Lecture 13: Transformers

Claire Cardie, Tanya Goyal

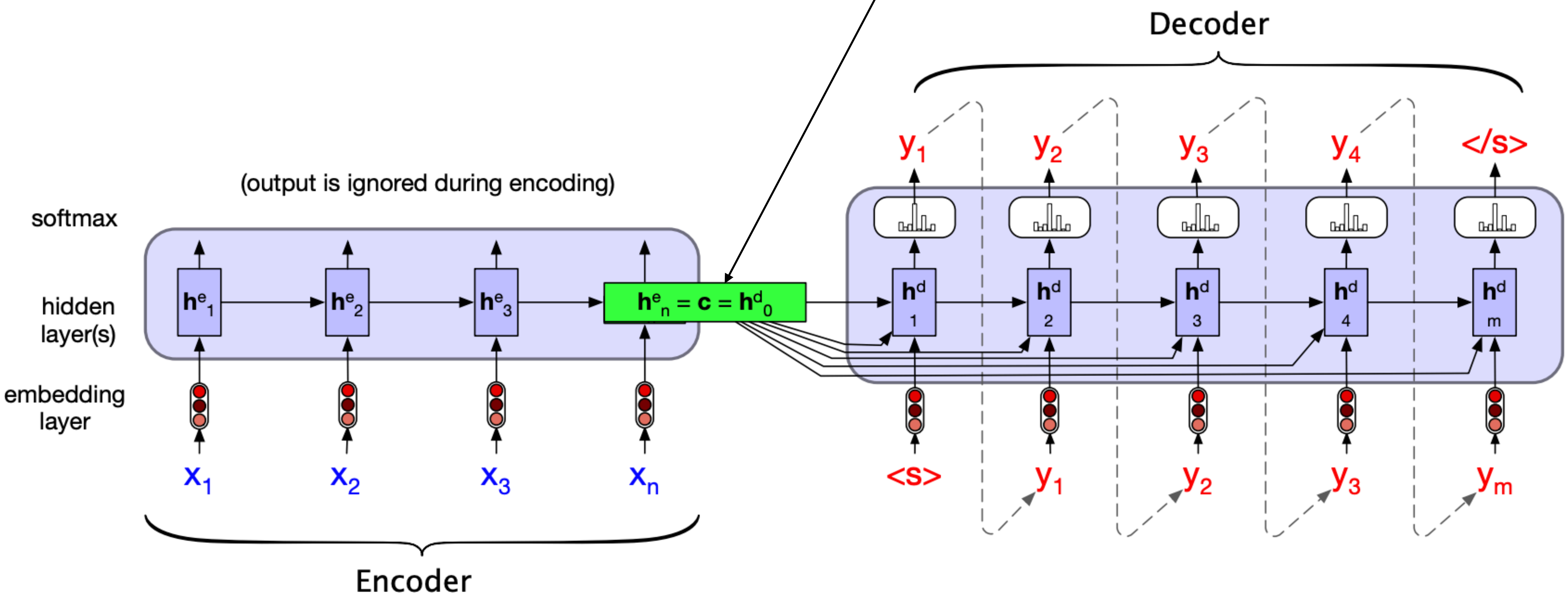
CS 4740 (and crosslists): Introduction to Natural Language Processing

Today

- Recap: Attention in RNNs
- Transformers
  - Self-Attention
    - Single-head
    - Multi-head
  - Position Embedding

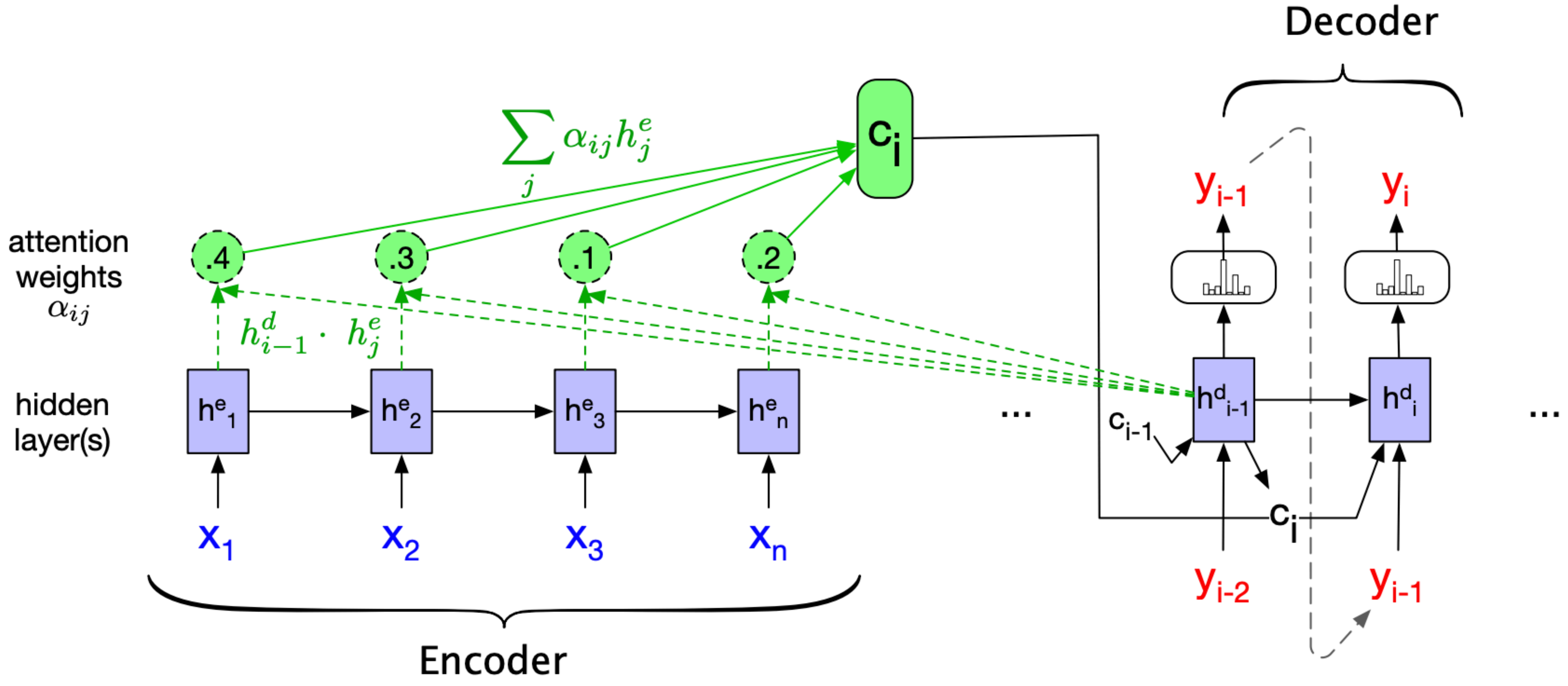
Recap: motivation for attention in the enc/dec framework

If we knew a single relevant encoder state h_t^{enc} to use for our particular decoding step, we could use that instead of a $\boxed{\text{fixed } c^{\text{enc}}}$.



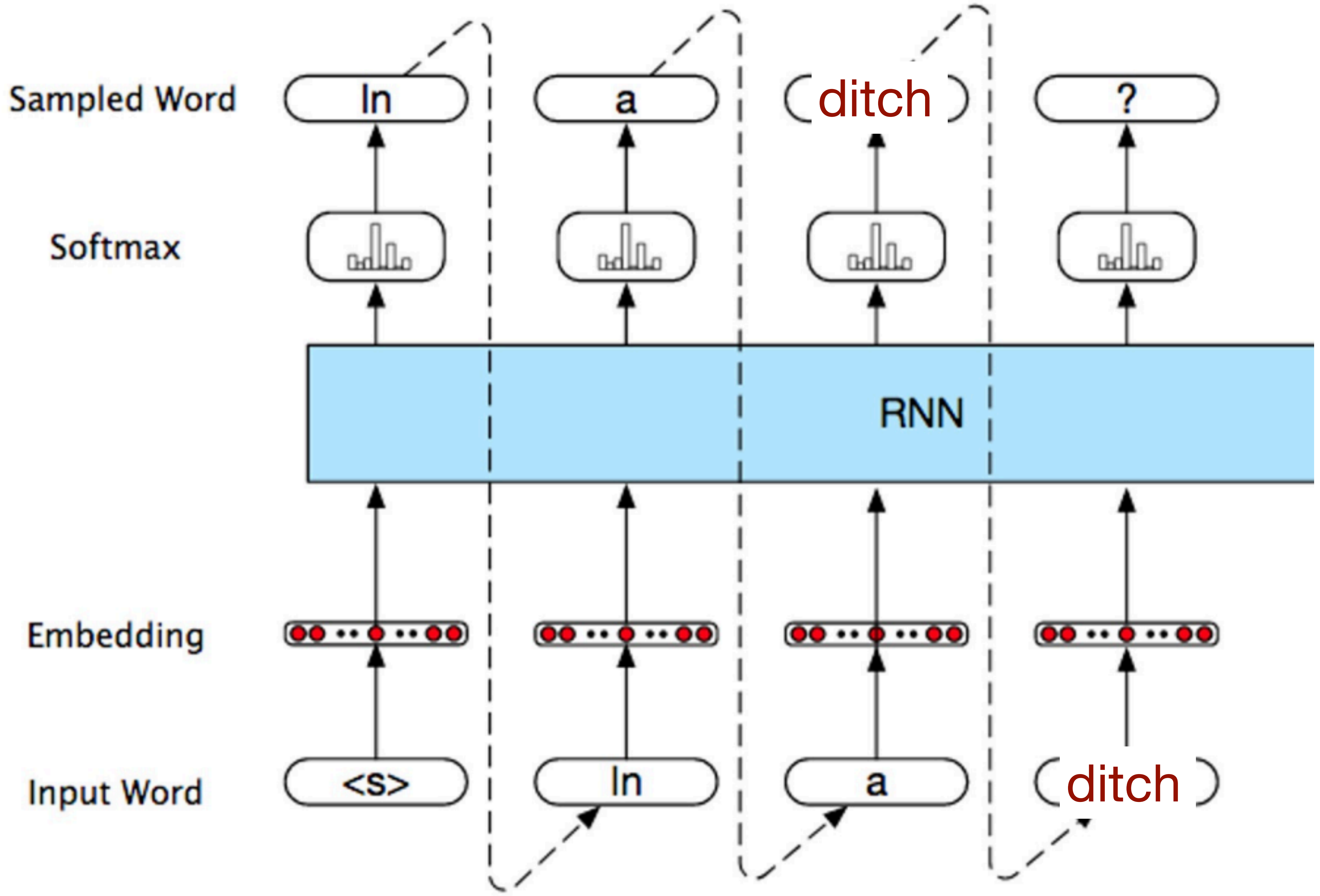
Recap: Attention: allow all enc. hidden states to participate to a weighted degree

Recap: Attention allows all enc. hidden states to participate to a weighted degree



