BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Attention Models & Transformers

S. P. Vimal, CSIS Department (WILP Division)
vimalsp@pilani.bits-pilani.ac.in

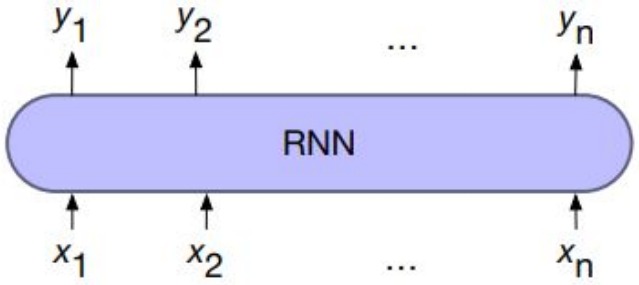
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Topics

- ★ Brief review of Sequence Models (RNN, LSTMS)
- ★ Encoder - Decoder Models
- ★ Attention
- ★ Transformers
- ★ Vision Transformers

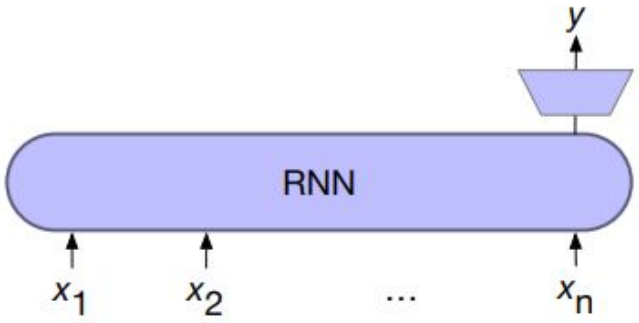
References and Recommendation for Readings: Draft Chapter 9 and Draft Chapter 10 , Speech and Language Processing. Daniel Jurafsky & James H. Martin.

RNN Architectures for NLP Tasks



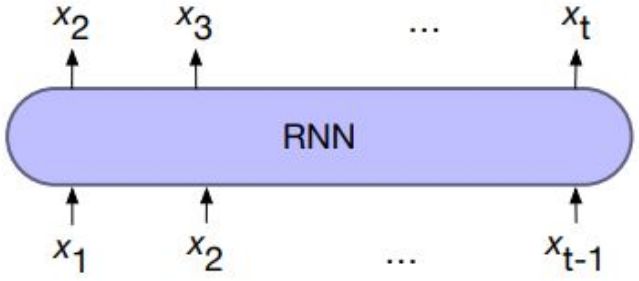
a) sequence labeling

Ex: POS Tagging, Named Entity Tagging



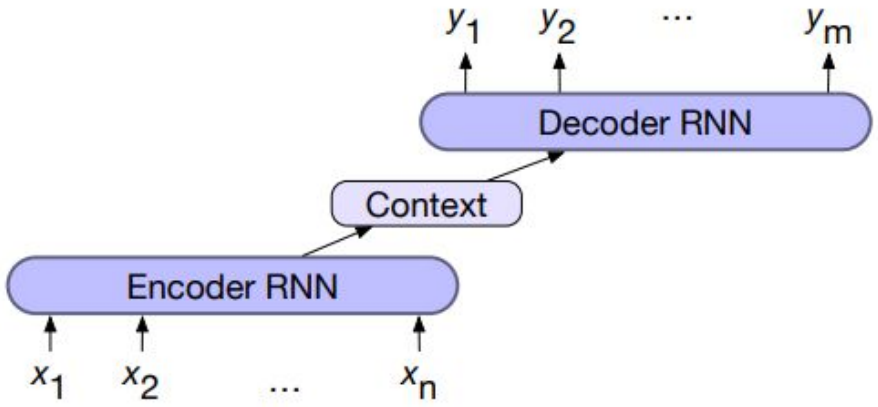
b) sequence classification

Ex: Sentiment Analysis



c) language modeling

Ex: Predict Next Word



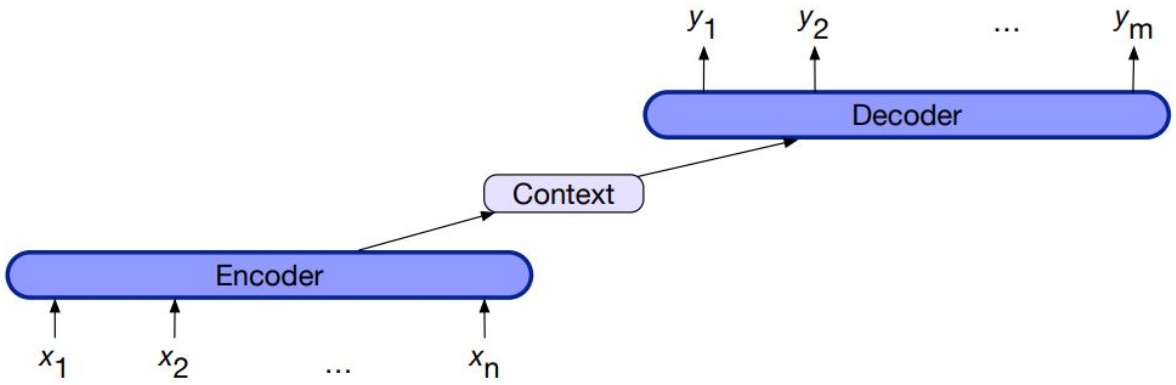
d) encoder-decoder

Ex: Language Translation

Encoder - Decoder

- **Goal:** Develop an architecture capable of generating *contextually appropriate*, *arbitrary length*, output sequences

- **Applications**:
  - Machine translation
  - Summarization
  - Question answering
  - Dialogue modeling.



Encoder - Decoder

Encoder:

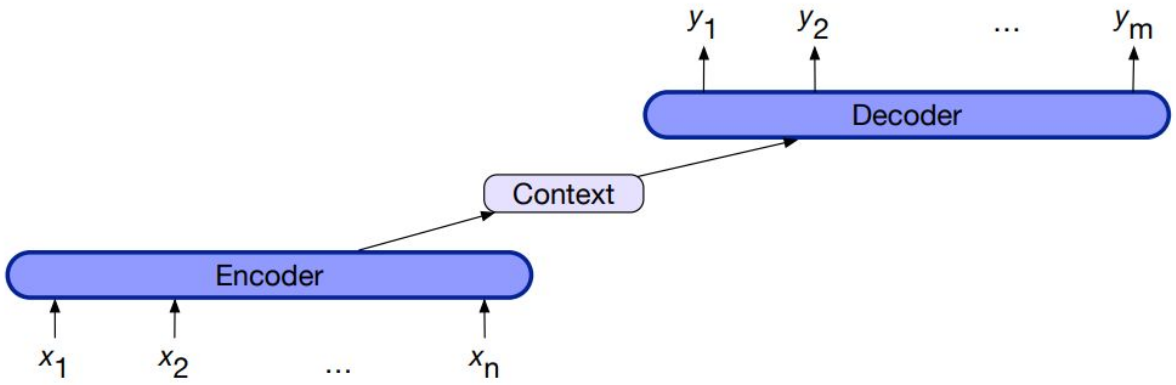
- Input sequence ($x_{1...n}$) $\rightarrow$ Sequence of contextualized representations, ($h_{1...n}$)
- Ex: LSTM, CNN, Transformers etc.

Context:

- c, a function of ($h_{1...n}$)

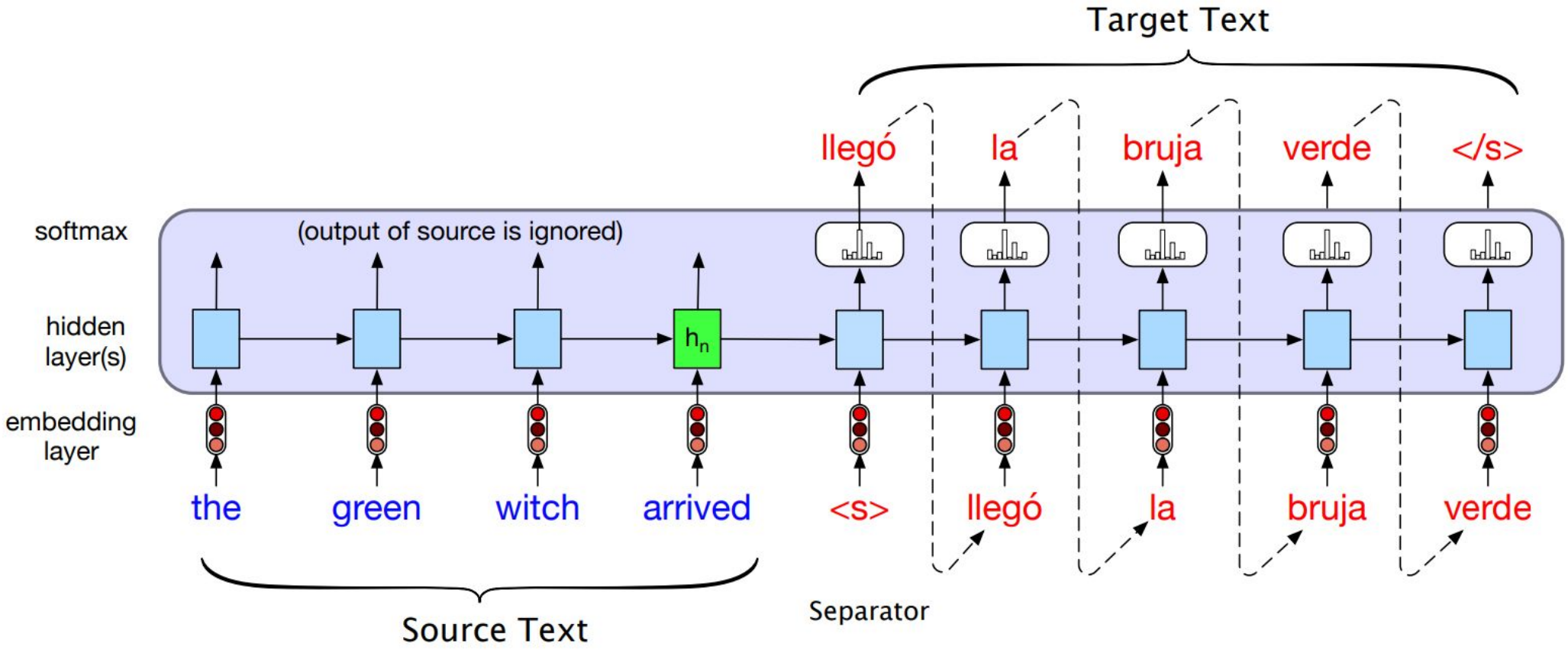
Decoder:

- c $\rightarrow$ arbitrary length sequence of hidden states ($h_{1...m}$) $\rightarrow$ sequence of output states ($y_{1...m}$)



Encoder - Decoder

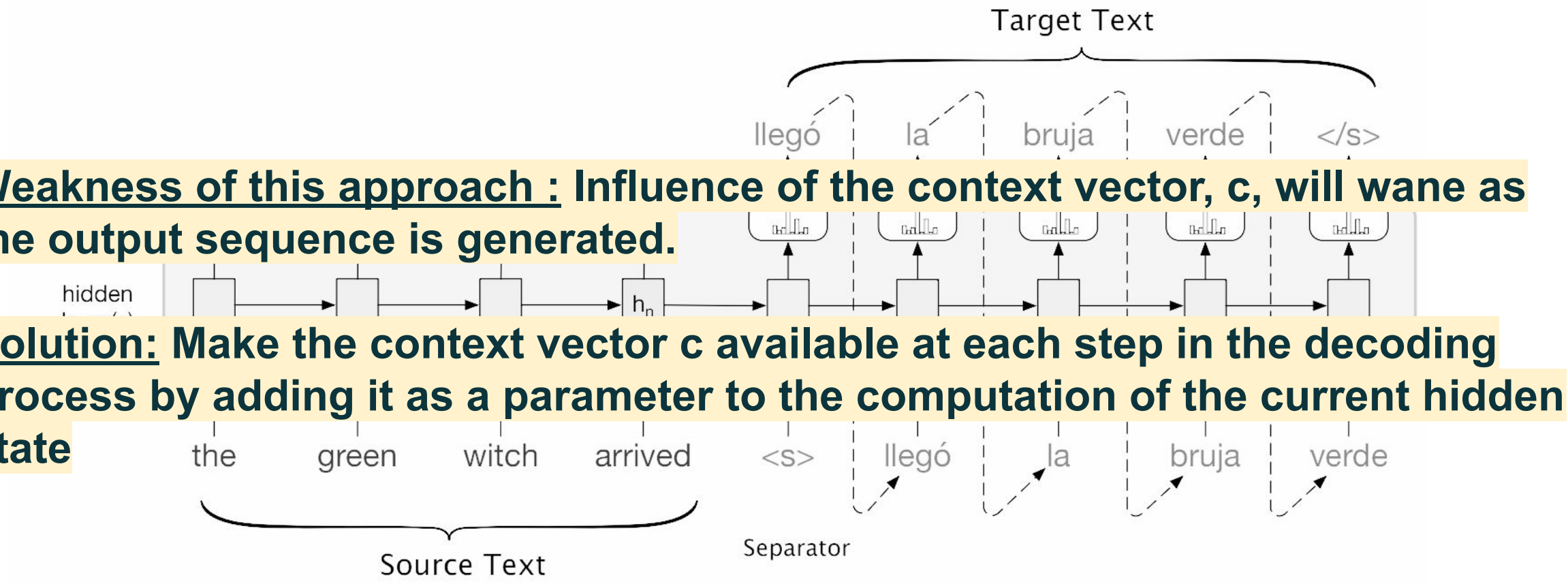
Encoder - Decoder for Language Translation



Source and target sentences are concatenated with a separator token in between, and the decoder uses context information from the encoder's last hidden state

Encoder - Decoder

Encoder - Decoder for Language Translation



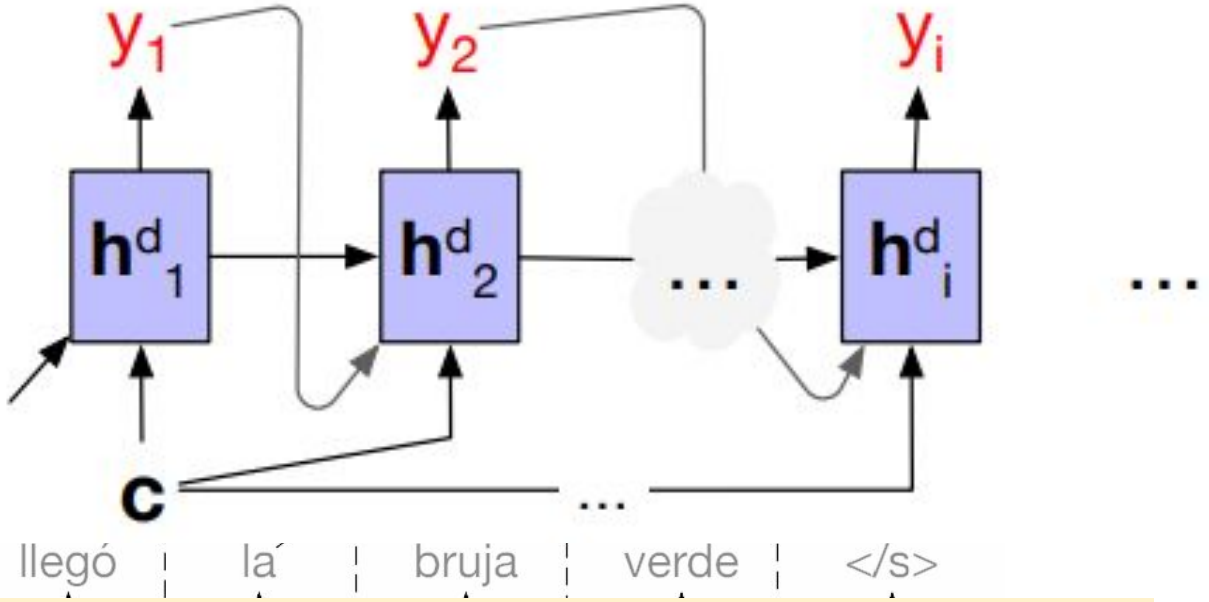
Weakness of this approach : Influence of the context vector, c, will wane as the output sequence is generated.

Solution: Make the context vector c available at each step in the decoding process by adding it as a parameter to the computation of the current hidden state

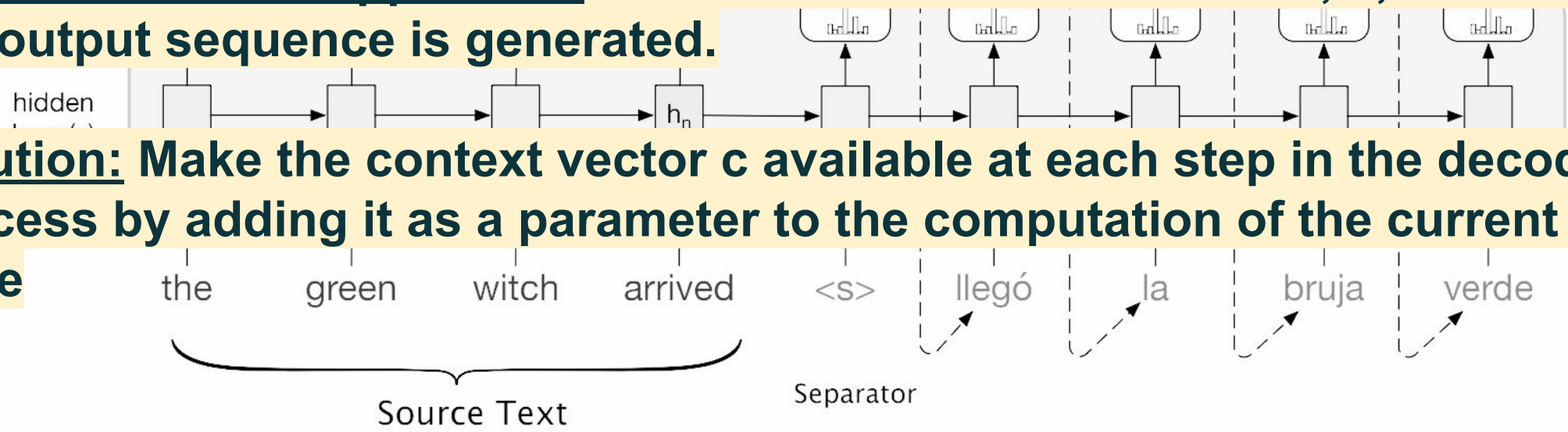
Source and target sentences are concatenated with a separator token in between, and the decoder uses context information from the encoder's last hidden state

Encoder - Decoder

Encoder - Decoder for Language Translation



Weakness of this approach : Influence of the context vector, c, will wane as the output sequence is generated.

