Amirkabir University of Technology
(Tehran Polytechnic)

Natural Language Processing

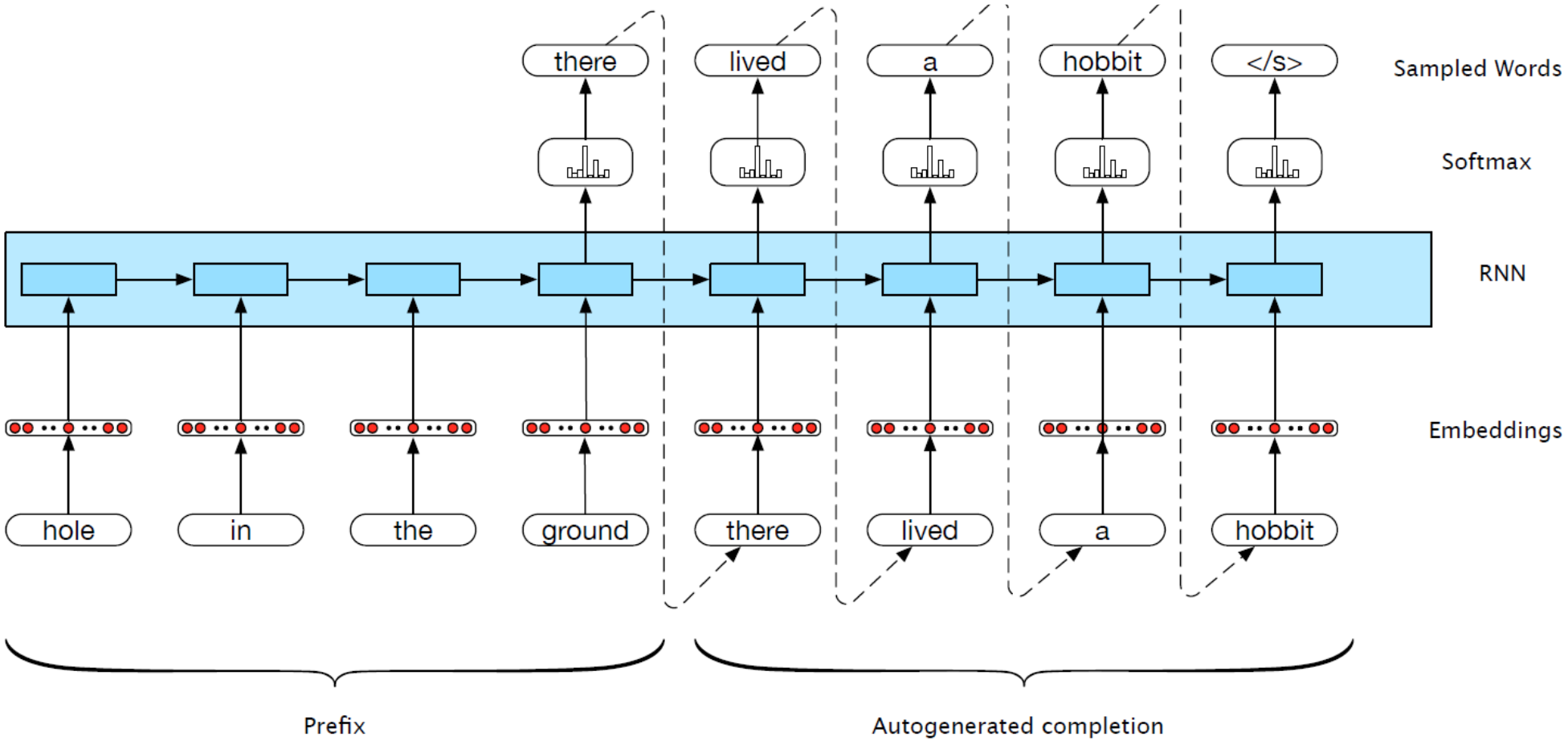
Lecture 12: Attention and Transformers

Amirkabir University of Technology
Dr Momtazi

Outline

- **Encoder-decoder**
- Self Attention Network: Transformers

Text Generation with RNNs



From (autoregressive) generation to Machine Translation

- Training data are parallel text e.g., English / French

  there lived a hobbit *vivait un hobbit*

  ……..

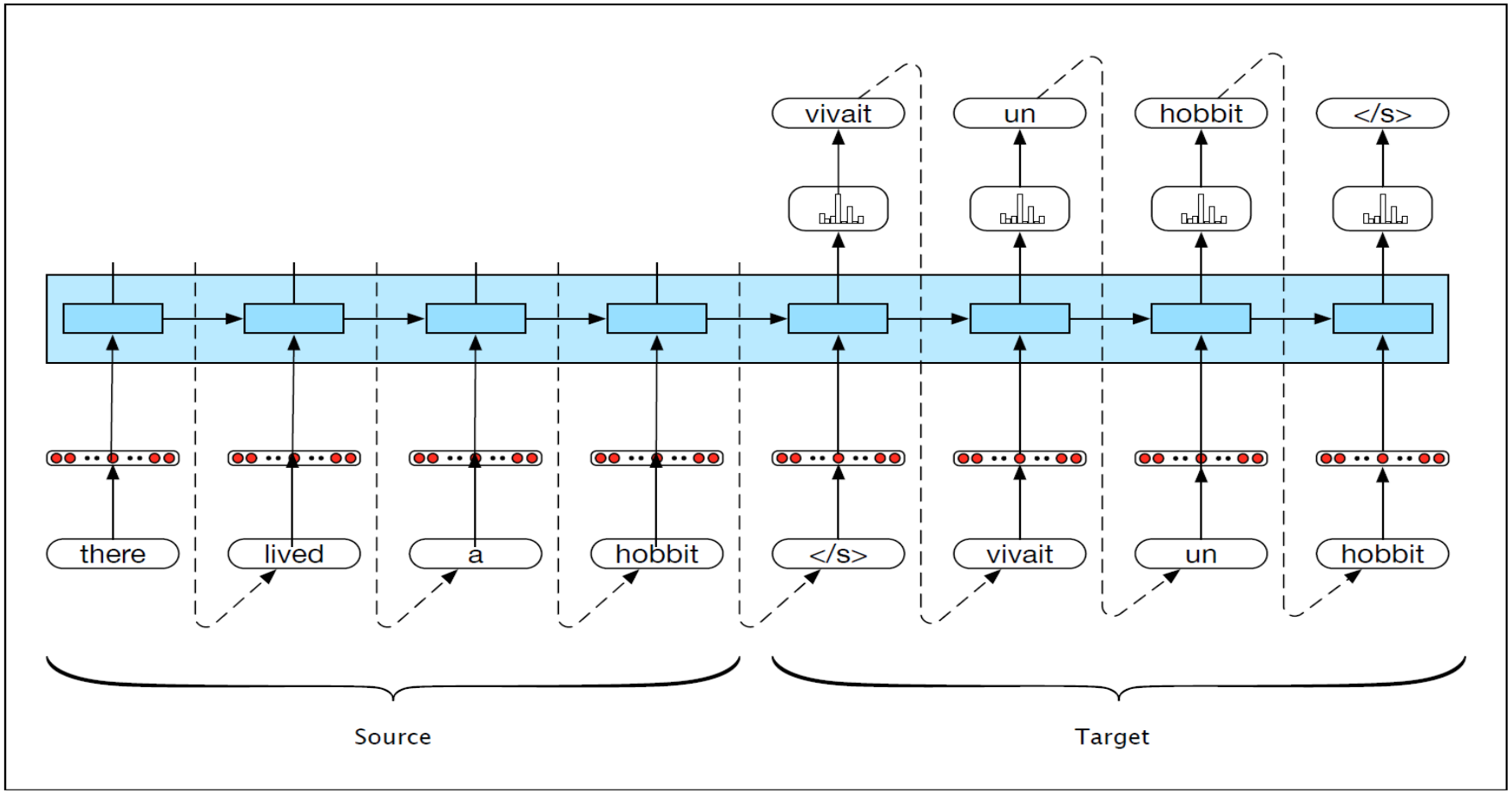
- Build an RNN language model on the concatenation of source and target

  there lived a hobbit <\s> *vivait un hobbit* <\s>

  ……..

From (autoregressive) generation to Machine Translation

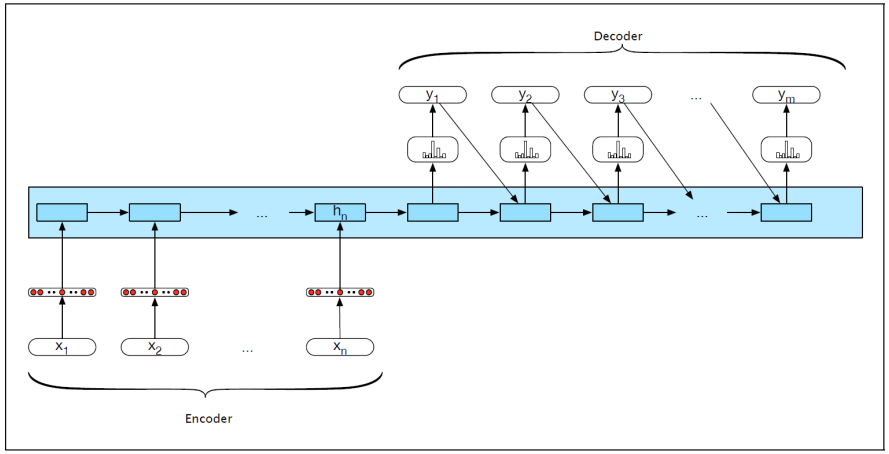
- Translation as Sentence Completion !



