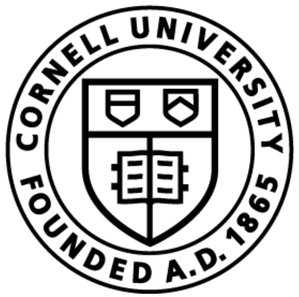
Lecture 7: Attention + Transformers



Cornell Bowers CIS
Computer Science

Tanya Goyal

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

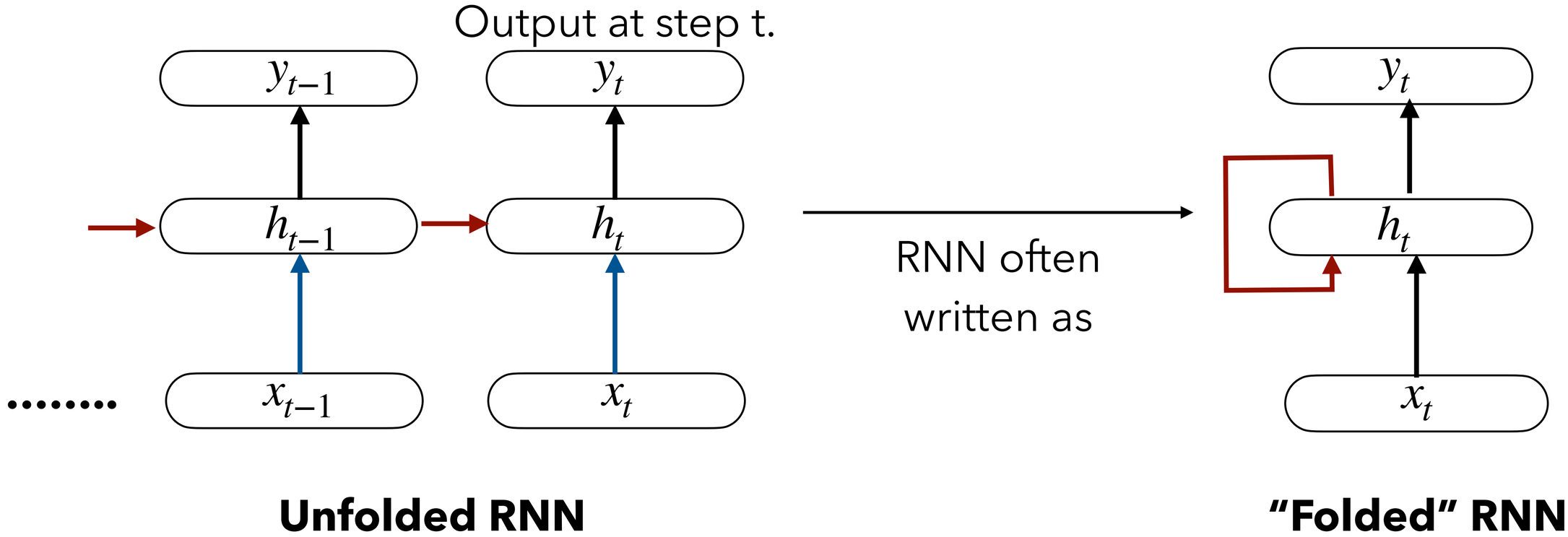
Upcoming deadlines + Today

- **Grade Releases**
  - HW1 grades released for all components. Please file regrades by **March 17, 5.00 p.m.**
  - Midterm grades released. Please file regrades by **March 23, 11.59 p.m.**
- **Upcoming Deadlines**
  - Final submission due March 20, 11.59 p.m.
    - At max 2 slip days can be used.

- **Today**
  - Recap: RNNs + Attention
  - Transformers

Recap: RNNs

- "Recurrent": A network that contains a cycle.
- Hidden layer activation depends on
  - Input/last layer **and** activation of the hidden layer at the previous time step.



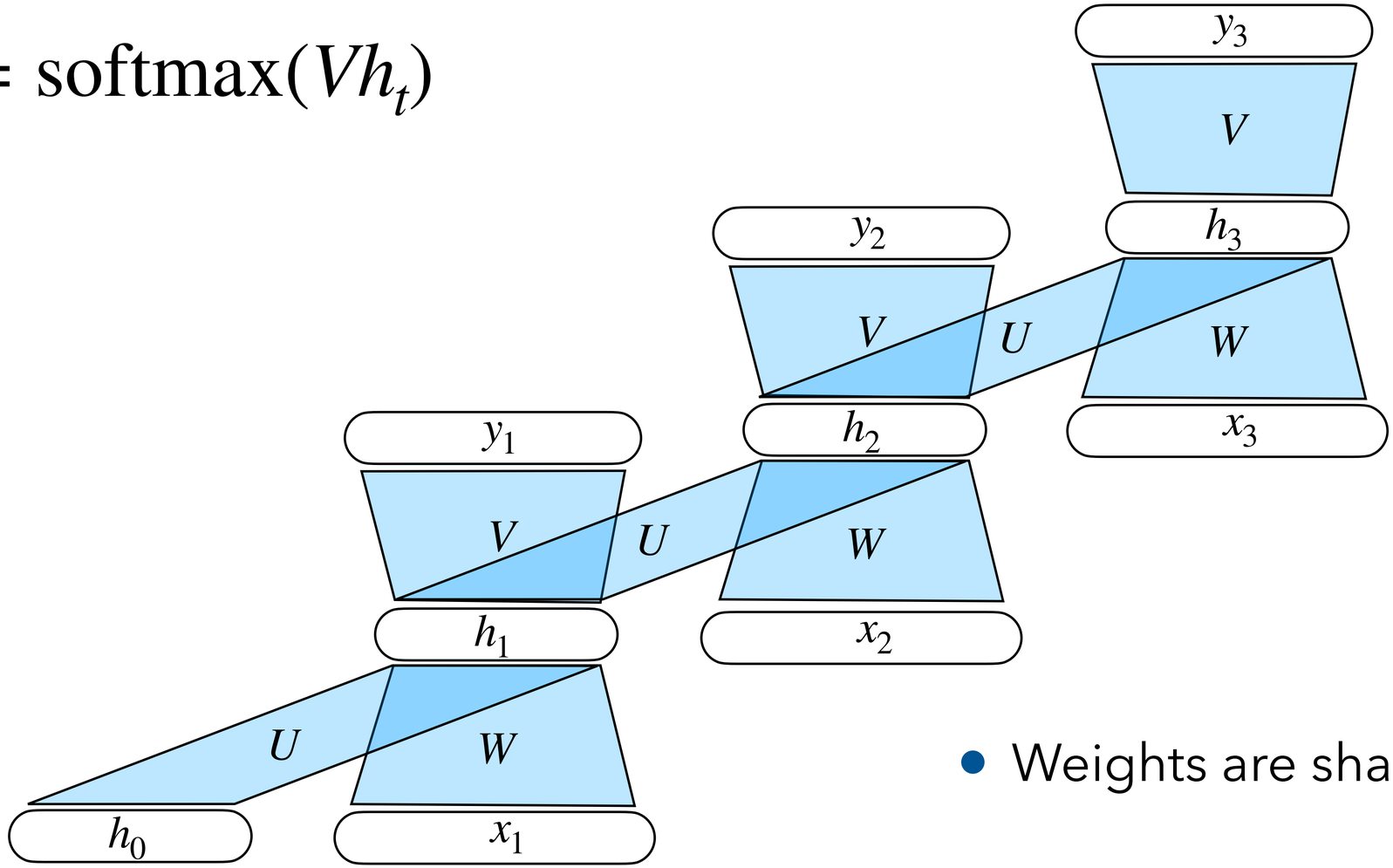
Unfolded RNN

"Folded" RNN

Recap: Unrolling the RNN in time

$$h_t = f(Uh_{t-1} + Wx_t)$$

$$y_t = \text{softmax}(Vh_t)$$



- Weights are shared across the time steps

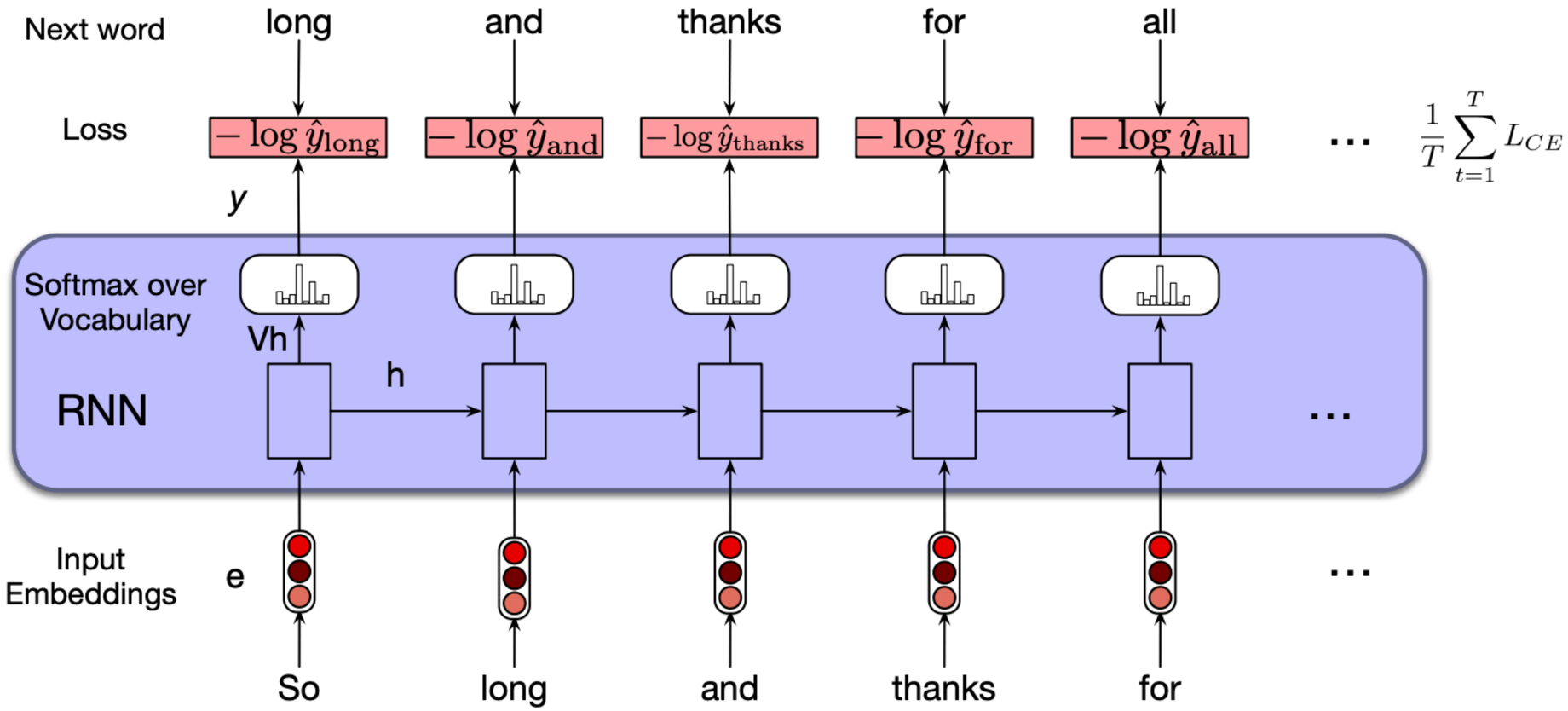
Recap: RNNs for Language Modeling/Text Generation

