

# CS 383 – Computational Text Analysis

## Lecture 15 Attention & Transformers

Adam Poliak

03/15/2023

Slides adapted from Daniel Khashabi, Chris Manning

# Announcements

- Reading 05
  - Due tonight
- HW06:
  - Due Monday night
  - Notebook from lab (plus two more questions)
  - Logistic Regression in Pytorch
    - More practice with Pytorch and data loading!
- Office hours this week:
  - Normal Thursday slot

# Outline

Recap – RNNs, Seq2Seq

Attention

Self-attention

Transformer

Pytorch demo (if time)

# Machine Learning in a nutshell

In a ML model, what are we training?

- **Parameters!**

How do we train parameters in supervised learning?

*train parameters == figure out values for the parameters*

- Update weights by using them to make predictions and seeing **how far off our predictions** are
  - **Loss function!**

Algorithm to learn weights?

- **SGD**
- Others exist but not covering them

# RNN - motivation

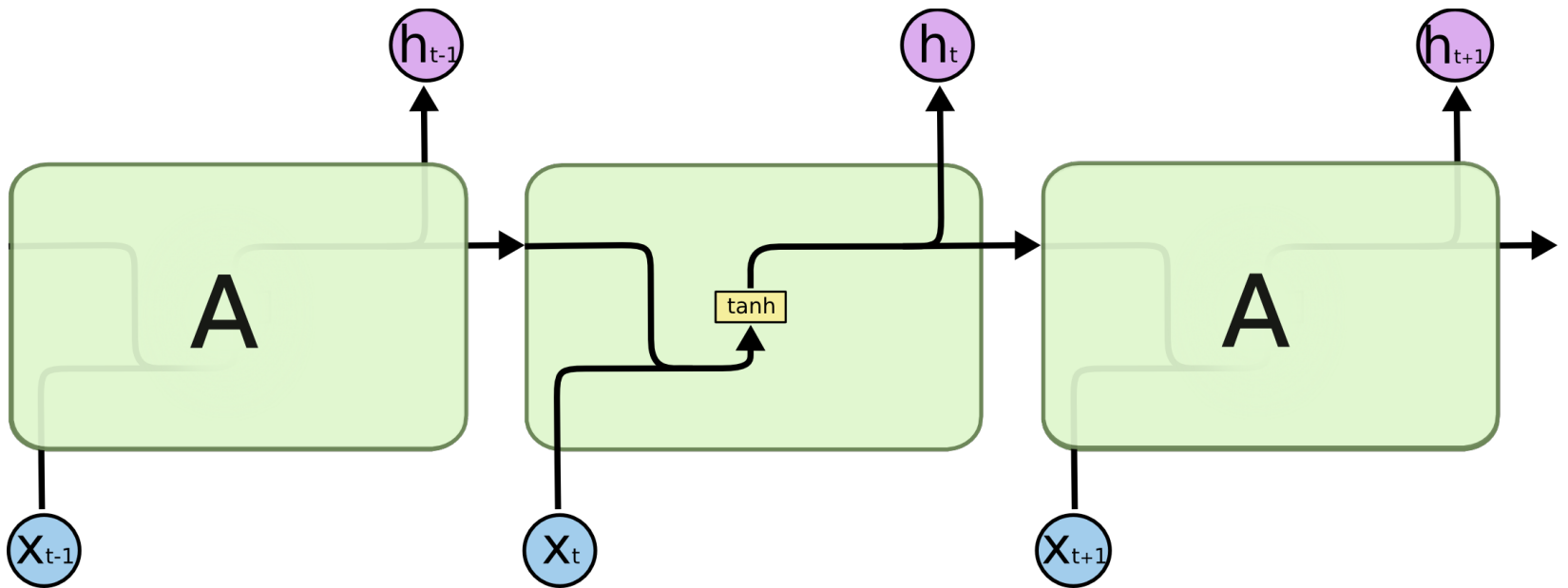
How can we model a **long** (possibly infinite) context using a finite **model**?

Recursion

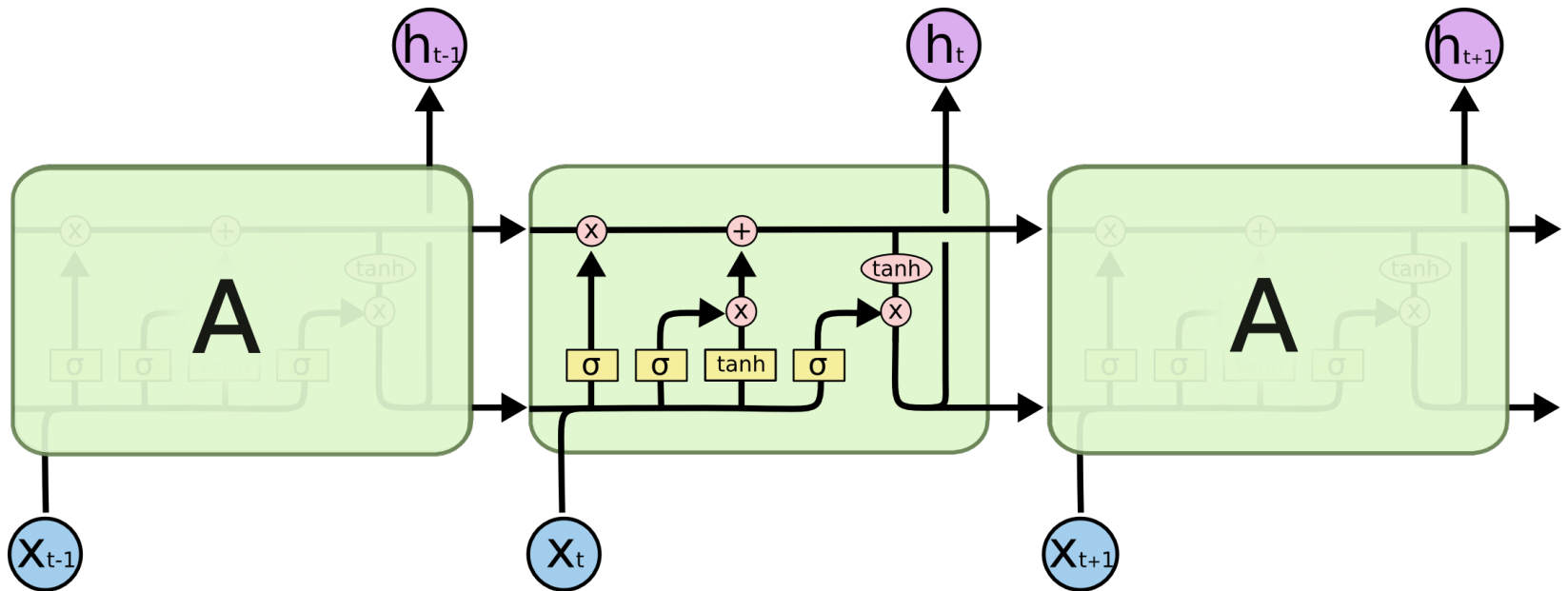
**Recurrent Neural Networks** are a family of NNs that learn sequential data via **recursive dynamics**


# RNN internal

$$h_t = \tanh(x_t W_{ih}^T + b_{ih} + h_{t-1} W_{hh}^T + b_{hh})$$





# LSTM internal



  
Neural Network  
Layer

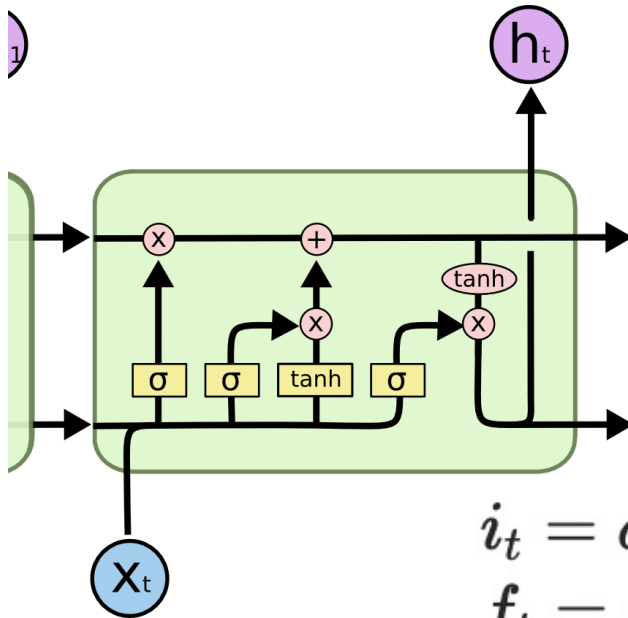
  
Pointwise  
Operation

  
Vector  
Transfer

  
Concatenate

  
Copy

# LSTM internal



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



# LSTMs success

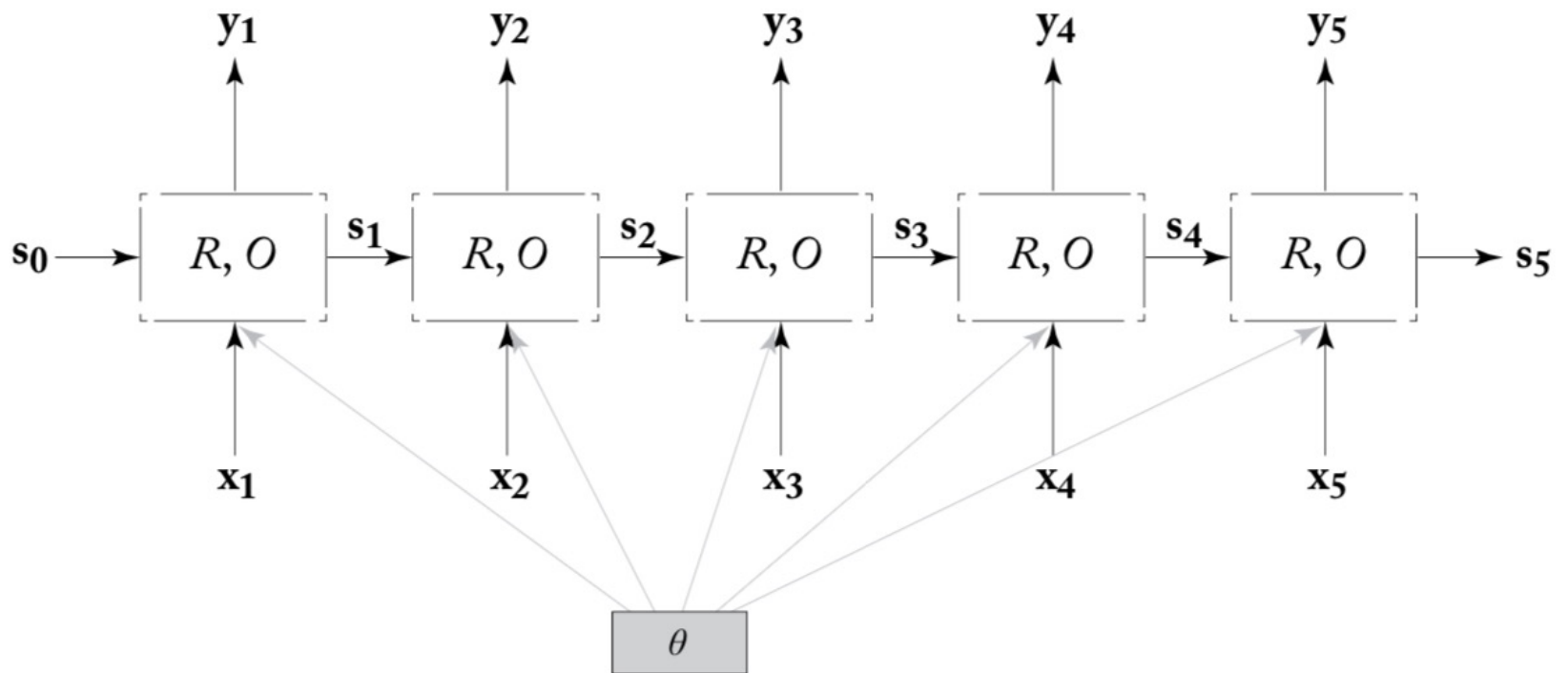
In 2013–2015, LSTMs started achieving state-of-the-art results