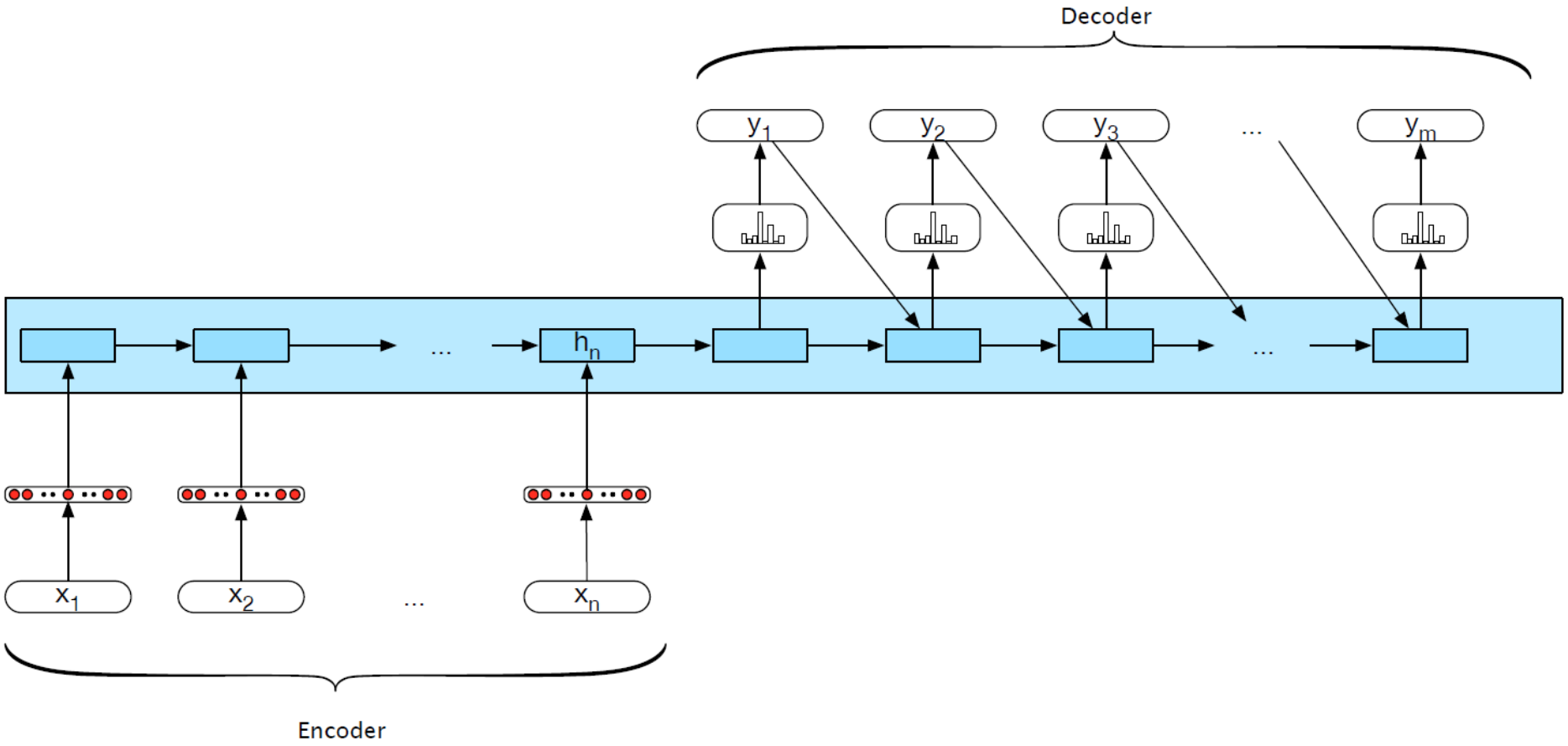
Encoder Decoder Networks

- **Limiting design choices**
  - **E** and **D** assumed to have the same internal structure (here RNNs)
  - Final state of the **E** is the only context available to **D**
  - This context is only available to **D** as its initial hidden state

- Encoder generates a contextualized representation of the input (last state)
- Decoder takes that state and autoregressively generates a sequence of outputs

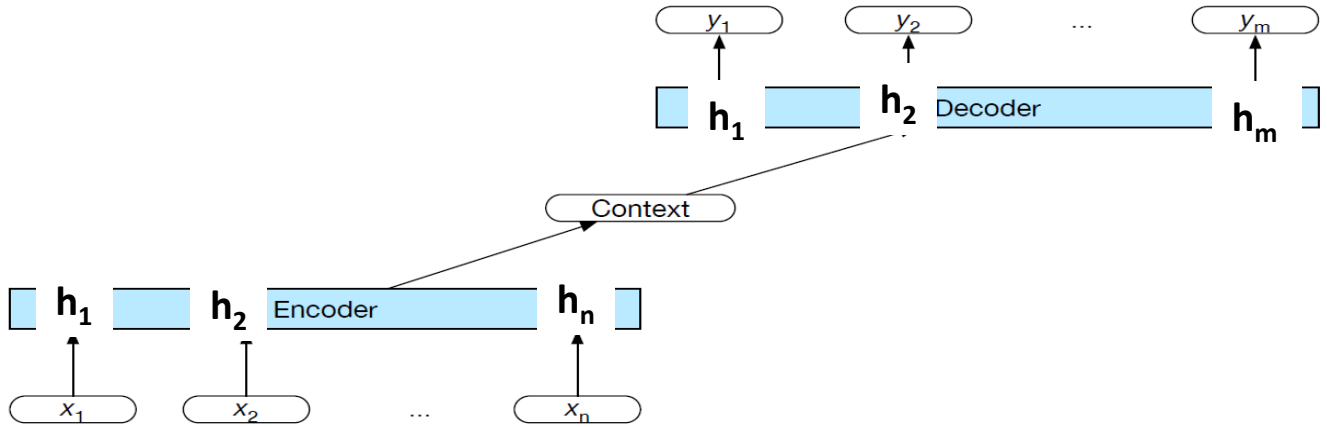


Encoder Decoder Networks



General Encoder Decoder Networks

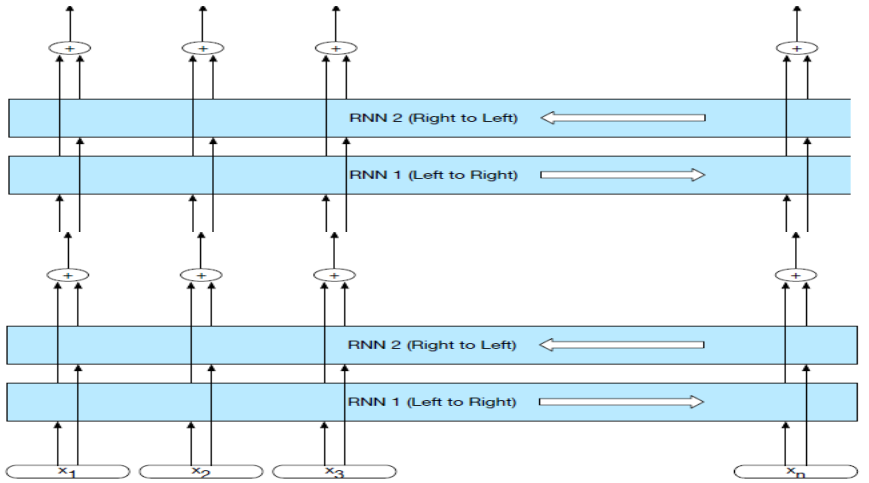
- Abstracting away from these choices
  - **Encoder**: accepts an input sequence, $\mathbf{x}_{1:n}$ and generates a corresponding sequence of contextualized representations, $\mathbf{h}_{1:n}$
  - **Context vector c**: function of $\mathbf{h}_{1:n}$ and conveys the essence of the input to the decoder
  - **Decoder**: accepts **c** as input and generates an arbitrary length sequence of hidden states $\mathbf{h}_{1:m}$ from which a corresponding sequence of output states $\mathbf{y}_{1:m}$ can be obtained.