Self-study: design RNNs for token-level classification (e.g. HW2), sentence-level classification (e.g. sentiment classification) tasks.

Q. Design an RNN architecture for the language modeling task?

Train for next token prediction task.

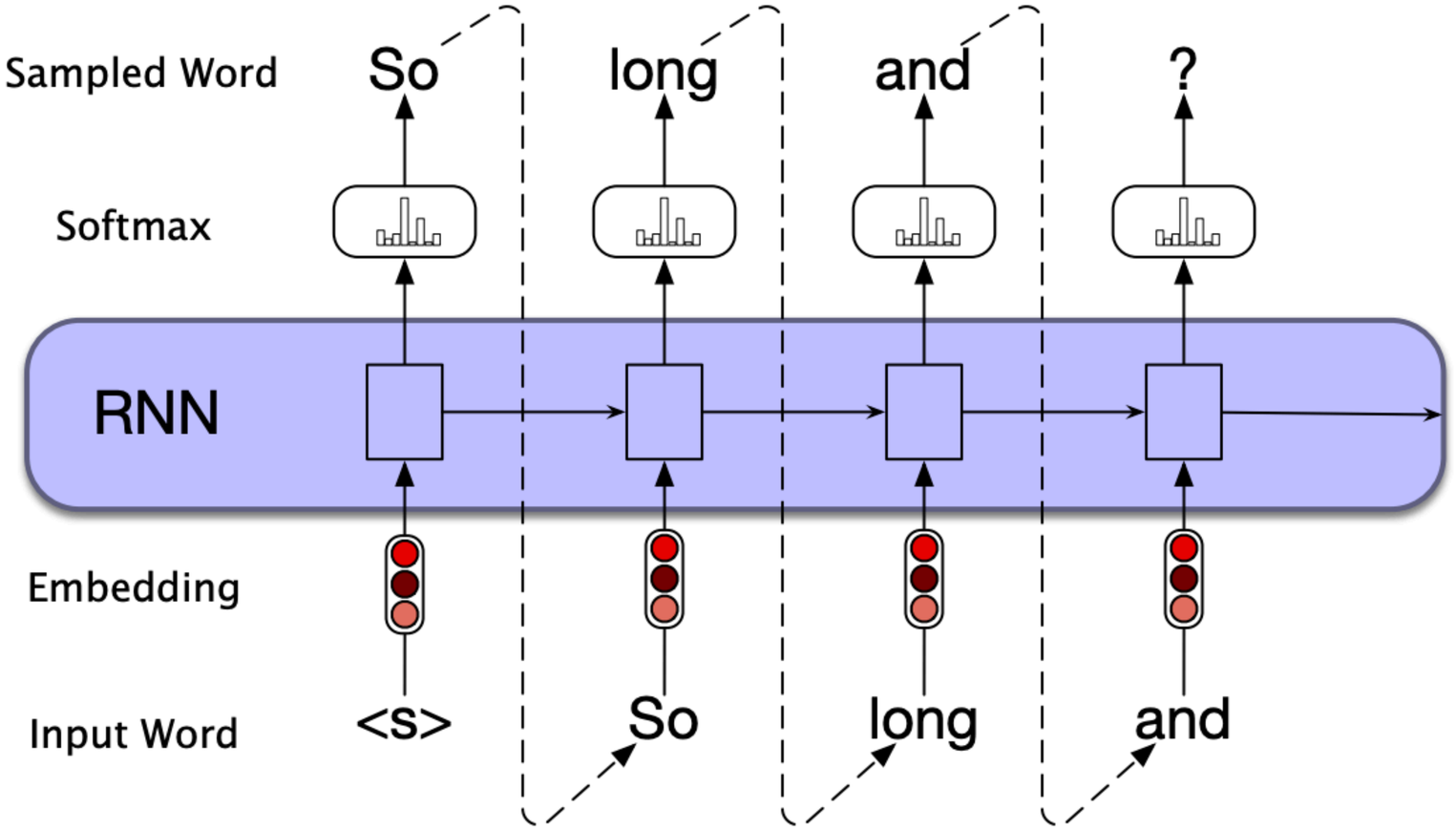
- At each time step, maximize the probability of predicting the next token in the ground truth.



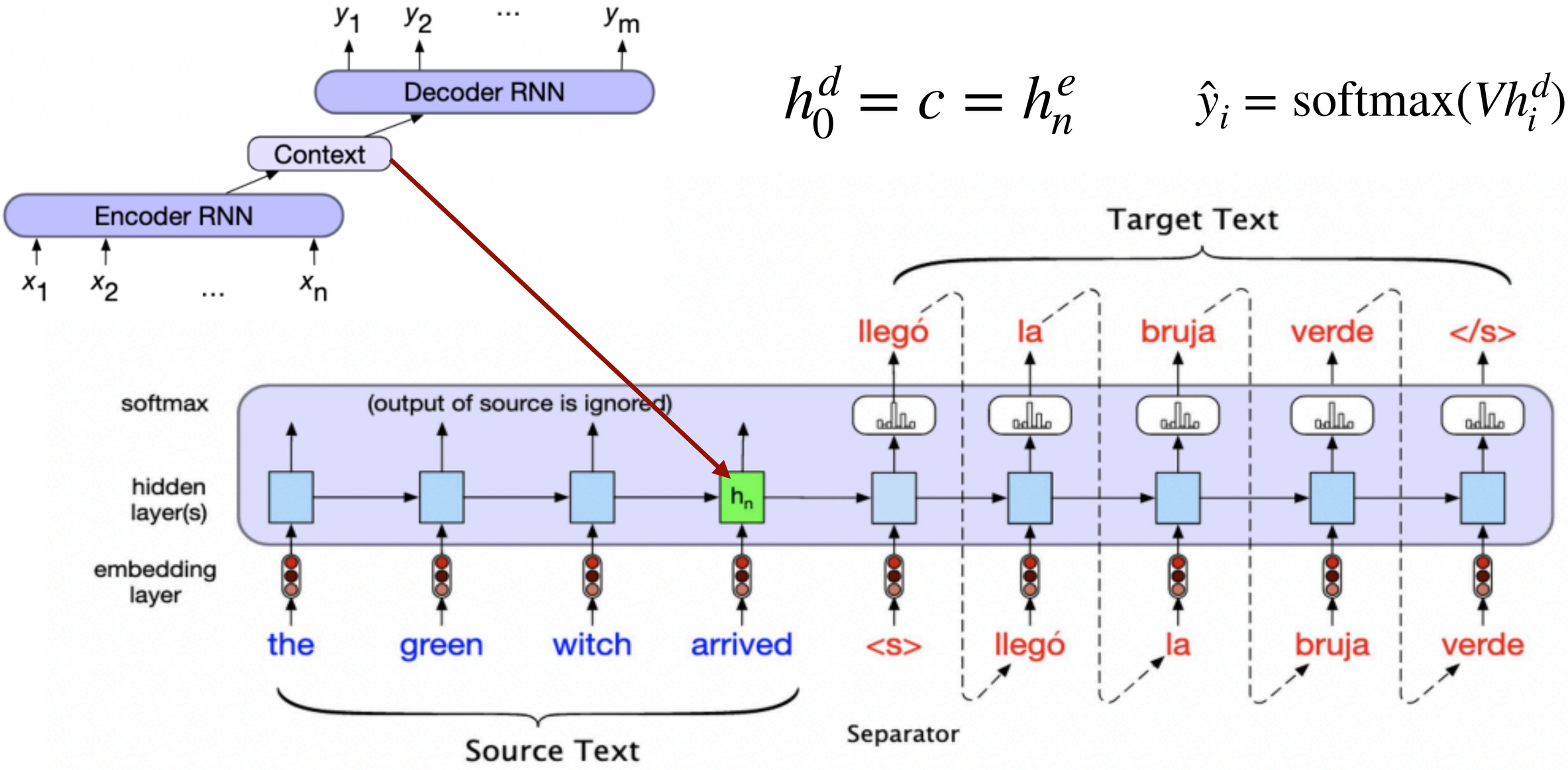
Recap: RNNs for Language Modeling/Text Generation

Q. Design an RNN architecture for the language modeling task?

Autoregressive generation:
One word is generated at each time step, conditioning on the word generated in the previous time step.



Recap: Encoder-Decoder RNNs



$$h_0^d = c = h_n^e \qquad \hat{y}_i = \text{softmax}(Vh_i^d)$$

Recap: Issue w/ simple Encoder-Decoder RNN

c is a bottleneck.

At the decoder end: Only the first time step of the decoder gets c, i.e., the encoded representation of the input text, as input for current step computation. **Can we make this available at each decoder step?**

Recap: Issue w/ simple Encoder-Decoder RNN

c is a bottleneck.

