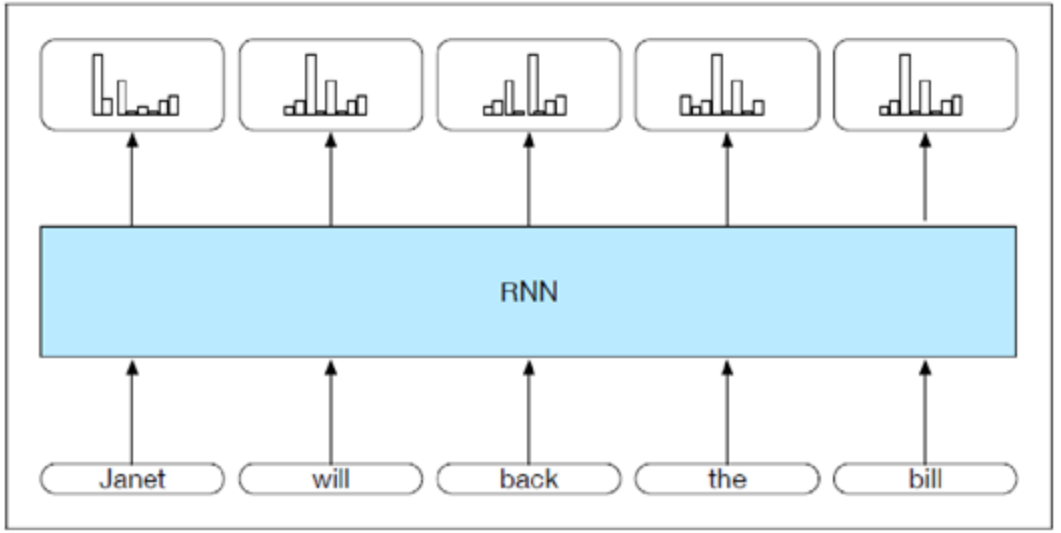
- **Encoder Decoder**
- Attention
- Transformers

Encoder-Decoder

- **RNN:** input sequence is transformed into output sequence in a one-to-one fashion



- **Goal:** Develop an architecture capable of generating *contextually appropriate*, *arbitrary length*, output sequences
- **Applications**:
  - Machine translation,
  - Summarization,
  - Question answering,
  - Dialogue modeling.

Simple recurrent neural network illustrated as a feed-forward network

Most significant change: new set of weights, U

- connect the hidden layer from the previous time step to the current hidden layer.
- determine how the network should make use of past context in calculating the output for the current input.

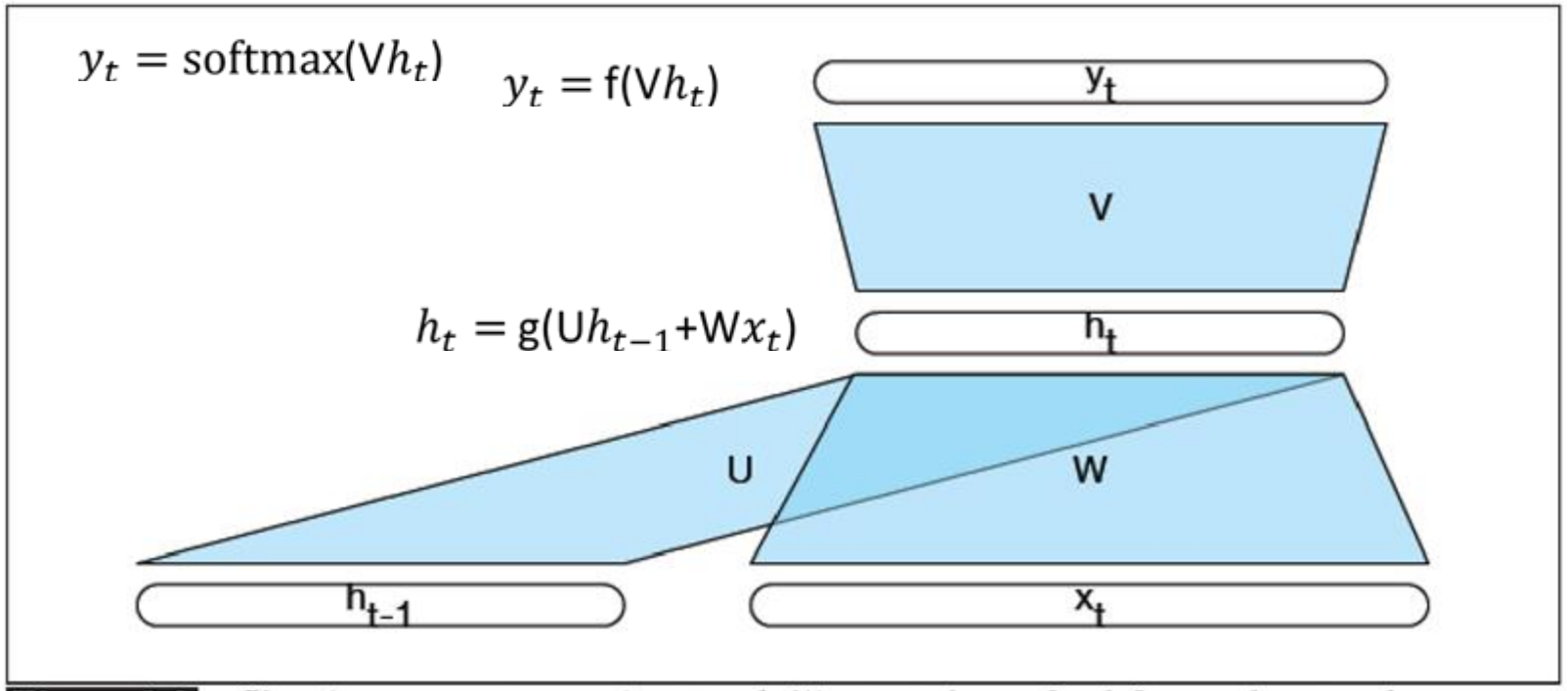
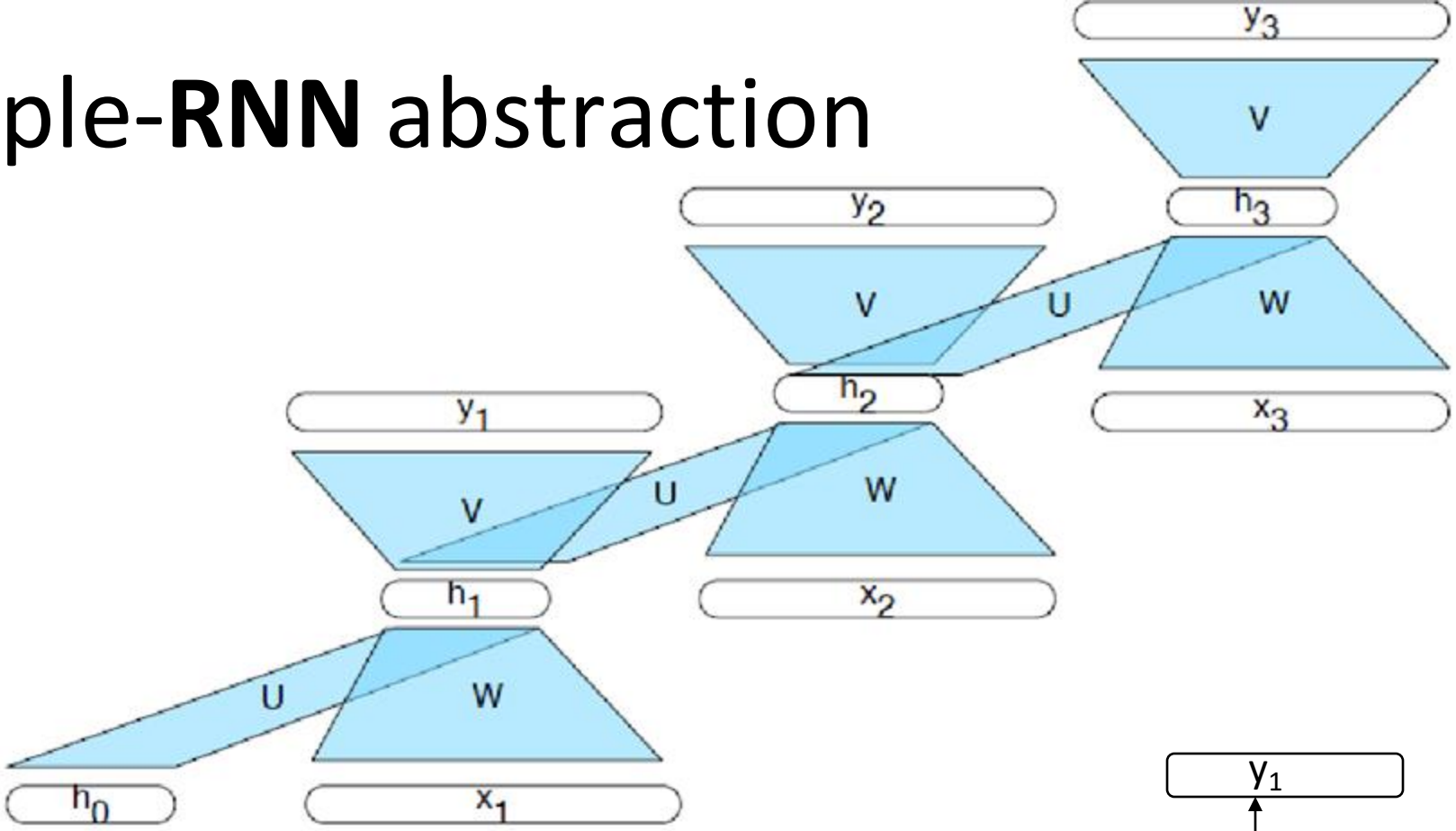
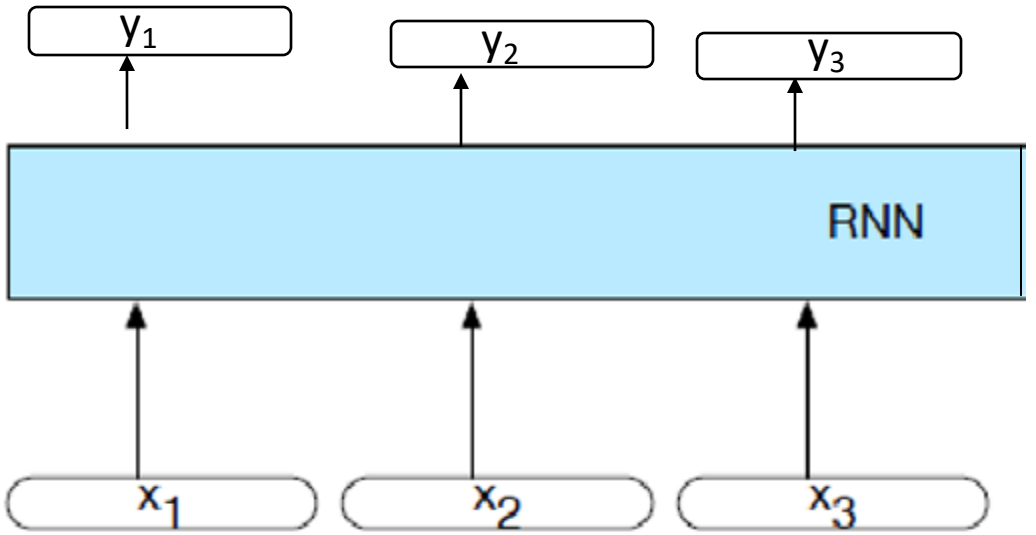


Figure 9.3 Simple recurrent neural network illustrated as a feed-forward network.

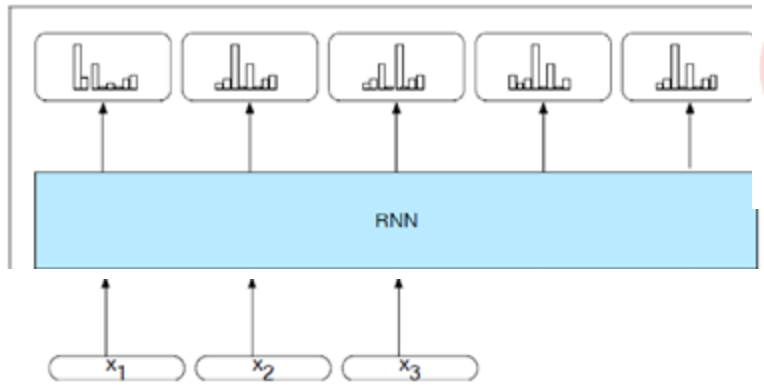
Simple-**RNN** abstraction

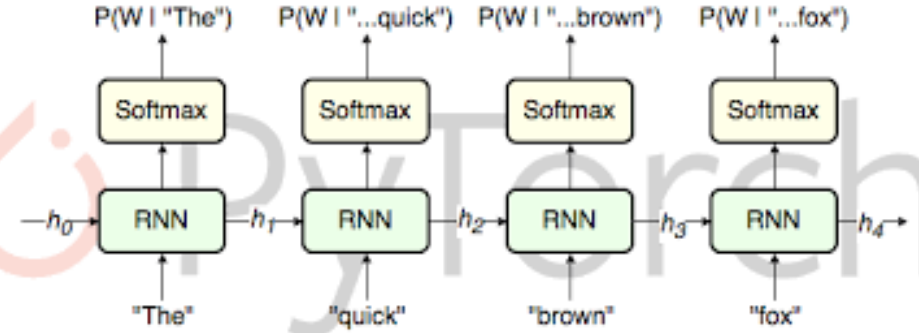




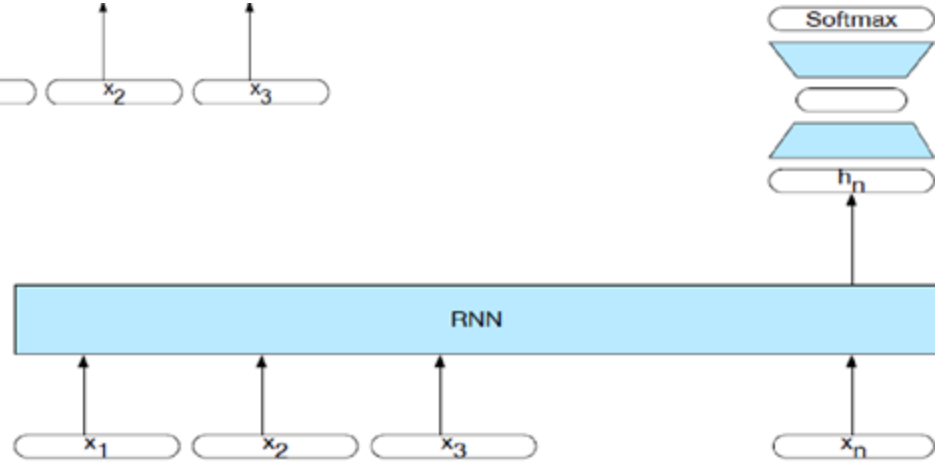
RNN Applications

- Language Modeling

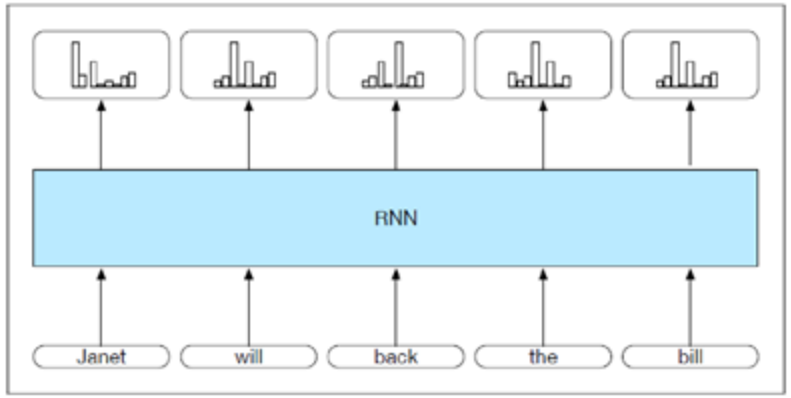




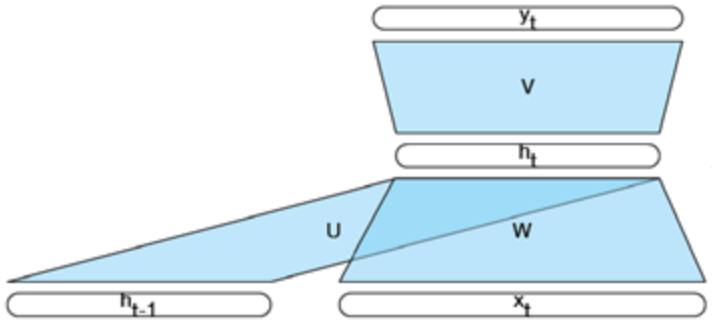
- Sequence Classification (Sentiment, Topic)