Popular architectural choices: Encoder

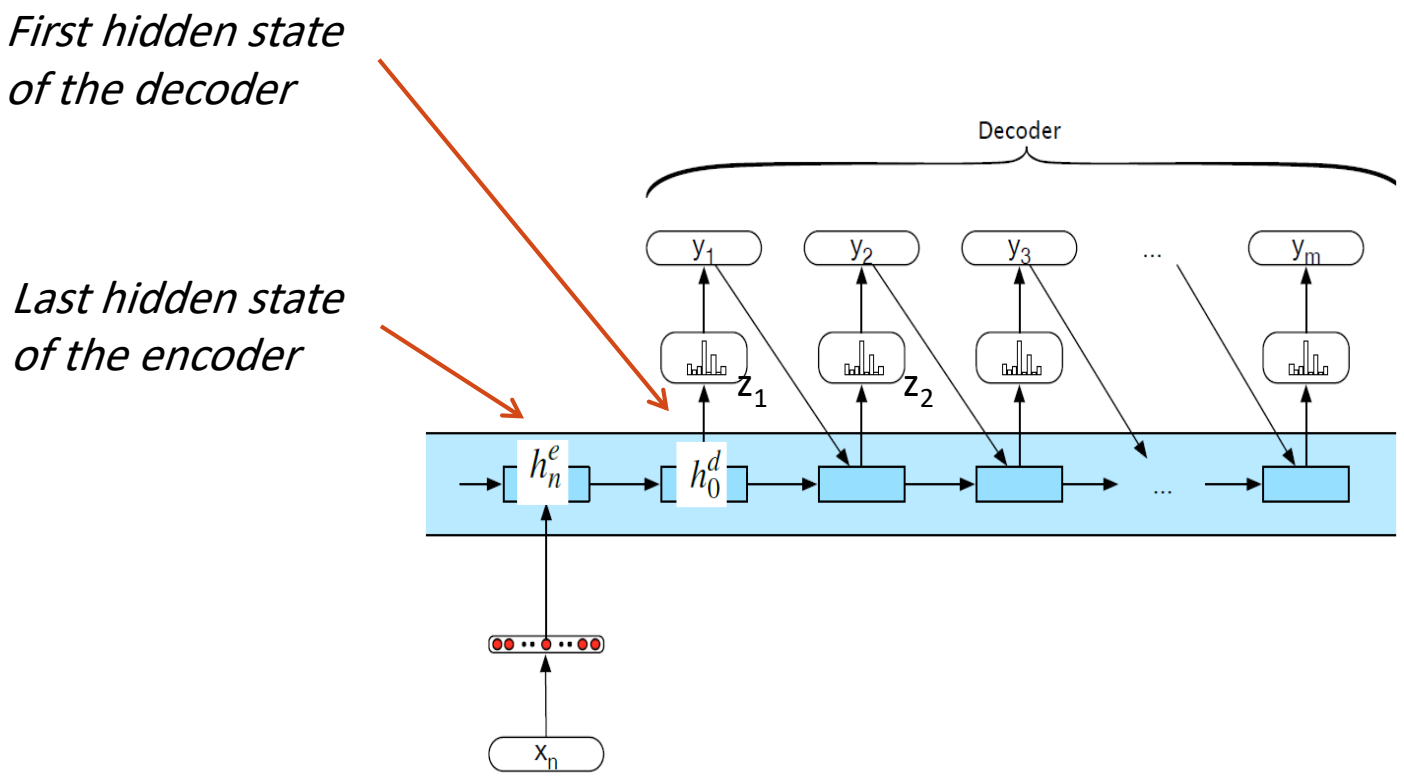
Widely used encoder design: **stacked Bi-LSTMs**

- Contextualized representations for each time step**:**
  - **Hidden states from top layers** from the forward and backward passes



Decoder Basic Design

- Produce an output sequence an element at a time



Decoder

- Enhancement:
  - Context available at each step of decoding

- How output y is chosen
  - **Soft-max** distribution (OK for generating novel output, not OK for e.g. MT or Summarization)
  - **Most likely output** (doesn't guarantee individual choices being made make sense together)

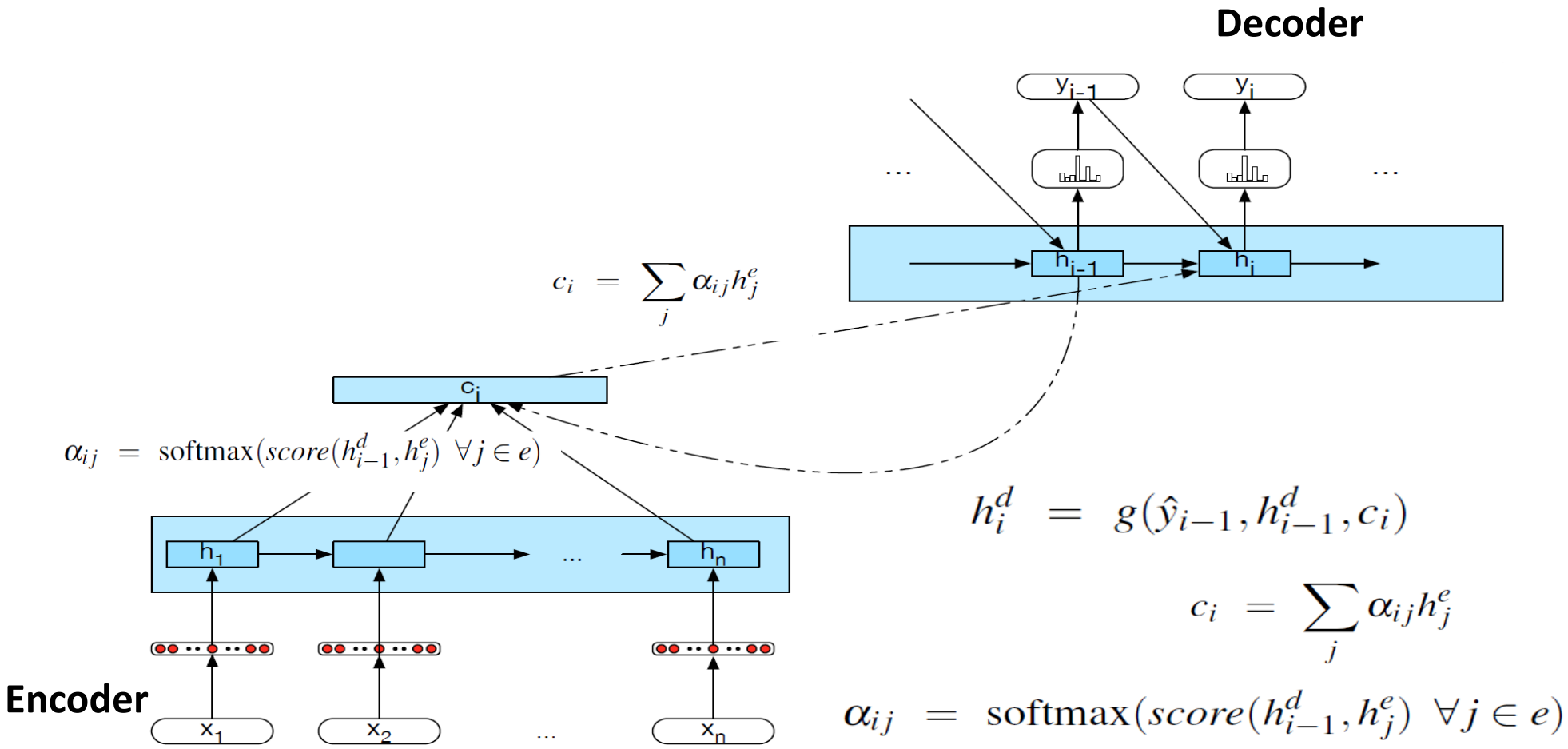
Attention Mechanism

- Flexible context: Attention
- **Context vector c**: function of $\mathbf{h}_{1:n}$ and conveys the essence of the input to the decoder.
- **Flexible?**
  - Different for each $\mathbf{h}_i$
  - Flexibly combining the $\mathbf{h}_j$

Attention Mechanism

- Dynamically derived context
  - Replace static context vector with dynamic $\mathbf{c_i}$
  - Derived from the encoder hidden states at each point i during decoding
  - **Ideas**:
    - Should be a linear combination of those states
- Computing c_i
  - Compute a vector of scores that capture the relevance of each encoder hidden state to the decoder state
- Computing c_i From scores to weights
  - Create vector of weights by normalizing scores
- **Goal achieved**: compute a fixed-length context vector for the current decoder state by taking a weighted average over all the encoder hidden states.

