CS 383: Machine Learning

Prof Adam Poliak

Fall 2024

11/21/2024

Lecture 26

Announcements – Remaining Assignments

HW07: due Wednesday 11/27

HW08: due Friday 12/06 (might extend this too)

Project Presentations – 12/17 10:00am

- Poll for another time

Outline

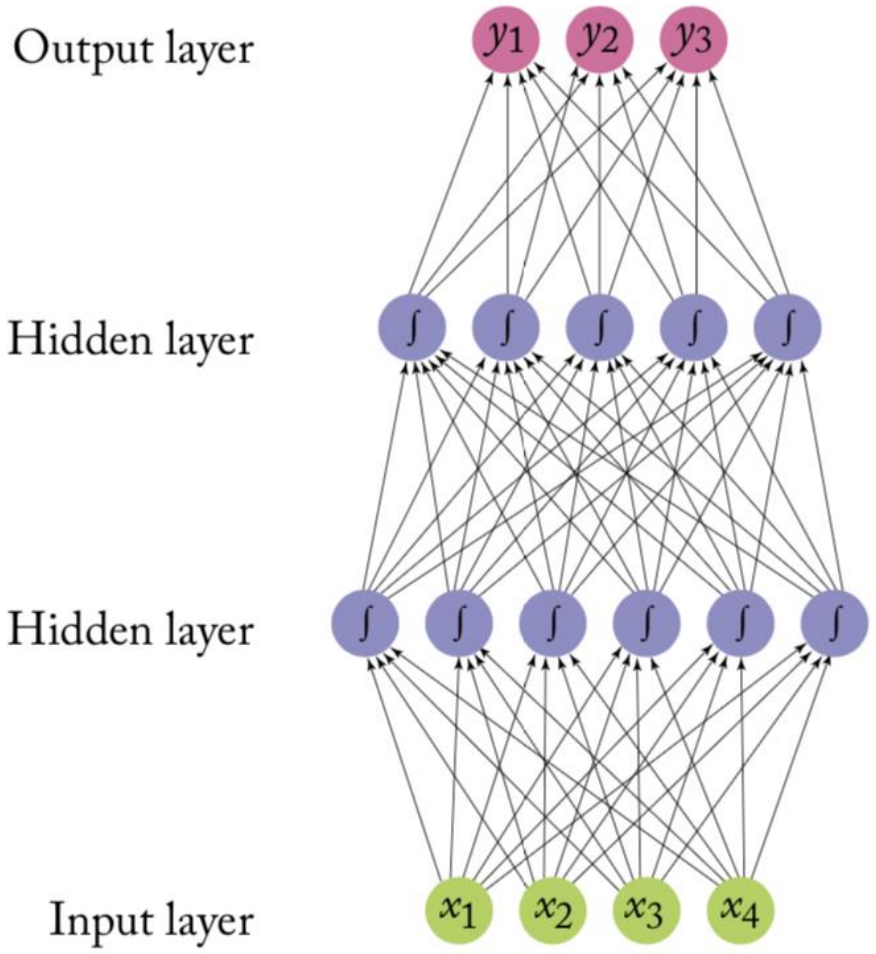
Word Embedding & FNN issue

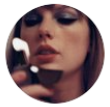
RNN - review

Attention

Self Attention

Classify a tweet as viral or not

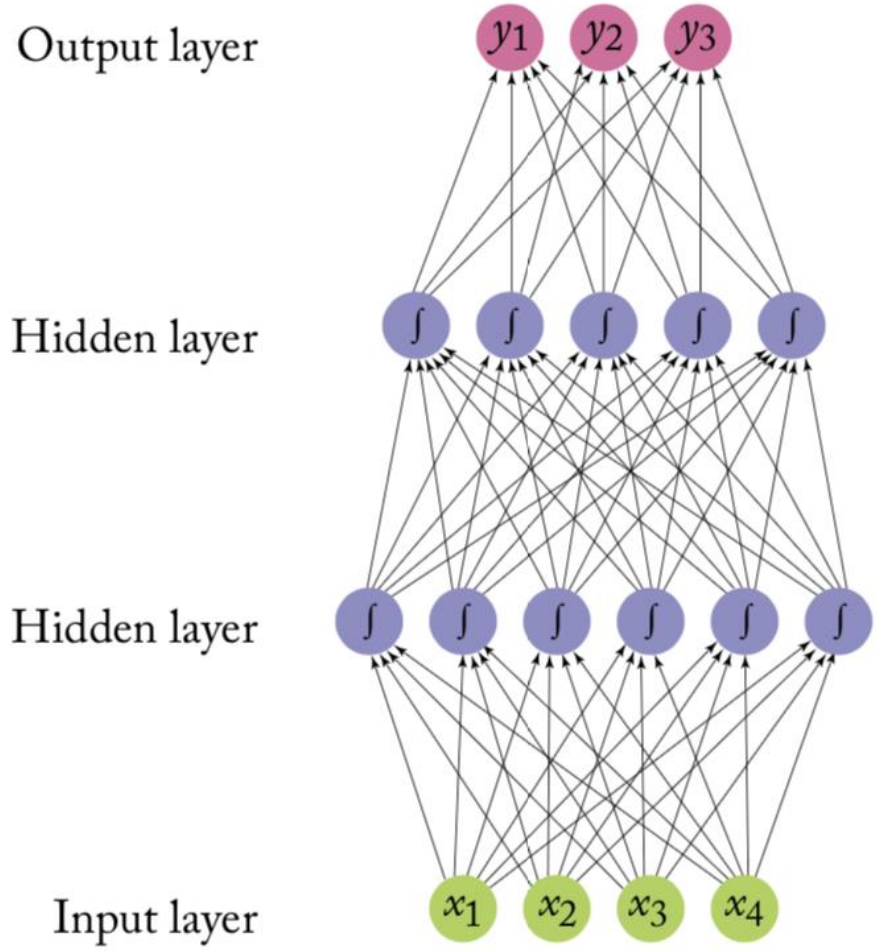


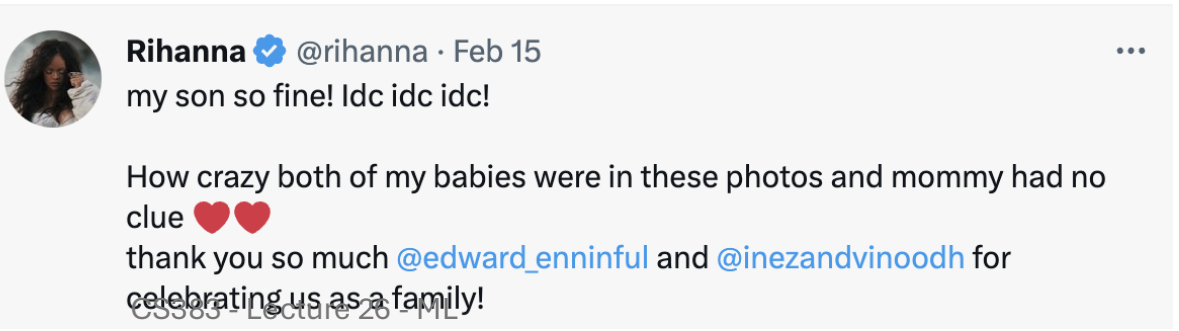


Taylor Swift @taylorswift13 · Jan 27

The Lavender Haze video is out now. There is lots of lavender. There is lots of haze. There is my incredible costar @laith_ashley who I absolutely adored working with.

Classify a tweet as viral or not





Rihanna @rihanna · Feb 15

my son so fine! Idc idc idc!

How crazy both of my babies were in these photos and mommy had no clue ❤️❤️
thank you so much @edward_enninful and @inezandvinoodh for celebrating us as a family!

Approach 1: Feature Engineering

Create a fixed-length vector to represent the input

Examples:

- Length of vector is V: size of vocabulary
- Value at index i represents static of word i
  - How many times word i appears in input
  - Term-frequency inverse document frequency of word i
    - Penalize words that appear in lots of inputs

Approach 1: Feature Engineering

Create a fixed-length vector to represent the input

$x = w_1, w_2, \ldots w_n$

$z_0 = [count(w_1), count(w_2), count(w_3), \ldots, count(w_v)]$

$\hat{y} = MLP(z_o)$

Approach 2: Deep Averaging Network

Represent each document as a continuous bag of words, averaging the word embeddings

$$x = w_1, w_2, \ldots w_n$$

$$z_0 = CBOW(w_1, w_2, \ldots w_n). CBOW = \sum_i E[w_i]$$

$$\hat{y} = MLP(z_o)$$

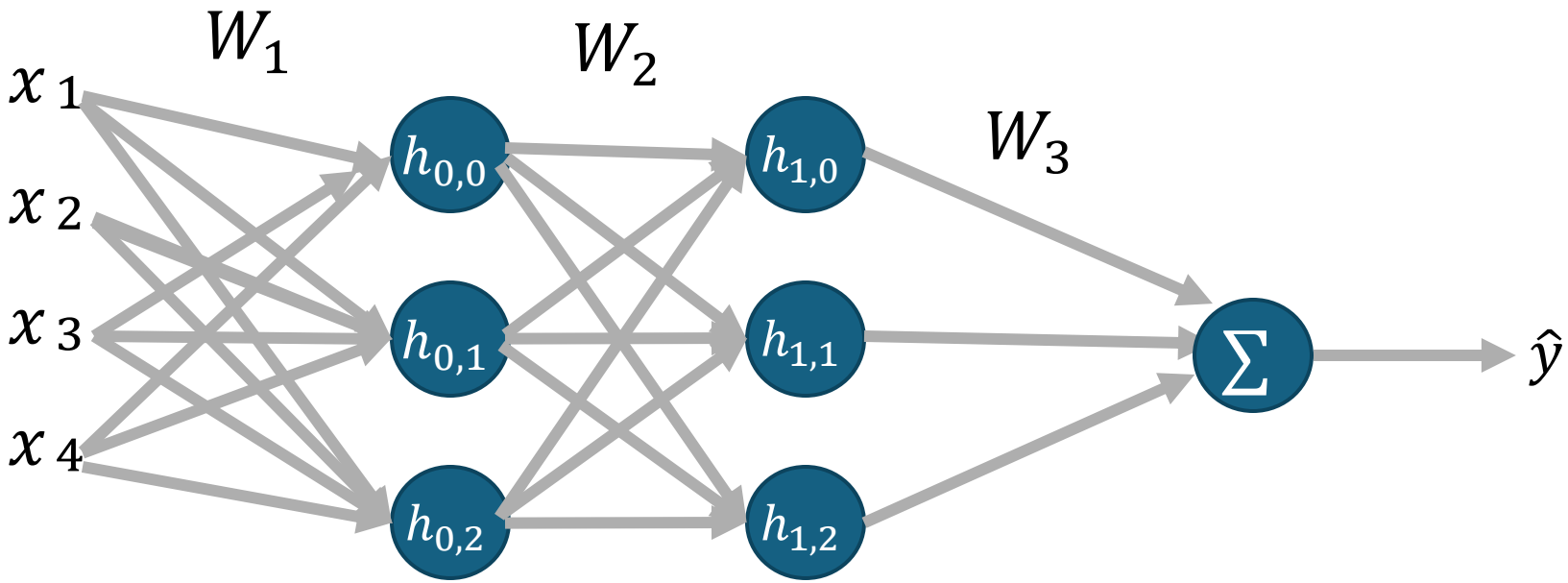
Multilayer Perceptron

Feed-forward NN

$$MLP_1 = g(xW_1 + b_1)W_2 + b_2$$

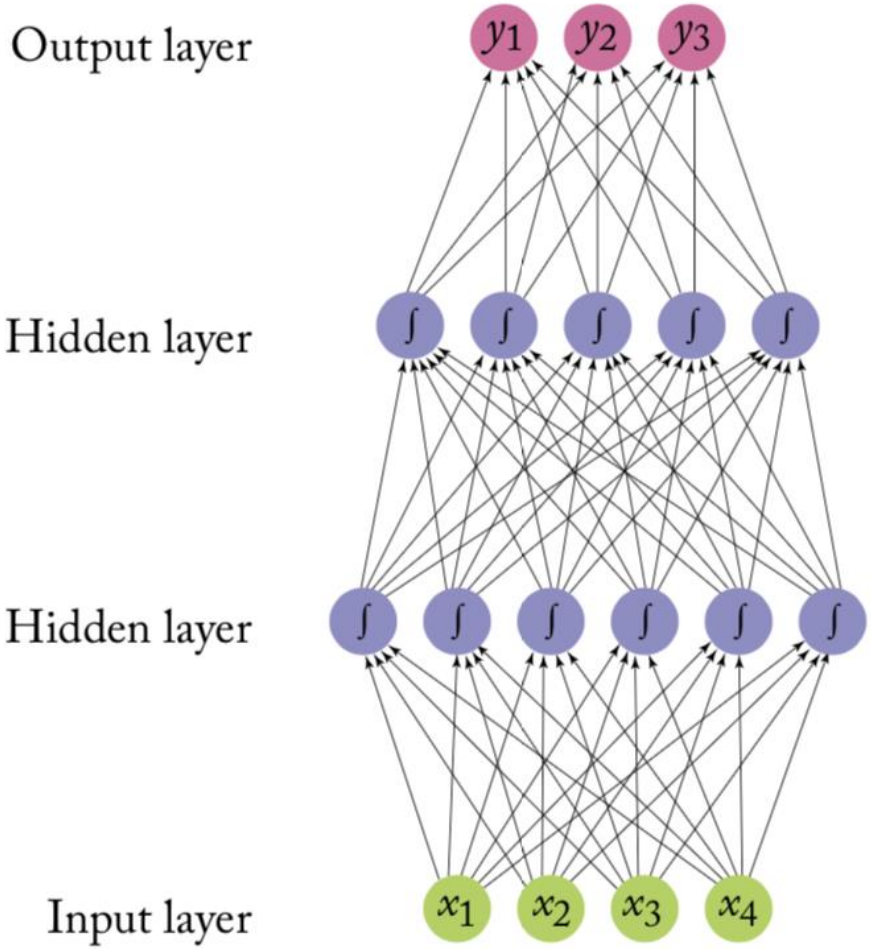
$$MLP_2 = g(g(xW_1 + b_1)W_2 + b_2)\,W_3 + b_3$$

MLP_2



$$\boldsymbol{h}_0 = \sigma(xW_1) \qquad \boldsymbol{h}_1 = \sigma(\sigma(xW_1)W_2$$

MLP_2


Output layer
Hidden layer
Hidden layer
Input layer
y_1
y_2
y_3
x_1
x_2
x_3
x_4

FFN’s issues

Fixed input size

Solutions:

1. Create a fixed length representation
2. Recurrent Neural Networks

Outline

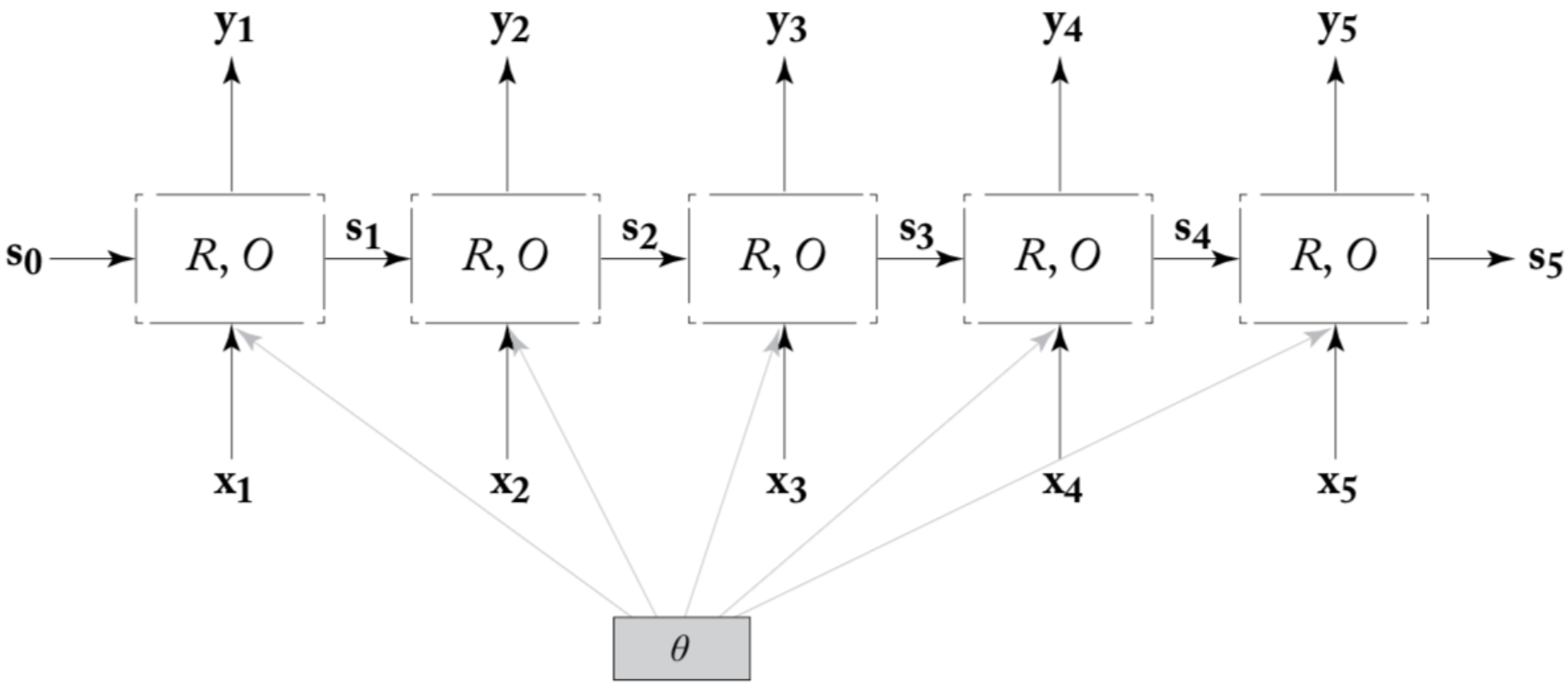
Word Embedding & FNN issue

RNN - review

Attention

Self Attention

Unrolling RNN



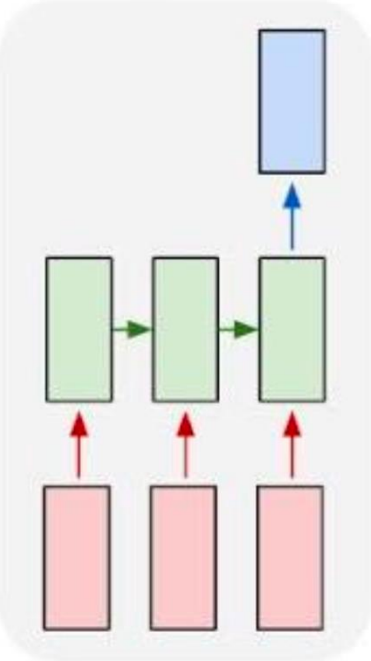
Extracting a vector from RNN for prediction

Acceptor:

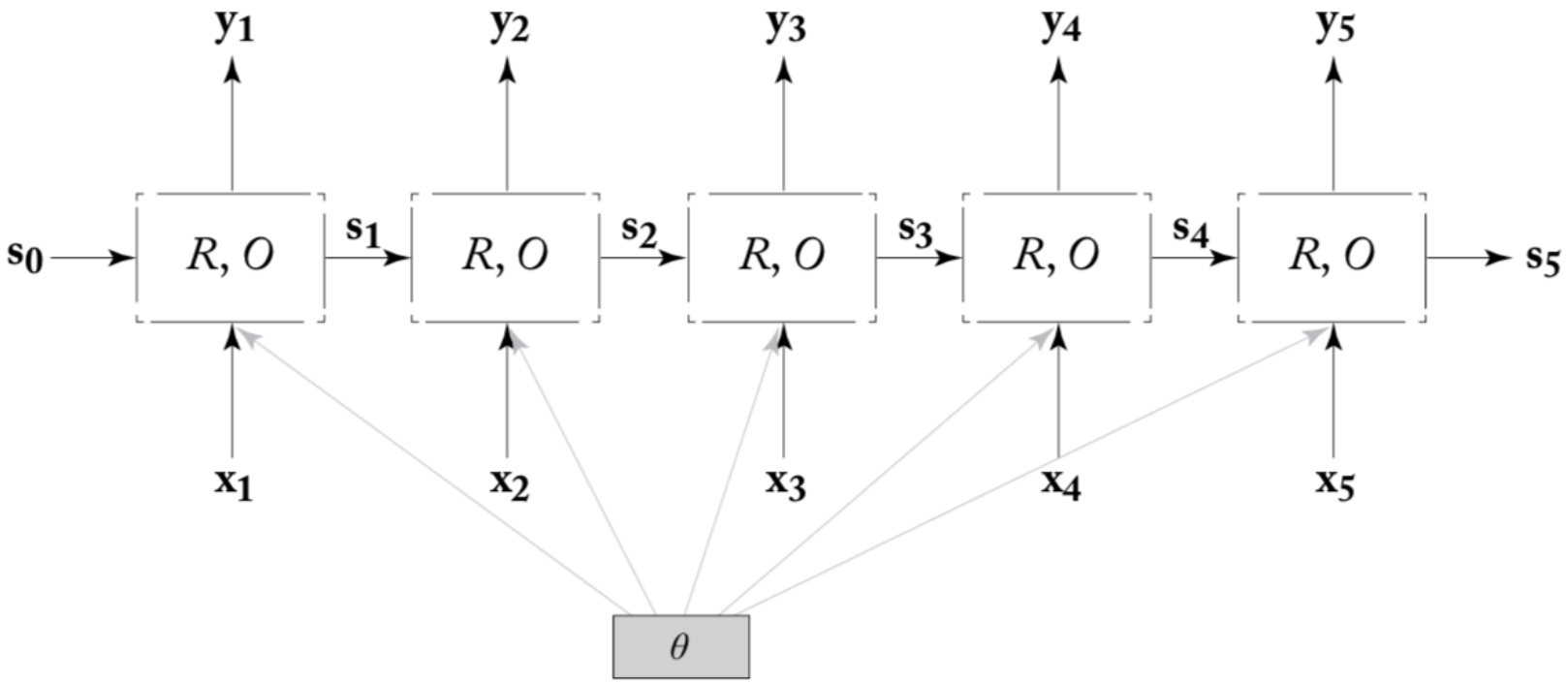
- Take the output of the last cell

Transducer:

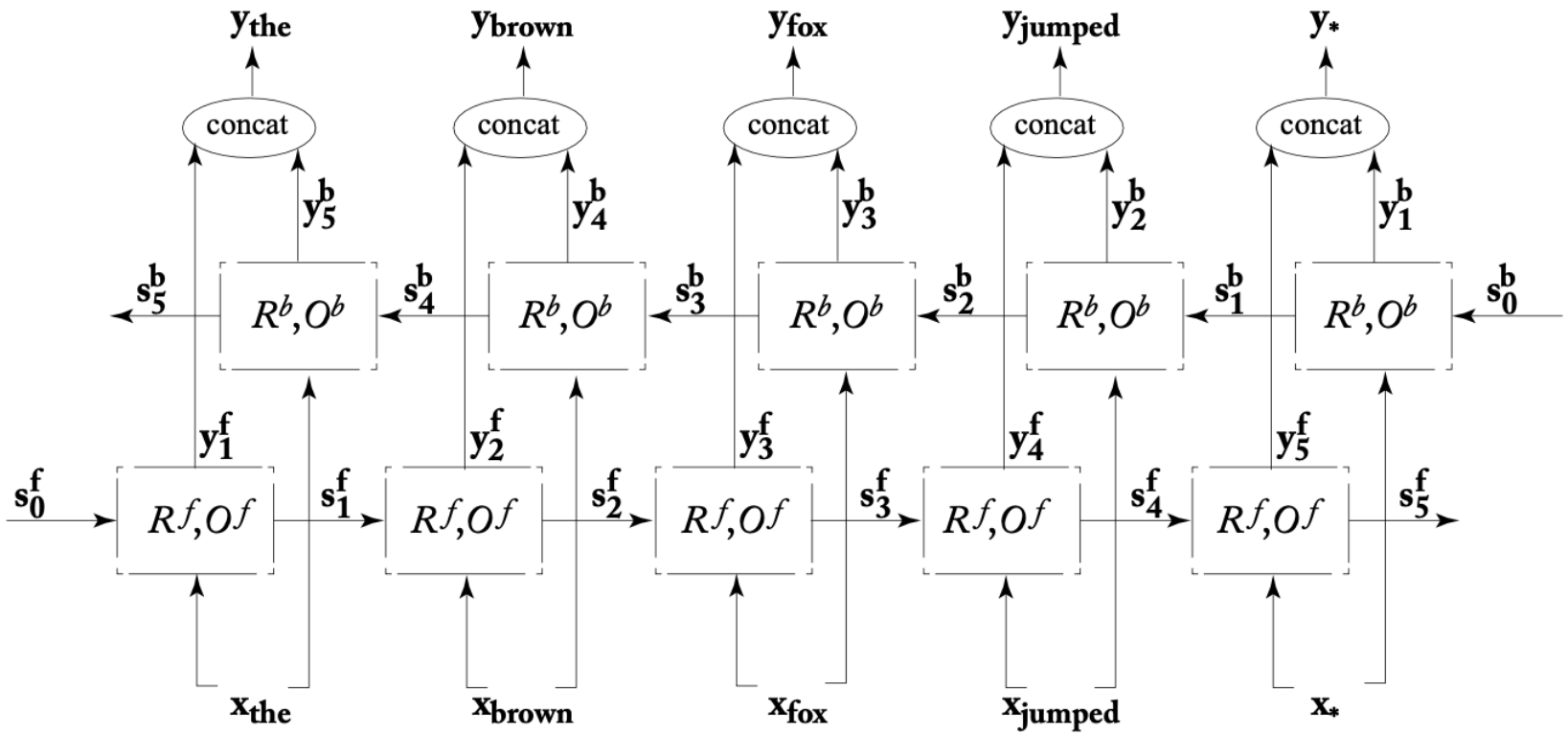
- Combine the output from each input
- Max Pooling
- Mean Pooling
- Mean Pooling



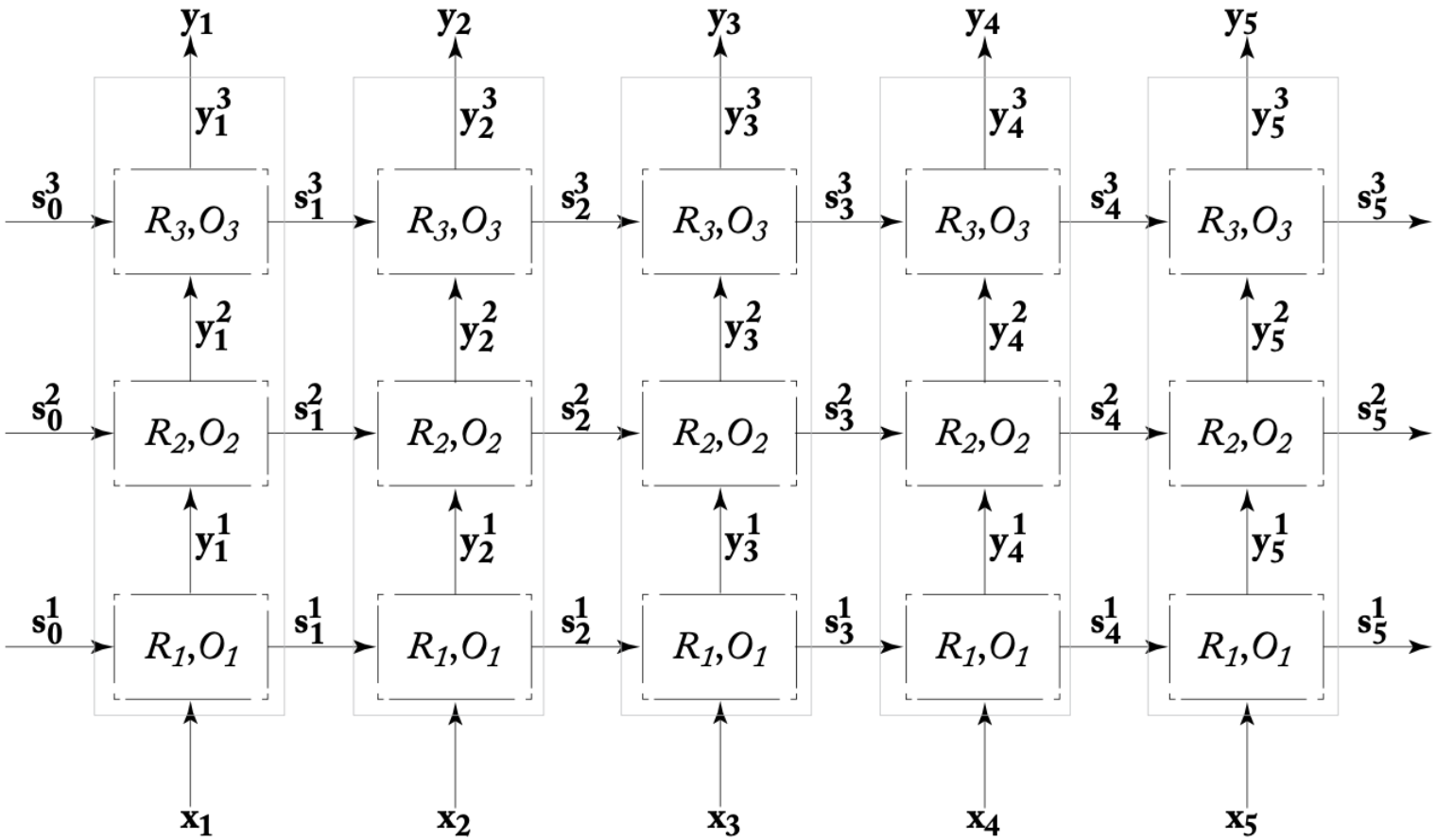
How else can we expand this?