- Sequence to Sequence

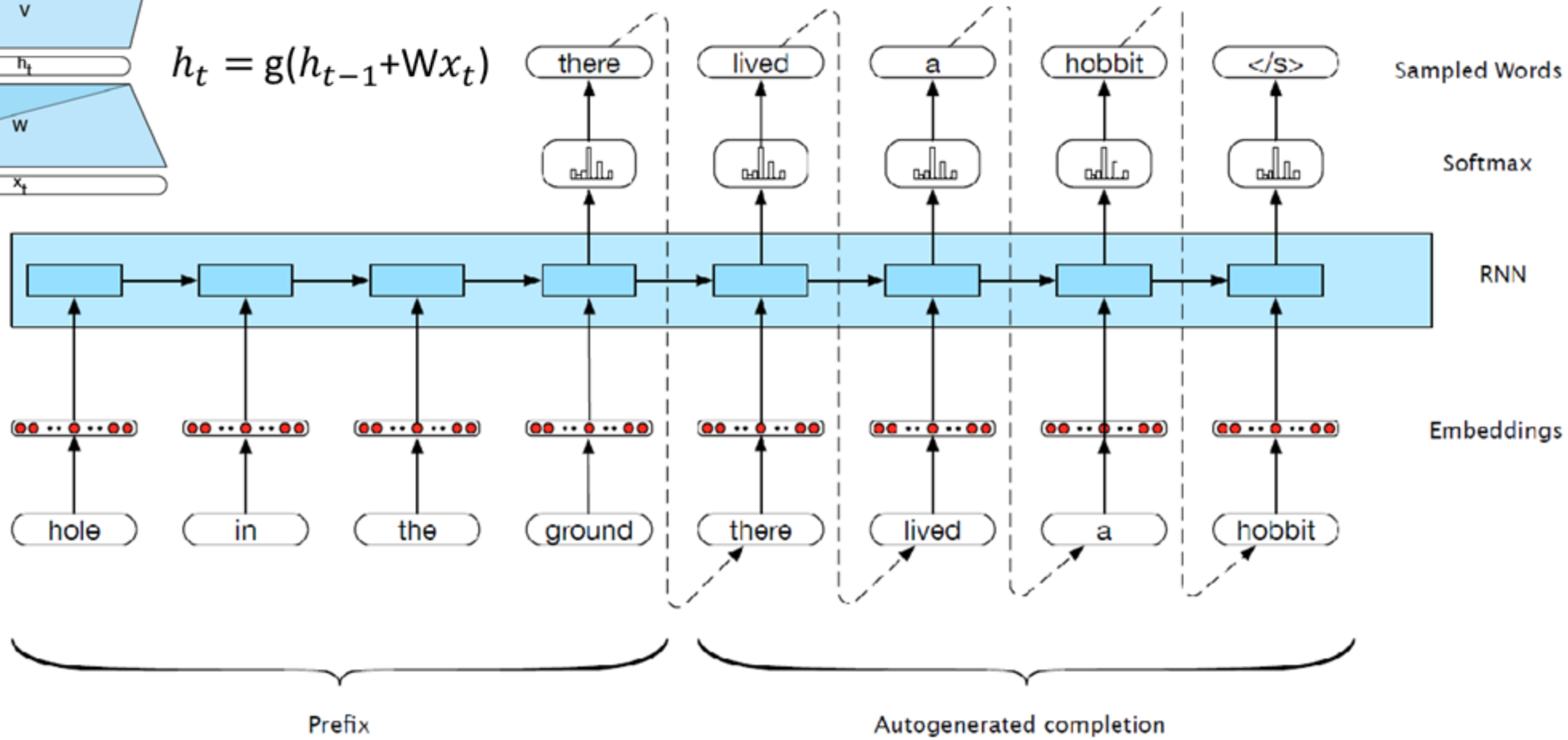


Sentence Completion using an RNN



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

$$h_t = g(h_{t-1} + Wx_t)$$



- **Trained Neural Language Model** can be used to generate novel sequences
- Or to **complete** a given sequence (until end of sentence token <\s> is generated)

Extending (autoregressive) generation to Machine Translation

word generated at each time step is conditioned on word from previous step.

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

  there lived a hobbit *vivait un hobbit*

  ……..

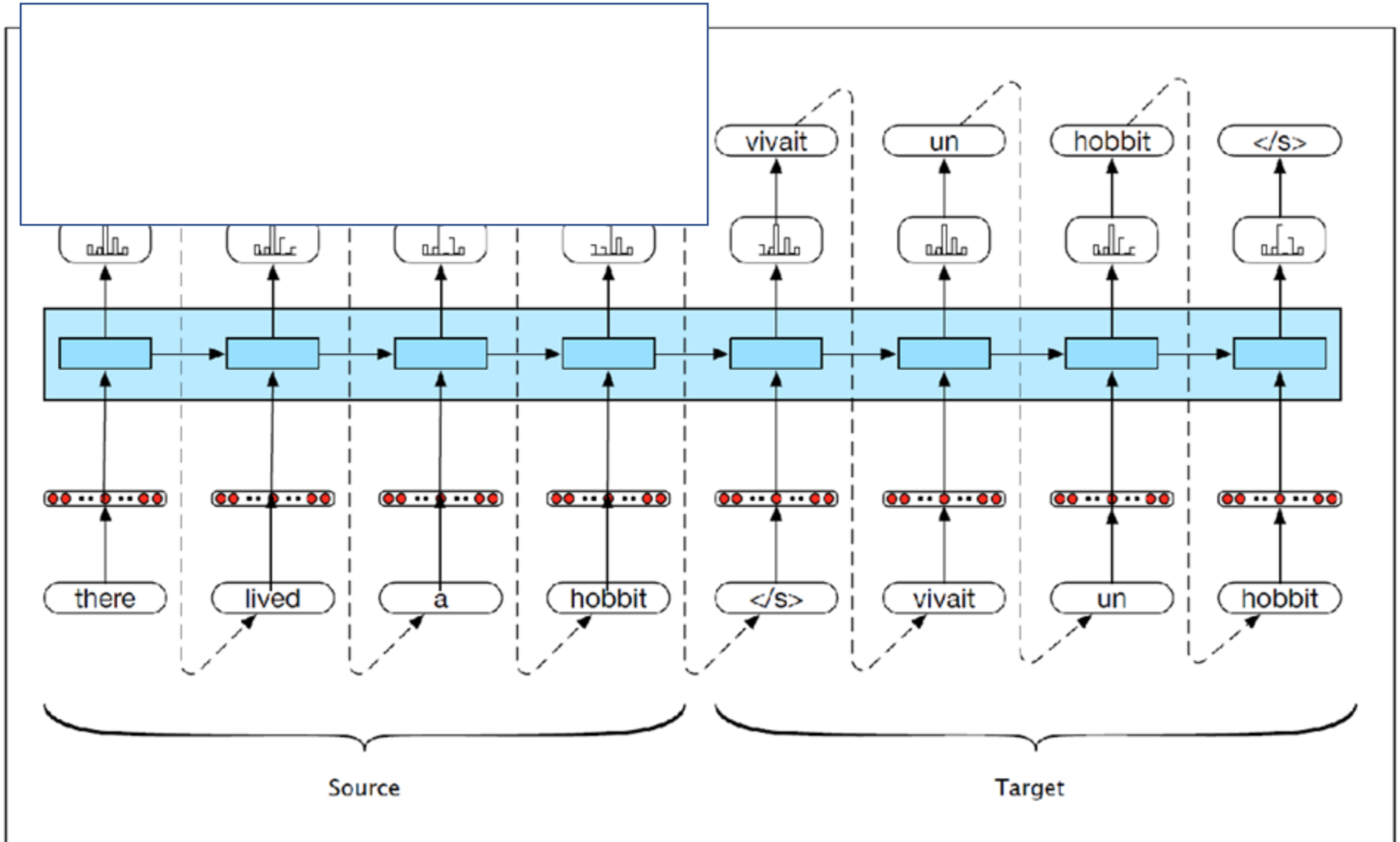
- Build an RNN language model on the concatenation of source and target

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

  ……..

Extending (autoregressive) generation to Machine Translation

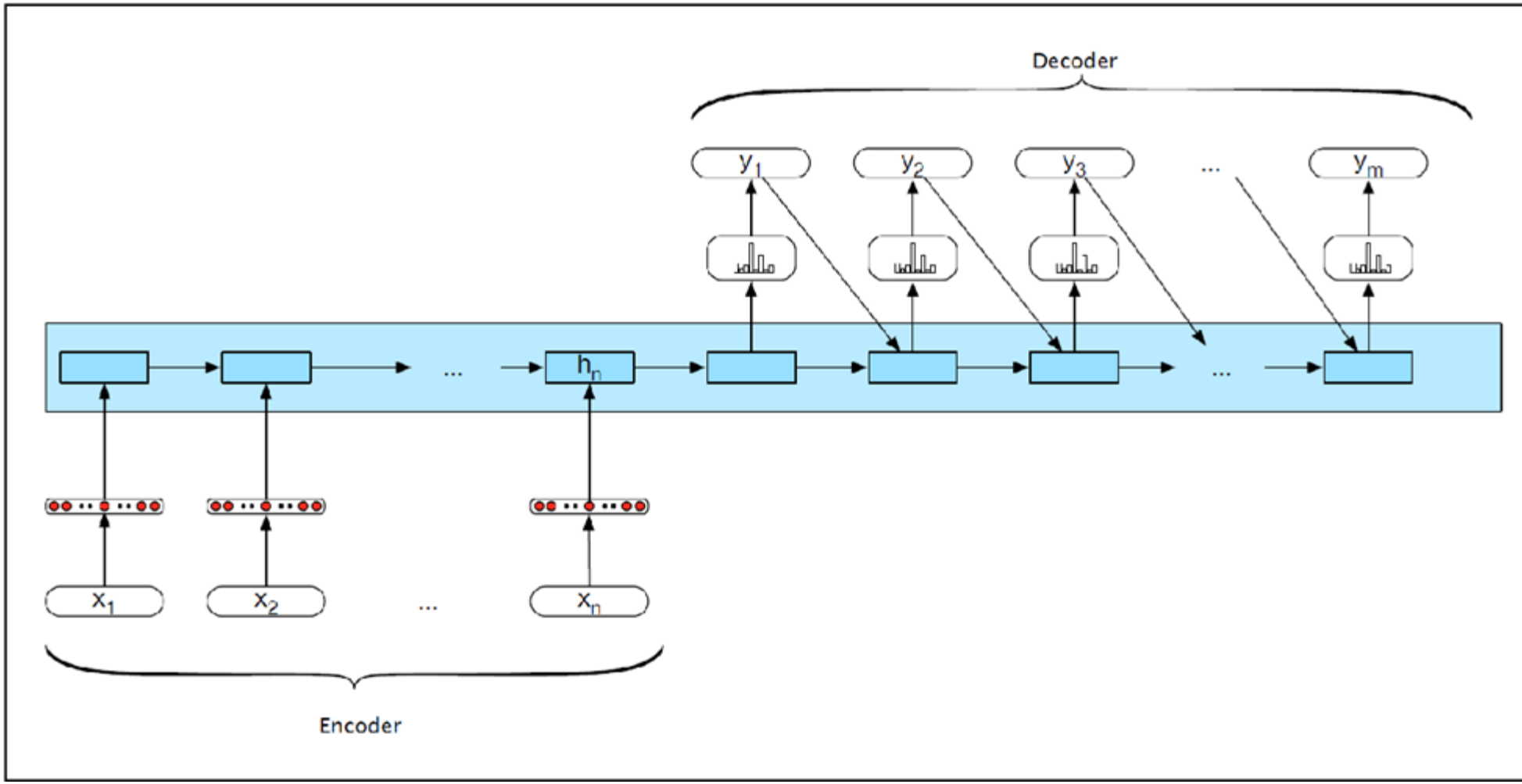
- Translation as Sentence Completion !



(simple) **E**ncoder **D**ecoder 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.

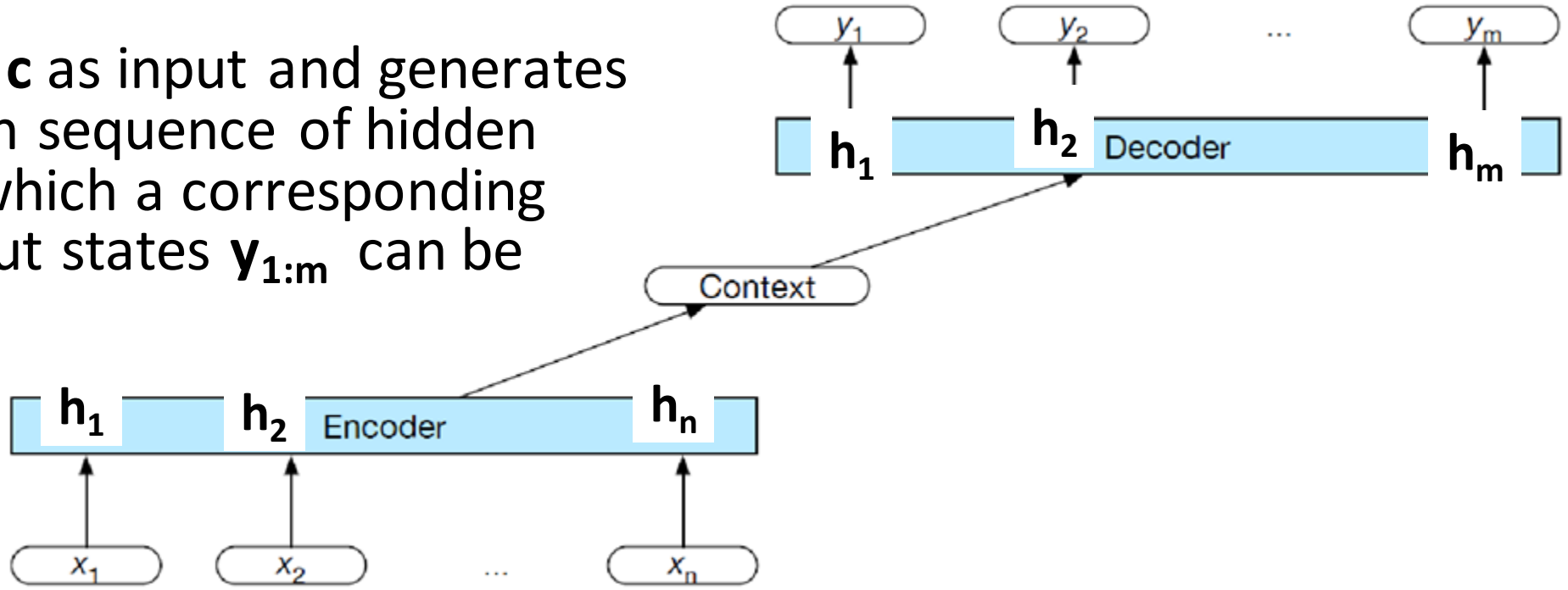


- Encoder generates a contextualized representation of the input (last state).
- Decoder takes that state and autoregressively generates a sequence of outputs