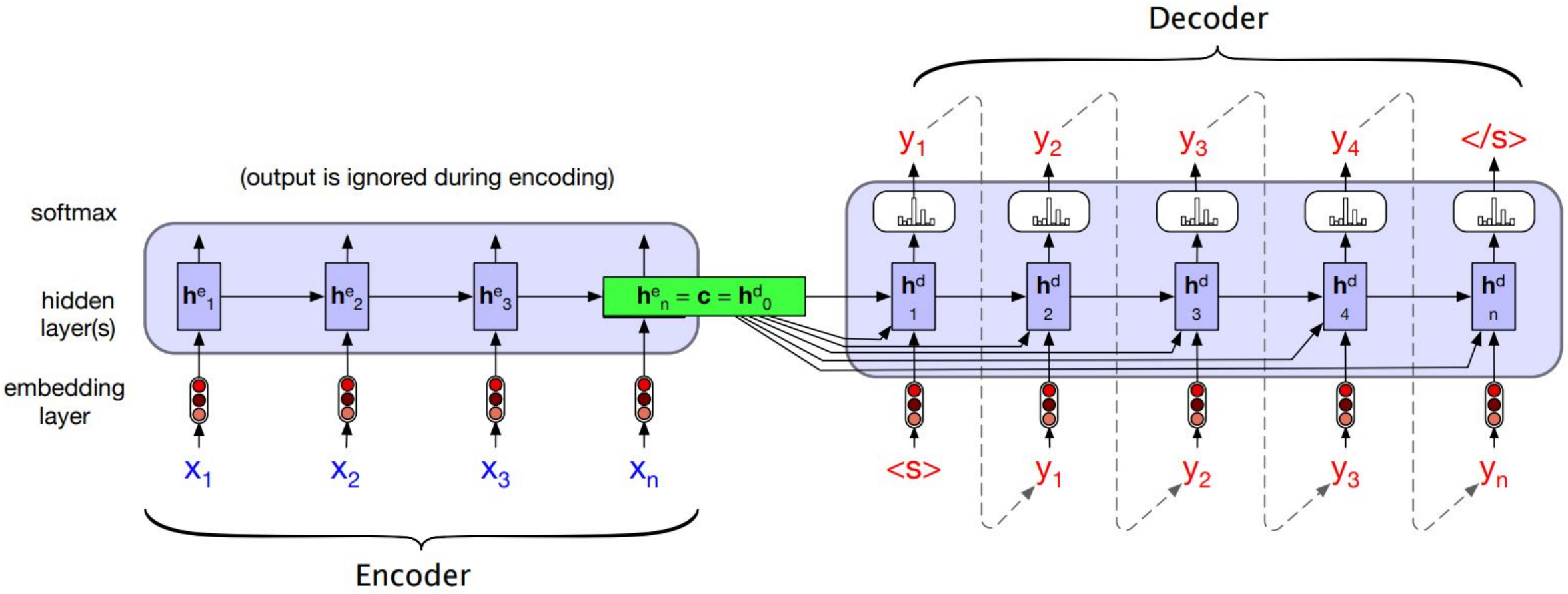


Solution: Make the context vector c available at each step in the decoding process by adding it as a parameter to the computation of the current hidden state

Source and target sentences are concatenated with a separator token in between, and the decoder uses context information from the encoder's last hidden state

Encoder - Decoder

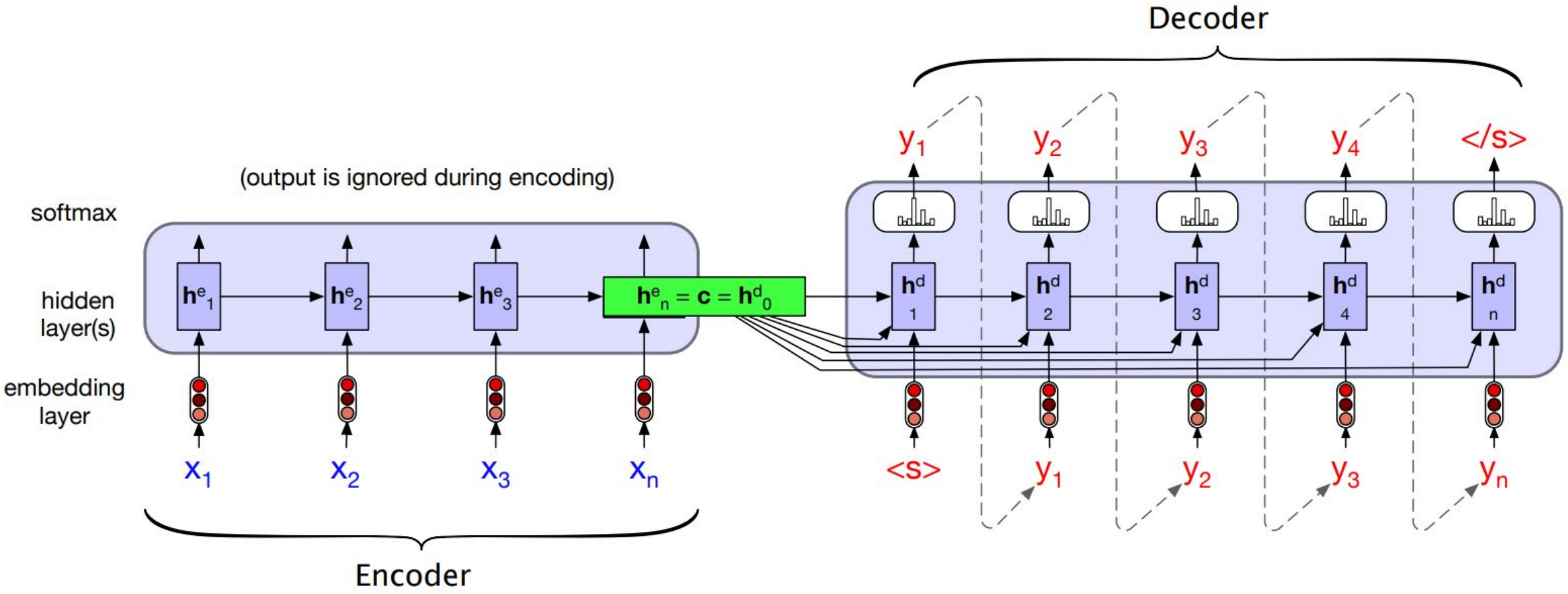
Encoder - Decoder for Language Translation



Encoder - Decoder

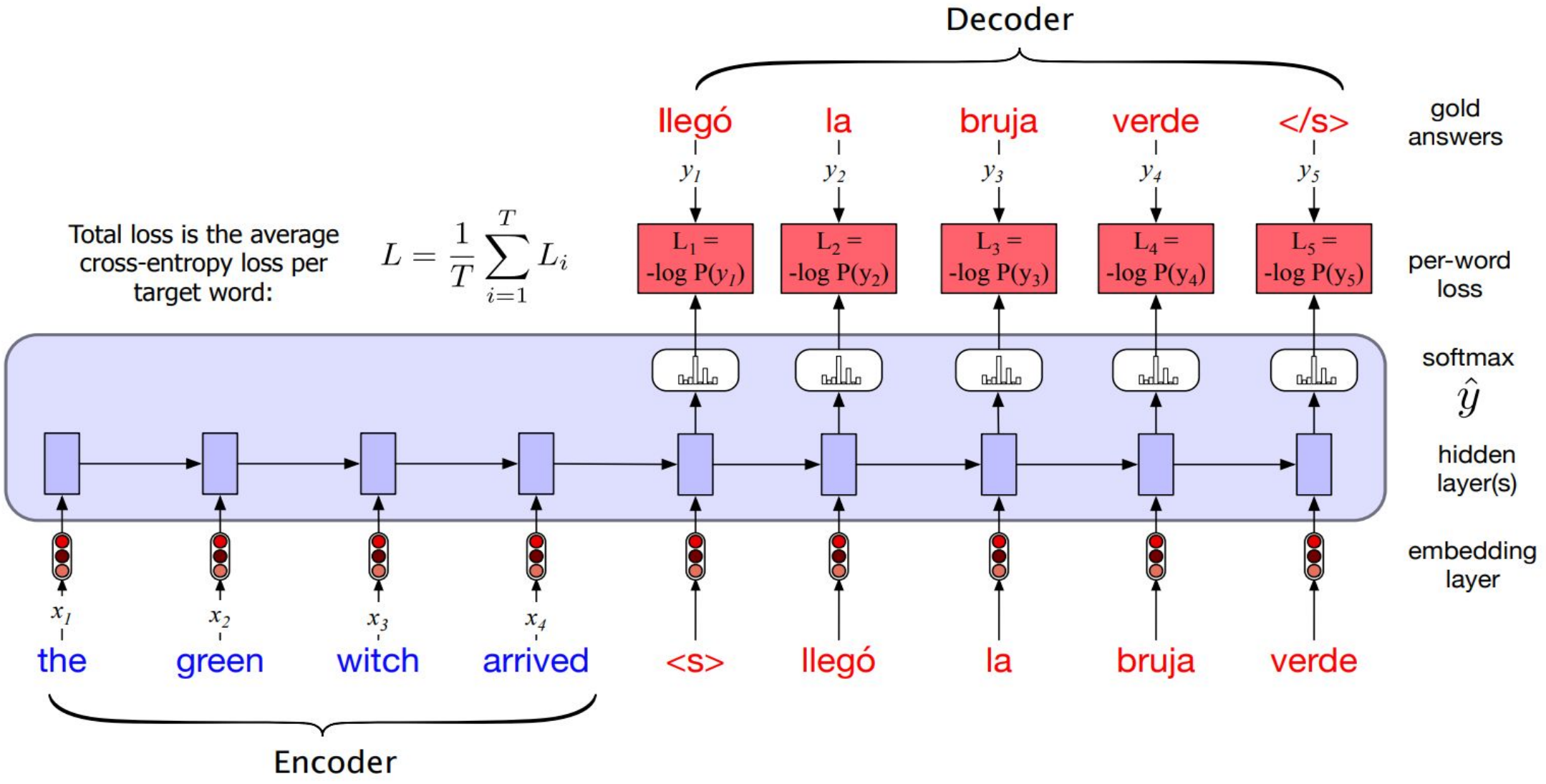
Encoder - Decoder for Language Translation

$$
\begin{aligned}
\mathbf{c} &= \mathbf{h}_n^e \\
\mathbf{h}_0^d &= \mathbf{c} \\
\mathbf{h}_t^d &= g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c}) \\
\mathbf{z}_t &= f(\mathbf{h}_t^d) \\
y_t &= \text{softmax}(\mathbf{z}_t)
\end{aligned}
$$