- SOTA in tasks like handwriting recognition, speech recognition, machine translation, parsing, and image captioning, LMs
- LSTMs became the dominant approach for most NLP tasks until very recently

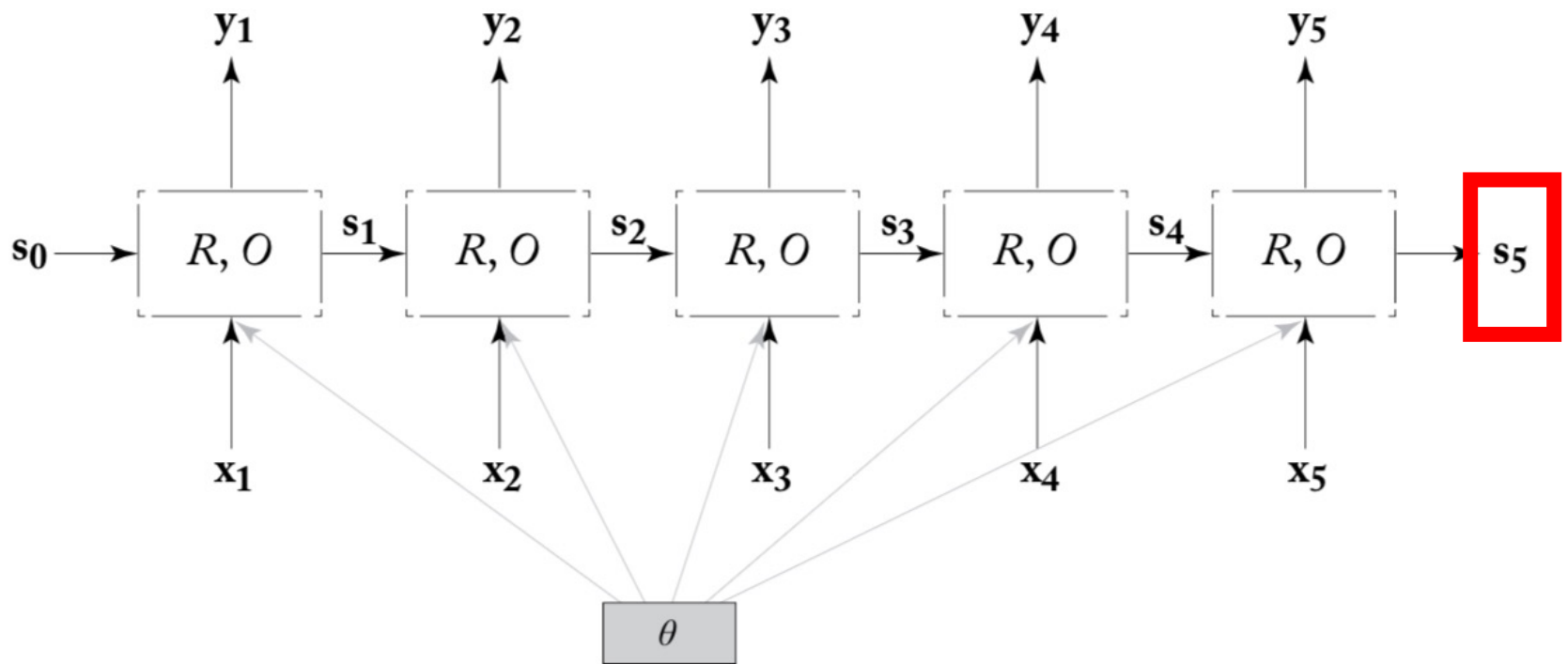
Since 2020, other approaches (e.g., Transformers) have become dominant for many tasks

- WMT (Machine Translation conf + competition):
  - In WMT 2016, the summary report contains “RNN” 44 times
  - In WMT 2019: “RNN” 7 times, “Transformer” 105 times

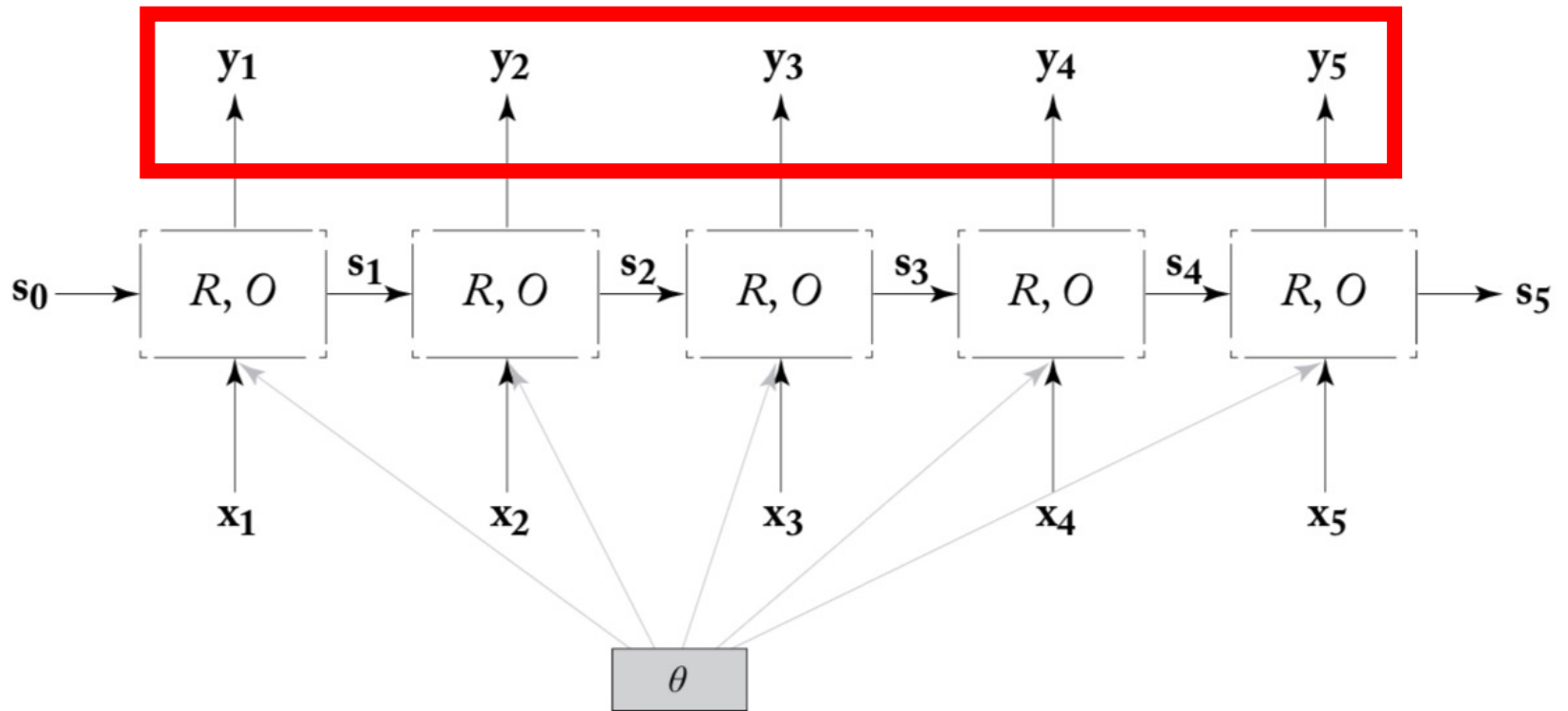
# Extracting representation from RNN layer



# Extracting representation from RNN layer



# Extracting representation from RNN layer



# Min pool implementation

```
tensor = torch.rand((100, 20, 300))
```

Creates a tensor of size:

```
torch.Size([100, 20, 300])
```

What might 100, 20, and 300 indicate if this is what comes out of an RNN?

# Min pool implementation

2 approaches in pytorch:

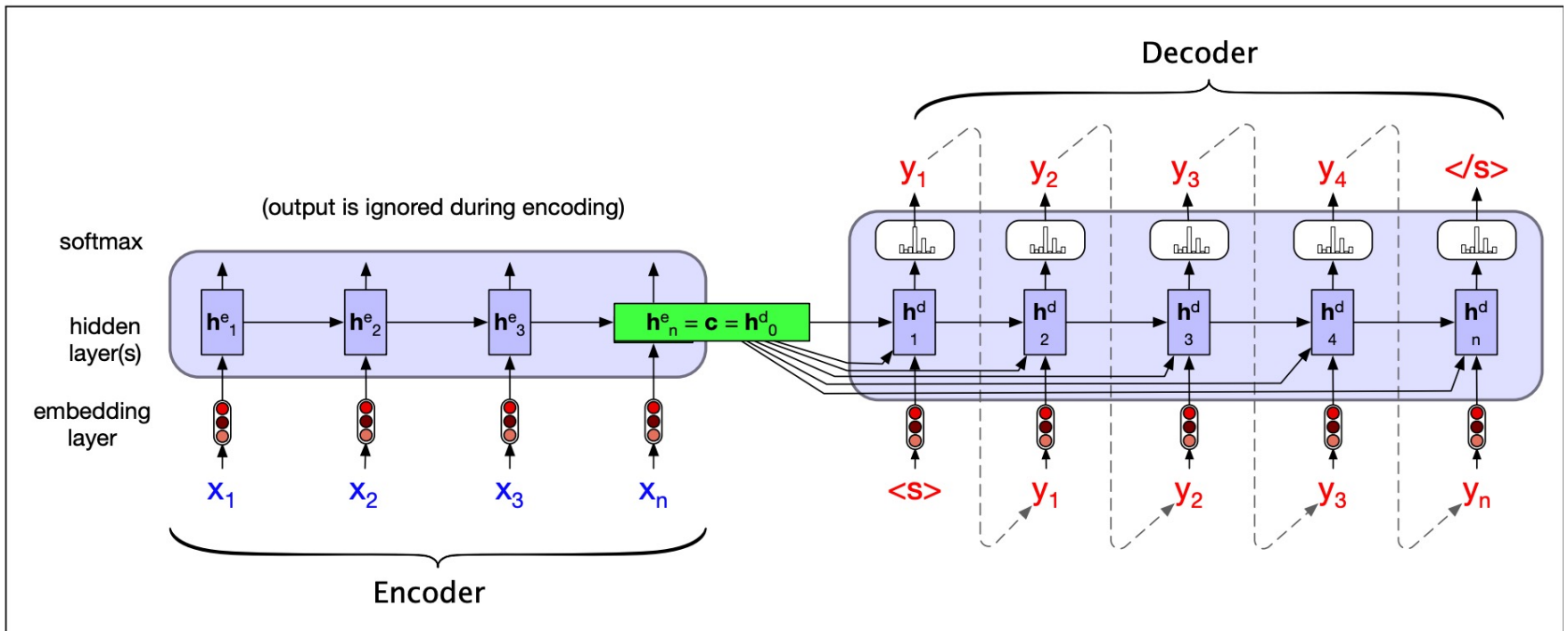
1. `.mean()`

2. `torch.nn.functional.avg_pool2d`

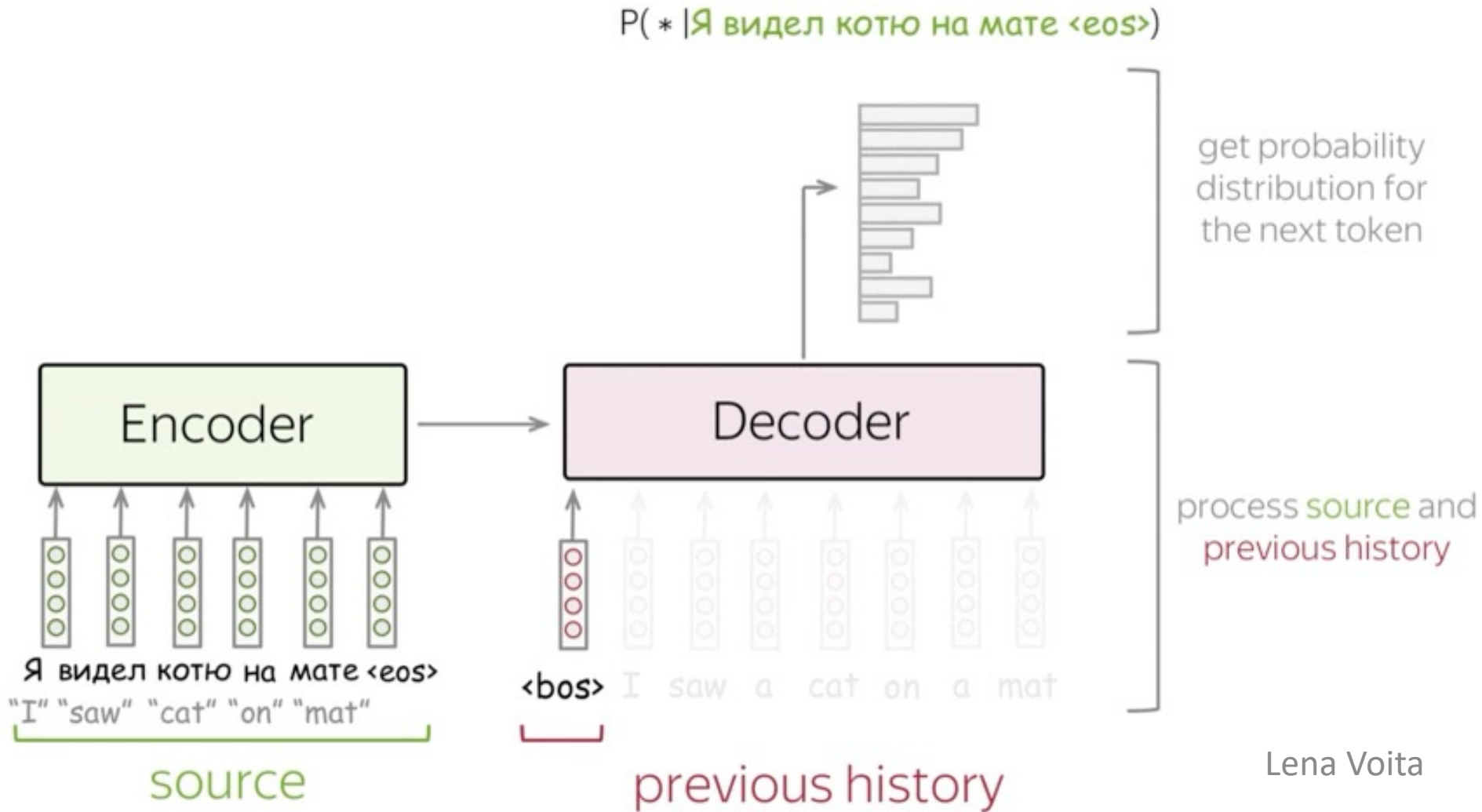
[https://pytorch.org/docs/stable/generated/torch.nn.functional.avg\\_pool2d.html#torch.nn.functional.avg\\_pool2d](https://pytorch.org/docs/stable/generated/torch.nn.functional.avg_pool2d.html#torch.nn.functional.avg_pool2d)

# Encoder-decoder

Decoder only uses information from last hidden cell!