General Encoder Decoder Networks

Abstracting away from these choices

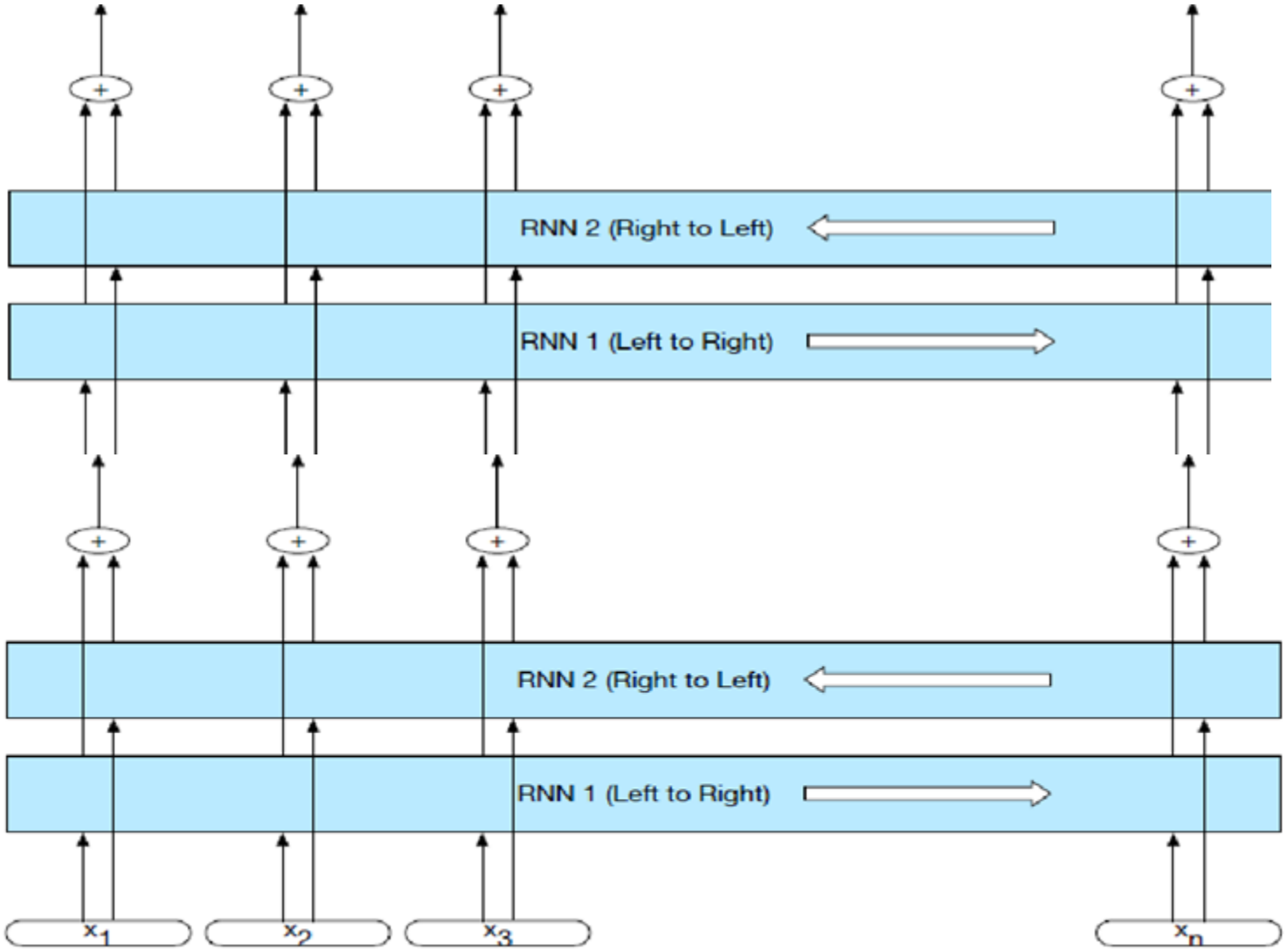
1. **Encoder**: accepts an input sequence, $\mathbf{x_{1:n}}$ and generates a corresponding sequence of contextualized representations, $\mathbf{h_{1:n}}$
2. **Context vector c**: function of $\mathbf{h_{1:n}}$ and conveys the essence of the input to the decoder.
3. **Decoder**: accepts **c** as input 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**

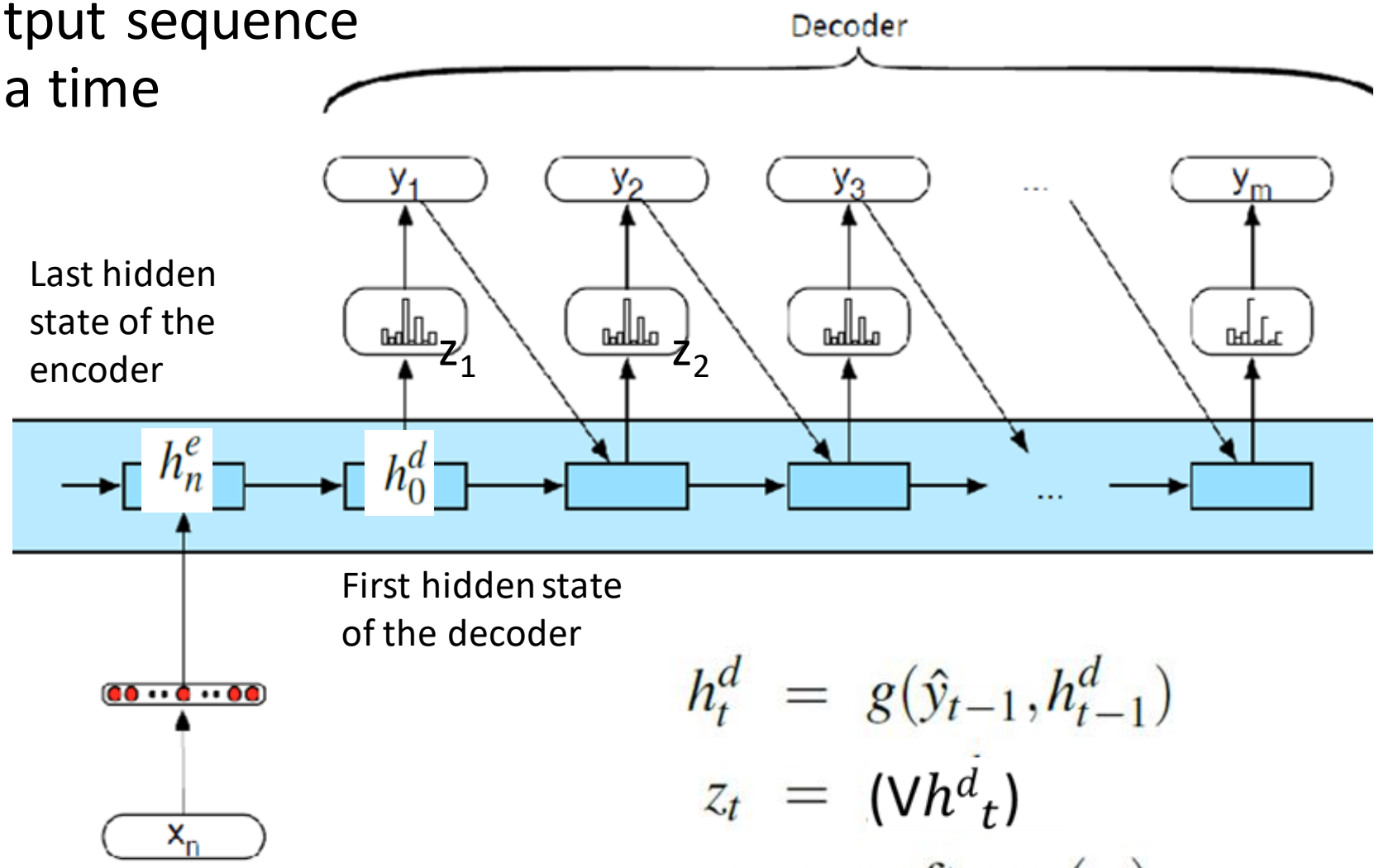
- Contextualized representations for each time step**: hidden states from top layers** from the forward and backward passes



Decoder Basic Design

- produce an output sequence an element at a time

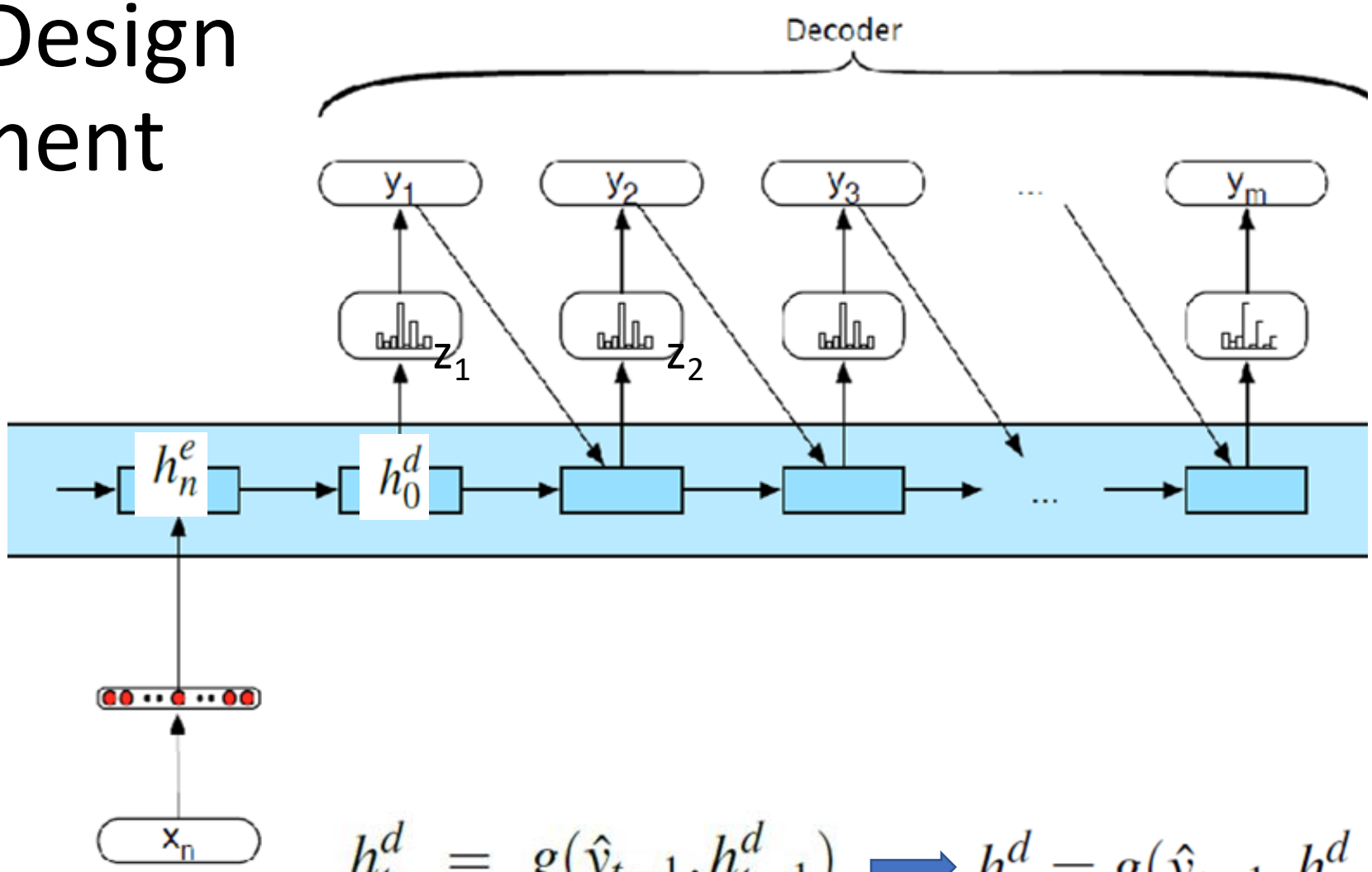
$$c = h_n^e$$
$$h_0^d = c$$



$$h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d)$$
$$z_t = (Vh^d{}_t)$$
$$y_t = \text{softmax}(z_t)$$

Decoder Design Enhancement

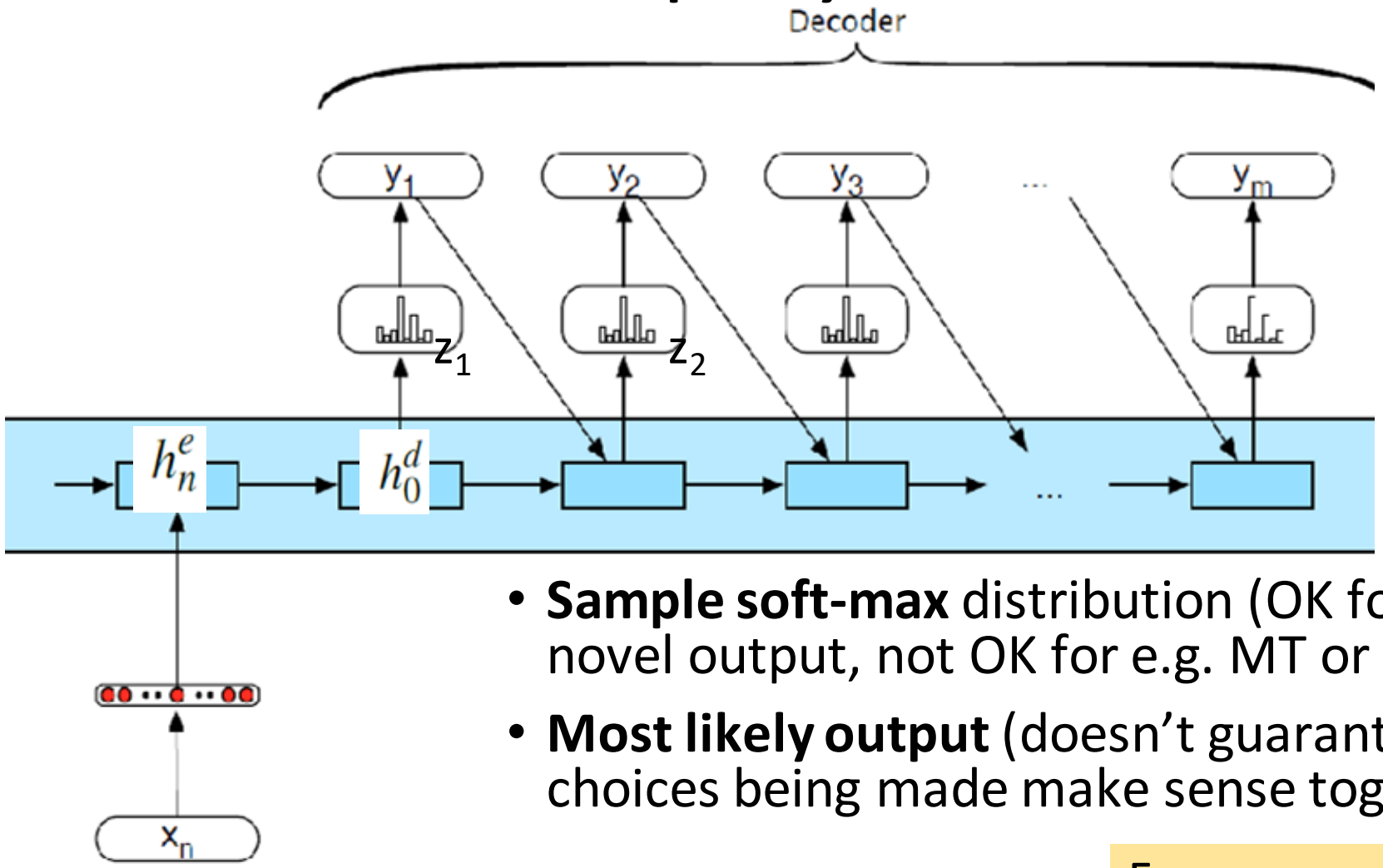
$$c = h_n^e$$
$$h_0^d = c$$



$$h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d) \rightarrow h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d, c)$$
$$z_t = f(h_t^d)$$
$$y_t = \text{softmax}(z_t)$$

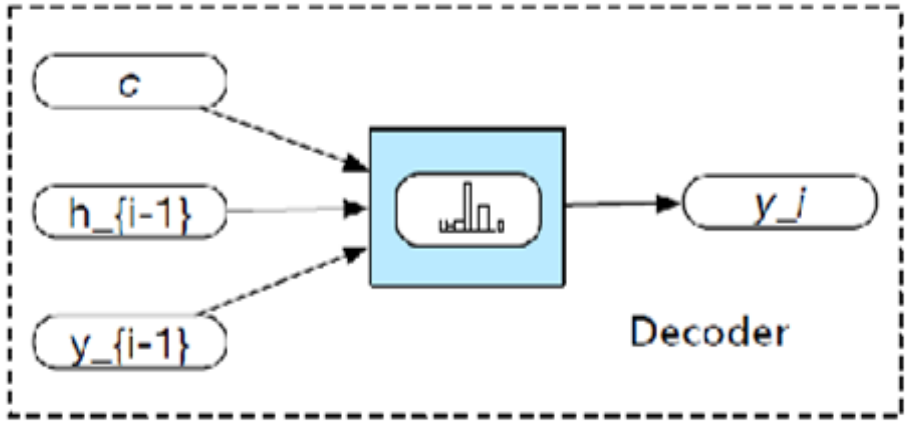
Context available at each step of decoding

