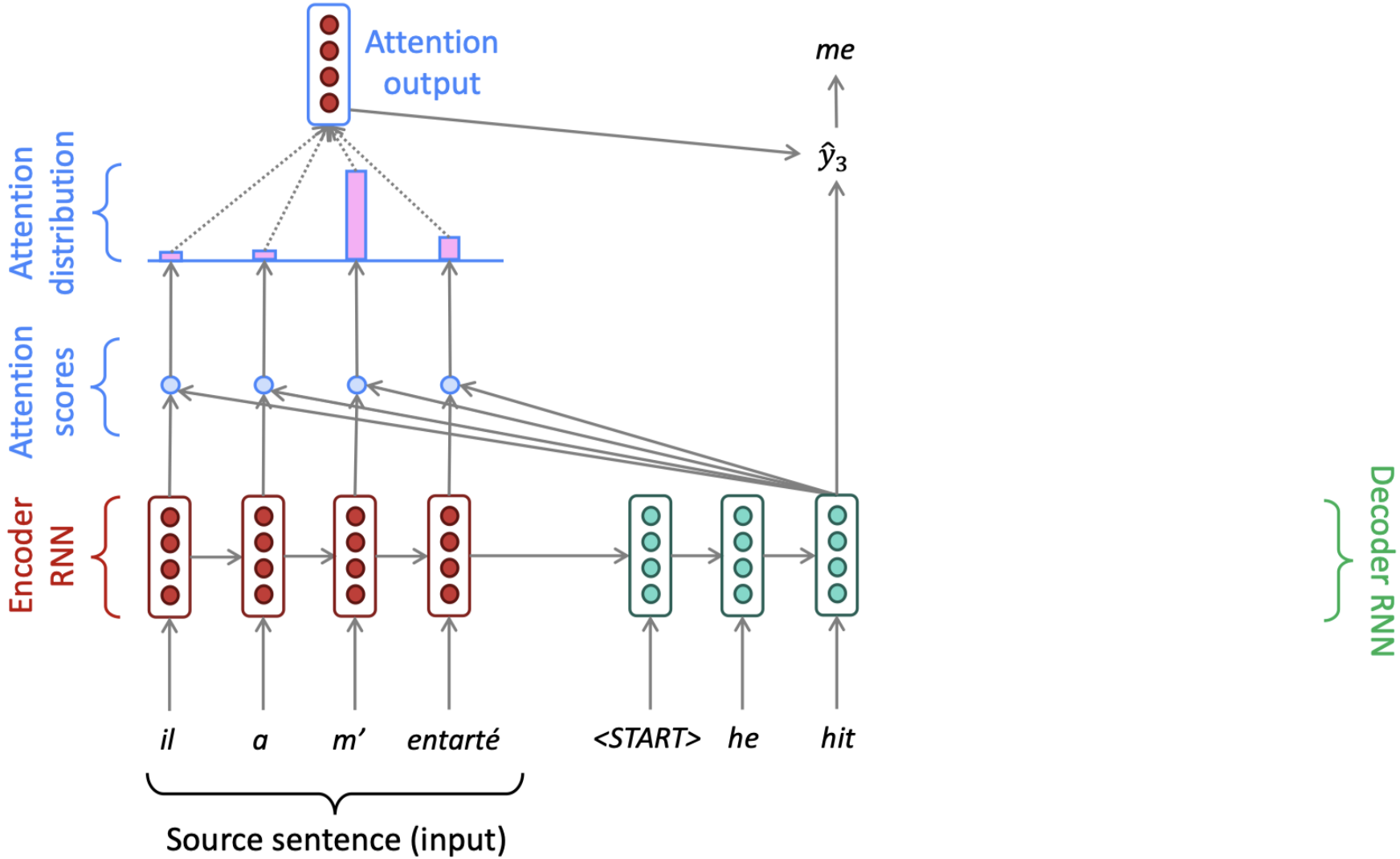
Sequence-to-sequence with attention



Sequence-to-sequence with attention



Sequence-to-sequence with attention



Attention equations

- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep *t*, we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention scores e^t for this step:

$$\boldsymbol{e}^t = [\boldsymbol{s}_t^T \boldsymbol{h}_1, \ldots, \boldsymbol{s}_t^T \boldsymbol{h}_N] \in \mathbb{R}^N$$

- We take softmax to get the attention distribution α^t for this step (this is a probability distribution and sums to 1)

$$\alpha^t = \text{softmax}(\boldsymbol{e}^t) \in \mathbb{R}^N$$

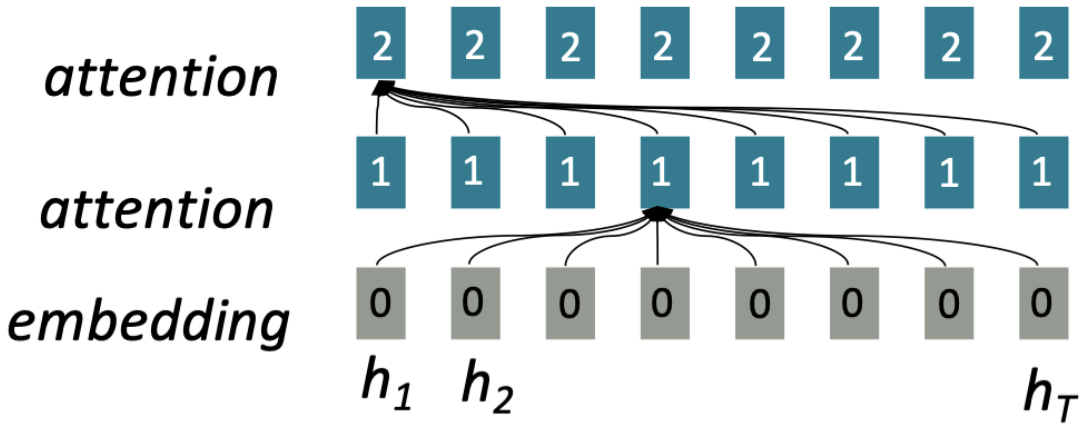
- We use α^t to take a weighted sum of the encoder hidden states to get the attention output $\boldsymbol{a}_t$

$$\boldsymbol{a}_t = \sum_{i=1}^{N} \alpha_i^t \boldsymbol{h}_i \in \mathbb{R}^h$$

- Concatenate the attention and the decoder hidden state to proceed $[\boldsymbol{a}_t; \boldsymbol{s}_t] \in \mathbb{R}^{2h}$

Attention is parallelizable, and solves bottleneck issues.

- **Attention** treats each word's representation as a **query** to access and incorporate information from **a set of values.**
  - We saw attention from the **decoder** to the **encoder**; today we'll think about attention **within a single sentence**.
- Number of unparallelizable operations does not increase with sequence length.
- Maximum interaction distance: O(1), since all words interact at every layer!



All words attend to all words in previous layer; most arrows here are omitted

Attention is great

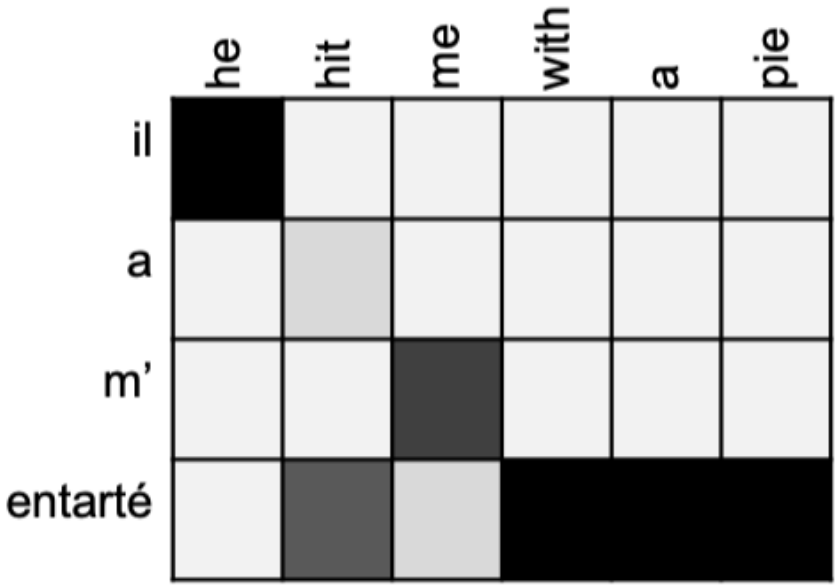
北京大学 PEKING UNIVERSITY

- Attention significantly improves NMT performance
  - It's very useful to allow decoder to focus on certain parts of the source
- Attention provides a more "human-like" model of the MT process
  - You can look back at the source sentence while translating, rather than needing to remember it all
- Attention solves the bottleneck problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention helps with the vanishing gradient problem
  - Provides shortcut to faraway states

Attention is great

北京大学 PEKING UNIVERSITY

- Attention provides some interpretability
  - By inspecting attention distribution, we see what the decoder was focusing on
  - We get (soft) alignment for free!



There are several attention variants

- We have some *values* $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N \in \mathbb{R}^{d_1}$ and a *query* $\boldsymbol{s} \in \mathbb{R}^{d_2}$

- Attention always involves:
  1. Computing the *attention scores* $\boldsymbol{e} \in \mathbb{R}^N$ ← There are multiple ways to do this
  2. Taking softmax to get *attention distribution* α:

$$\alpha = \text{softmax}(\boldsymbol{e}) \in \mathbb{R}^N$$

  3. Using attention distribution to take weighted sum of values:

$$\boldsymbol{a} = \sum_{i=1}^{N} \alpha_i \boldsymbol{h}_i \in \mathbb{R}^{d_1}$$

thus obtaining the *attention output* $\boldsymbol{a}$ (sometimes called the *context vector*)

There are several attention variants

There are several ways you can compute $e \in \mathbb{R}^N$ from $h_1, \ldots, h_N \in \mathbb{R}^{d_1}$ and $s \in \mathbb{R}^{d_2}$:

- Basic dot-product attention: $e_i = s^T h_i \in \mathbb{R}$
  - Note: this assumes $d_1 = d_2$. This is the version we saw earlier.
- Multiplicative attention: $e_i = s^T W h_i \in \mathbb{R}$ [Luong, Pham, and Manning 2015]
  - Where $W \in \mathbb{R}^{d_2 \times d_1}$ is a weight matrix. Perhaps better called "bilinear attention"
- Reduced-rank multiplicative attention: $e_i = s^T (U^T V) h_i = (Us)^T (Vh_i)$
  - For low rank matrices $U \in \mathbb{R}^{k \times d_2}, V \in \mathbb{R}^{k \times d_1}, k \ll d_1, d_2$

Remember this when we look at Transformers next week!

- Additive attention: $e_i = v^T \tanh(W_1 h_i + W_2 s) \in \mathbb{R}$ [Bahdanau, Cho, and Bengio 2014]
  - Where $W_1 \in \mathbb{R}^{d_3 \times d_1}, W_2 \in \mathbb{R}^{d_3 \times d_2}$ are weight matrices and $v \in \mathbb{R}^{d_3}$ is a weight vector.
  - d_3 (the attention dimensionality) is a hyperparameter
  - "Additive" is a weird/bad name. It's really using a feed-forward neural net layer.

Attention is a general Deep Learning technique

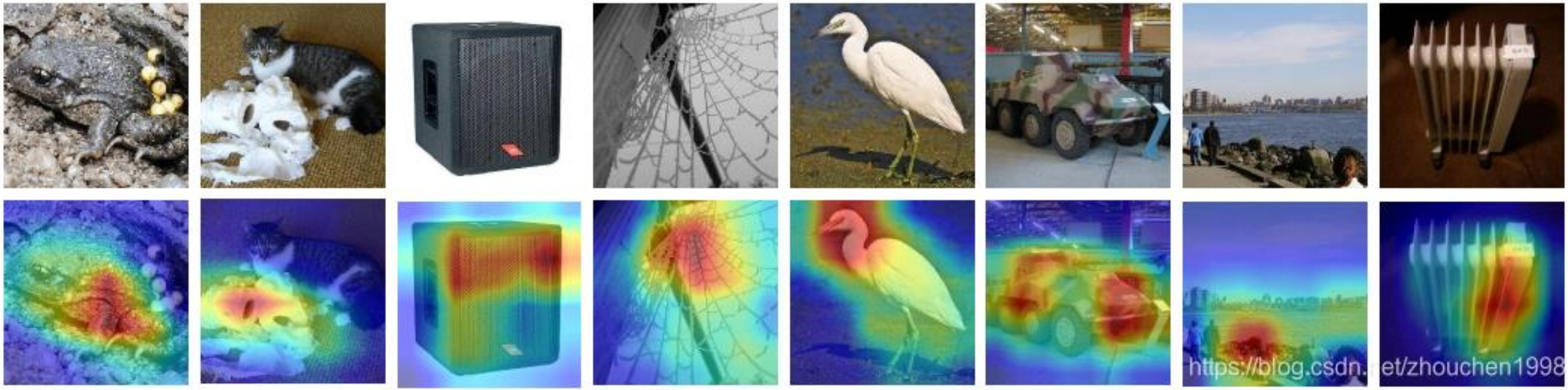
北京大学 PEKING UNIVERSITY

- We've seen that attention is a great way to improve the sequence-to-sequence model for Machine Translation.
- However: You can use attention in many architectures (not just seq2seq) and many tasks (not just MT)

- **More general definition of attention**:
  - Given a set of vector *values*, and a vector *query*, **attention** is a technique to compute a weighted sum of the values, dependent on the query.

- We sometimes say that the query *attends to* the values.
- For example, in the seq2seq + attention model, each decoder hidden state (query) *attends to* all the encoder hidden states (values).