Bi-directional



Stack more layers



Pytorch - nn.RNN

Parameters:

- **input_size** – The number of expected features in the input *x*
- **hidden_size** – The number of features in the hidden state *h*
- **num_layers** – Number of recurrent layers. E.g., setting `num_layers=2` would mean stacking two RNNs together to form a *stacked RNN*, with the second RNN taking in outputs of the first RNN and computing the final results. Default: 1
- **nonlinearity** – The non-linearity to use. Can be either `'tanh'` or `'relu'`. Default: `'tanh'`
- **bias** – If `False`, then the layer does not use bias weights *b_ih* and *b_hh*. Default: `True`
- **batch_first** – If `True`, then the input and output tensors are provided as (*batch, seq, feature*) instead of (*seq, batch, feature*). Note that this does not apply to hidden or cell states. See the Inputs/Outputs sections below for details. Default: `False`
- **dropout** – If non-zero, introduces a *Dropout* layer on the outputs of each RNN layer except the last layer, with dropout probability equal to `dropout`. Default: 0
- **bidirectional** – If `True`, becomes a bidirectional RNN. Default: `False`

RNNs – long input

RNNs can remember anything (in theory)

Sometimes its important to forget

Solution: Long-Short Term Memory (LSTM)

Pytorch – nn.LSTM

```
>>> rnn = nn.LSTM(10, 20, 2)
>>> input = torch.randn(5, 3, 10)
>>> h0 = torch.randn(2, 3, 20)
>>> c0 = torch.randn(2, 3, 20)
>>> output, (hn, cn) = rnn(input, (h0, c0))
```

LSTM

CLASS torch.nn.LSTM(*input_size*, *hidden_size*, *num_layers=1*, *bias=True*, *batch_first=False*, *dropout=0.0*, *bidirectional=False*, *proj_size=0*, *device=None*, *dtype=None*) [SOURCE]

Apply a multi-layer long short-term memory (LSTM) RNN to an input sequence. For each element in the input sequence, each layer computes the following function:

$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

where h_t is the hidden state at time *t*, c_t is the cell state at time *t*, x_t is the input at time *t*, h_{t-1} is the hidden state of the layer at time *t-1* or the initial hidden state at time *0*, and i_t, f_t, g_t, o_t are the input, forget, cell, and output gates, respectively. σ is the sigmoid function, and $\odot$ is the Hadamard product.

In a multilayer LSTM, the input $x_t^{(l)}$ of the l -th layer ($l \geq 2$) is the hidden state $h_t^{(l-1)}$ of the previous layer multiplied by dropout $\delta_t^{(l-1)}$ where each $\delta_t^{(l-1)}$ is a Bernoulli random variable which is 0 with probability `dropout`.

If `proj_size > 0` is specified, LSTM with projections will be used. This changes the LSTM cell in the following way. First, the dimension of h_t will be changed from `hidden_size` to `proj_size` (dimensions of W_{hi} will be changed accordingly). Second, the output hidden state of each layer will be multiplied by a learnable projection matrix: $h_t = W_{hr}h_t$. Note that as a consequence of this, the output of LSTM network will be of different shape as well. See Inputs/Outputs sections below for exact dimensions of all variables. You can find more details in https://arxiv.org/abs/1402.1128.

Parameters

- **input_size** – The number of expected features in the input *x*
- **hidden_size** – The number of features in the hidden state *h*
- **num_layers** – Number of recurrent layers. E.g., setting `num_layers=2` would mean stacking two LSTMs together to form a *stacked LSTM*, with the second LSTM taking in outputs of the first LSTM and computing the final results. Default: 1
- **bias** – If `False`, then the layer does not use bias weights *b_ih* and *b_hh*. Default: `True`
- **batch_first** – If `True`, then the input and output tensors are provided as (*batch, seq, feature*) instead of (*seq, batch, feature*). Note that this does not apply to hidden or cell states. See the Inputs/Outputs sections below for details. Default: `False`

Outline

Word Embedding & FNN issue

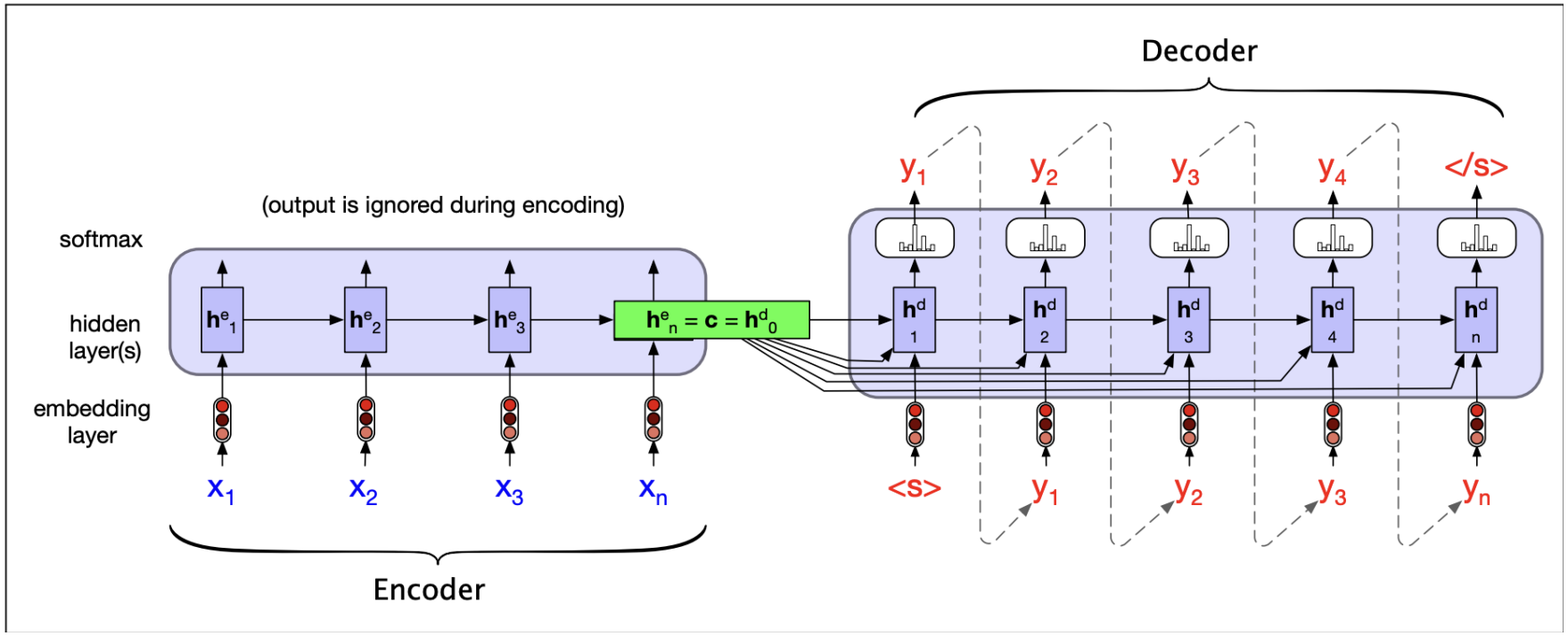
RNN - review

Attention

Self Attention

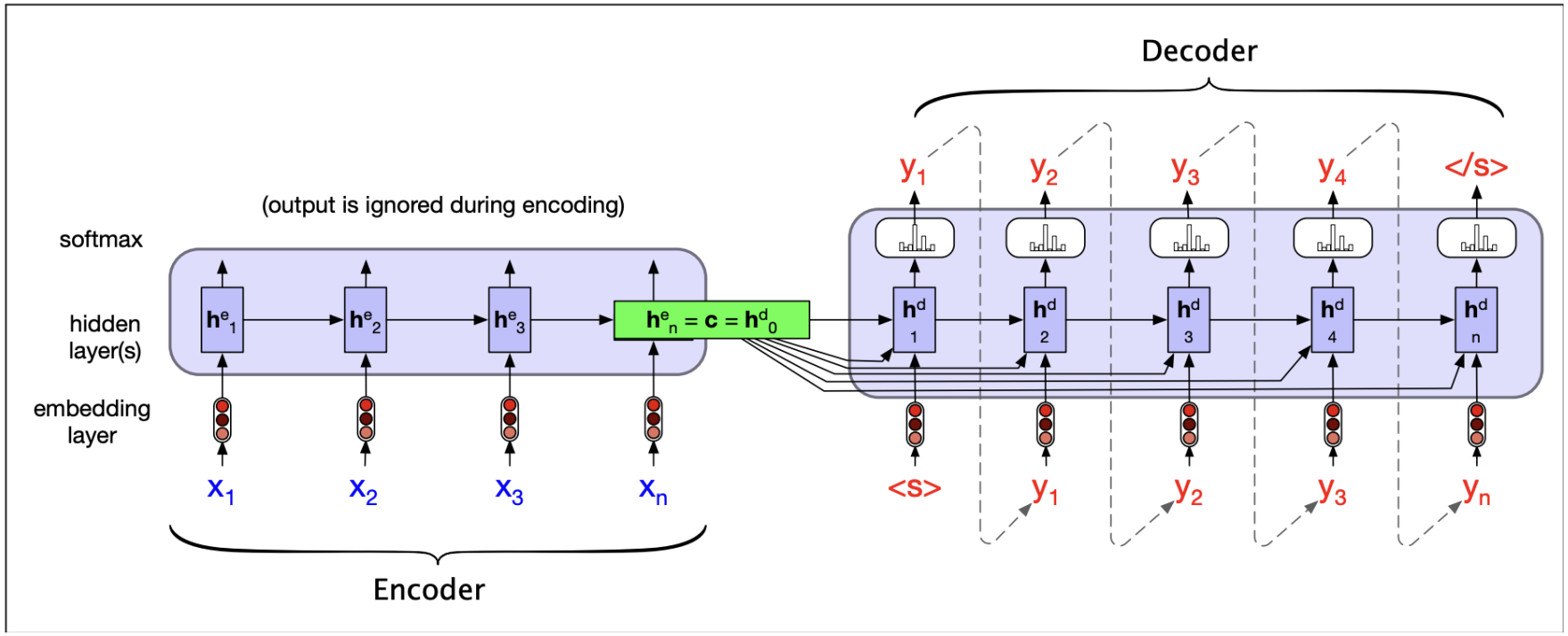
Encoder-decoder

Decoder only uses information from last hidden cell!



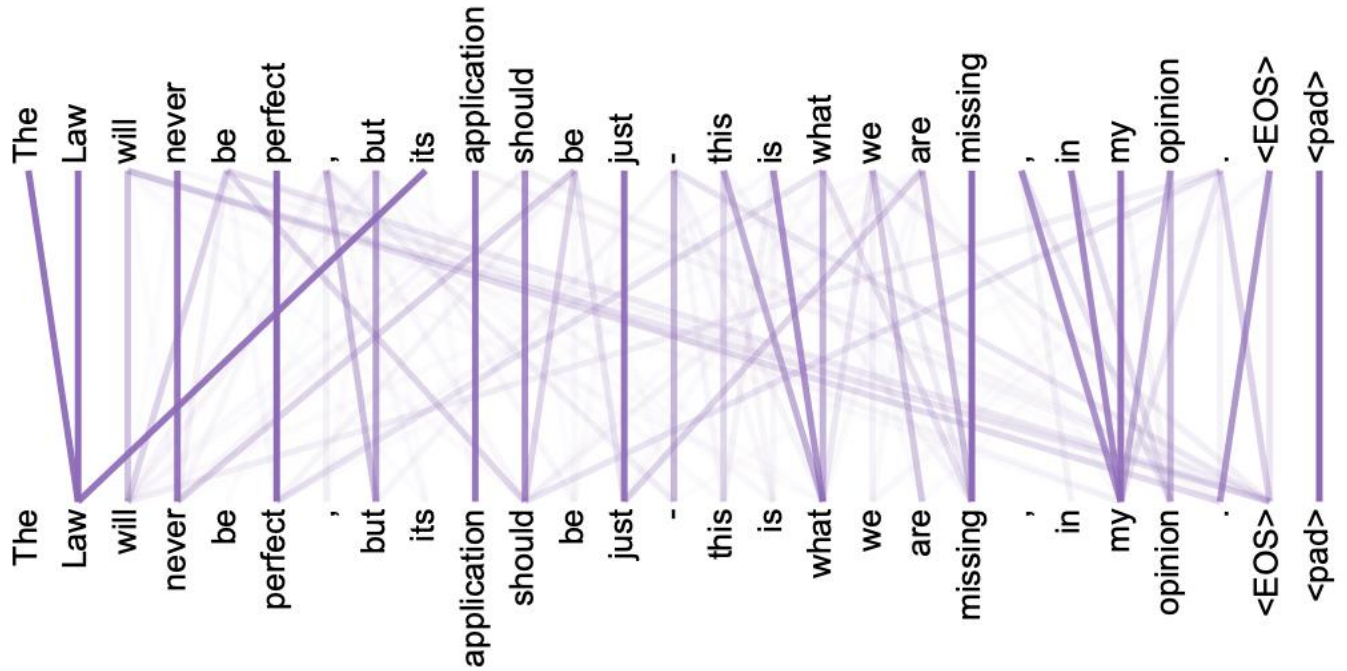
Bottleneck

Last hidden cell is a bottleneck



Solution: Attention!

Core idea: on each step of the decoder, use direct connection to the encoder to focus on a particular part of the source sequence

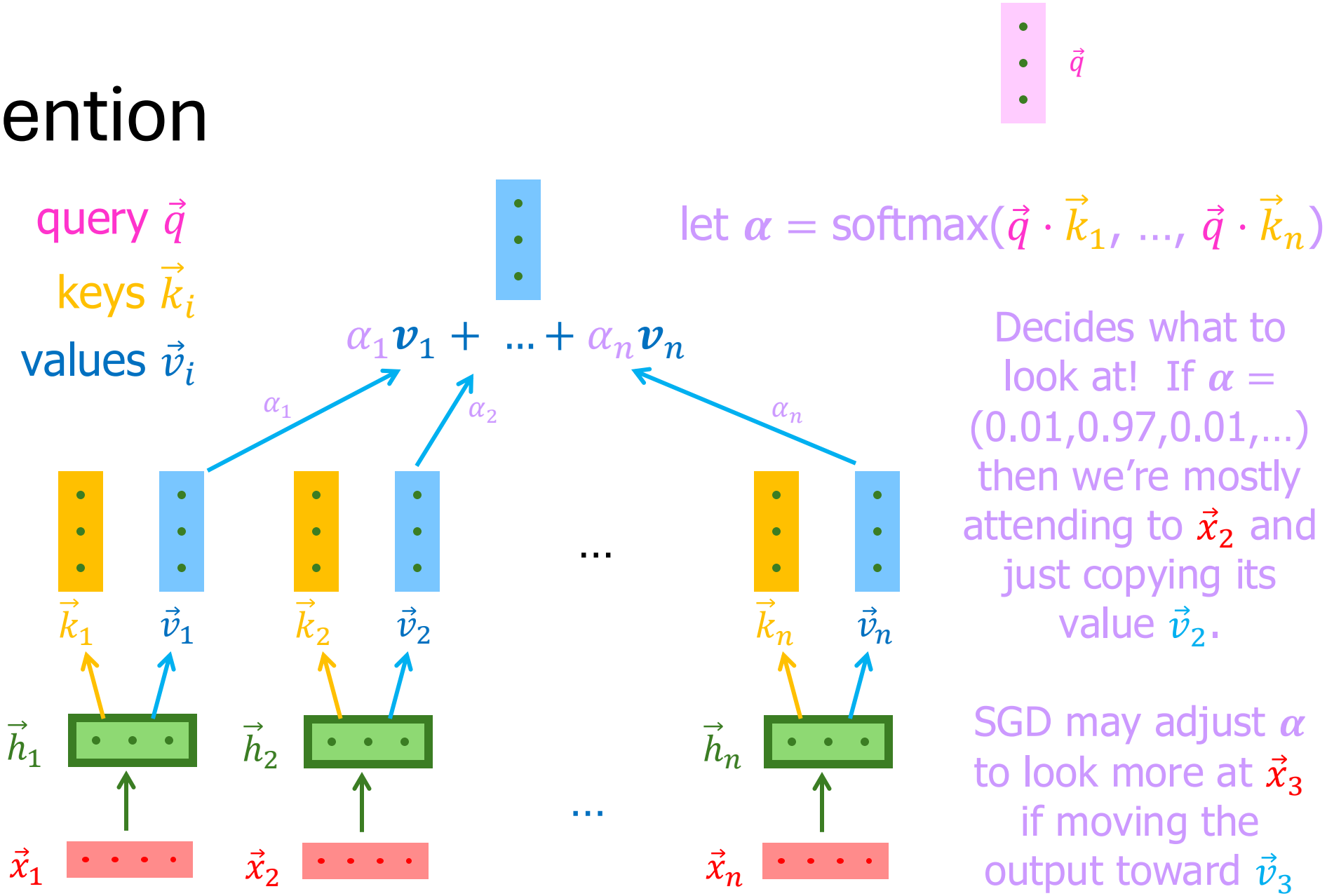


Attention is all you need Vaswani et al 2017

Have we been cramming?

- RNNs etc. – decide early what to keep
  - Encode context into $\mathbb{R}^d$
  - Hope it supports later decoding needs
    - E.g., any reading comprehension question
- Attention – keep it all, decide later what to look at
  - At each decoding step, get to look back at all of the n encoded context objects, each in $\mathbb{R}^d$
  - Take a weighted average of them, where the weights depend on a query created at decoding time
  - This average "completes the encoding" into $\mathbb{R}^d$

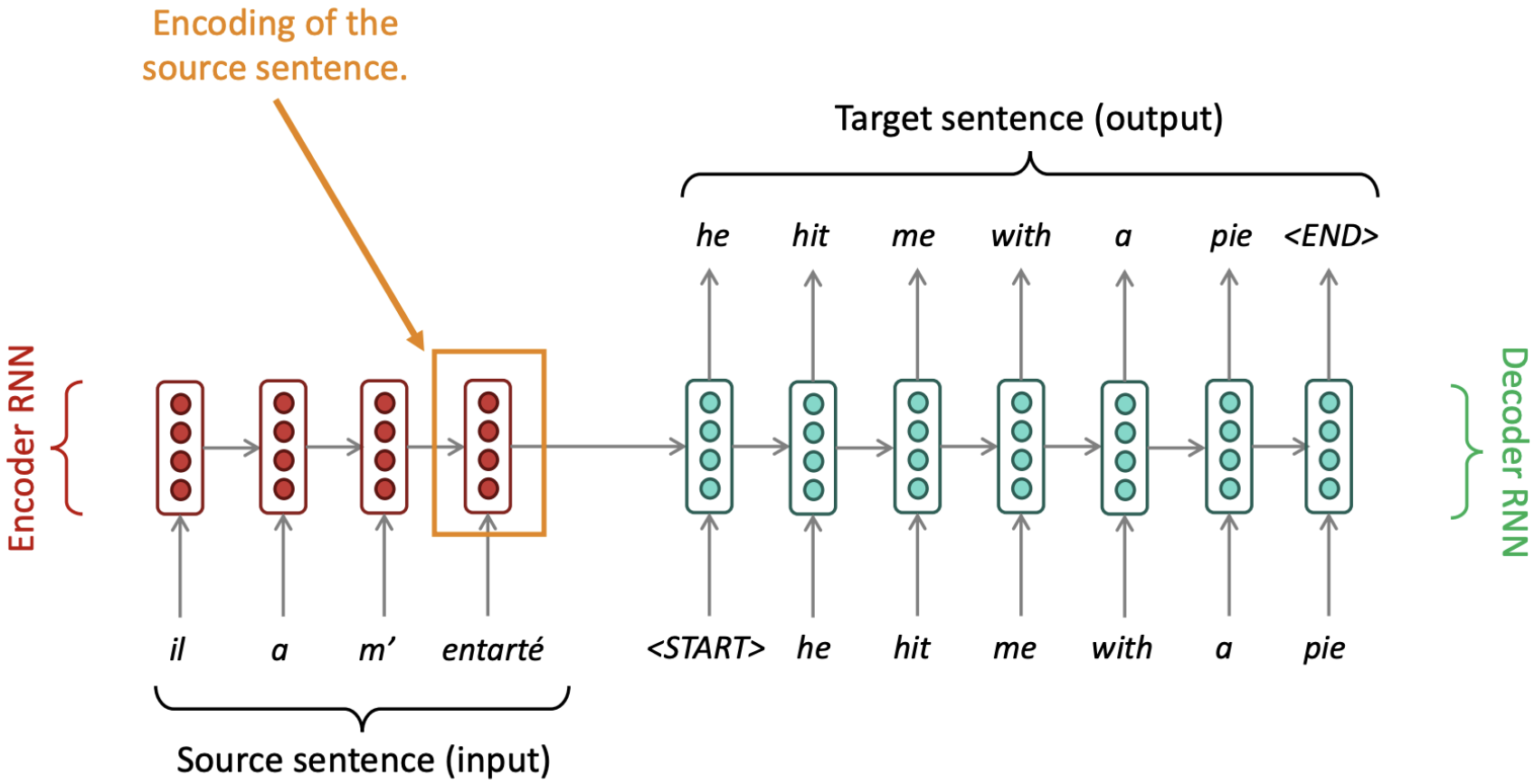
Attention



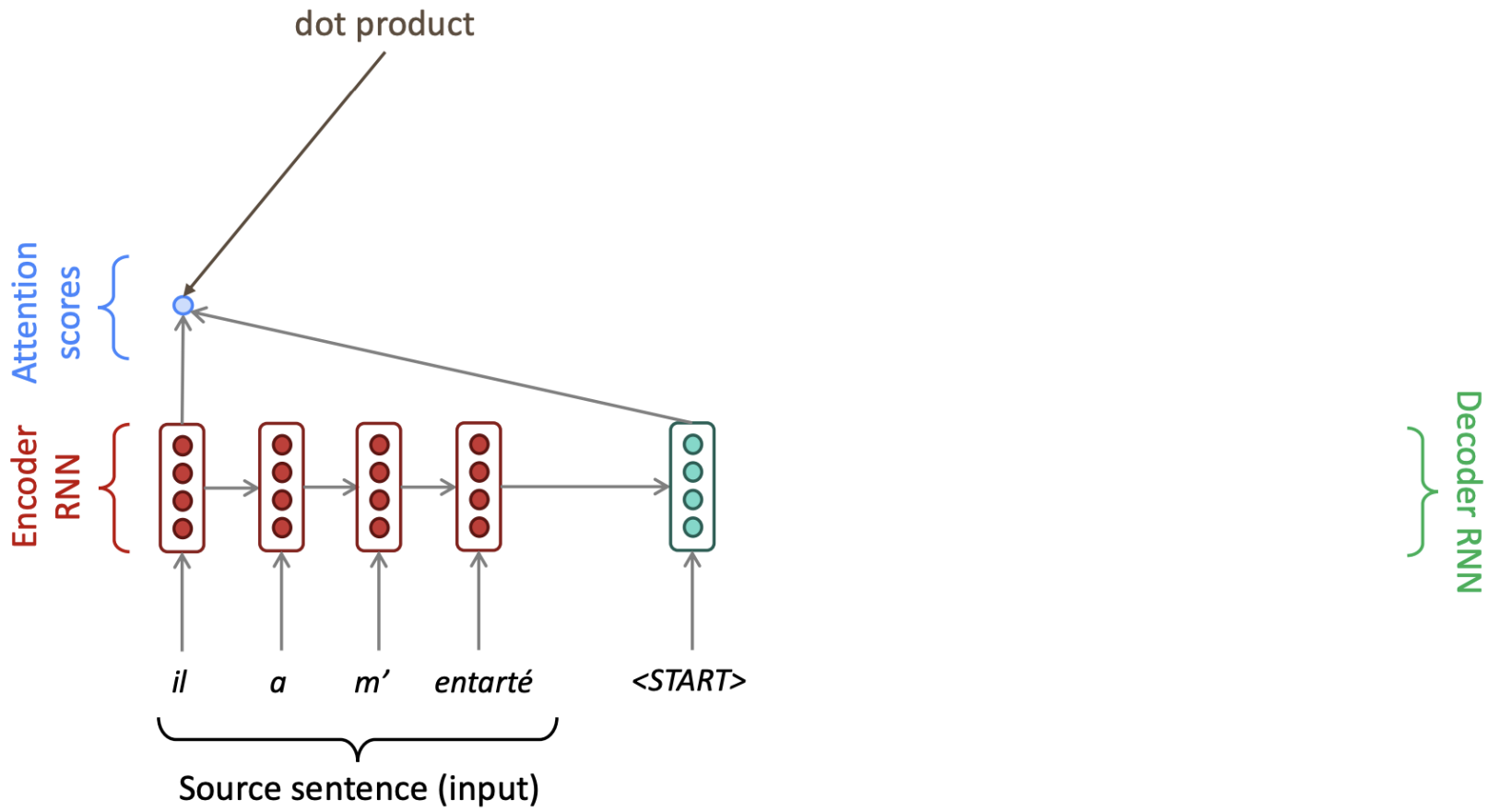
Decides what to look at! If α = (0.01,0.97,0.01,…) then we're mostly attending to $\vec{x}_2$ and just copying its value $\vec{v}_2$.

SGD may adjust α to look more at $\vec{x}_3$ if moving the output toward $\vec{v}_3$ would help loss.