Decoder: How output *y* is chosen

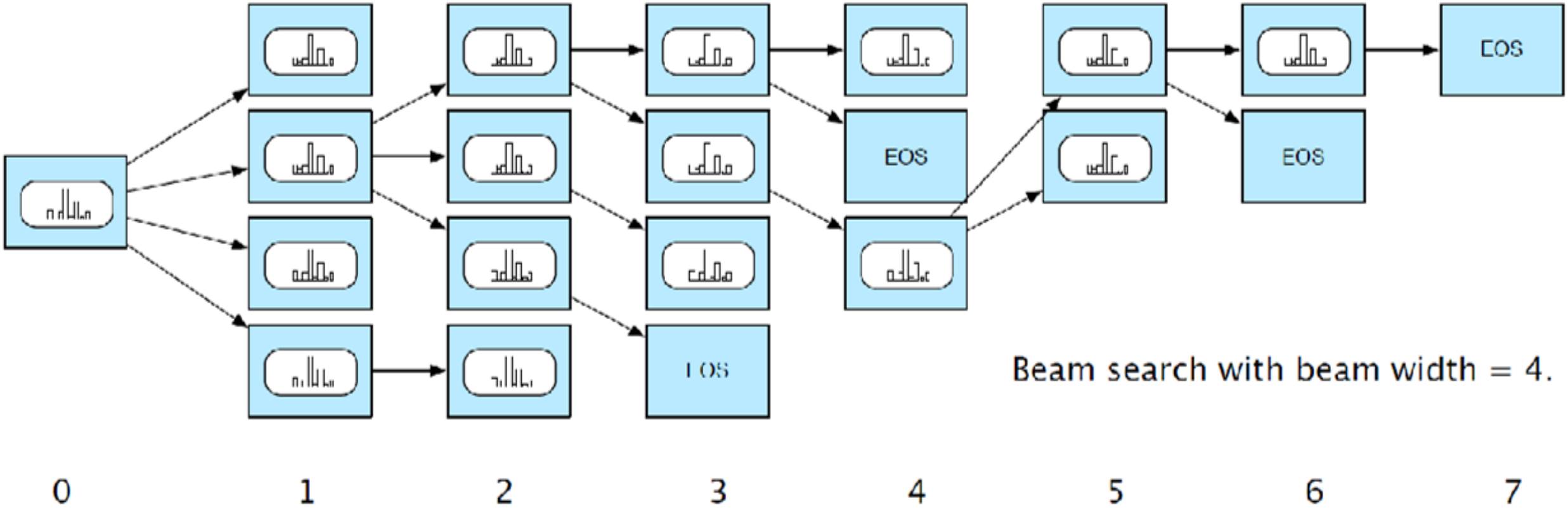


- **Sample soft-max** distribution (OK for generating novel output, not OK for e.g. MT or Summ)
- **Most likely output** (doesn't guarantee individual choices being made make sense together)

For sequence labeling we used **Viterbi** – here not possible ☹



- 4 most likely "words" decoded from initial state
- Feed each of those in decoder and keep most likely 4 sequences of two words
- Feed most recent word in decoder and keep most likely 4 sequences of three words …….
- When EOS is generated. Stop sequence and reduce Beam by 1



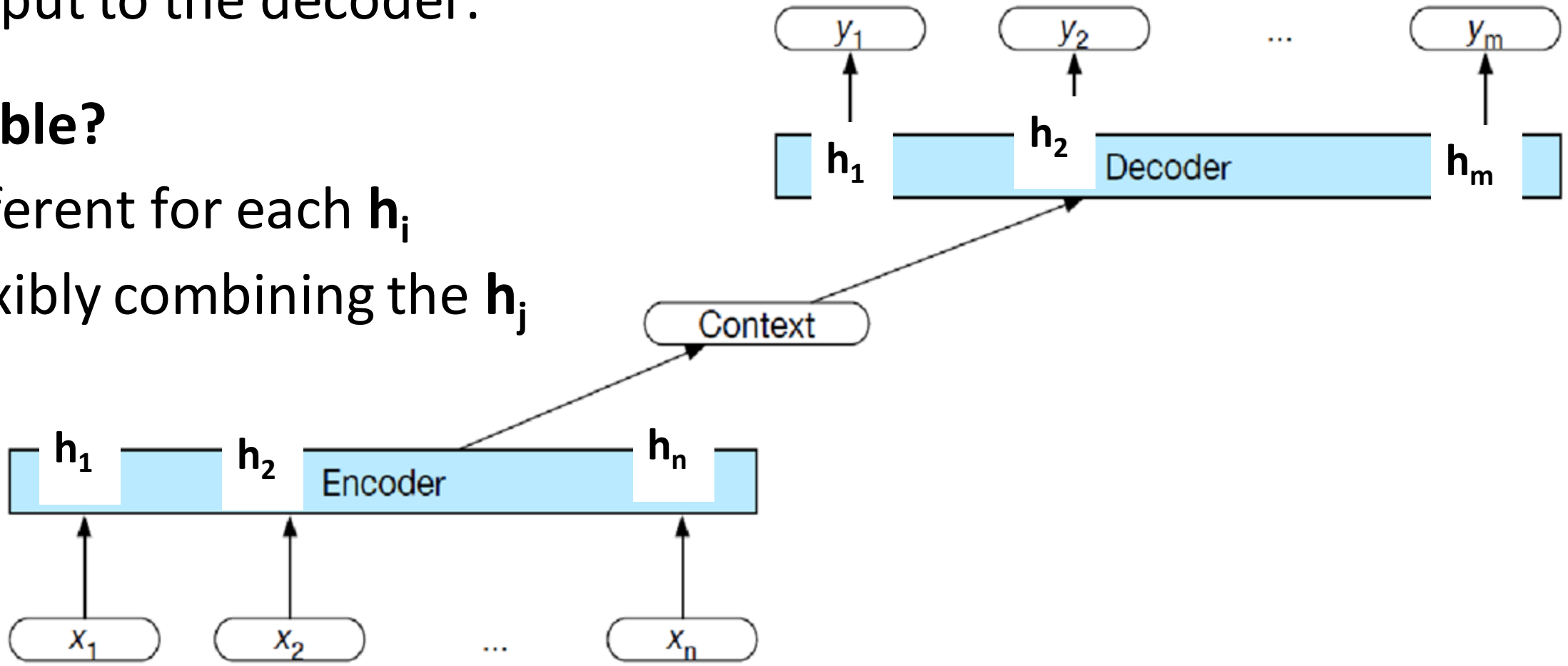
- Encoder Decoder
- **Attention**
- Transformers

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 (1): dynamically derived context

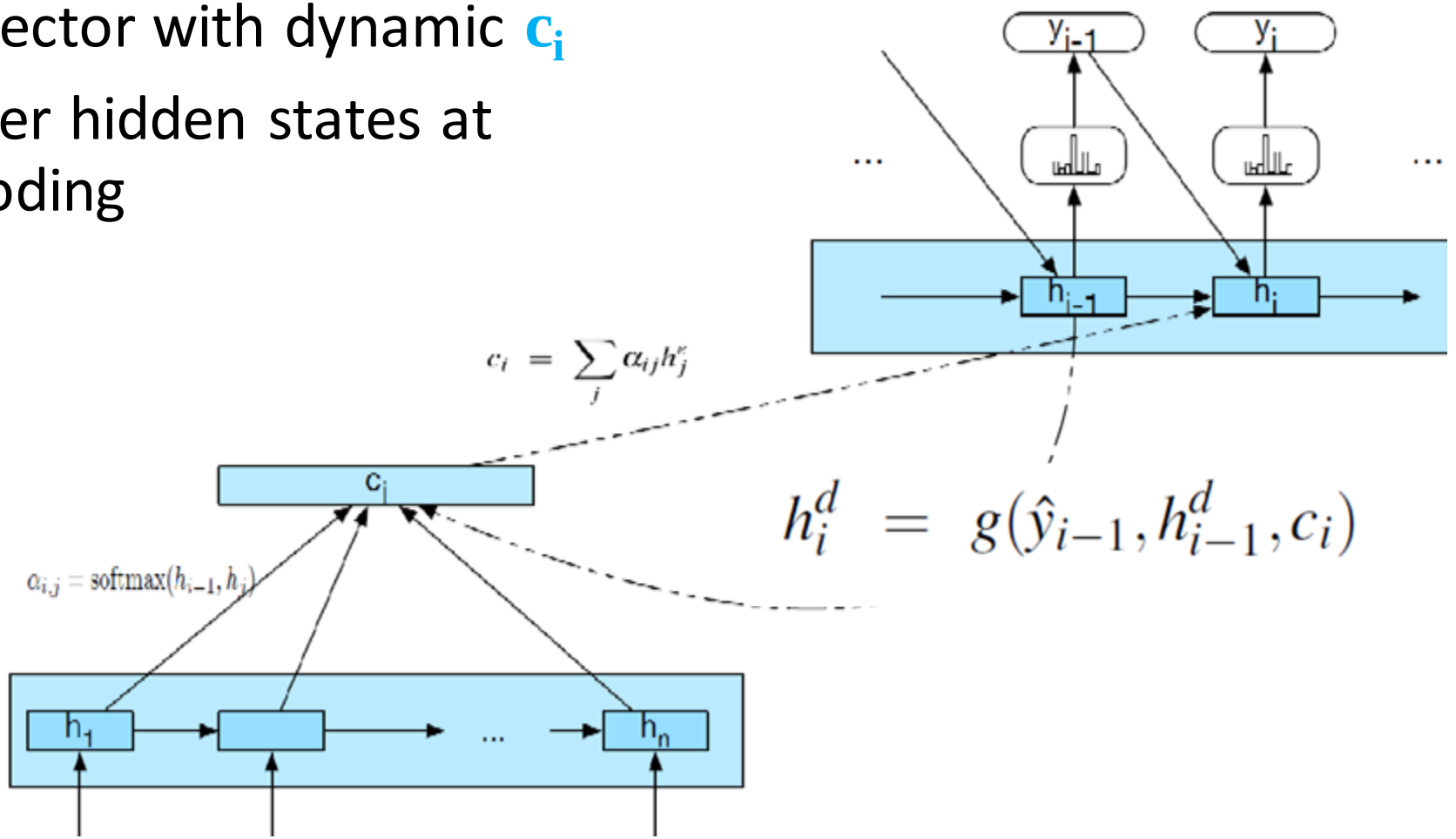
- Replace static context vector with dynamic $\mathbf{c_i}$
- derived from the encoder hidden states at each point i during decoding

Ideas:

- should be a linear combination of those states

$$c_i = \sum_j \alpha_{ij} h_j^e$$

- α_{ij} should depend on ?



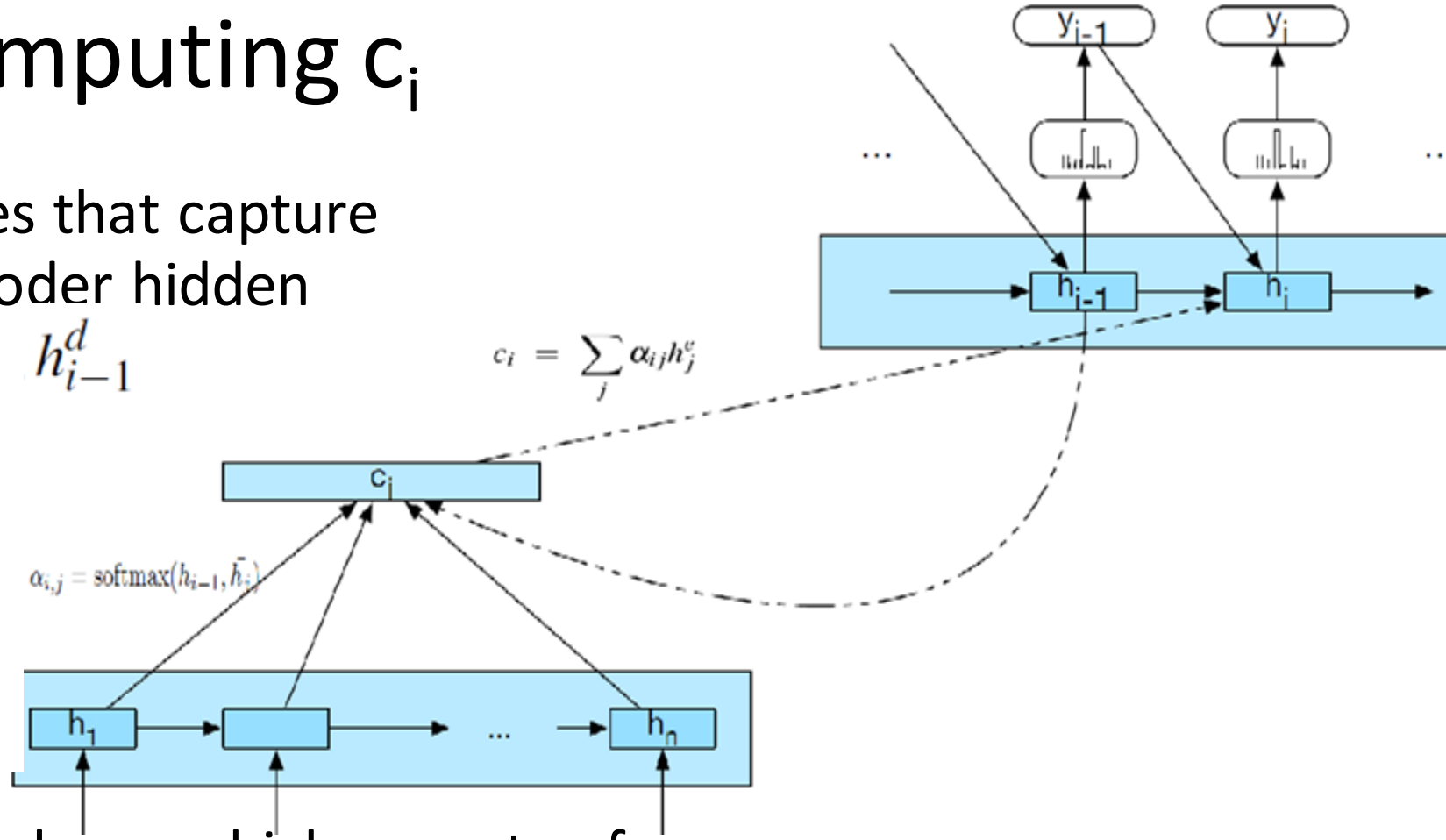
Attention (2): computing c_i

- Compute a vector of scores that capture the relevance of each encoder hidden state to the decoder state h_{i-1}^d

$$score(h_{i-1}^d, h_j^e)$$

- Just the similarity

$$score(h_{i-1}^d, h_j^e) = h_{i-1}^d \cdot h_j^e$$



- Give network the ability to learn which aspects of similarity between the decoder and encoder states are important to the current application.