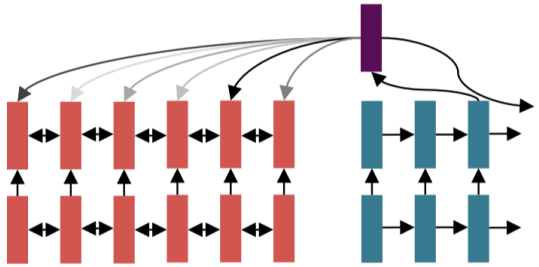
Attention in computer vision



Do we even need recurrence at all?

北京大学 PEKING UNIVERSITY

- Abstractly: Attention is a way to pass information from a sequence (x) to a neural network input. (h_t)
  - This is also *exactly* what RNNs are used for – to pass information!
  - **Can we just get rid of the RNN entirely?** Maybe attention is just a better way to pass information!

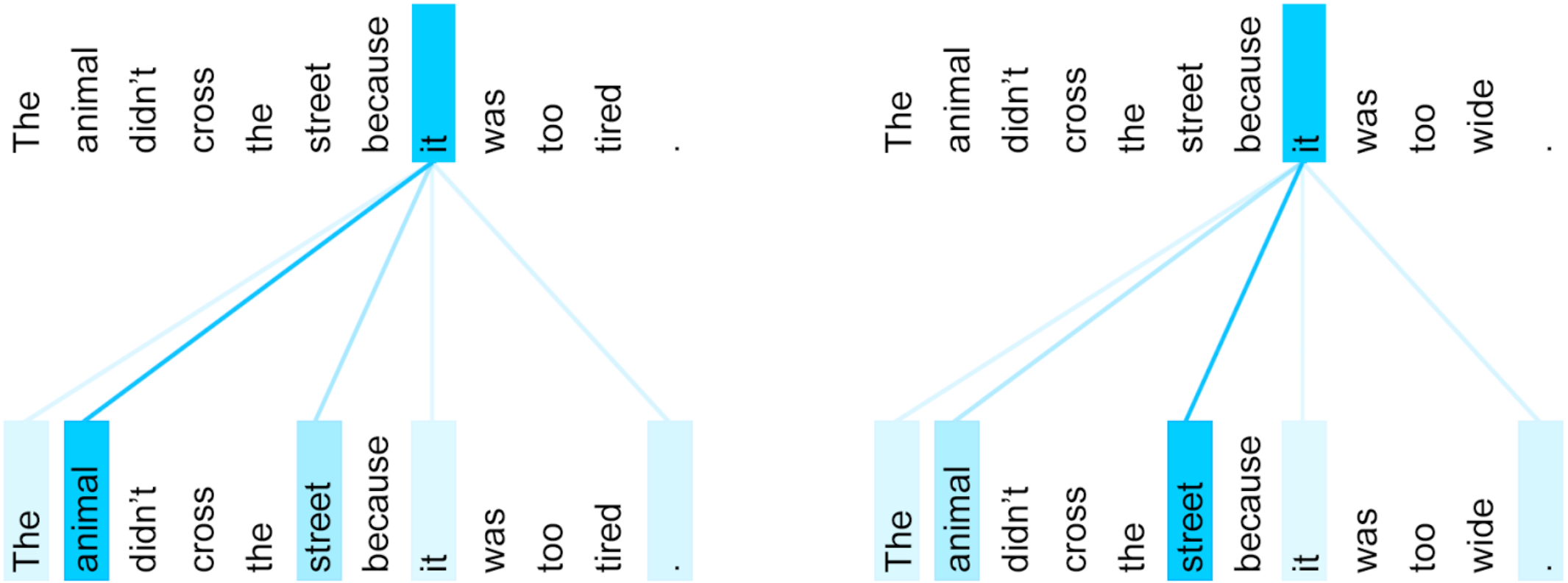


Lots of trial and error

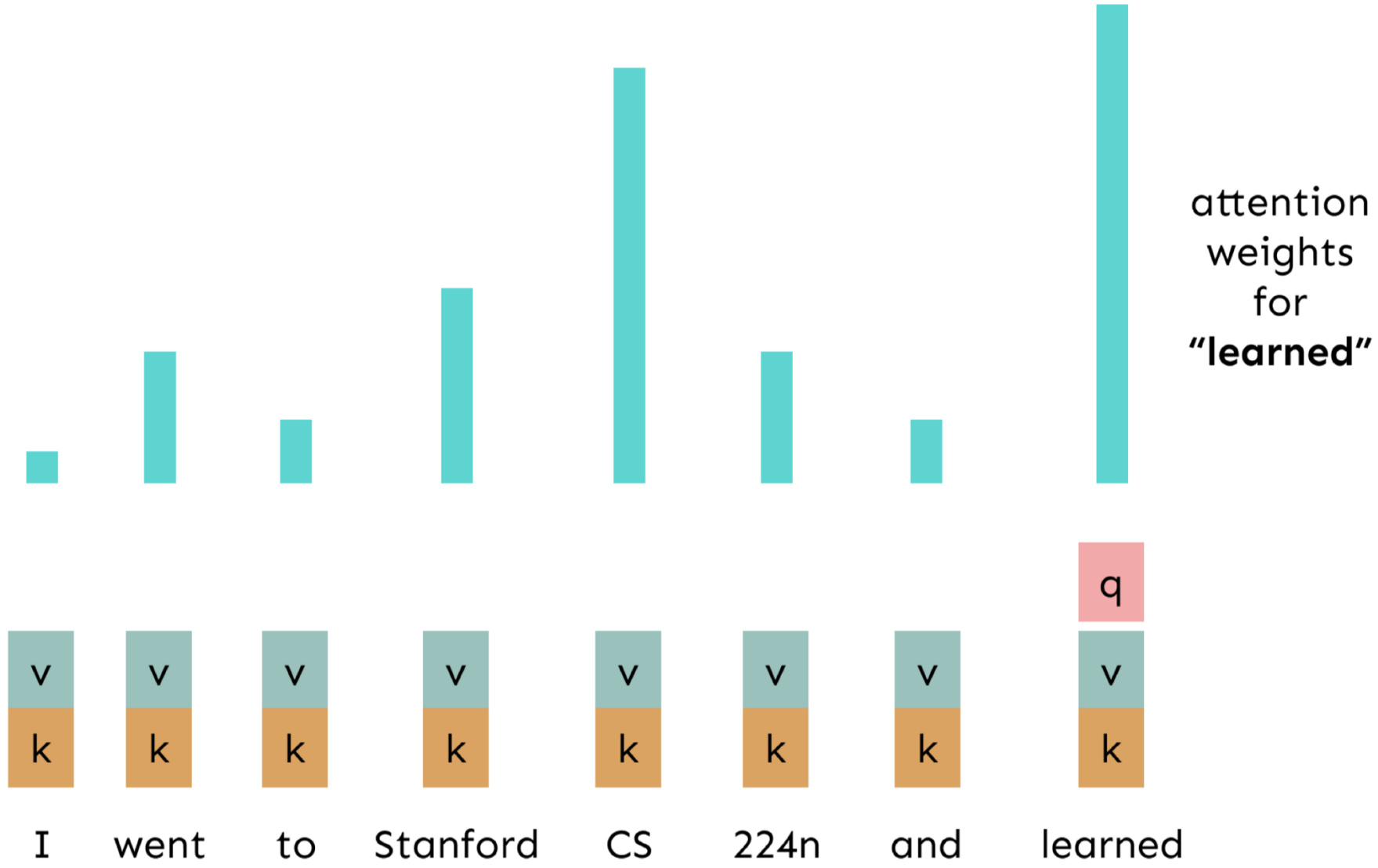
2014-2017ish Recurrence

2021 ??????

The building block we need: self attention



Self-Attention Hypothetical Example



Self-Attention: keys, queries, values from the same sequence

Let $\boldsymbol{w}_{1:n}$ be a sequence of words in vocabulary V, like *Zuko made his uncle tea*.

For each $\boldsymbol{w}_i$, let $\boldsymbol{x}_i = E\boldsymbol{w}_i$, where $E \in \mathbb{R}^{d \times |V|}$ is an embedding matrix.

1. Transform each word embedding with weight matrices Q, K, V, each in $\mathbb{R}^{d \times d}$

$\boldsymbol{q}_i = Q\boldsymbol{x}_i$ (queries) $\quad \boldsymbol{k}_i = K\boldsymbol{x}_i$ (keys) $\quad \boldsymbol{v}_i = V\boldsymbol{x}_i$ (values)

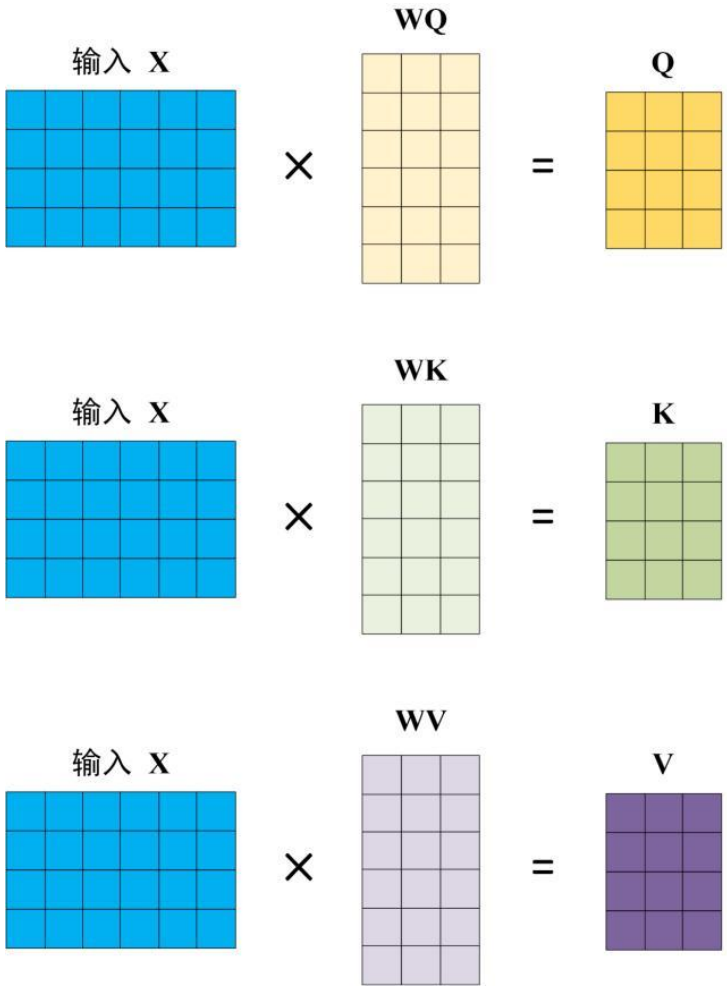
2. Compute pairwise similarities between keys and queries; normalize with softmax

$$\boldsymbol{e}_{ij} = \boldsymbol{q}_i^\top \boldsymbol{k}_j \qquad \boldsymbol{\alpha}_{ij} = \frac{\exp(\boldsymbol{e}_{ij})}{\sum_{j'} \exp(\boldsymbol{e}_{ij'})}$$