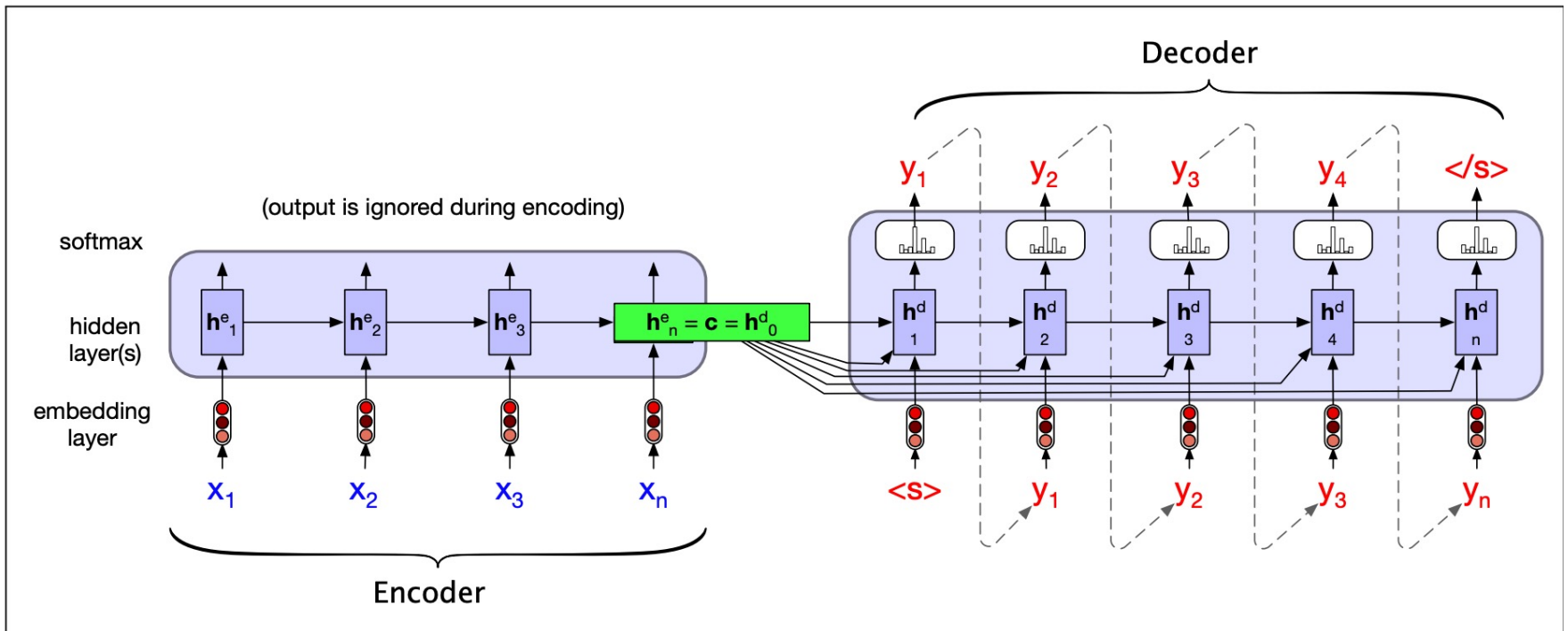
# Encoder-decoder in action





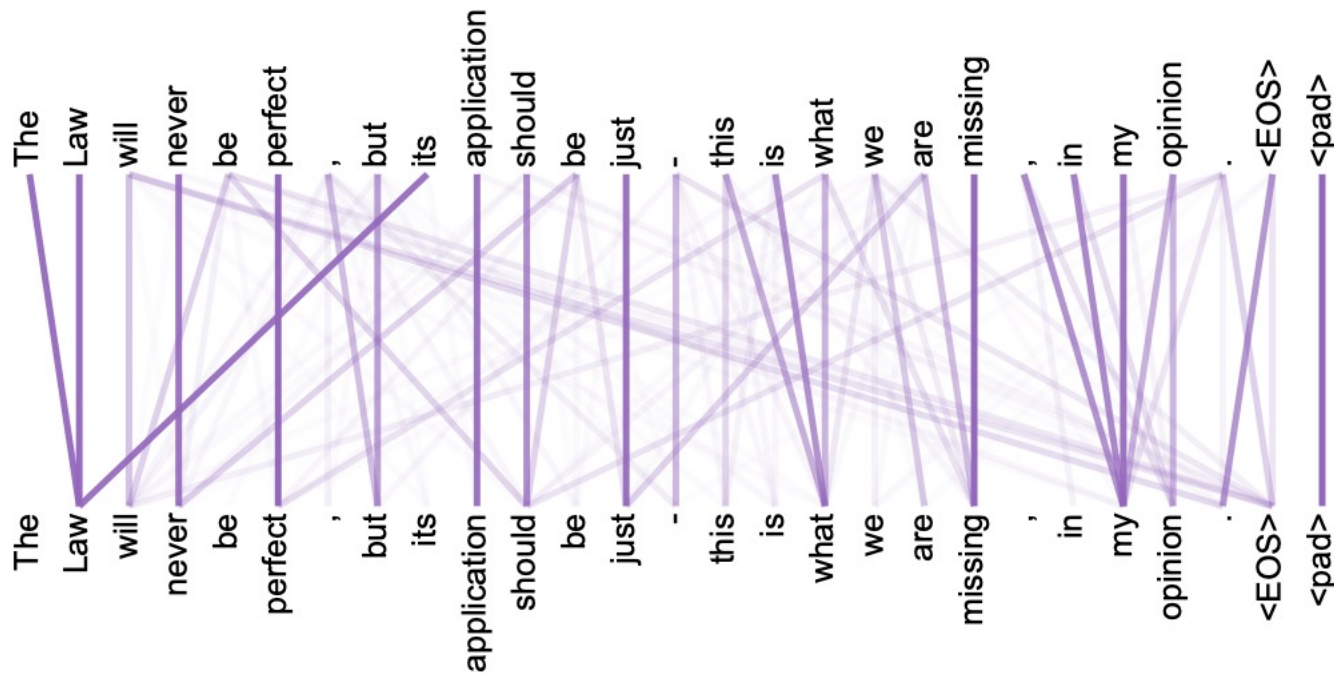
# Bottleneck

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

# Outline

Recap – RNNs, Seq2Seq

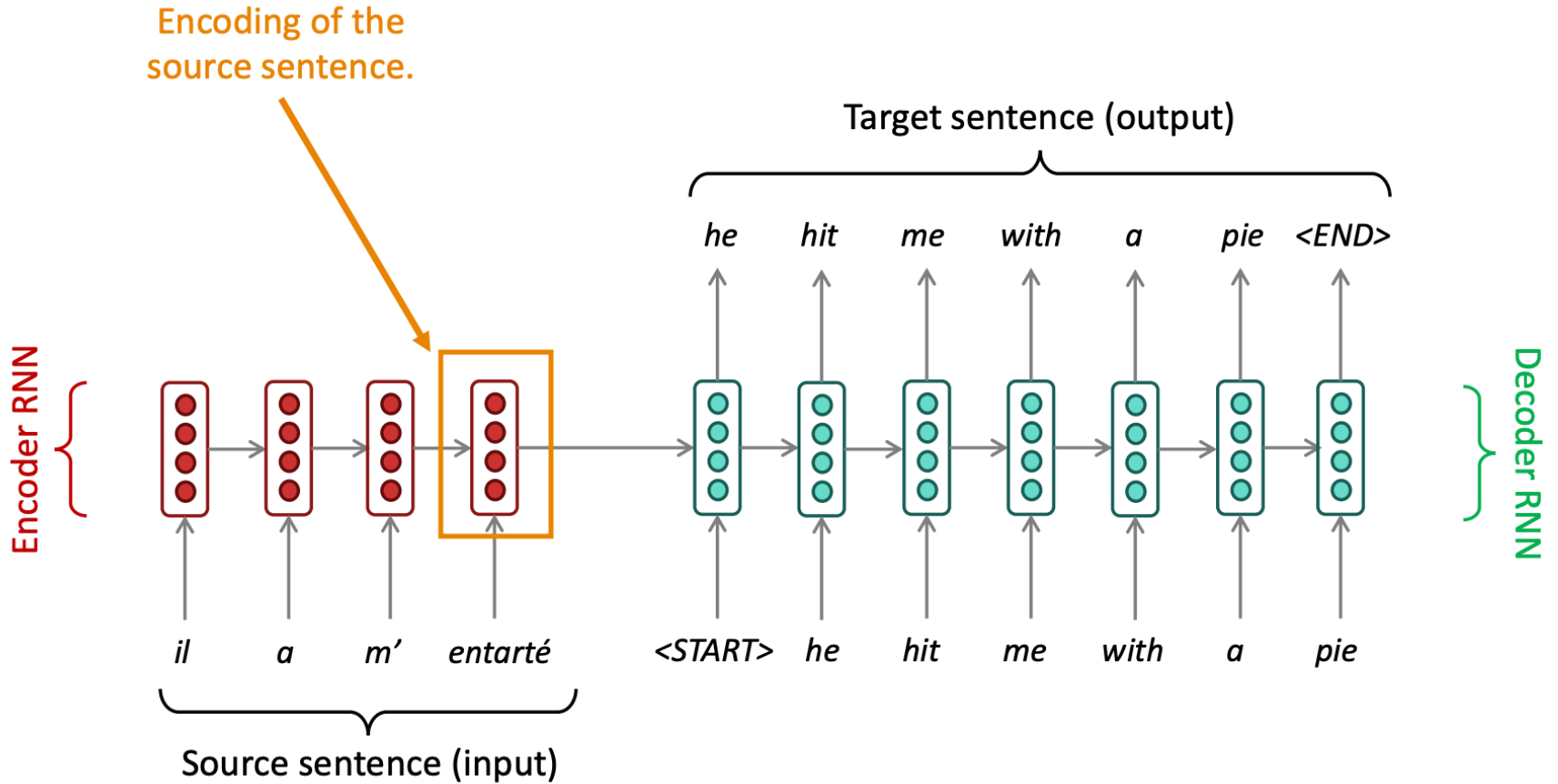
## **Attention**

Self-attention

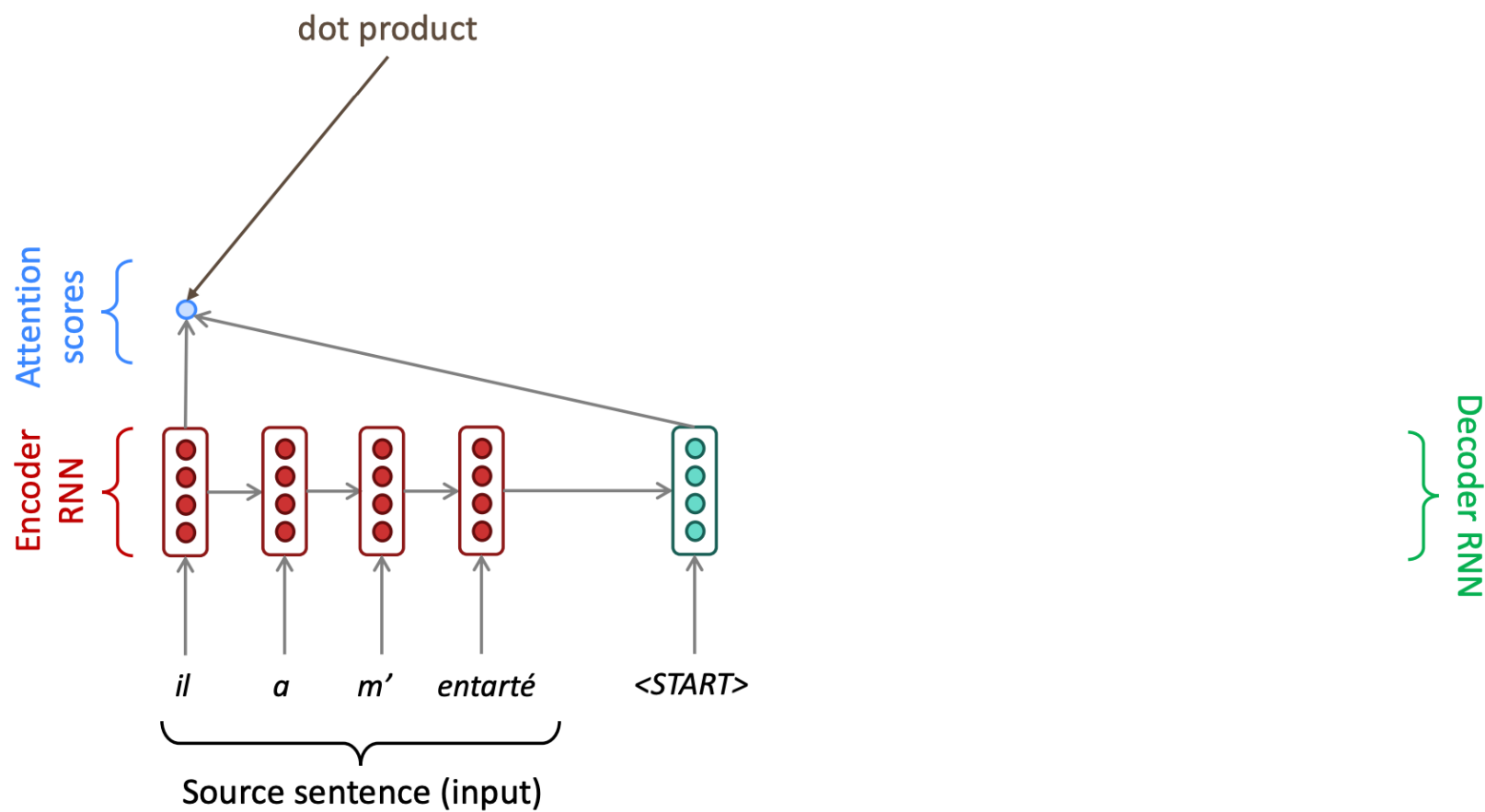
Transformer

Pytorch demo (if time)