Attention Mechanism

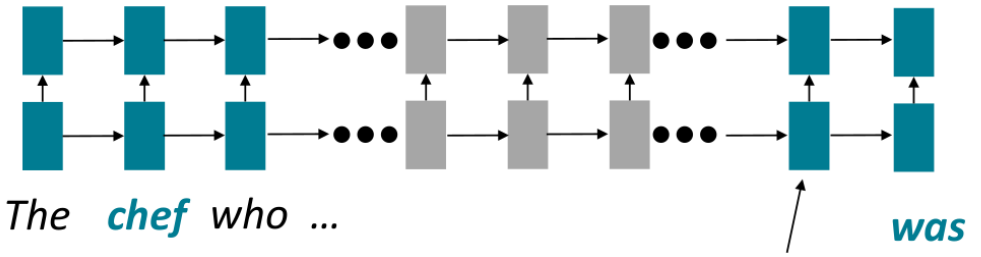


Outline

- Encoder-decoder
- **Self Attention Network: Transformers**

RNNs' Shortcomings

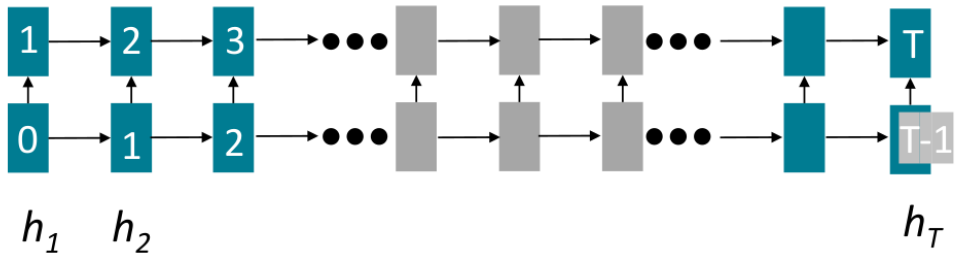
- RNNs take O(sequence length) steps for distant word pairs to interact means:
  - Hard to learn long-distance dependencies (because gradient problems!)
  - Linear order of words is "baked in"; we already know linear order isn't the right way to think about sentences…



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

RNNs' Shortcomings

- Forward and backward passes have O(sequence length) unparallelizable operations
  - GPU 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

If not recurrence, then what? How about word windows?

- Word window models aggregate local contexts
  - (Also known as 1D convolution; we'll go over this in depth later!)
  - Number of unparallelizable operations does not increase sequence length!

- What about long-distance dependencies?
  - Stacking word window layers allows interaction between farther words

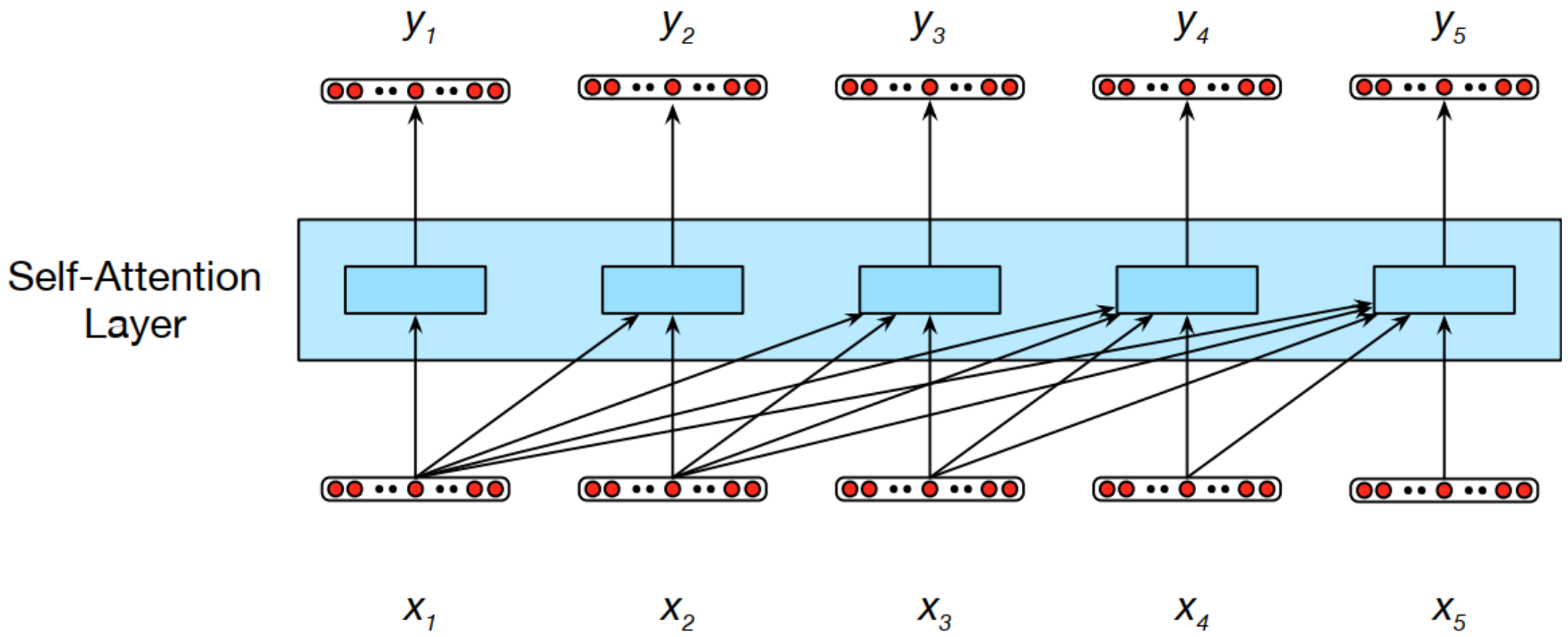
- Maximum Interaction distance = sequence length / window size
  - (But if your sequences are too long, you'll just ignore long-distance context)

Self Attention

- 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;
  - Now, we'll think about attention within a single sentence.

- A self attention layer maps an input sequence to an output sequence
- At each step of input sequence, we have access to other inputs up to now
- Computation for each item is performed independently
- The ability of parallel inference and training

Self Attention



Self Attention

- Attention treats each word's representation as a query to access and incorporate information from a set of values.
- At the core of an attention-based approach is the ability to compare an item of interest to a collection of other items in a way that reveals their relevance in the current context.
- In the case of self-attention, the set of comparisons are to other elements within a given sequence.
- The result of these comparisons is then used to compute an output for the current input.
- e.g., computation of y3 is based on a set of comparisons between the input x3 and its preceding elements x1 and x2, and to x3 itself.
- The simplest form of comparison between elements in a self-attention layer is a dot product. We refer to the result of this comparison as a score

Self Attention

$$Score(x_i, x_j) = x_i \,.\, x_j$$

- Continuing with our example, the first step in computing y_3 would be to compute three scores: $x_3 \cdot x_1$, $x_3 \cdot x_2$ and $x_3 \cdot x_3$.
- Then to make effective use of these scores, we'll normalize them with a softmax to create a vector of weights, α_{ij}
- α_{ij} indicates the proportional relevance of each input to the input element *i* that is the current focus of attention
- Given the proportional scores in α, we then generate an output value y_i by taking the sum of the inputs seen so far, weighted by their respective α value.

Self Attention

$$Score(x_i, x_j) = x_i \cdot x_j$$

$$a_{ij} = softmax\left(score(x_i, x_j)\right) \; \forall j \leq i$$

$$= \frac{exp\left(score(x_i, x_j)\right)}{\sum_{k=1}^{i} exp(score(x_i, x_k))}$$

- Continuing with our example, the first step in computing y_3 would be to compute three scores: $x_3 \cdot x_1$, $x_3 \cdot x_2$ and $x_3 \cdot x_3$.
- Then to make effective use of these scores, we'll normalize them with a softmax to create a vector of weights, α_{ij}
- α_{ij} indicates the proportional relevance of each input to the input element *i* that is the current focus of attention

Self Attention

- Given the proportional scores in α, we then generate an output value y_i by taking the sum of the inputs seen so far, weighted by their respective α value.

$$y_i = \sum_{j \leq i} \alpha_{ij} x_j$$

Self Attention

Each input embedding plays three different roles during the course of the attention process:

- As the current focus of attention when being compared to all of the other preceding inputs.

=> We'll refer to this role as a **query**.

- In its role as a preceding input being compared to the current focus of attention.

=> We'll refer to this role as a **key**.

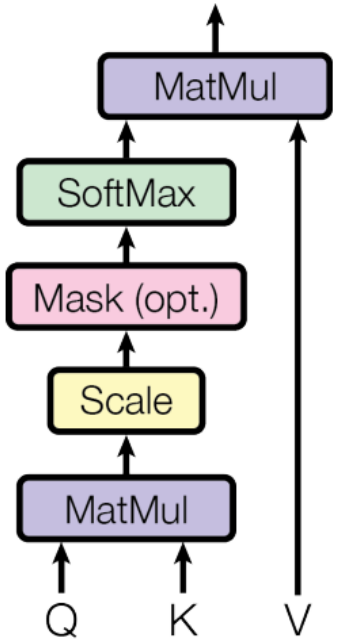
- And finally, as a **value** used to compute the output for the current focus of attention.