Mini quiz

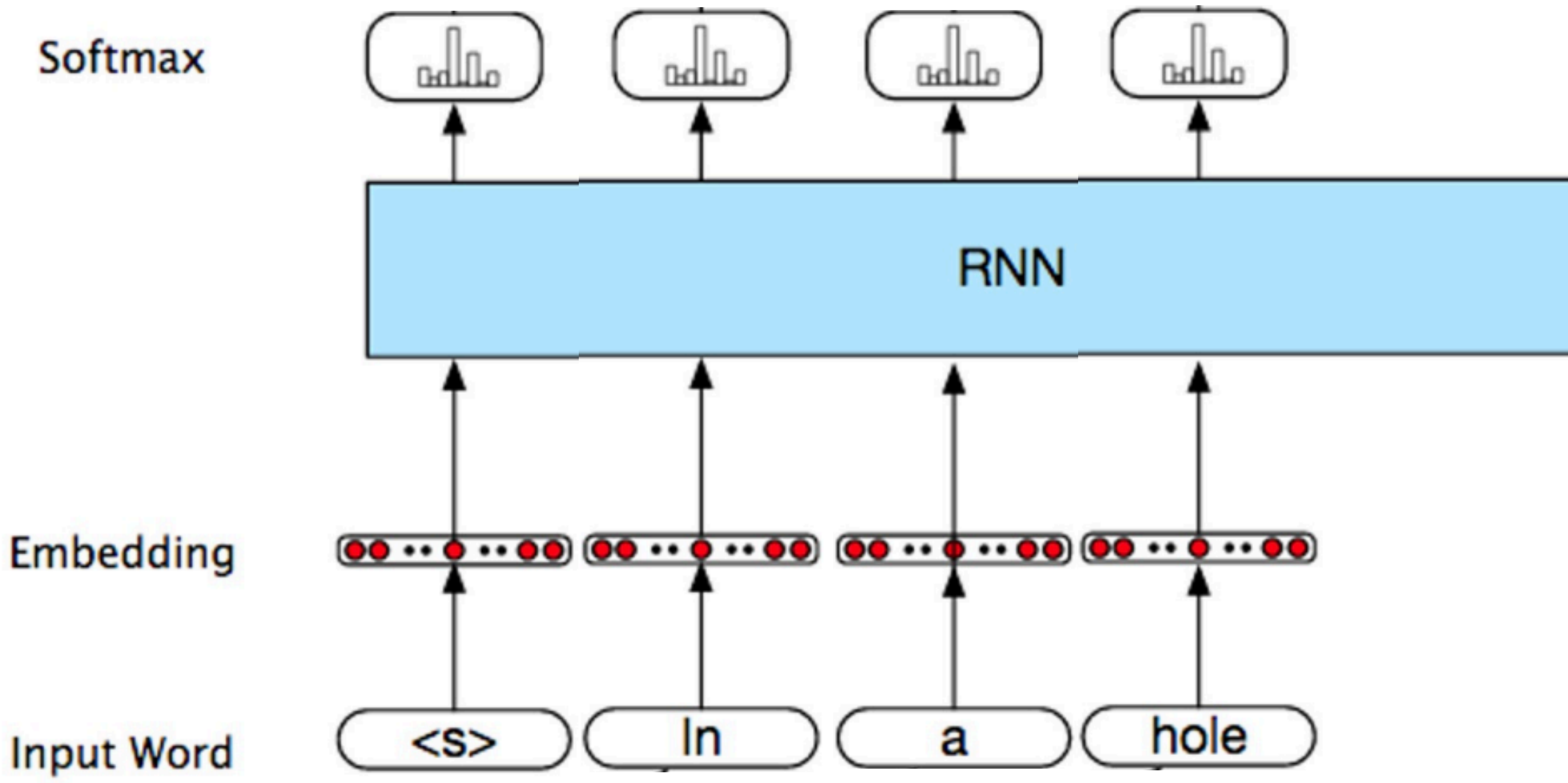
Q1: What is teacher forcing? Why do we use it?



Autoregressive Generation

Gold seq.: <s> In a hole in the wall …

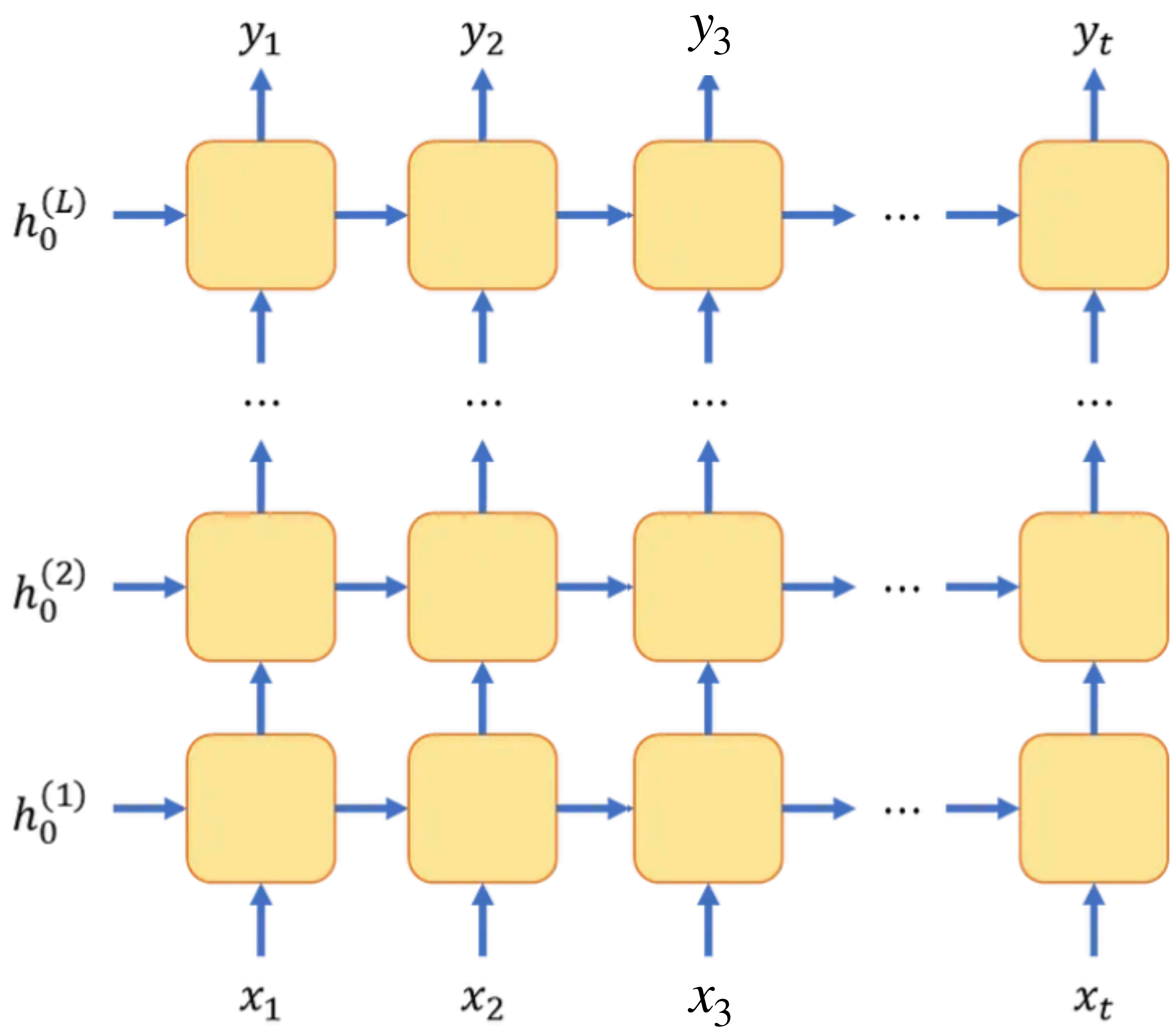
Compute loss wrt to gold seq.



Input is always the gold sequence..

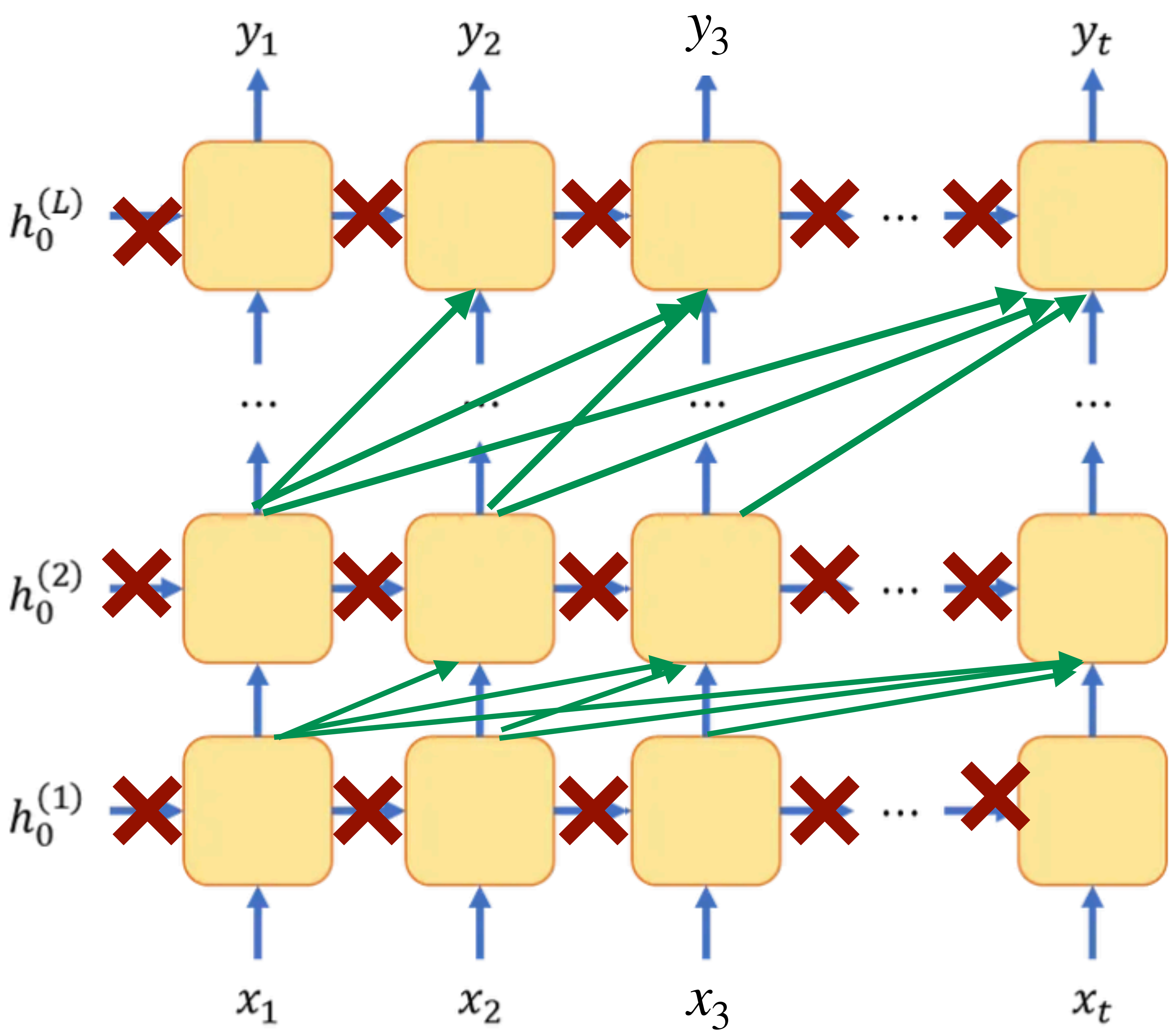
<u>Teacher Forcing during training</u>

With attention, do we need recurrence? Maybe not!