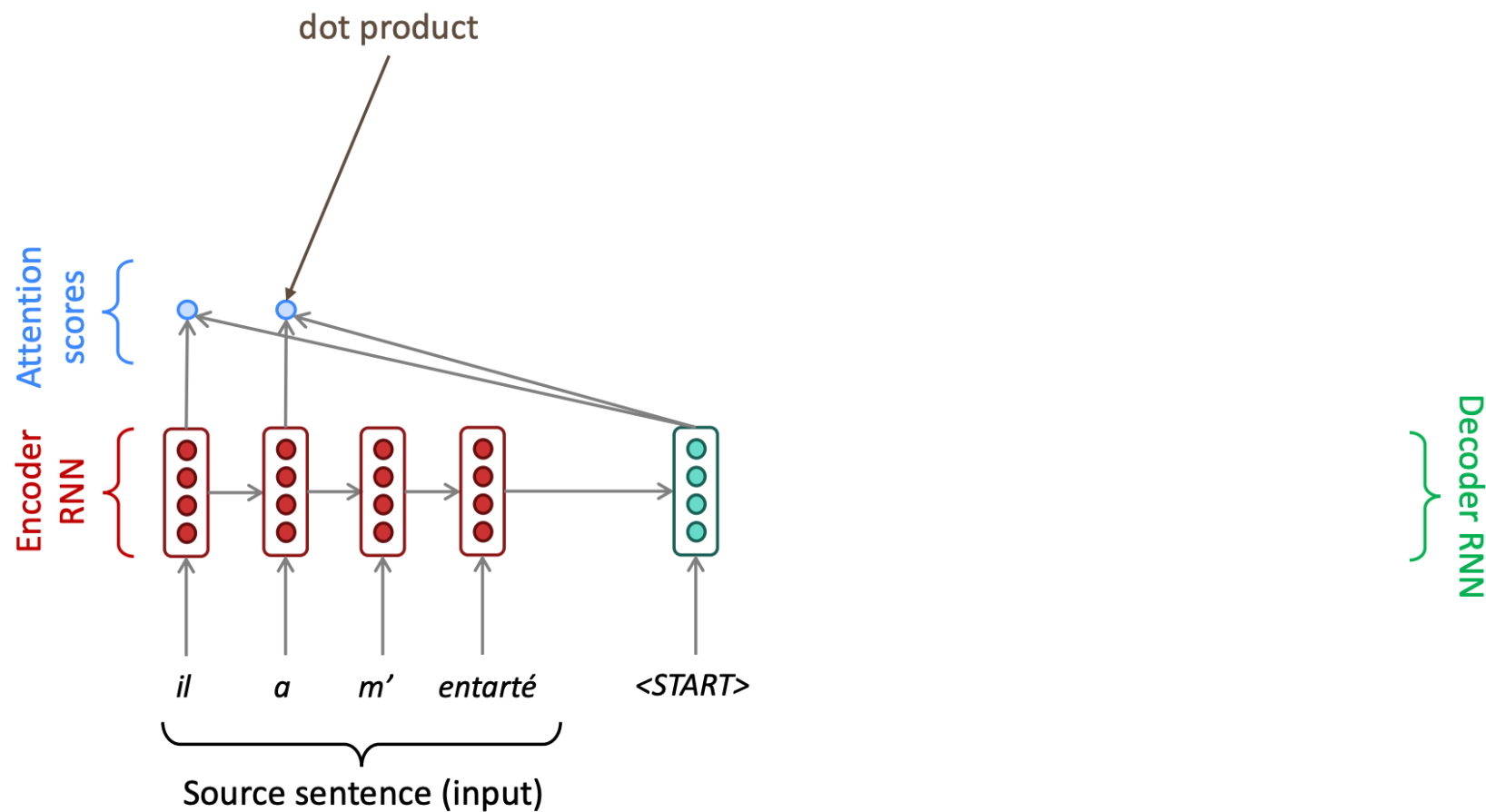
# Seq2Seq Model



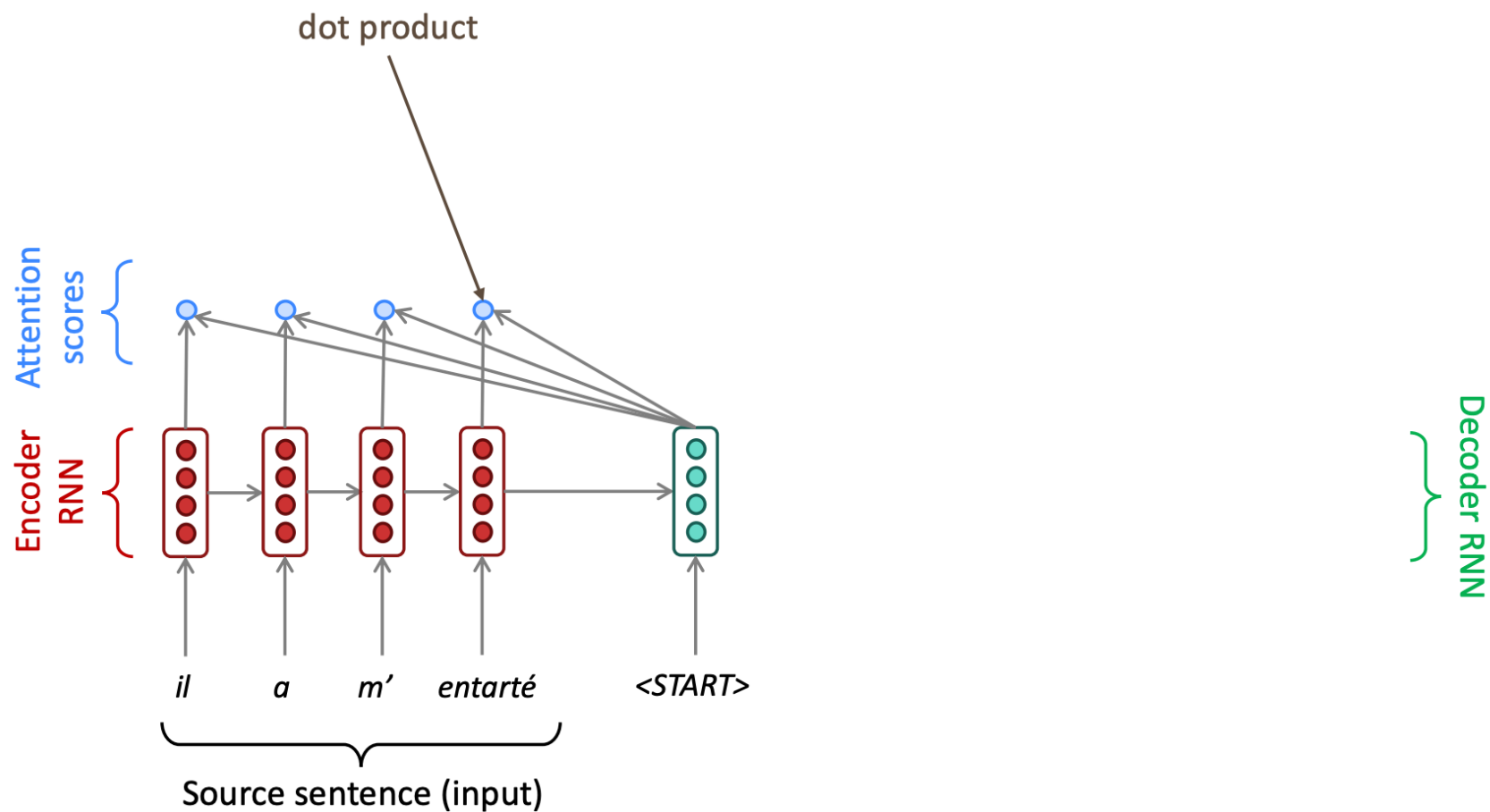
# Seq2Seq w/ Attention



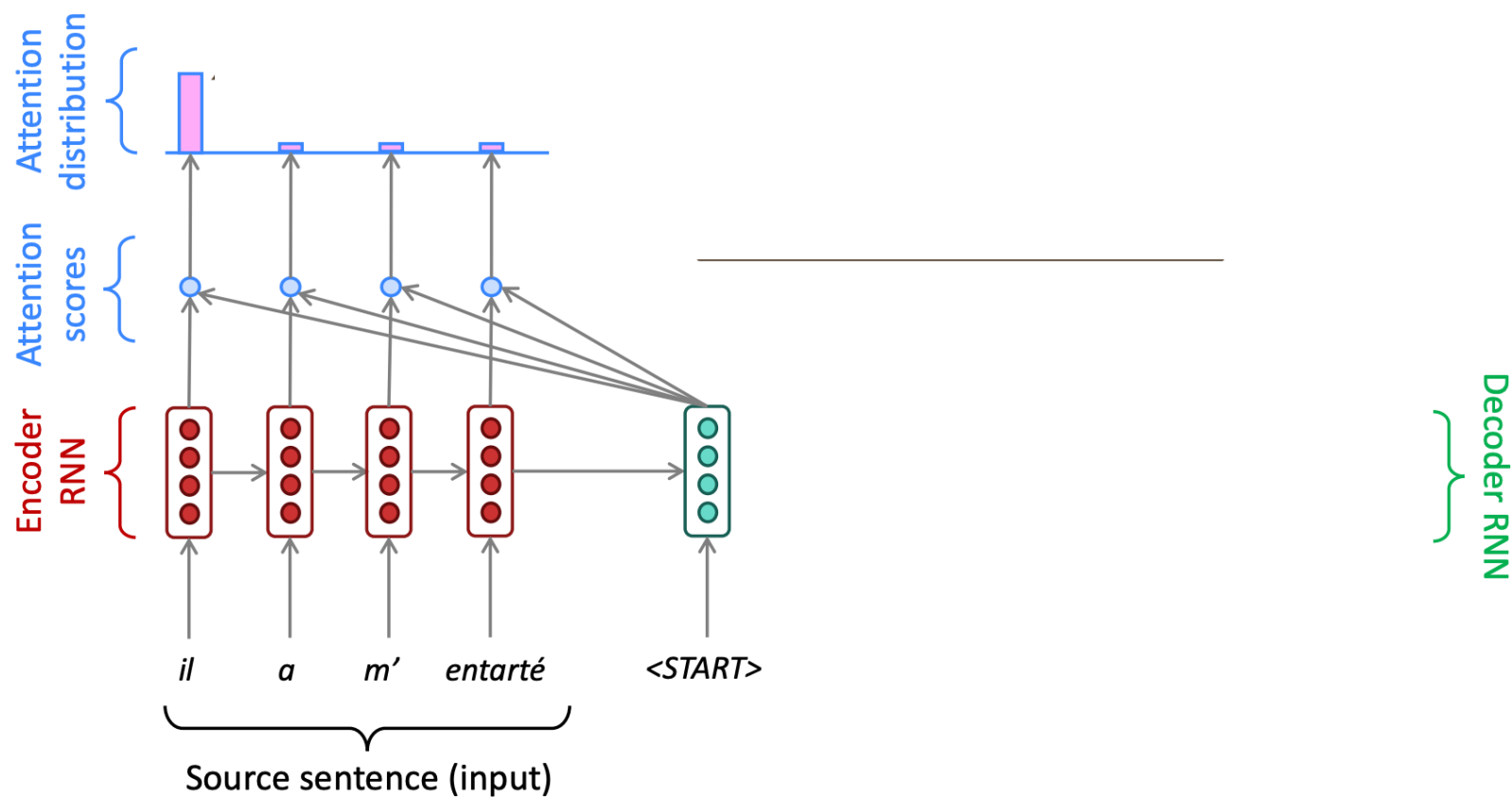
# Seq2Seq w/ Attention



# Seq2Seq w/ Attention

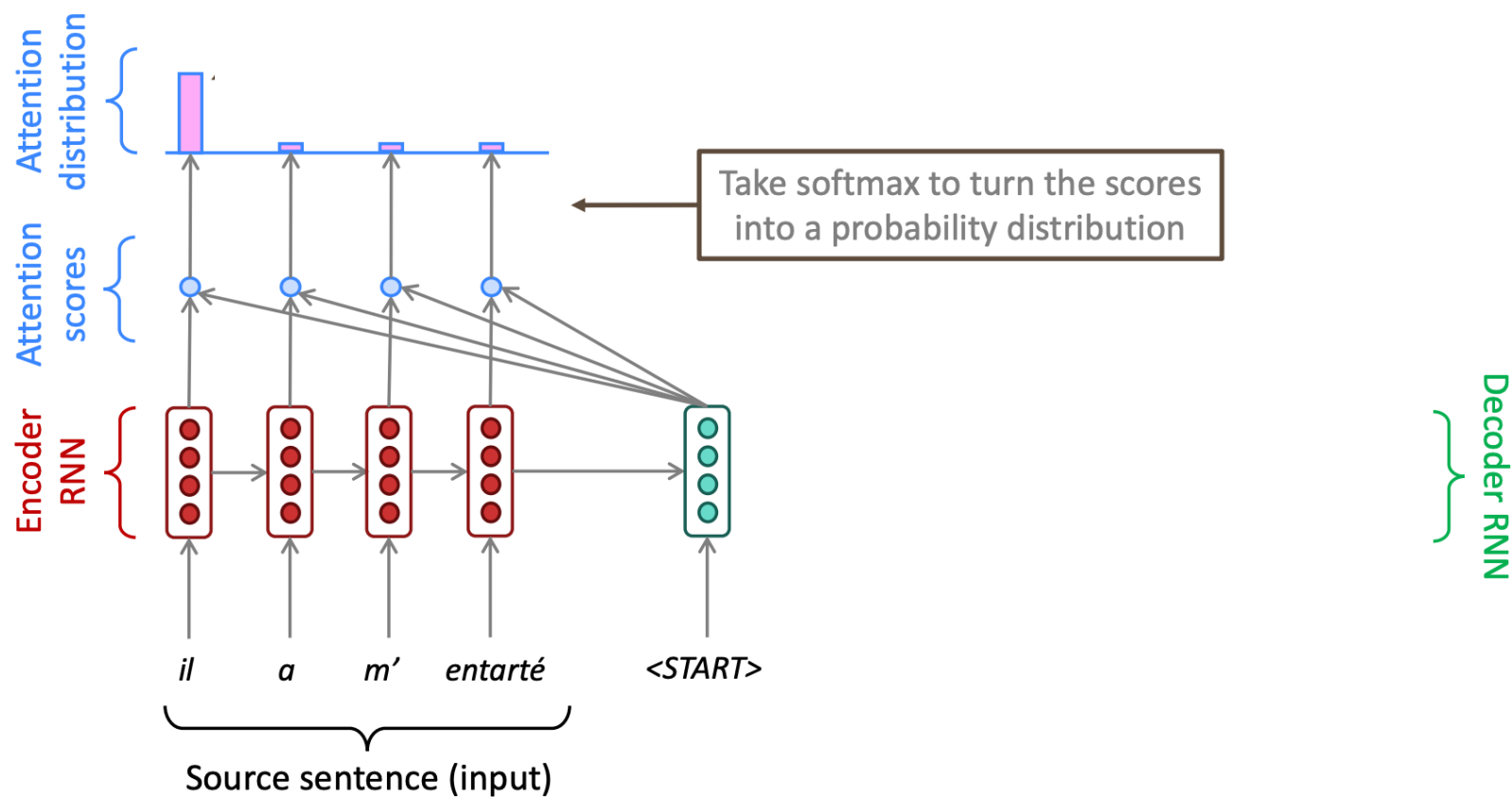


# Seq2Seq w/ Attention

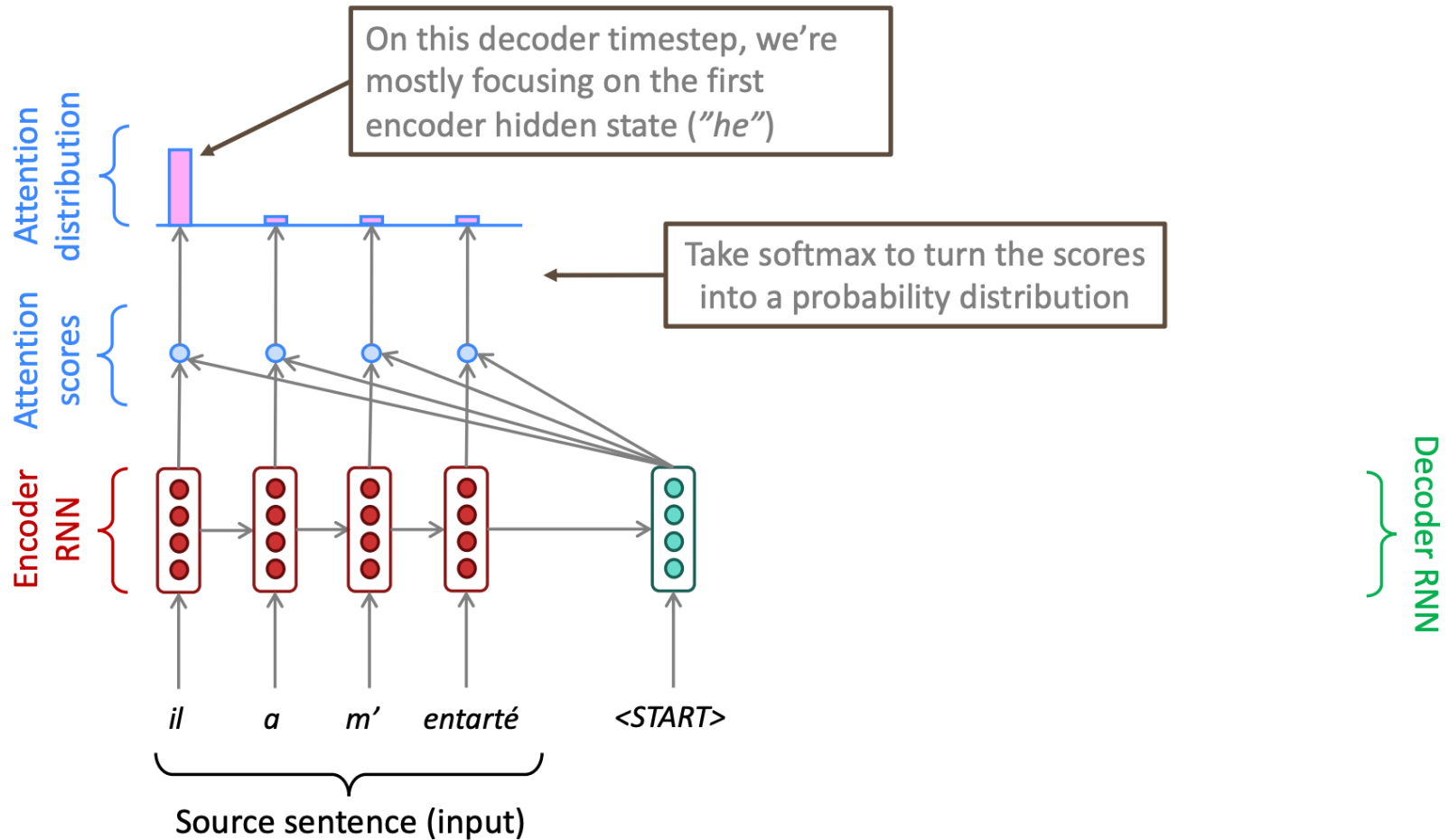