3. Compute output for each word as weighted sum of values

$$\boldsymbol{o}_i = \sum_j \boldsymbol{\alpha}_{ij} \boldsymbol{v}_i$$

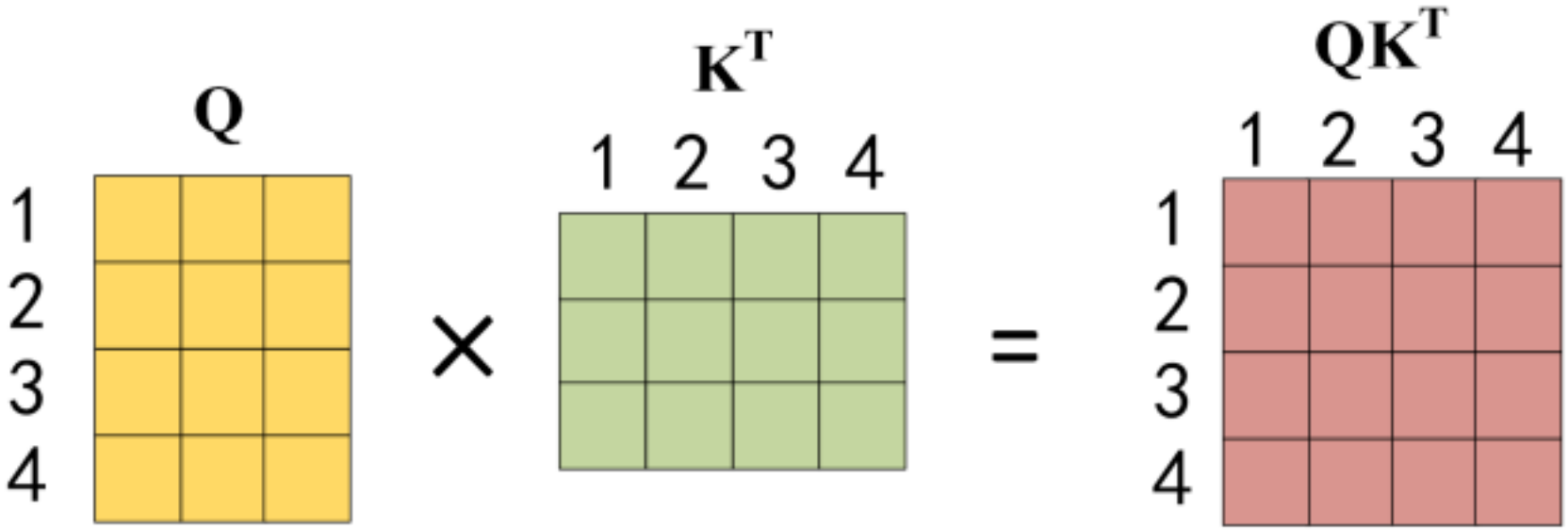
Self-Attention: matrix representation



Self-Attention: matrix representation

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

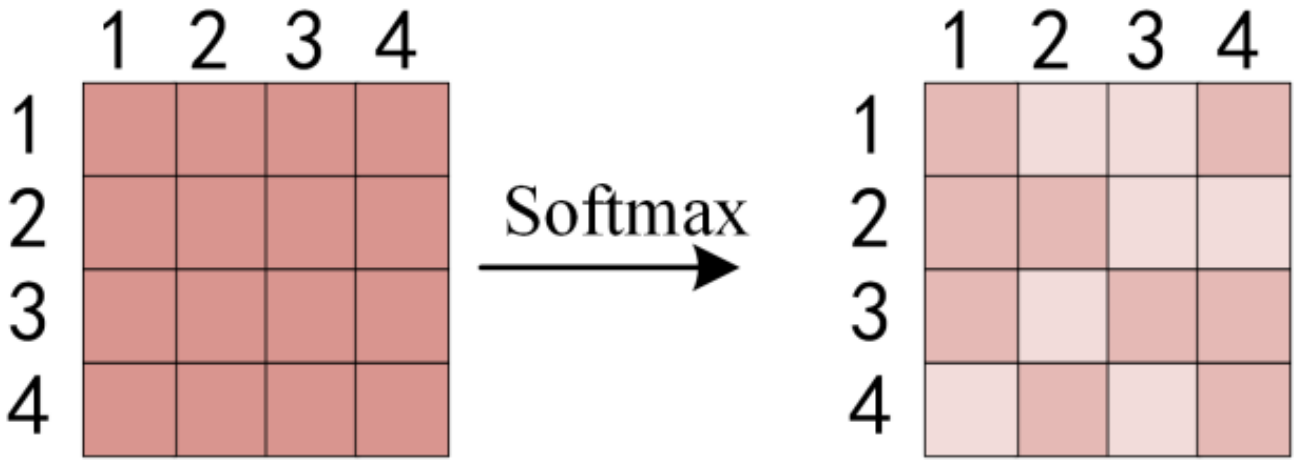
d_k是Q, K矩阵的列数，即向量维度



Self-Attention: matrix representation

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

d_k是Q,K矩阵的列数，即向量维度

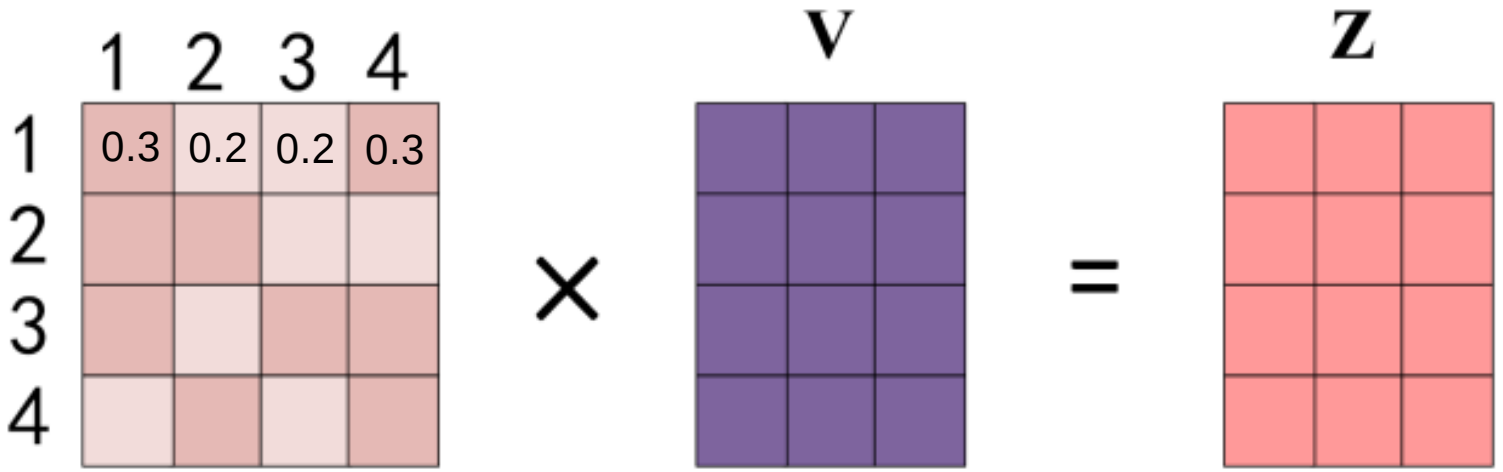


Self-Attention: matrix representation

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

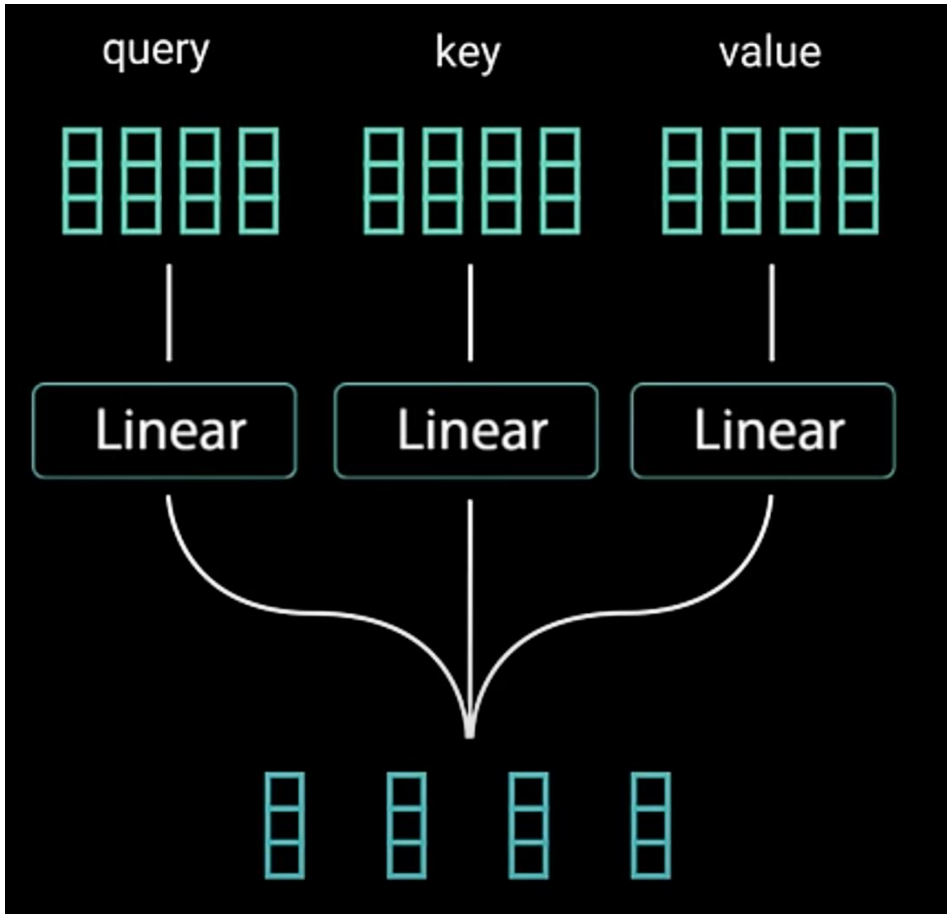
d_k是Q, K矩阵的列数，即向量维度

Very efficient
To calculate



More examples on self-attention

Get Q, K, and V

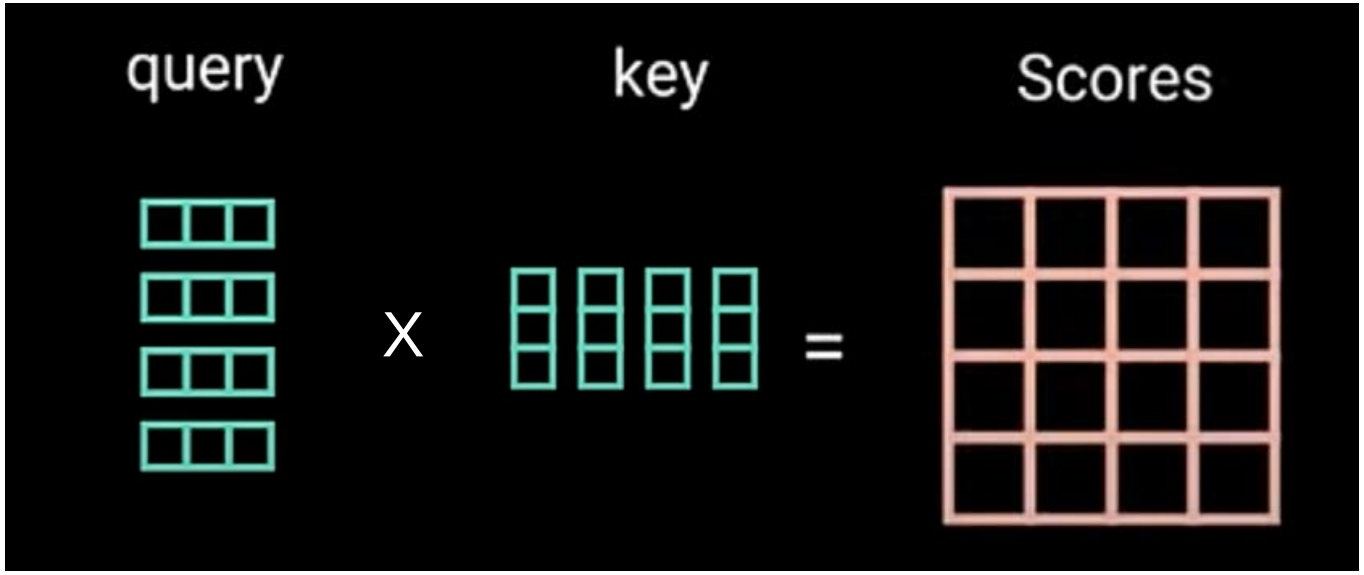


[https://www.youtube.com/watch?v=4Bdc55j80l8]