At the encoder end: The encoder's job is to create a contextualized representation of the **entire** input in one vector. Difficult to do for long sequences, where information at the beginning may not be well represented. **Can we make a different contextual representation c_i available at each decoder step, that contains relevant information for that time step?**

Attention Recap

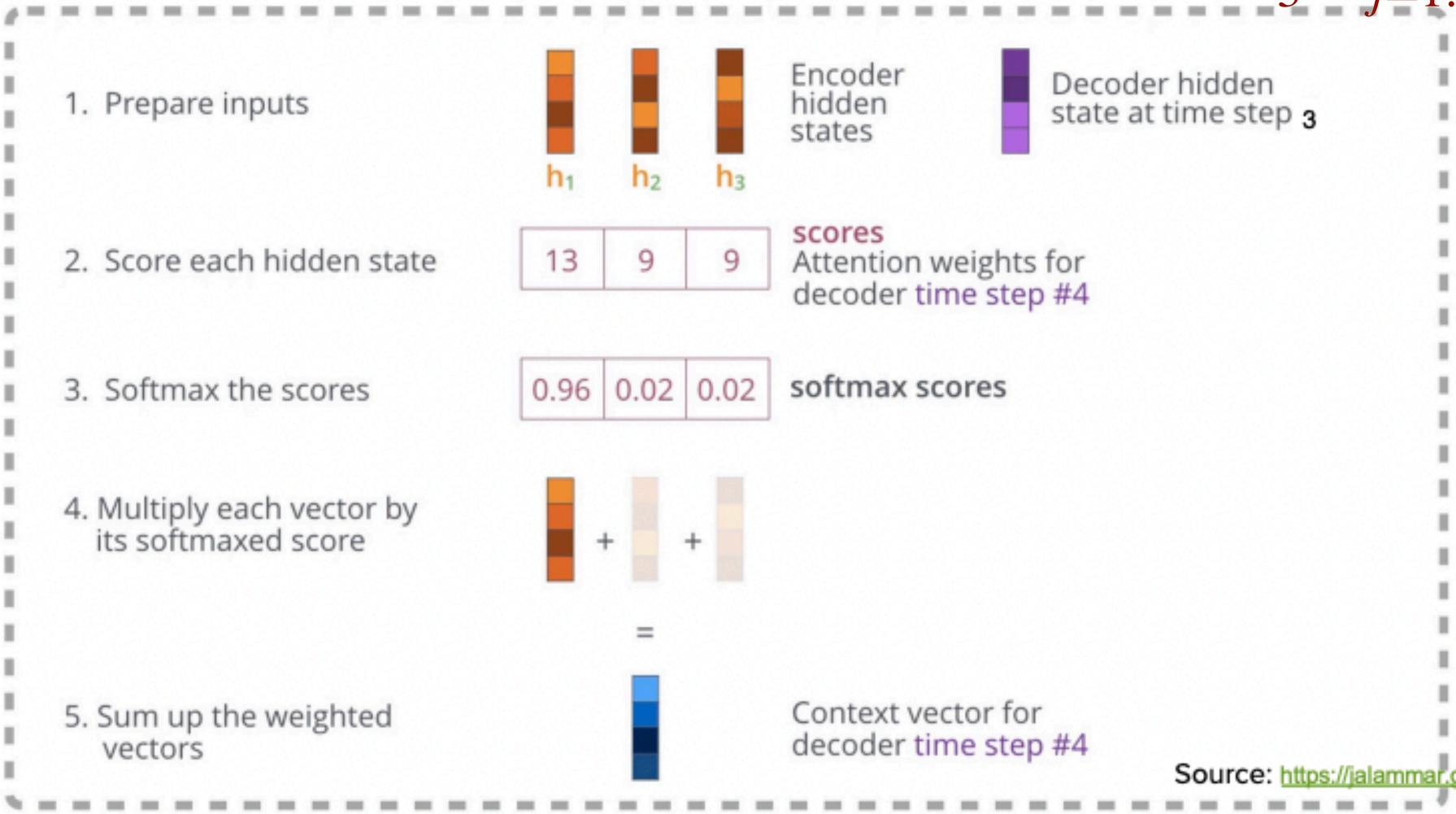
Attention

Attention

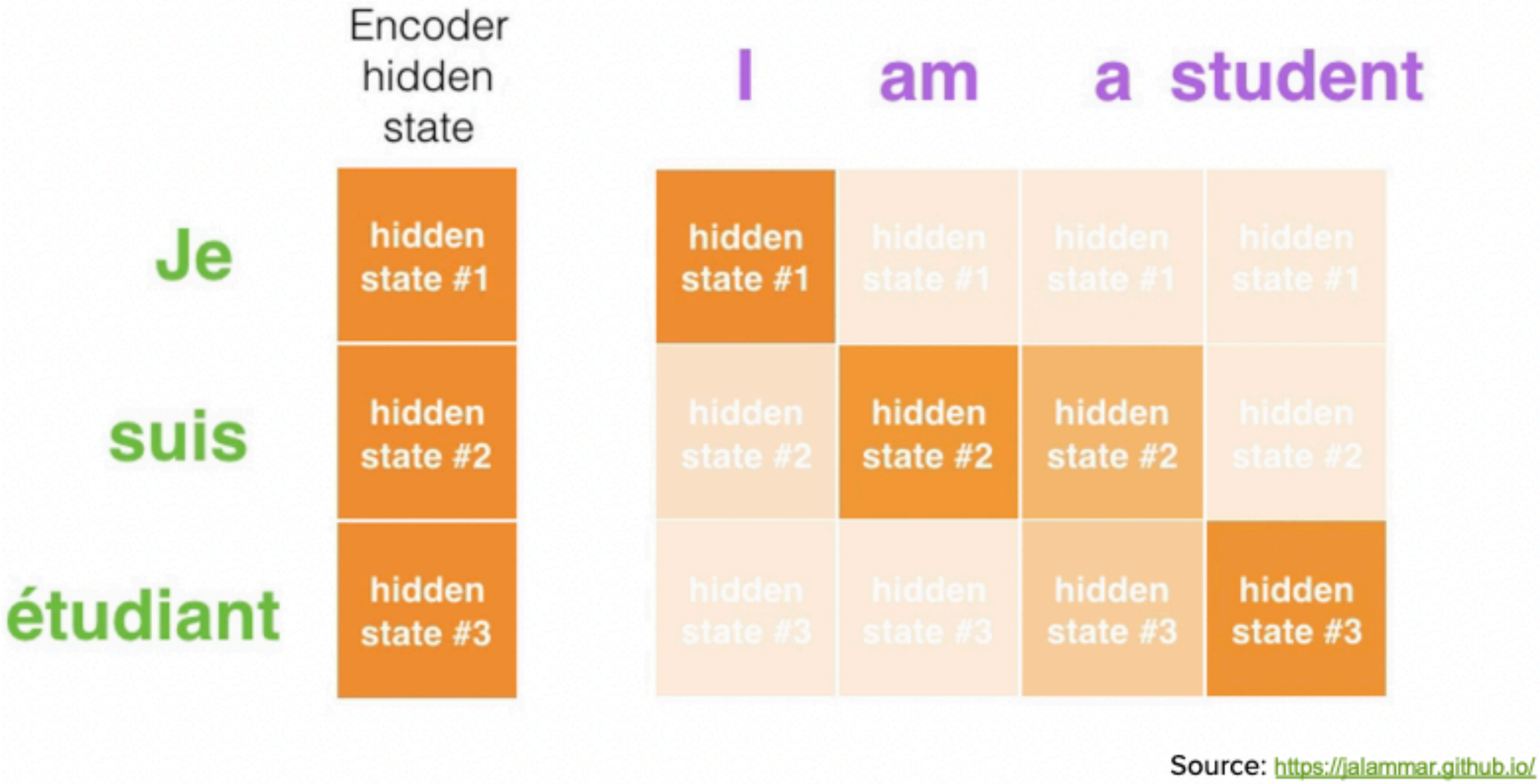
Attention

Attention as a soft, averaging look-up table

For time step 4 of decoder (only shows how to compute $\mathbf{c}_4 = f(h_3^d, h_{j=1:n}^e)$



What is actually does?



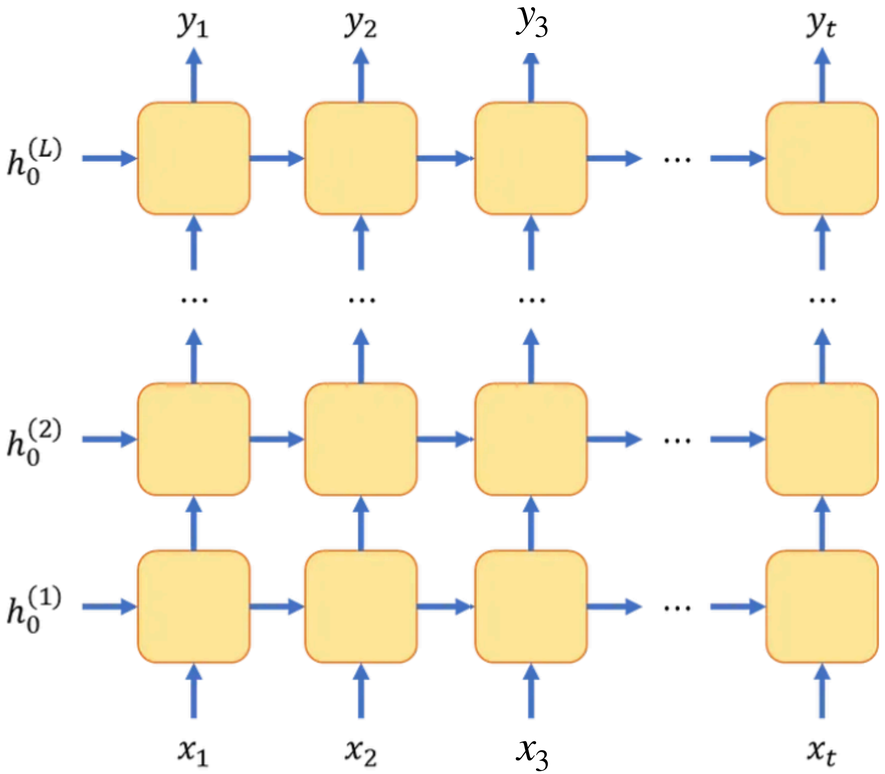
Source: https://jalammar.github.io/

Limitations of RNNs

- Longer sequences can lead to vanishing gradients.