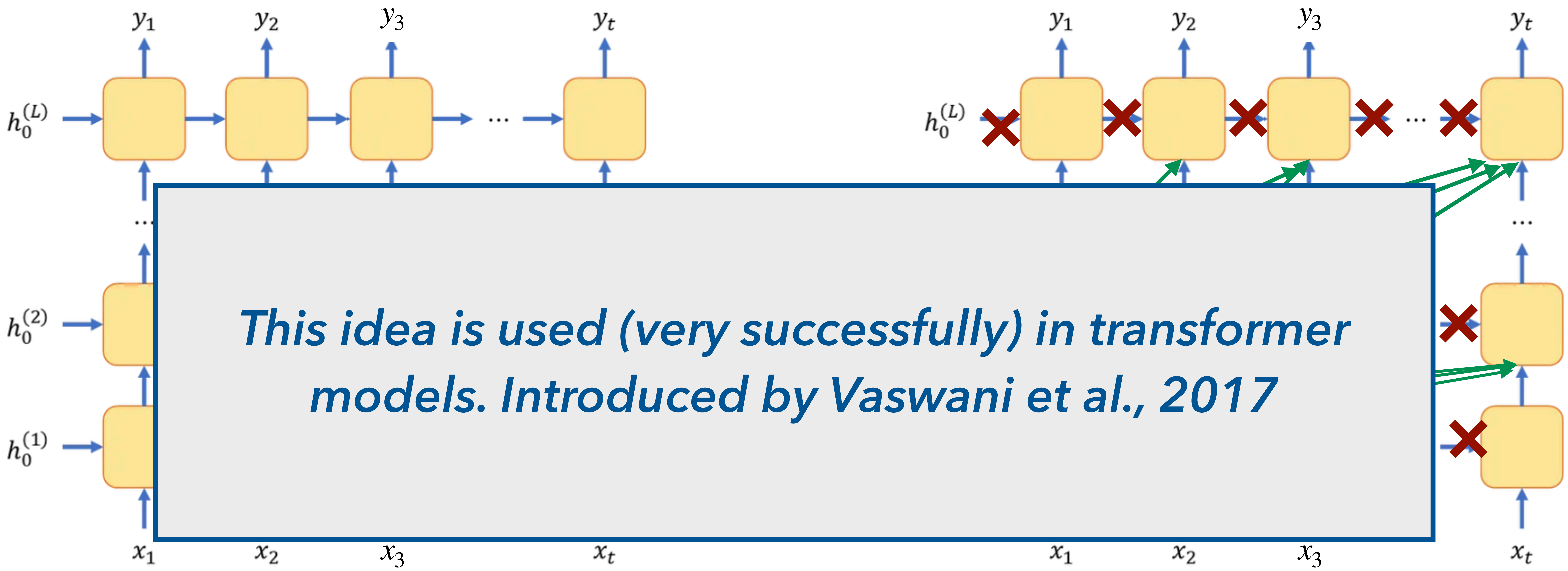
<u>Multi-layer RNN</u>

Computation at time *i* takes into account the computation (hidden values) from time *i-1*.



Above: the computation at time i just looks at the outputs from the previous layer.
Computations at the same layer are parallelizable!

With attention, do we need recurrence? Maybe not!



Multi-layer RNN

Computation at time *i* takes into account the computation (hidden values) from time *i-1*.

Above: the computation at time i just looks at the outputs from the previous layer.
Computations at the same layer are parallelizable!

Why should we learn about transformers?

- Transformer (variants) are the backbone of all powerful LLMs today!

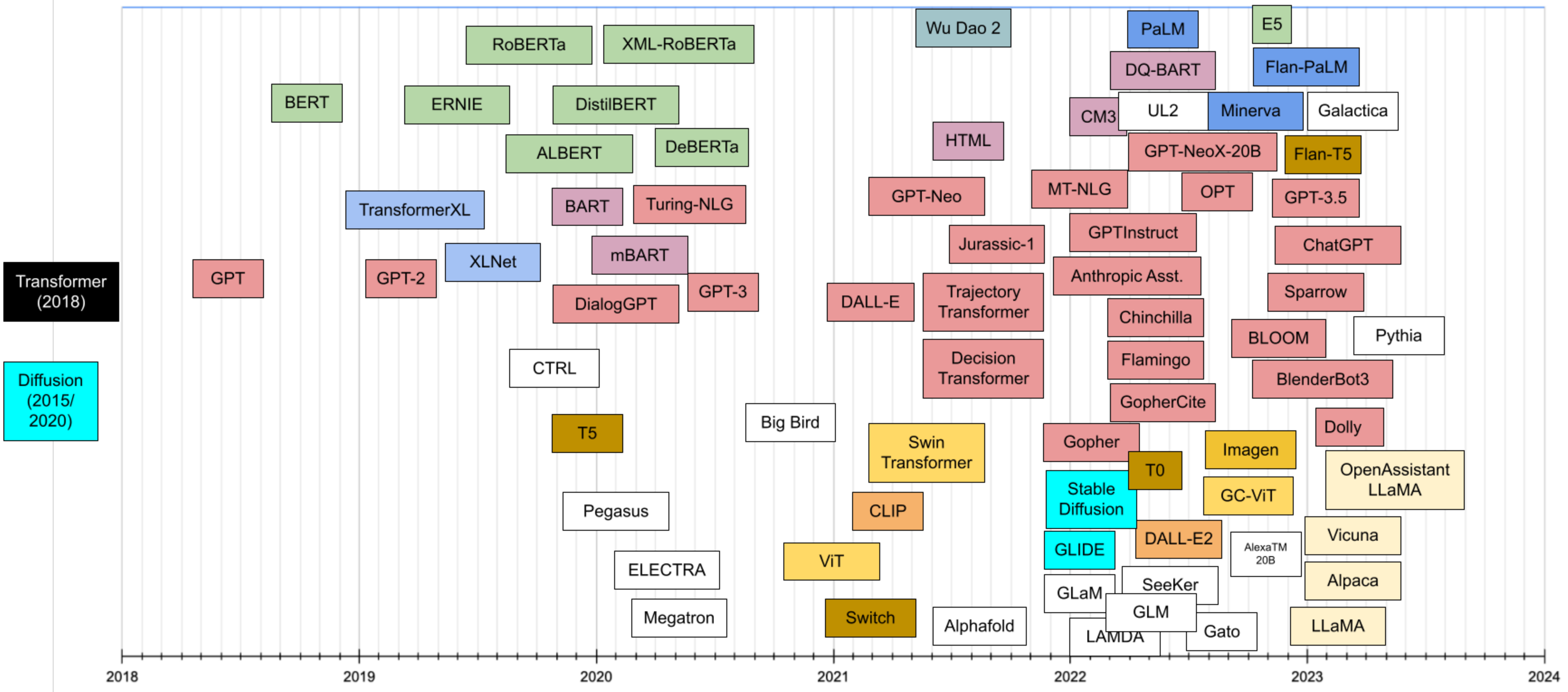
ChatGPT

Claude
Sonnet 3.5

deepseek

- Tons of visualizations to trace influence of transformers architecture:
  - Amatriain's: https://amatriain.net/blog/transformer-models-an-introduction-and-catalog-2d1e9039f376/
  - Victor Gaske's: https://ai.v-gar.de/ml/transformer/timeline/

Why should we learn about transformers?



Why should we learn about transformers?

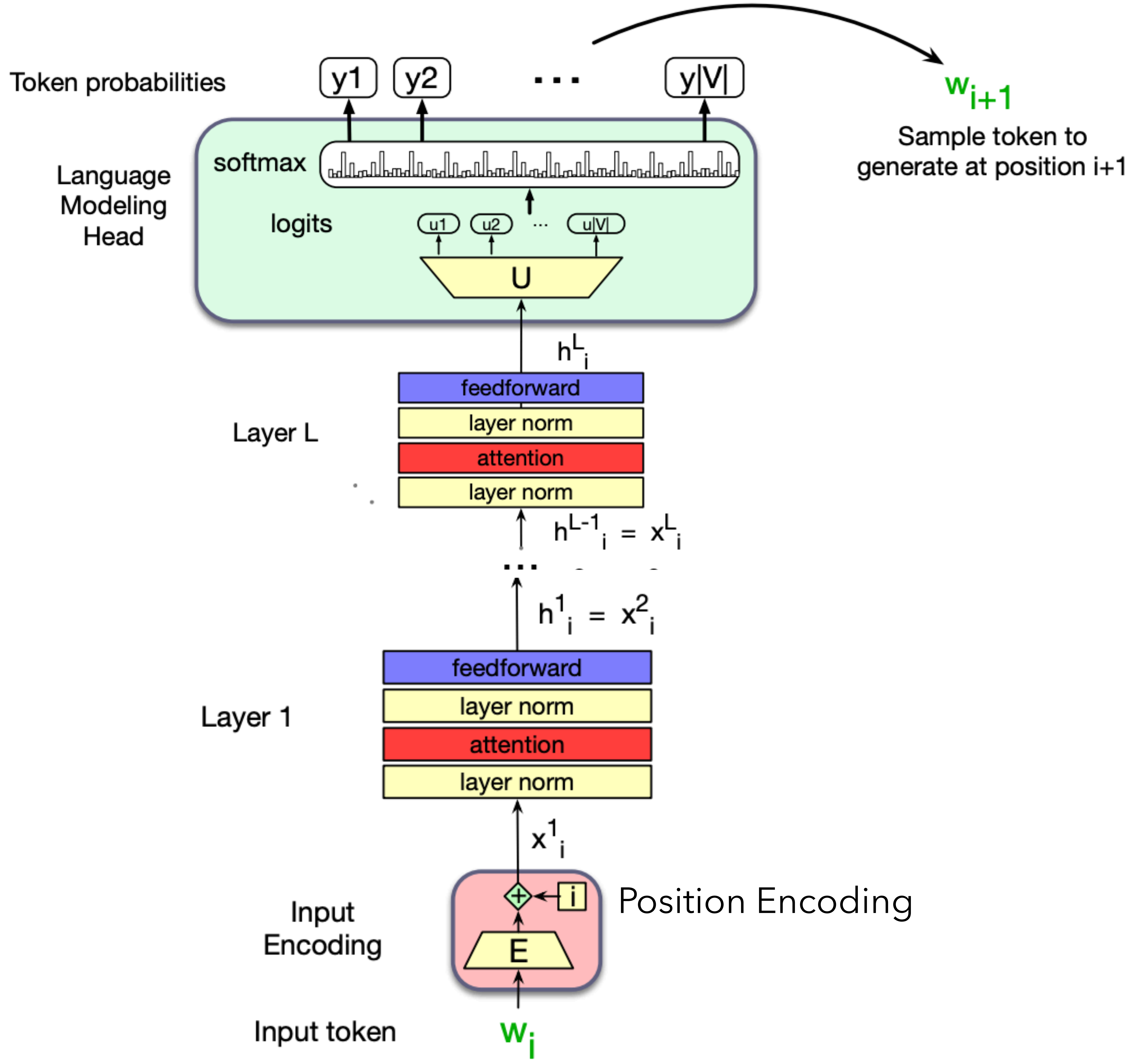
- Public Wager: https://www.isattentionallyouneed.com/

- Proposition: On January 1, 2027, a Transformer-like model will continue to hold the state-of-the-art position in most benchmarked tasks in natural language processing.