More examples on self-attention

北京大学 PEKING UNIVERSITY

Get Scores

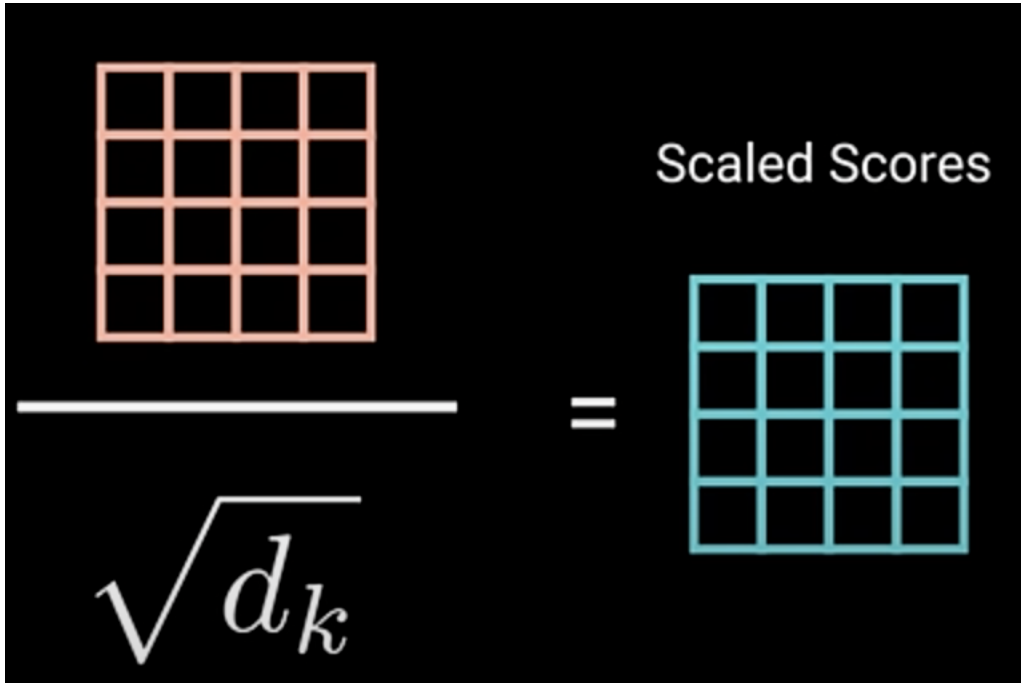


	Hi	how	are	you
Hi	98	27	10	12
how	27	89	31	67
are	10	31	91	54
you	12	67	54	92

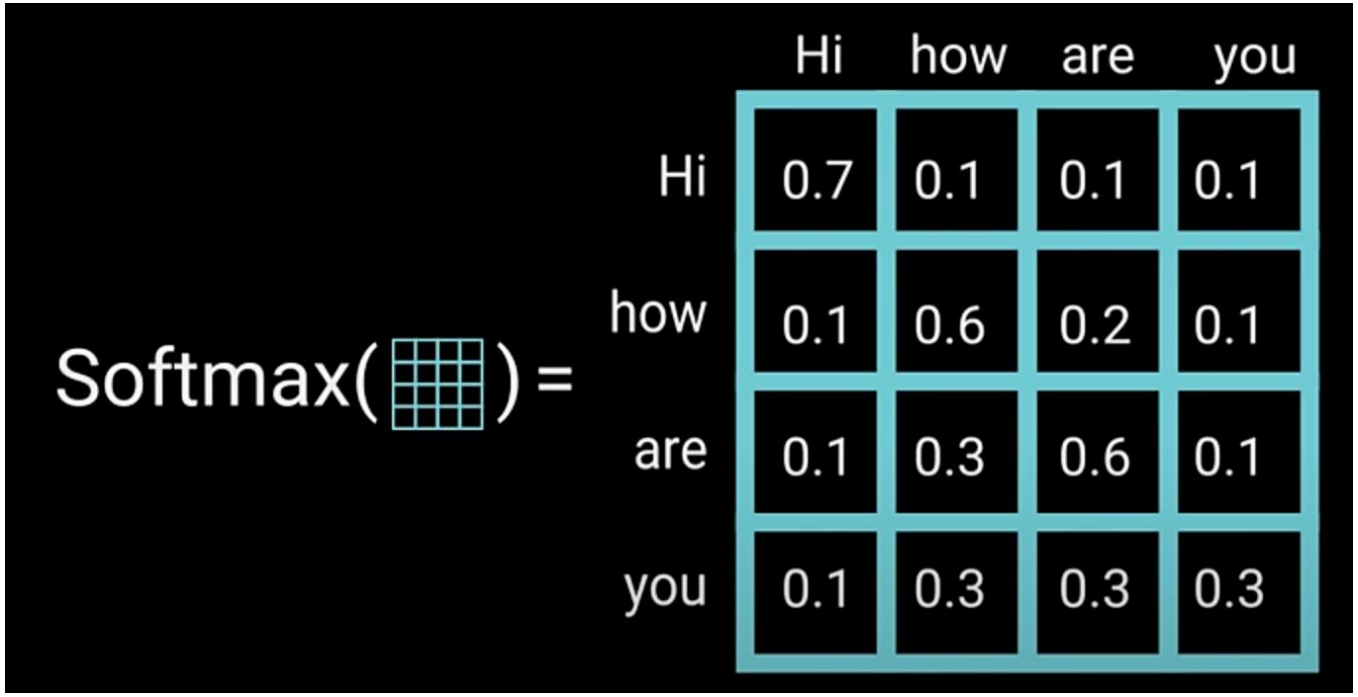
[https://www.youtube.com/watch?v=4Bdc55j80l8]

More examples on self-attention

Scaled scores



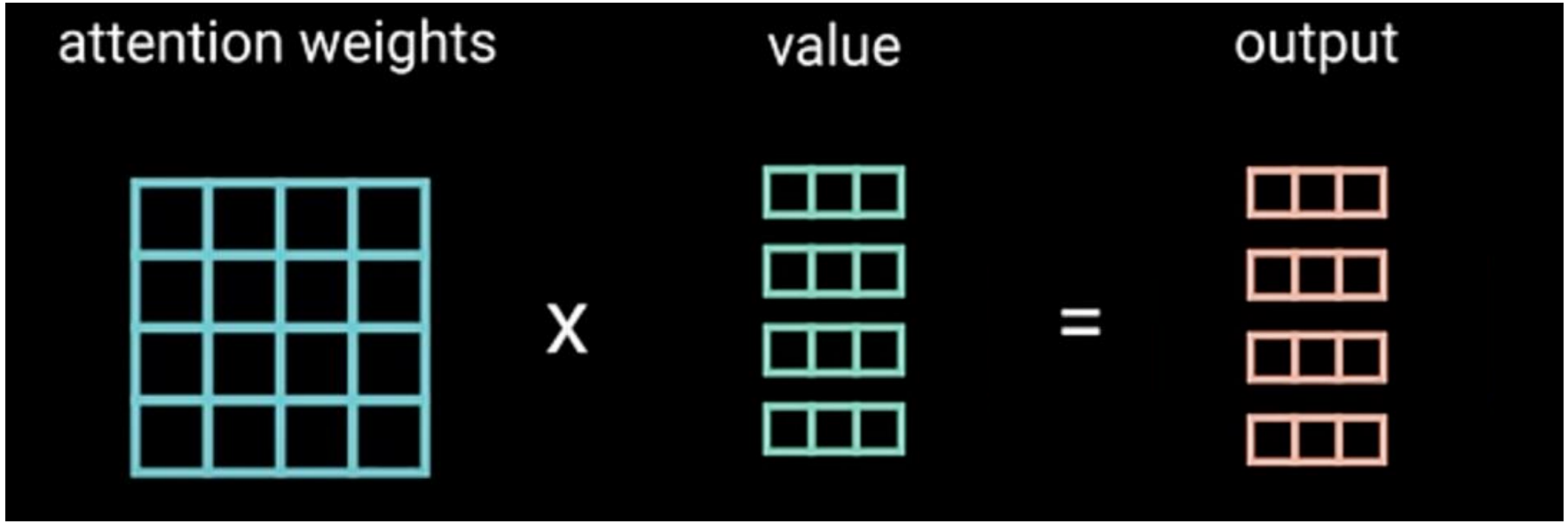
Get weights



[https://www.youtube.com/watch?v=4Bdc55j80l8]

More examples on self-attention

Get output



[https://www.youtube.com/watch?v=4Bdc55j80l8]

Barriers and solutions for Self-Attention as a building block

Barriers	Solutions
• Doesn't have an inherent notion of order!	→

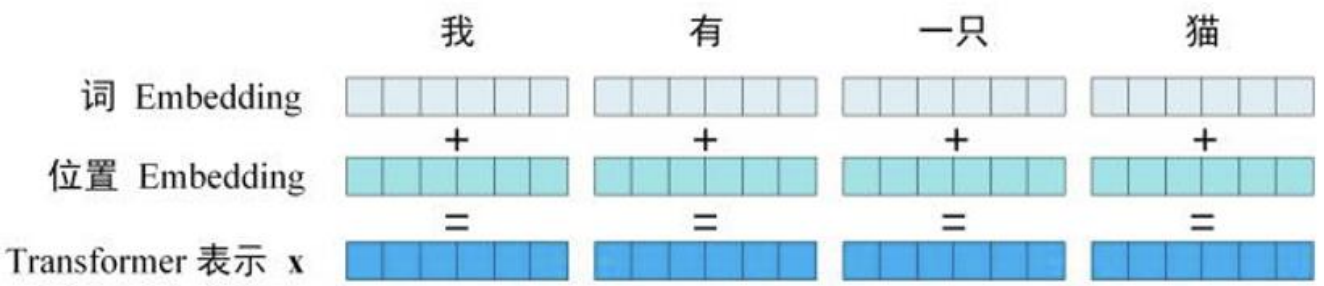
Position embedding

- Since self-attention doesn't build in order information, we need to encode the order of the sentence in our keys, queries, and values.
- Consider representing each **sequence index** as a **vector**

$$\boldsymbol{p}_i \in \mathbb{R}^d, \text{ for } i \in \{1,2,\dots,n\} \text{ are position vectors}$$

- Easy to incorporate this info into our self-attention block: just add the $\boldsymbol{p}_i$ to our inputs!
- Recall that $\boldsymbol{x}_i$ is the embedding of the word at index i. The positioned embedding is:

$$\tilde{\boldsymbol{x}}_i = \boldsymbol{x}_i + \boldsymbol{p}_i$$



Position embedding

- **Learned absolute position representations:** Let all p_i be learnable parameters!
  Learn a matrix $\boldsymbol{p} \in \mathbb{R}^{d \times n}$, *and let each* $\boldsymbol{p}_i$ *be a column of that matrix!*

- Pros:
  - Flexibility: each position gets to be learned to fit the data
- Cons:
  - Definitely can't extrapolate to indices outside $1, \ldots, n$.
- Most systems use this!

Barriers and solutions for Self-Attention as a building block

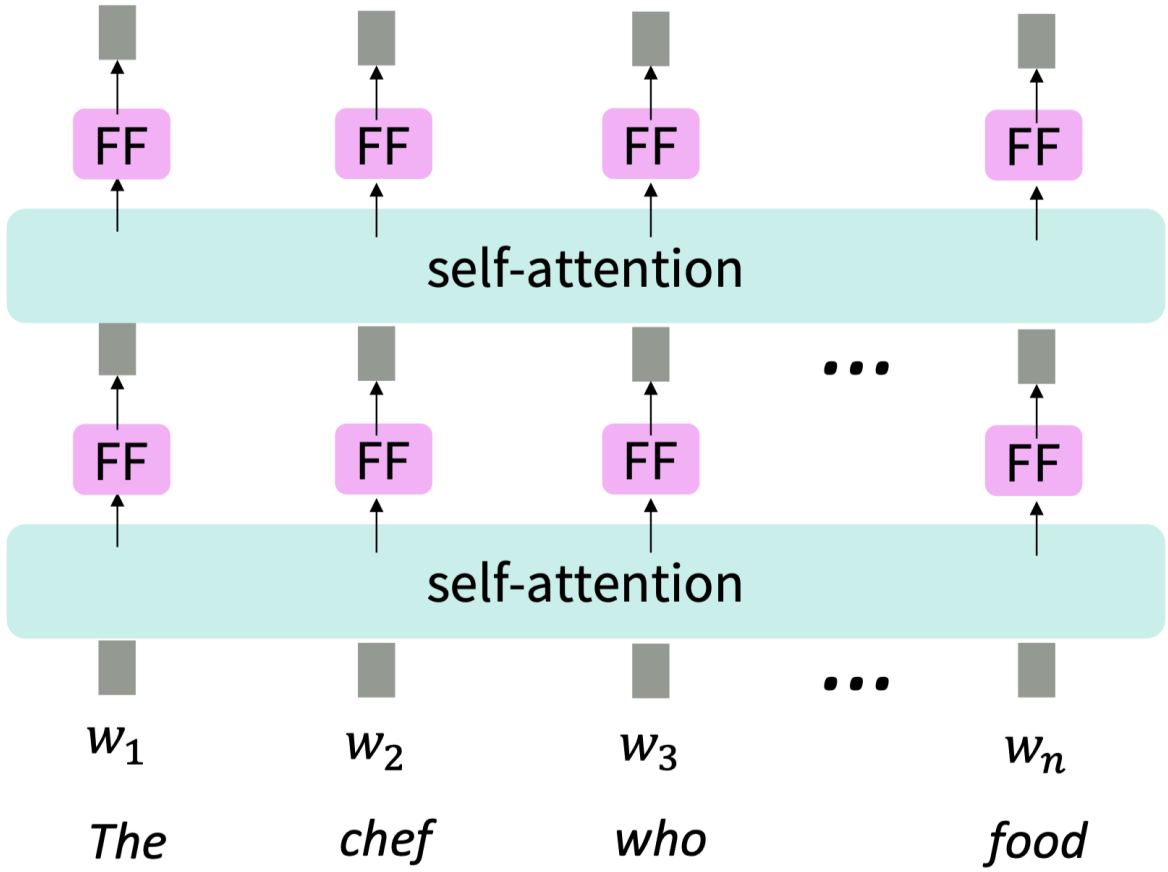
Barriers	Solutions
• Doesn't have an inherent notion of order!	• Add position representations to the inputs
• No nonlinearities for deep learning! It's all just weighted averages	

Adding nonlinearities in self-attention

- Note that there are no elementwise nonlinearities in self-attention; stacking more self-attention layers just re-averages **value** vectors (Why? Look at the notes!)

- Easy fix: add a **feed-forward network** to post-process each output vector.

$$m_i = MLP(\text{output}_i)$$
$$= W_2 * \text{ReLU}(W_1\ \text{output}_i + b_1) + b_2$$



Intuition: the FF network processes the result of attention

40

Barriers and solutions for Self-Attention as a building block

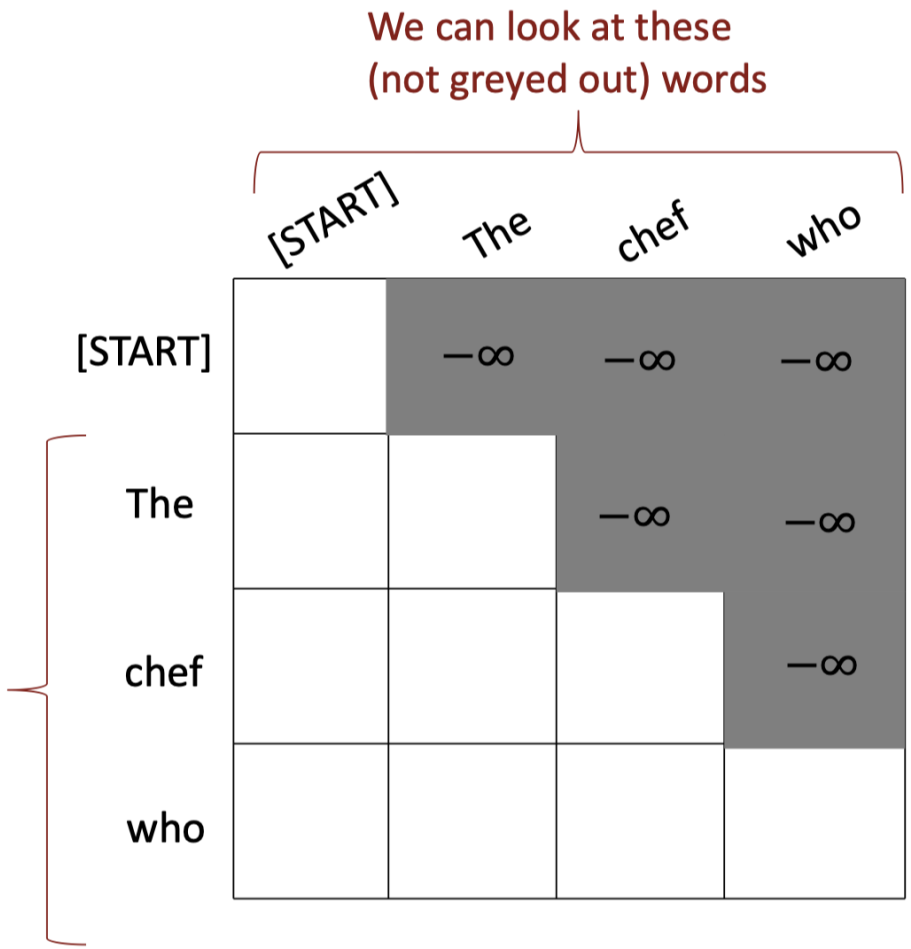
Barriers	Solutions
• Doesn't have an inherent notion of order!	• Add position representations to the inputs
• No nonlinearities for deep learning magic! It's all just weighted averages	• Easy fix: apply the same feedforward network to each self-attention output.