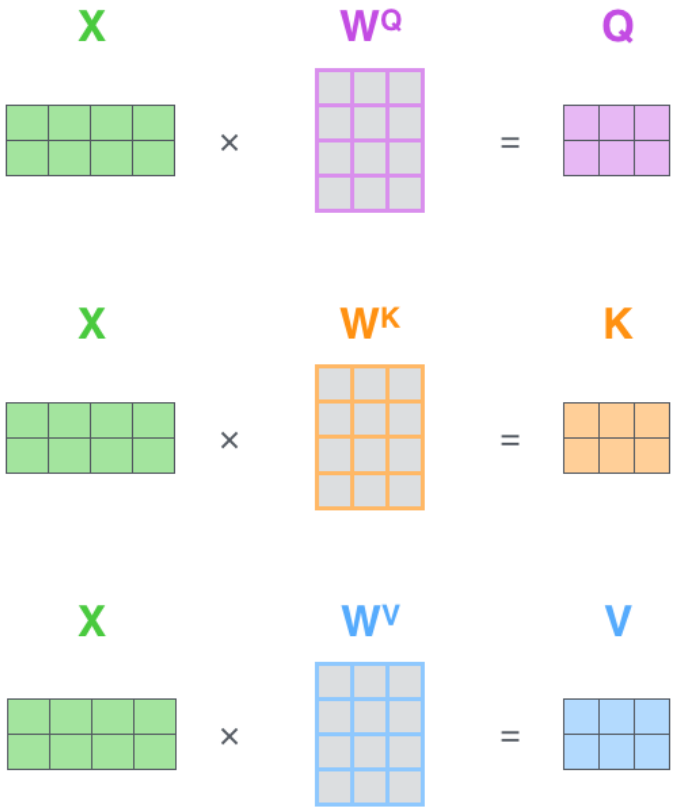
$$score(x_i, x_j) = q_i \cdot k_j$$

$$y_i = \sum_{j \leq i} a_{ij} v_j$$

$$score(x_i, x_j) = \frac{q_i k_j}{\sqrt{d_k}}$$

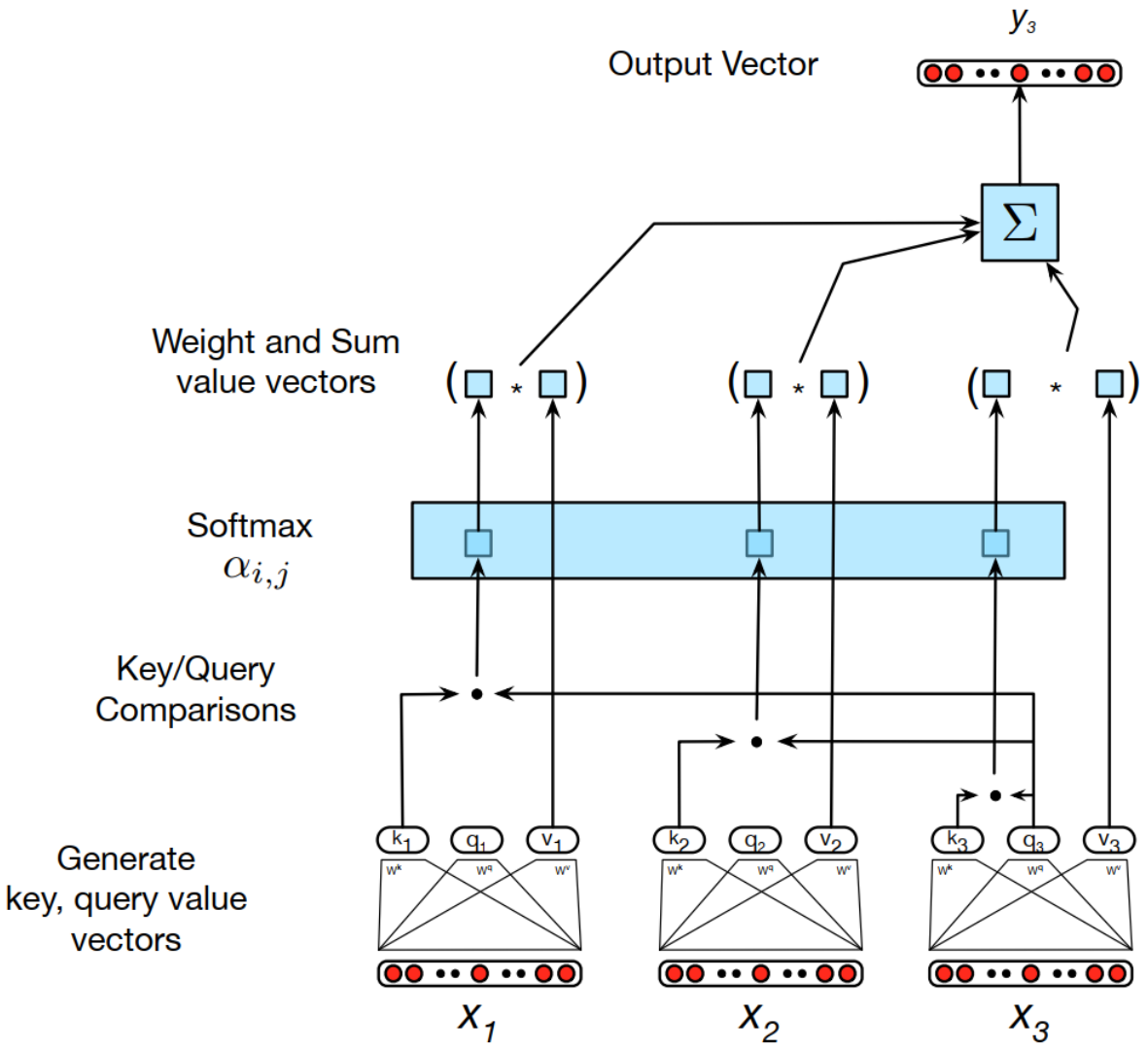
Self Attention





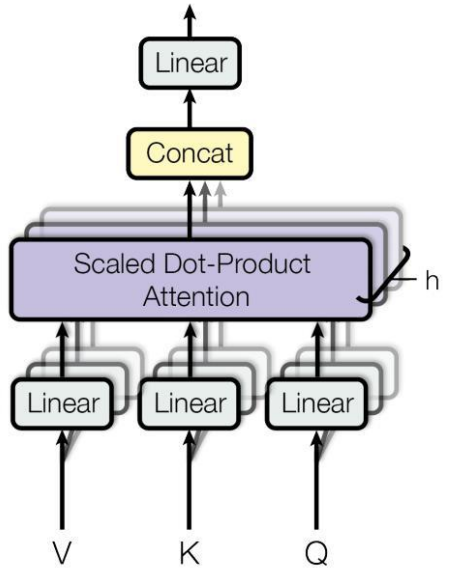
$$\text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right) V = Z$$

Self Attention



Multihead Attention

- The different words in a sentence can relate to each other in many different ways simultaneously.
  - Distinct syntactic, semantic, and discourse relationships
- It would be difficult for a single transformer block to learn to capture all of the different kinds of parallel relations among its inputs.
- Transformers address this issue with multihead self-attention layers.
- These are sets of self-attention layers, called heads, that reside in parallel layers at the same depth in a model, each with its own set of parameters.
- Given these distinct sets of parameters, each head can learn different aspects of the relationships that exist among inputs at the same level of abstraction.



Multihead Attention

1) This is our input sentence*

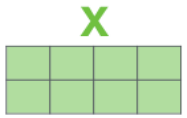
2) We embed each word*

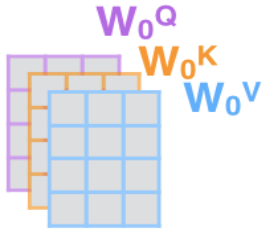
3) Split into 8 heads. We multiply X or R with weight matrices

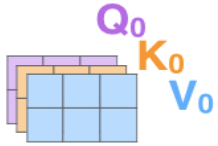
4) Calculate attention using the resulting Q/K/V matrices

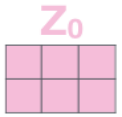
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