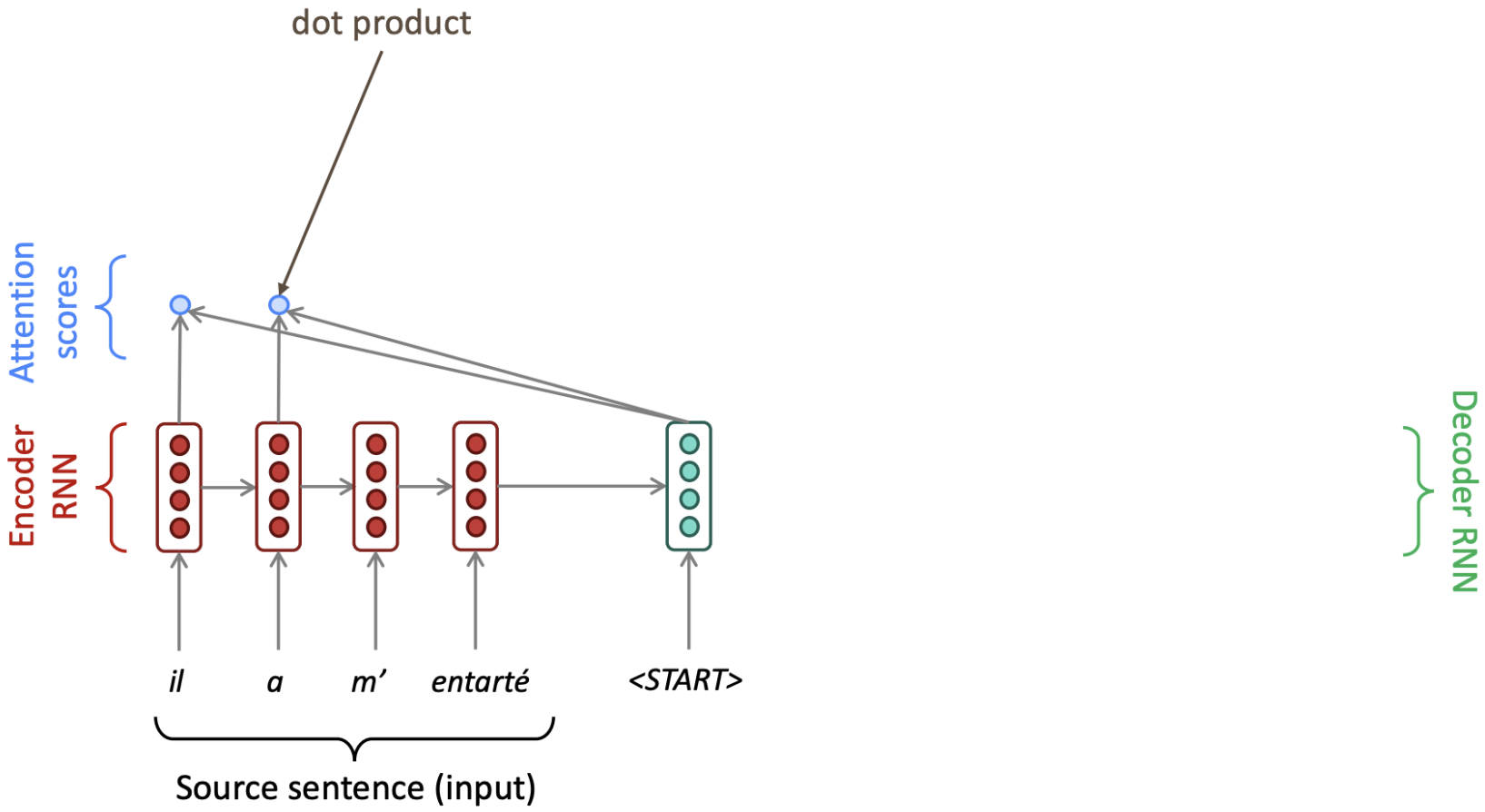
Seq2Seq Model



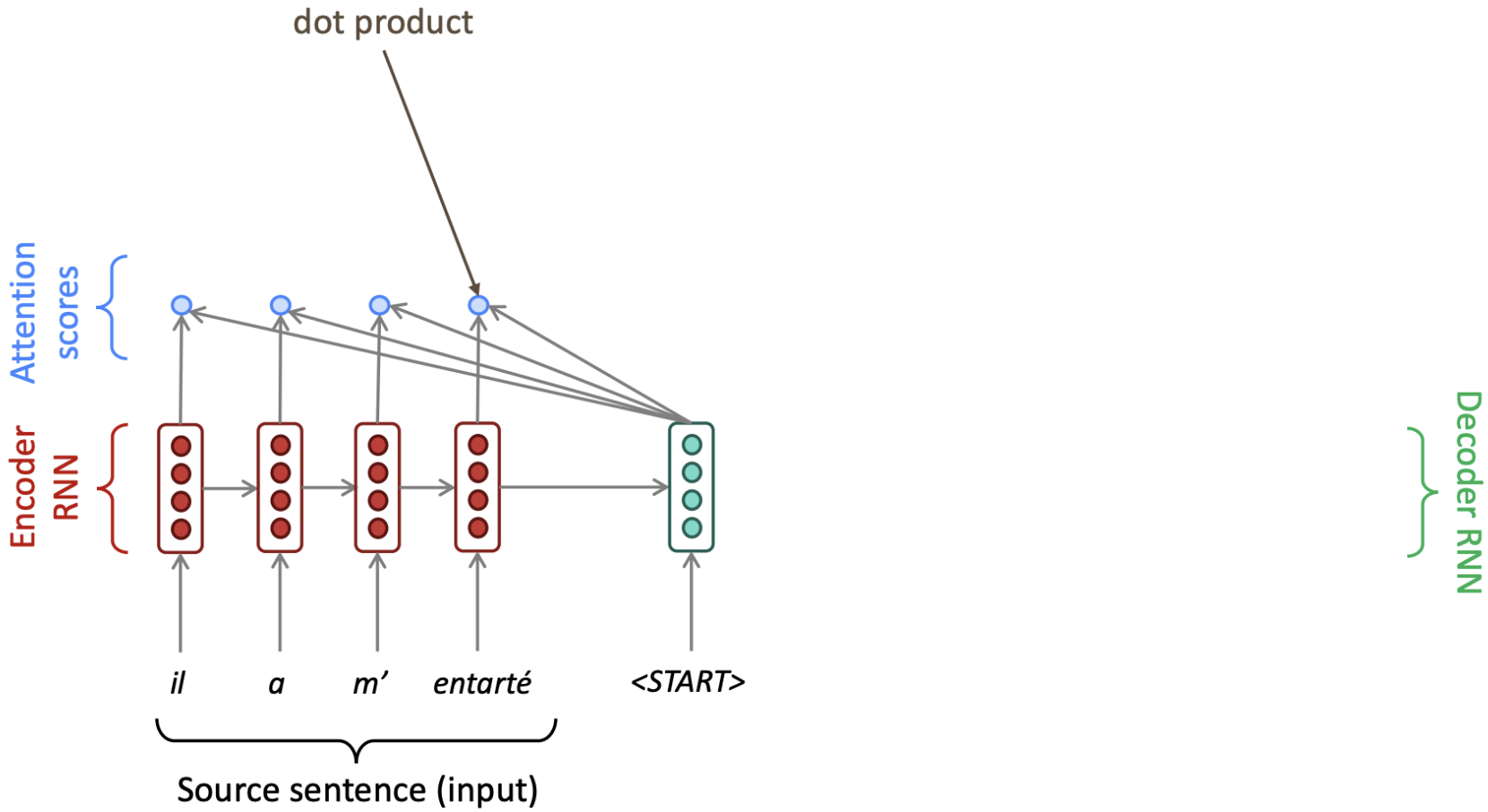
Seq2Seq w/ Attention



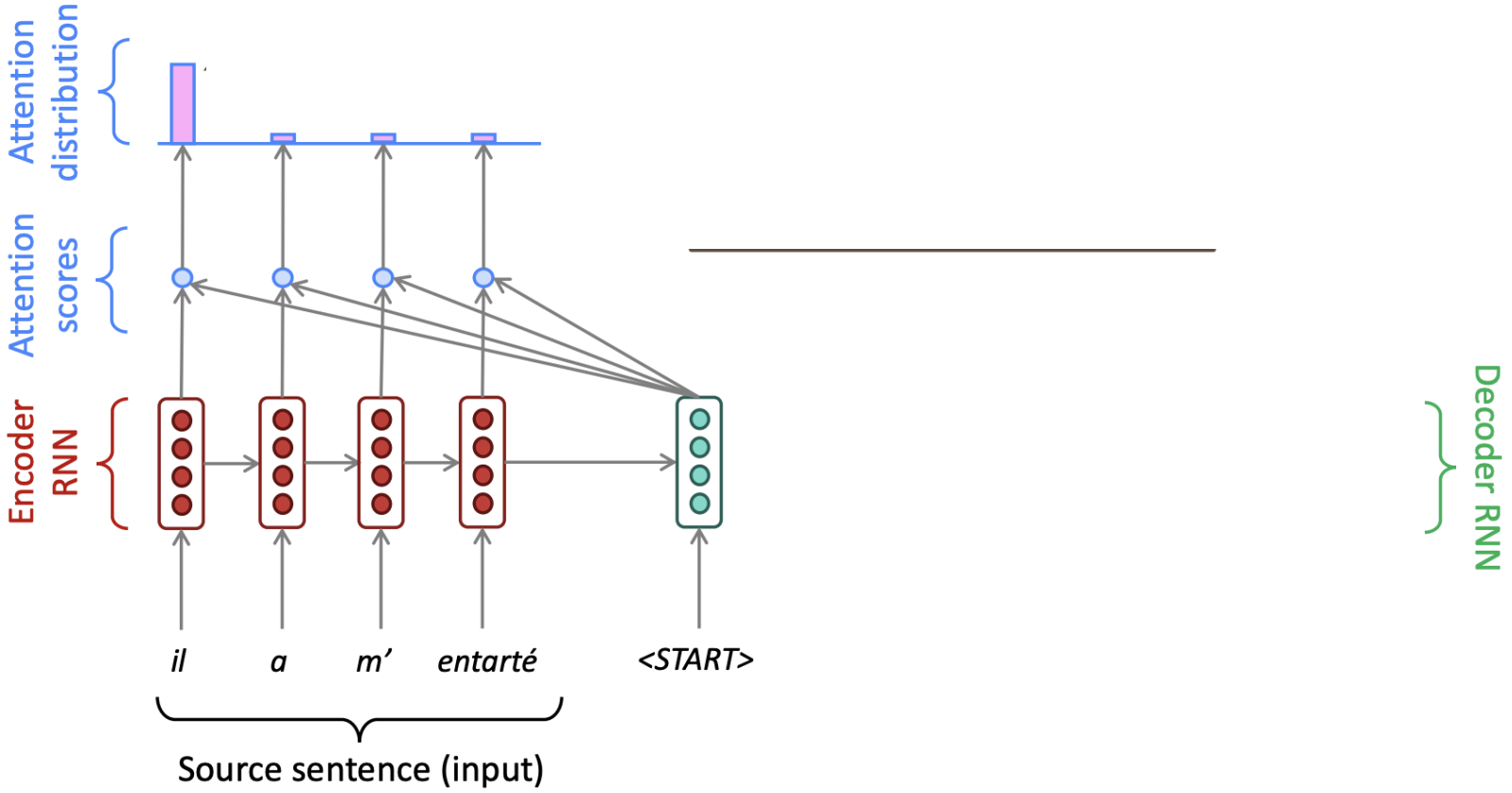
Seq2Seq w/ Attention



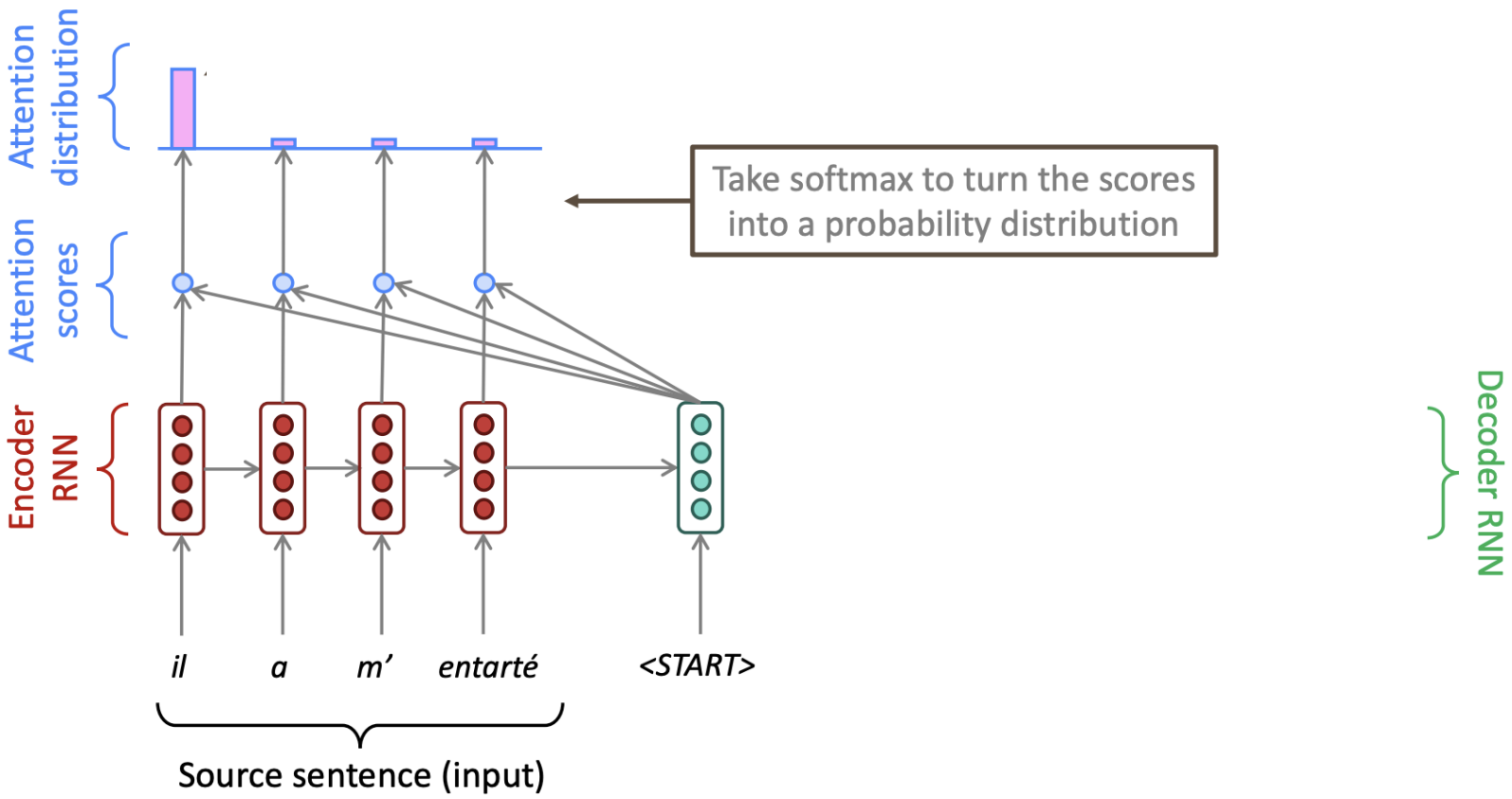
Seq2Seq w/ Attention



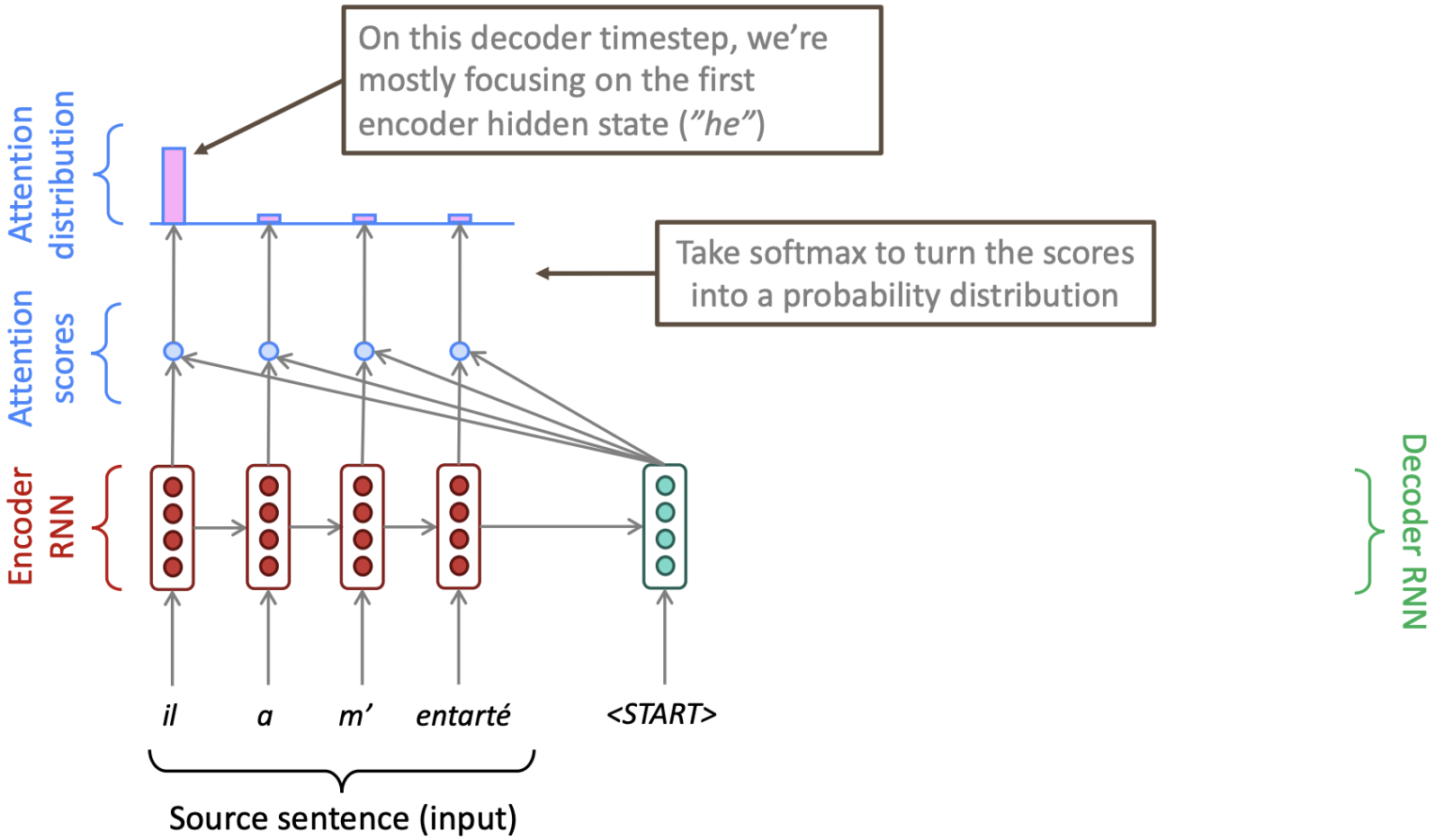
Seq2Seq w/ Attention



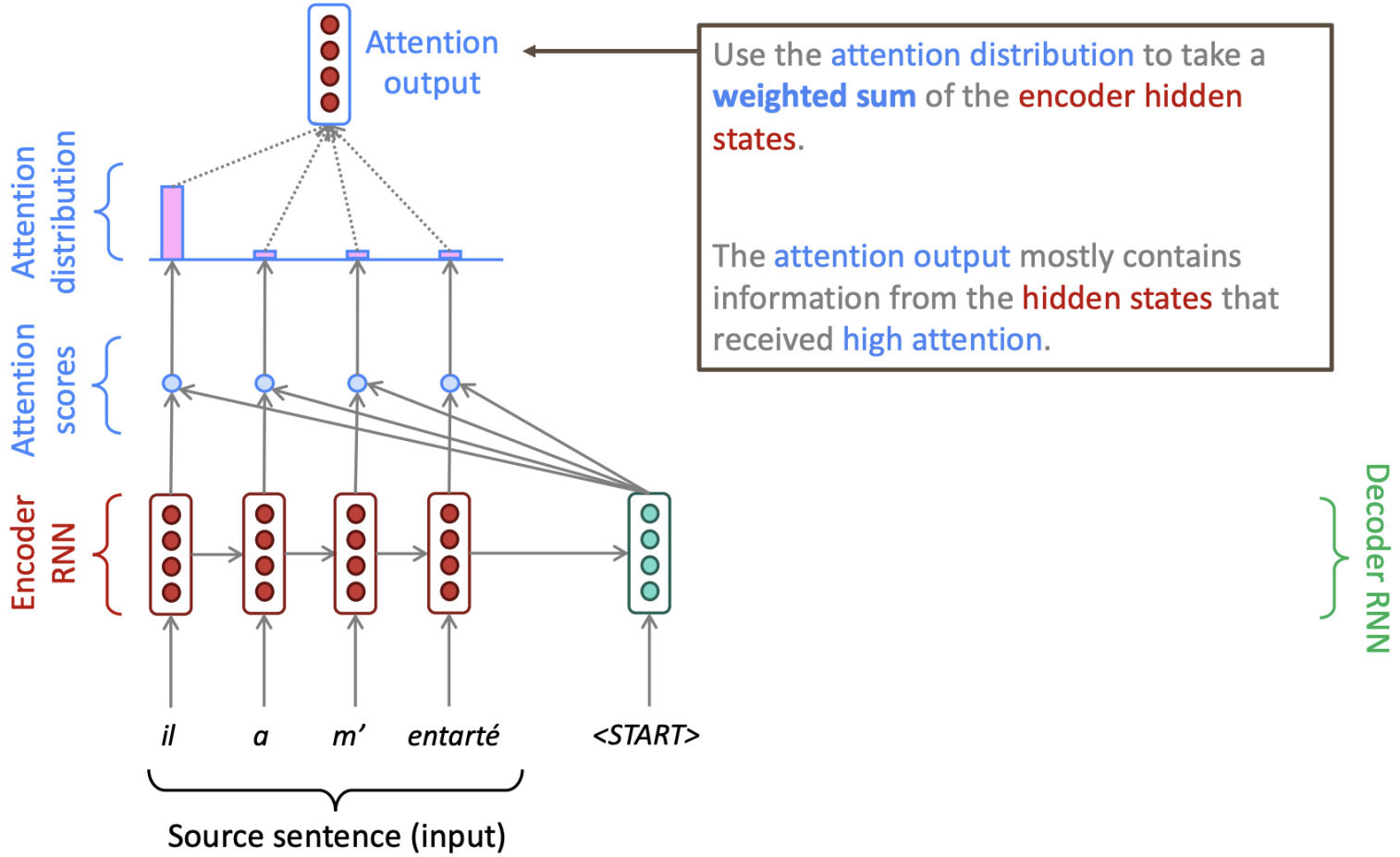
Seq2Seq w/ Attention



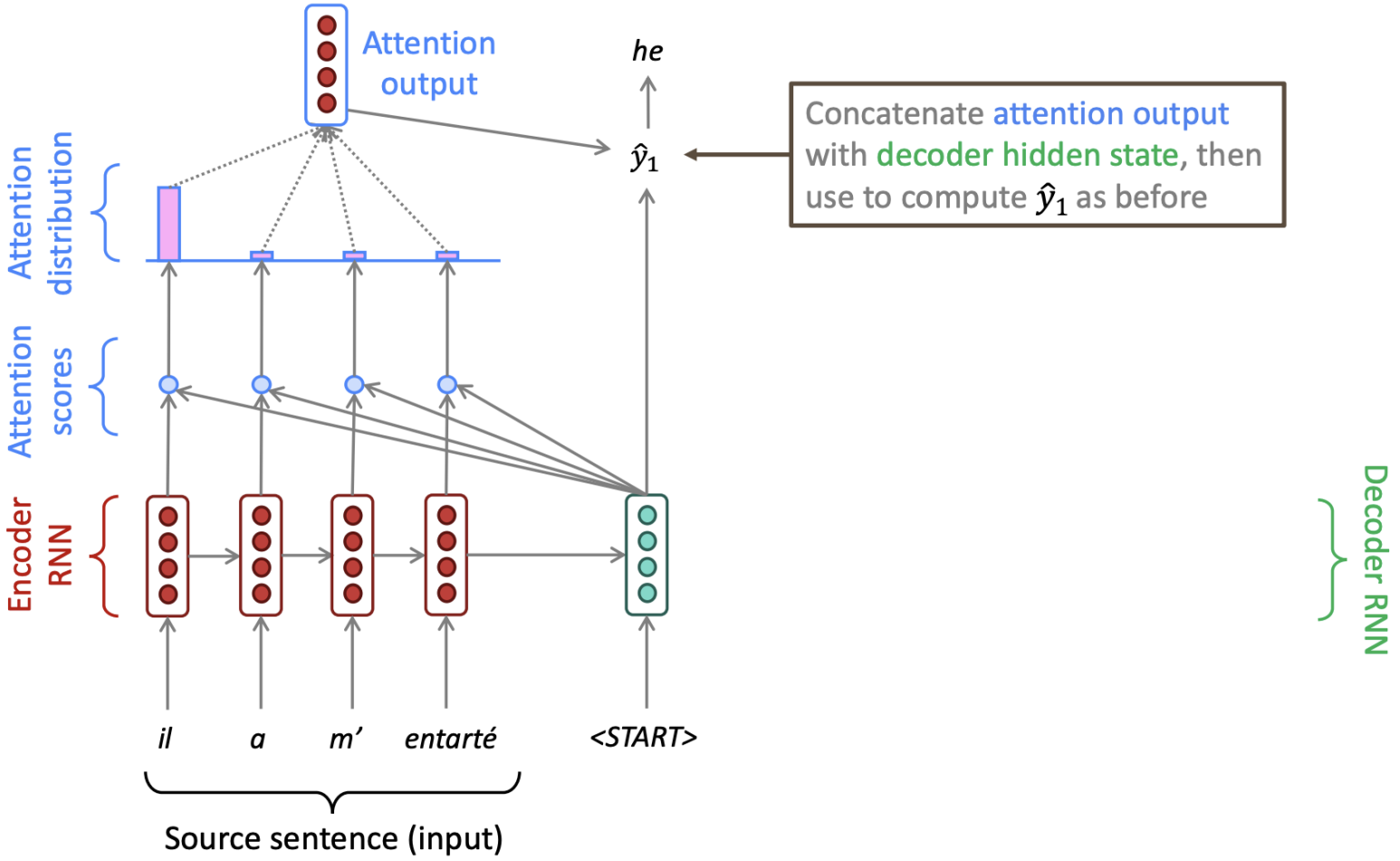
Seq2Seq w/ Attention



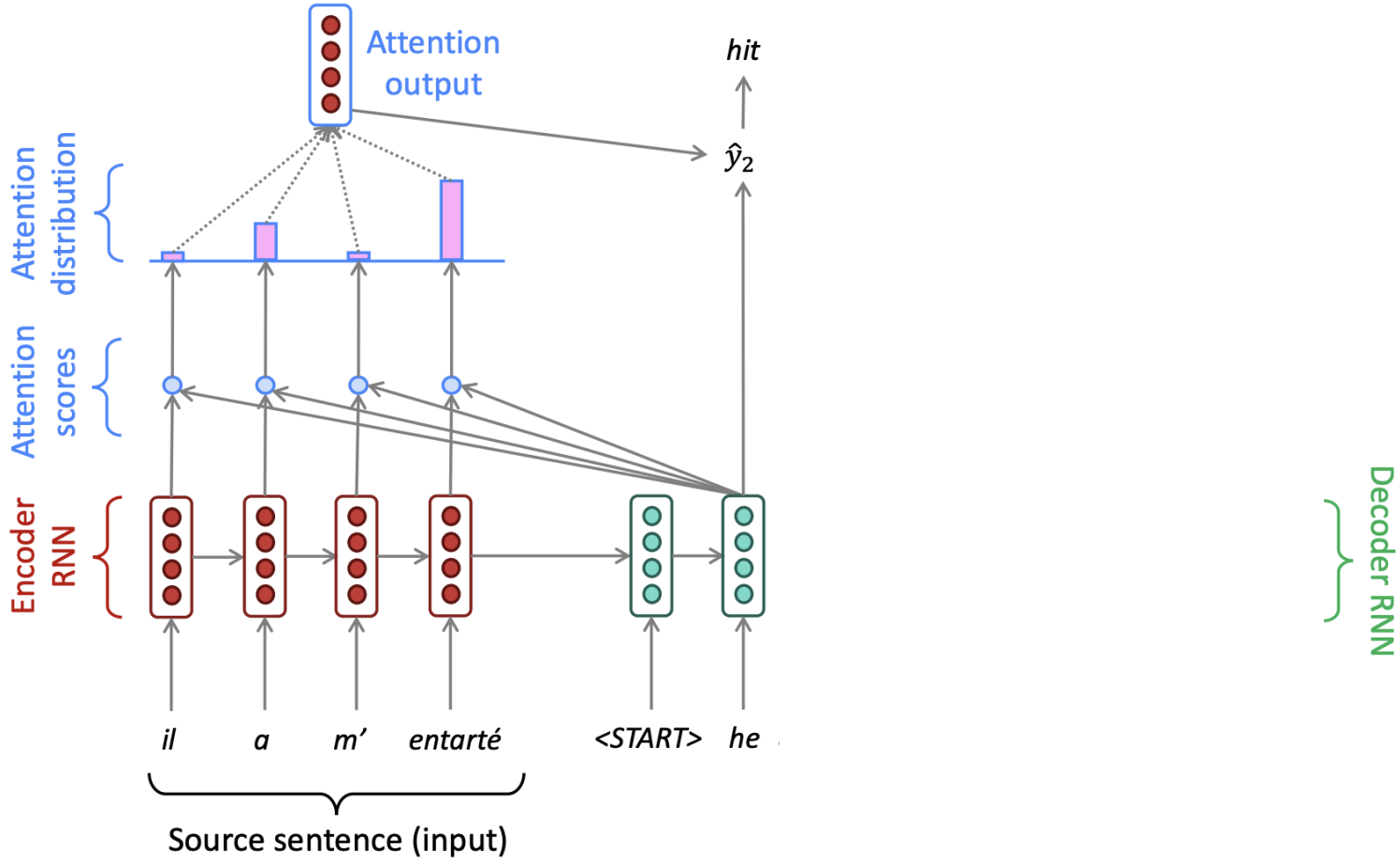
Seq2Seq w/ Attention



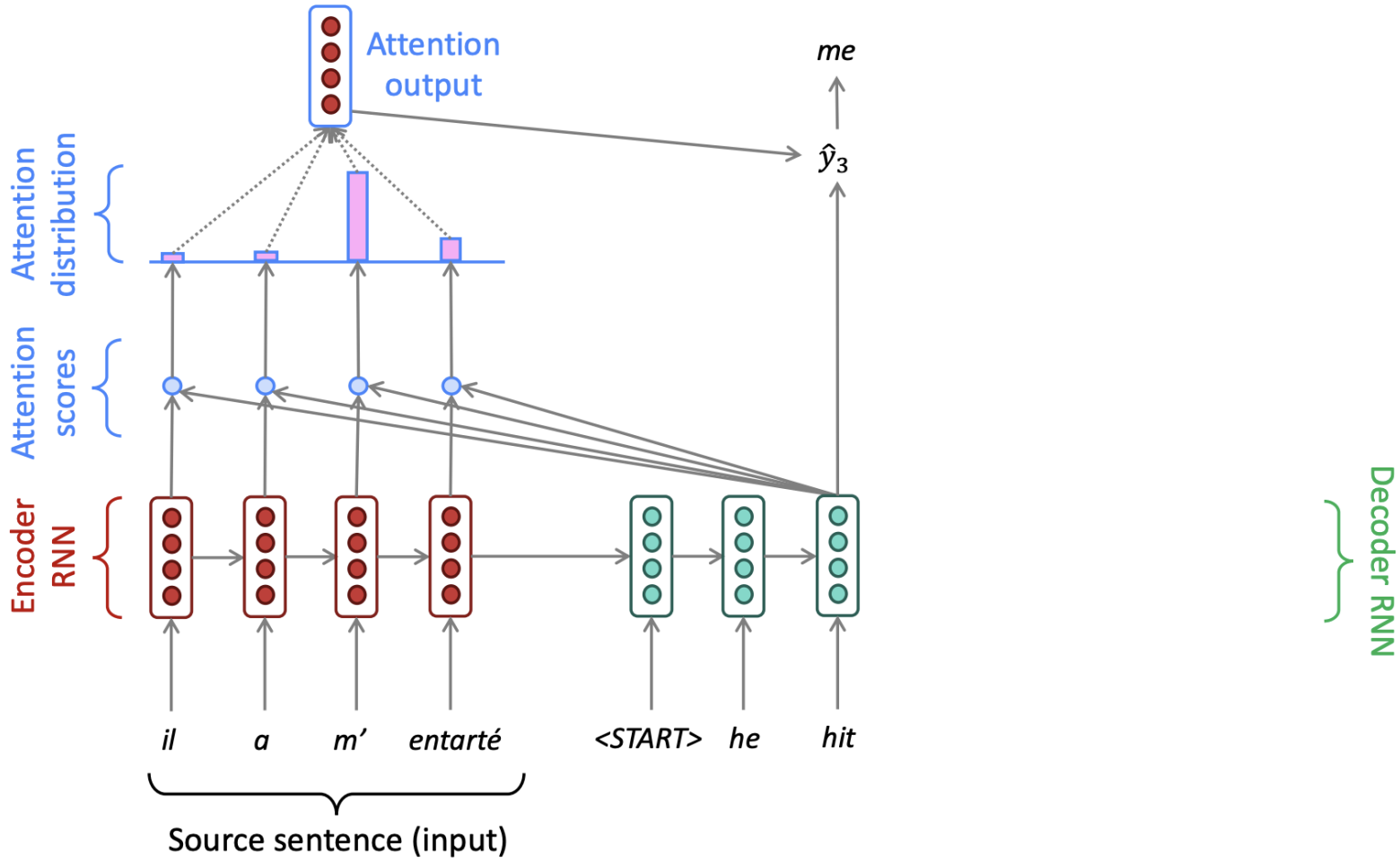
Seq2Seq w/ Attention



Seq2Seq w/ Attention



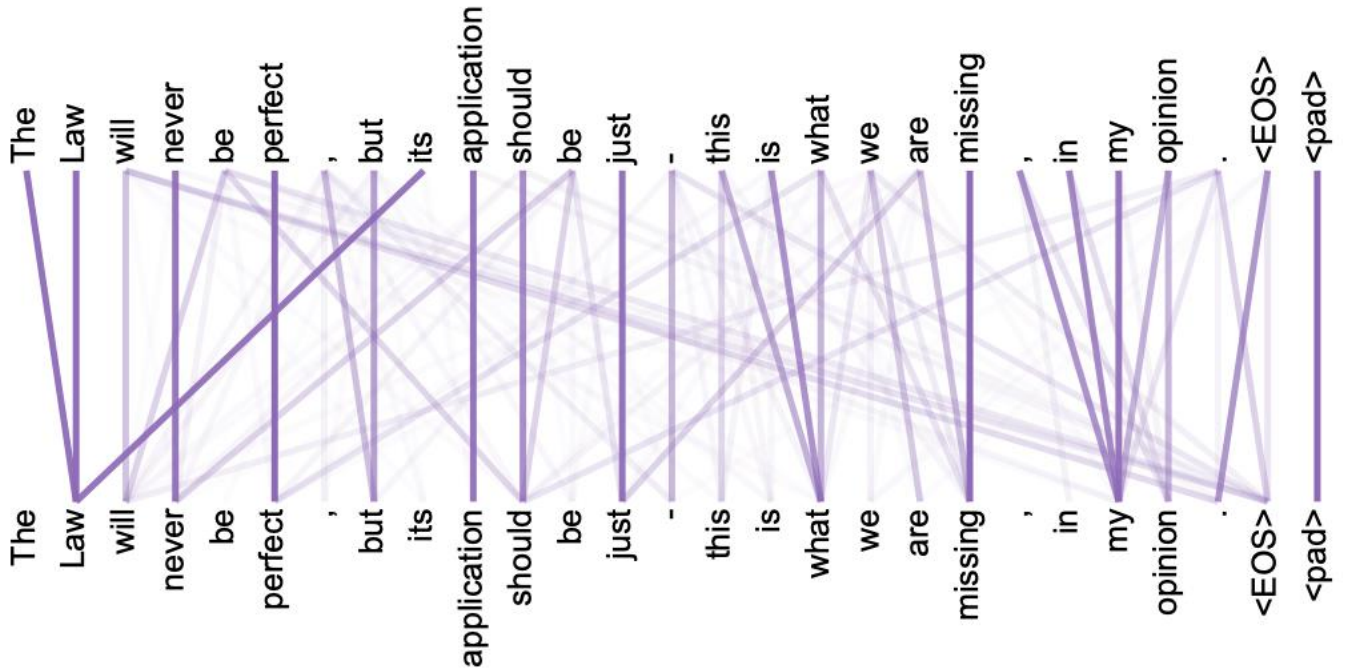
Seq2Seq w/ Attention



Attention pros!