Today: Transformer Models

- Introduced in Attention Is All You Need (Vaswani et al. NeurIPS 2017)
- A purely attention-based architecture (highly parallelizable), i.e. no recurrence
- Very deep model for NLP (12 layers)
- Originally envisioned for seq2seq tasks (encoder is 6 layers, decoder is 6 layers)
- The encoder and decoder are the same “architecture” applied differently
- **We will first look at the decoder-only transformer today**

Transformer Architecture (Decoder-only)



- We will build up to this!!
- Main components of a transformer model
  - **(Multi-head) Attention**
  - Feed forward
  - Layer Norm
  - Position Encoding

Simplified Attention

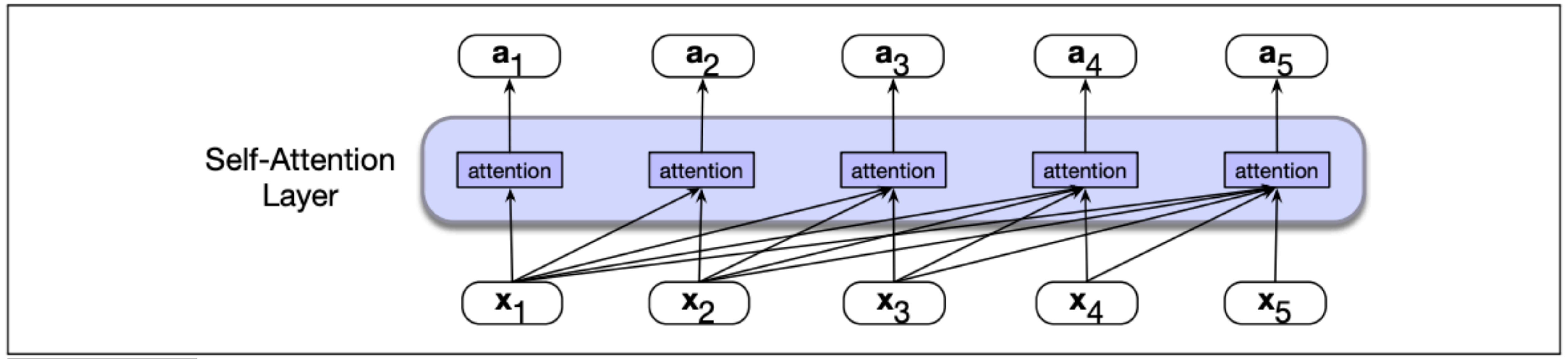
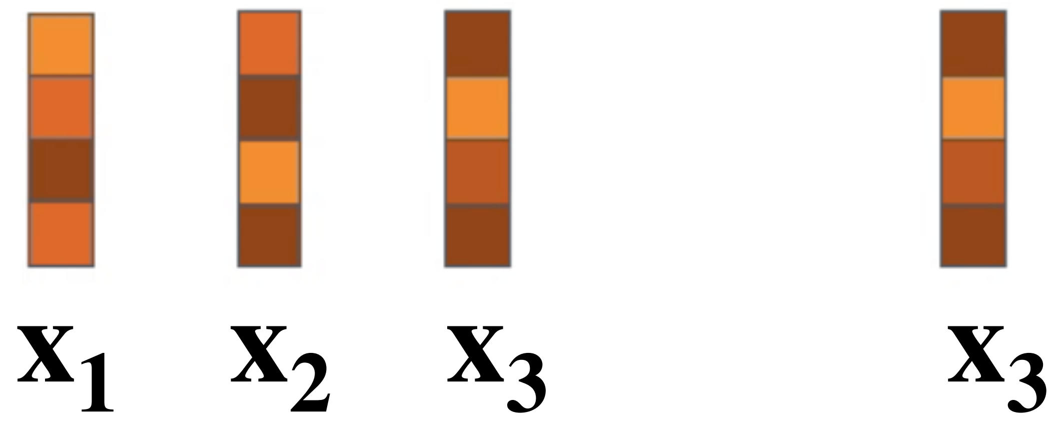


Figure 9.3 Information flow in causal self-attention. When processing each input $\mathbf{x}_i$, the model attends to all the inputs up to, and including $\mathbf{x}_i$.

- <u>Simplified attention</u> (Similar to RNNs)
  - $\alpha_{ij} = \text{softmax}\left(\text{score}\left(\mathbf{x_i}, \mathbf{x_j}\right)\right), \forall j \leq i$
  - $\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{x}_i$

<u>Computation at time step 3, ie. $\mathbf{a}_3$</u>

Step1: prepare inputs



Step2: compute scores

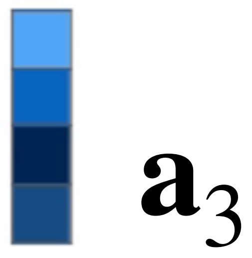
9	9	13

Step3: softmax scores (Attention weights)

0.02	0.02	0.96

Step4: multiply each vector by softmax scores

Step5: sum up the weighted vectors



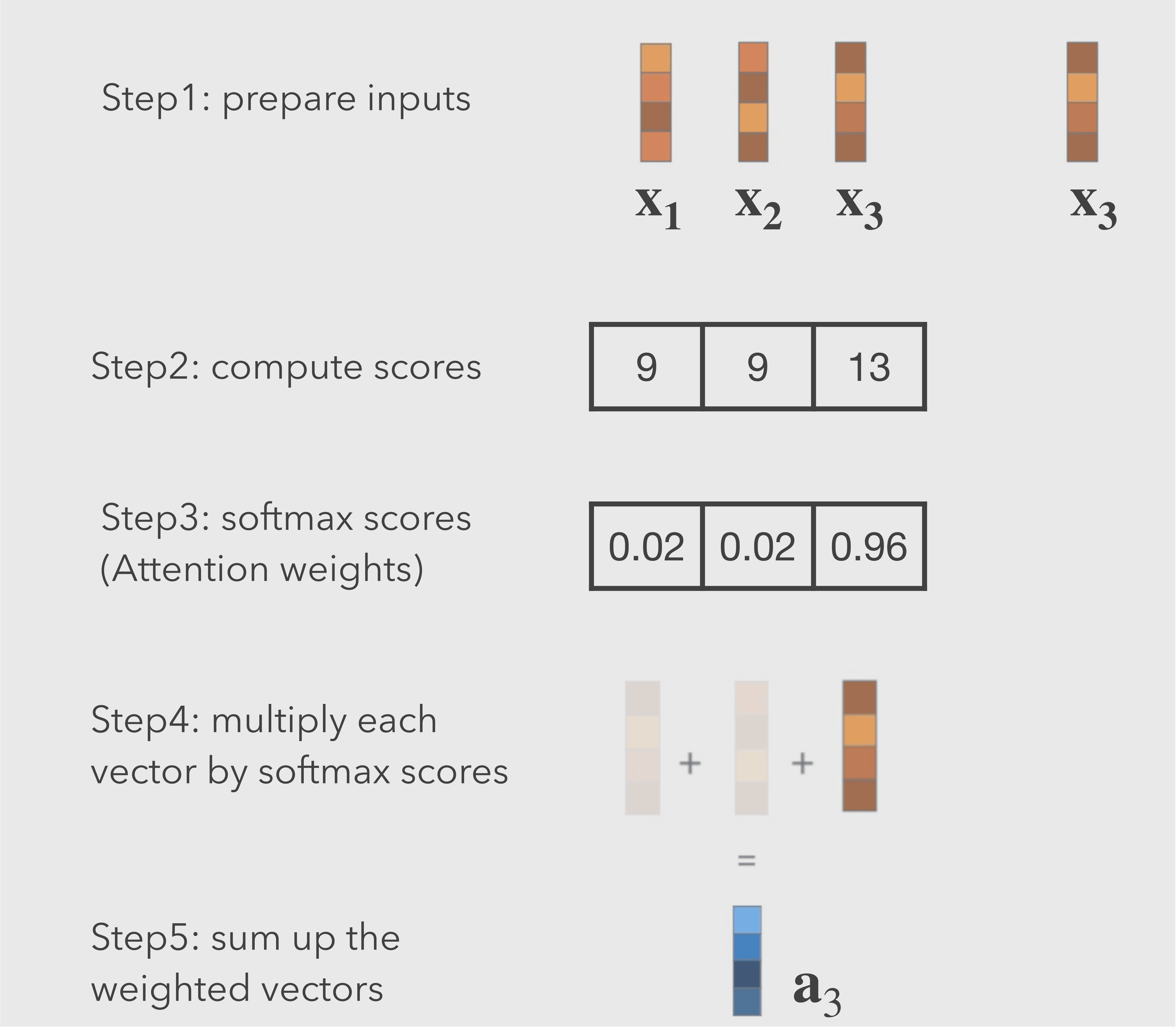
Attention in Transformer models

- Attention in Transformer architectures
  - For a given input $\mathbf{x_i}$ (could be the input at any layer of an encoder or decoder)
    create three different "roles" or "versions":

    query vector: $\mathbf{q}_i = \mathbf{x}_i\, W^Q$
    key vector: $\mathbf{k}_i = \mathbf{x}_i\, W^K$
    value vector: $\mathbf{v}_i = \mathbf{x}_i\, W^V$

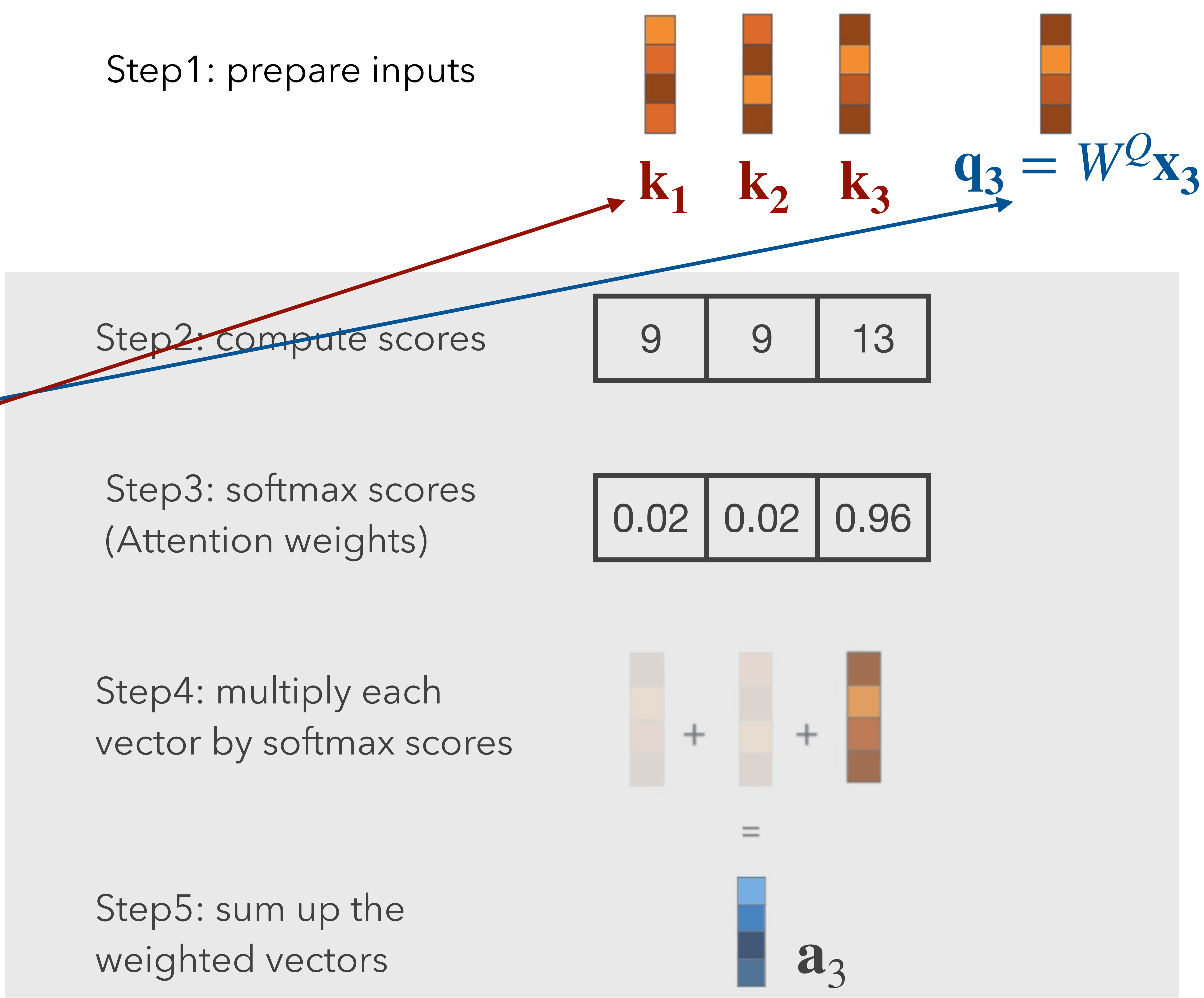
    W^Q, W^K, W^V are learned matrices

Computation at time step 3, ie. $\mathbf{a}_3$



Attention in Transformer models

Computation at time step 3, ie. $\mathbf{a}_3$



- Attention in Transformer architectures
  - For a given input $\mathbf{x_i}$ (could be the input at any layer of an encoder or decoder)

    create three different "roles" or "versions":

    query vector: $\mathbf{q}_i = \mathbf{x}_i\, W^Q$

    key vector: $\mathbf{k}_i = \mathbf{x}_i\, W^K$

    value vector: $\mathbf{v}_i = \mathbf{x}_i\, W^V$

    W^Q, W^K, W^V are learned matrices

Attention in Transformer models

- <u>Attention in Transformer architectures</u>
  - For a given input $\mathbf{x_i}$ (could be the input at any layer of an encoder or decoder)

    create three different "roles" or "versions":