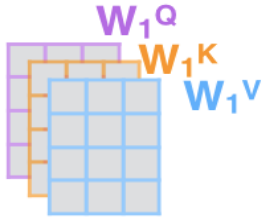
Thinking Machines

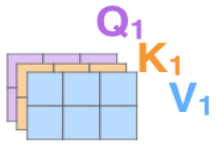


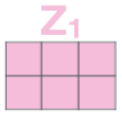


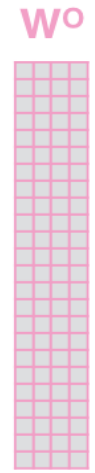


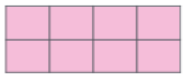




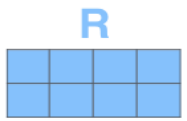


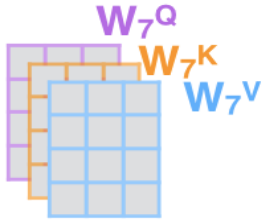


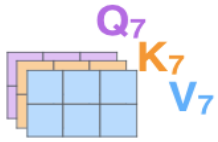


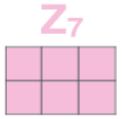


* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

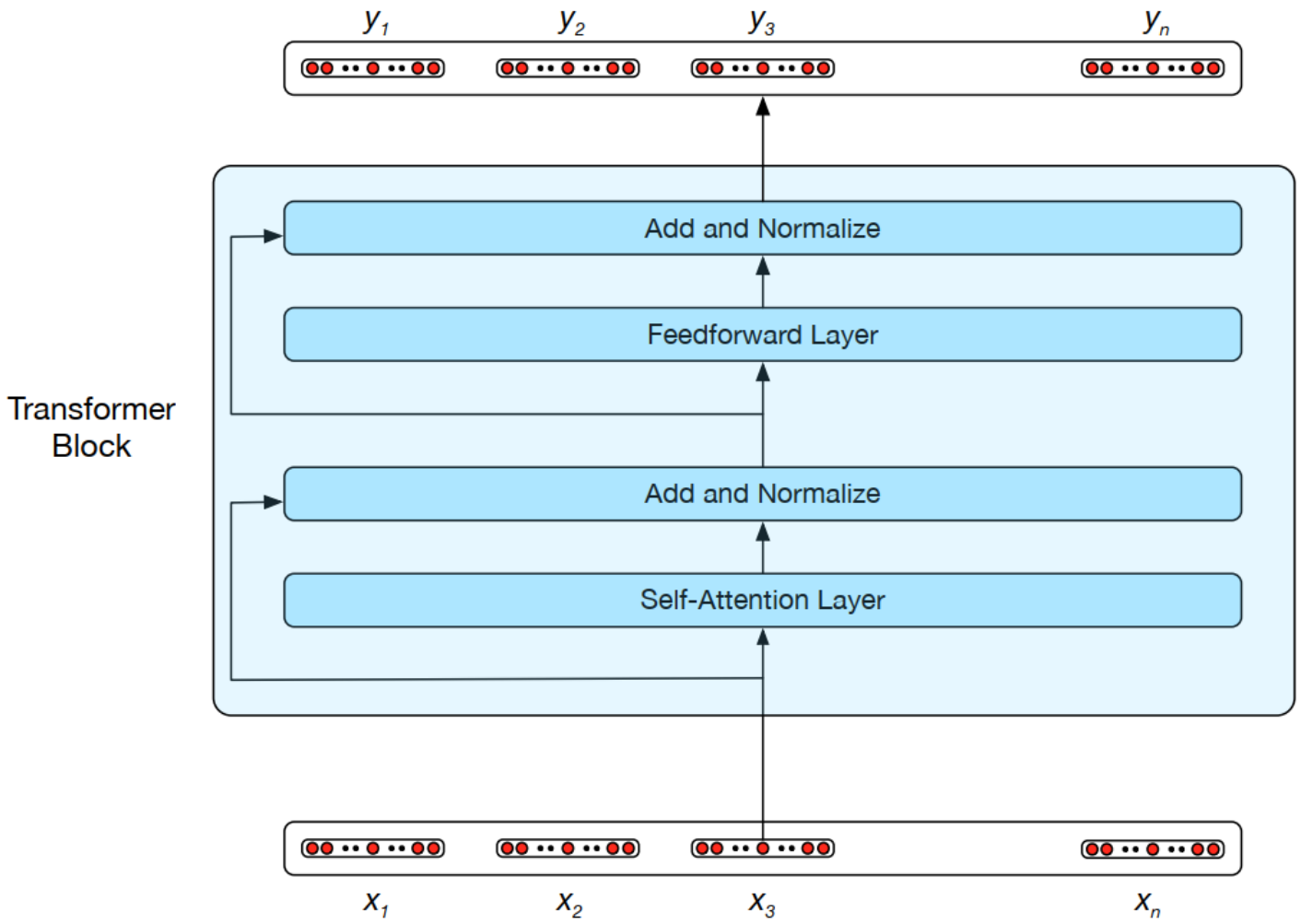




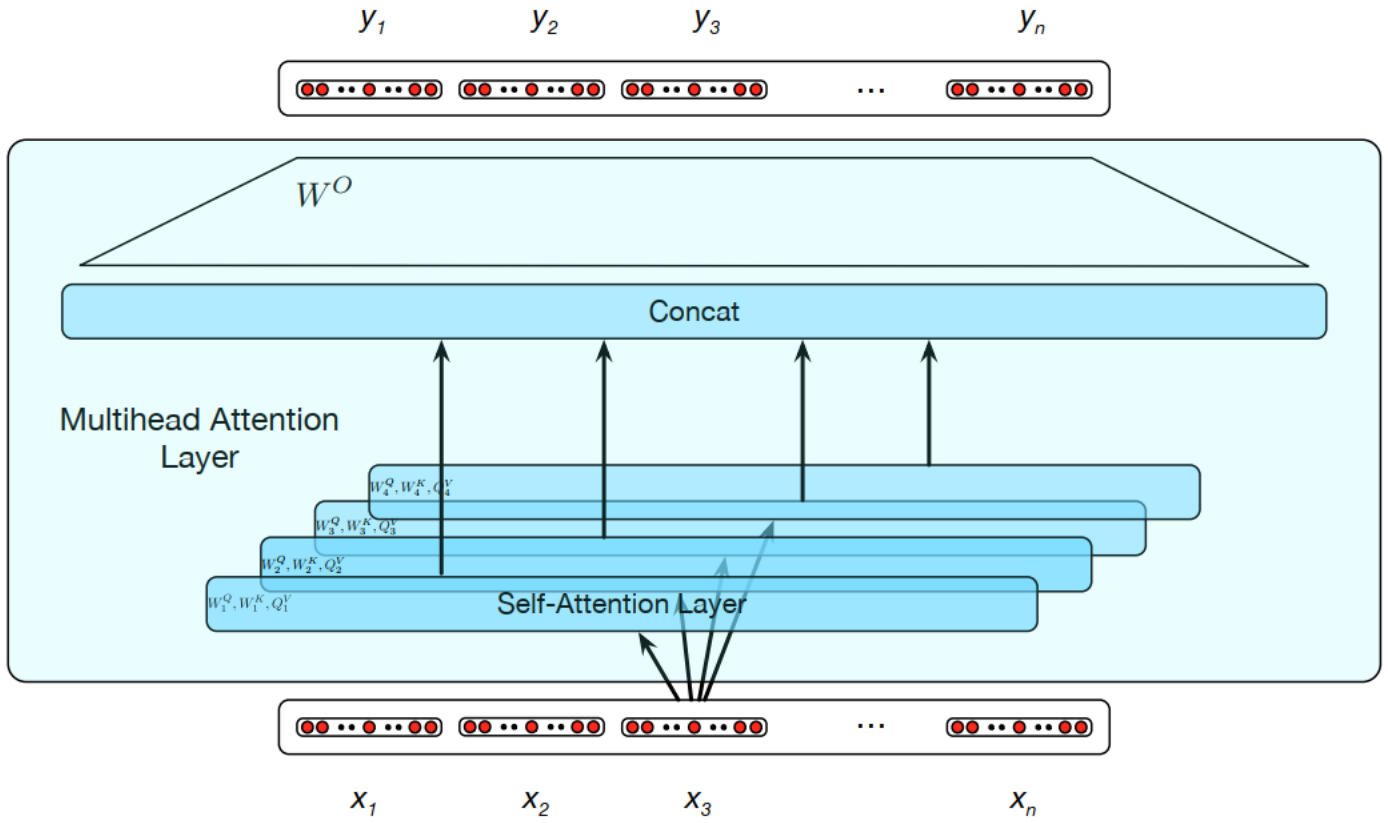




Multihead Attention

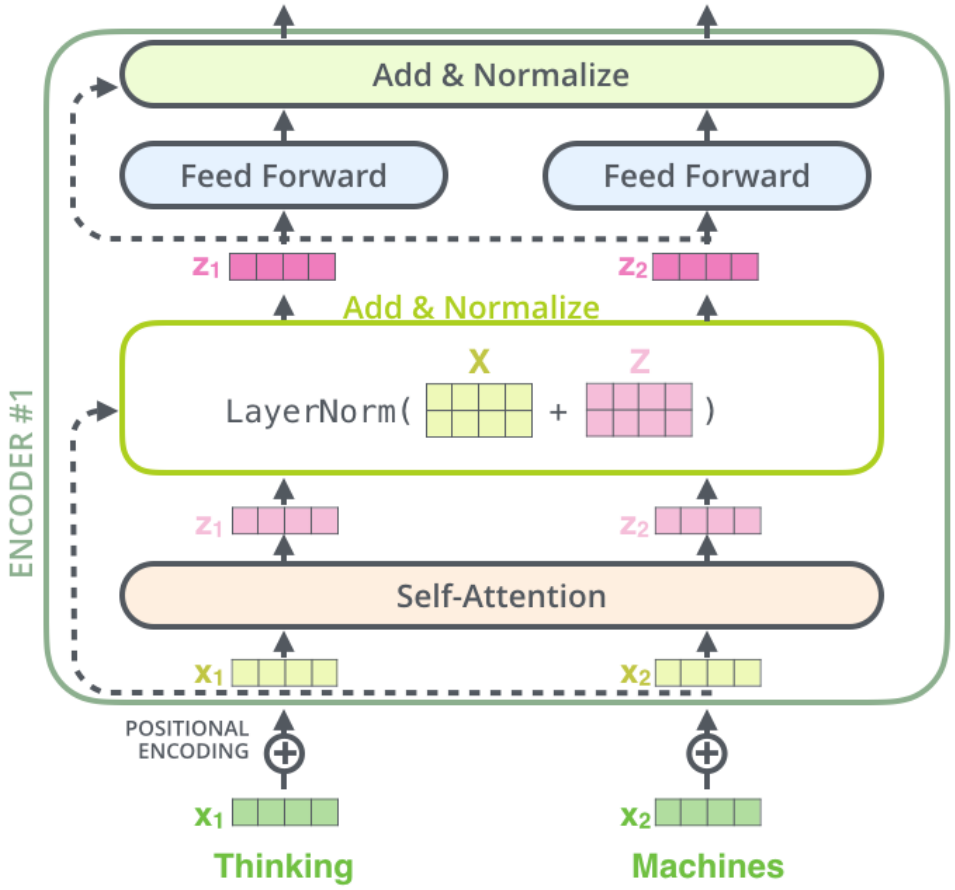


Multihead Attention



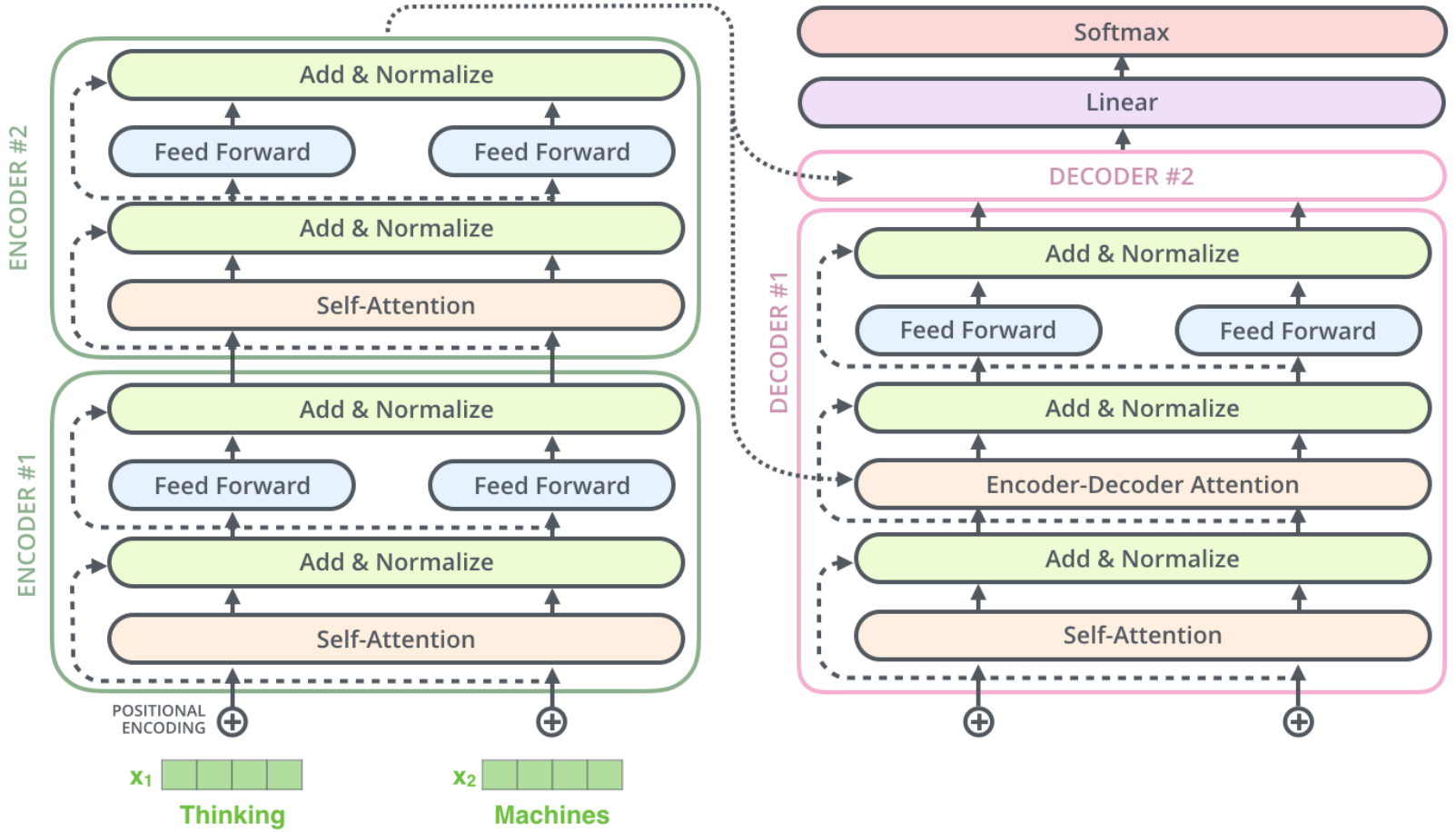
Encoder and Decoder in Transformers

- Encoder Structure



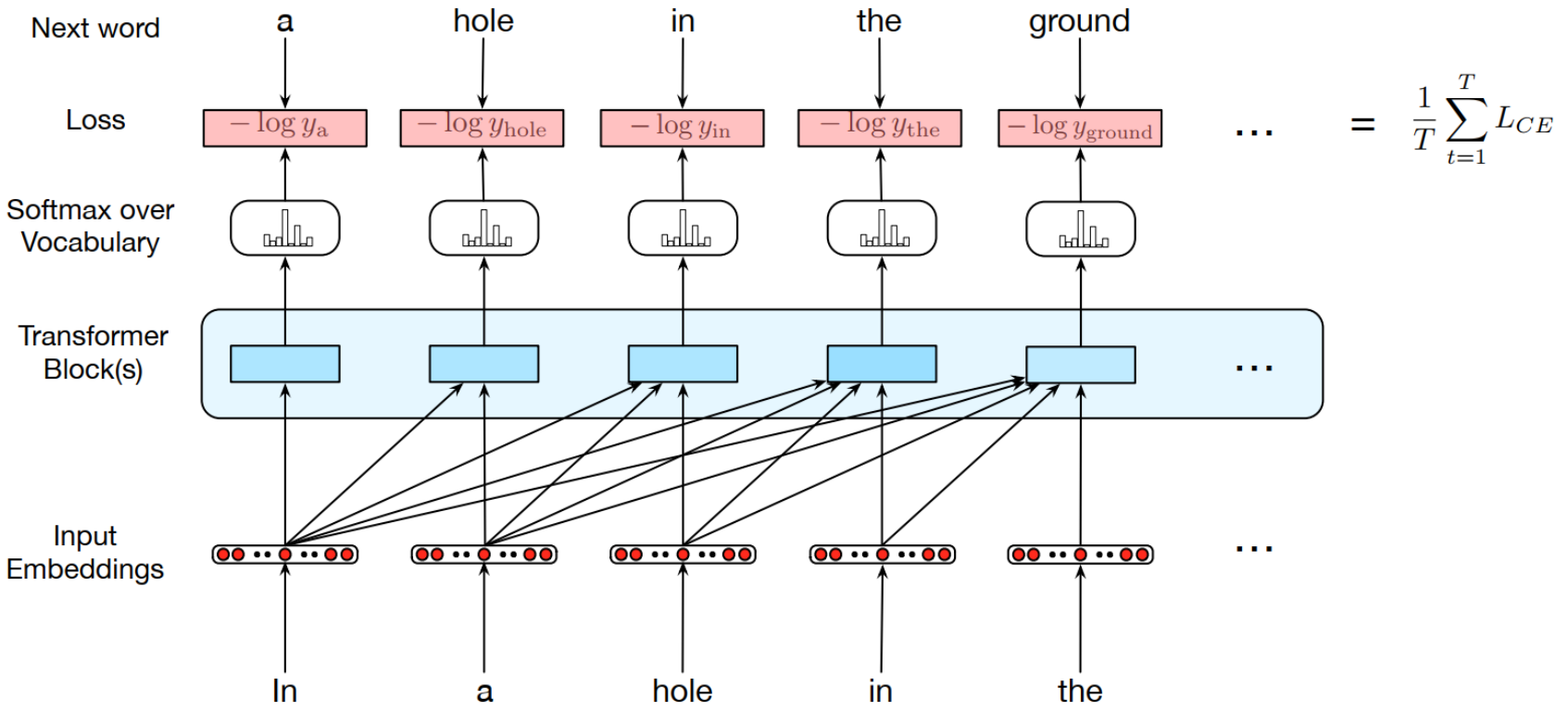
Encoder and Decoder in Transformers

- Decoder Structure



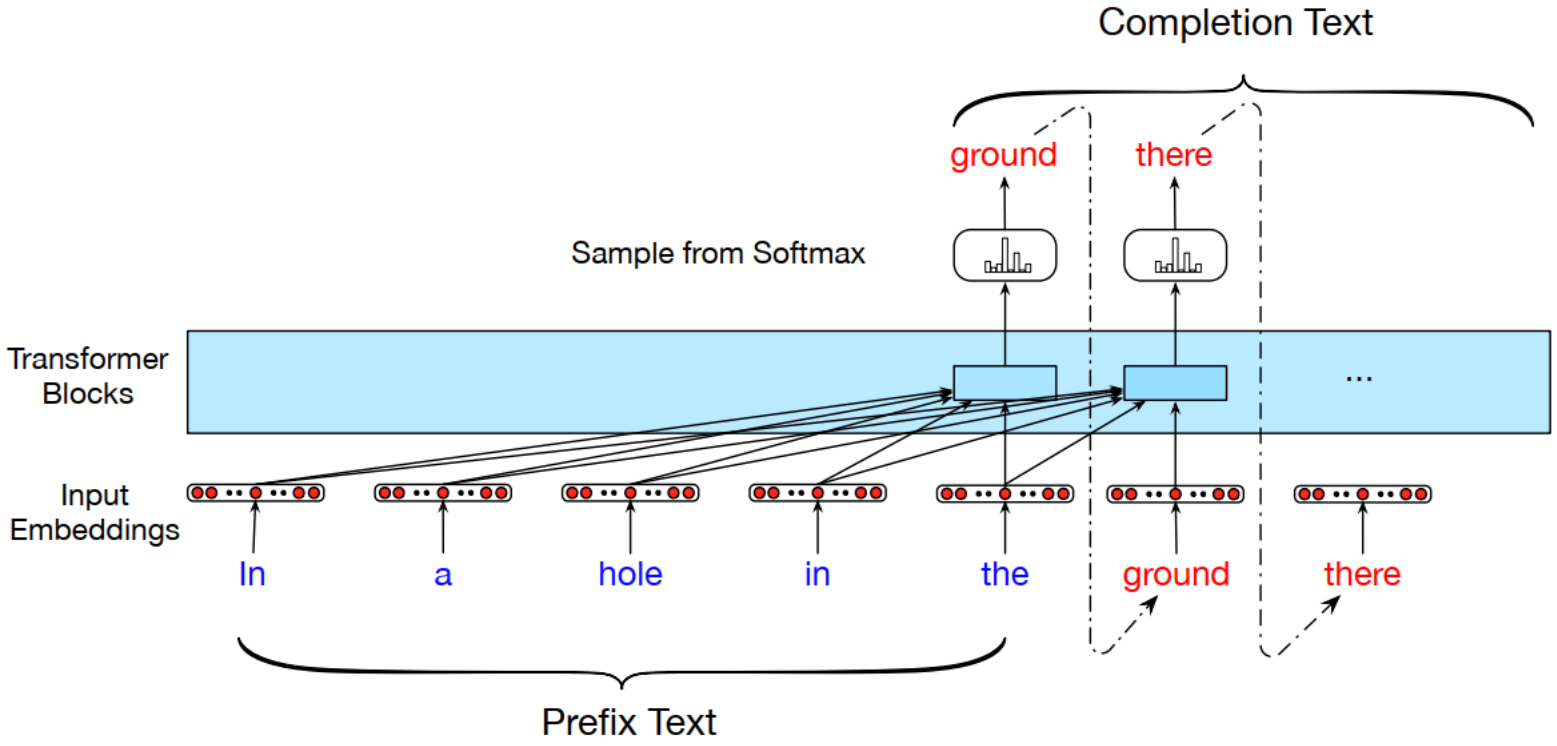
Transformers as Autoregressive LMs

- Training a Transformer as a language model