- Significantly improves performance
  - It's very useful to allow decoder to focus on certain parts of the source
- Solves the bottleneck problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Helps with vanishing gradient problem
  - Provides shortcut to faraway states
- Provides some interpretability
  - By inspecting attention distribution, we can see what the decoder was focusing on

Interpretability



Attention pros!

- Significantly improves performance
  - It's very useful to allow decoder to focus on certain parts of the source
- Solves the bottleneck problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Hwith vanishing gradient problem
  - Provides shortcut to faraway states
- Provides some interpretability
  - By inspecting attention distribution, we can see what the decoder was focusing on
- Can be applied to any neural model, not just decoder

Attention in a nutshell

For a new item, figure out how relevant each items is in a collection of different items

W/o attention: we are just relying on a naïve summary of the collection

Encoder-decoder setting:

- How relevant are all the words from the input to a single word in the output

Encoder-MLP setting:

- How relevant are all the words from the input to our prediction

Outline

Word Embedding & FNN issue

RNN - review

Attention

Self Attention

Self-attention in a nutshell

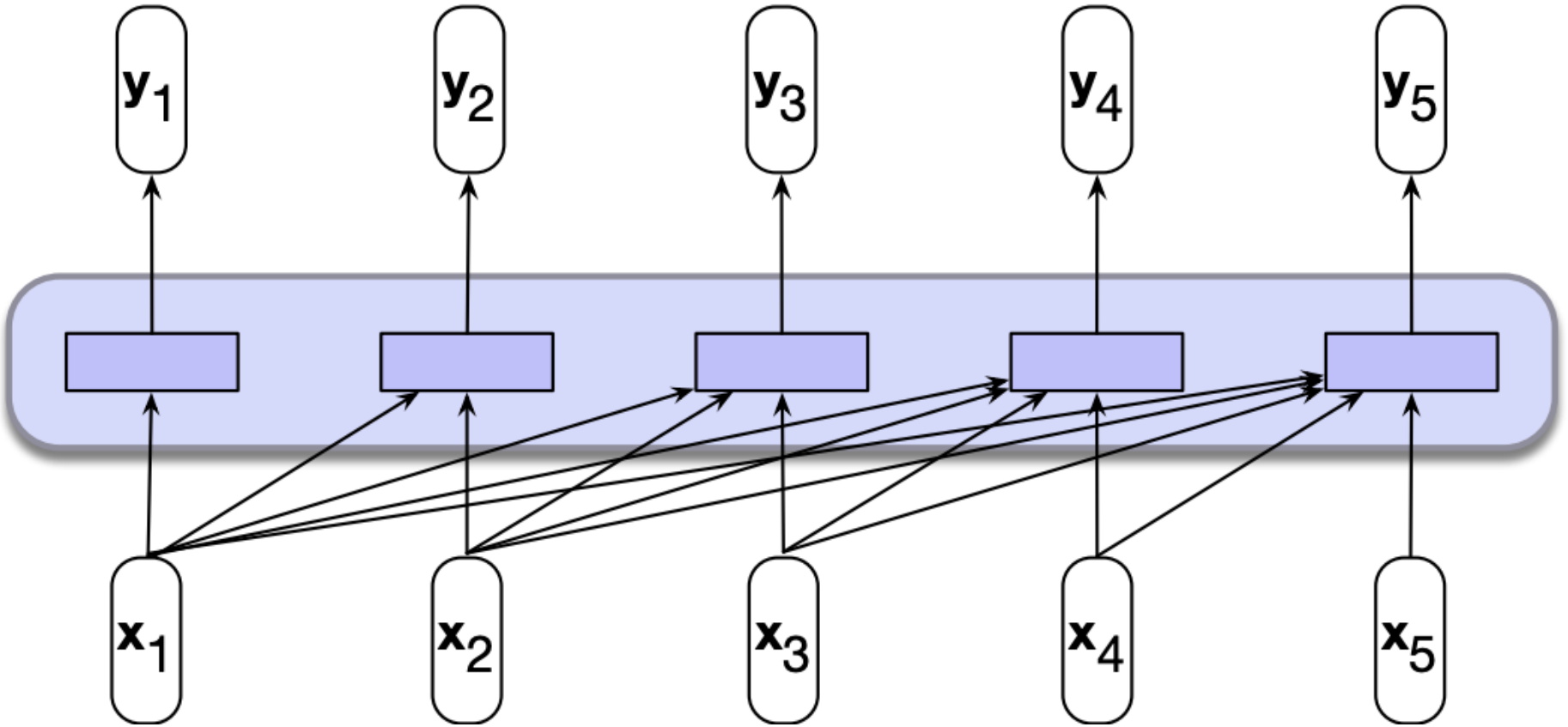
Attention:

- For a new item, figure out how relevant items are in a collection of different items

Self-attention

- How relevant are all the words from the input to a single word in the input

Self-attention



Terminology

Query:

Key:

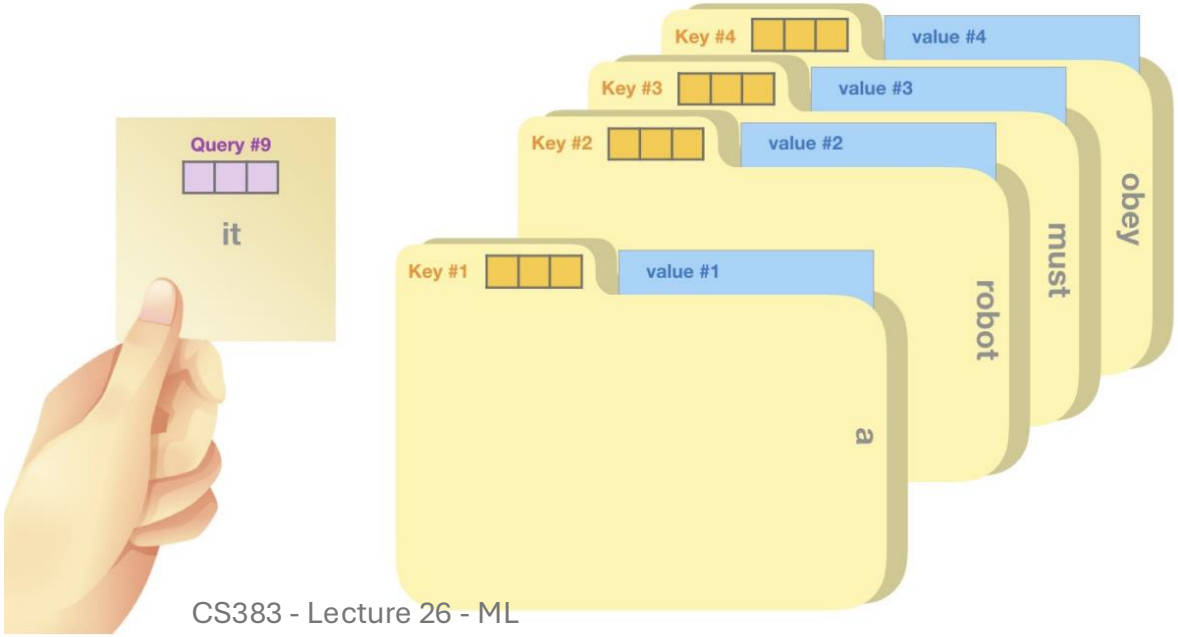
Value:

Terminology

Query: what to match

Key: the thing to match

Value: what to be extracted from the match

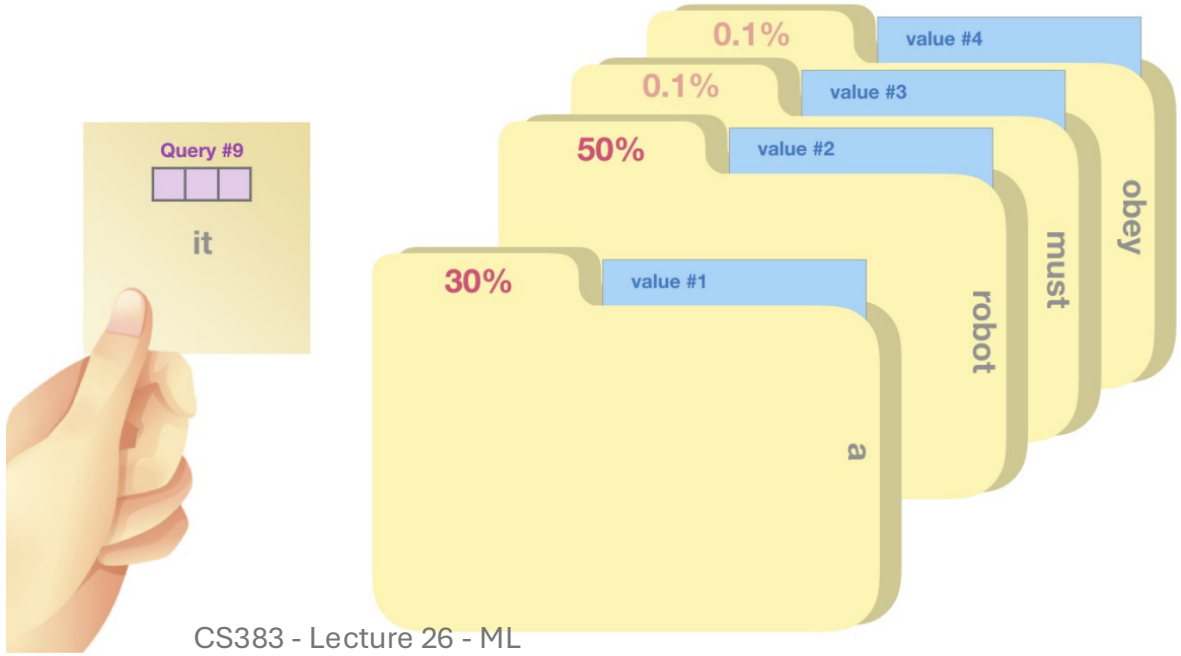


Terminology

Query: what to match

Key: the thing to match

Value: what to be extracted from the match



Terminology

Query: what to match:

$$q_i = W^q x_i$$

Key: the thing to match:

Value: what to be extracted from the match

Terminology

Query: what to match:

$$q_i = W^q x_i$$

Key: the thing to match:

$$k_i = W^k x_i$$

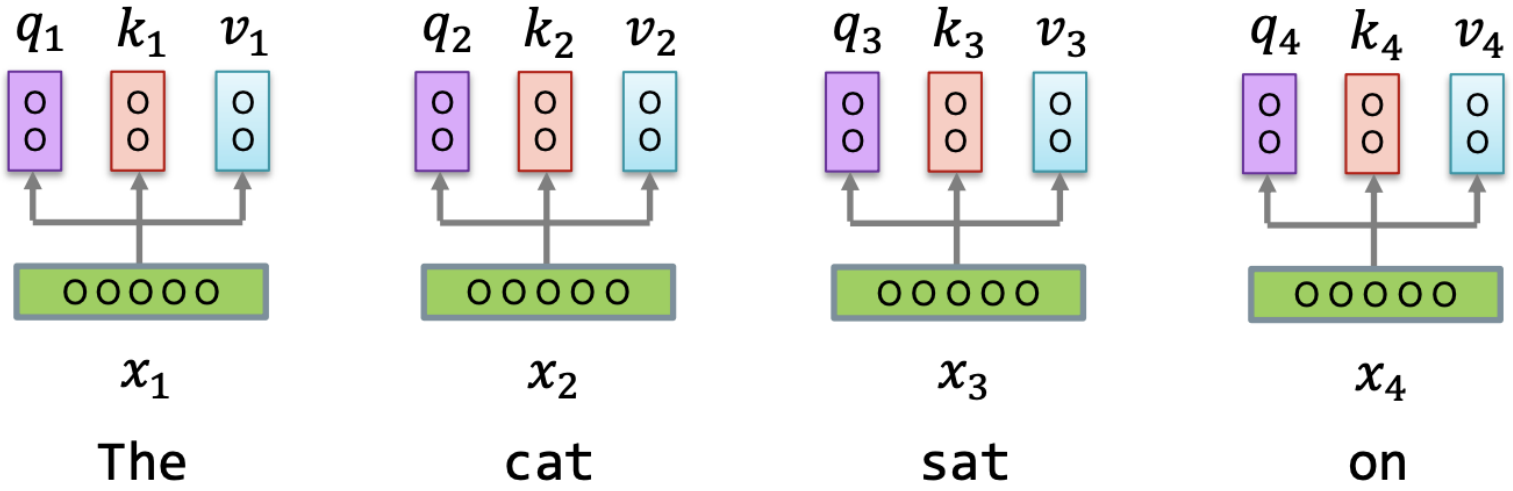
Value: what to be extracted from the match

$$v_i = W^v x_i$$

Output of each input cell

These are three representations of each input

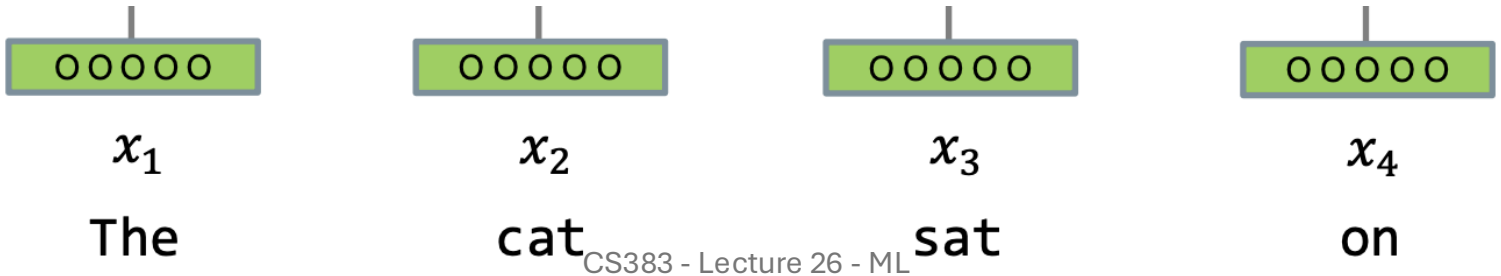
Each representation is created by multiplying the input by a weight matrix



Self-Attention Scores

When creating a representation for x_i, how much weight/focus/attention should we give to x_j

$\forall i, j \in |x|$ we must compute $score(x_i, x_j)$

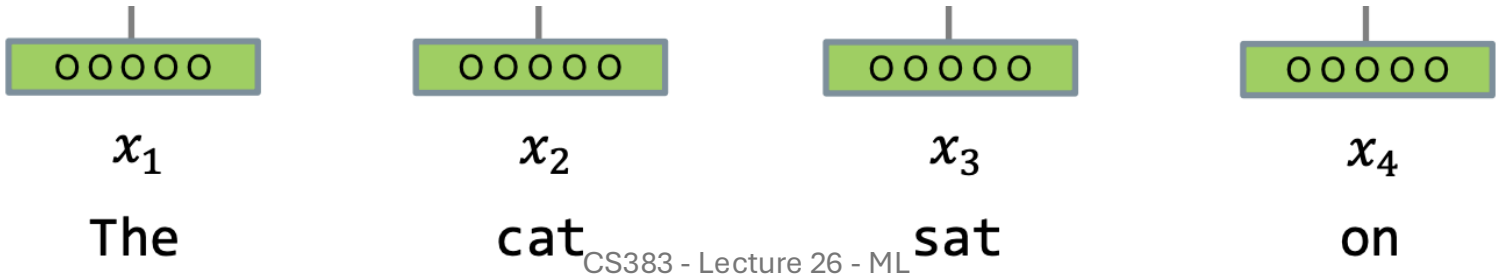


Self-Attention Scores

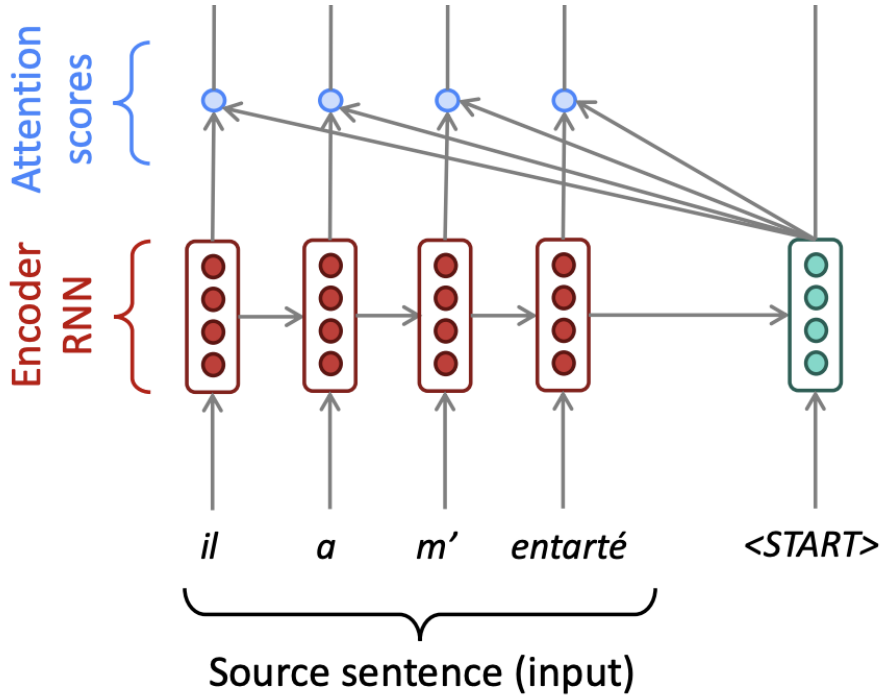
$\forall i, j \in |x|$ we must compute $score(x_i, x_j)$

Question: are these scores distance functions?

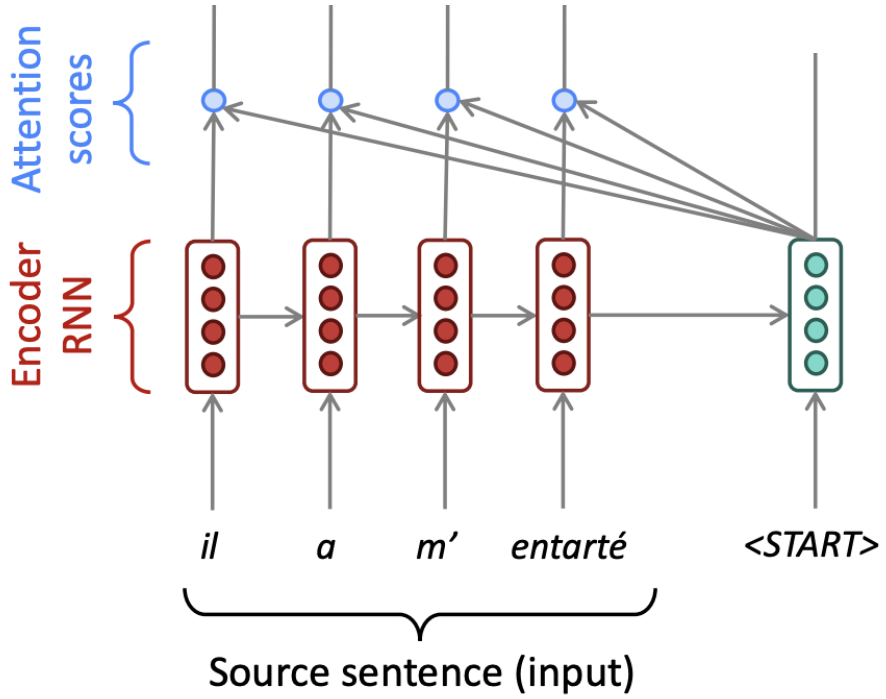
No! $score(x_i, x_j)$ shouldn't be equal to $score(x_i, x_j)$

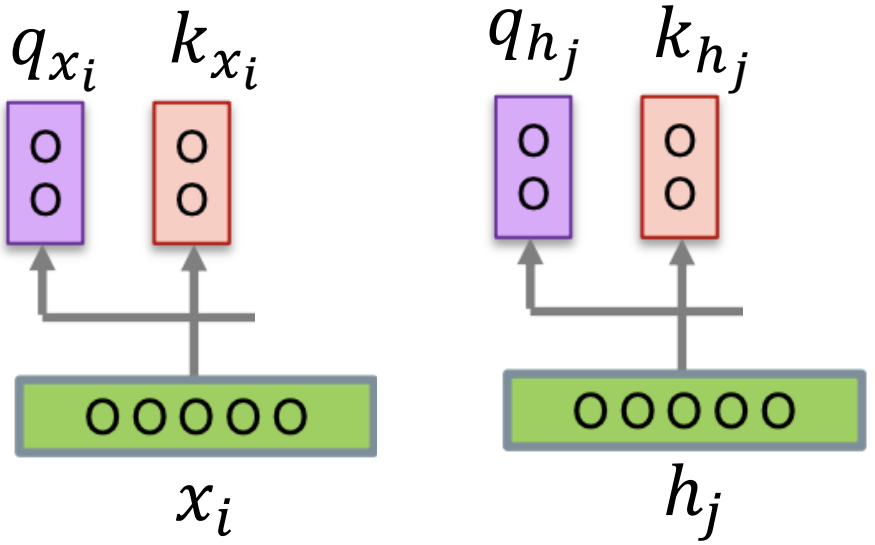


Attention score



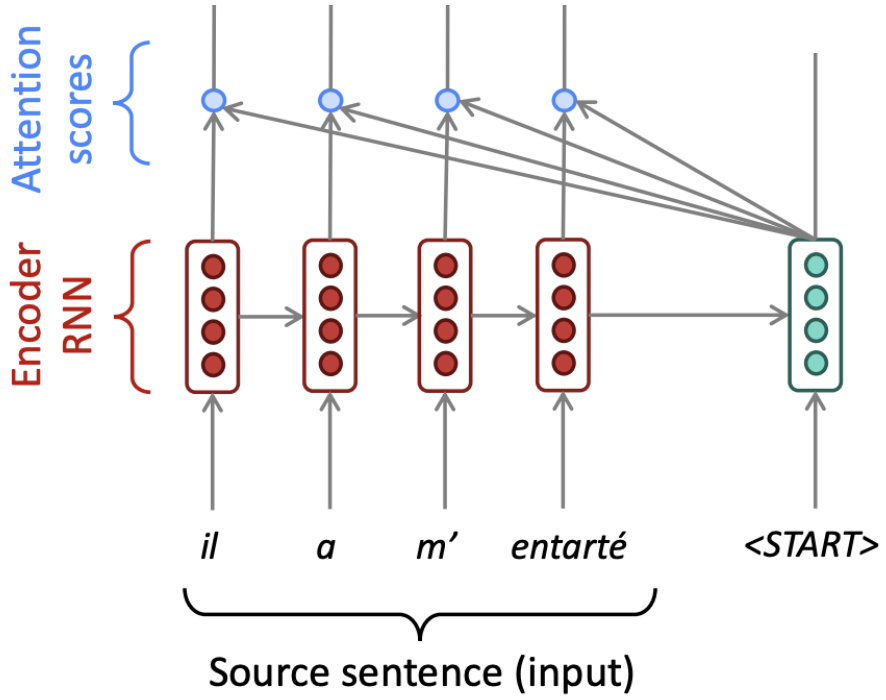
Attention score

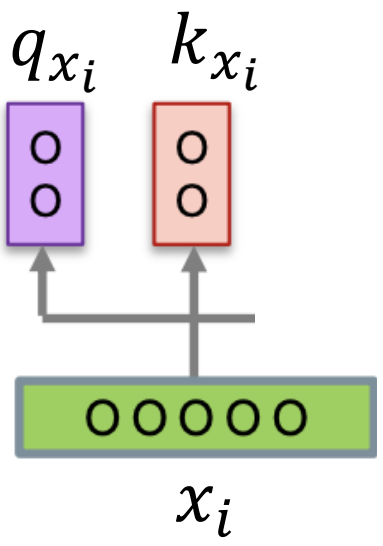


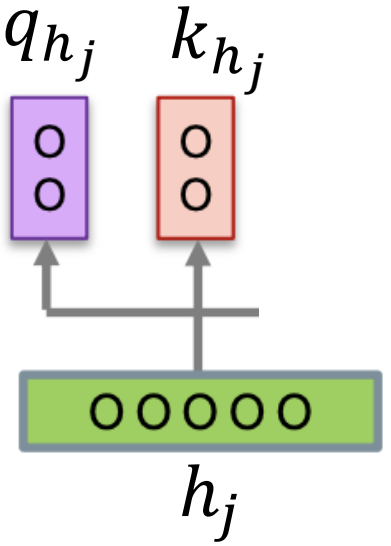


$$score(h_j, x_i) = ?$$

Attention score

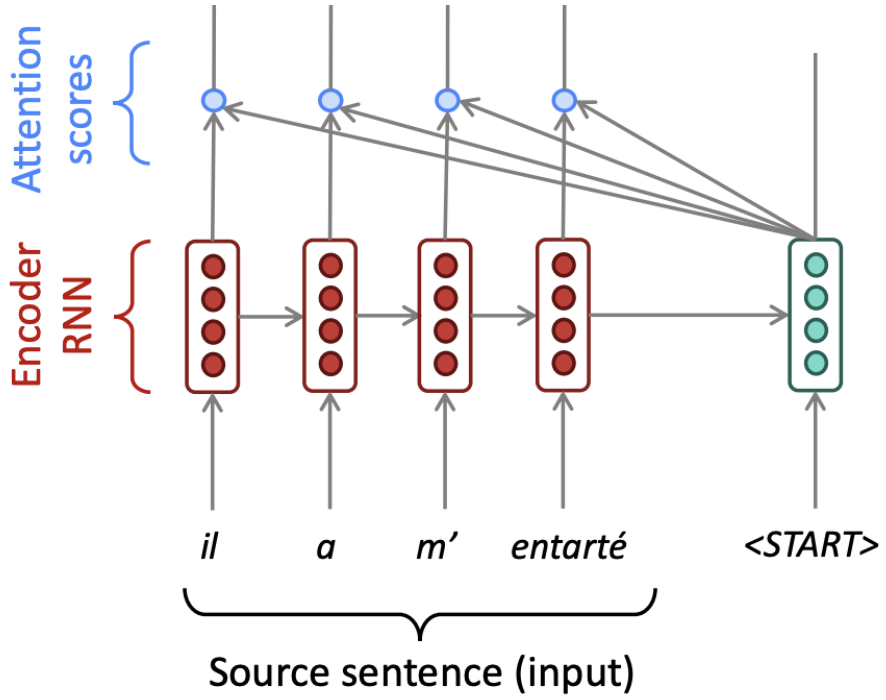


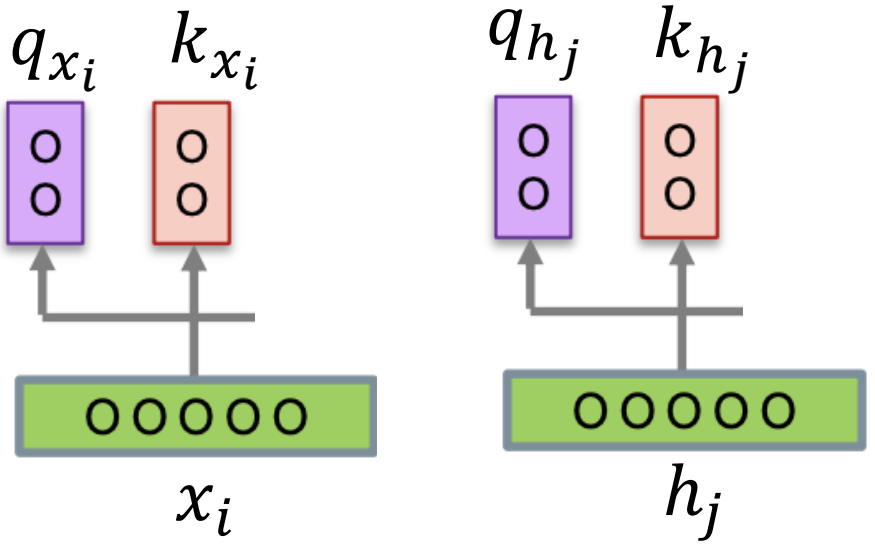




$$score(h_j, x_i) = q_{h_j} \cdot k_{x_i}$$

Attention score





$$score(h_j, x_i) = \frac{q_{h_j} \cdot k_{x_i}}{\sqrt{d_k}}$$

Attention Scores

$$A_{score} = \begin{bmatrix} \alpha_{1,1} & \cdots & \alpha_{1,n} \\ \vdots & \ddots & \vdots \\ \alpha_{m,1} & \cdots & \alpha_{m,n} \end{bmatrix}$$

$$= \begin{bmatrix} score(h_1, x_1) & \cdots & score(h_1, x_n) \\ \vdots & \ddots & \vdots \\ score(h_m, x_1) & \cdots & score(h_m, x_n) \end{bmatrix}$$

$$= \begin{bmatrix} \frac{q_{h_1} \cdot k_{x_1}}{\sqrt{d_k}} & \cdots & \frac{q_{h_1} \cdot k_{x_n}}{\sqrt{d_k}} \\ \vdots & \ddots & \vdots \\ \frac{q_{h_m} \cdot k_{x_1}}{\sqrt{d_k}} & \cdots & \frac{q_{h_m} \cdot k_{x_n}}{\sqrt{d_k}} \end{bmatrix}$$