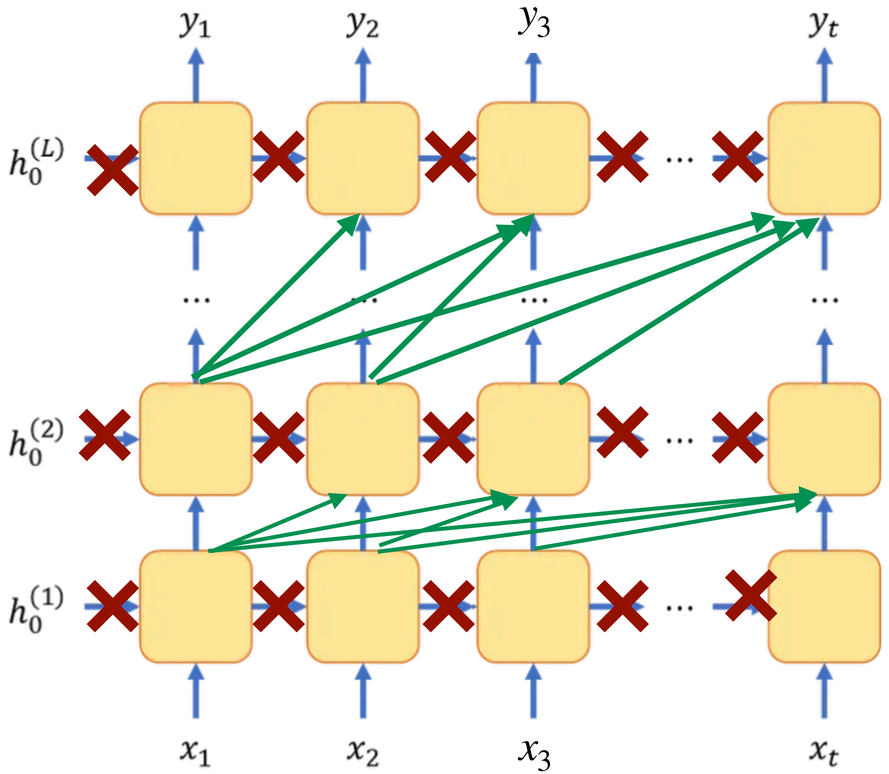
- RNNs lack parallelizability.
  - Forward and backward passes have O(sequence length) unparallelizable operations.
  - This is insufficient use of GPUs. They can perform a bunch of independent computations at once!
  - Makes training on very large datasets difficult.

With attention, do we need recurrence? Maybe not!



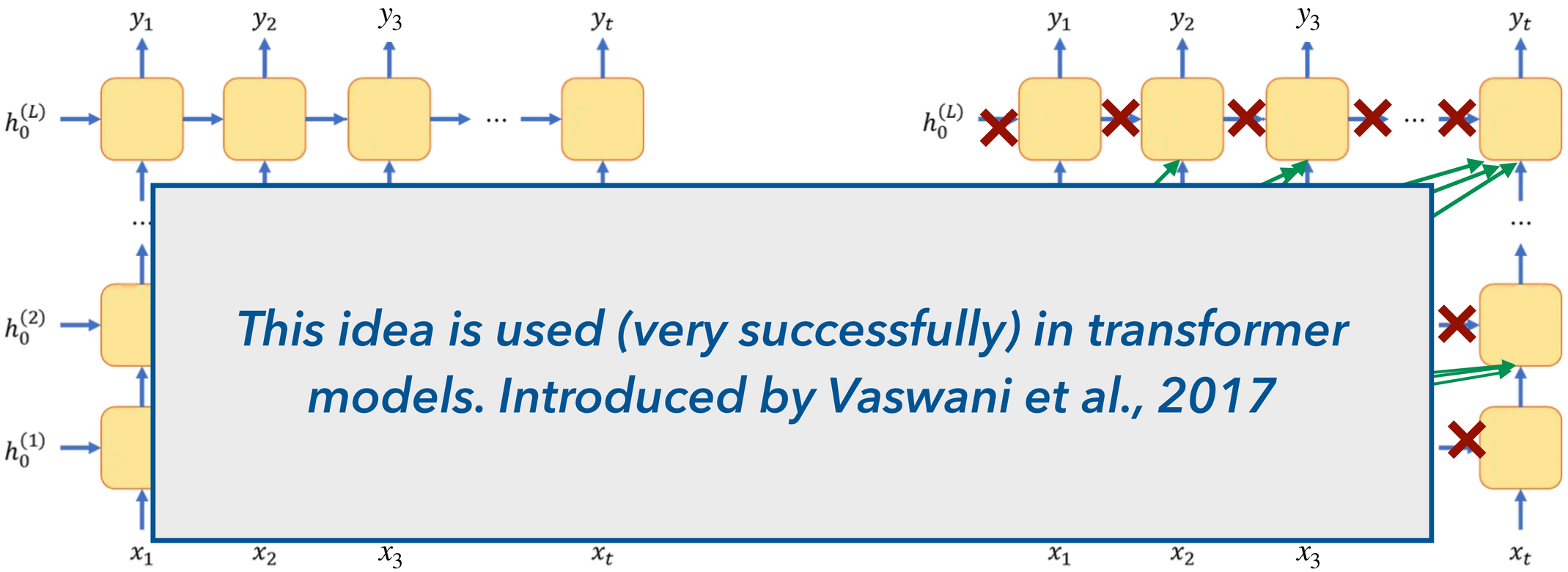
Multi-layer RNN

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!

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!

Transformers

Why should we learn about transformers?

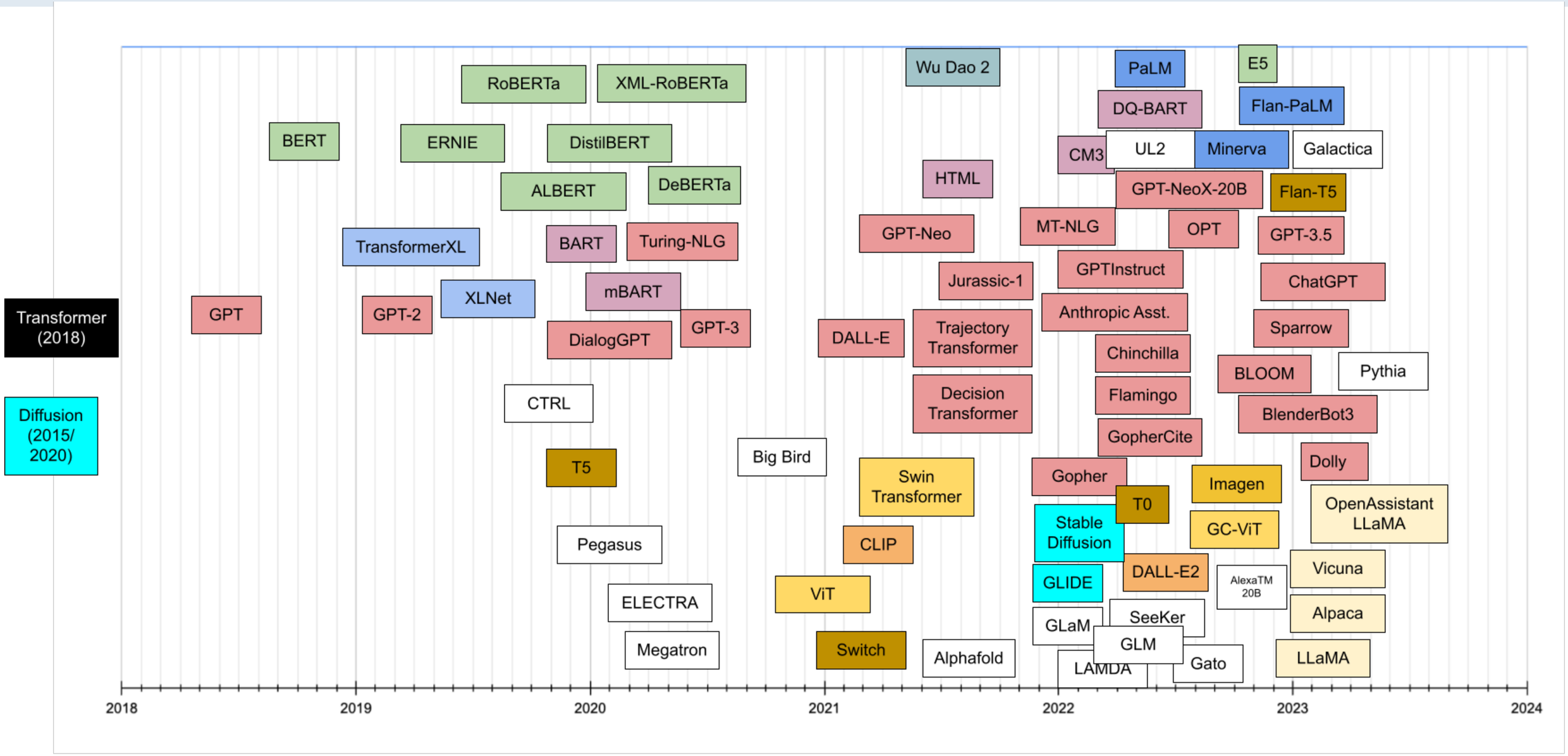
- Transformer (variants) are the backbone of all powerful LLMs today!

ChatGPT

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?