Transformers as Autoregressive LMs

- Autoregressive text completion with Transformers



Other Applications of Transformers

- Context Generation
- Text Summarization

Text Summarization with Transformers

Original Article

The only thing crazier than a guy in snowbound Massachusetts boxing up the powdery white stuff and offering it for sale online? People are actually buying it. For $89, self-styled entrepreneur Kyle Waring will ship you 6 pounds of Boston-area snow in an insulated Styrofoam box – enough for 10 to 15 snowballs, he says.

But not if you live in New England or surrounding states. "We will not ship snow to any states in the northeast!" says Waring's website, ShipSnowYo.com. "We're in the business of expunging snow!"

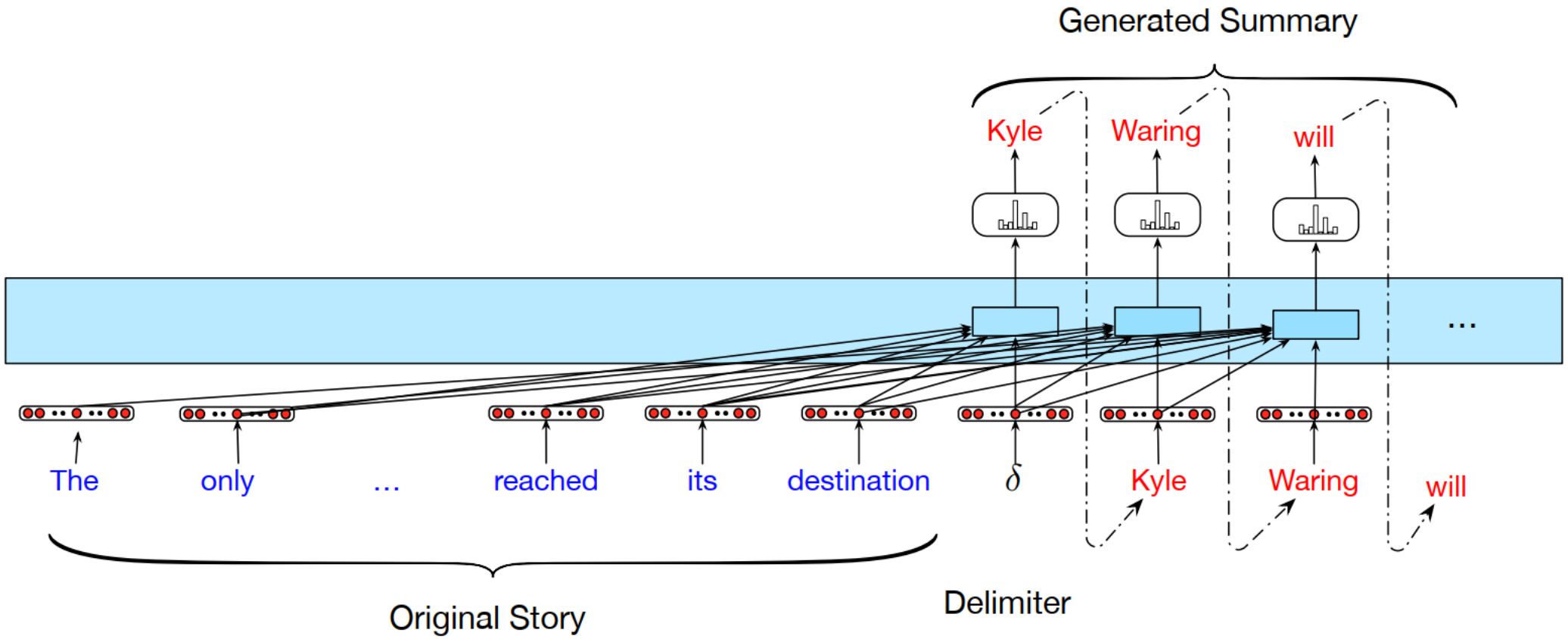
His website and social media accounts claim to have filled more than 133 orders for snow – more than 30 on Tuesday alone, his busiest day yet. With more than 45 total inches, Boston has set a record this winter for the snowiest month in its history. Most residents see the huge piles of snow choking their yards and sidewalks as a nuisance, but Waring saw an opportunity.

According to Boston.com, it all started a few weeks ago, when Waring and his wife were shoveling deep snow from their yard in Manchester-by-the-Sea, a coastal suburb north of Boston. He joked about shipping the stuff to friends and family in warmer states, and an idea was born. His business slogan: "Our nightmare is your dream!" At first, ShipSnowYo sold snow packed into empty 16.9-ounce water bottles for $19.99, but the snow usually melted before it reached its destination...

Summary

Kyle Waring will ship you 6 pounds of Boston-area snow in an insulated Styrofoam box – enough for 10 to 15 snowballs, he says. But not if you live in New England or surrounding states.

Text Summarization with Transformers



Further Reading

- Speech and Language Processing (3rd ed. draft)
  - Chapter 9 & 10