• Need to ensure we don't "look at the future" when predicting a sequence
  • Like in machine translation
  • Or language modeling	

Barriers and solutions for Self-Attention as a building block

- To use self-attention in **decoders**, we need to ensure we can't peek at the future.
- At every timestep, we could change the set of **keys and queries** to include only past words. (Inefficient!)
- To enable parallelization, we **mask out attention** to future words by setting attention scores to $-\infty$.

$$e_{ij} = \begin{cases} q_i^\top k_j, j \leq i \\ -\infty, j > i \end{cases}$$



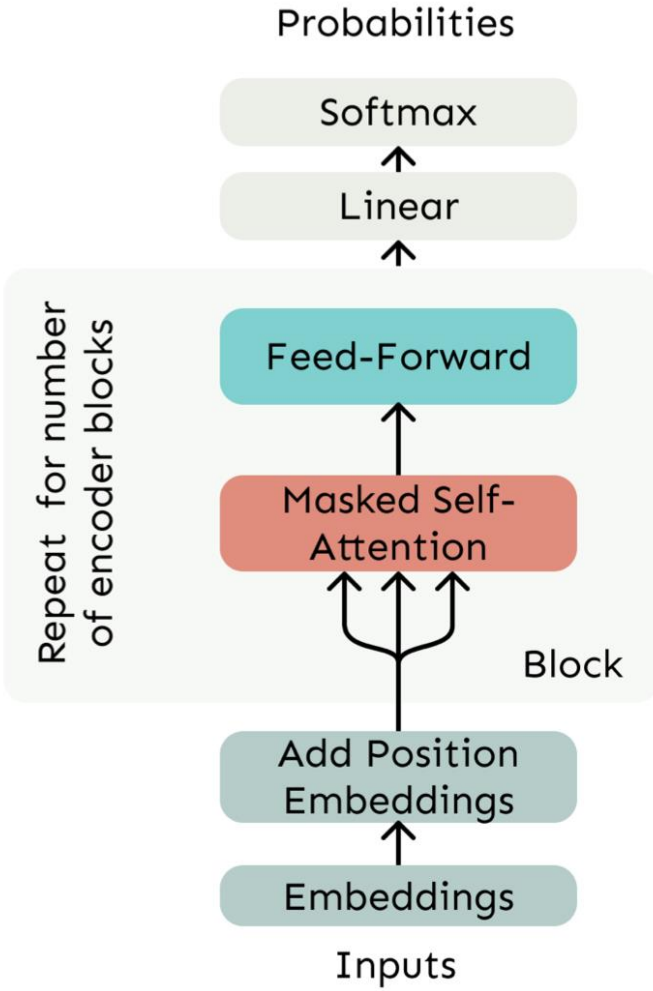
Barriers and solutions for Self-Attention as a building block

Barriers	Solutions
• Doesn't have an inherent notion of order!	• Add position representations to the inputs
• No nonlinearities for deep learning magic! It's all just weighted averages	• Easy fix: apply the same feedforward network to each self-attention output.

• Need to ensure we don't "look at the future" when predicting a sequence
  • Like in machine translation
  • Or language modeling
	• Mask out the future by artificially setting attention weights to 0!

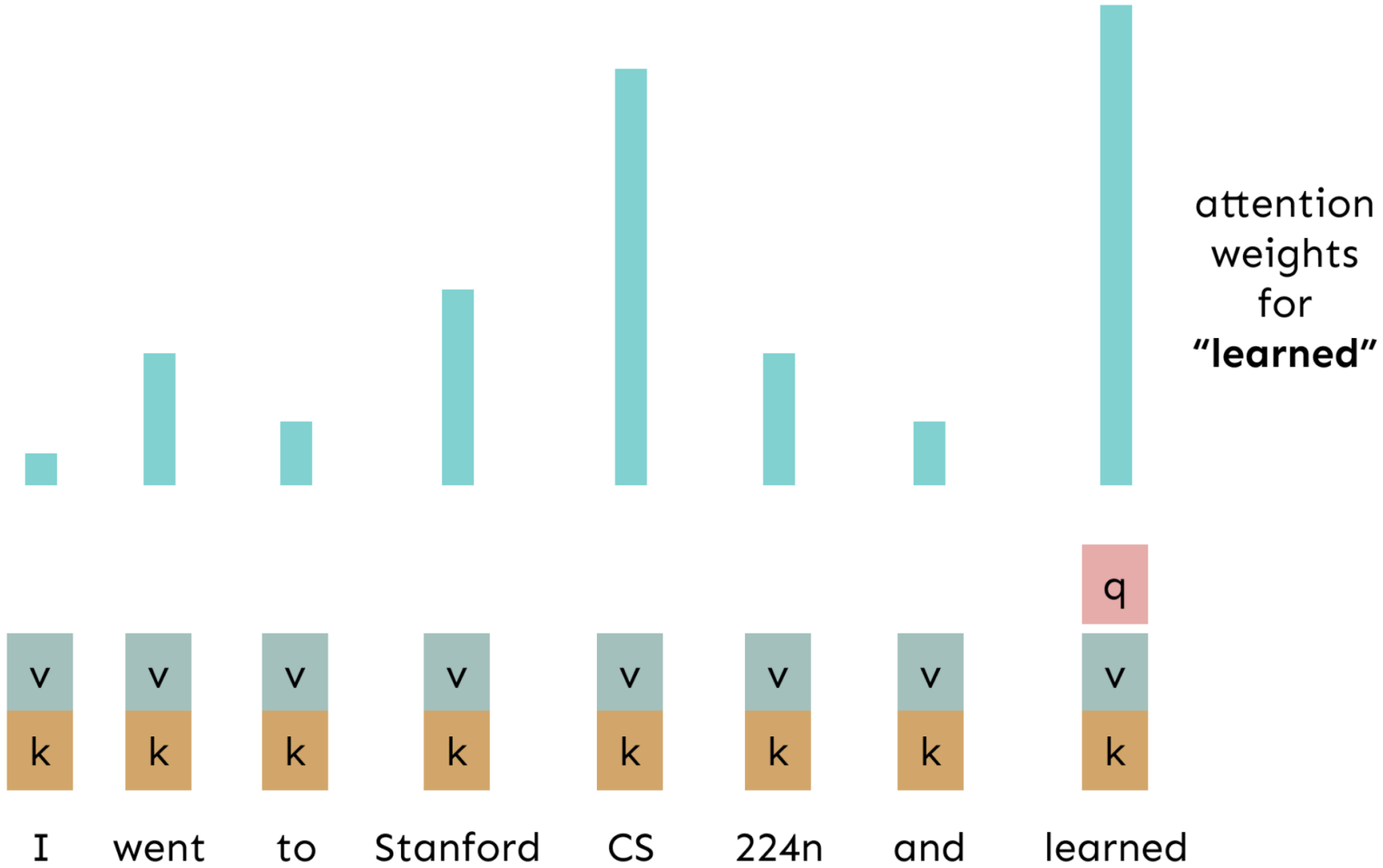
Necessities for a self-attention building block:

- **Self-attention**:
  - the basis of the method.
- **Position representations**:
  - Specify the sequence order, since self-attention is an unordered function of its inputs.
- **Nonlinearities**:
  - At the output of the self-attention block
  - Frequently implemented as a simple feed-forward network.
- **Masking**:
  - In order to parallelize operations while not looking at the future.
  - Keeps information about the future from "leaking" to the past.

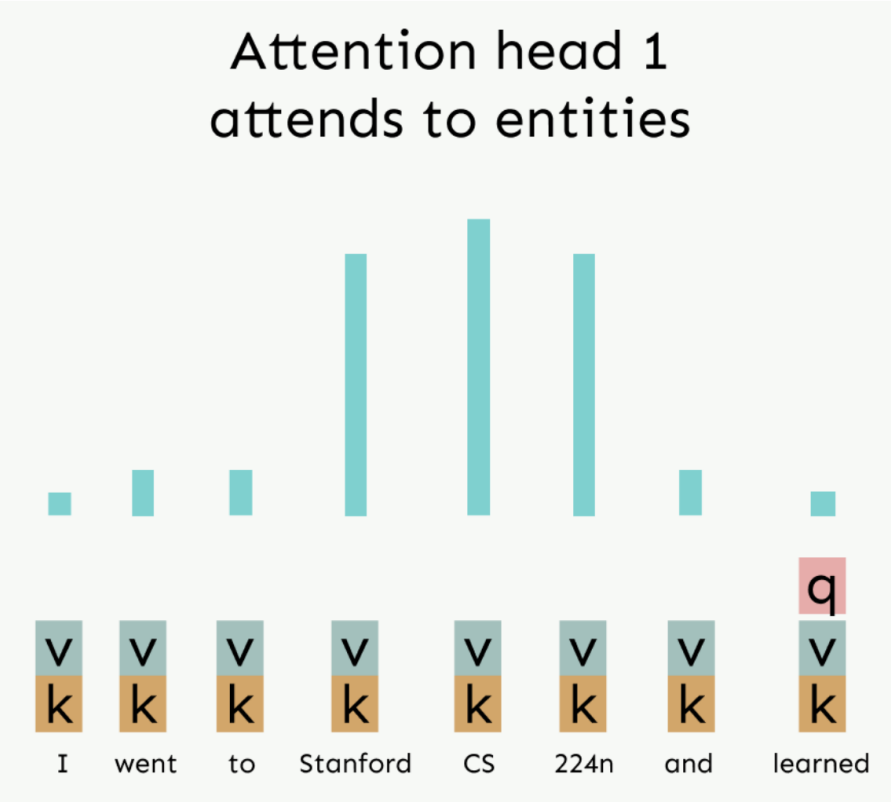


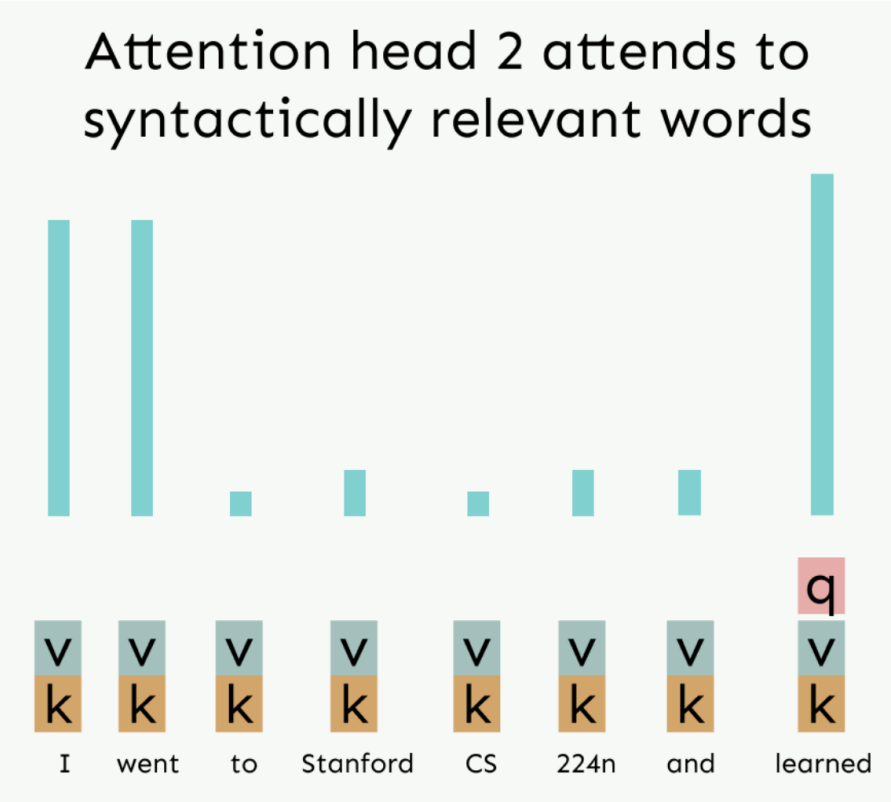
Transformer decoder

Recall the Self-Attention Hypothetical Example



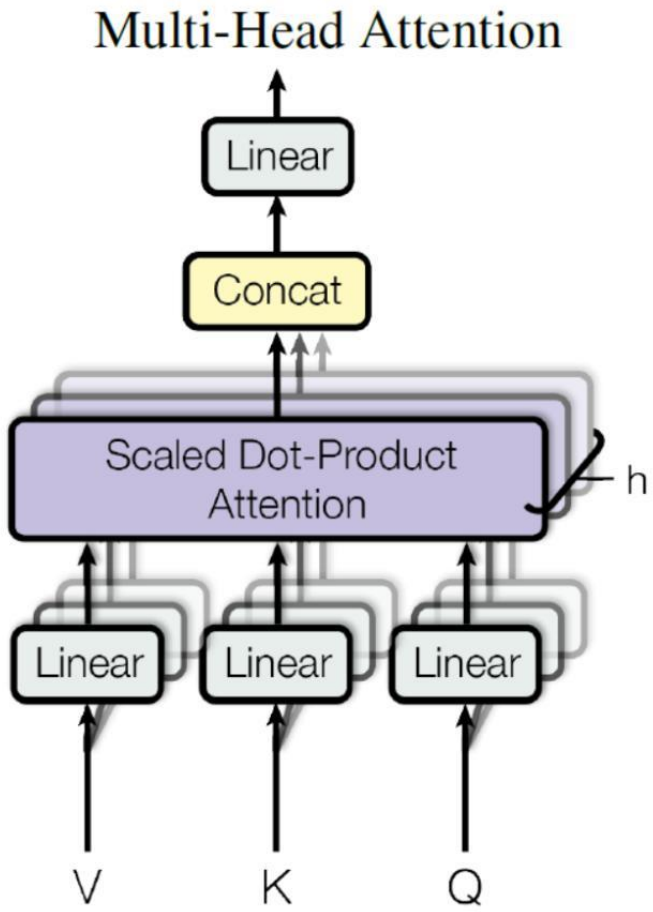
Hypothetical Example of Multi-Head Attention