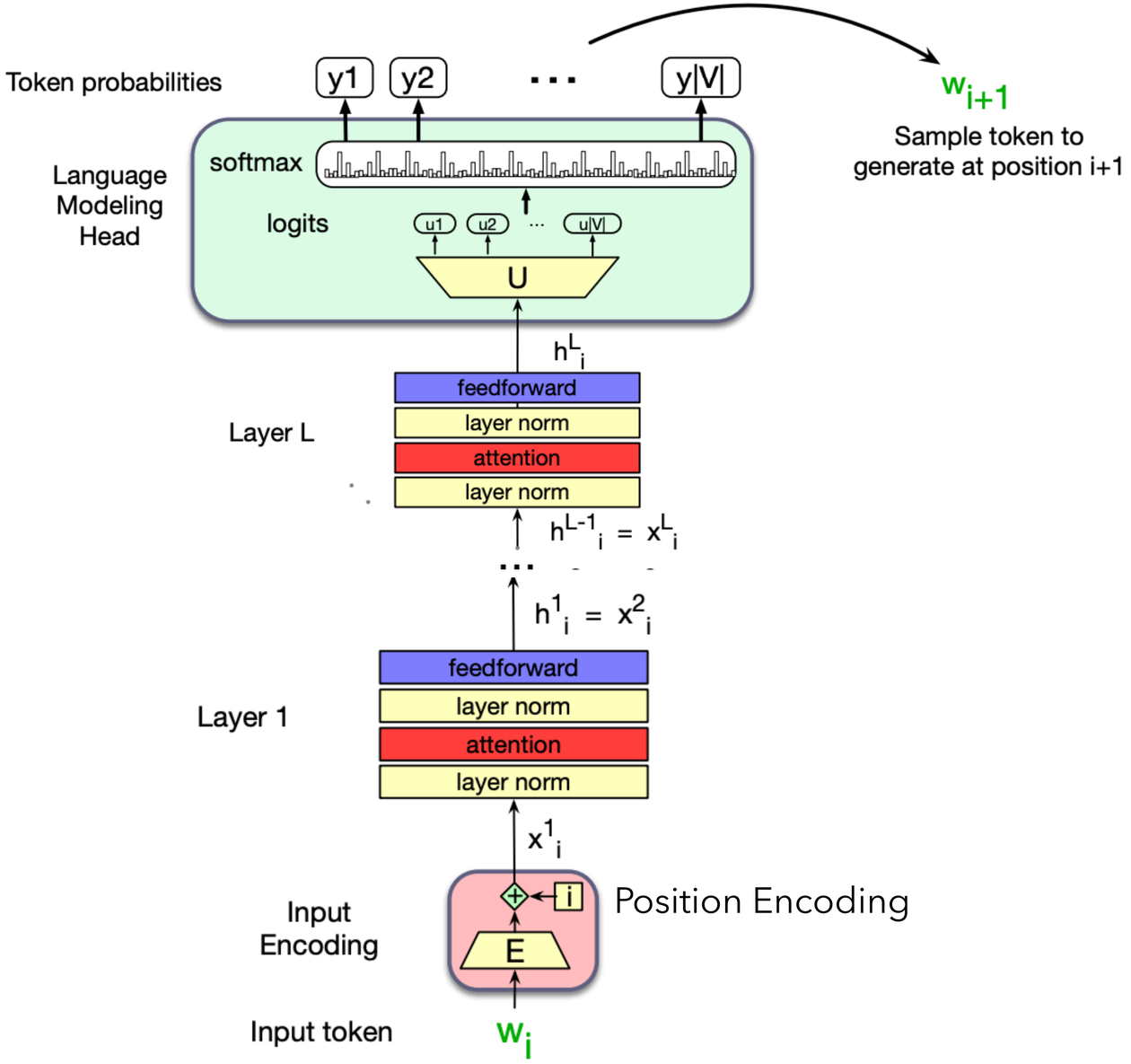
Why should we learn about transformers?

- 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

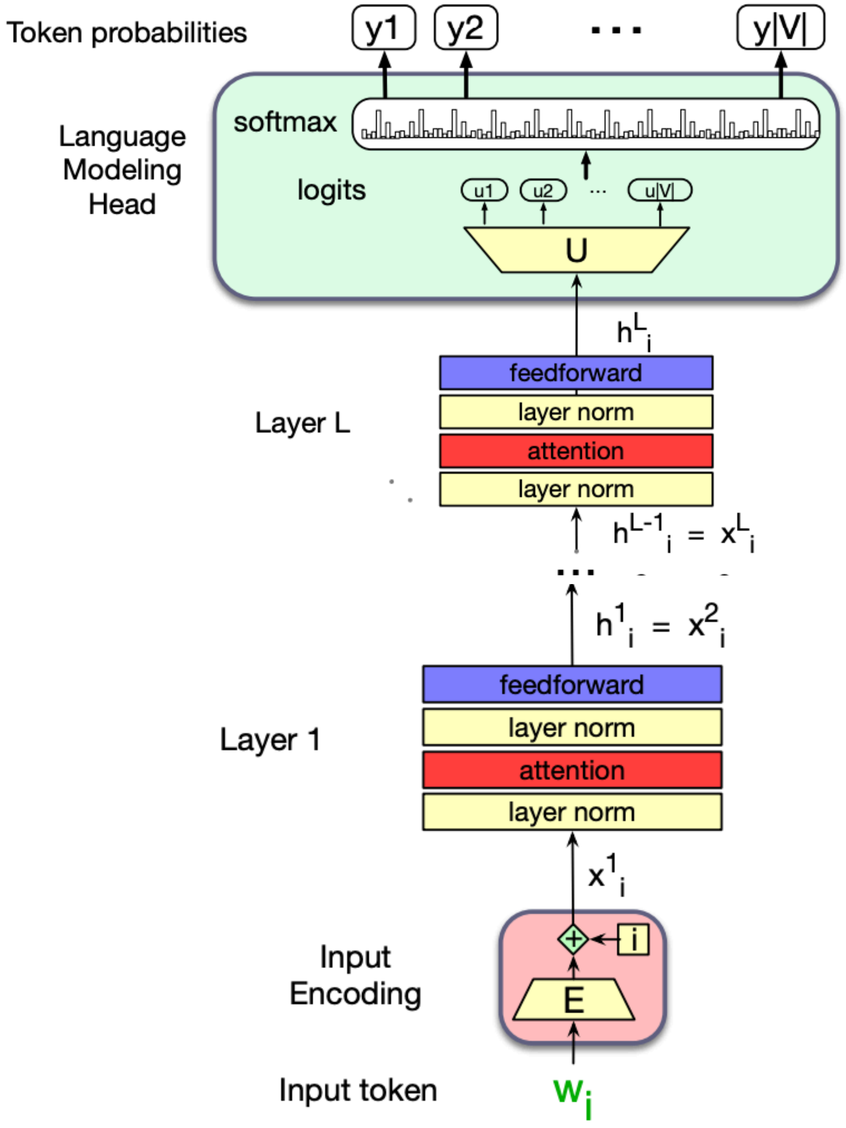
- Introduced in Attention Is All You Need (Vaswani et al. NeurIPS 2017)
- A purely attention-based architecture (highly parallelizable), i.e. no recurrence
- Very deep model for NLP (12 layers)
- Originally envisioned for seq2seq tasks (encoder is 6 layers, decoder is 6 layers)
- The encoder and decoder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

Transformer Architecture (Decoder-only)



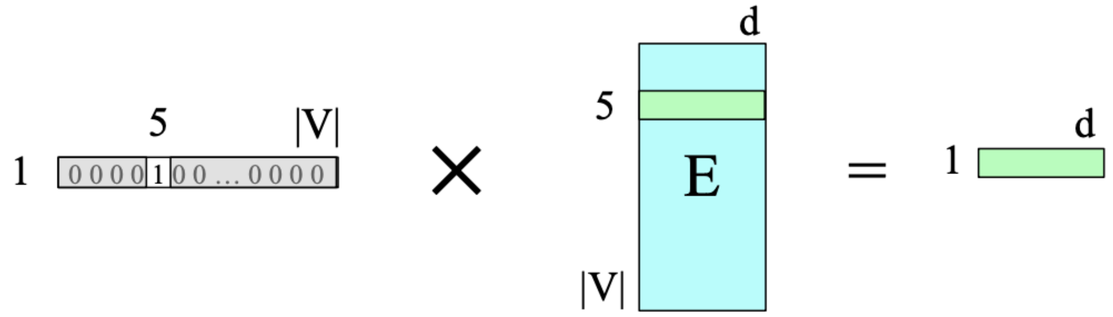
- We will build up to this!!
- Main components of a transformer model
  - **(Multi-head) Attention**
  - Feed forward
  - Layer Norm
  - Position Encoding