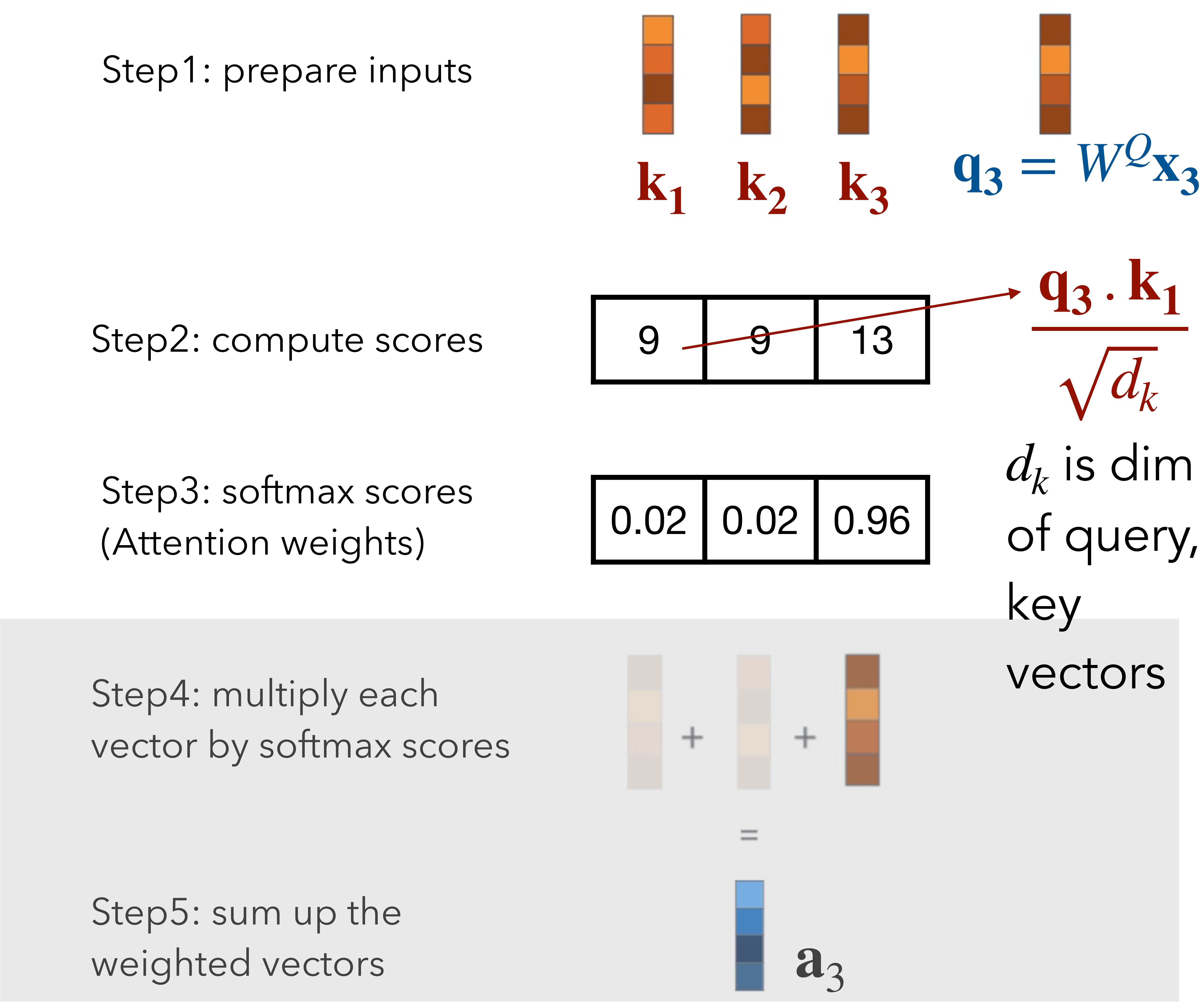
    query vector: $\mathbf{q}_i = \mathbf{x}_i\, W^Q$

    key vector: $\mathbf{k}_i = \mathbf{x}_i\, W^K$

    value vector: $\mathbf{v}_i = \mathbf{x}_i\, W^V$

    W^Q, W^K, W^V are learned matrices

<u>Computation at time step 3, ie. $\mathbf{a}_3$</u>



Attention in Transformer models

- Attention in Transformer architectures
  - For a given input $\mathbf{x_i}$ (could be the input at any layer of an encoder or decoder)

    create three different "roles" or "versions":

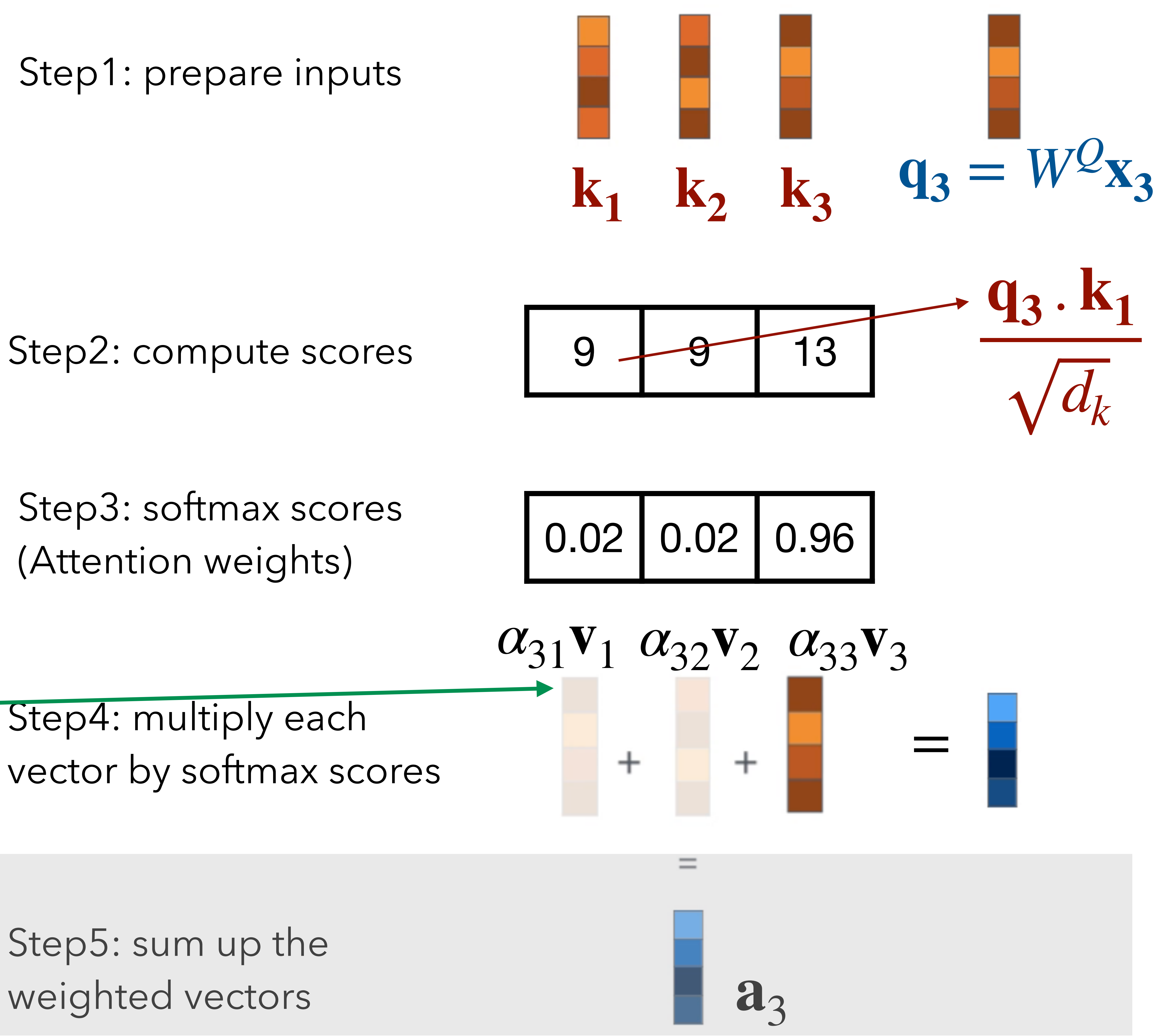
    query vector: $\mathbf{q}_i = \mathbf{x}_i\, W^Q$

    key vector: $\mathbf{k}_i = \mathbf{x}_i\, W^K$

    value vector: $\mathbf{v}_i = \mathbf{x}_i\, W^V$

    W^Q, W^K, W^V are learned matrices

Computation at time step 3, ie. $\mathbf{a}_3$



Attention in Transformer models

- Attention in Transformer architectures
  - For a given input $\mathbf{x_i}$ (could be the input at any layer of an encoder or decoder)

    create three different "roles" or "versions":