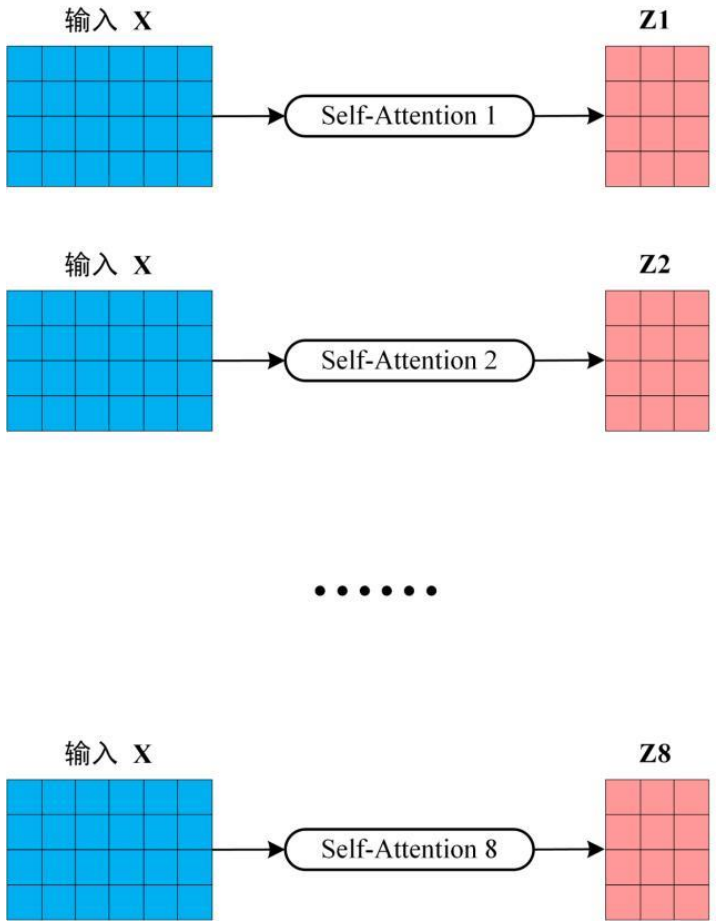




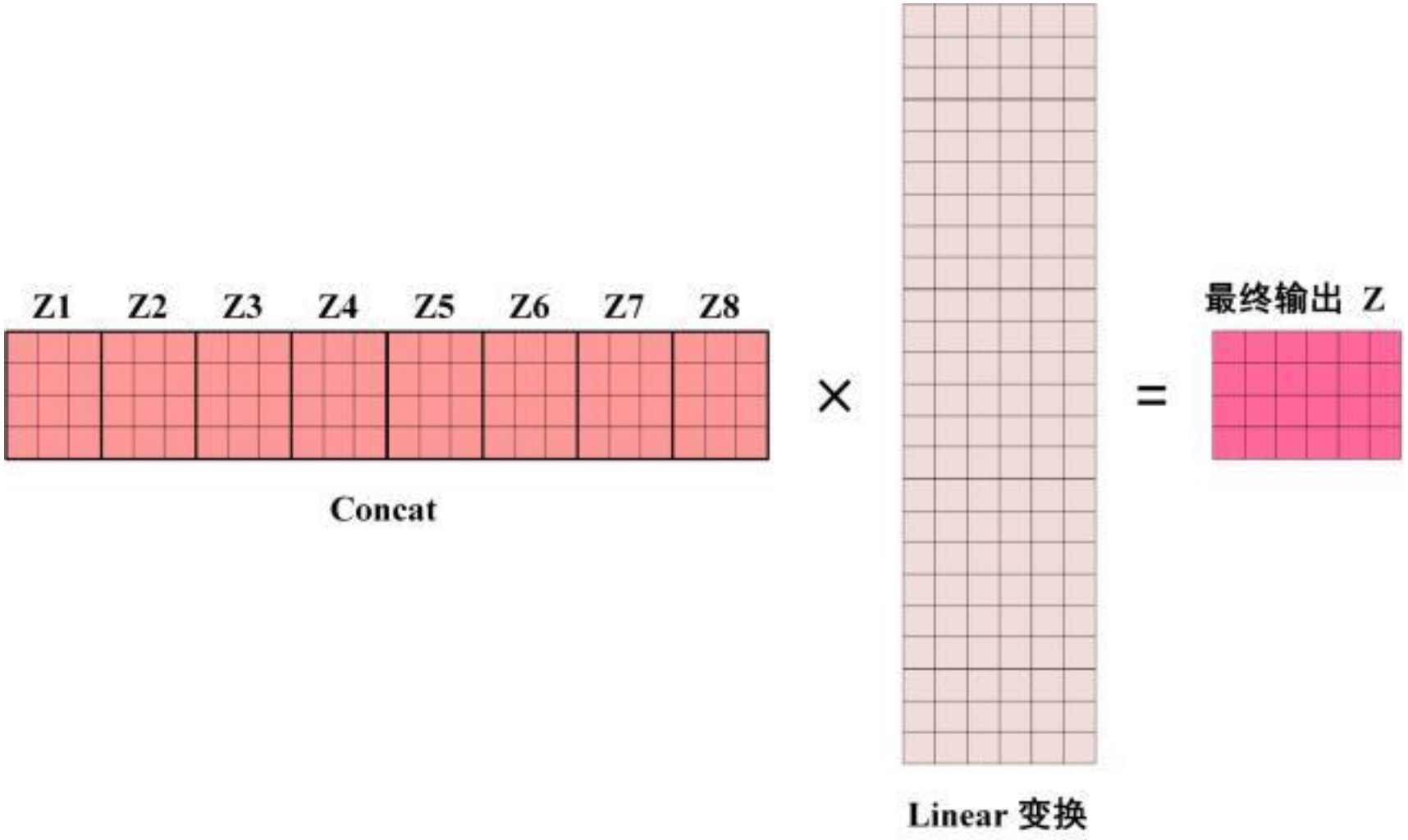
I went to Stanford CS 224n and learned

Use multi-head attention to achieve better representation





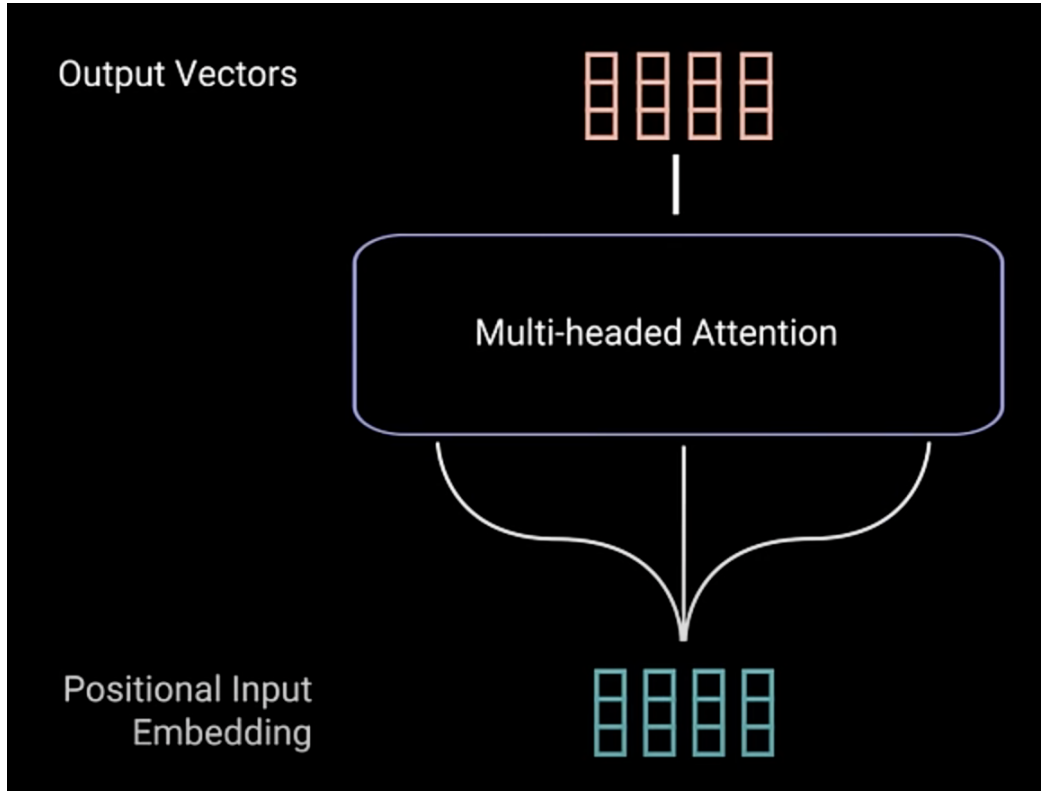
Multi-head attention: concatenation and linear transformation



Multi-head attention recap

Multi-head attention is to find representations for the input words

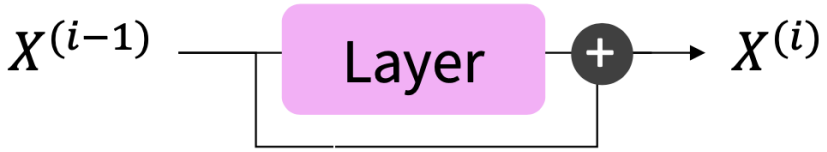
It provides better representations than single self-attention



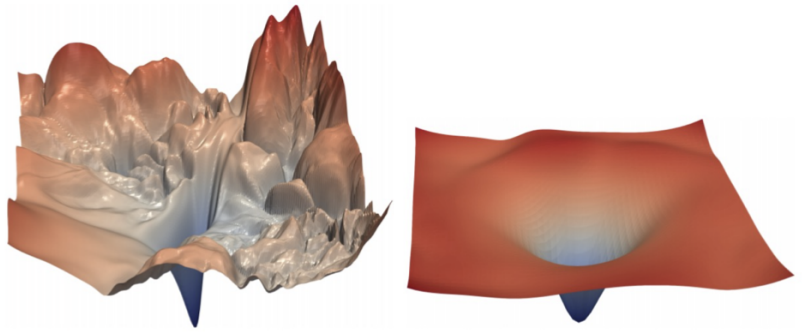
Residual connection

- **Residual connections** are a trick to help models train better.
  - Instead of $X^{(i)} = \text{Layer}(X^{(i-1)})$ (where i represents the layer)

  - We let $X^{(i)} = X^{(i-1)} + \text{Layer}(X^{(i-1)})$ (so we only have to learn "the residual" from the previous layer)



  - Gradient is **great** through the residual connection; it's 1!
  - Bias towards the identity function!



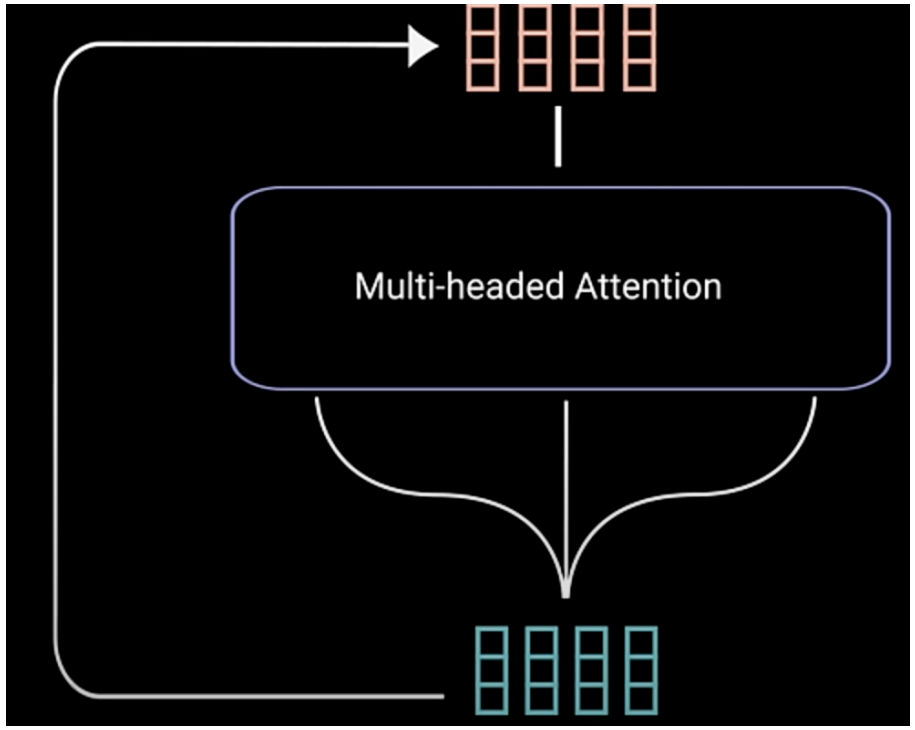
[no residuals] [residuals]

[Loss landscape visualization, Li et al., 2018, on a ResNet]

Residual connection

北京大学 PEKING UNIVERSITY

Residual connection



Output

Layer-normalization

- **Layer normalization** is a trick to help models train faster.