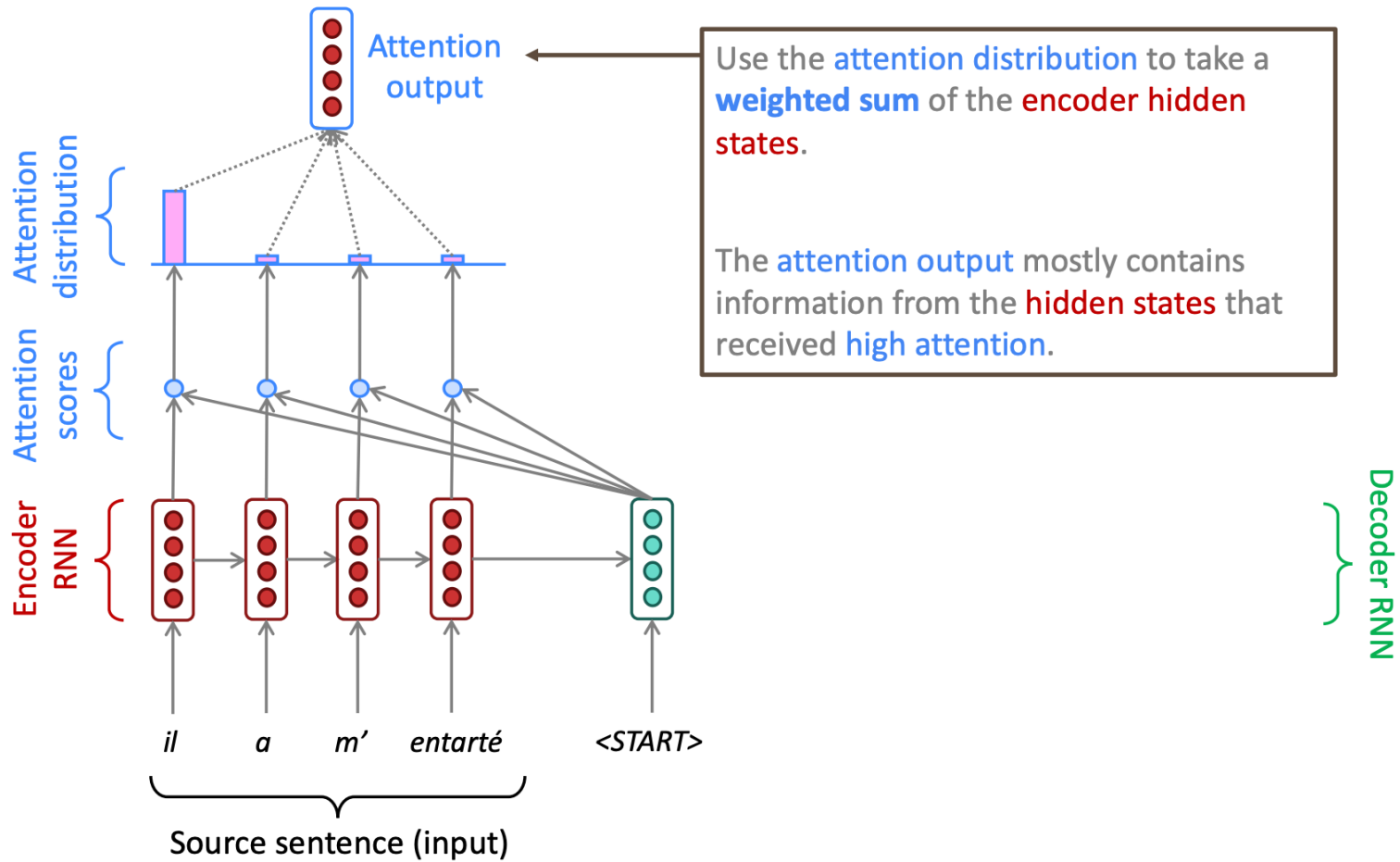
# Seq2Seq w/ Attention



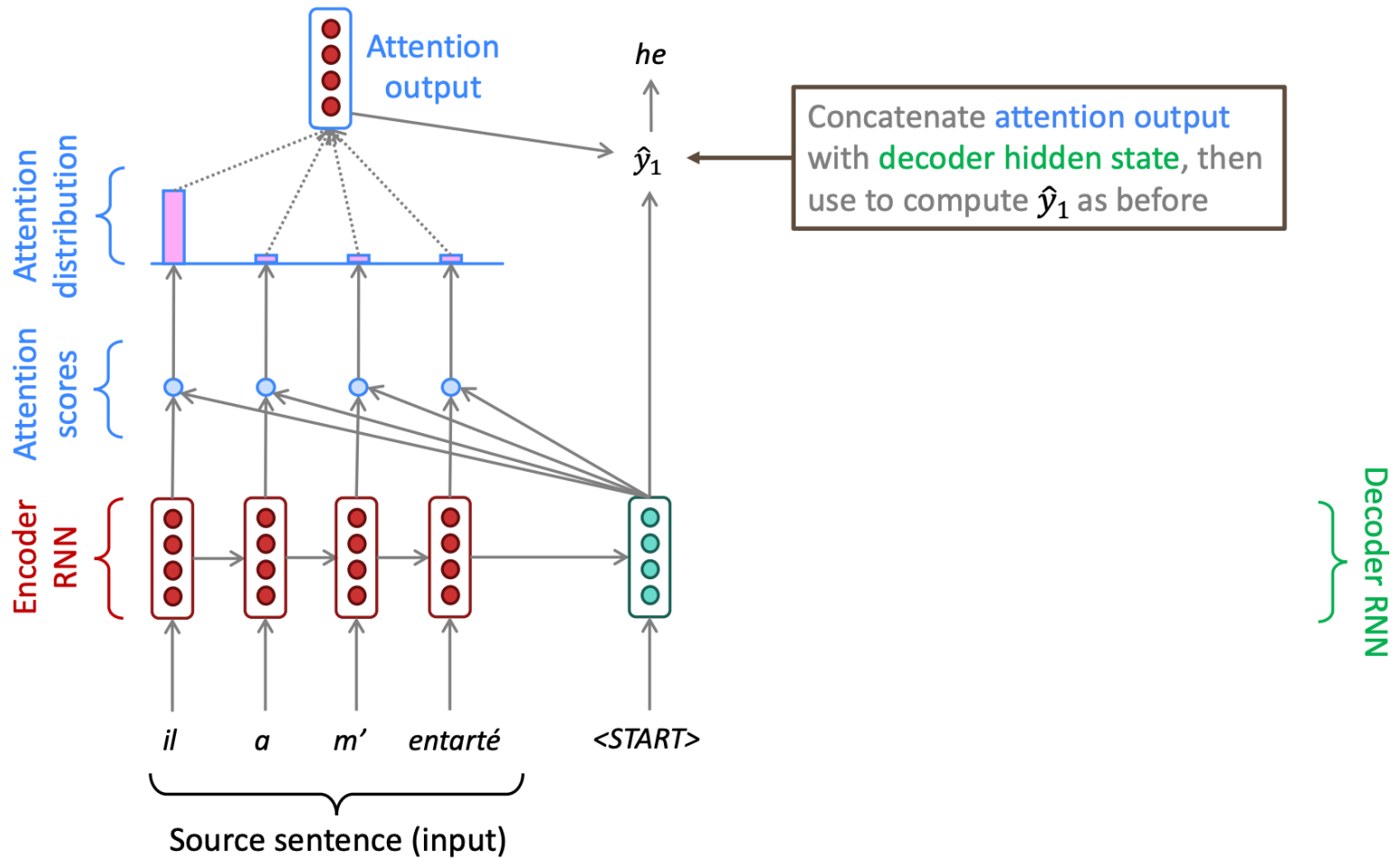
# Seq2Seq w/ Attention



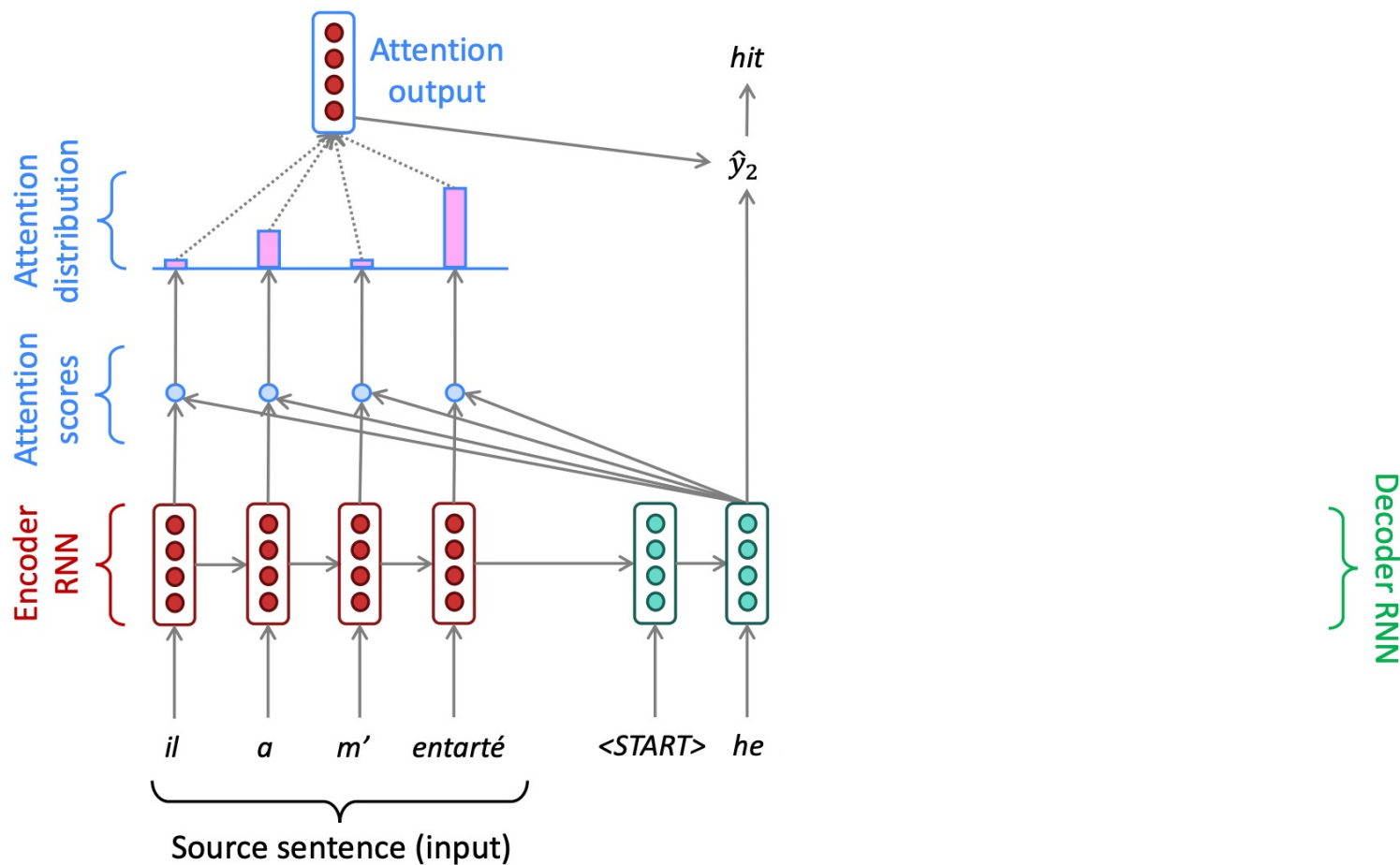
# Seq2Seq w/ Attention



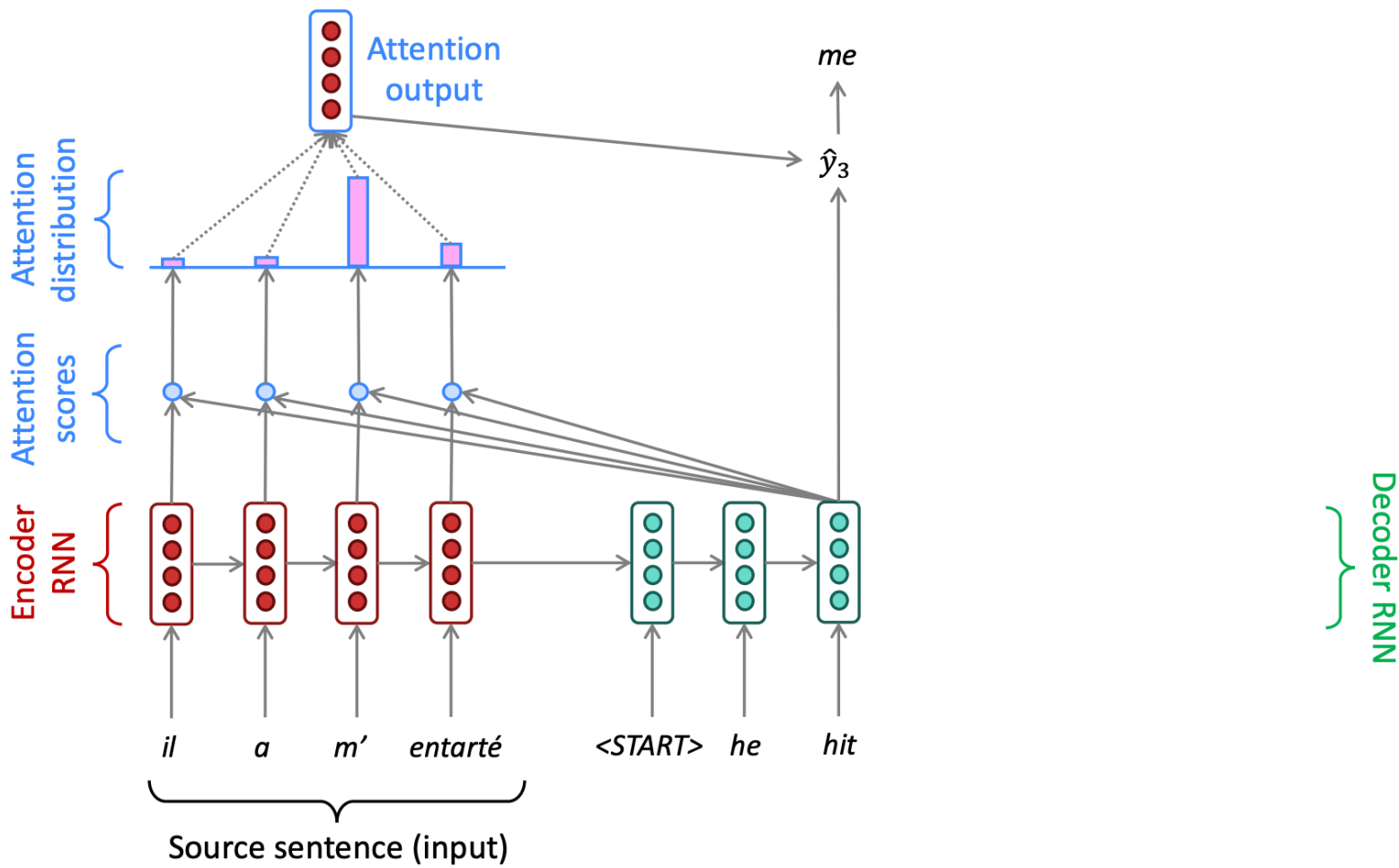