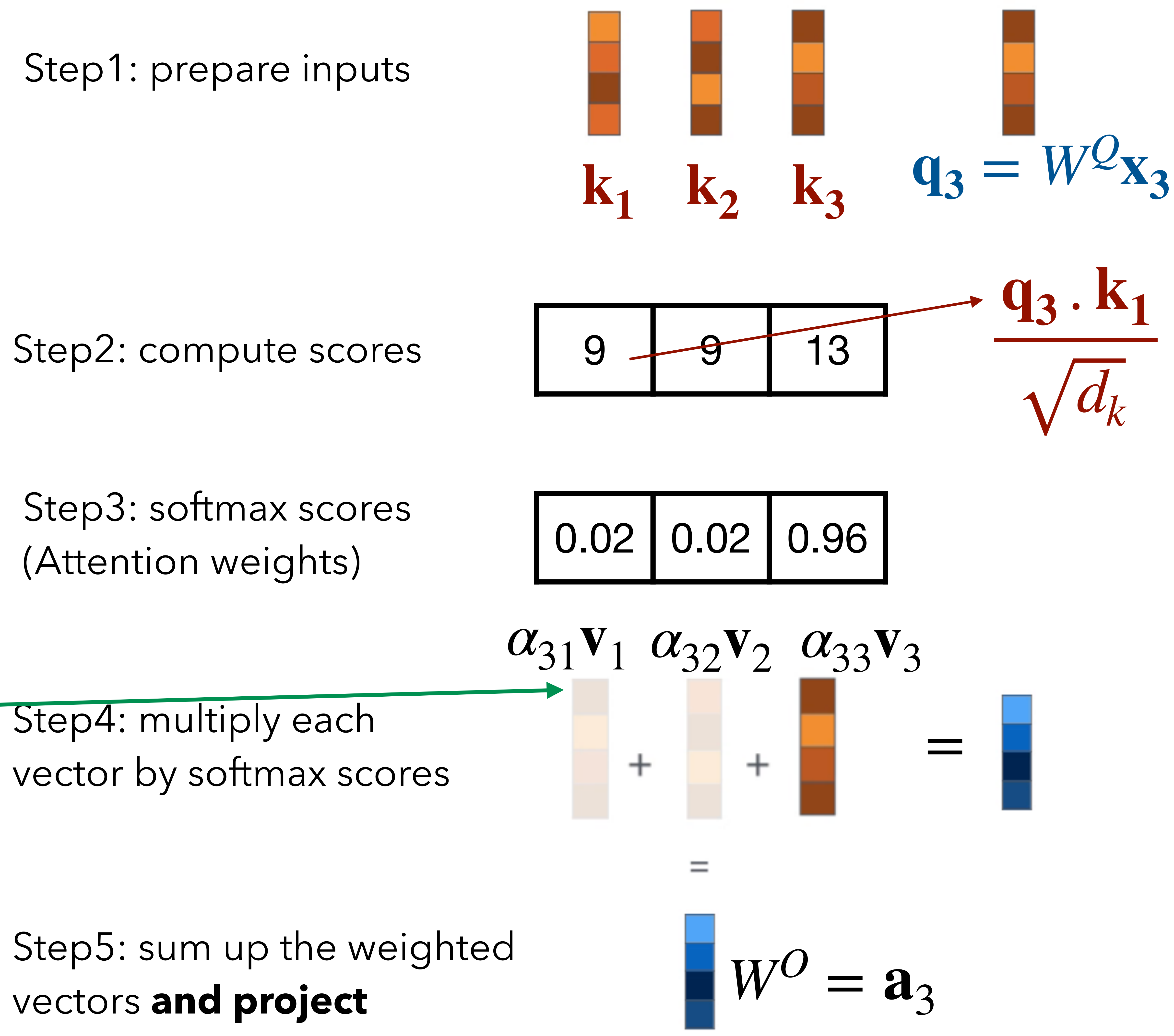
    query vector: $\mathbf{q}_i = \mathbf{x}_i W^Q$

    key vector: $\mathbf{k}_i = \mathbf{x}_i W^K$

    value vector: $\mathbf{v}_i = \mathbf{x}_i W^V$

    W^Q, W^K, W^V are learned matrices

Computation at time step 3, ie. $\mathbf{a}_3$



Attention in Transformer models

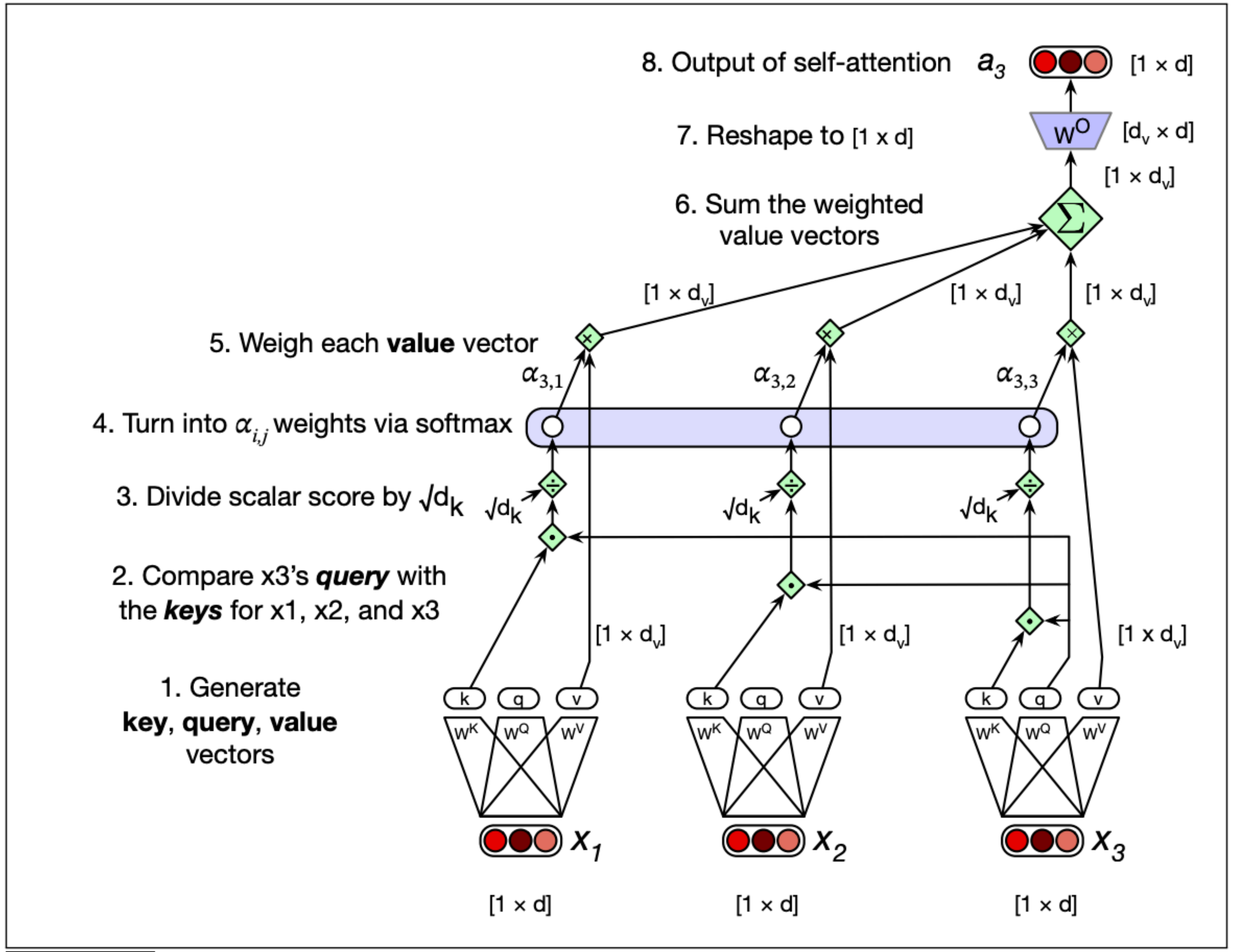


Figure 9.4 Calculating the value of $\mathbf{a}_3$, the third element of a sequence using causal (left-to-right) self-attention.

Computation at time step 3, ie. $\mathbf{a}_3$

Step1: prepare inputs

$\mathbf{k}_1$ $\mathbf{k}_2$ $\mathbf{k}_3$ $\mathbf{q}_3 = W^Q\mathbf{x}_3$

Step2: compute scores

9	9	13

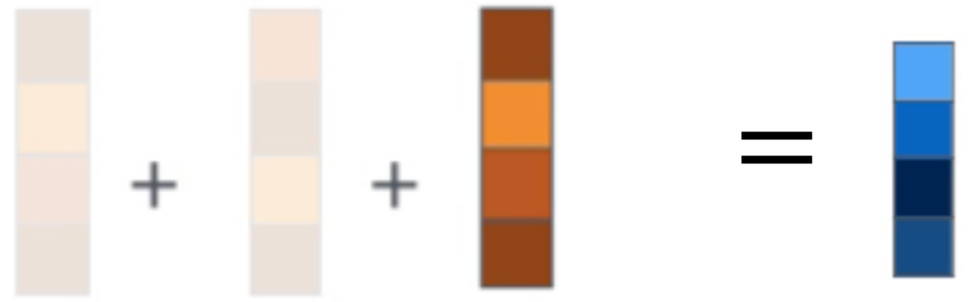
$$\frac{\mathbf{q}_3 \,.\, \mathbf{k}_1}{\sqrt{d_k}}$$

Step3: softmax scores (Attention weights)

0.02	0.02	0.96

Step4: multiply each vector by softmax scores

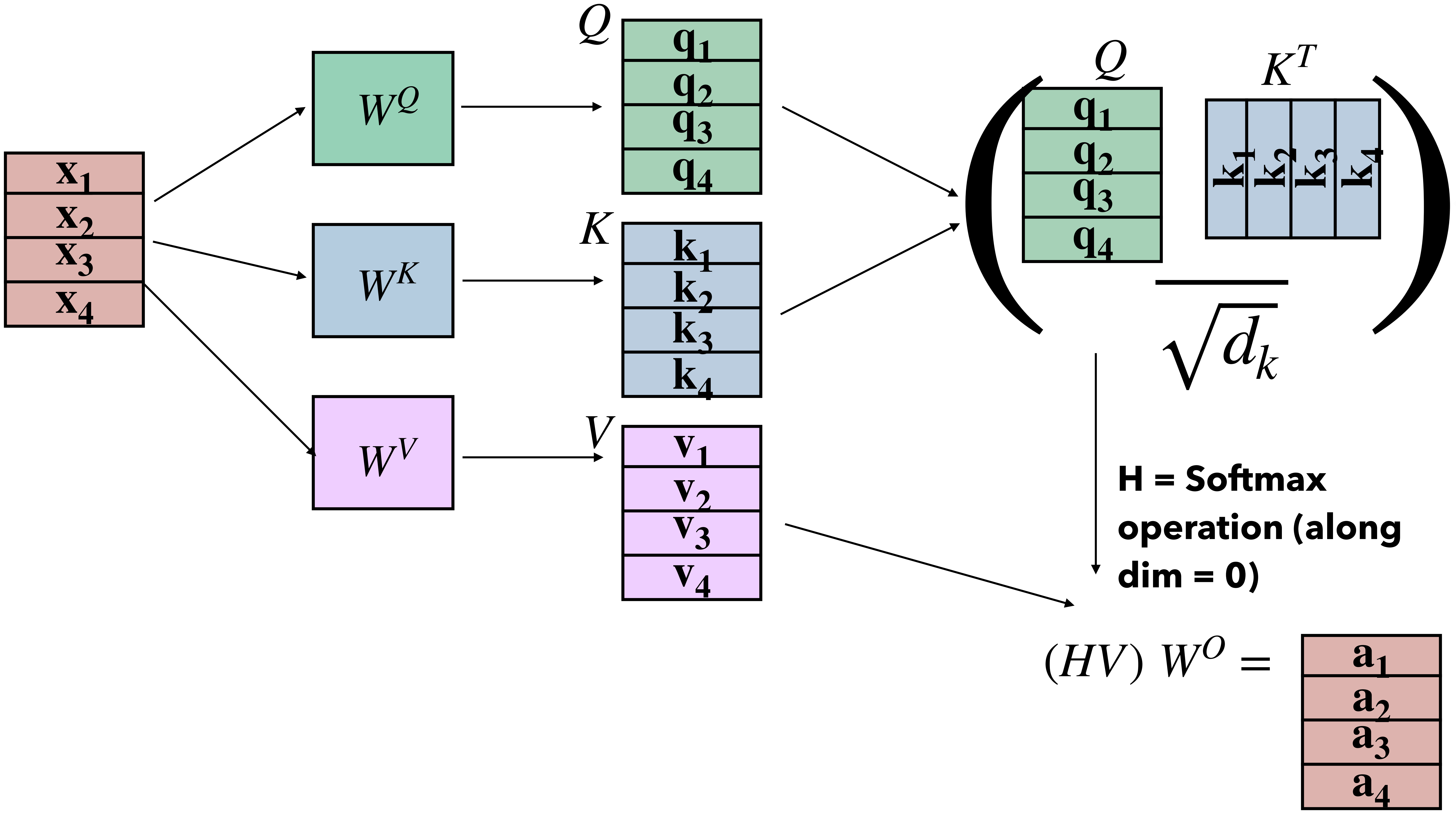
$\alpha_{31}\mathbf{v}_1$ $\alpha_{32}\mathbf{v}_2$ $\alpha_{33}\mathbf{v}_3$