- Idea: cut down on uninformative variation in hidden vector values by normalizing to unit mean and standard deviation **within each layer**.
  - LayerNorm's success may be due to its normalizing gradients [Xu et al., 2019]

- Let $x \in \mathbb{R}^d$ be an individual (word) vector in the model.
- Let $\mu = \sum_{j=1}^{d} x_j$; this is the mean; $\mu \in \mathbb{R}$.
- Let $\sigma = \sqrt{\frac{1}{d} \sum_{j=1}^{d} (x_j - \mu)^2}$; this is the standard deviation; $\sigma \in \mathbb{R}$.

- Then layer normalization computes:

Can be ignored

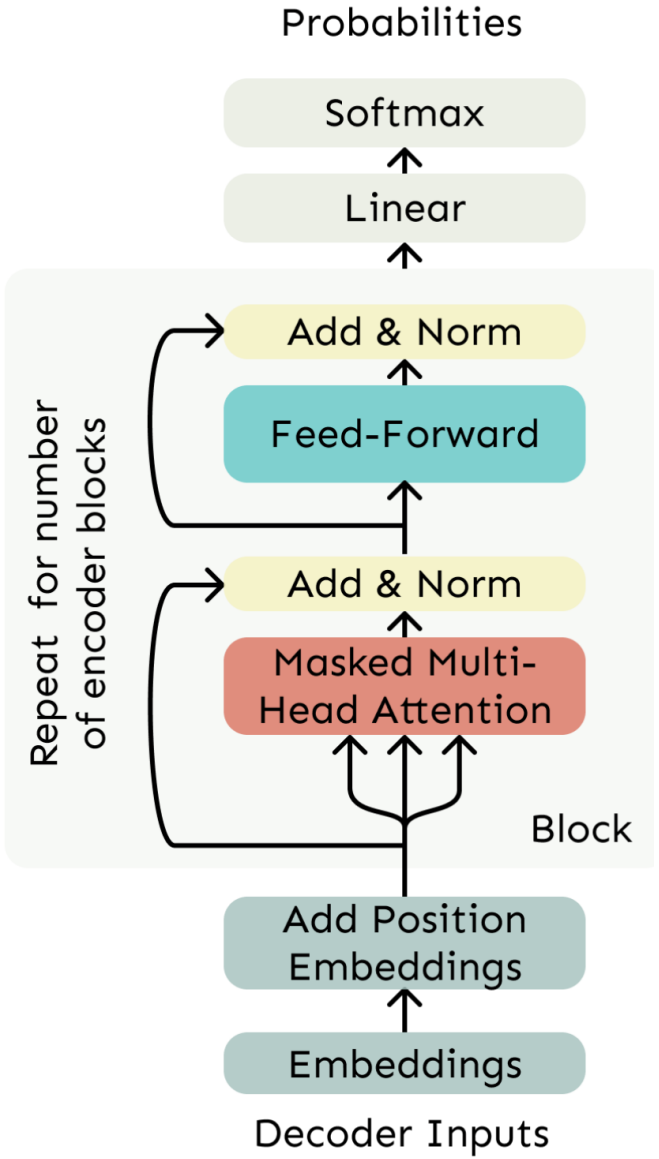
$$\text{output} = \frac{x - \mu}{\sqrt{\sigma + \epsilon}} * \gamma + \beta$$

Normalize by scalar mean and variance

Modulate by learned elementwise gain and bias

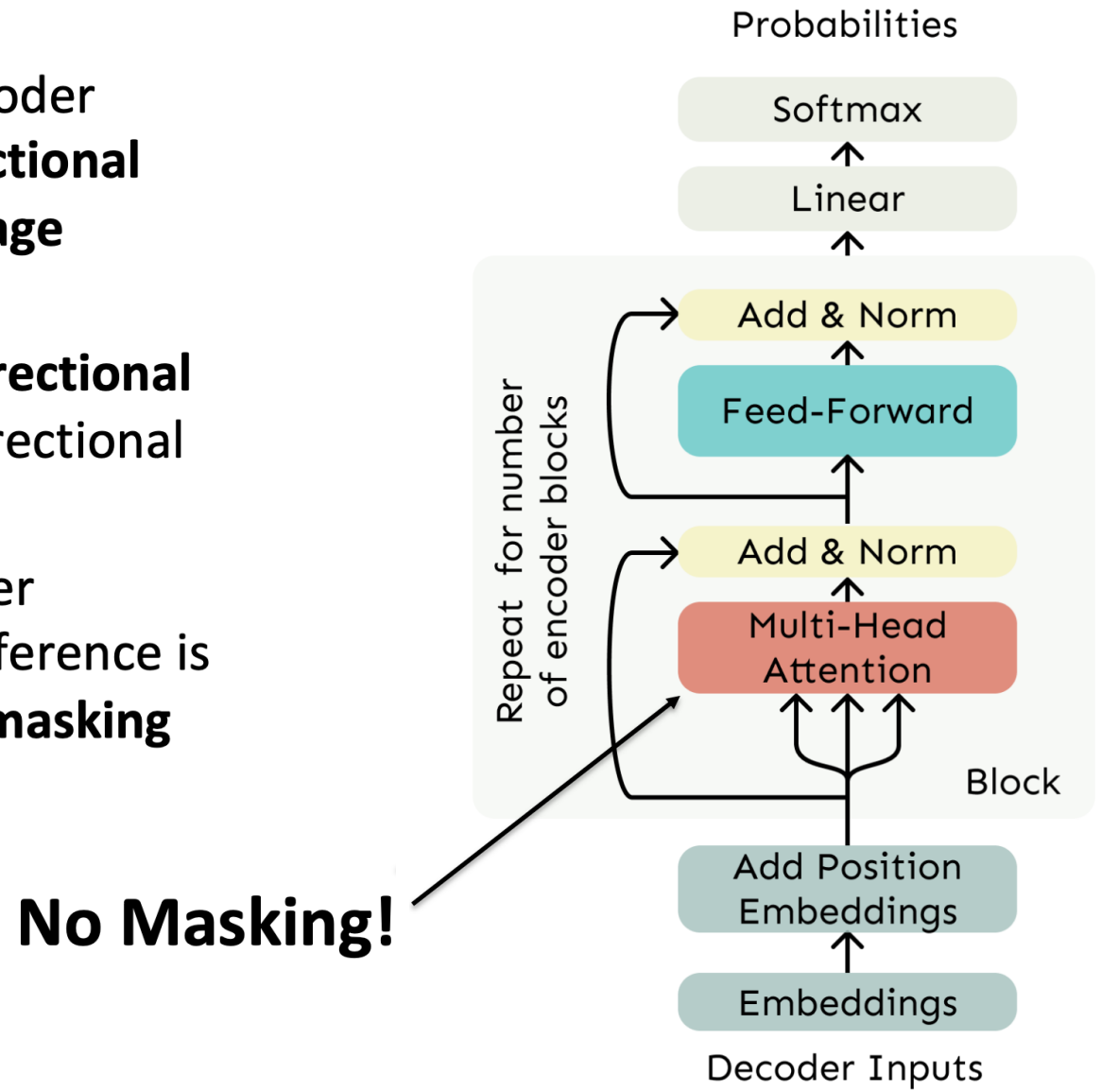
The transformer decoder

- The Transformer Decoder is a stack of Transformer Decoder **Blocks**.
- Each Block consists of:
  - Self-attention
  - Add & Norm
  - Feed-Forward
  - Add & Norm
- That's it! We've gone through the Transformer Decoder.

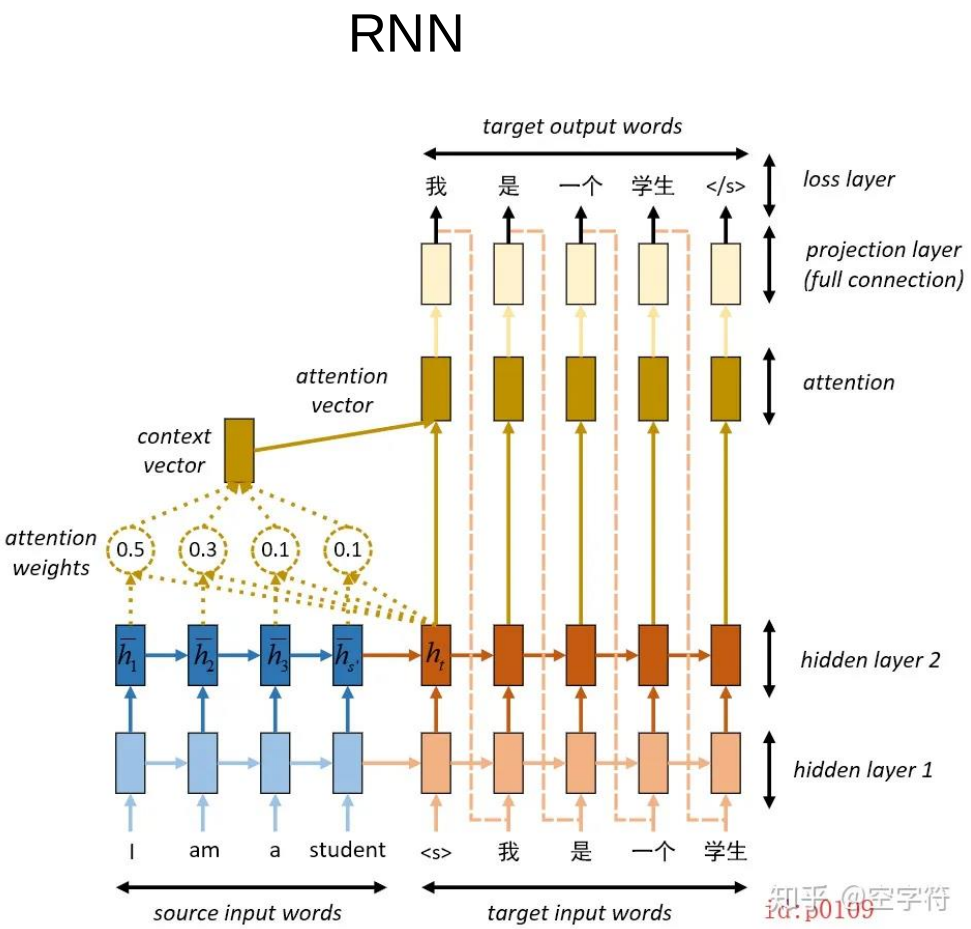


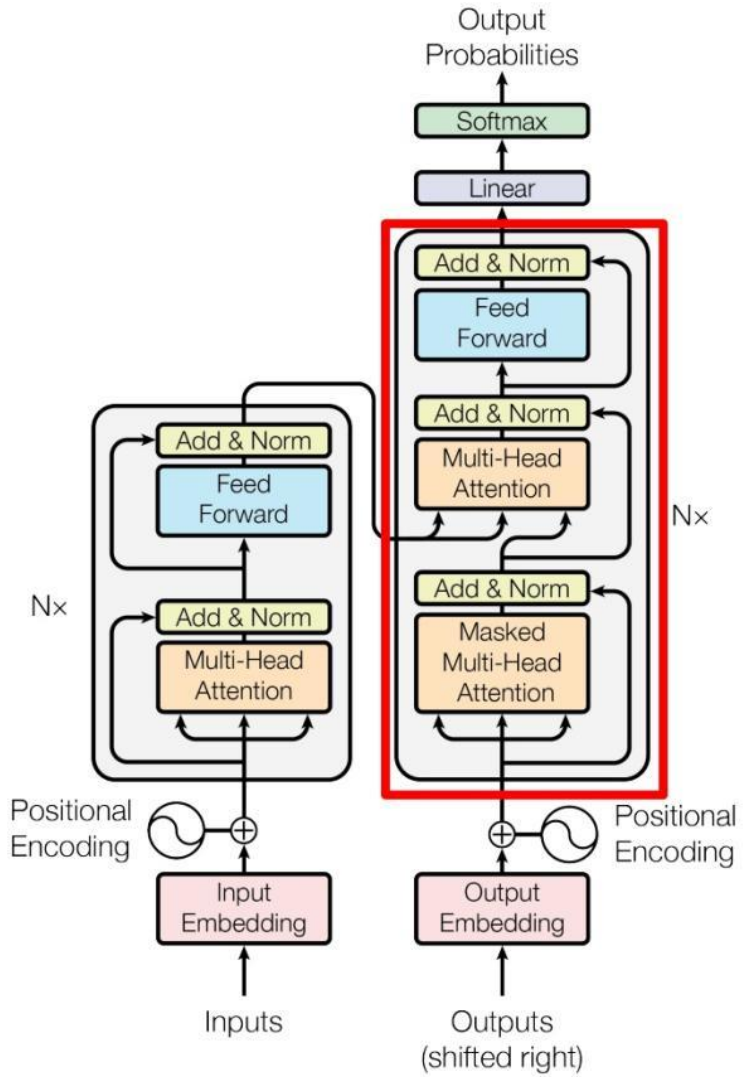
The transformer encoder

- The Transformer Decoder constrains to **unidirectional context**, as for **language models.**
- What if we want **bidirectional context**, like in a bidirectional RNN?
- This is the Transformer Encoder. The only difference is that we **remove the masking** in the self-attention.