Encoder - Decoder

Training

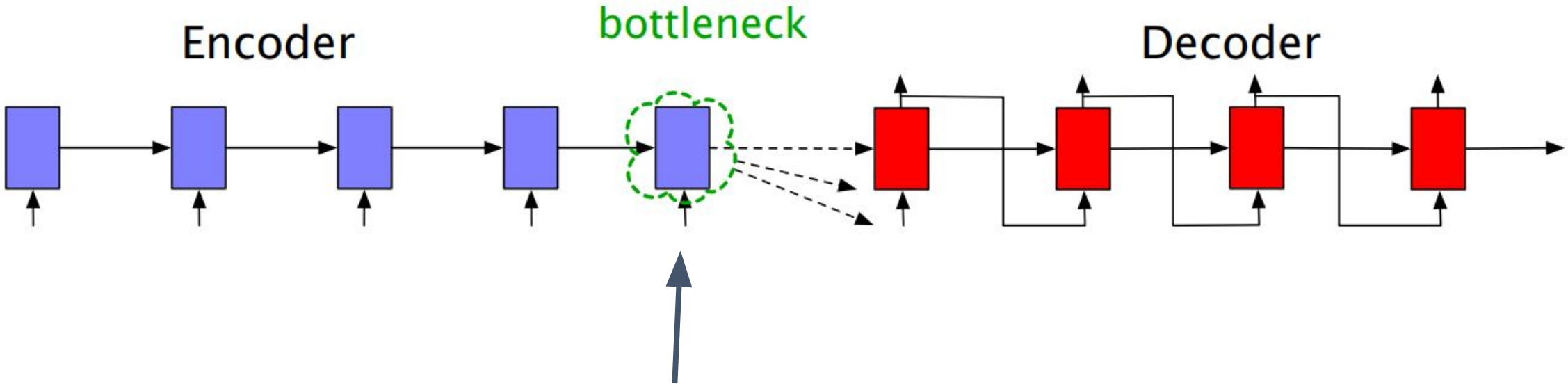


Encoder - Decoder

Teaching Forcing

- Force the system to use the gold target token from training as the next input x_{t+1}, rather than allowing it to rely on the (possibly erroneous) decoder output ˆyt .
- Speeds up training

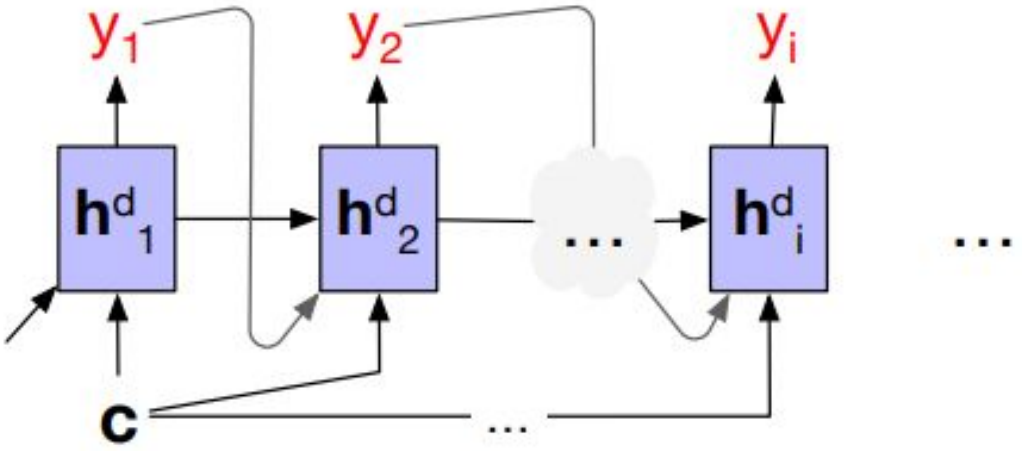
Attention !



In an encoder-decoder arch, the final hidden state acts as a bottleneck:

- It must represent absolutely everything about the meaning of the source text
- The only thing the decoder knows about the source text is what's in this context vector

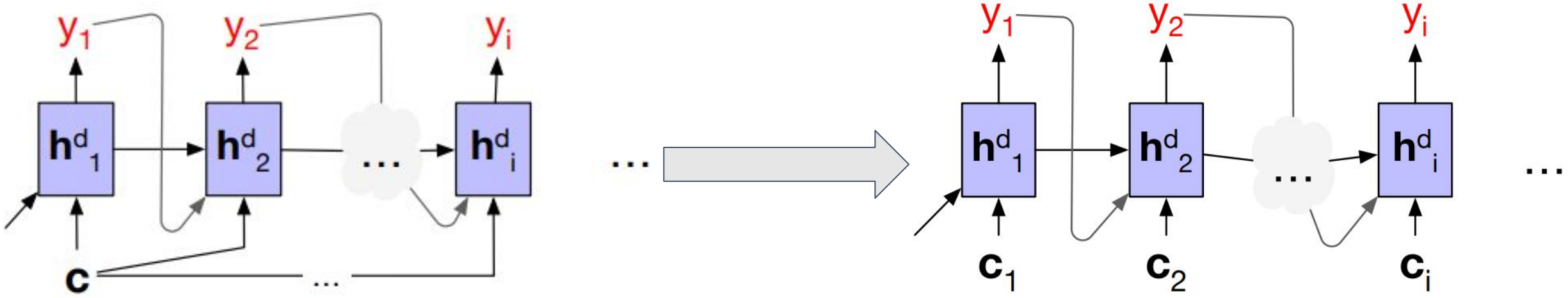
Attention !



Without attention, a decoder sees the same context vector , which is a static function of all the encoder hidden states

$$\mathbf{h}_t^d = g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c})$$

Attention !



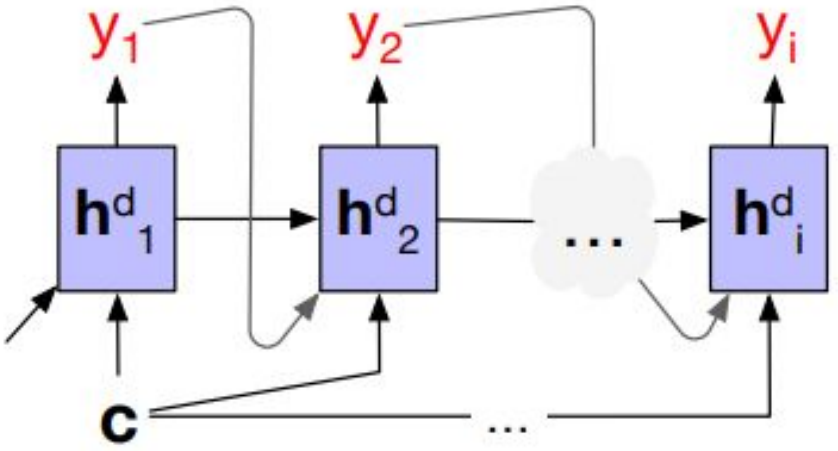
Without attention, a decoder sees the same context vector , which is a static function of all the encoder hidden states

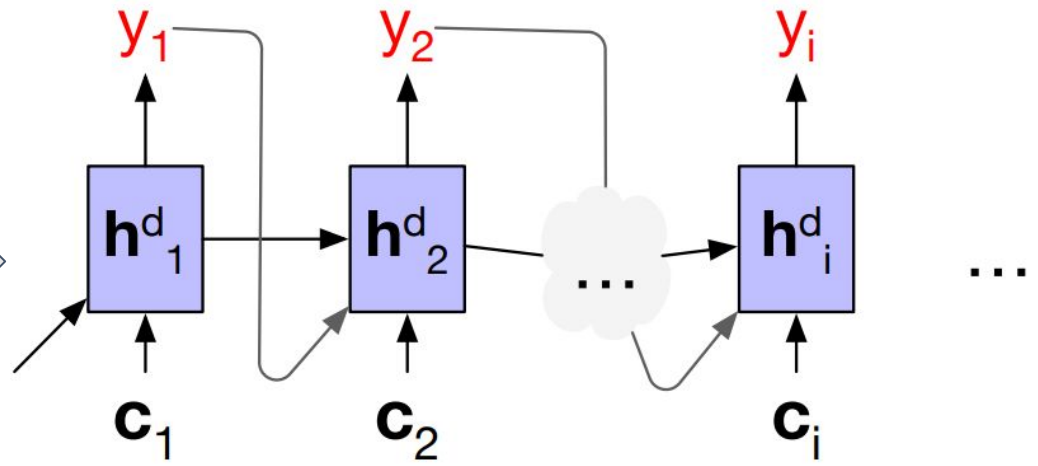
With attention, decoder to sees a different, dynamic, context, which is a function of all the encoder hidden states

$$\mathbf{h}_t^d = g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c})$$

$$\mathbf{h}_i^d = g(\hat{y}_{i-1}, \mathbf{h}_{i-1}^d, \mathbf{c}_i)$$

Attention !





Without attention, a decoder sees the same context vector , which is a static function of all the encoder hidden states

With attention, decoder to sees a different, dynamic, context, which is a function of all the encoder hidden states

$$\mathbf{h}_t^d = g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c})$$

$$\mathbf{h}_i^d = g(\hat{y}_{i-1}, \mathbf{h}_{i-1}^d, \mathbf{c}_i)$$

With attention, decoder gets information from all the hidden states of the encoder, not just the last hidden state of the encoder

Each context vector is obtained by taking a weighted sum of all the encoder hidden states.

The weights focus on ('attend to') a particular part of the source text that is relevant for the token the decoder is currently producing

Attention !

Step -1 : Find out how relevant each encoder state is to the present decoder state $\mathbf{h}_{i-1}^{d}$

Compute a score of similarity between $\mathbf{h}_{i-1}^{d}$ and all the encoder states : $score(\mathbf{h}_{i-1}^{d}, \mathbf{h}_{j}^{e})$

Dot Product Attention : $$\text{score}(\mathbf{h}_{i-1}^{d}, \mathbf{h}_{j}^{e}) = \mathbf{h}_{i-1}^{d} \cdot \mathbf{h}_{j}^{e}$$

Step -2 : Normalize all the scores with softmax to create a vector of weights, $\alpha_{i,j}$

$\alpha_{i,j}$ indicates the proportional relevance of each encoder hidden state j to the prior hidden decoder state, $\mathbf{h}_{i-1}^{d}$

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

$$= \frac{\exp(\text{score}(\mathbf{h}_{i-1}^{d}, \mathbf{h}_{j}^{e})}{\sum_{k} \exp(\text{score}(\mathbf{h}_{i-1}^{d}, \mathbf{h}_{k}^{e}))}$$

Attention !

Step -3 : Given the distribution in α, compute a fixed-length context vector for the current decoder state by taking a weighted average over all the encoder hidden states

$$\mathbf{c}_i = \sum_j \alpha_{ij}\, \mathbf{h}_j^e$$

Attention !