Step5: sum up the weighted vectors **and project**

$W^O = \mathbf{a}_3$

Attention Computation (matrix form)



$$\oplus \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Causal mask so that inputs at time i only attend to previous inputs (Aside: in transformer encoders, inputs at time i attend to all other encoder inputs)

Multi-headed Attention

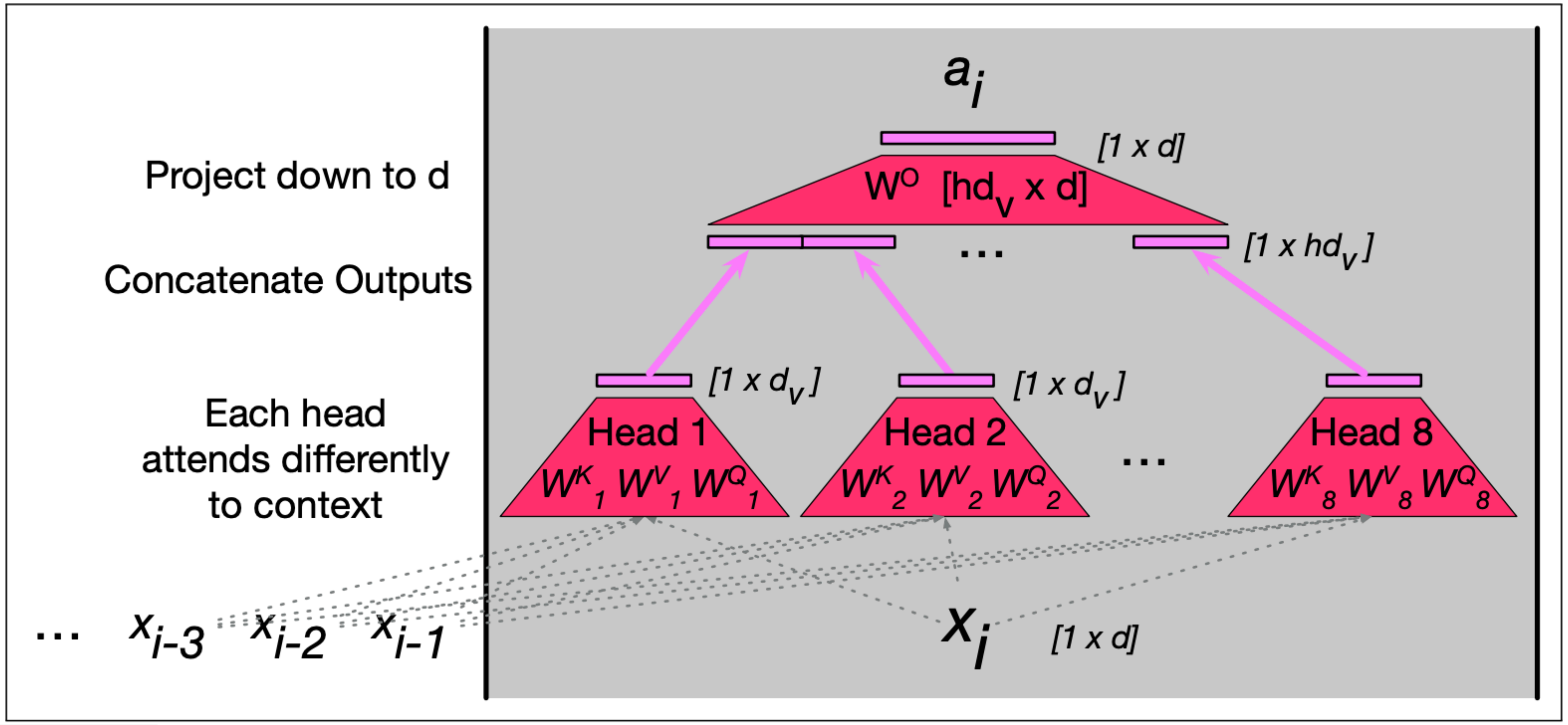
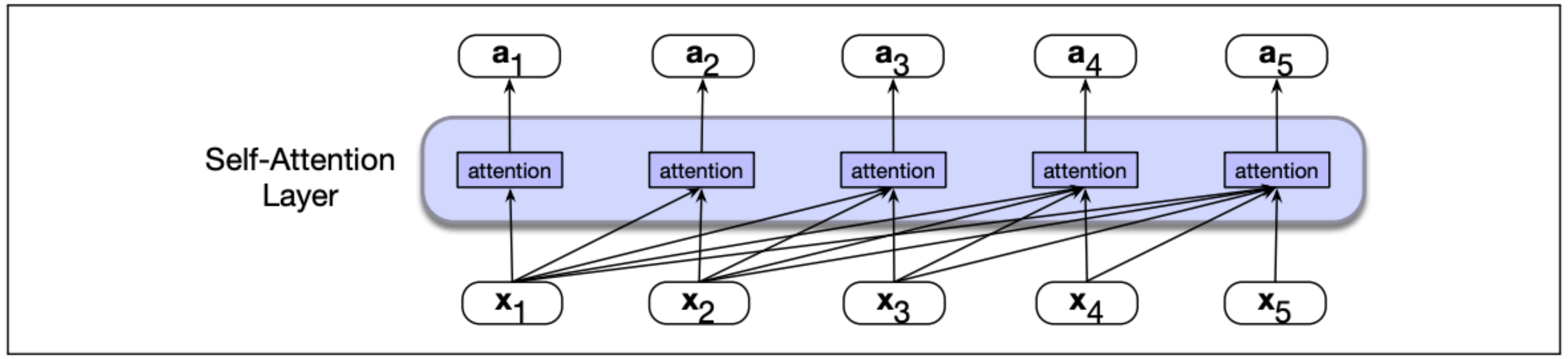


Figure 9.5 The multi-head attention computation for input $\mathbf{x}_i$, producing output $\mathbf{a}_i$. A multi-head attention layer has A heads, each with its own key, query and value weight matrices. The outputs from each of the heads are concatenated and then projected down to d, thus producing an output of the same size as the input.

- Multiple heads –> multiple ``independent'' projections (keys, queries, values) for each input.
- Each head has different W^Q, W^K, W^V matrices

- Different heads can potentially capture different phenomenon.

Zooming out

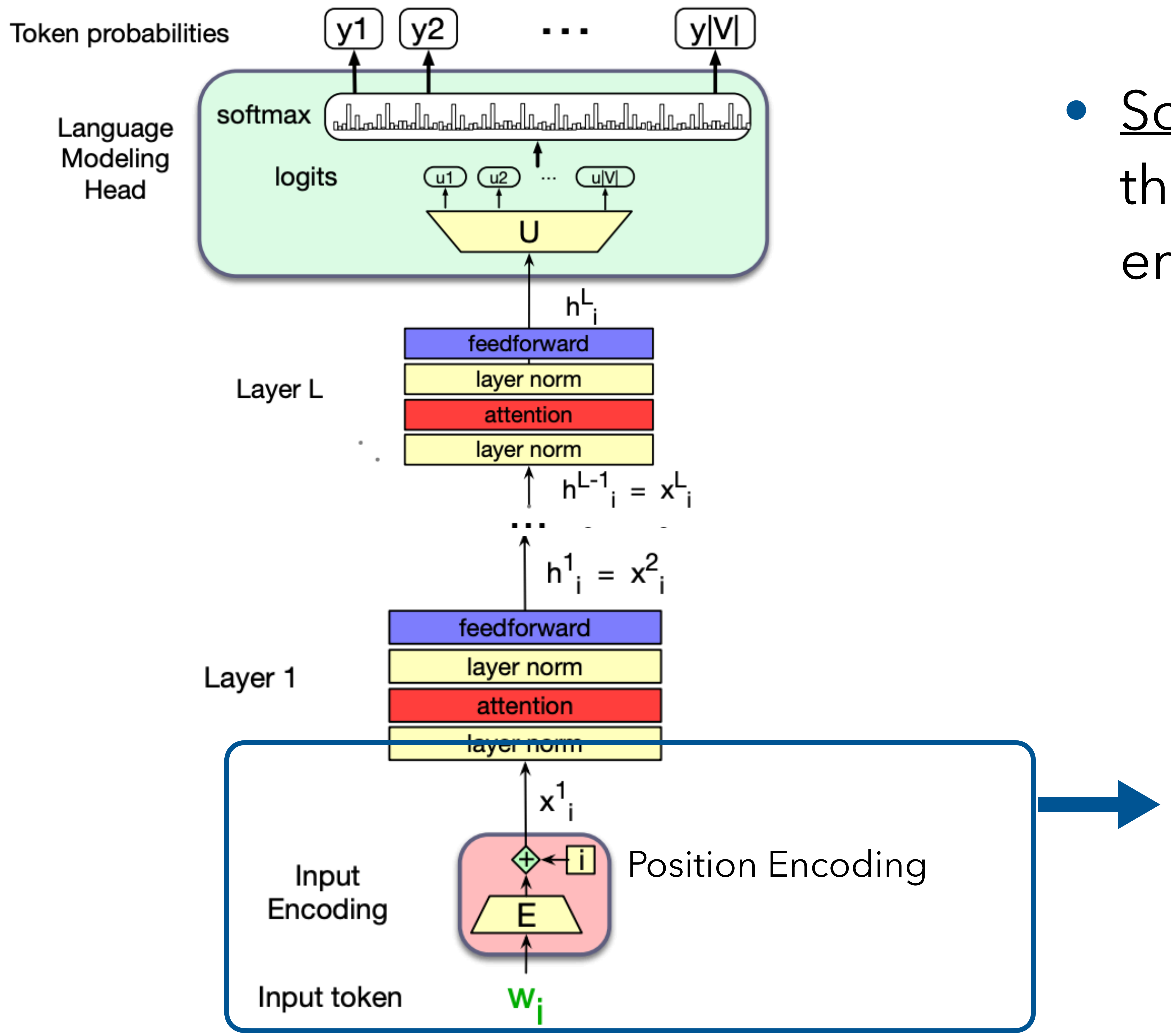


- <u>Self-attention layer</u> transformed the input $\mathbf{x_i}$ to output $\mathbf{a_i}$
- Word order information is lost!