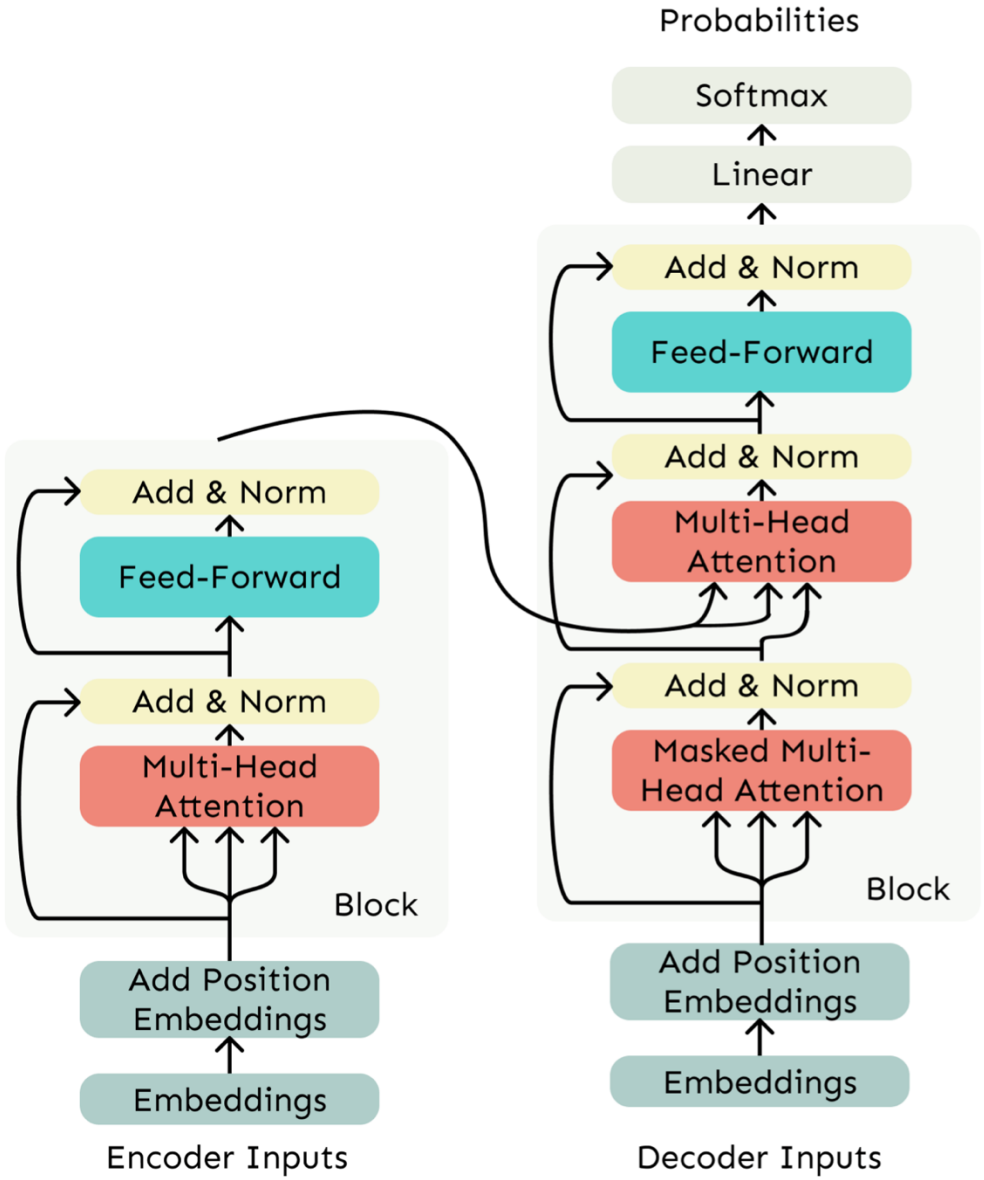
The transformer



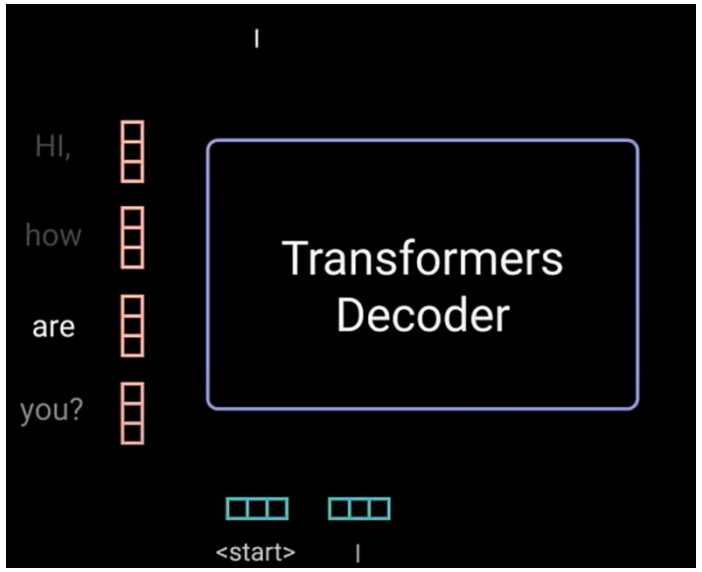


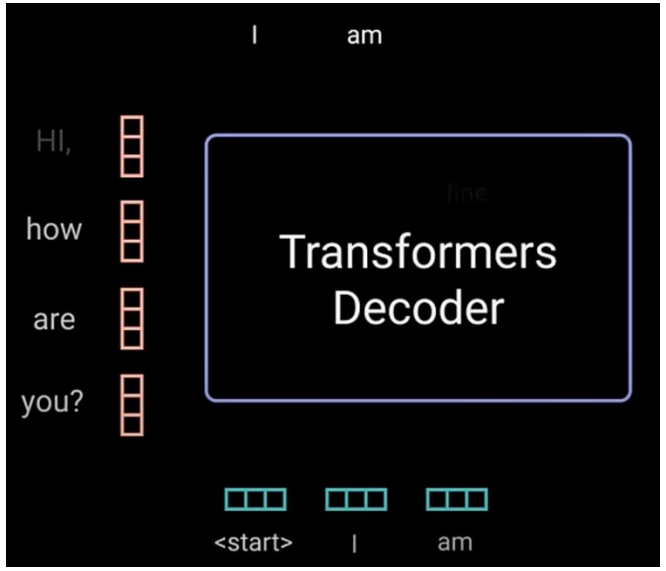
Cross attention

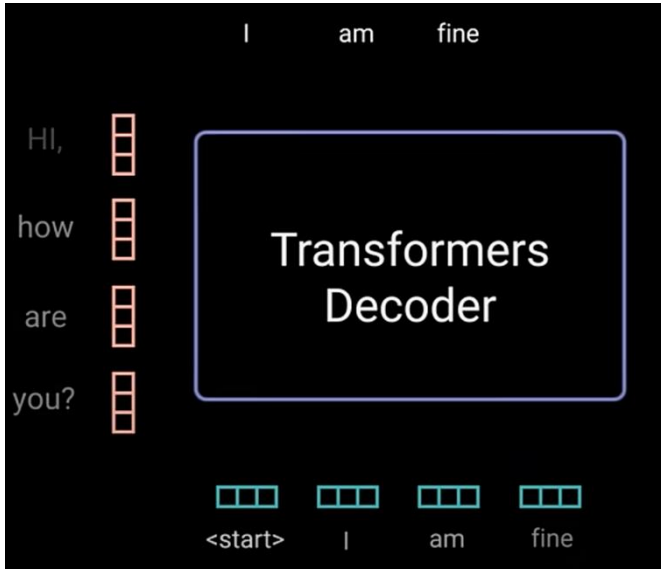
- Recall that in machine translation, we processed the source sentence with a **bidirectional** model and generated the target with a **unidirectional model**.
- For this kind of seq2seq format, we often use a Transformer Encoder-Decoder.
- We use a normal Transformer Encoder.
- Our Transformer Decoder is modified to perform **cross-attention** to the output of the Encoder.



Cross attention

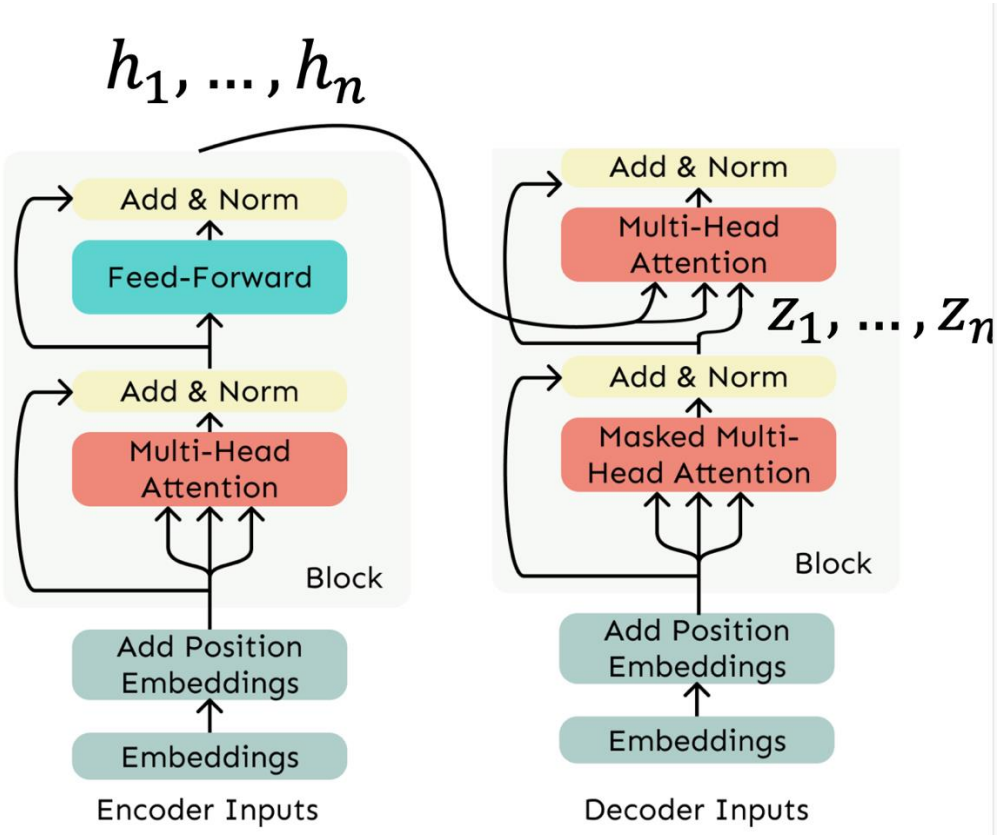






Cross-attention

- We saw that self-attention is when keys, queries, and values come from the same source.
- Let $h_1, \ldots, h_n$ be **output** vectors **from** the Transformer **encoder**; $x_i \in \mathbb{R}^d$
- Let $z_1, \ldots, z_n$ be input vectors from the Transformer **decoder**, $z_i \in \mathbb{R}^d$
- Then keys and values are drawn from the **encoder** (like a memory):
  - $k_i = Kh_i, v_i = Vh_i$.
- And the queries are drawn from the **decoder**, $q_i = Qz_i$.



Great Results with Transformers

北京大学 PEKING UNIVERSITY

Next, document generation!

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

The old standard

Transformers all the way down.

Great Results with Transformers

On this popular aggregate benchmark, for example:

All top models are Transformer (and pretraining)-based.

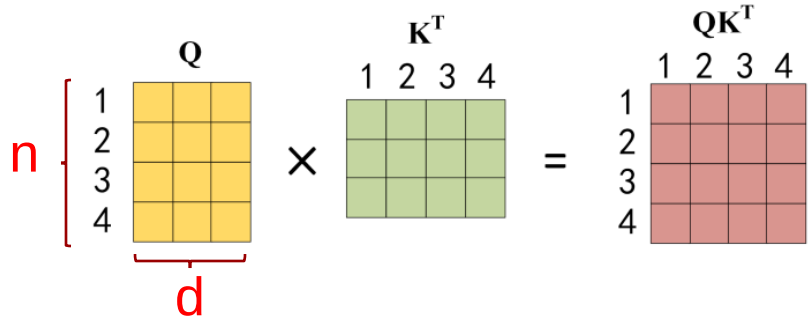
Rank	Name	Model	URL	Score
1	DeBERTa Team - Microsoft	DeBERTa / TuringNLRv4		90.8
2	HFL iFLYTEK	MacALBERT + DKM		90.7
3	Alibaba DAMO NLP	StructBERT + TAPT		90.6
4	PING-AN Omni-Sinitic	ALBERT + DAAF + NAS		90.6
5	ERNIE Team - Baidu	ERNIE		90.4
6	T5 Team - Google	T5		90.3

Quadratic computation as a function of sequence length

- **Quadratic compute in self-attention (today)**:
  - Computing all pairs of interactions means our computation grows **quadratically** with the sequence length!
  - For recurrent models, it only grew linearly!

Quadratic computation as a function of sequence length

- One of the benefits of self-attention over recurrence was that it's highly parallelizable.
- However, its total number of operations grows as $O(n^2 d)$, where n is the sequence length, and d is the dimensionality.



- Think of d as around $\mathbf{1,000}$ (though for large language models it's much larger!).
  - So, for a single (shortish) sentence, $n \leq 30$; $n^2 \leq \mathbf{900}$.
  - In practice, we set a bound like $n = 512$.
  - **But what if we'd like $n \geq 50,000$?** For example, to work on long documents?