Step -3 : Given the distribution in α, compute a fixed-length context vector for the current decoder state by taking a weighted average over all the encoder hidden states

$$\mathbf{c}_i = \sum_j \alpha_{ij} \mathbf{h}_j^e$$

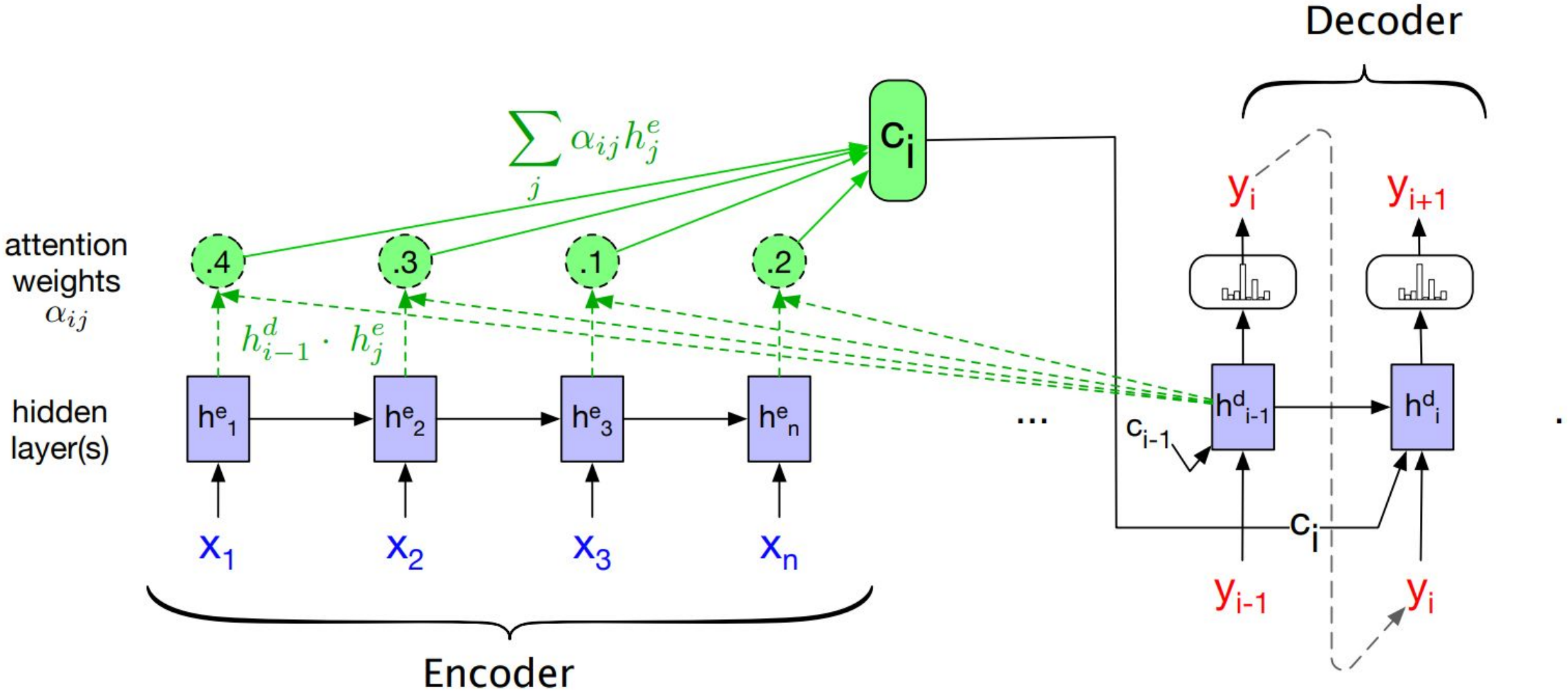
Plus : In step-1, we can get a more powerful scoring function by parameterizing the score with its own set of weights, W_s:

$$\text{score}(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e) = \mathbf{h}_{t-1}^d \mathbf{W}_s \mathbf{h}_j^e$$

W_s , is trained during normal end-to-end training,
W_s , gives the network the ability to learn which aspects of similarity between the decoder and encoder states are important to the current application.

Attention !

$$\mathbf{h}_i^d = g(\hat{y}_{i-1}, \mathbf{h}_{i-1}^d, \mathbf{c}_i)$$



Transformers

- 2017, NIPS, Vaswani et. al., **Attention Is All You Need** !!!
- Transformers **map sequences** of input vectors $(x_1,...,x_n)$ to sequences of output vectors $(y_1,...,y_n)$ **of the same length**
- Made up of **transformer blocks** in which the key component is **self-attention** layers

  [Self-attention allows a network to directly extract and use information from arbitrarily large contexts directly !!!]

- Transformers are **not based on recurrent connections** ⇒ Parallel implementations possible ⇒ Efficient to scale (comparing LSTM)

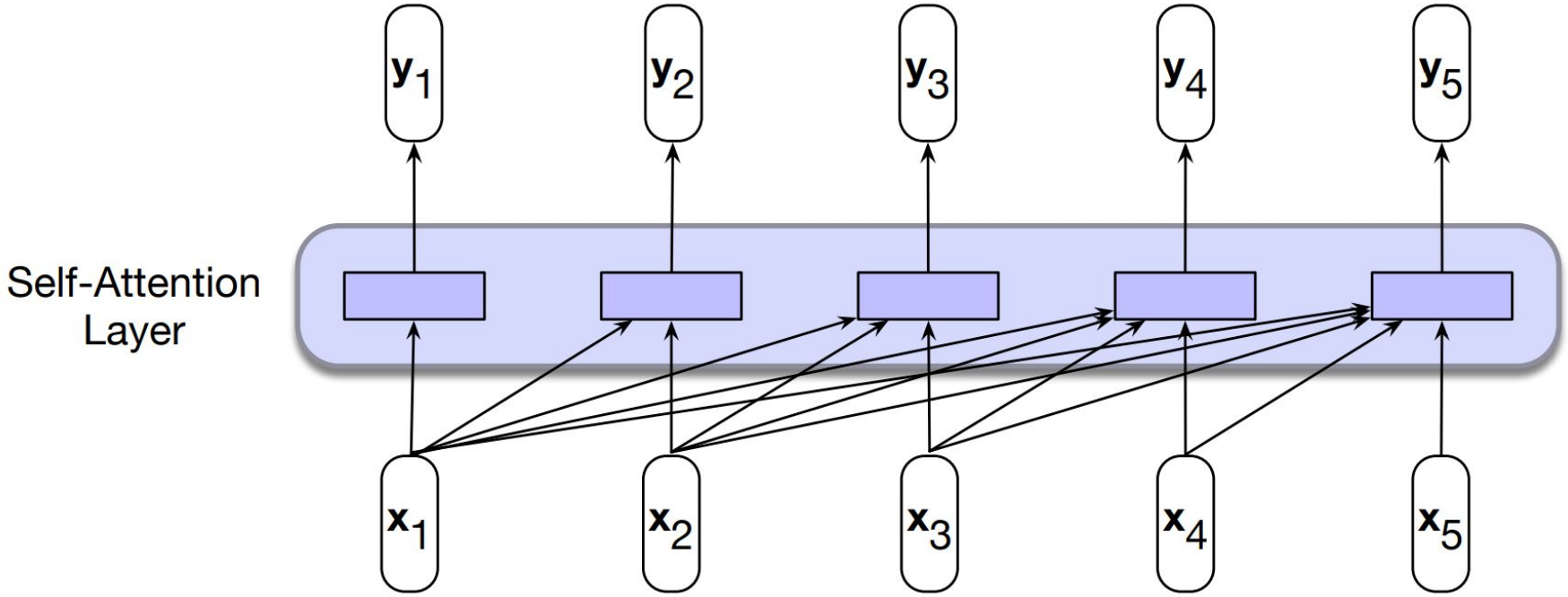
Self-Attention | Transformers

- Attention ⇒ Ability to compare an item of interest to a collection of other items in a way that reveals their relevance in the current context.
- Self-attention ⇒

  > Set of ***comparisons*** are to other elements ***within a given sequence***

  > Use these comparisons to compute an output for the current input

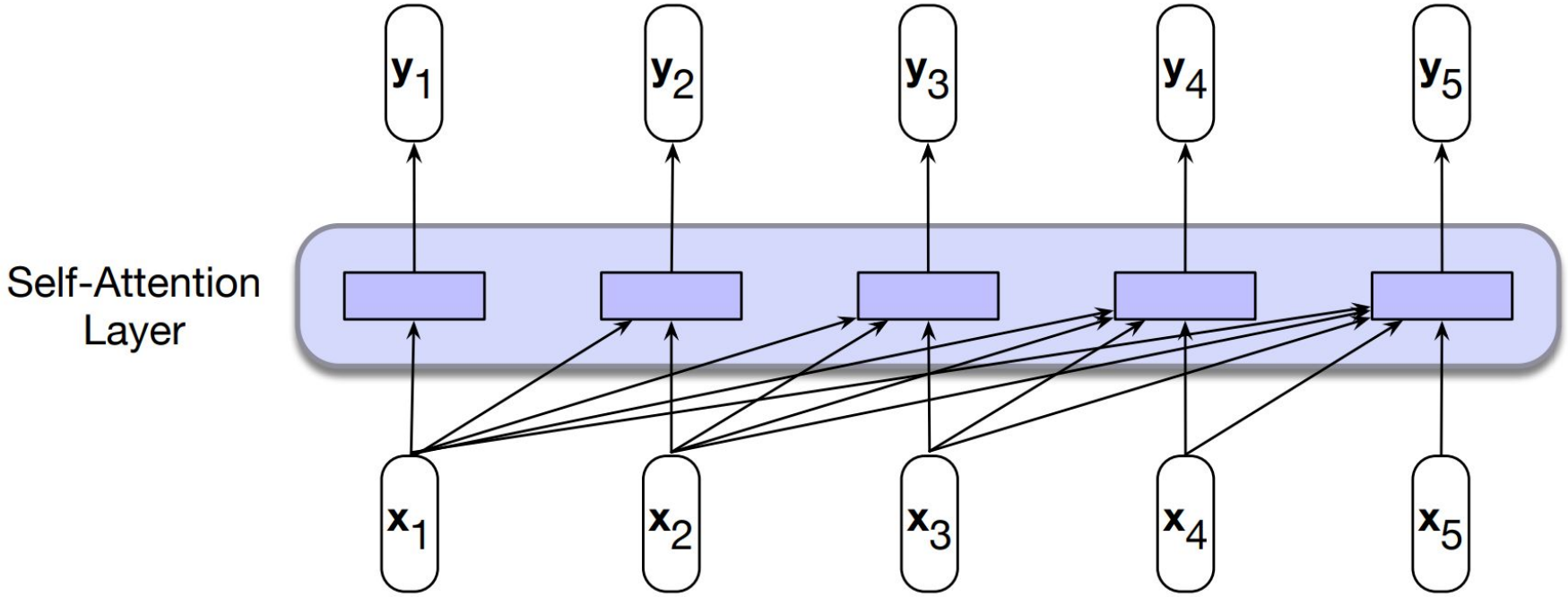
Self-Attention | Transformers



In processing each element of the sequence, the model attends to all the inputs up to, and including, the current one.