$$score(h_{i-1}^d, h_j^e) = h_{t-1}^d W_s h_j^e$$

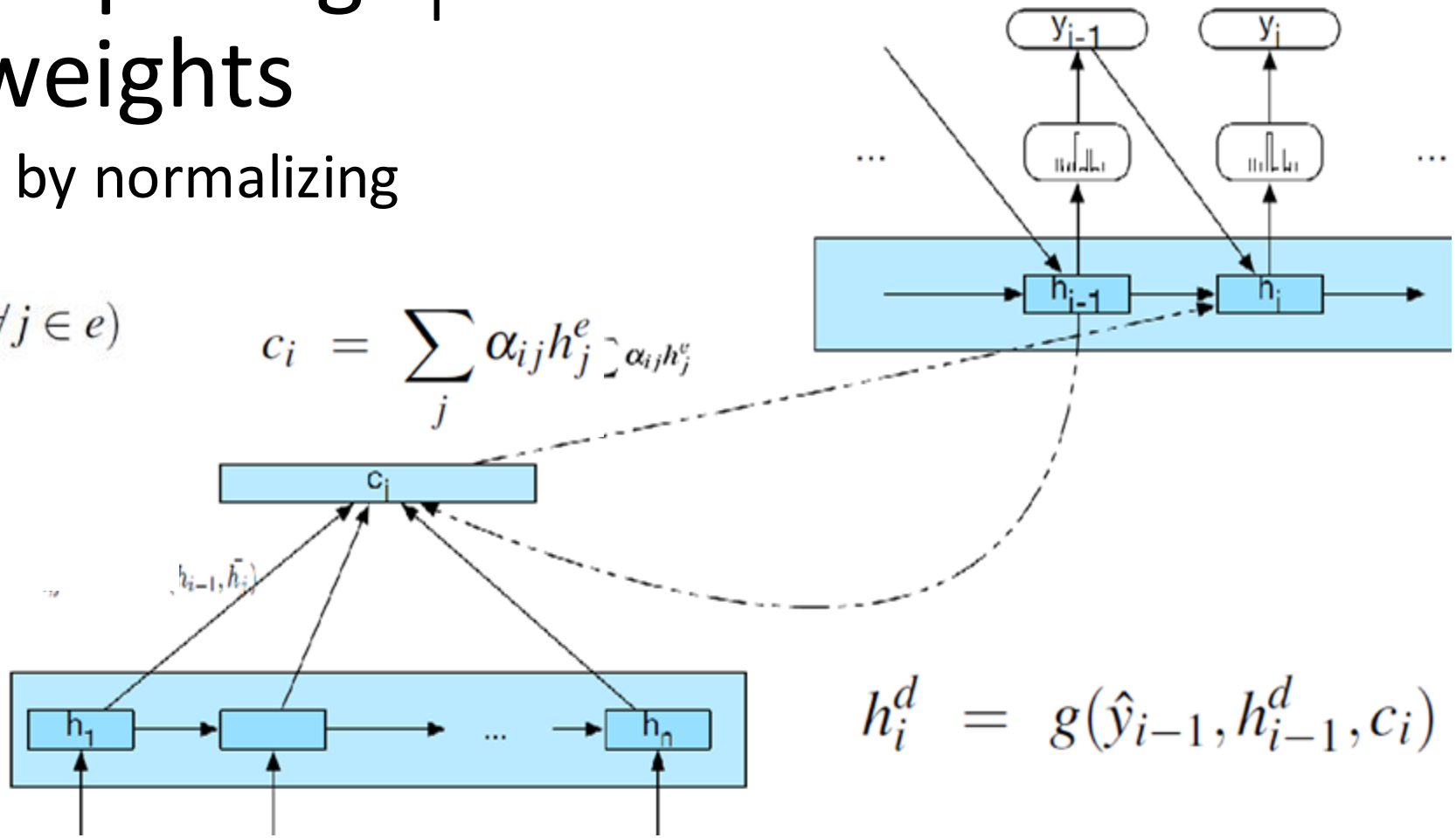
Attention (3): computing c_i From scores to weights

- Create vector of weights by normalizing scores

$$\alpha_{ij} = \text{softmax}(score(h^d_{i-1}, h^e_j) \ \forall j \in e)$$

$$= \frac{exp(score(h^d_{i-1}, h^e_j)}{\sum_k exp(score(h^d_{i-1}, h^e_k))}$$

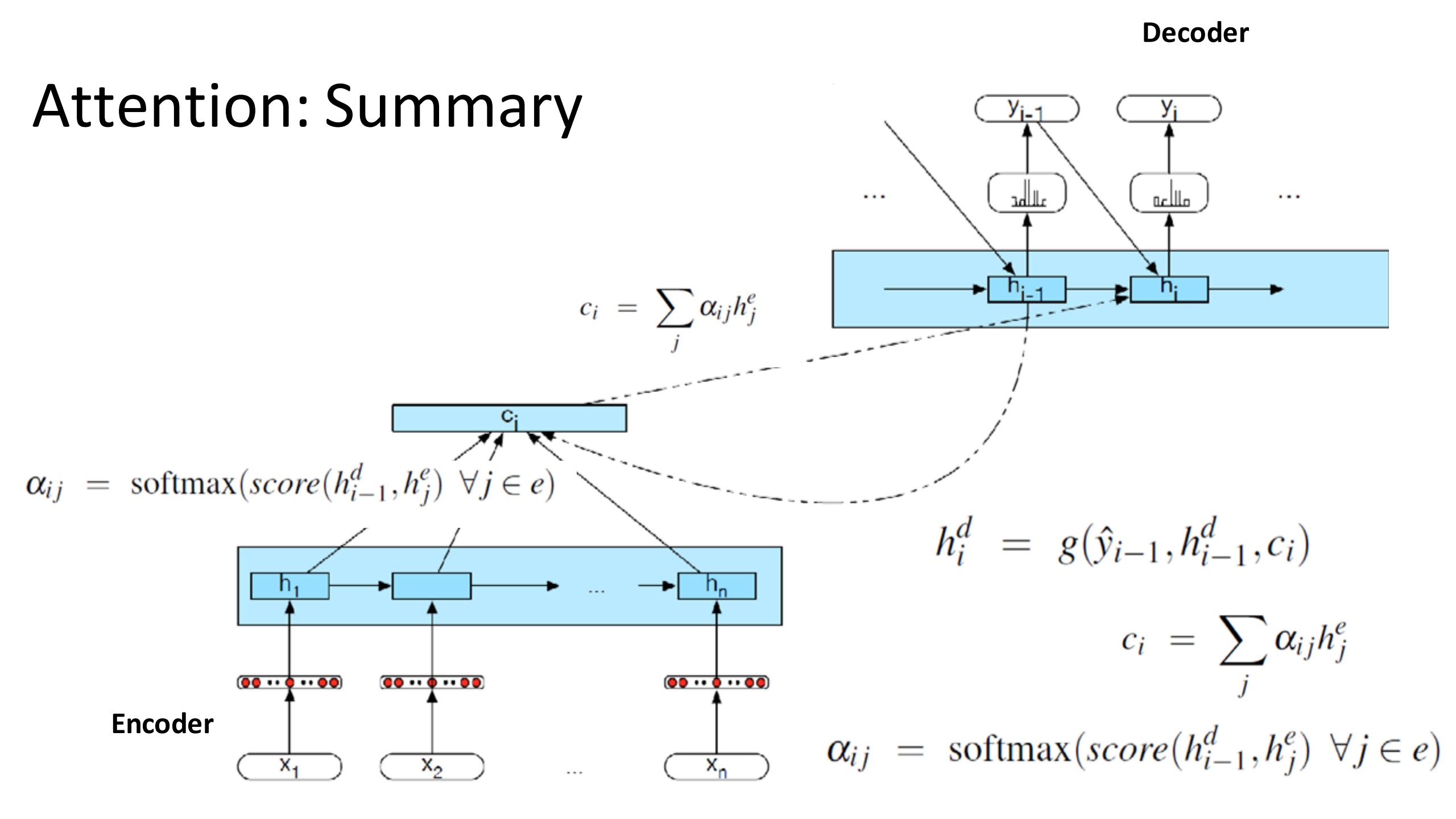
$$c_i = \sum_j \alpha_{ij} h^e_j$$



$$h^d_i = g(\hat{y}_{i-1}, h^d_{i-1}, c_i)$$

- **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: Summary



$$c_i = \sum_j \alpha_{ij} h_j^e$$

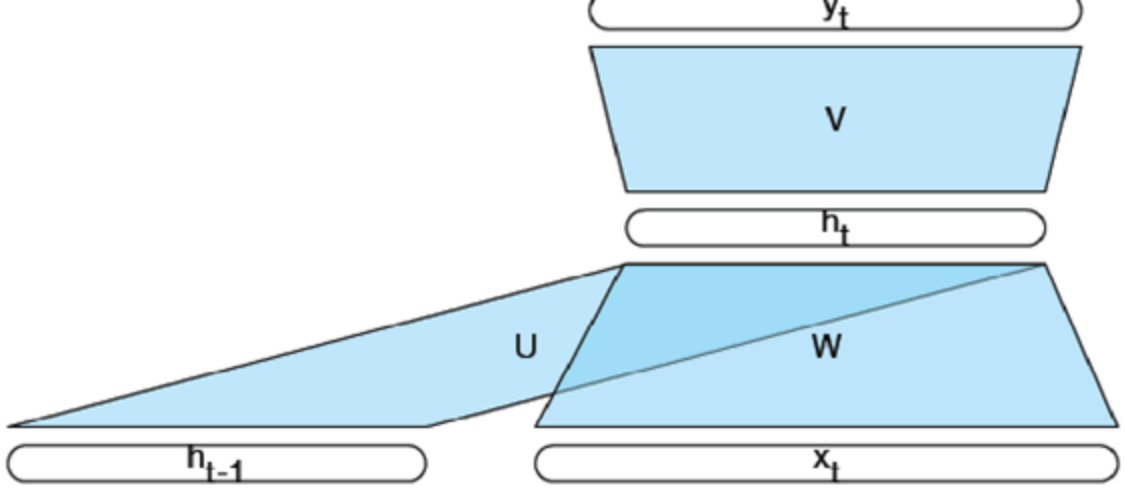
$$\alpha_{ij} = \text{softmax}(score(h_{i-1}^d, h_j^e) \ \forall j \in e)$$

$$h_i^d = g(\hat{y}_{i-1}, h_{i-1}^d, c_i)$$

$$c_i = \sum_j \alpha_{ij} h_j^e$$

$$\alpha_{ij} = \text{softmax}(score(h_{i-1}^d, h_j^e) \ \forall j \in e)$$

Explain Y. Goldberg different r



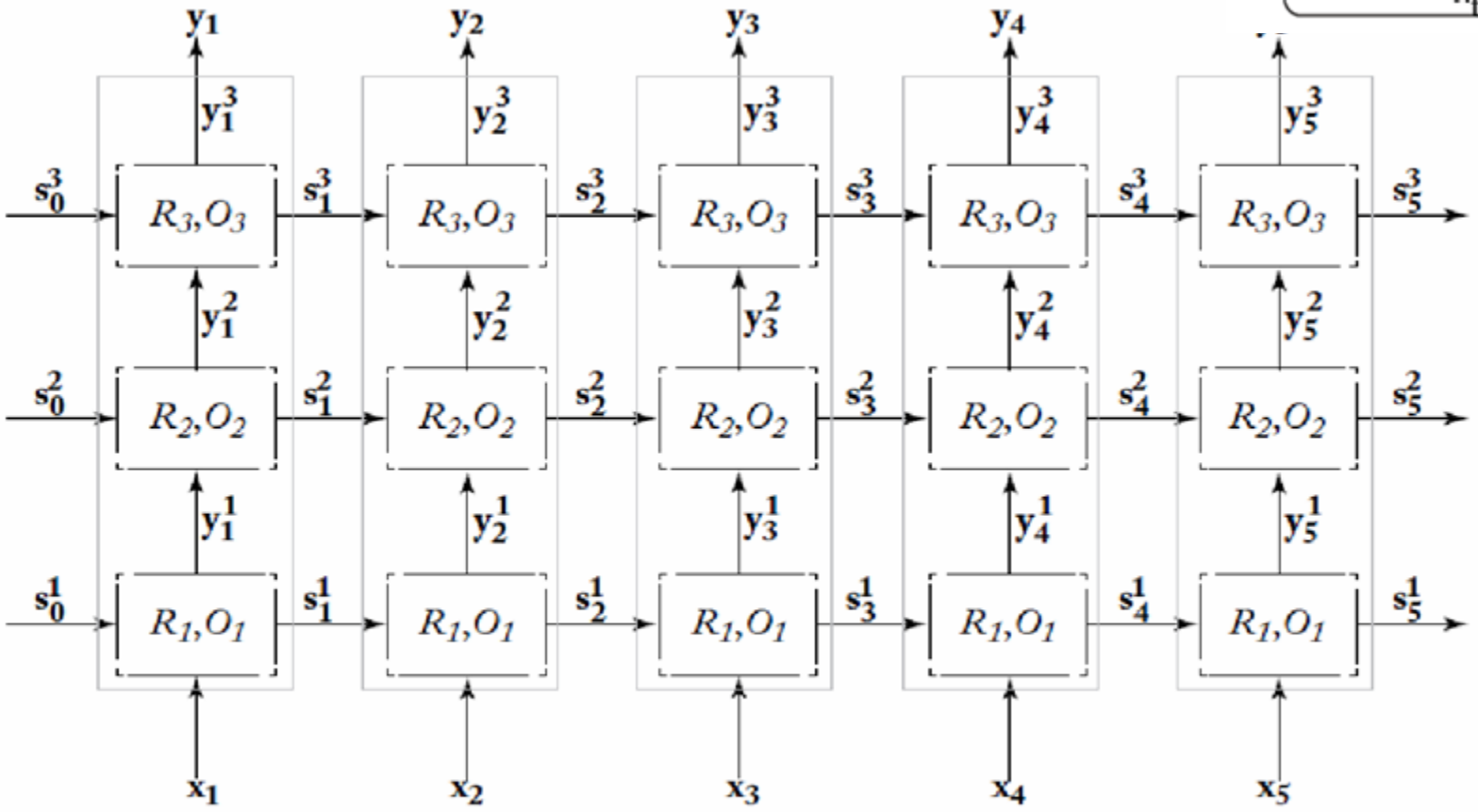


Figure 14.7: A three-layer ("deep") RNN architecture.

Intro to Encoder-Decoder and Attention (Goldberg's notation)

$$h_i^d = g(\hat{y}_{i-1}, h_{i-1}^d, c_i)$$

$$c_i = \sum_j \alpha_{ij} h_j^e$$

$$\alpha_{ij} = \text{softmax}(score(h_{i-1}^d, h_j^e) \ \forall j \in e)$$

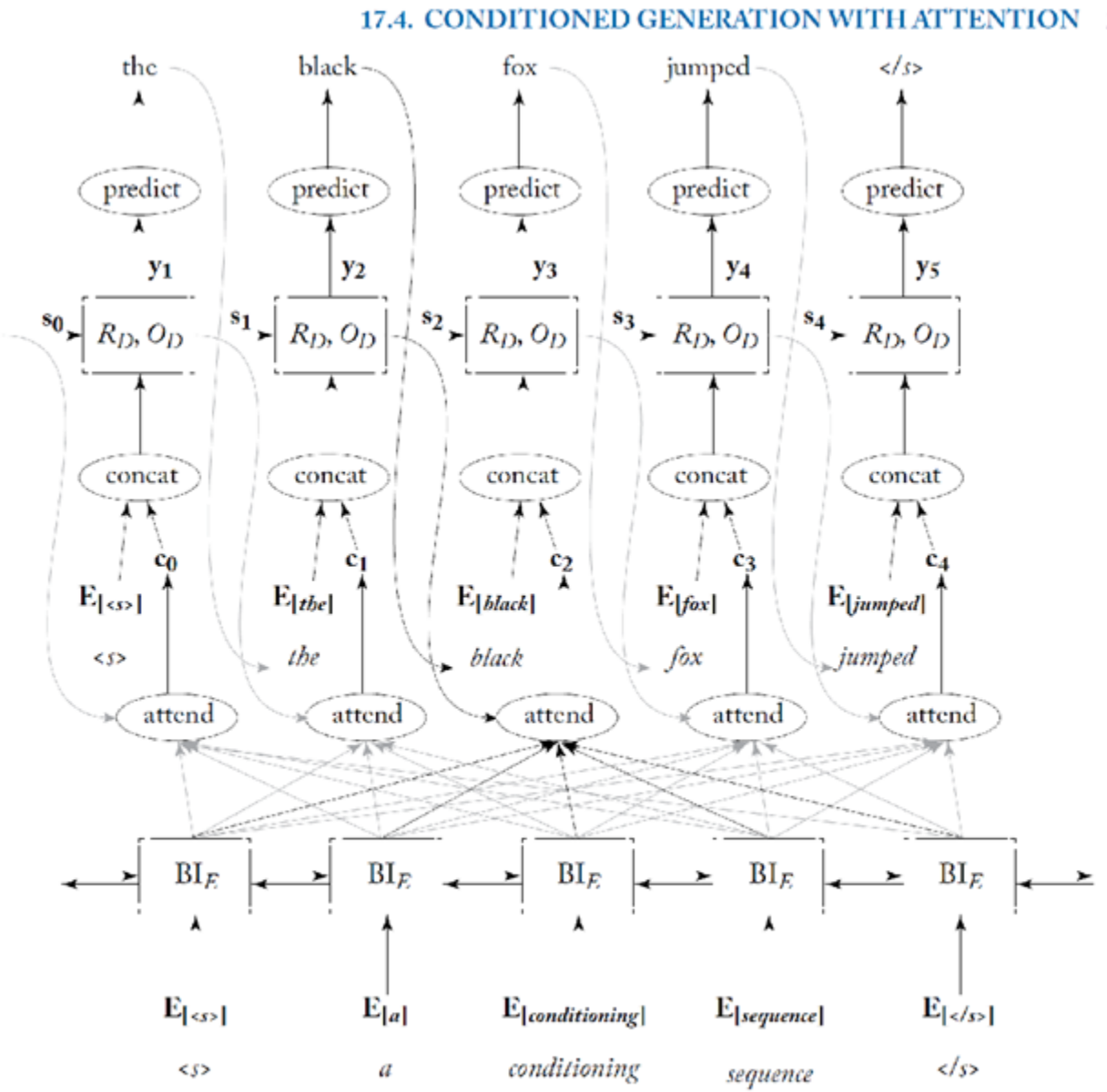


Figure 17.5: Sequence-to-sequence RNN generator with attention.