# Seq2Seq w/ Attention



# Seq2Seq w/ Attention



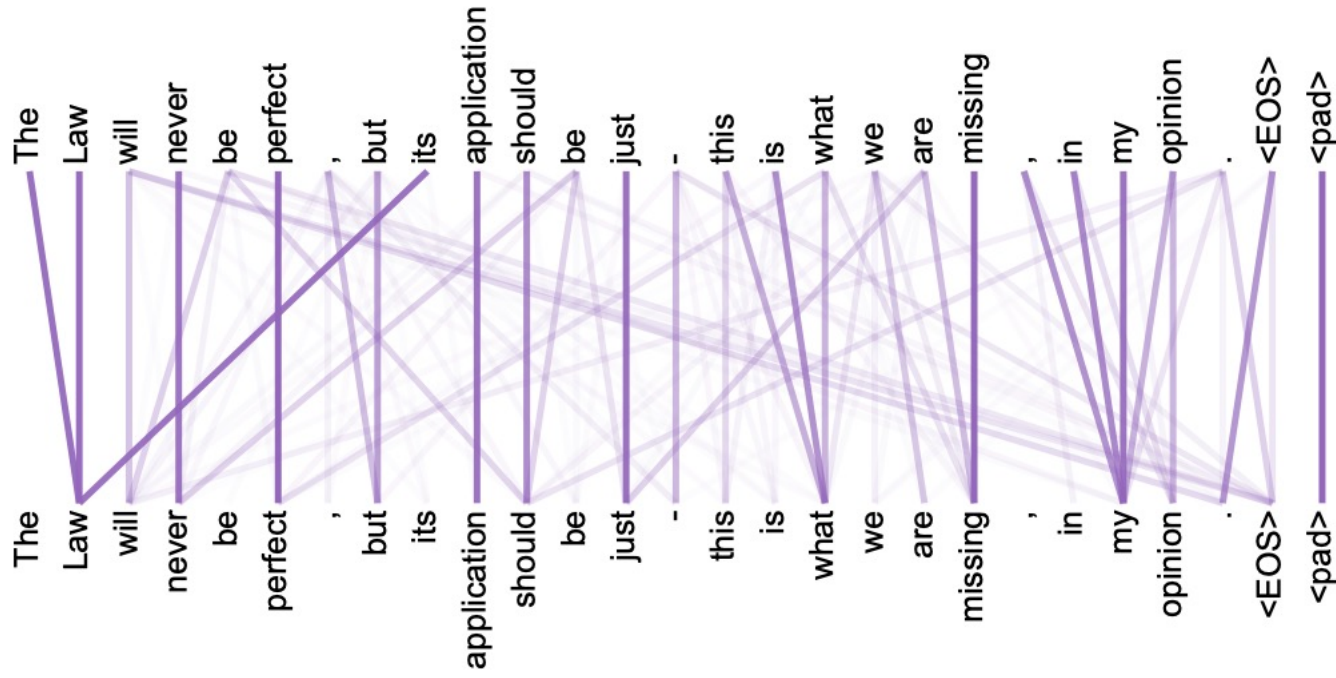
# Seq2Seq w/ Attention



# 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 item 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

Recap – RNNs, Seq2Seq

Attention

**Self-attention**

Transformer

Pytorch demo (if time)