Unlike RNNs, the computations at each time step are independent of all the other steps and therefore can be performed in parallel.

Self-Attention | Transformers



$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$$

$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \;\; \forall j \leq i$$

$$= \frac{\exp(\text{score}(\mathbf{x}_i, \mathbf{x}_j))}{\sum_{k=1}^{i} \exp(\text{score}(\mathbf{x}_i, \mathbf{x}_k))} \;\; \forall j \leq i$$

$$\mathbf{y}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{x}_j$$

Self-Attention | Transformers

- Let us understand how transformers uses self-attention !

Self-Attention | Transformers

$$\mathbf{W}^V, \mathbf{W}^K, \mathbf{W}^Q \in \mathbb{R}^{d \times d}$$

In Vaswani et al., 2017, d was 1024.

- Let us understand how transformers uses self-attention !

$$\mathbf{q}_i = \mathbf{W}^{\mathbf{Q}}\mathbf{x}_i \qquad \mathbf{k}_i = \mathbf{W}^{\mathbf{K}}\mathbf{x}_i \qquad \mathbf{v}_i = \mathbf{W}^{\mathbf{V}}\mathbf{x}_i$$

Query, Q	Key, K	Value, V
As the current focus of attention when being compared to all of the other preceding inputs.	In its role as a preceding input being compared to the current focus of attention.	As a value used to compute the output for the current focus of attention

Three different roles each x_i (input embedding) , in the computation of self attention

Self-Attention | Transformers

$$\mathbf{W}^V, \mathbf{W}^K, \mathbf{W}^Q \in \mathbb{R}^{d \times d}$$

In Vaswani et al., 2017, d was 1024.

- Let us understand how transformers uses self-attention !

$$\mathbf{q}_i = \mathbf{W}^{\mathbf{Q}}\mathbf{x}_i \qquad \mathbf{k}_i = \mathbf{W}^{\mathbf{K}}\mathbf{x}_i \qquad \mathbf{v}_i = \mathbf{W}^{\mathbf{V}}\mathbf{x}_i$$

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}$$

The simple dot product can be an arbitrarily large; scaled dot-product is used in transformers;

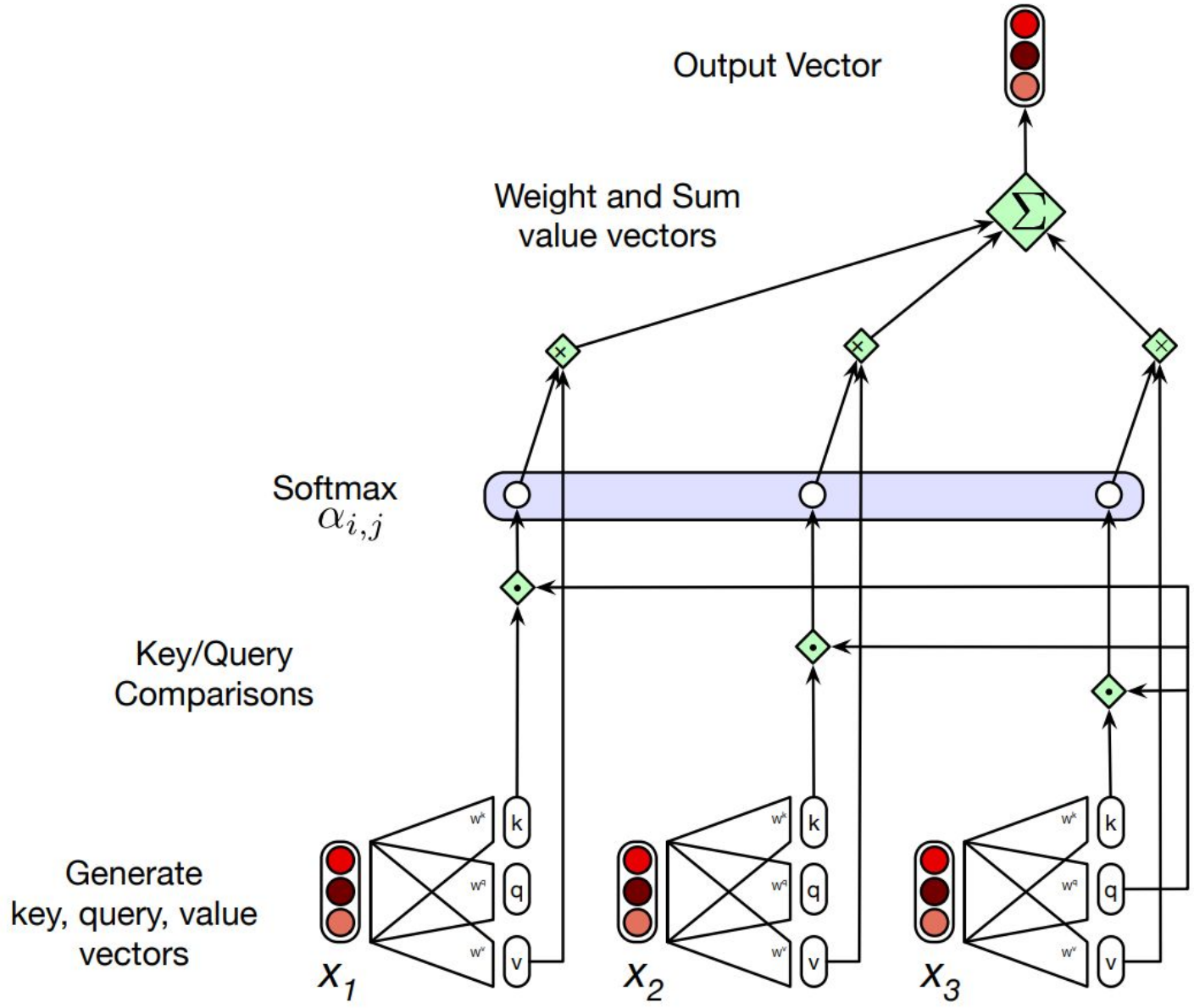
$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \ \forall j \le i$$

$$= \frac{\exp(\text{score}(\mathbf{x}_i, \mathbf{x}_j))}{\sum_{k=1}^{i} \exp(\text{score}(\mathbf{x}_i, \mathbf{x}_k))} \ \forall j \le i$$

$$\mathbf{y}_i = \sum_{j \le i} \alpha_{ij} \mathbf{v}_j$$

Self-Attention | Transformers

- Each output, y_i , is computed independently
- Entire process can be parallelized



Calculating the value of y3, the third element of a sequence using causal (left-to-right) self-attention

Self-Attention | Transformers

- Pack the input embeddings of the N input tokens into a single matrix

$$\mathbf{X} \in \mathbb{R}^{N \times d}$$

> Each row of X is the embedding of one token of the input

- Multiply X by the key, query, and value (dxd) matrices

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^{\mathbf{Q}}; \quad \mathbf{K} = \mathbf{X}\mathbf{W}^{\mathbf{K}}; \quad \mathbf{V} = \mathbf{X}\mathbf{W}^{\mathbf{V}}$$

Self-Attention | Transformers

$$\text{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\intercal}{\sqrt{d_k}}\right)\mathbf{V}$$