We can store all the q's and k's in a matrix as well

Attention Scores

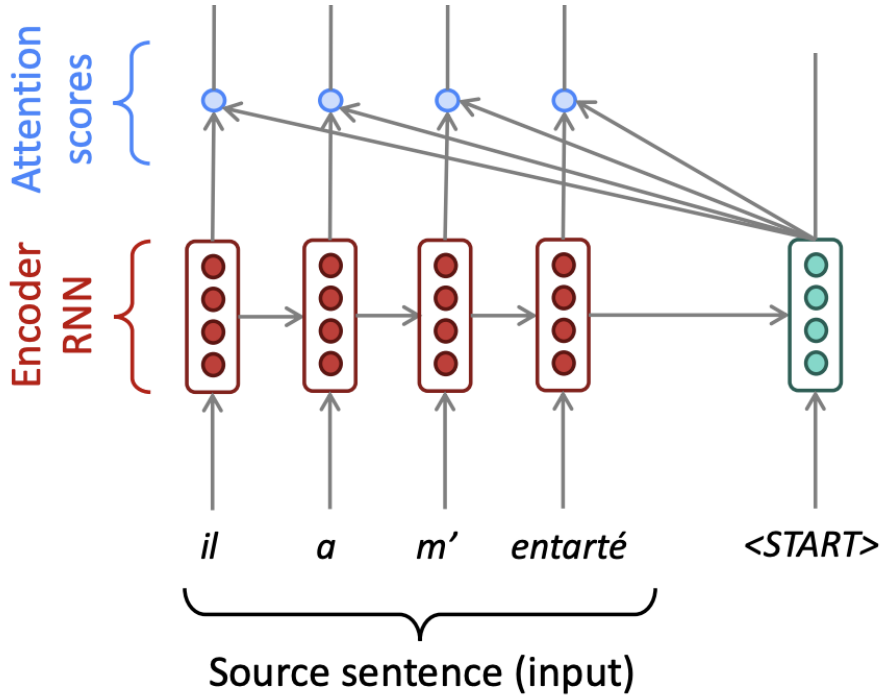
$$A_{score} = \begin{bmatrix} \alpha_{1,1} & \cdots & \alpha_{1,n} \\ \vdots & \ddots & \vdots \\ \alpha_{m,1} & \cdots & \alpha_{m,n} \end{bmatrix}$$

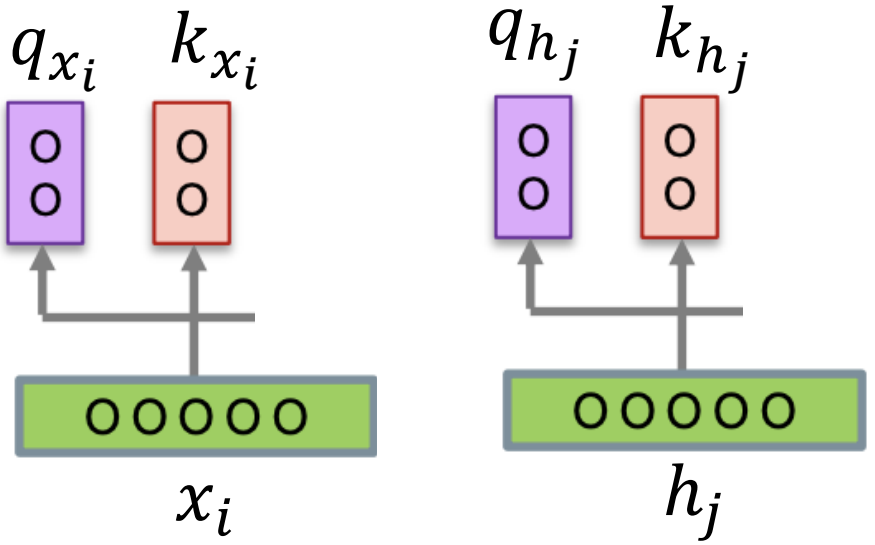
$$= \begin{bmatrix} \frac{q_{h_1} \cdot k_{x_1}}{\sqrt{d_k}} & \cdots & \frac{q_{h_1} \cdot k_{x_n}}{\sqrt{d_k}} \\ \vdots & \ddots & \vdots \\ \frac{q_{h_m} \cdot k_{x_1}}{\sqrt{d_k}} & \cdots & \frac{q_{h_m} \cdot k_{x_n}}{\sqrt{d_k}} \end{bmatrix}$$

$$A_{score} = \frac{QK^T}{\sqrt{d_k}}$$

We can store all the q's and k's in a matrix as well

Attention score

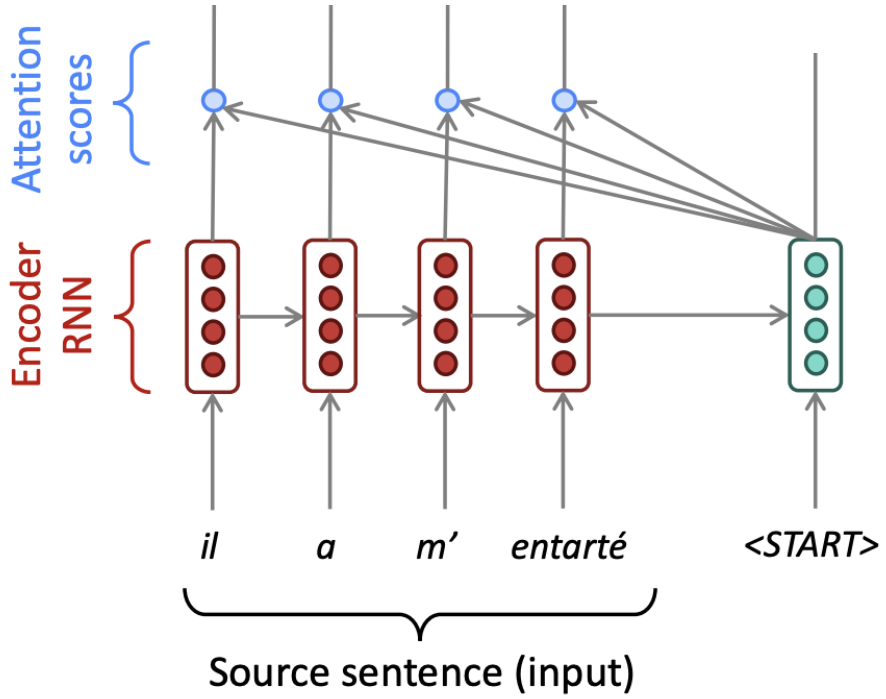


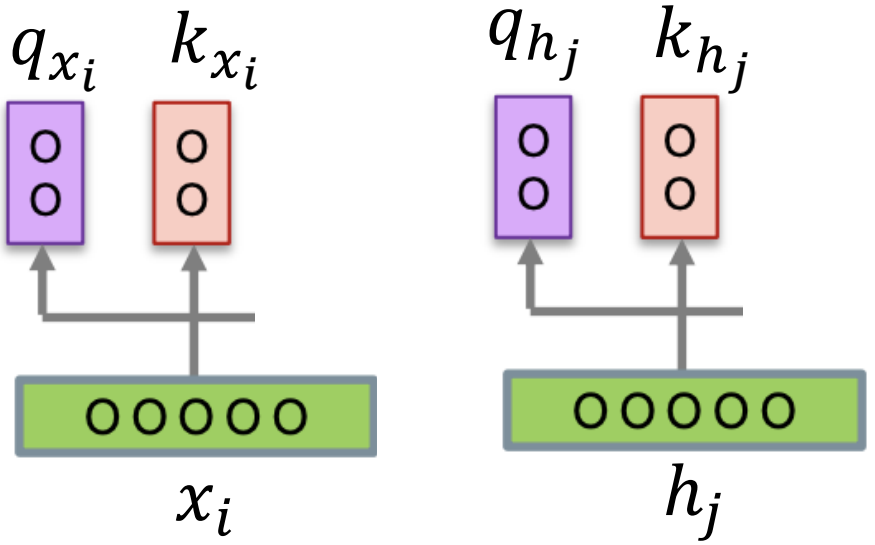


$$score(h_j, x_i) = \frac{q_{h_j} \cdot k_{x_i}}{\sqrt{d_k}}$$

$$A_{score} = \frac{QK^T}{\sqrt{d_k}}$$

Attention

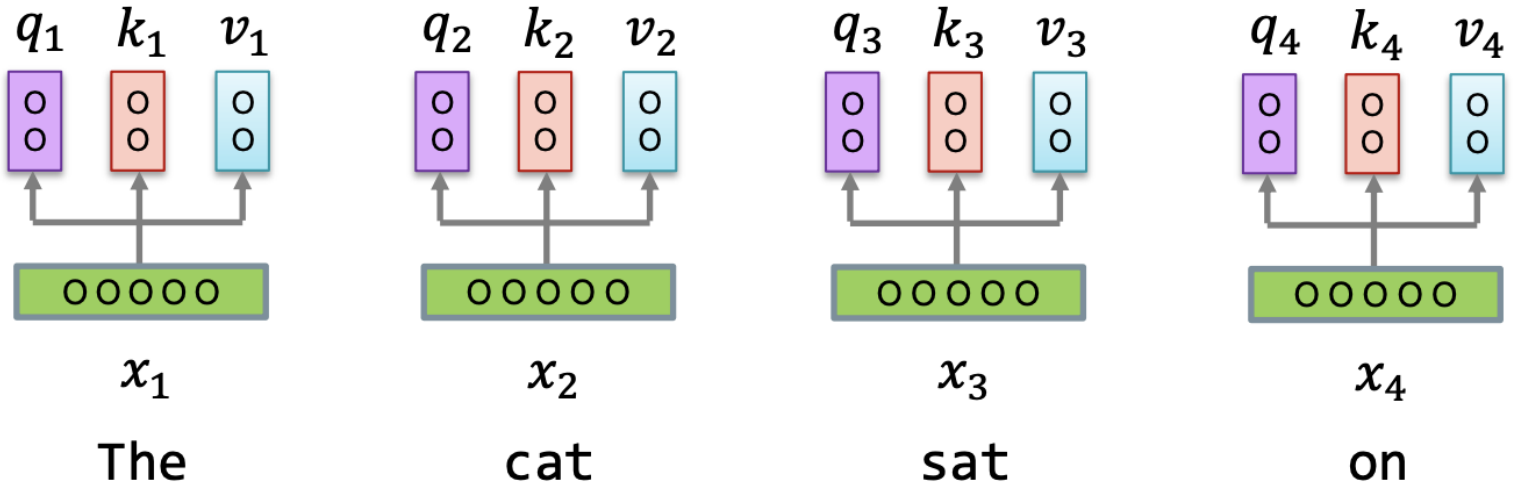




$$A = softmax(\frac{QK^T}{\sqrt{d_k}})$$

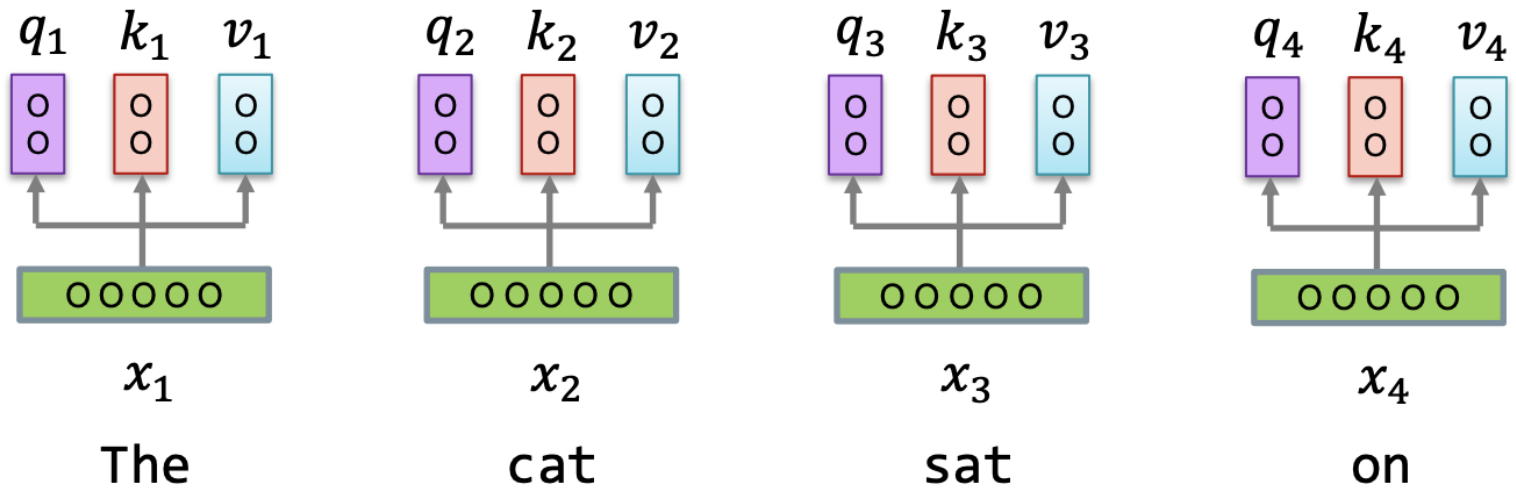
Self-attention

When creating a representation for x_i, how much weight/focus/attention should we give to x_j

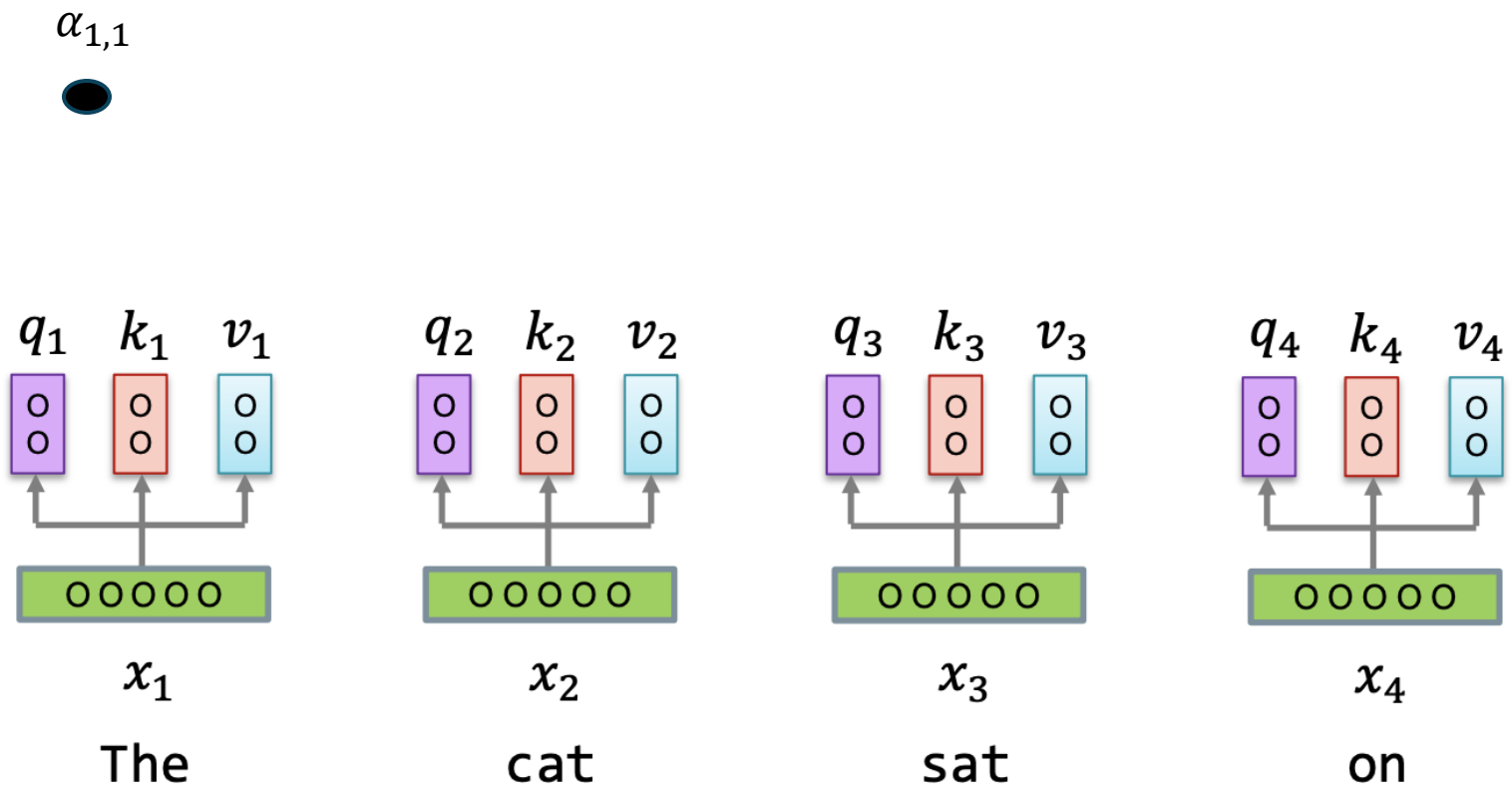


Self-attention

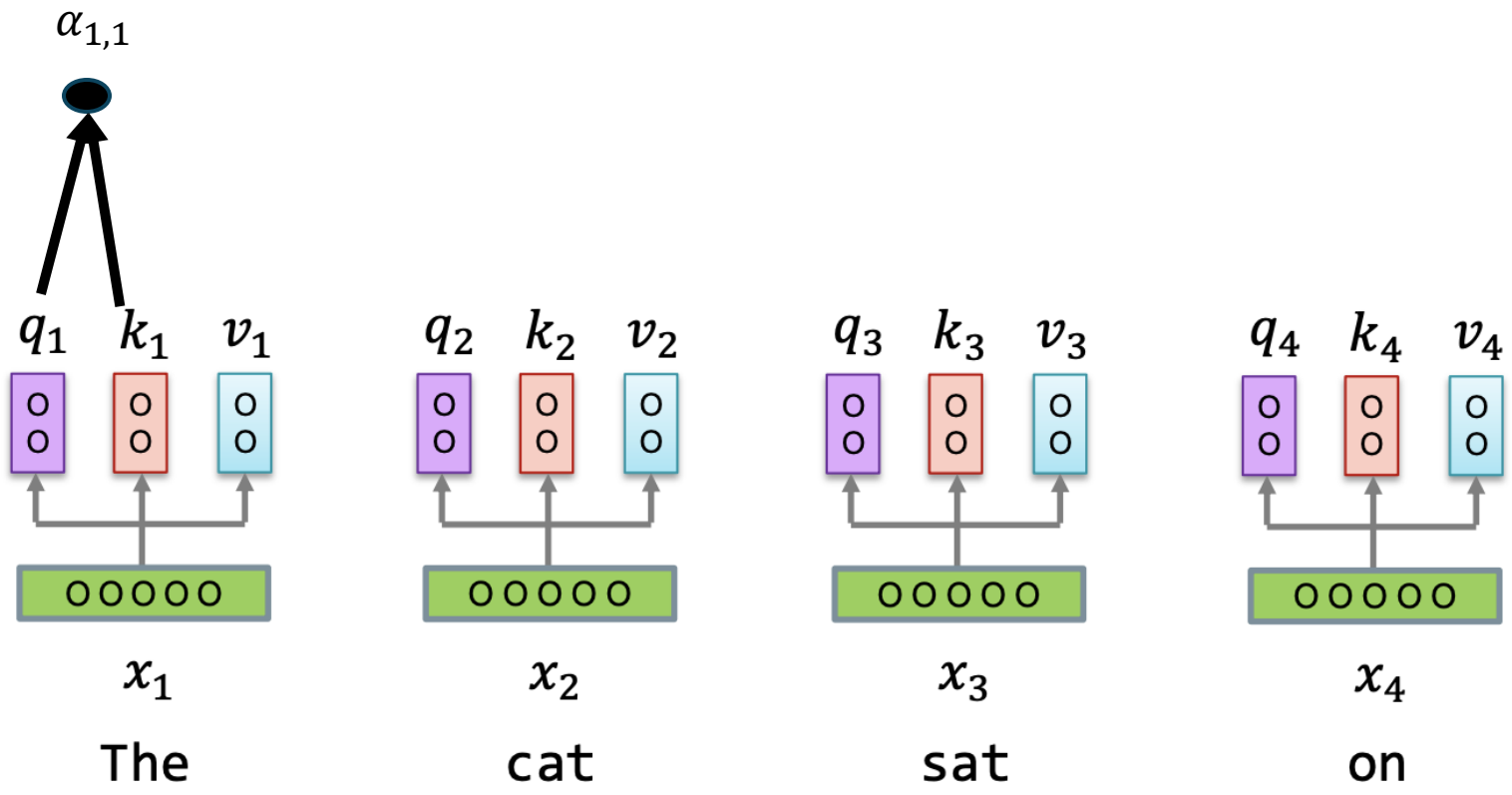
When creating a representation for x_1, how much weight/focus/attention should we give to x_j