- Encoder Decoder
- Attention
- **Transformers** (self-attention)

Transformers (Attention is all you need 2017)

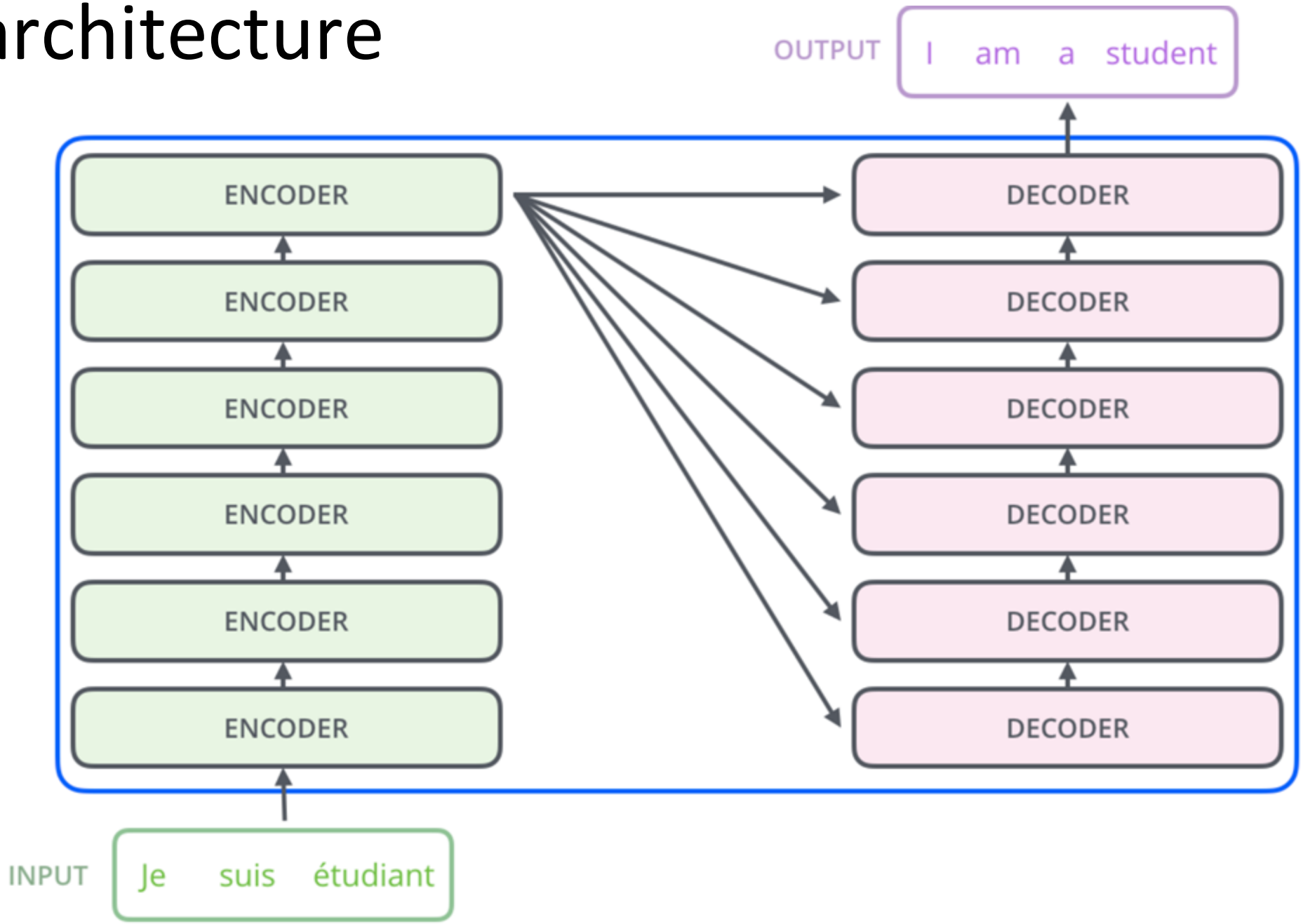
- **Just an introduction:** These are two valuable resources to learn more details on the architecture and implementation
- Also Assignment 4 will help you learn more about Transformers

- http://nlp.seas.harvard.edu/2018/04/03/attention.html

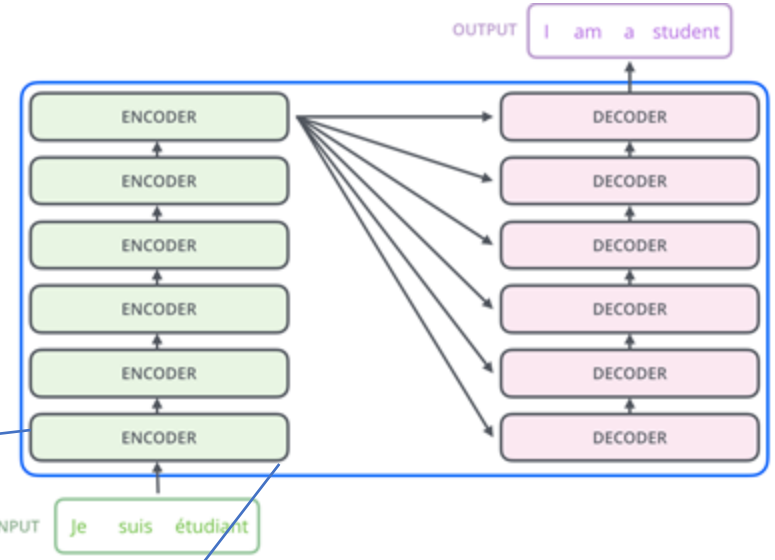
- https://jalammar.github.io/illustrated-transformer/ (slides come from this source)

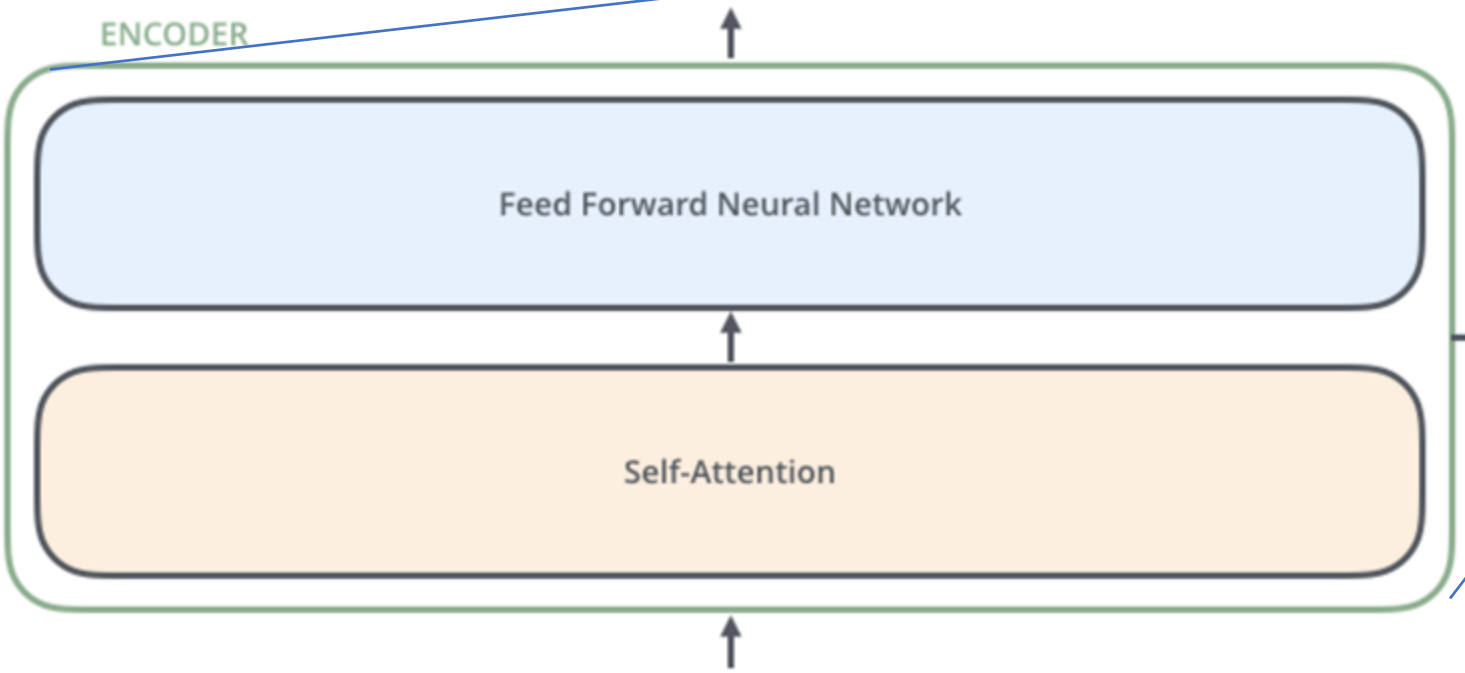
High-level architecture

- Will only look at the ENCODER(s) part in detail



The **encoders** are **all identical in structure** (yet they do not share weights). Each one is broken down into two sub-layers



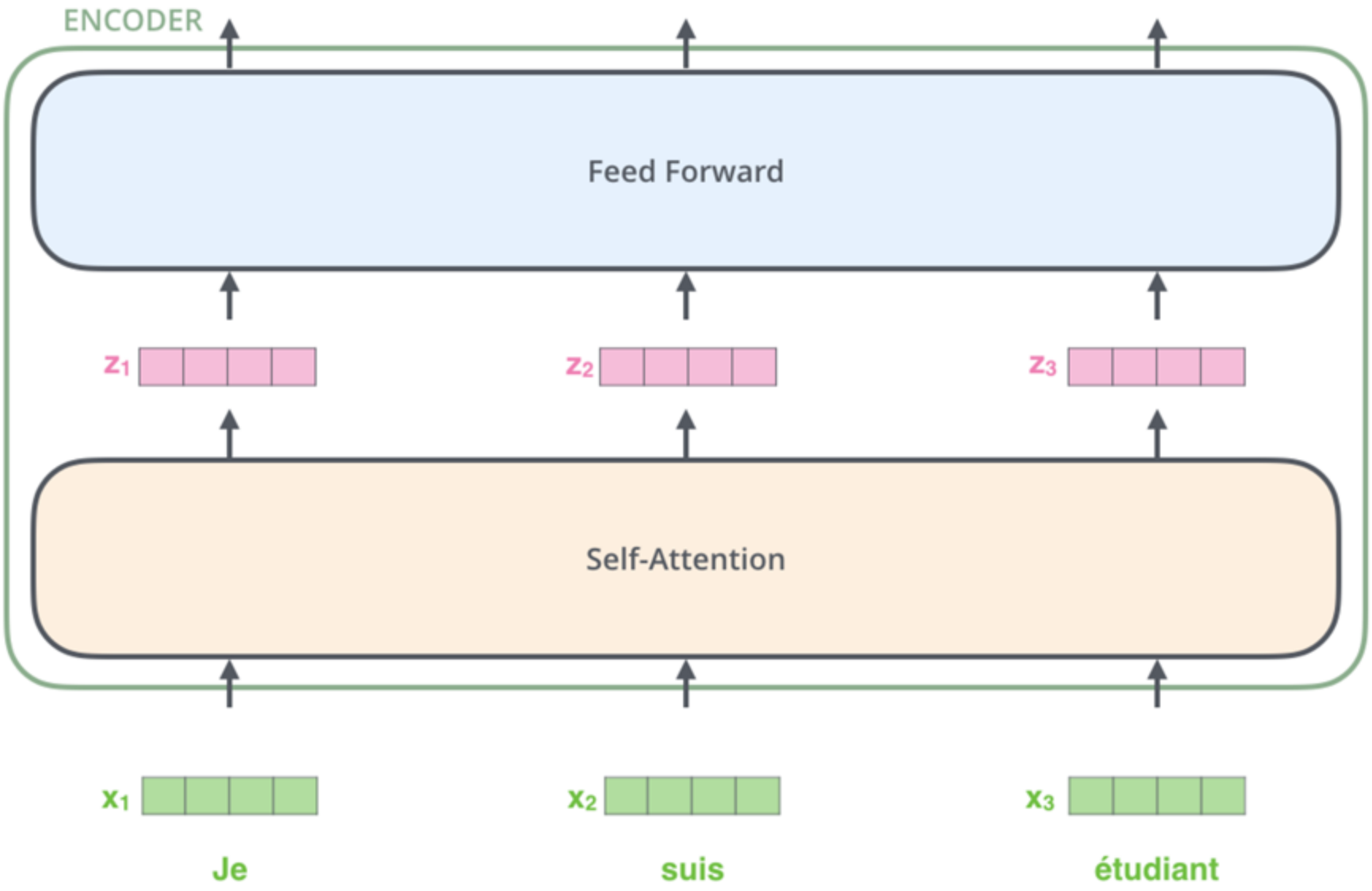


outputs of the self-attention are fed to a feed-forward neural network. The exact same one is independently applied to each position.

helps the encoder look at other words in the input sentence as it encodes a specific word.

Key property of Transformer: word in each position flows through its own path in the encoder.

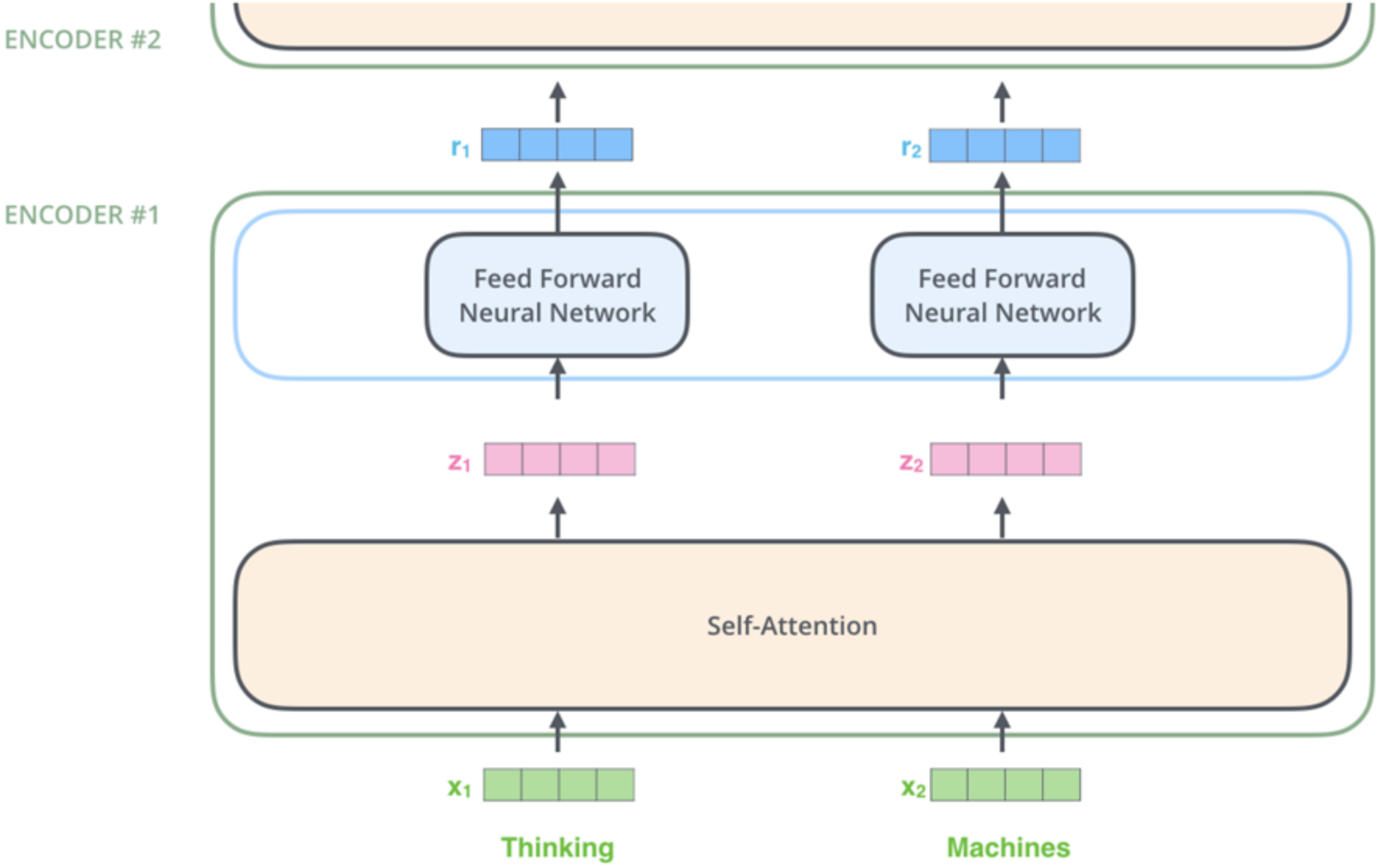
- There are dependencies between these paths in the self-attention layer.
- Feed-forward layer does not have those dependencies => various paths can be executed in parallel !