$\mathbf{QK}^\intercal$ Matrix

N

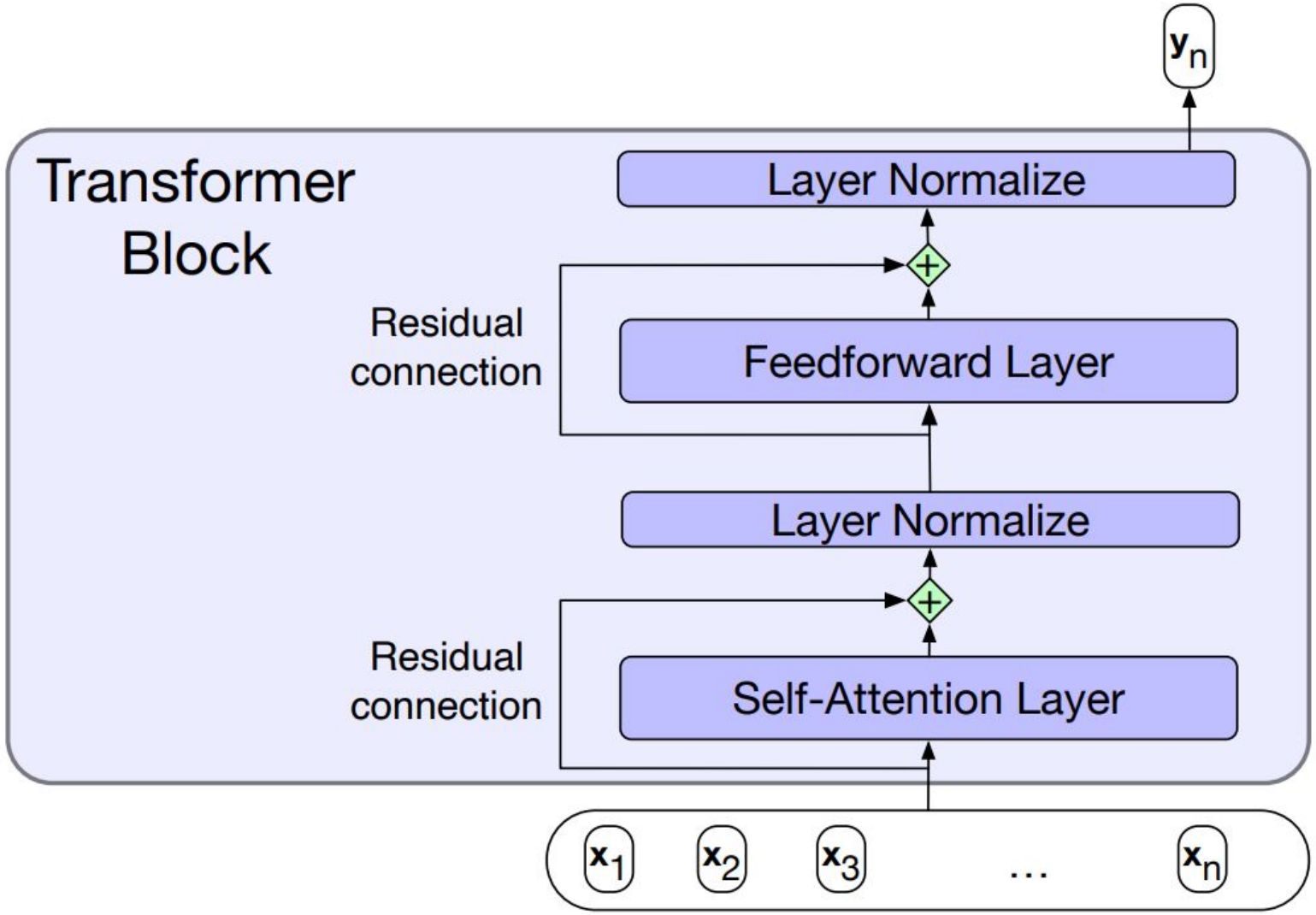
q1•k1	−∞	−∞	−∞	−∞
q2•k1	q2•k2	−∞	−∞	−∞
q3•k1	q3•k2	q3•k3	−∞	−∞
q4•k1	q4•k2	q4•k3	q4•k4	−∞
q5•k1	q5•k2	q5•k3	q5•k4	q5•k5

N

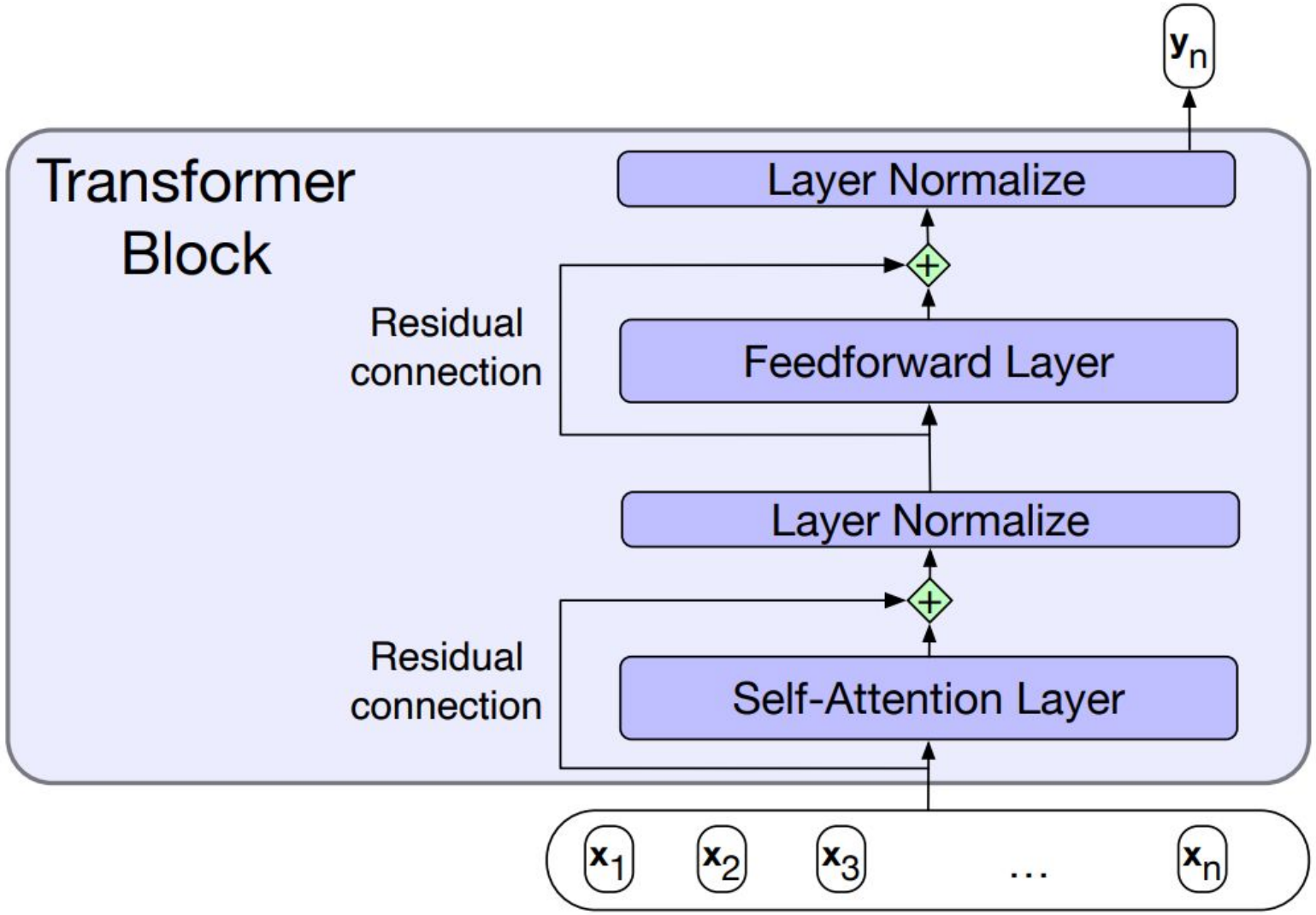
Note: Upper-triangle portion of the comparisons matrix zeroed out (set to −∞, which the softmax will turn to zero)

Transformer Blocks | Transformers

$$\mathbf{z} = \text{LayerNorm}(\mathbf{x} + \text{SelfAttention}(\mathbf{x}))$$
$$\mathbf{y} = \text{LayerNorm}(\mathbf{z} + \text{FFN}(\mathbf{z}))$$



Transformer Blocks | Transformers



$$\mathbf{z} = \text{LayerNorm}(\mathbf{x} + \text{SelfAttention}(\mathbf{x}))$$
$$\mathbf{y} = \text{LayerNorm}(\mathbf{z} + \text{FFN}(\mathbf{z}))$$

LayerNorm:

$$\mu = \frac{1}{d_h}\sum_{i=1}^{d_h} x_i$$

$$\sigma = \sqrt{\frac{1}{d_h}\sum_{i=1}^{d_h}(x_i - \mu)^2}$$

$$\hat{\mathbf{x}} = \frac{(\mathbf{x} - \mu)}{\sigma}$$

$$\text{LayerNorm} = \gamma\,\hat{\mathbf{x}} + \beta$$

Multihead-Attention | Transformers

- Different words in a sentence can relate to each other in many different ways simultaneously

  >> A single transformer block to learn to capture all of the different kinds of parallel relations among its inputs is inadequate.

- Multihead self-attention layers

  >> Heads ⇒ sets of self-attention layers, that reside in parallel layers at the same depth in a model, each with its own set of parameters.

  >> Each head learn different aspects of the relationships that exist among inputs at the same level of abstraction

Multihead-Attention | Transformers

- Different words in a sentence can relate to each other in many different ways simultaneously

  >> A single transformer block to learn to capture all of the different kinds of parallel relations among its inputs is inadequate.

- Multihead self-attention layers

  >> Heads ⇒ sets of self-attention layers, that reside in parallel layers at the same depth in a model, each with its own set of parameters.

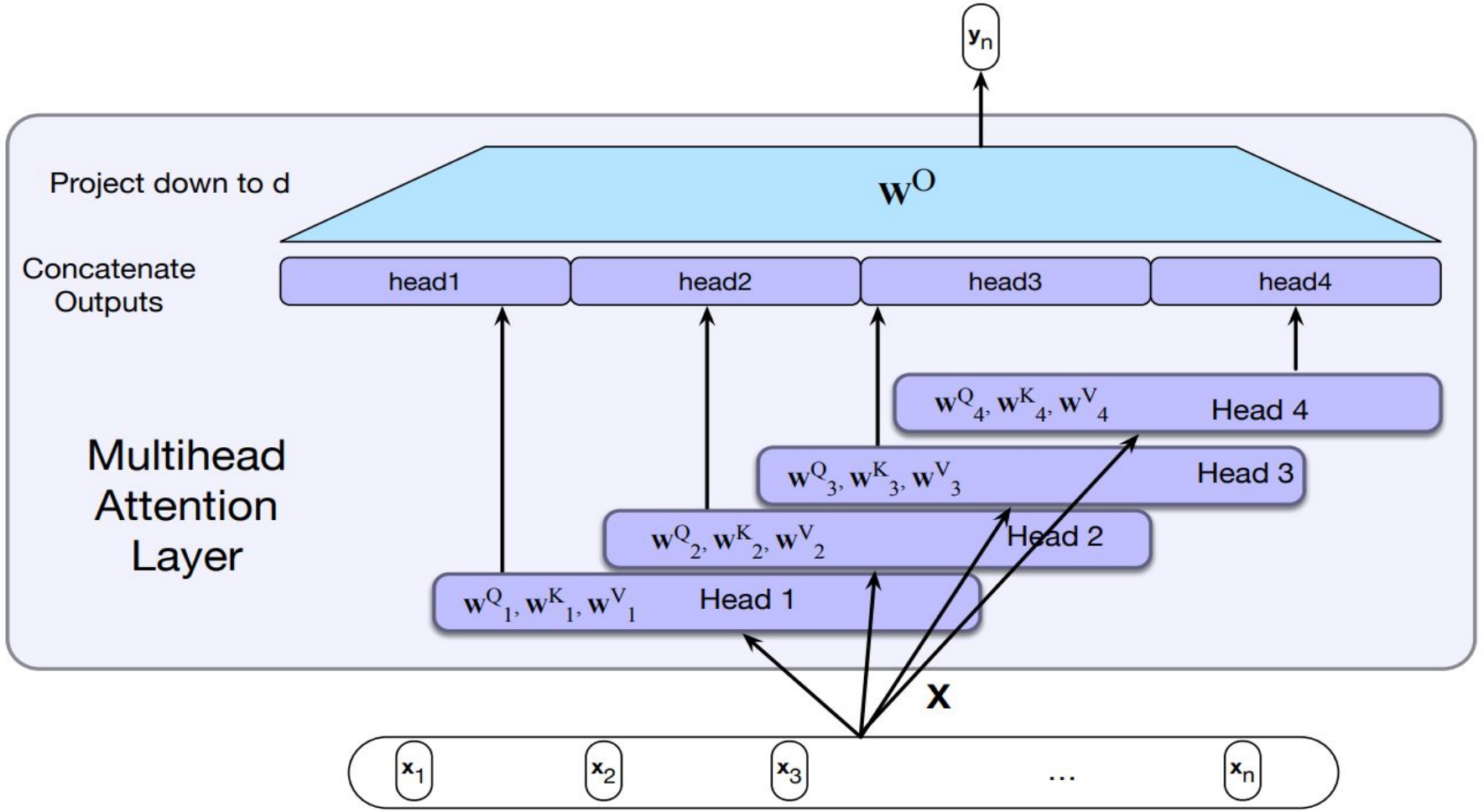
  >> Each head learn different aspects of the relationships that exist among inputs at the same level of abstraction