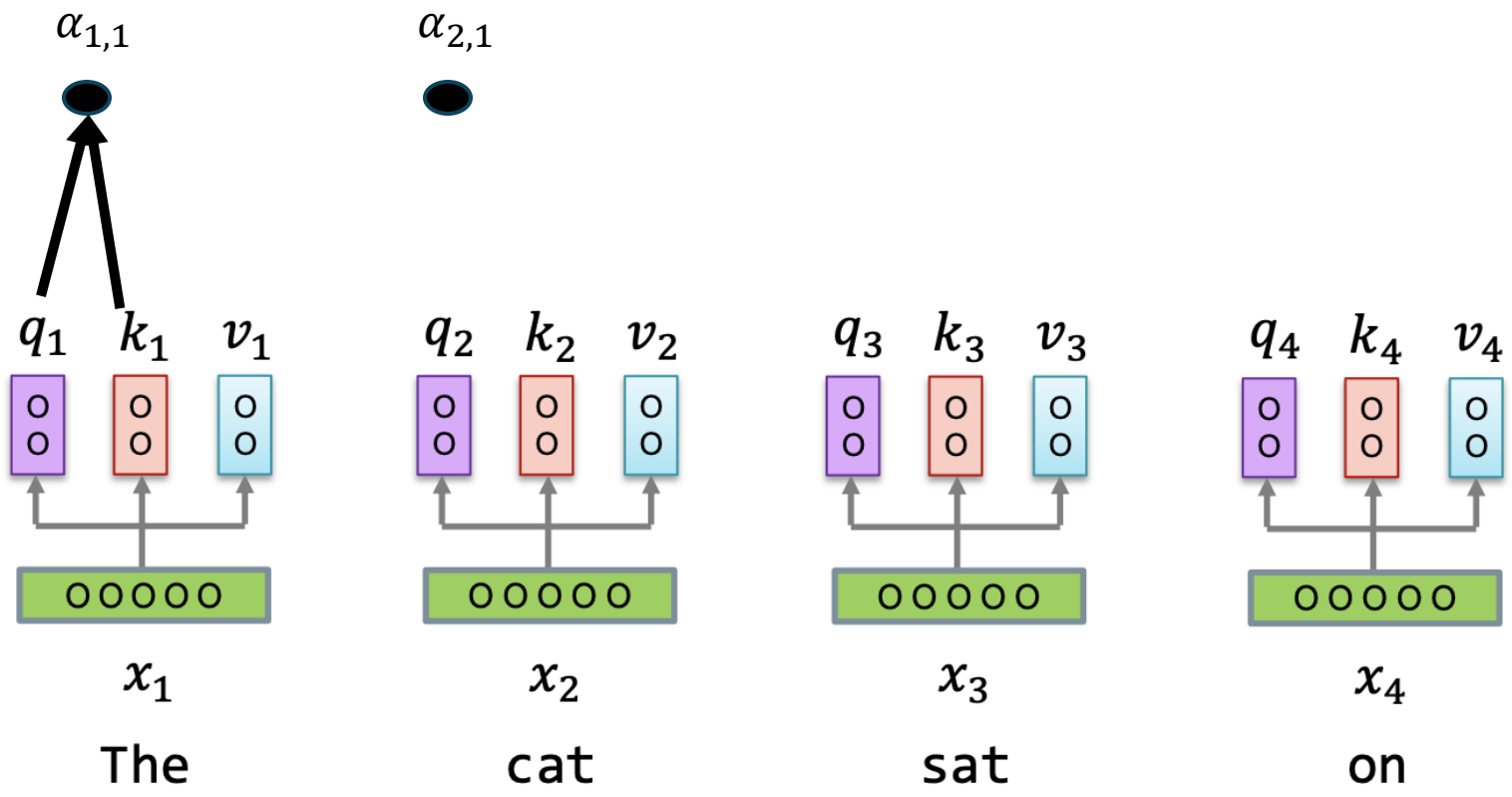
Self-attention



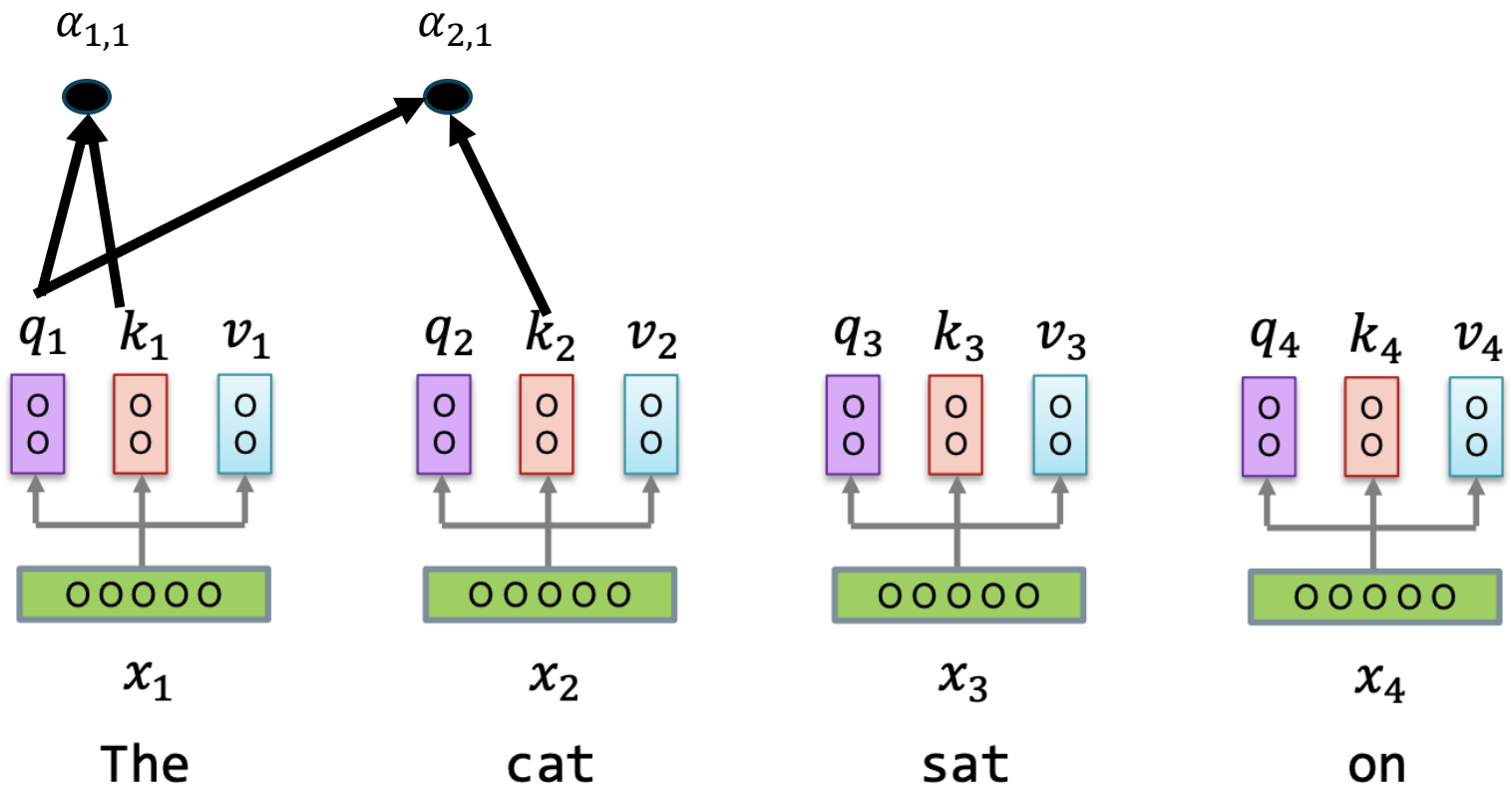
Self-attention



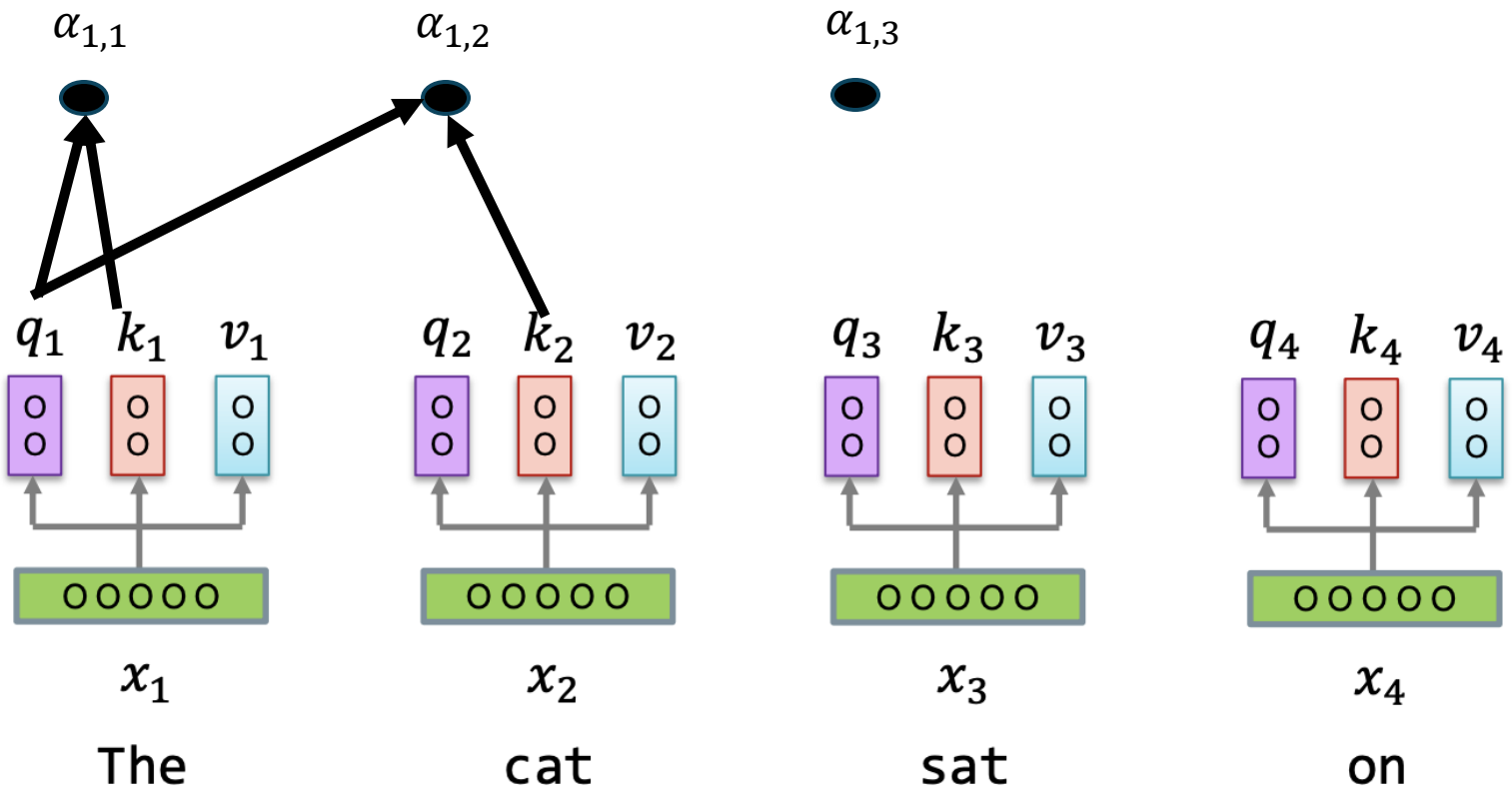
Self-attention



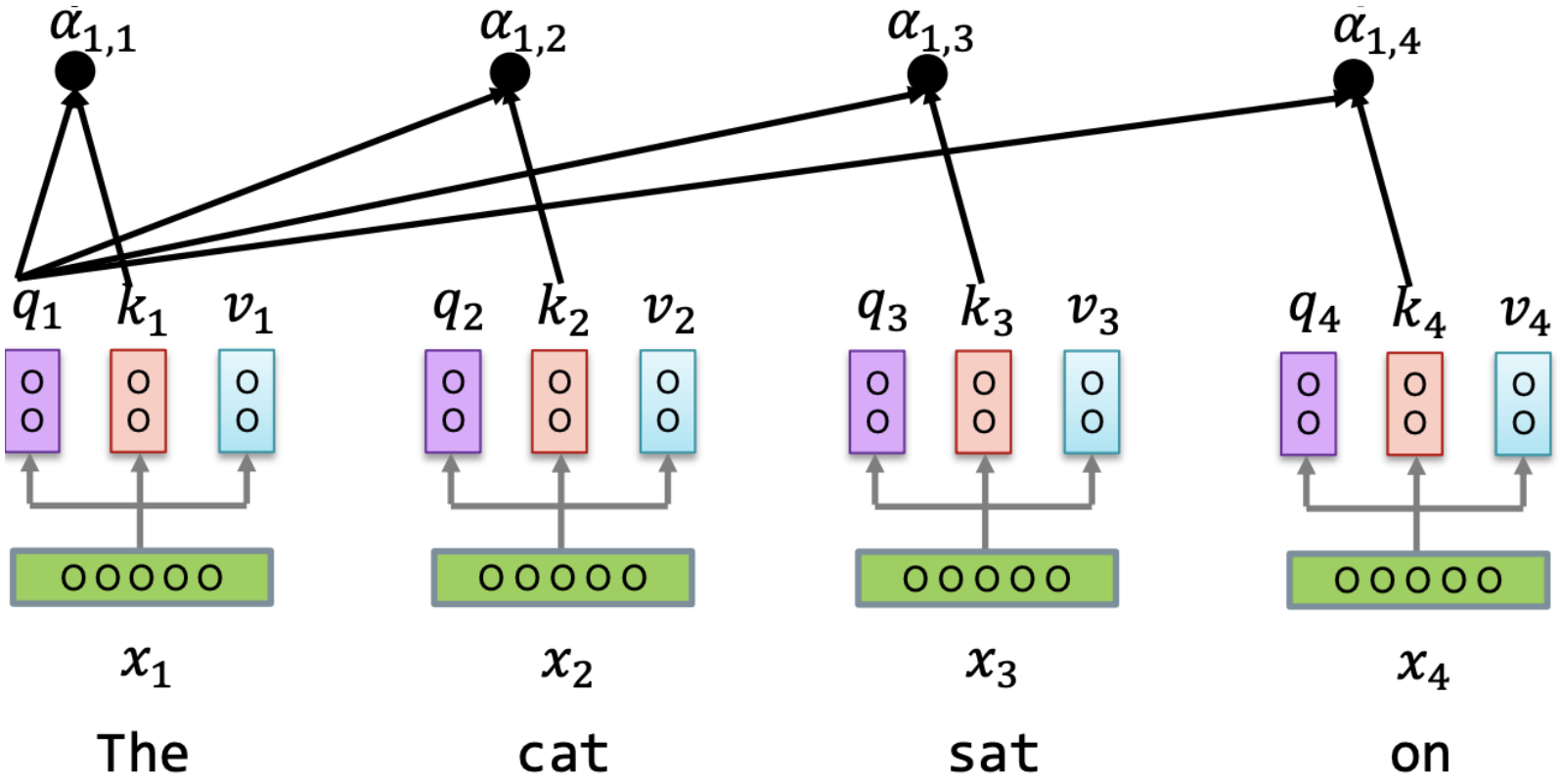
Self-attention



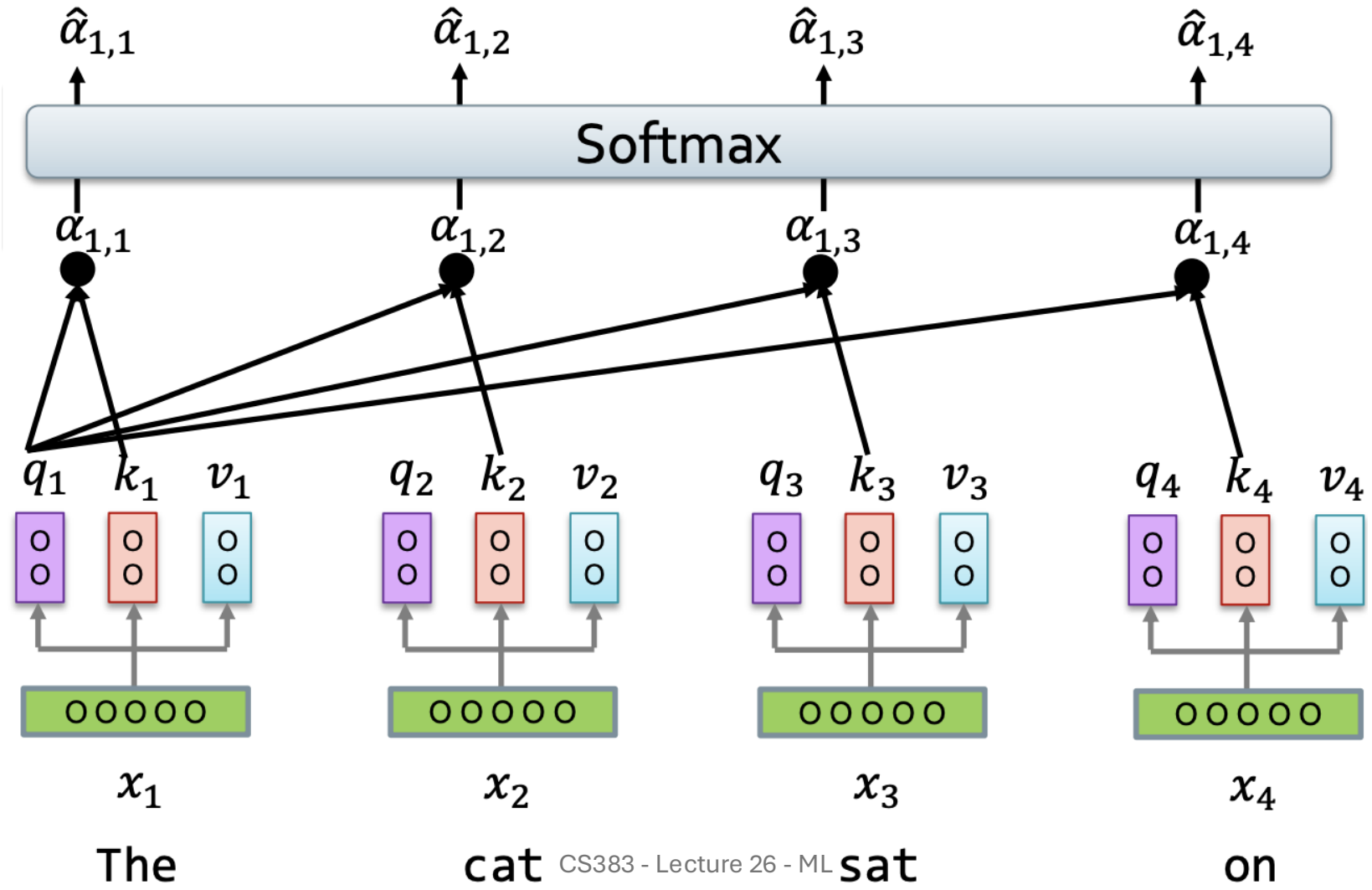
Self-attention



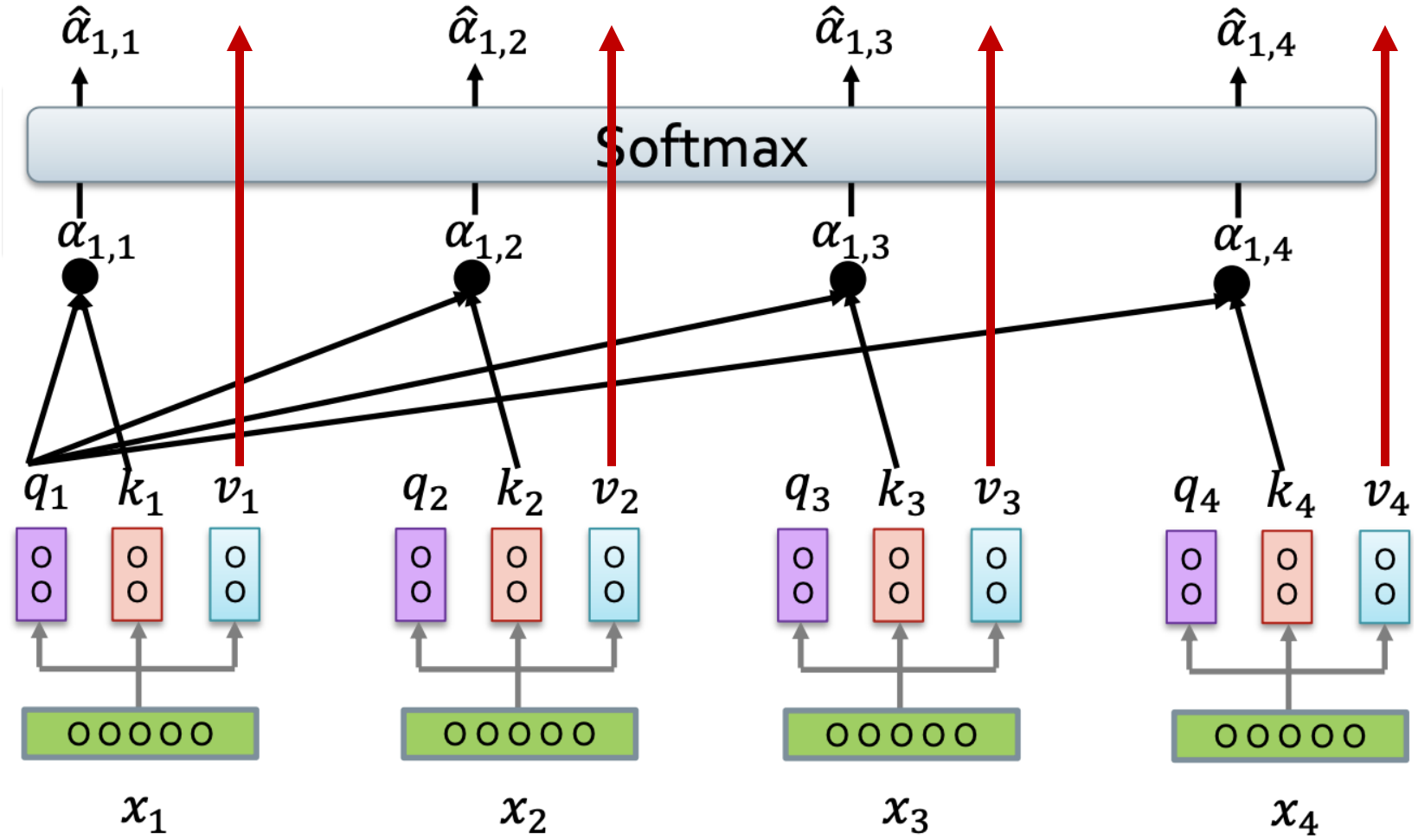
Self-attention



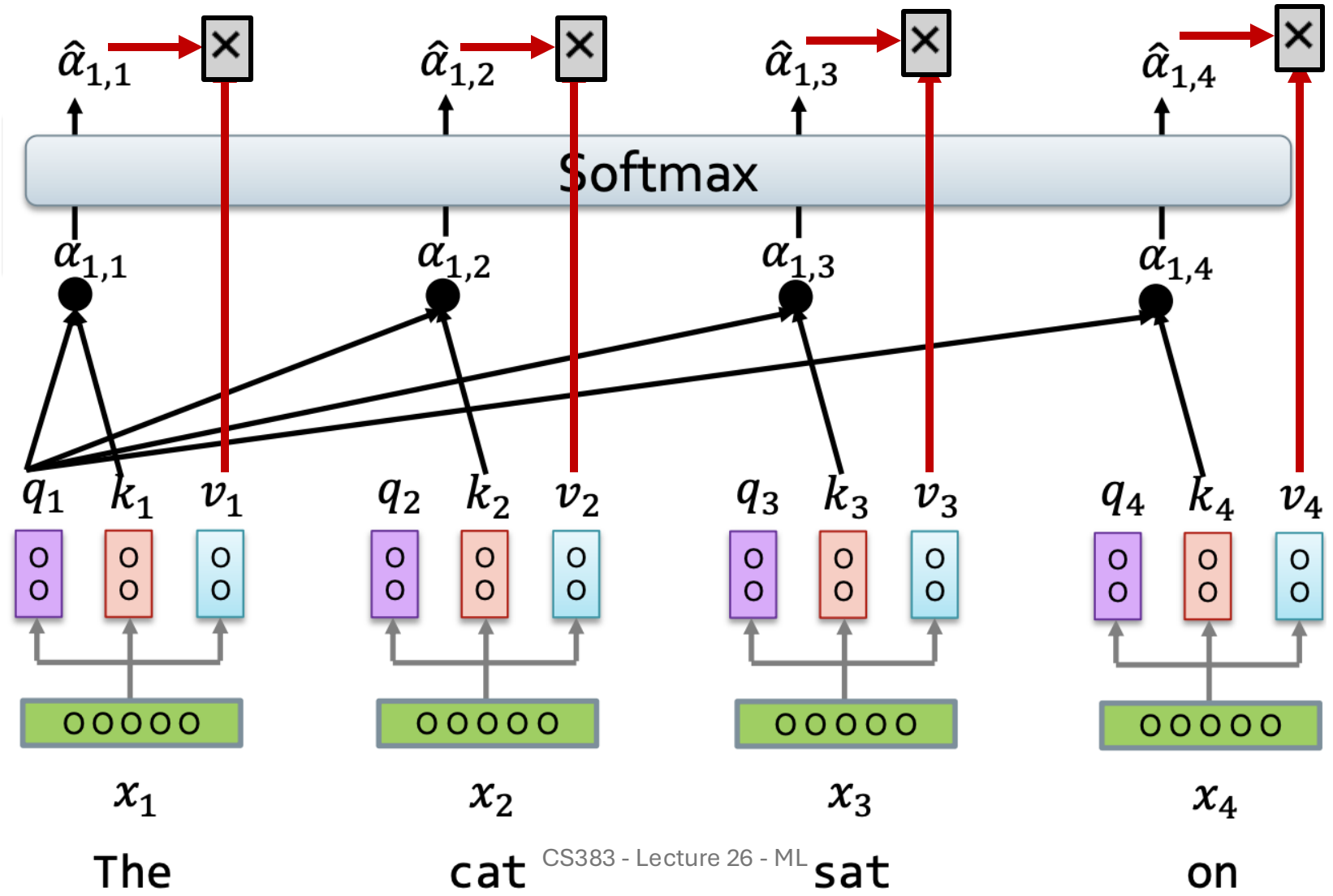
Self-attention



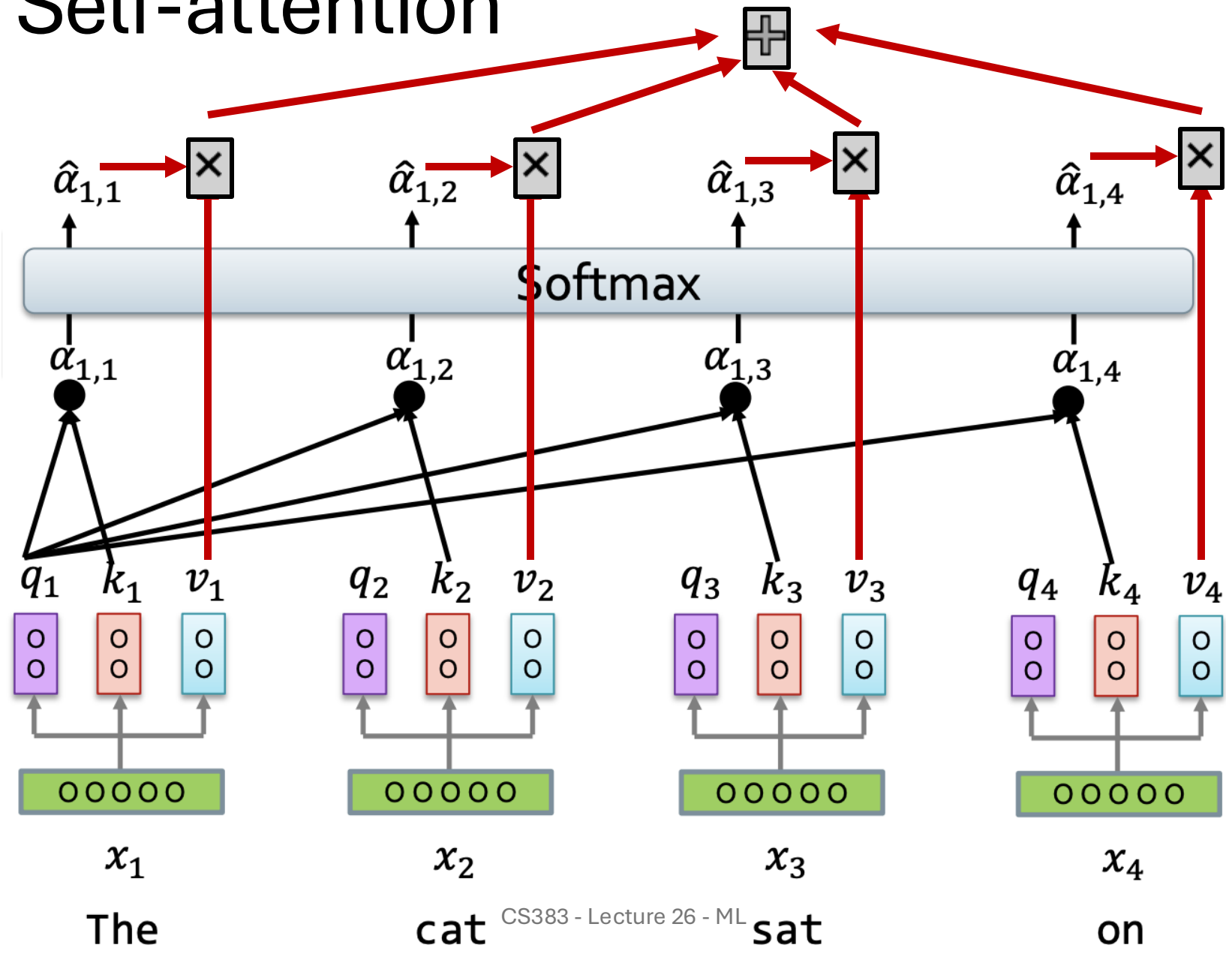
Self-attention



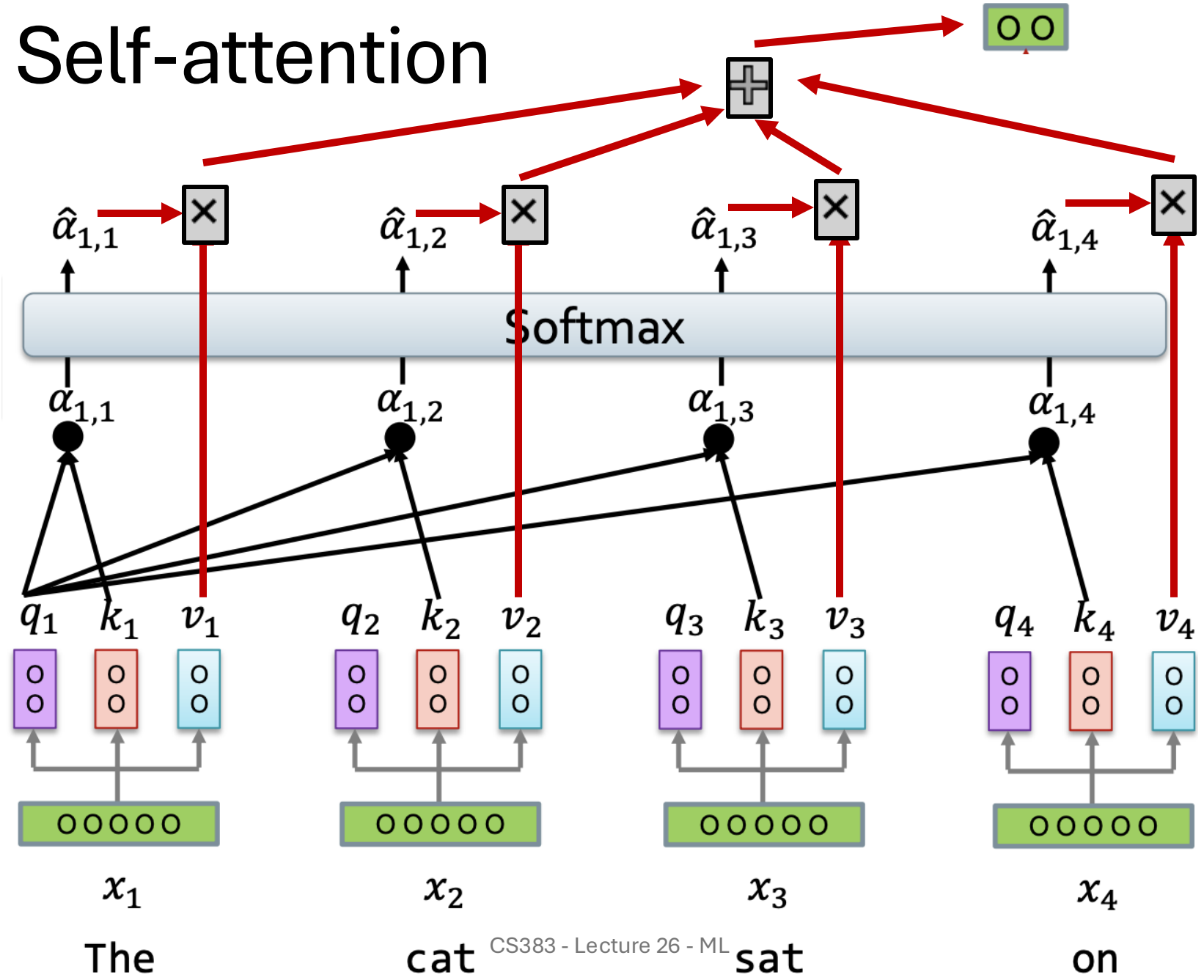
Self-attention



Self-attention

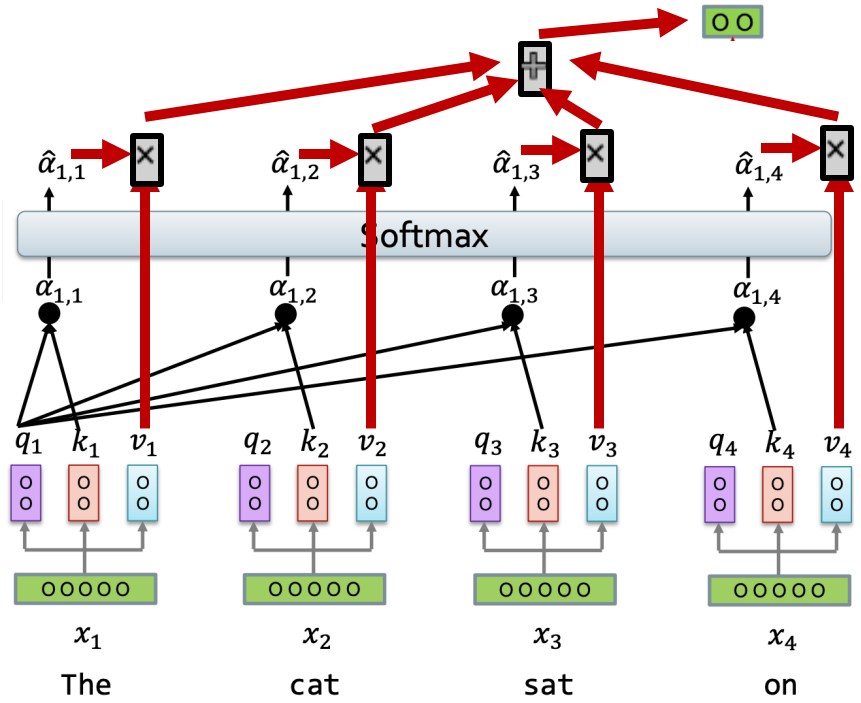


Self-attention



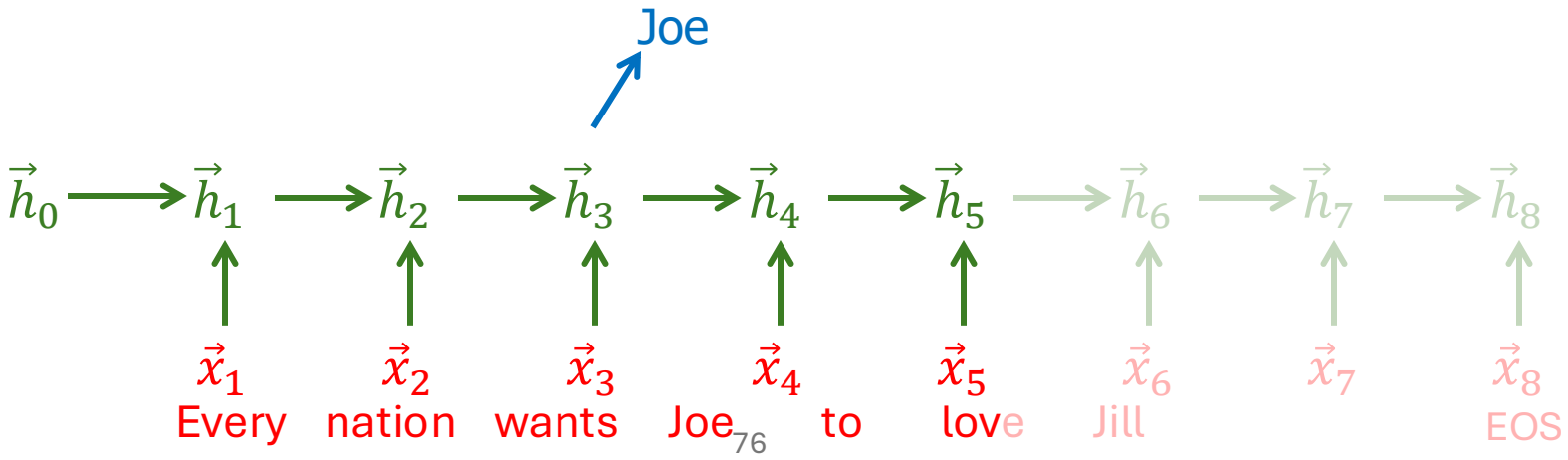
Self-attention

$$A = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

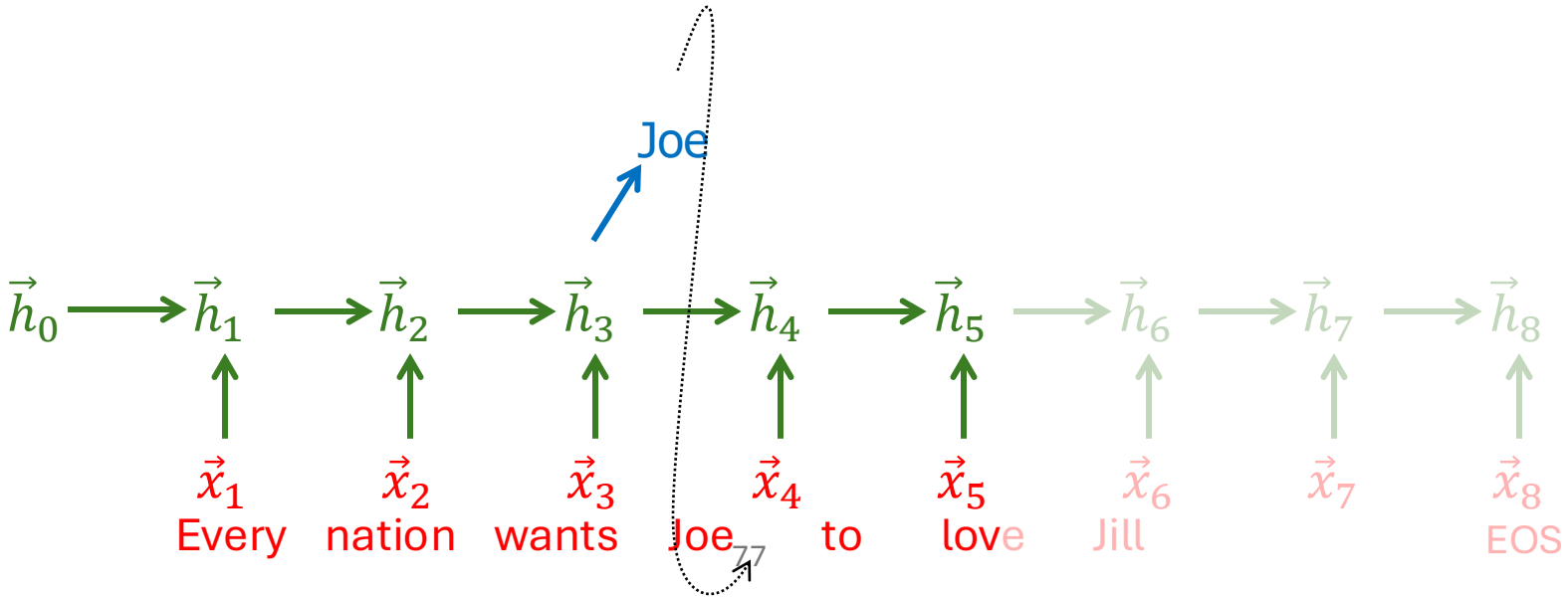


This is the main idea behind a **transformer**

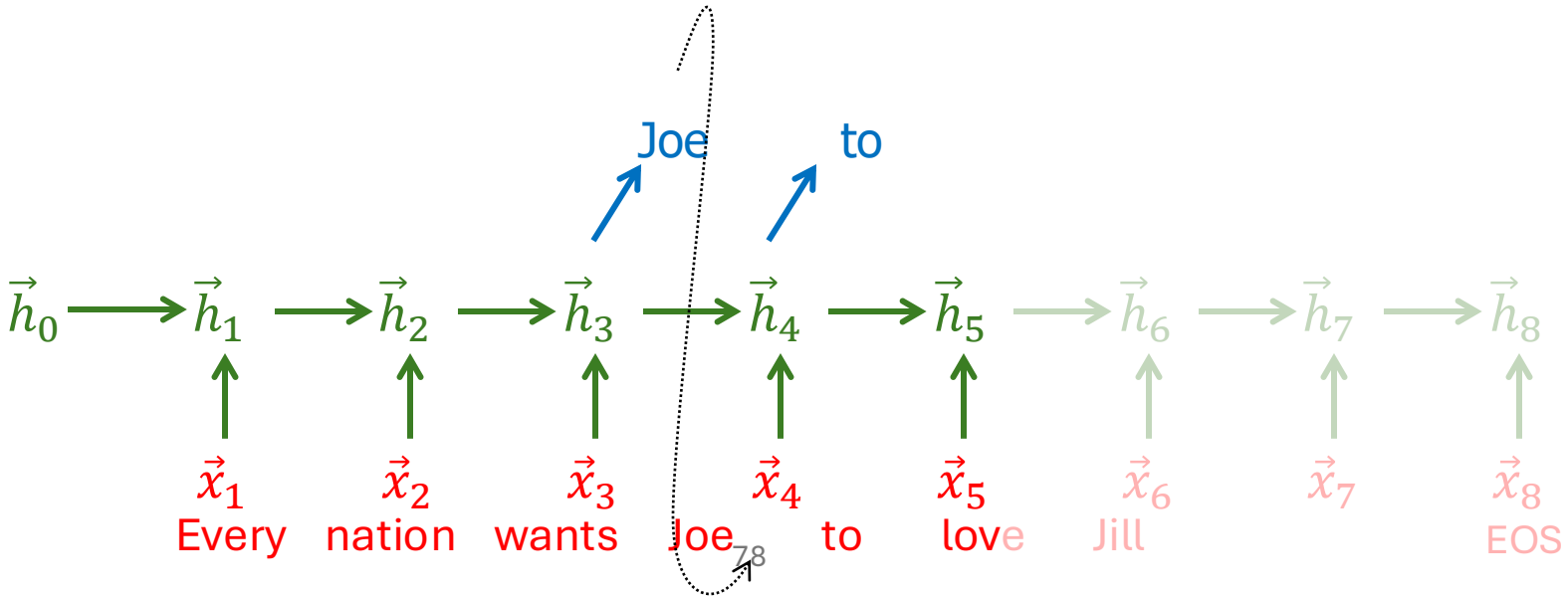
RNN LM


Joe
$\vec{h}_0$ $\vec{h}_1$ $\vec{h}_2$ $\vec{h}_3$ $\vec{h}_4$ $\vec{h}_5$ $\vec{h}_6$ $\vec{h}_7$ $\vec{h}_8$
$\vec{x}_1$ $\vec{x}_2$ $\vec{x}_3$ $\vec{x}_4$ $\vec{x}_5$ $\vec{x}_6$ $\vec{x}_7$ $\vec{x}_8$
Every nation wants Joe76 to love Jill EOS

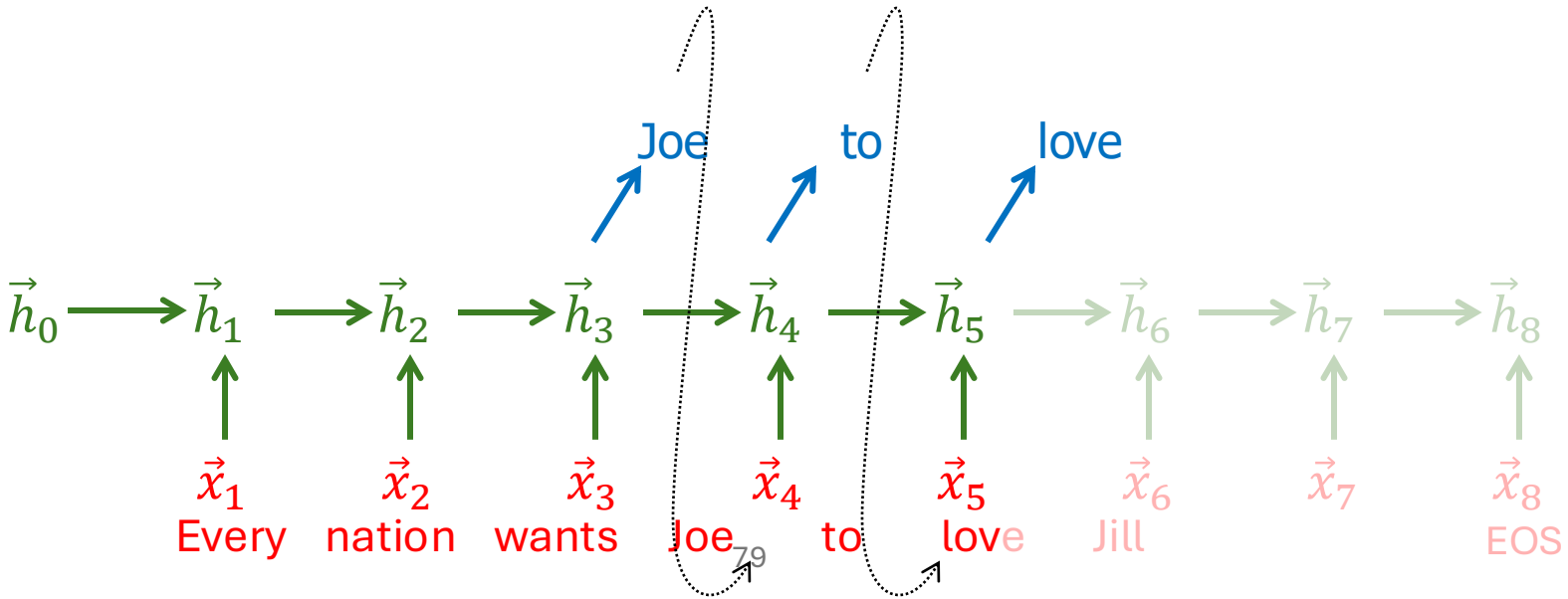
RNN LM


Joe
$\vec{h}_0$ $\vec{h}_1$ $\vec{h}_2$ $\vec{h}_3$ $\vec{h}_4$ $\vec{h}_5$ $\vec{h}_6$ $\vec{h}_7$ $\vec{h}_8$
$\vec{x}_1$ $\vec{x}_2$ $\vec{x}_3$ $\vec{x}_4$ $\vec{x}_5$ $\vec{x}_6$ $\vec{x}_7$ $\vec{x}_8$