Input Layer



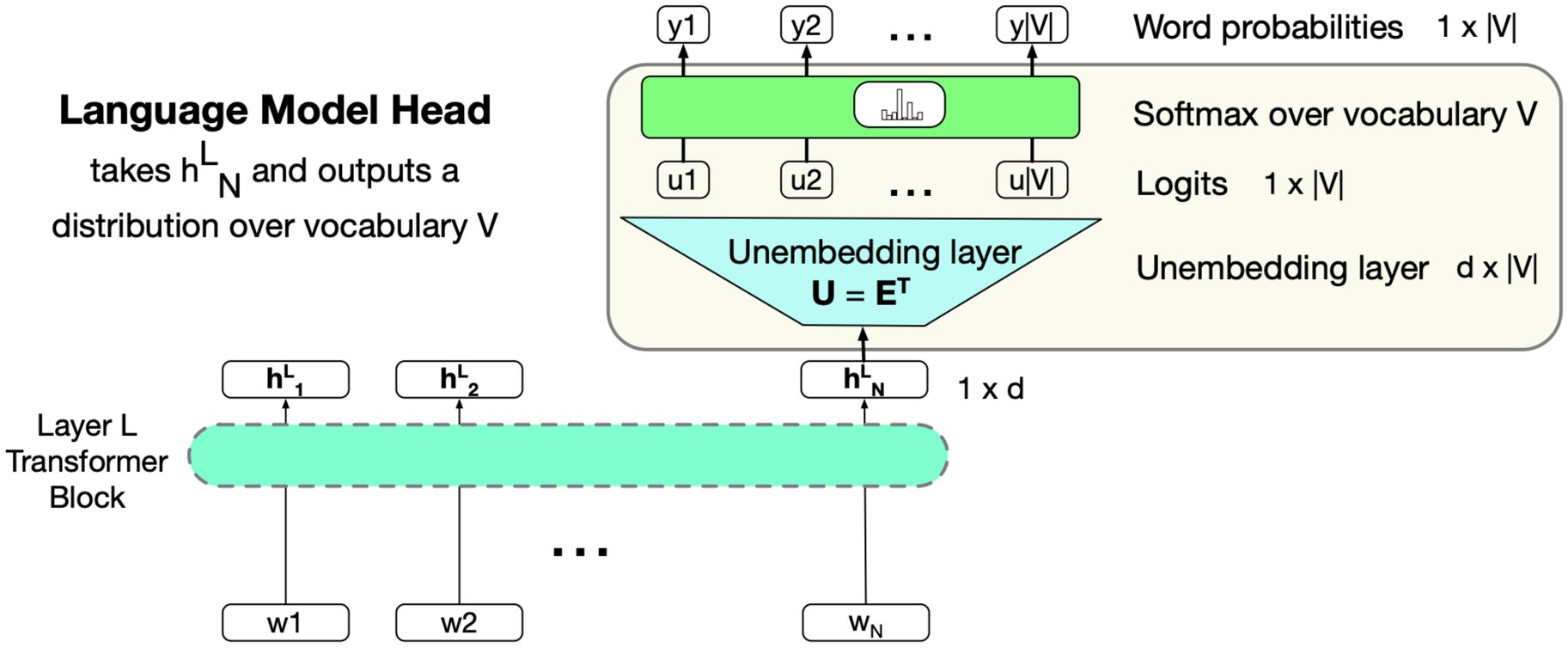
Input token w_i represented as a one-hot vector.

Matrix-multiplication with the Embedding matrix E gives d-dim vector representation of w_i



Output Layer

- Remember: We are studying a decoder-only Language Model.
- Training objective: Predict the next token, given preceding tokens.
- Final output: probability distribution over the vocabulary. Next word sampled this distribution.



Simplified Attention

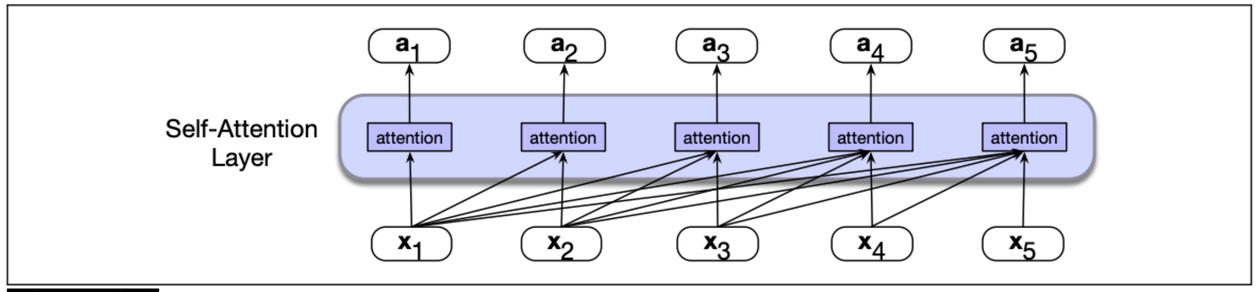


Figure 9.3 Information flow in causal self-attention. When processing each input $\mathbf{x}_i$, the model attends to all the inputs up to, and including $\mathbf{x}_i$.

- Goal: Given inputs x_i from the previous layer, transform them into a_i to encode more information from the context.

Simplified Attention

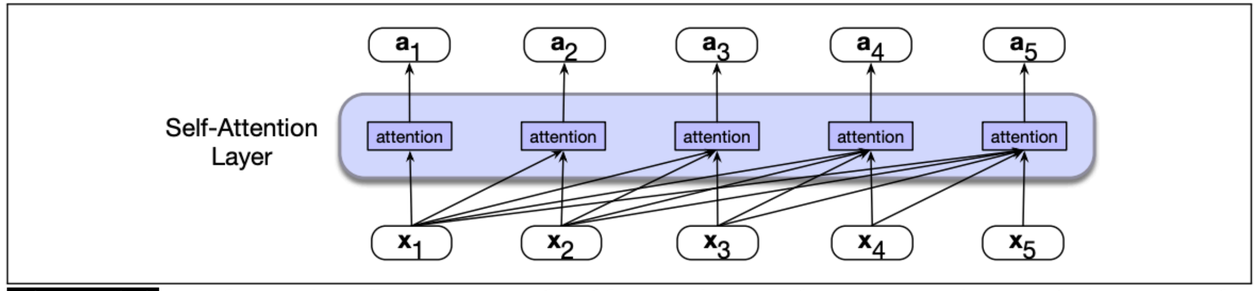
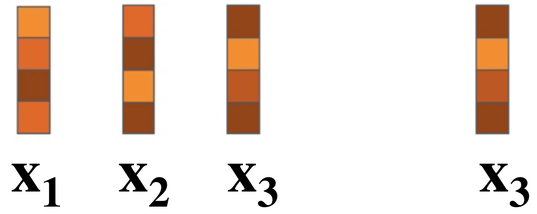


Figure 9.3 Information flow in causal self-attention. When processing each input $\mathbf{x}_i$, the model attends to all the inputs up to, and including $\mathbf{x}_i$.

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

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

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. $\mathbf{a}_3$

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

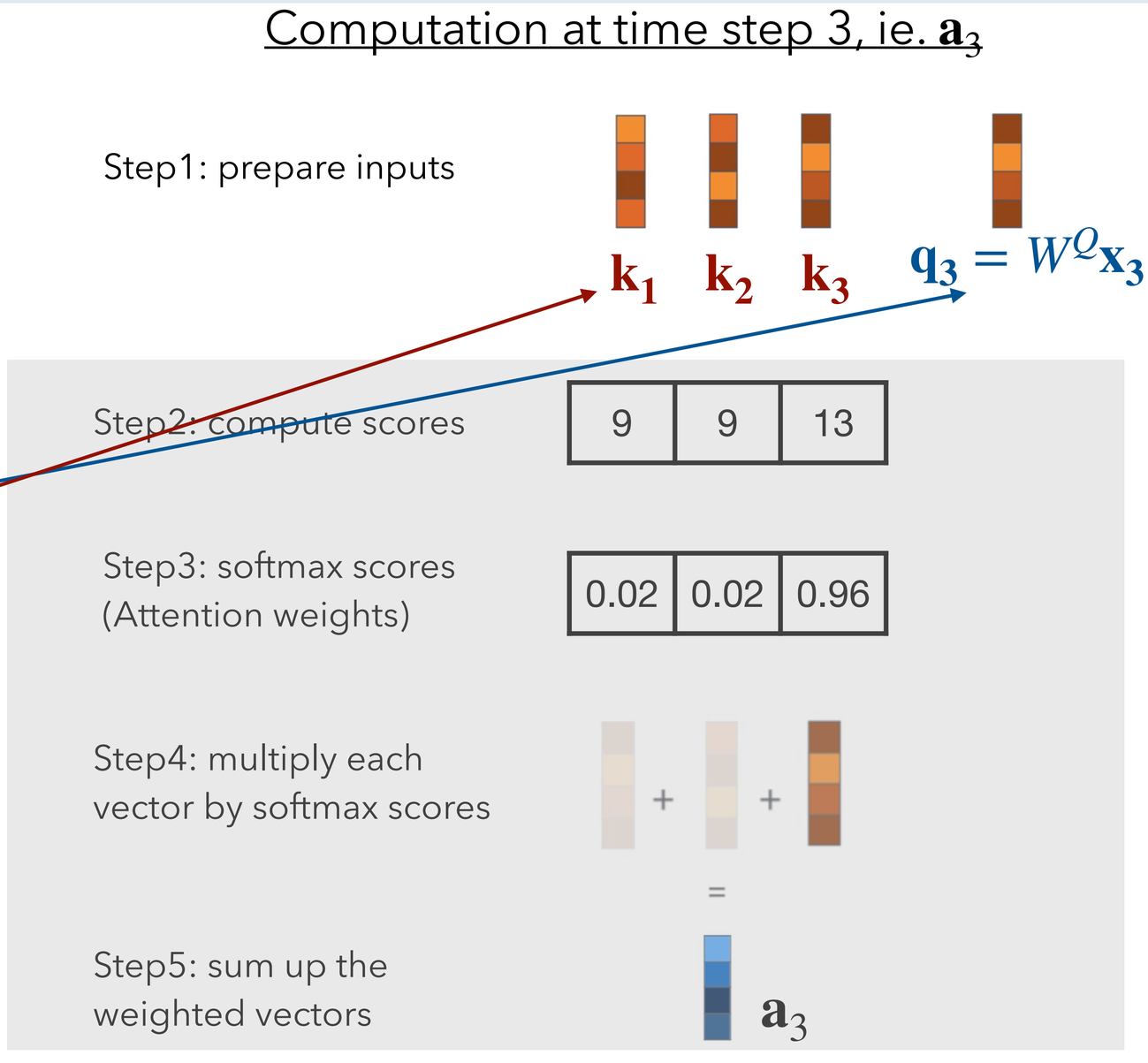
- Attention in Transformer architectures
  - For a given input $\mathbf{x_i}$ (could be the input at any layer of an encoder or decoder) create three different "roles" or "versions":

    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



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

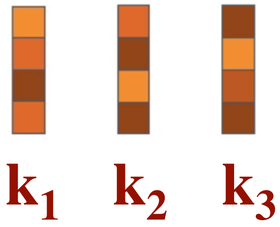
    key vector: $\mathbf{k}_i = \mathbf{x}_i \, W^K$

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

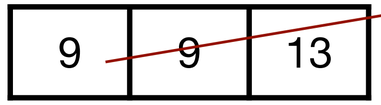
    W^Q, W^K, W^V are learned matrices

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

Step1: prepare inputs



Step2: compute scores



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

d_k is dim of query, key vectors

Step3: softmax scores (Attention weights)