Word embeddings

Visually clearer on two words

- dependencies in self-attention layer.
- No dependencies in Feed-forward layer



Word embeddings

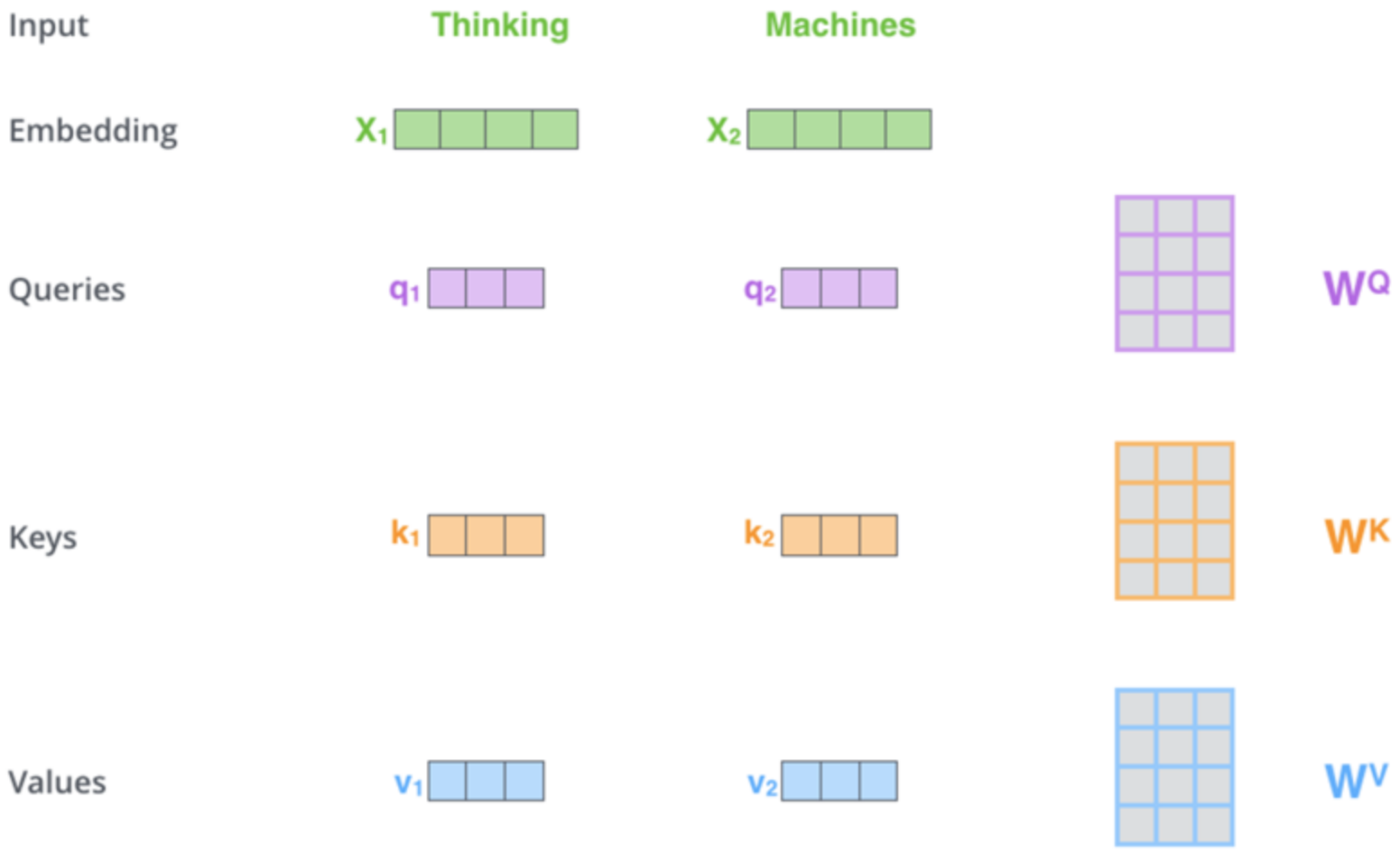
Self-Attention

While processing **each word** it allows to look at other positions in the input sequence for clues to build a better encoding for **this word**.

Step1: create three vectors from each of the encoder's input vectors:

Query, a Key, Value (typically smaller dimension).

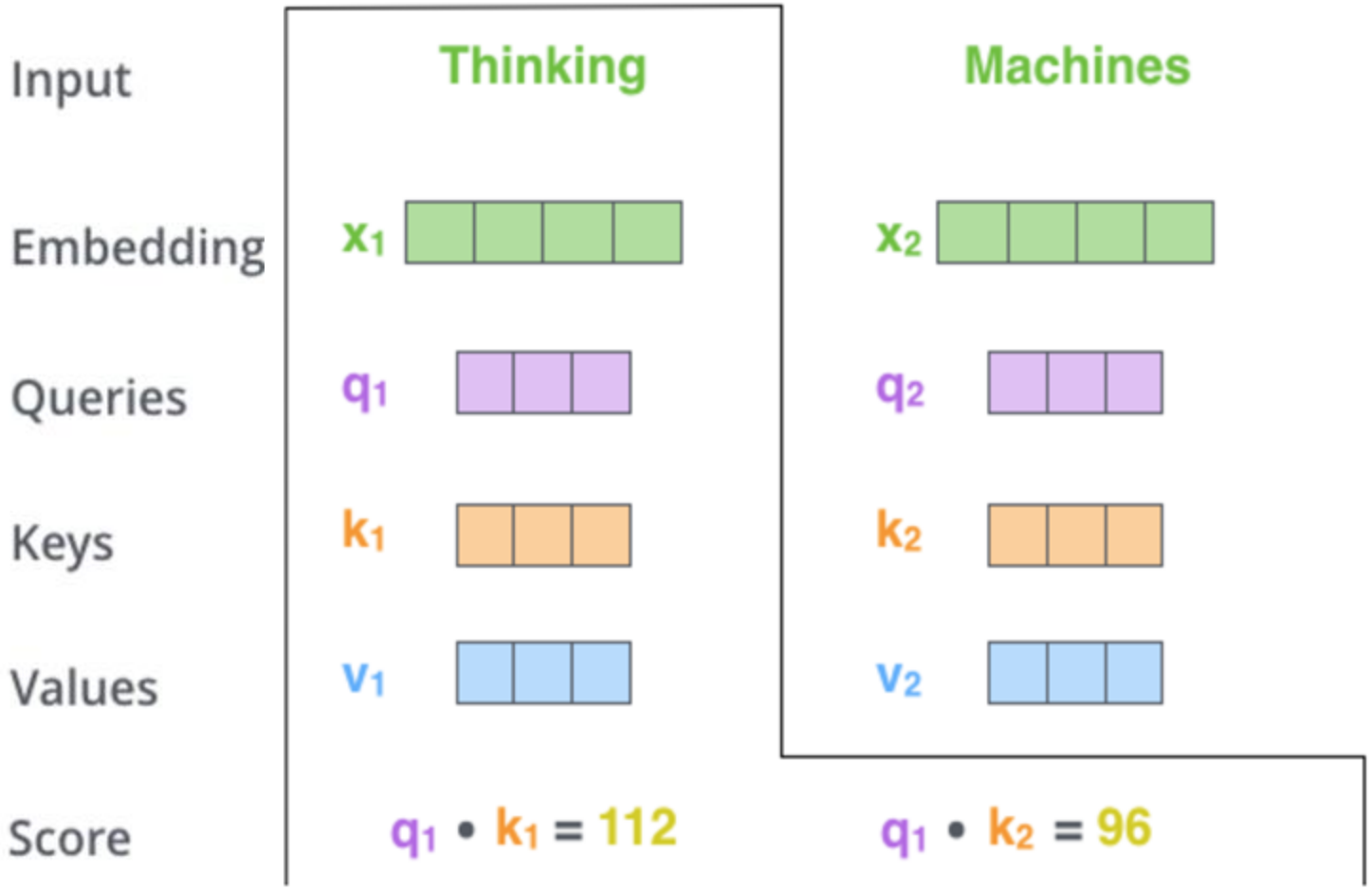
by multiplying the embedding by three matrices that we **trained** during the training process.



Self-Attention

Step 2: calculate a score (like we have seen for regular attention!) how much focus to place on other parts of the input sentence as we encode a word at a certain position.

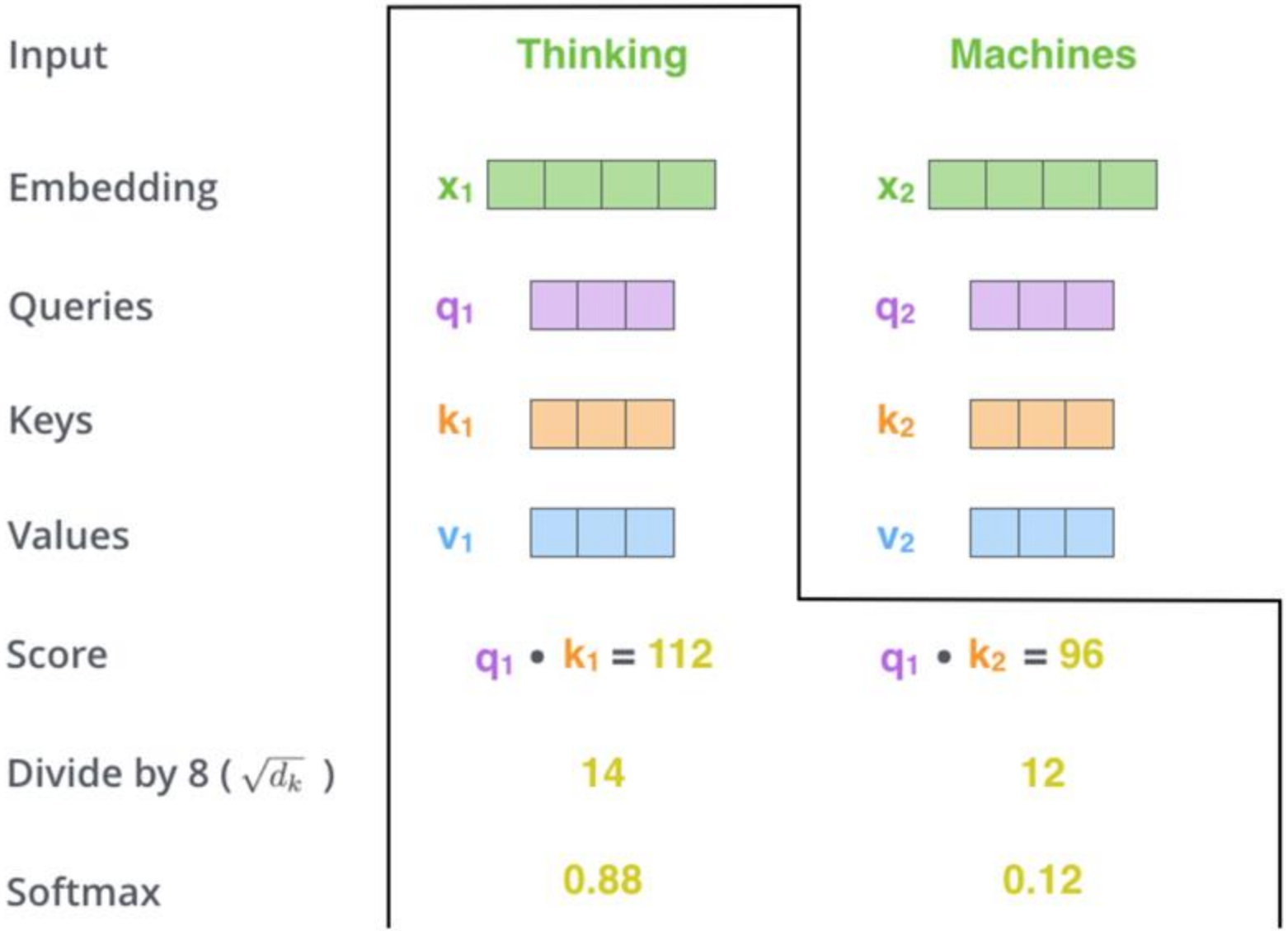
Take dot product of the query vector with the key vector of the respective word we're scoring.



E.g., Processing the self-attention for word "Thinking" in position #1, the first score would be the dot product of q1 and k1. The second score would be the dot product of q1 and k2.

Self Attention

- **Step 3** divide scores by the square root of the dimension of the key vectors (more stable gradients).
- **Step 4** pass result through a softmax operation. (all positive and add up to 1)



Intuition: softmax score determines how much each word will be expressed at this position.