Every nation wants Joe to love Jill EOS
77

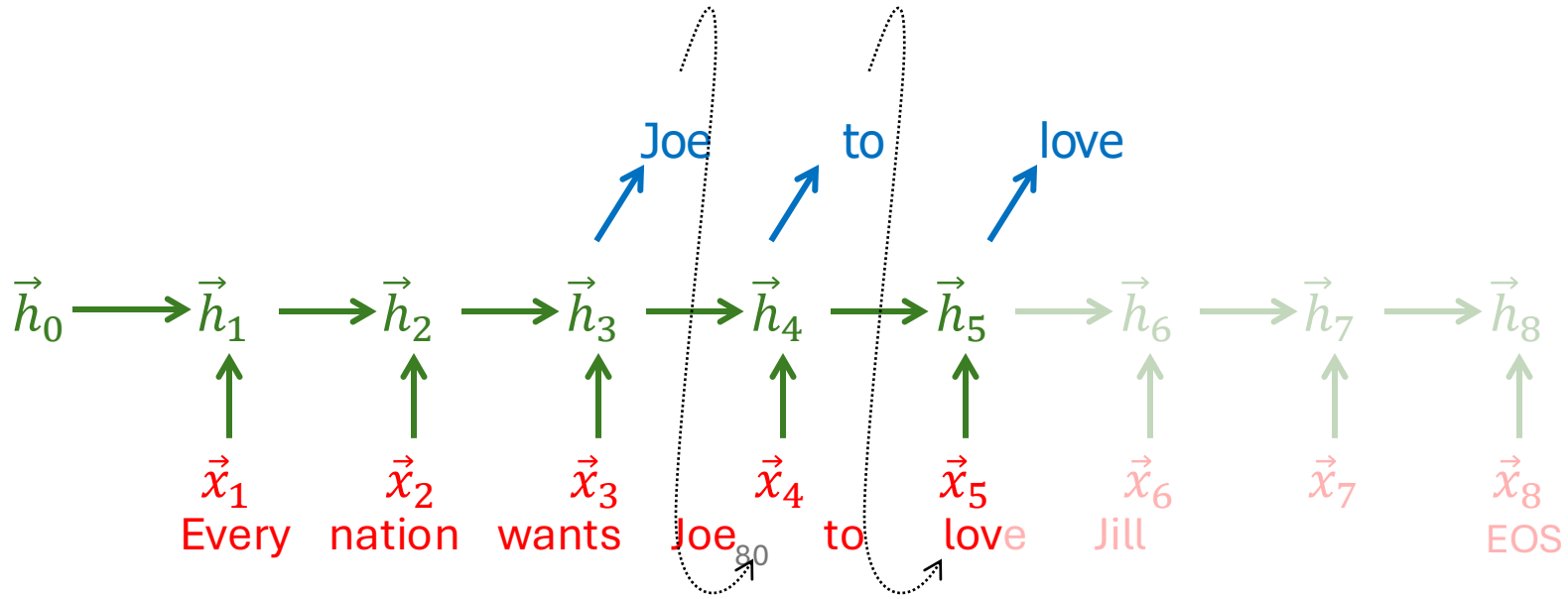
RNN LM


Joe
to
$\vec{h}_0$ $\vec{h}_1$ $\vec{h}_2$ $\vec{h}_3$ $\vec{h}_4$ $\vec{h}_5$ $\vec{h}_6$ $\vec{h}_7$ $\vec{h}_8$
$\vec{x}_1$ $\vec{x}_2$ $\vec{x}_3$ $\vec{x}_4$ $\vec{x}_5$ $\vec{x}_6$ $\vec{x}_7$ $\vec{x}_8$
Every nation wants Joe to love Jill EOS
78

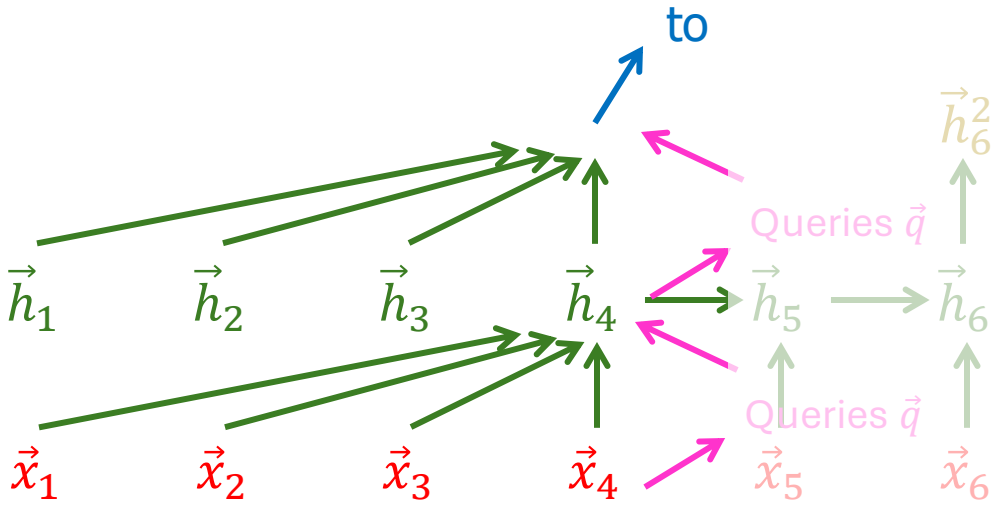
RNN LM


Joe
to
love
$\vec{h}_0$
$\vec{h}_1$
$\vec{h}_2$
$\vec{h}_3$
$\vec{h}_4$
$\vec{h}_5$
$\vec{h}_6$
$\vec{h}_7$
$\vec{h}_8$
$\vec{x}_1$
$\vec{x}_2$
$\vec{x}_3$
$\vec{x}_4$
$\vec{x}_5$
$\vec{x}_6$
$\vec{x}_7$
$\vec{x}_8$
Every
nation
wants
Joe
to
love
Jill
EOS

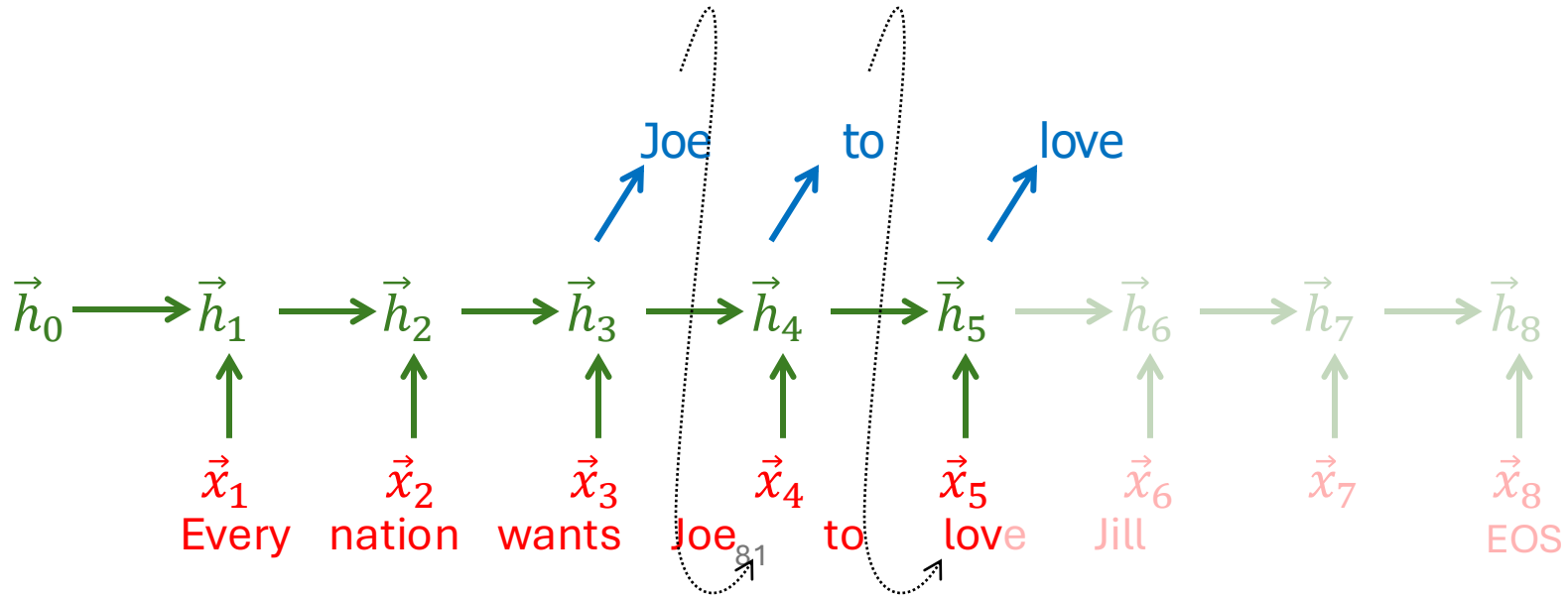
RNN LM



Transformer (self-attention) LM

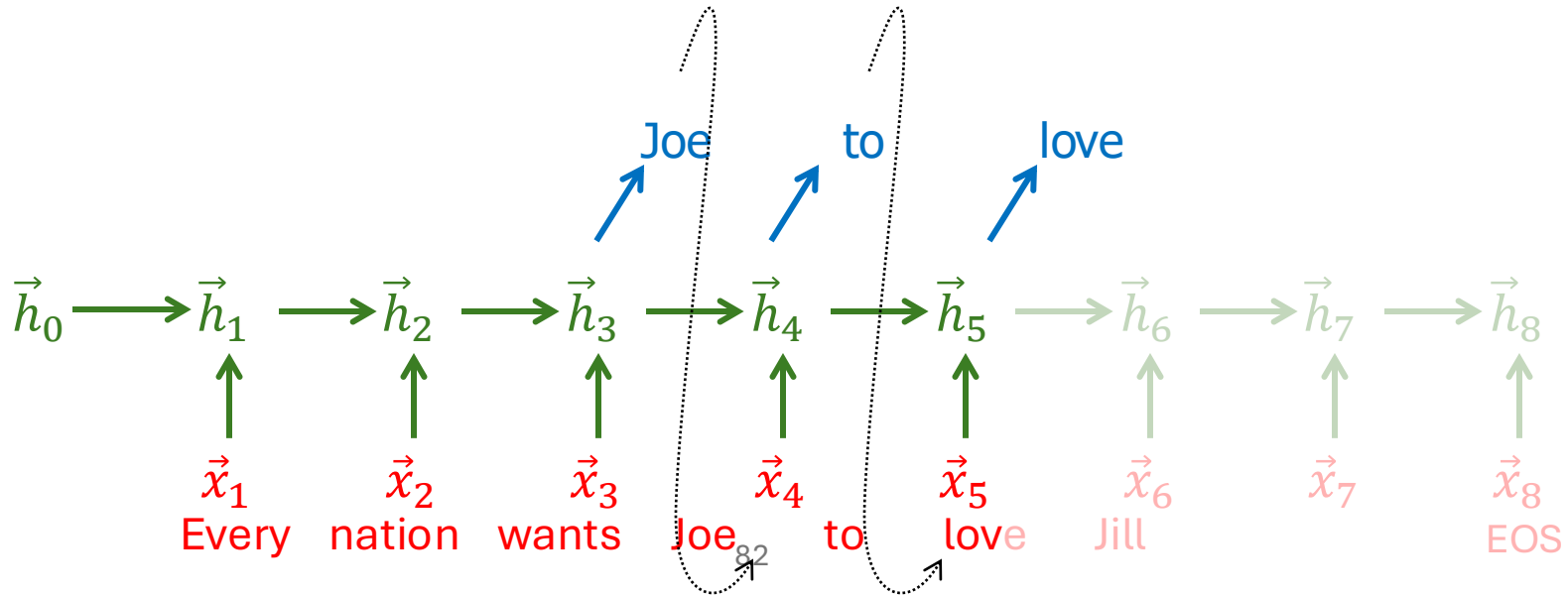


RNN LM

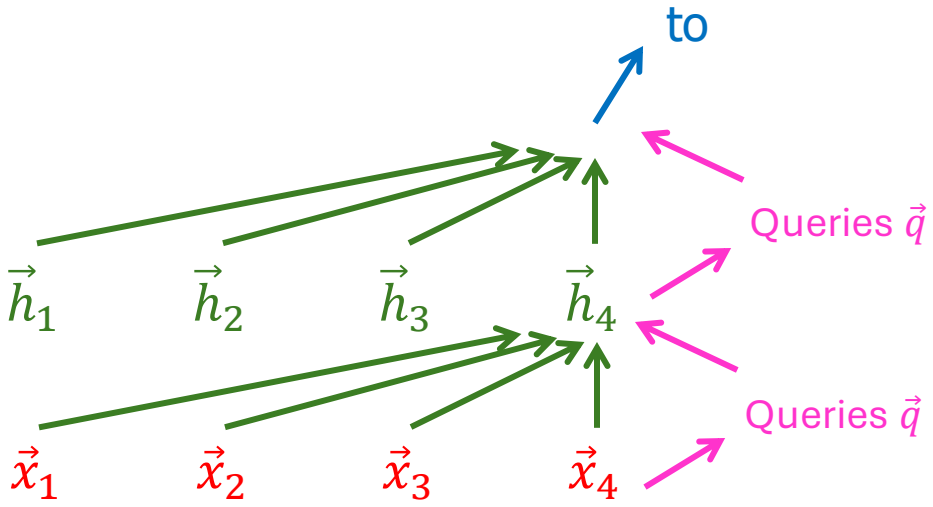


Transformer (self-attention) LM

RNN LM

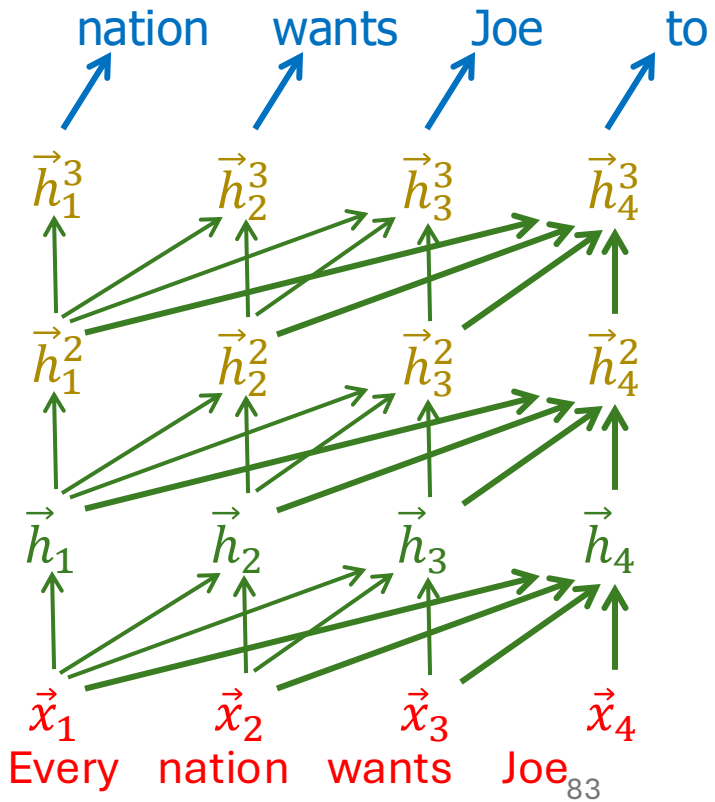


Transformer (self-attention) LM



Training can be parallelized

At training time, the whole sentence is known.
Layer-L representations can be computed in parallel, with each word attending to the layer-(L-1) representations of itself and previous words



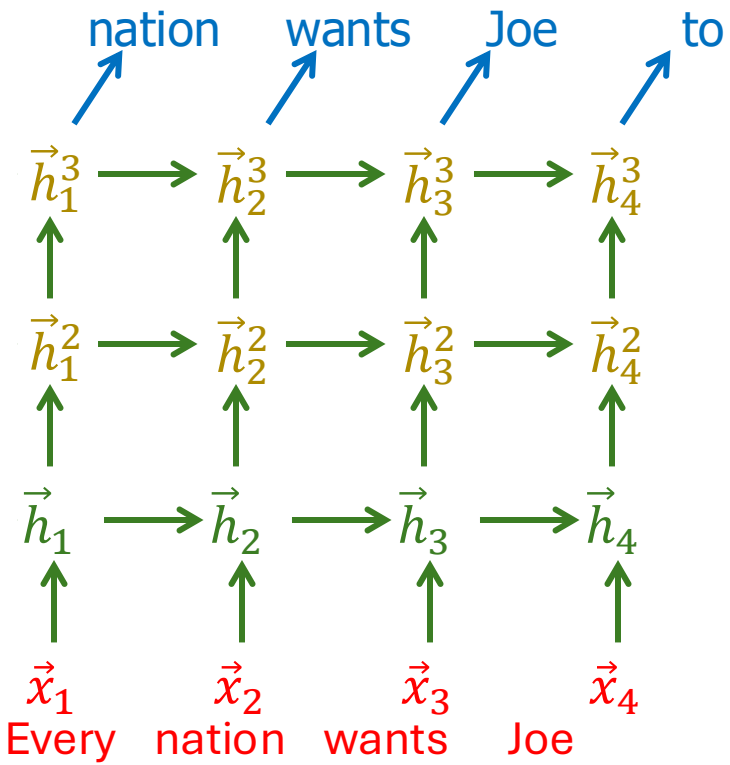
(oops, to predict the very first word, we needed $\vec{x}_0 =< s >$!
It's missing from our diagrams.)