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

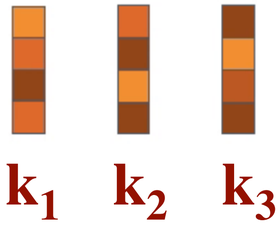
key vector: $\mathbf{k}_i = \mathbf{x}_i \, W^K$

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

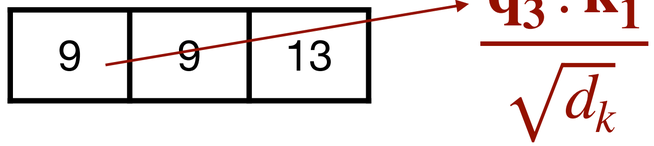
W^Q, W^K, W^V are learned matrices

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

Step1: prepare inputs

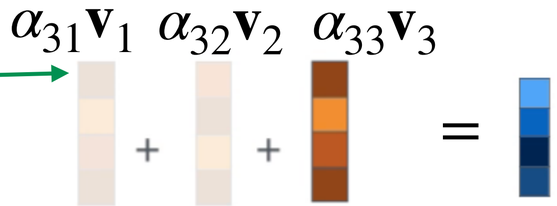


Step2: compute scores



Step3: softmax scores (Attention weights)

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

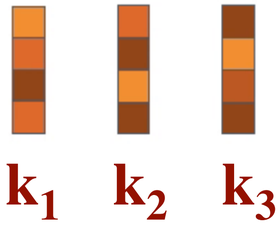
key vector: $\mathbf{k}_i = \mathbf{x}_i\, W^K$

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

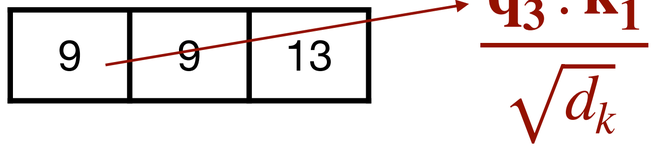
W^Q, W^K, W^V are learned matrices

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

Step1: prepare inputs

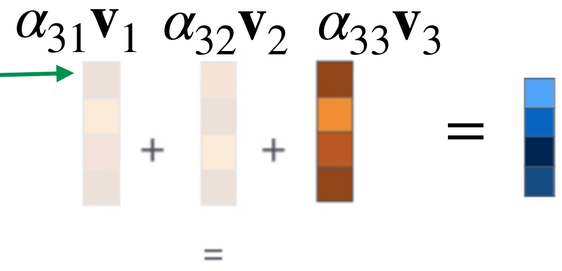


Step2: compute scores

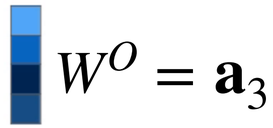


Step3: softmax scores (Attention weights)

Step4: multiply each vector by softmax scores



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



Attention in Transformer models

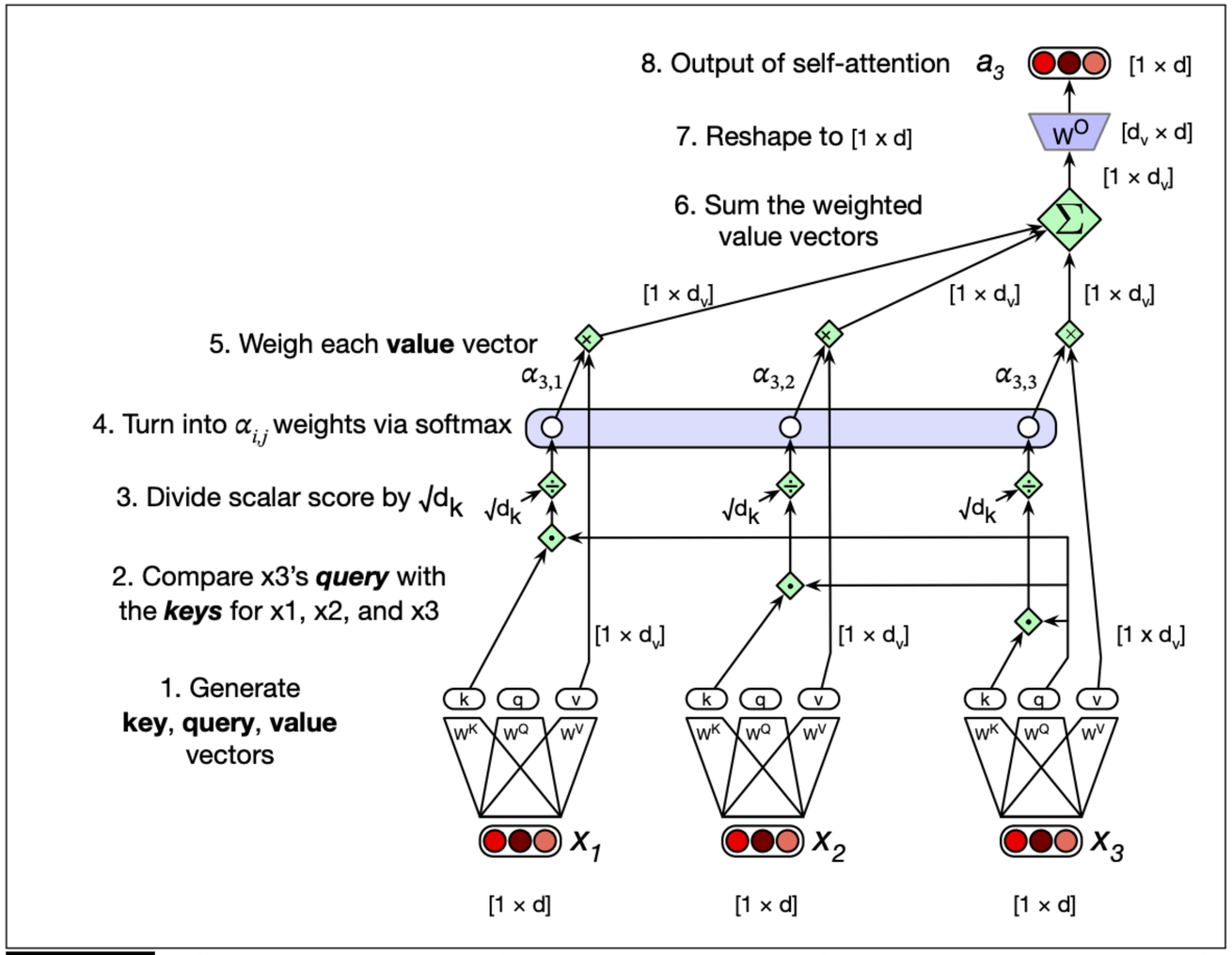
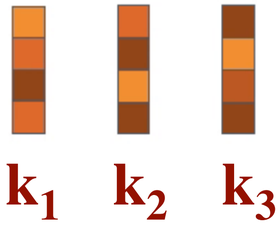


Figure 9.4 Calculating the value of $\mathbf{a}_3$, the third element of a sequence using causal (left-to-right) self-attention.

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

Step1: prepare inputs



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

Step2: compute scores

9	9	13

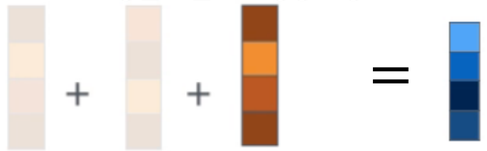
$$\frac{\mathbf{q}_3 \cdot \mathbf{k}_1}{\sqrt{d_k}}$$

Step3: softmax scores (Attention weights)

0.02	0.02	0.96

Step4: multiply each vector by softmax scores

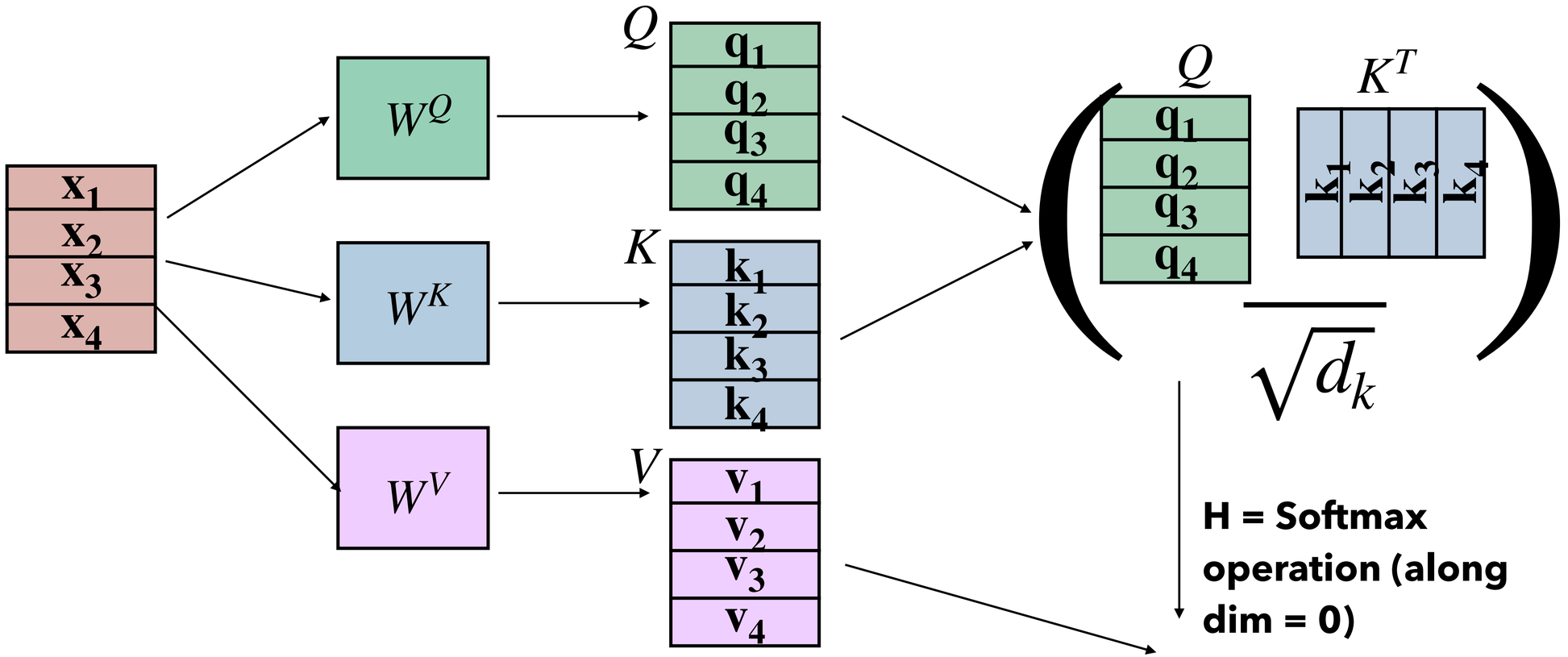
$\alpha_{31}\mathbf{v}_1$ $\alpha_{32}\mathbf{v}_2$ $\alpha_{33}\mathbf{v}_3$