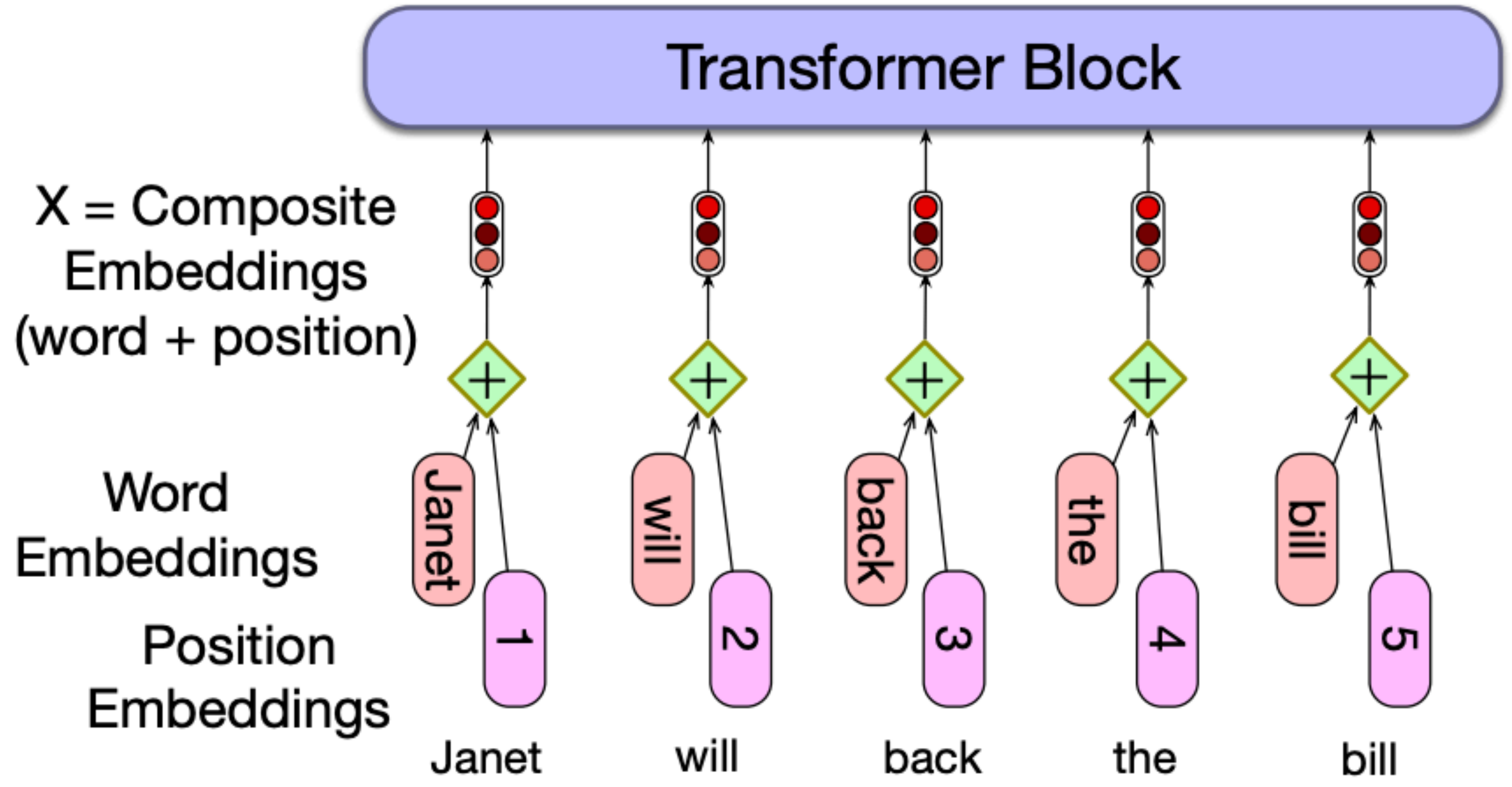
An <u>old</u> dog and a <u>young</u> boy ….

- *boy* attends to both *<u>old</u>* and *<u>young</u>*. We *<u>young</u>* to have a higher influence on *boy*'s hidden representation than *<u>old</u>*. Attention does not ensure this.
- Q: How do RNNs include this information?

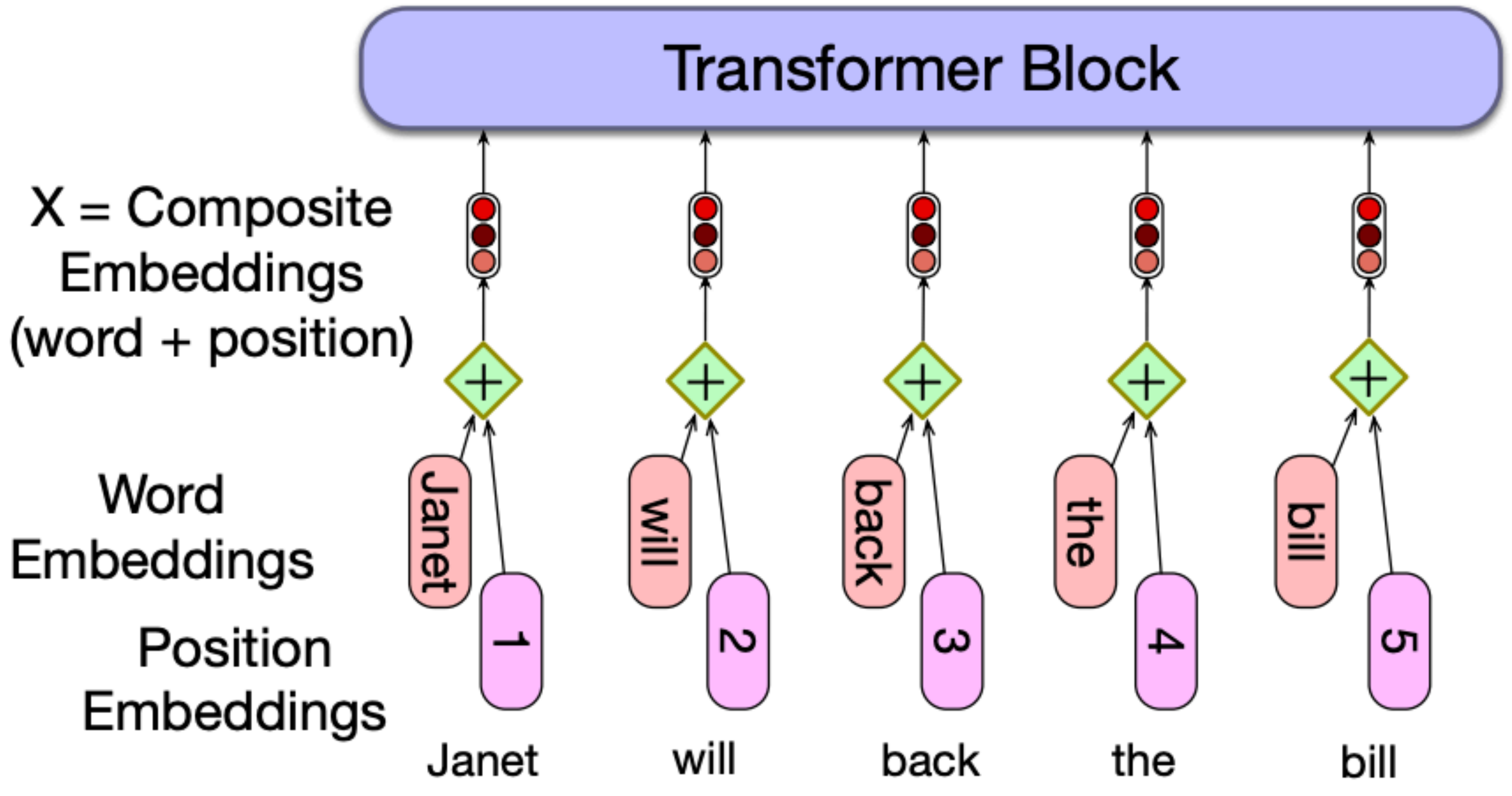
Let's go back to our transformer arch



- <u>Solution:</u> Add a "position" embedding to the word embedding to produce a new embedding of the same dimension.



Let's go back to our transformer arch

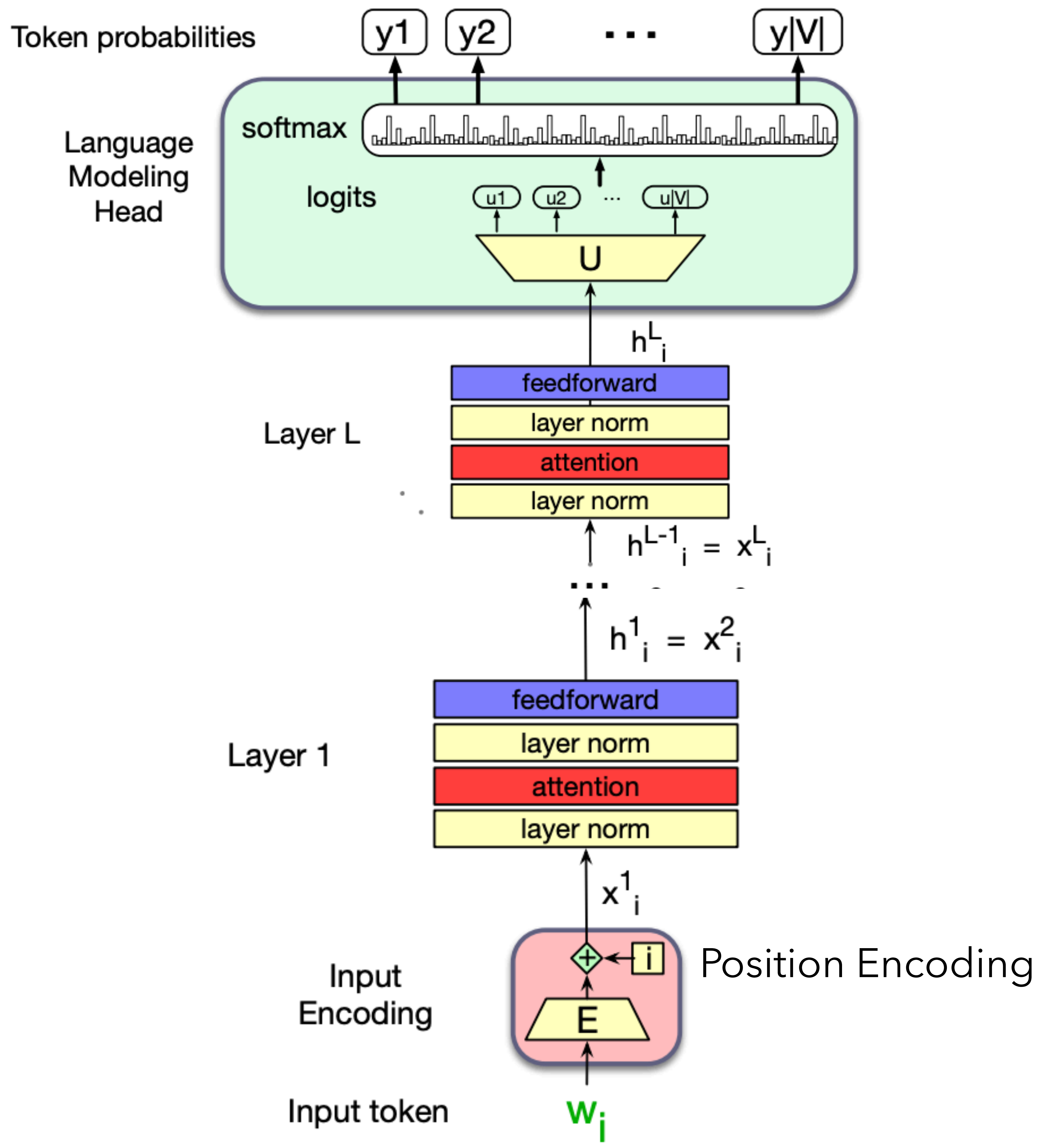


- Solution: Add a "position" embedding to the word embedding to produce a new embedding of the same dimension.

How do we get these positional embeddings?

- Assume all sequences will have length between 0 to N (say 512). Randomly initialize embeddings for each position.
- These will get trained with other transformer parameters.

Let's go back to our transformer arch



- Today:
  - Multi-head self-attention
  - Position Embeddings

- Next Class:
  - Layer Norm
  - Feedforward layer
  - Putting it all together
  - Encoder Decoder

Slide Acknowledgements

- Earlier versions of this course offerings including materials from Claire Cardie, Marten van Schijndel, Lillian Lee.
-