$$\text{MultiHeadAttention}(\mathbf{X}) = (\mathbf{head}_1 \oplus \mathbf{head}_2 \ldots \oplus \mathbf{head}_h)\mathbf{W}^O$$

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_i^Q \; ; \; \mathbf{K} = \mathbf{X}\mathbf{W}_i^K \; ; \; \mathbf{V} = \mathbf{X}\mathbf{W}_i^V$$

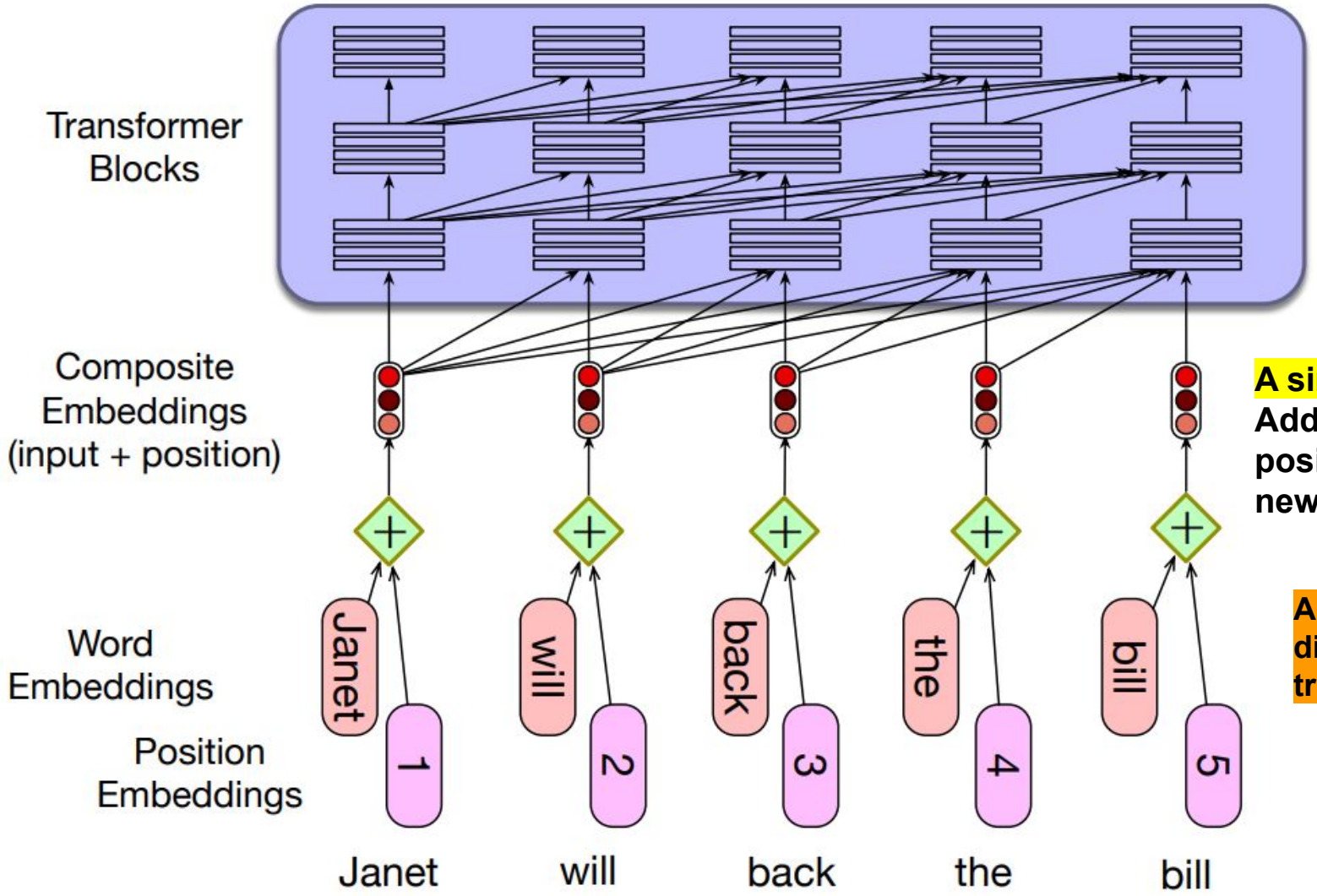
$$\mathbf{head}_i = \text{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$$

Multihead-Attention | Transformers



Each of the multihead self-attention layers is provided with its own set of key, query and value weight matrices. The outputs from each of the layers are concatenated and then projected down to d, thus producing an output of the same size as the input so layers can be stacked.

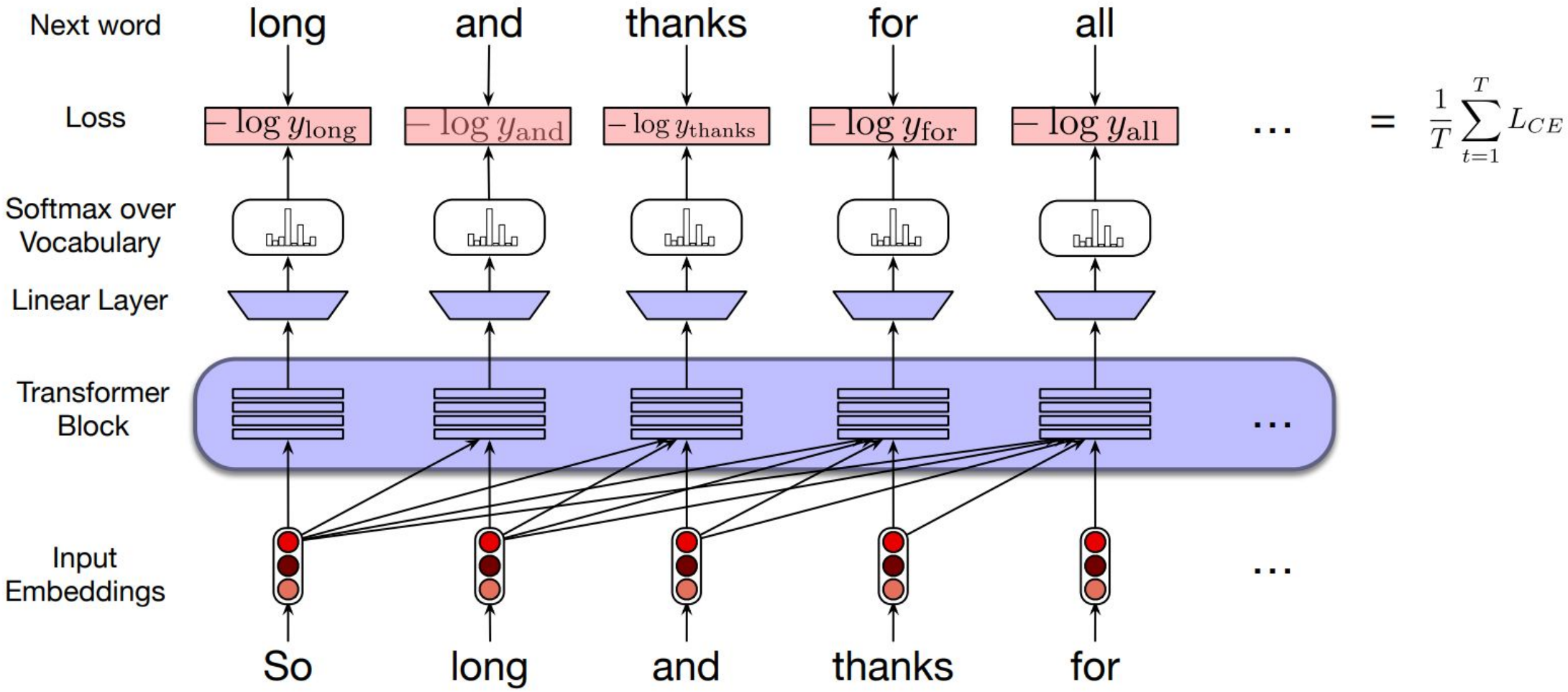
Positional Embeddings | Transformers



A simple way to model position:
Add an embedding representation of the absolute position to the input word embedding to produce a new embedding of the same dimensionality.

A combination of sine and cosine functions with differing frequencies was used in the original transformer work.

Transformers as Language Models



Vision Transformers (ViT)

- Ref: Alexey et. al.(2021) , ICLR, An Image Is Worth 16x16 Words: Transformers For Image Recognition At Scale
-

Readings

1. Draft Chapter 9 and Draft Chapter 10 , Speech and Language Processing. Daniel Jurafsky & James H. Martin.
2. Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).
3. Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." arXiv preprint arXiv:2010.11929 (2020).

Thank you