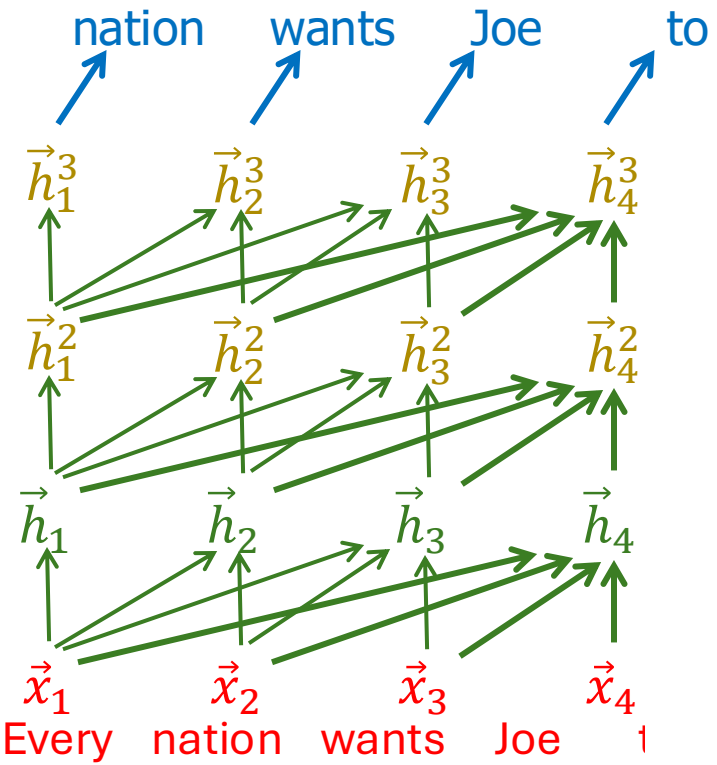
RNN vs. Transformer

Training, on GPU, per layer ↓

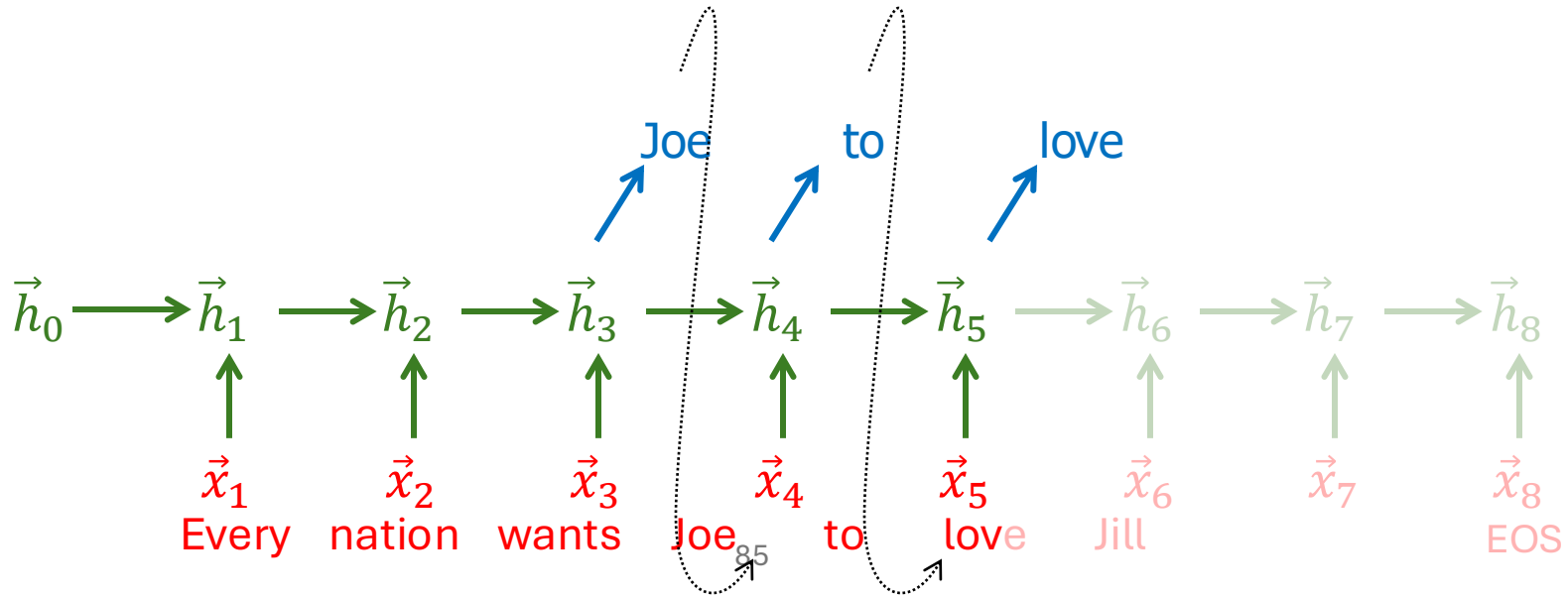
Computations: ☺ O(n)
serial steps: ☹ O(n) due to →

☹ $O(n^2)$
☺ O(1): all ↗↑ in parallel
+ O(log n) to sum n inputs

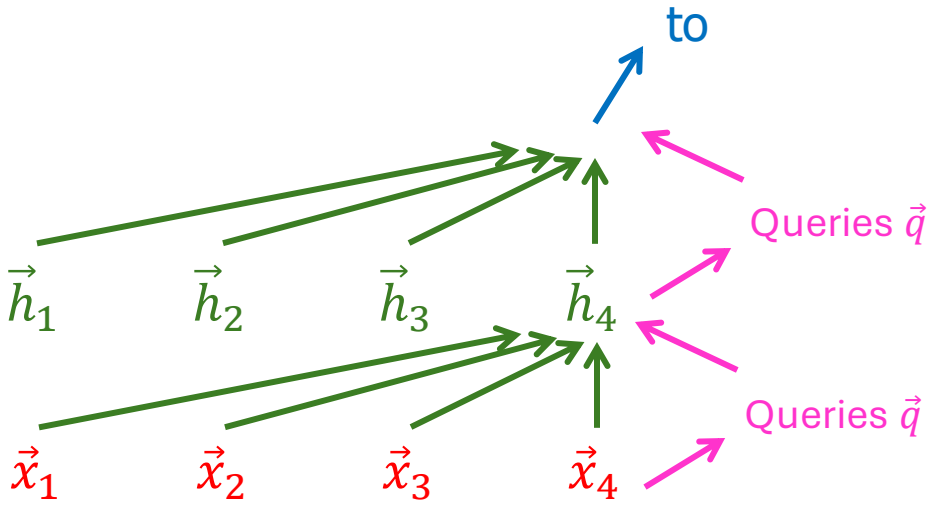




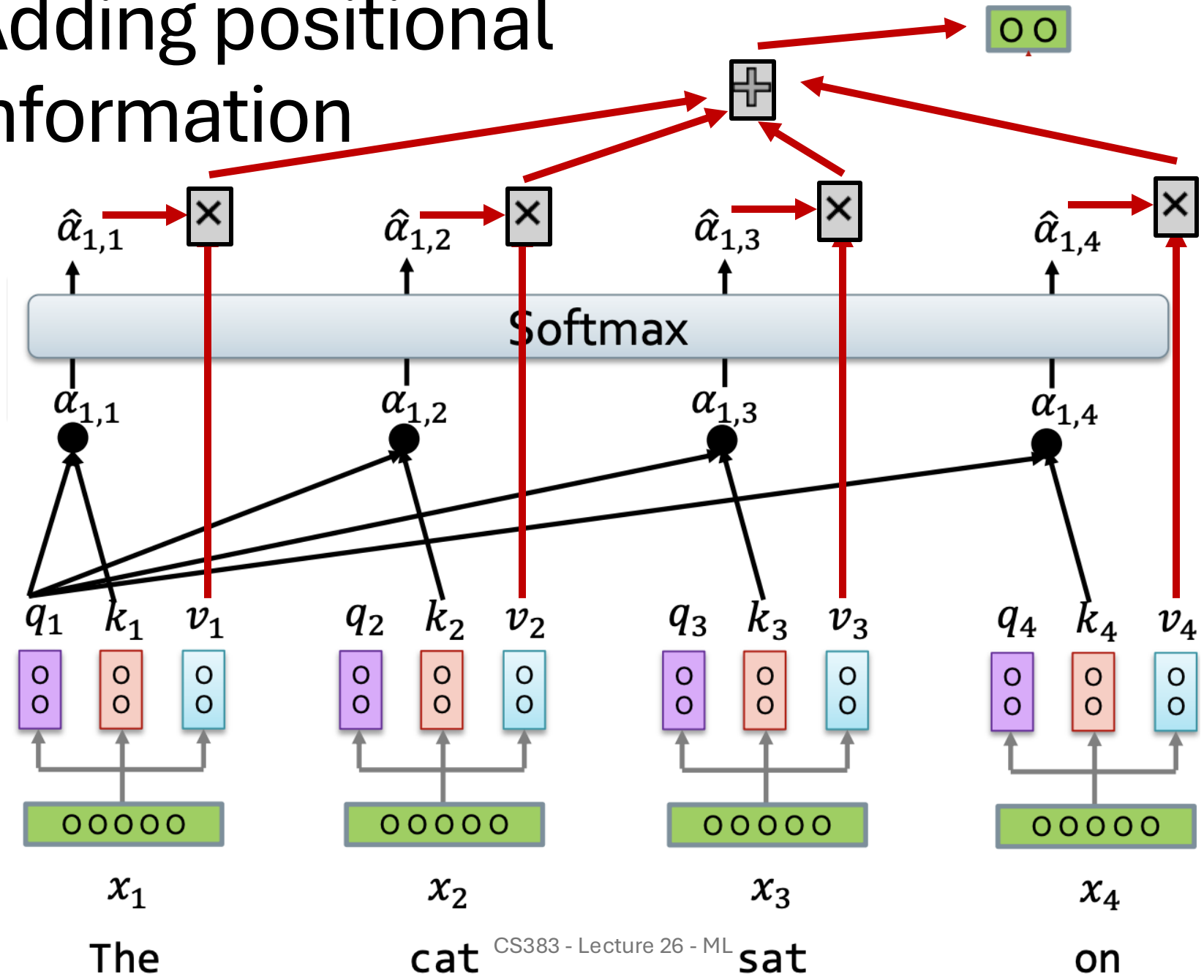
RNN LM



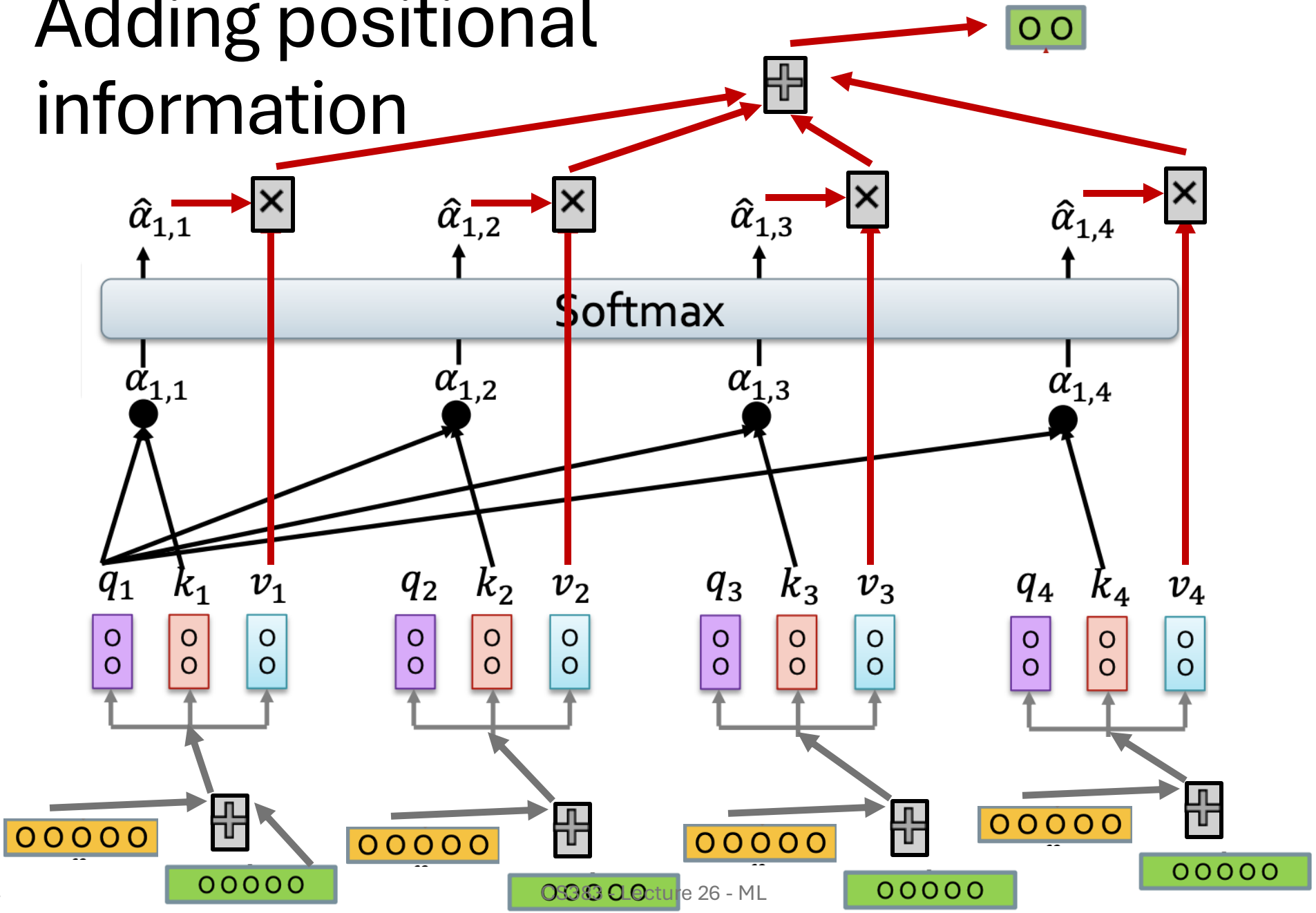
Transformer (self-attention) LM



Adding positional information

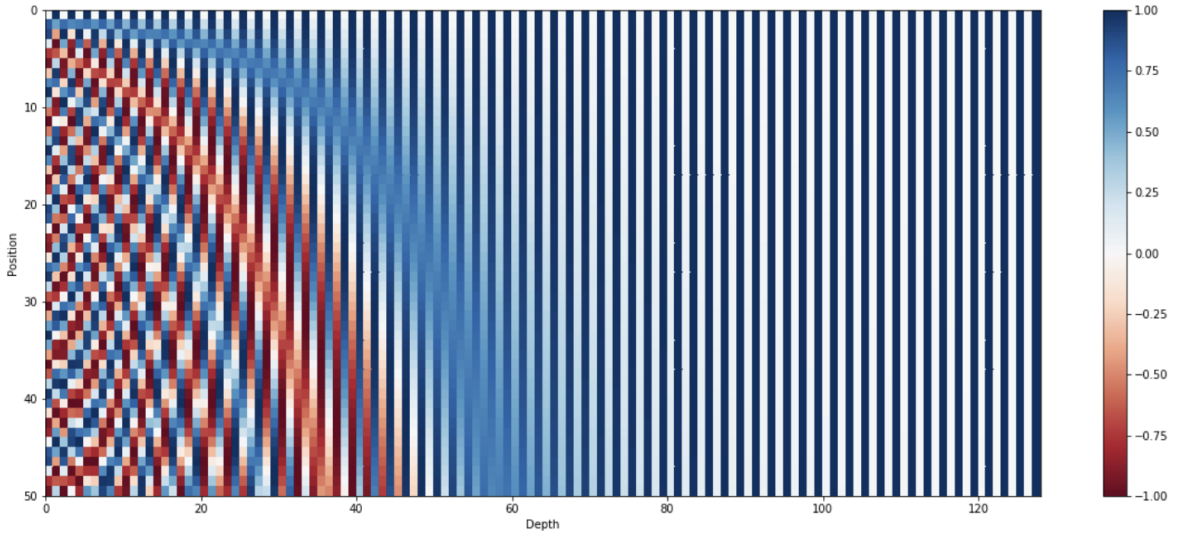


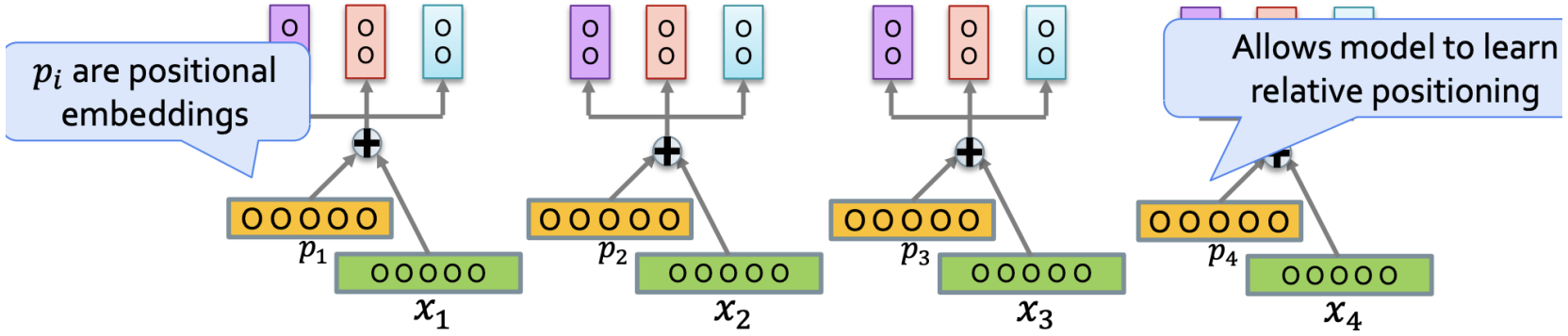
Adding positional information



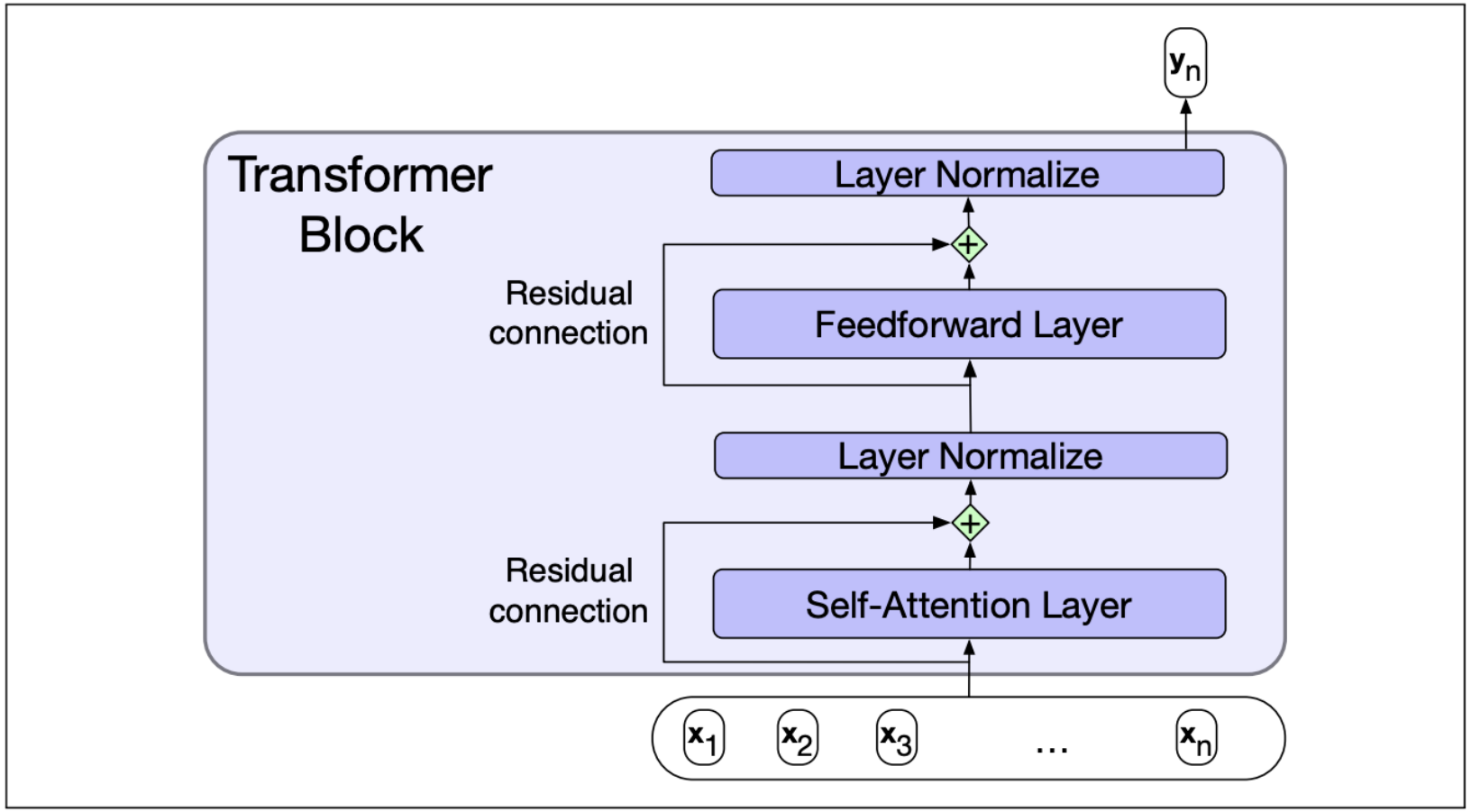
Adding positional information

An approach:
Sine/Cosine encoding





Transformer block



Transformer block

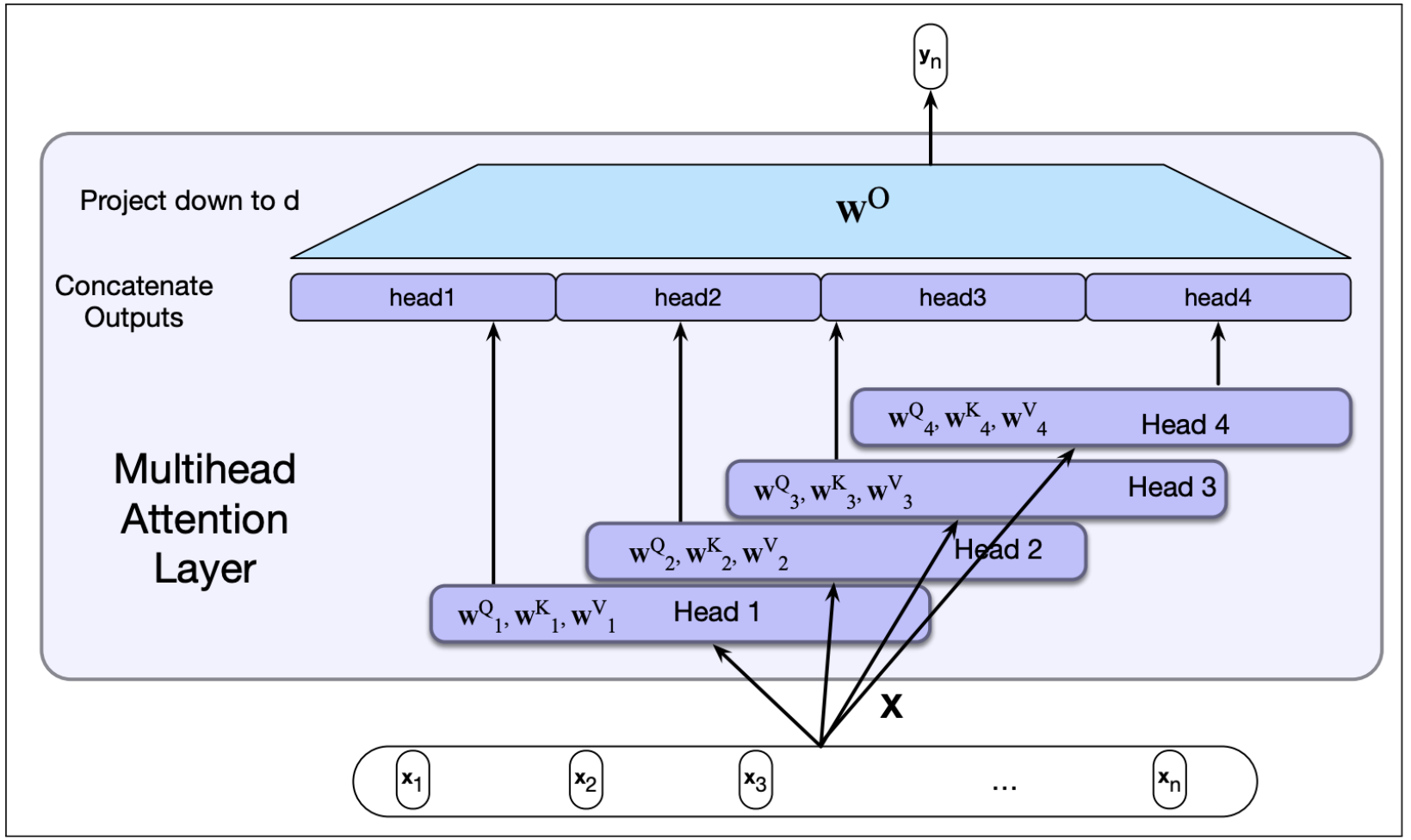
Residual connection:

- Passes information from a lower layer to a higher layer directly (w/out going through intermediate layers)

Layer normalization

- Ensures the values in a layer are in an appropriate range
- Based on normalization/z-scores in statistics (we'll cover normalization later this semester)

Multi-head attention



Transformers as LM

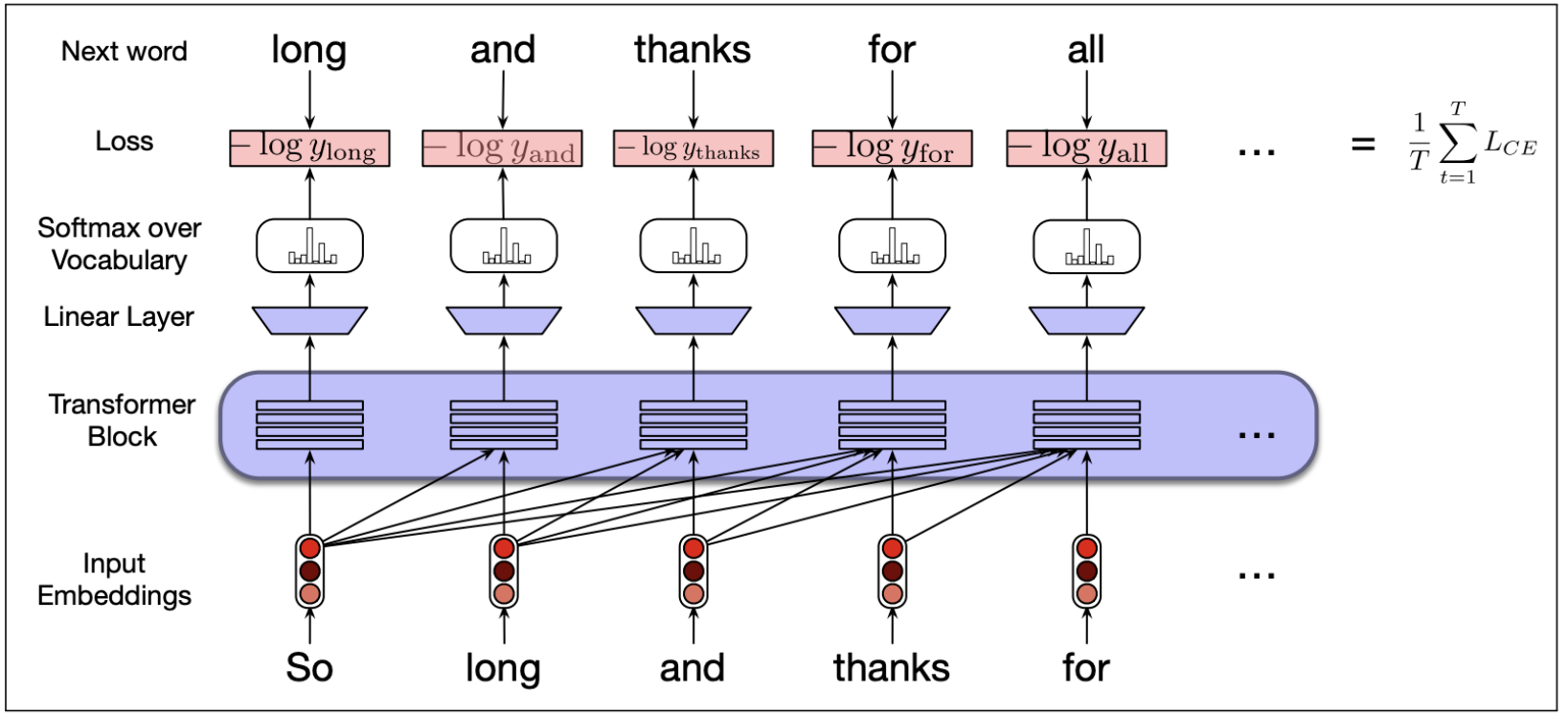


Figure 10.7 Training a transformer as a language model.

Training transformers

parameters in transformer >> # parameters in LSTM/RNN

So, training requires a lot of data

We can pre-train a transformer, and then use it as a sentence-representation/feature extracter

Led to State-of-the-art models

Current LLMs use Transformers