Self Attention

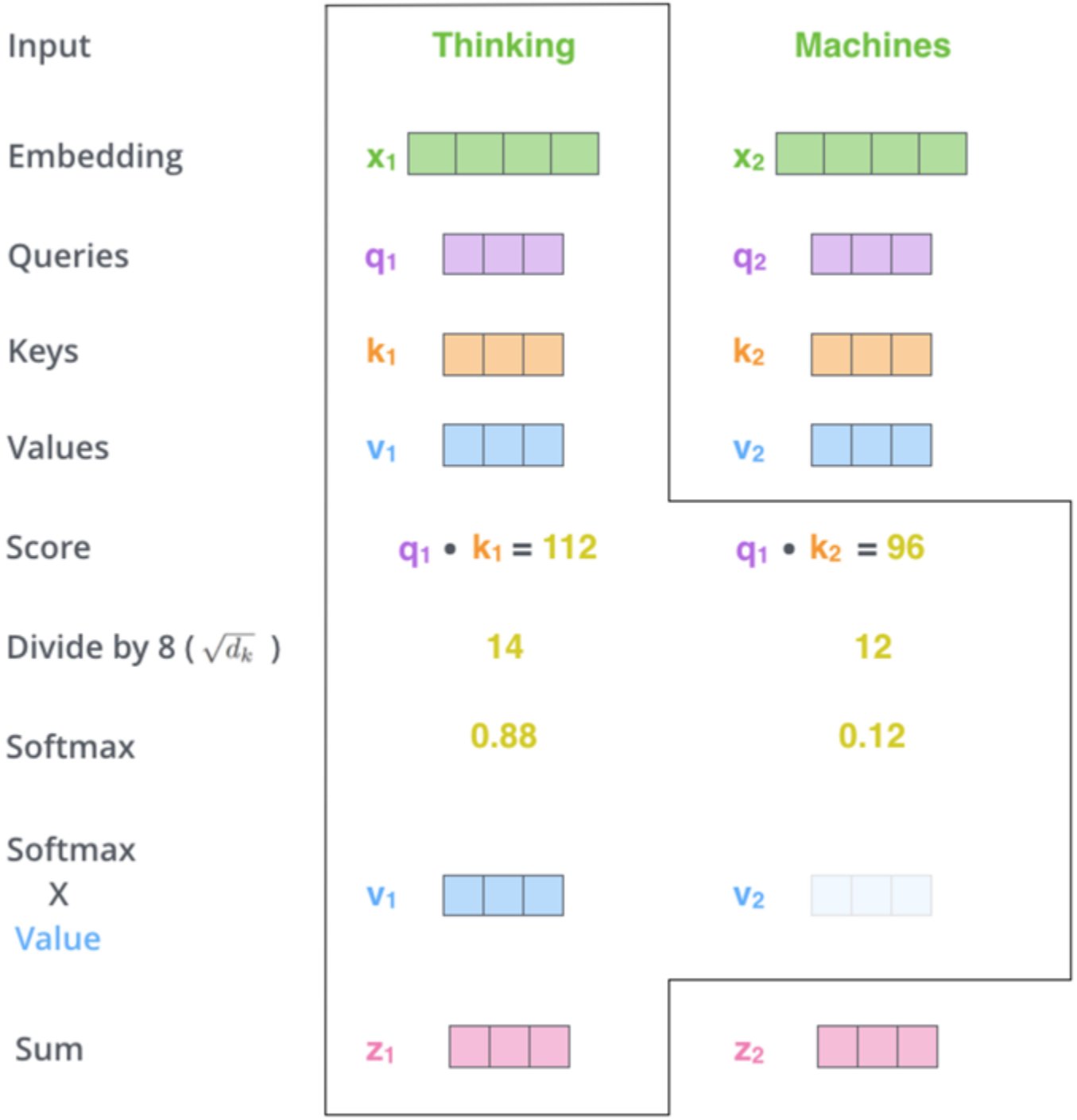
- **Step6** : sum up the weighted value vectors. This produces the output of the self-attention layer at this position

More details:

- What we have seen for a word is done **for all words** (using matrices)
- Need to **encode position** of words
- And improved using a mechanism called "**multi-headed**" attention

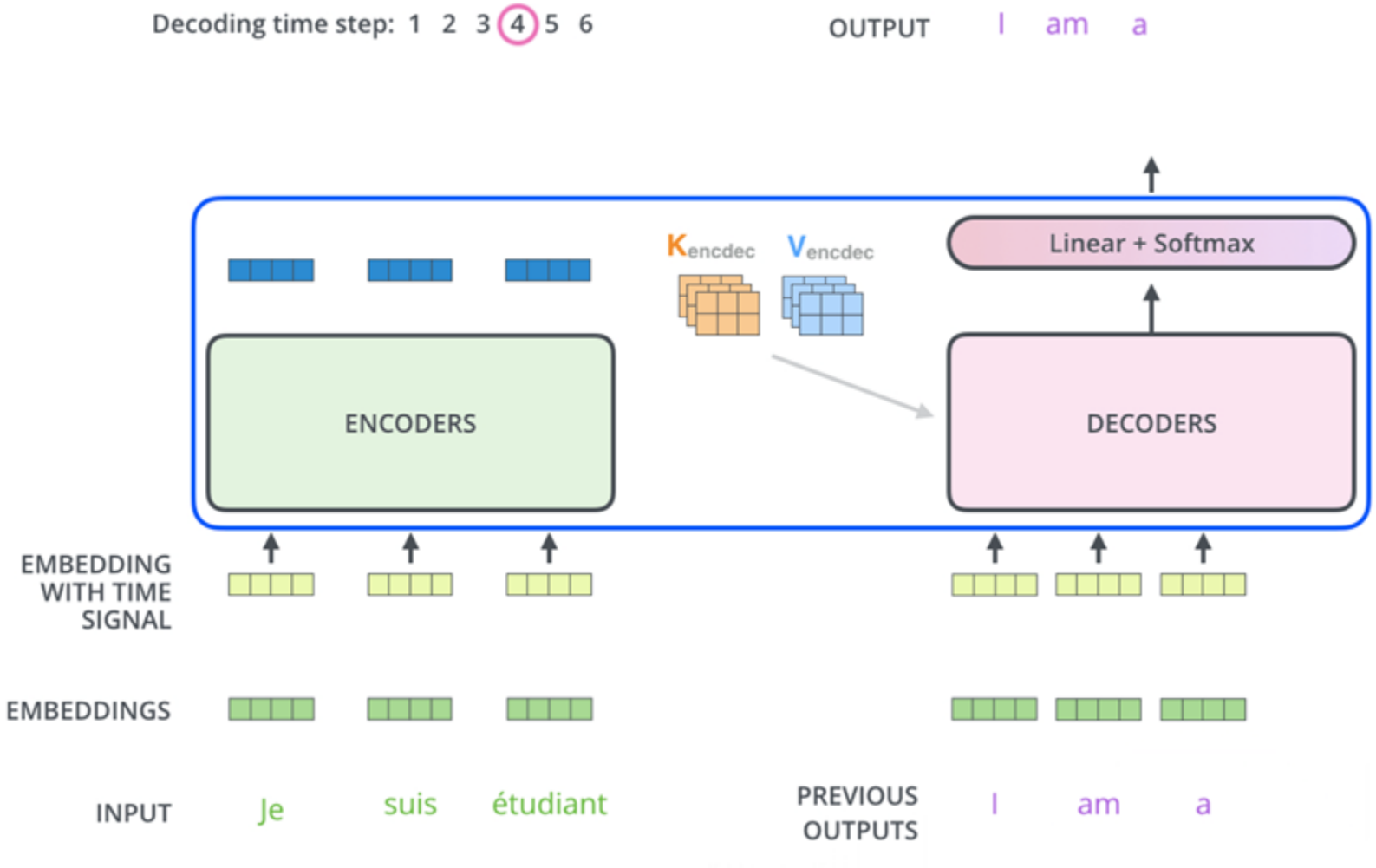
(kind of like multiple filters for CNN)

see https://jalammar.github.io/illustrated-transformer/



The Decoder Side

- Relies on most of the concepts on the encoder side
- See animation on https://jalammar.github.io/illustrated-transformer/



Read carefully!

Assignment-4 out tonight
Due Nov 13

Will send out two G-Forms
- On title and possibly group composition of the project (fill out by Friday)
- On your preferences for which paper to present

Next class: Mon Nov. 9

- **Project proposal** (submit your write-up and copy of your slides on Canvas; Write-up: 1-2 pages single project, 3-4 pages for group project)
- **Project proposal Presentation**
  - Approx. 3.5 min presentation + 1.5 min for questions (8 tot. mins if you are in a group)
  - For content, follow instructions at course project web page
  - Please have your presentation ready on your laptop to minimize transition delays
  - We will start in the usual zoom room @noon (sharp)

Today March 11

- Encoder Decoder
- Attention
- Transformers
- **Very Brief Intro Pragmatics**

Pragmatics: Example

(i) A: So can you please come over here again right now

(ii) B: Well, I have to go to Edinburgh today sir

(iii) A: Hmm. How about this Thursday?

What information can we infer about the context in which this (short and insignificant) exchange occurred ?

we can make a great number of detailed **(Pragmatic) inferences** about the nature of the context in which it occurred

Pragmatics: Conversational Structure

(i) A: So can you please come over here again right now

(ii) B: Well, I have to go to Edinburgh today sir

(iii) A: Hmm. How about this Thursday?

Not the end of a conversation (nor the beginning)

Pragmatic knowledge: Strong expectations about the structure of conversations

- **Pairs e.g., request <-> response**
- **Closing/Opening forms**

Pragmatics: Dialog Acts

(i) A: So can you please come over here again right now?
(ii) B: Well, I have to go to Edinburgh today sir
(iii) A: Hmm. How about this Thursday?

Not a Y/N info seeking question like *"can you run for 1h?"*
It is a request for an action

- *A is **requesting** B to come at time of speaking,*
- *B **implies he can't** (or would rather not)*
- *A **repeats the request** for some other time.*

Pragmatic assumptions relying on:

- **mutual knowledge (B knows that A knows that…)**
- **co-operation (must be a response… triggers inference)**
- **topical coherence (who should do what on Thur?)**

Pragmatics: Specific Act (Request)

(i) A: So can you please come over here again right now

(ii) B: Well, I have to go to Edinburgh today sir

(iii) A: Hmm. How about this Thursday?

- *A wants B to come over*
- *A believes it is possible for B to come over*
- *A believes B is not already there*
- *A believes he is not in a position to order B to…*

Pragmatic knowledge: speaker beliefs and intentions underlying the act of requesting

Assumption: A behaving rationally and sincerely

Pragmatics: Deixis

(i) A: So can you please come over here again right now
(ii) B: Well, I have to go to Edinburgh today sir
(iii) A: Hmm. How about this Thursday?

- *A assumes B knows where A is*
- *Neither A nor B are in Edinburgh*
- *The day in which the exchange is taking place is not Thur., nor Wed. (or at least, so A believes)*

Pragmatic knowledge: References to space and time wrt space and time of speaking

Additional Notes (not required)

From Yoav Artzi (these are links)

Contextualized word representations

Annotated Transformer, Illustrated Transformer, ELMo, **BERT**, **The Illustrated BERT, ELMo, and co.**

Transformers

- WIKIPEDIA:
- However, unlike RNNs, Transformers do not require that the sequence be processed in order. So, if the data in question is natural language, the Transformer does not need to process the beginning of a sentence before it processes the end. Due to this feature, the Transformer allows for much more parallelization than RNNs during training.[1]

The Transformer uses multi-head attention in three different ways:

- In "encoder-decoder attention" layers, the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. This allows every position in the decoder to attend over all positions in the input sequence. This mimics the typical encoder-decoder attention mechanisms in sequence-to-sequence models such as [38, 2, 9].
- The encoder contains self-attention layers. In a self-attention layer all of the **keys, values and queries come from the same place, in this case, the output of the previous layer in the encoder**. Each position in the encoder can attend to all positions in the previous layer of the encoder.
- Similarly, self-attention layers in the decoder **allow each position in the decoder to attend to all positions in the decoder up to and including that position**. We need to prevent leftward information flow in the decoder to preserve the **auto-regressive** property. We implement this inside of scaled dot-product attention by masking out (setting to infinity) all values in the input of the softmax which correspond to illegal connections. See Figure 2.

At each step the model is auto-regressive
[10], consuming the previously generated symbols as additional input when generating the next.
The autoregressive model specifies that the output variable depends linearly on its own previous values and on a stochastic term (an imperfectly predictable term);

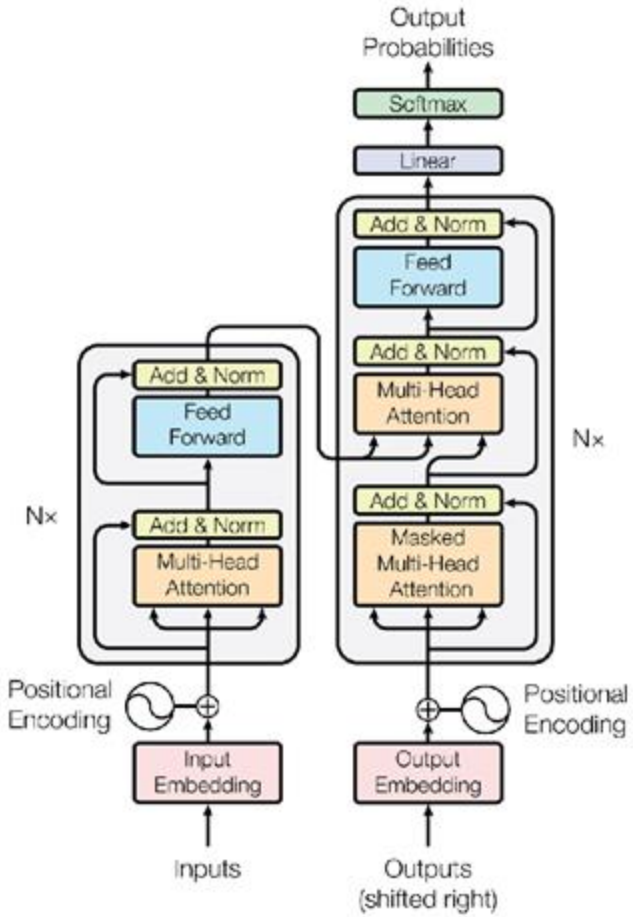


Figure 1: The Transformer - model architecture.

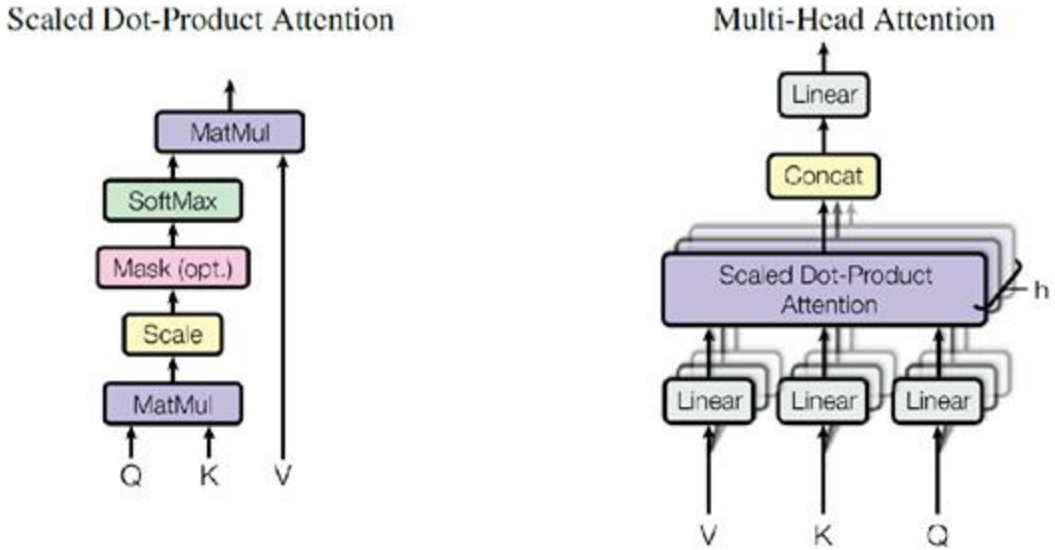


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

3.3 Position-wise Feed-Forward Networks

In addition to attention sub-layers, each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically. This consists of two linear transformations with a ReLU activation in between.

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{2}$$

While the linear transformations are the same across different positions, they use different parameters from layer to layer. Another way of describing this is as two convolutions with kernel size 1. The dimensionality of input and output is $d_{\text{model}} = 512$, and the inner-layer has dimensionality $d_{ff} = 2048$.

- Positional Encodings

tokens in the sequence. To this end, we add "positional encodings" to the input embeddings at the bottoms of the encoder and decoder stacks. The positional encodings have the same dimension d_{model} as the embeddings, so that the two can be summed. There are many choices of positional encodings, learned and fixed [9].

In this work, we use sine and cosine functions of different frequencies:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

where pos is the position and i is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$. We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}.

We also experimented with using learned positional embeddings [9] instead, and found that the two versions produced nearly identical results (see Table 3 row (E)). We chose the sinusoidal version because it may allow the model to extrapolate to sequence lengths longer than the ones encountered during training.

In geometry, an **affine transformation**, **affine map**[1] or an **affinity** (from the Latin, *affinis*, "connected with") is a function between affine spaces which preserves points, straight lines and planes. Also, sets of parallel lines remain parallel after an affine transformation. An affine transformation does not necessarily preserve angles between lines or distances between points, though it does preserve ratios of distances between points lying on a straight line.

Additional resources !

References/Resources to explain transformers in cpsc503. And possible question for assignment

http://jalammar.github.io/illustrated-transformer/

Combined with this http://nlp.seas.harvard.edu/2018/04/03/attention.html

Medium article that I read a while back and thought that s a nice intro to the transformer: https://medium.com/@adityathiruvengadam/transformer-architecture-attention-is-all-you-need-aeccd9f50d09

They first start with motivating attention in general and show problems of RNN/CNN architectures, then leading to the transformer. I especially liked some of the visualizations they have. But it is a relatively long read and unfortunately, at some points, it's not super consistent. I thought it might be still useful.

Google T5 Explores the Limits of Transfer Learning

Synced Follow
Nov 7, 2019 · 7 min read