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

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

Attention Computation (matrix form)



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

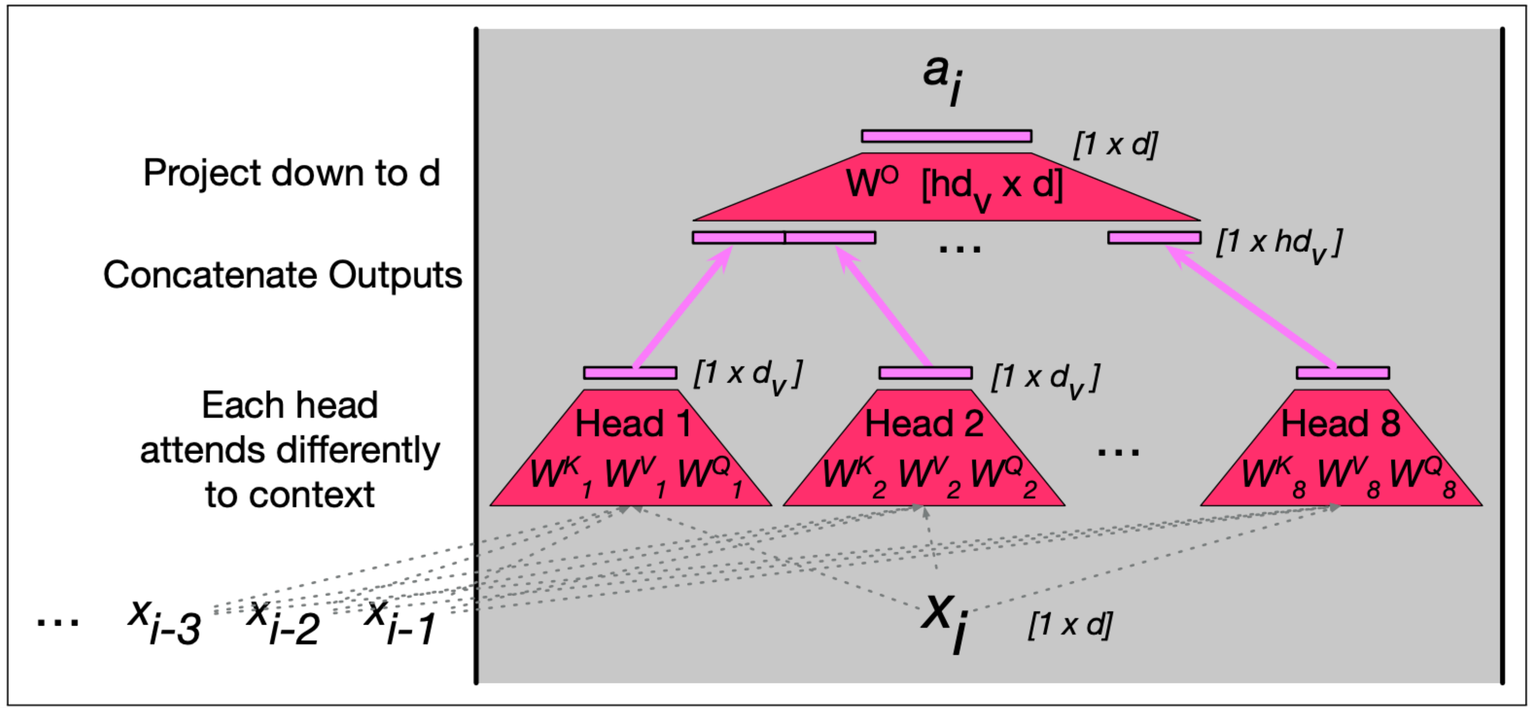
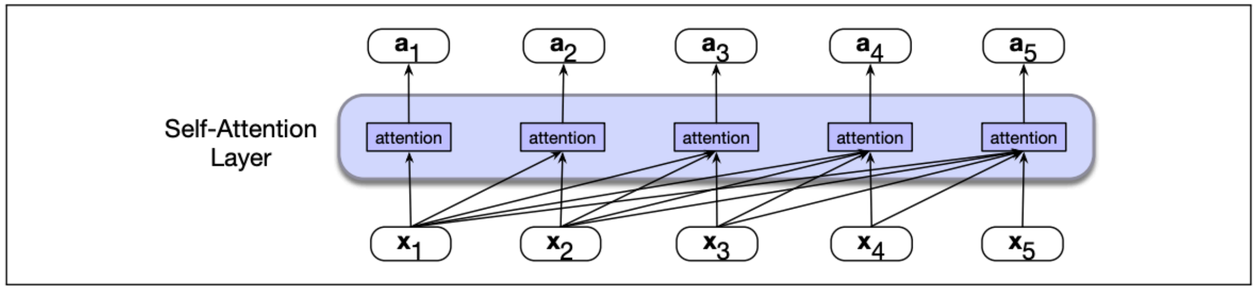


Figure 9.5 The multi-head attention computation for input $\mathbf{x}_i$, producing output $\mathbf{a}_i$. A multi-head attention layer has A heads, each with its own key, query and value weight matrices. The outputs from each of the heads are concatenated and then projected down to d, thus producing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

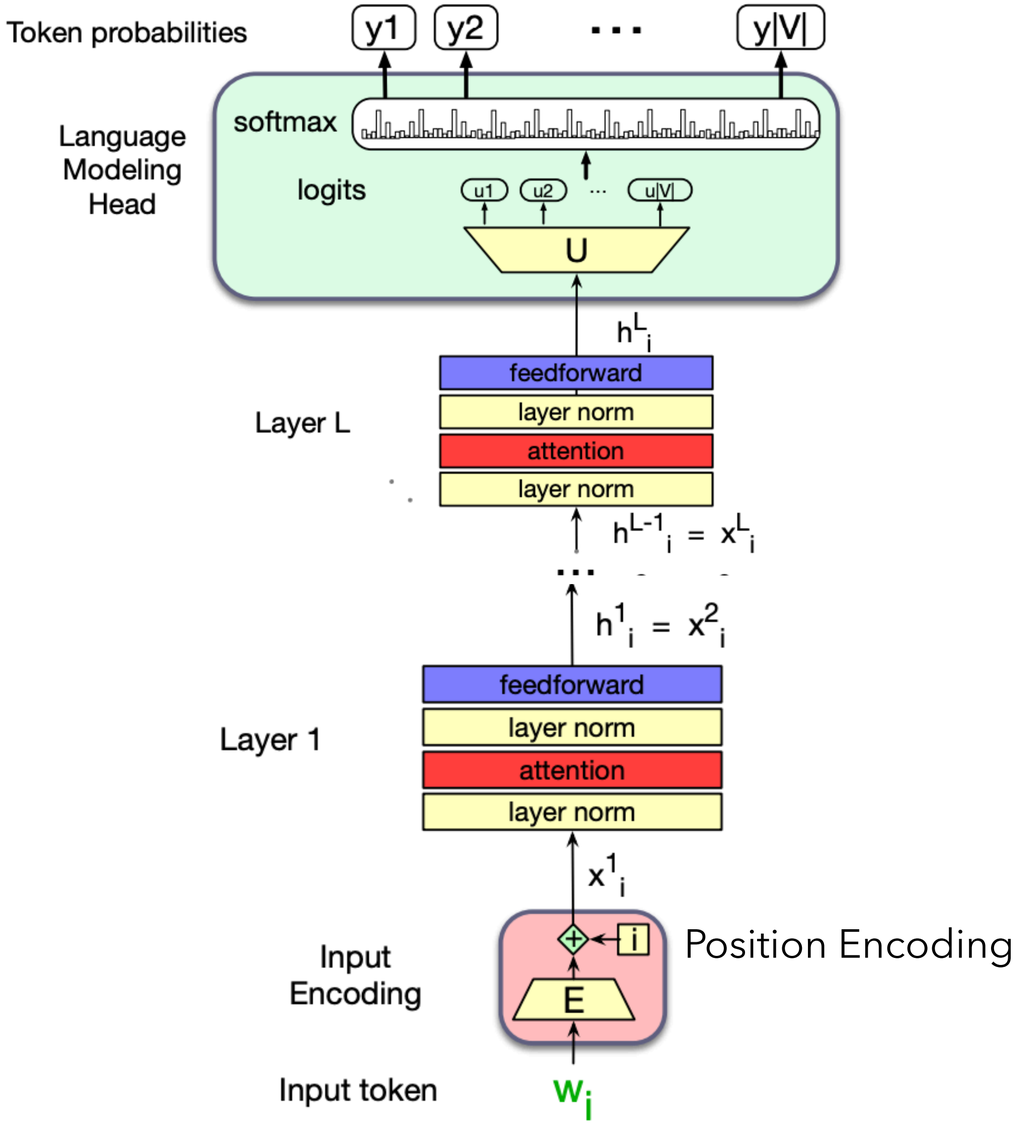


- Self-attention layer transformed the input $\mathbf{x_i}$ to output $\mathbf{a_i}$
- Word order information is lost!

An old dog and a young boy ….

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

Let's go back to our transformer arch



- Today:
  - Self-attention
  - Multi-head self-attention

- Next Class:
  - Position Embeddings
  - 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
- CS 288 course by Alane Suhr (UC Berkeley).