# Self-attention in a nutshell

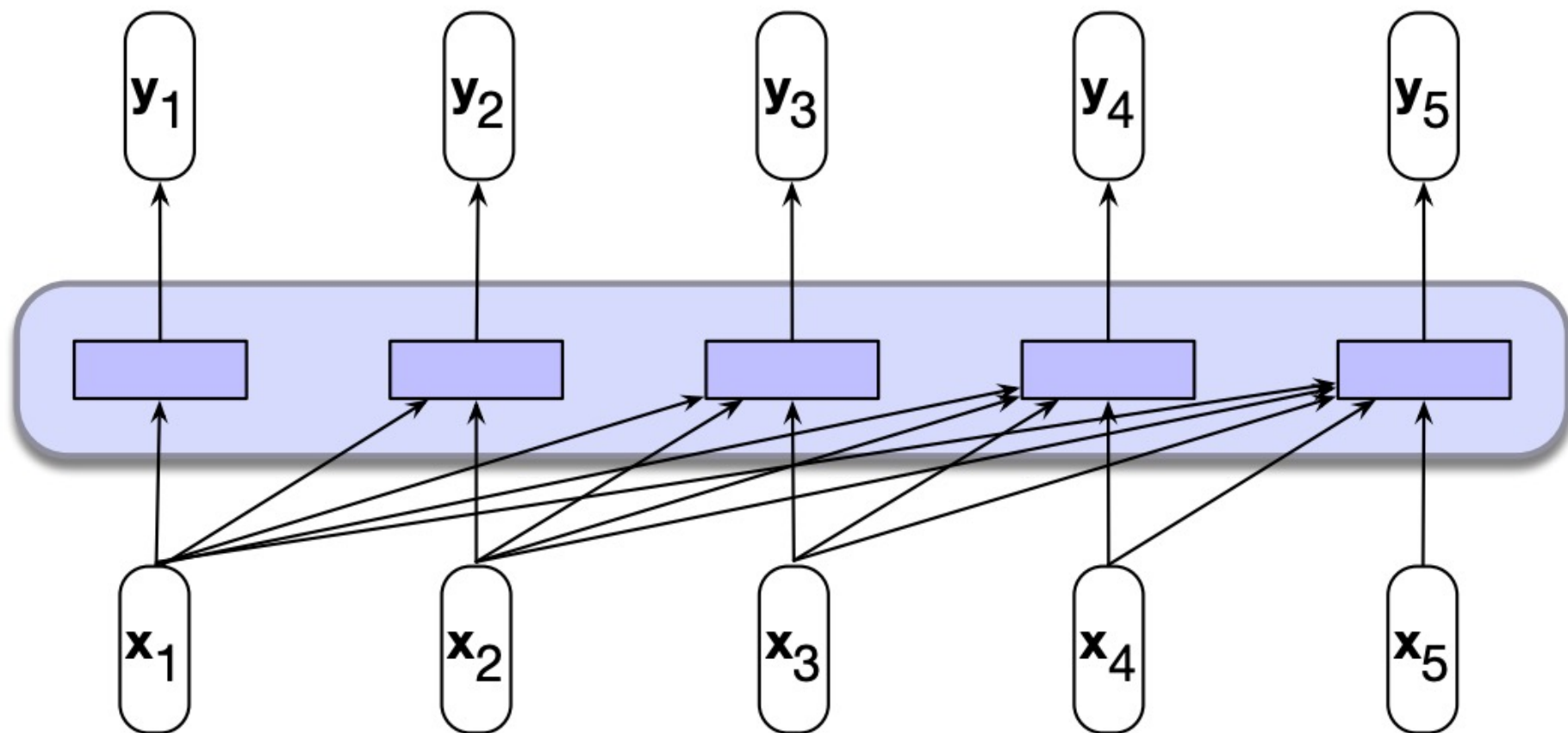
Attention:

- 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:

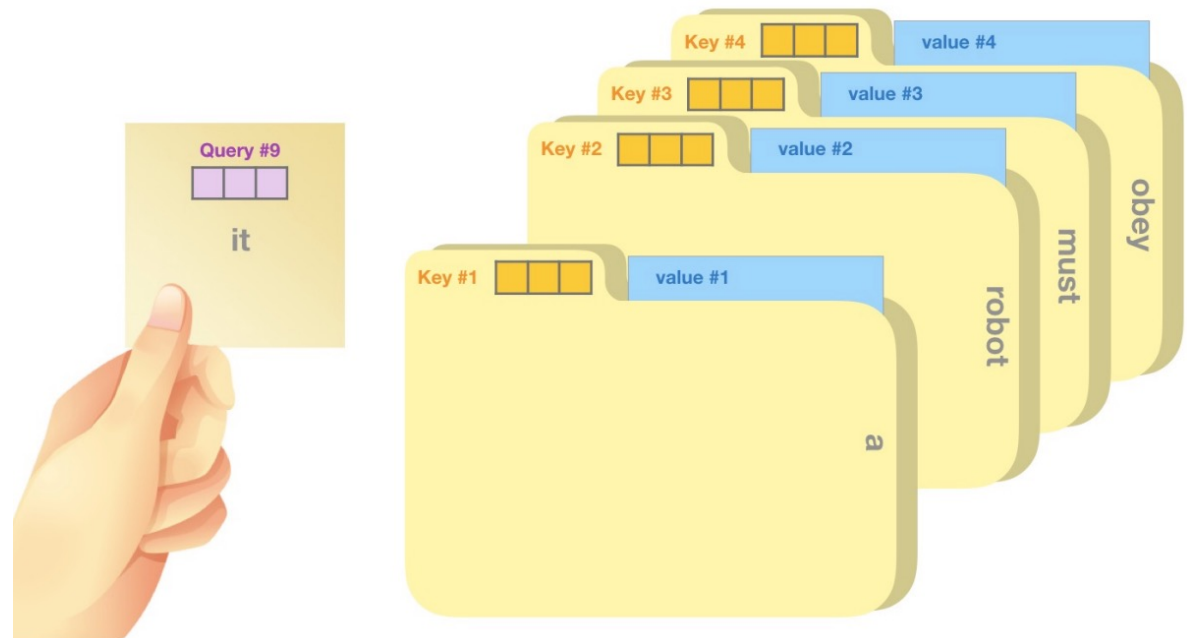
Value:

# Terminology

Query: what to match

Key: the thing to match

Value: what to be extracted from the match

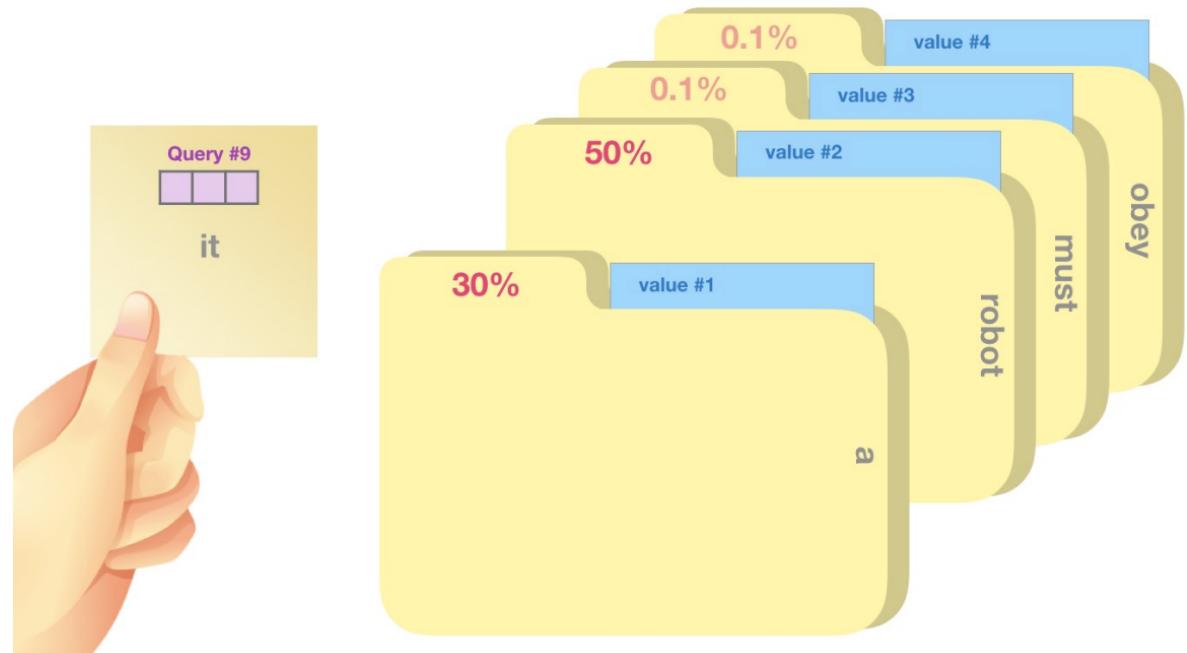


# Terminology

**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:

$$k_i = W^k x_i$$

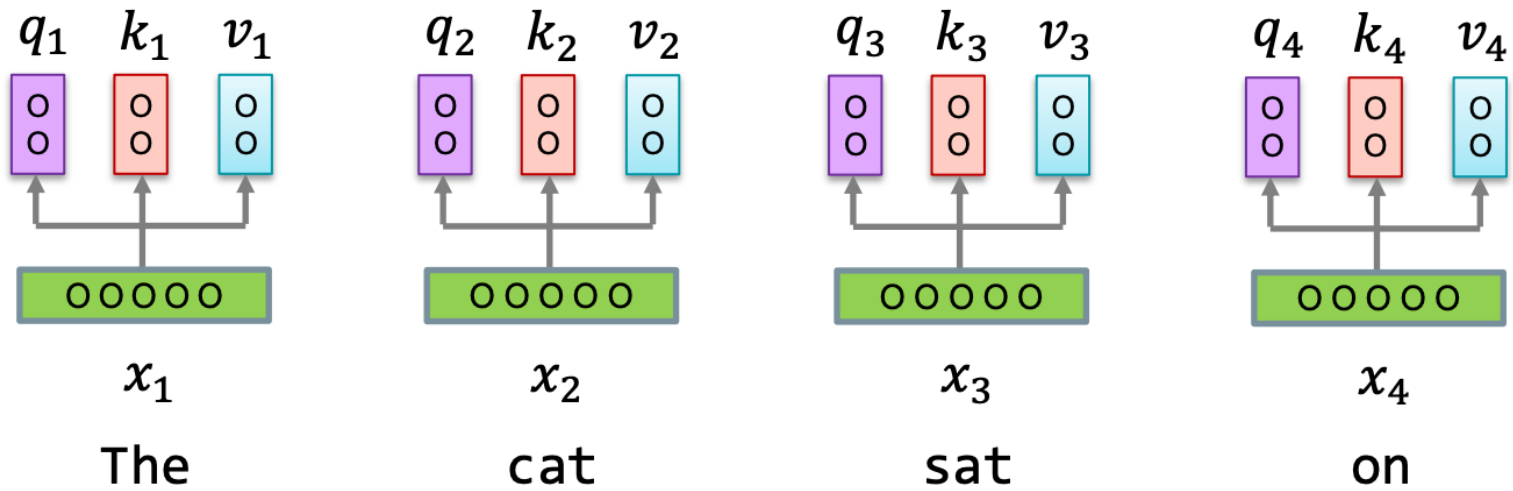
**Value:** what to be extracted from the match

$$v_i = W^v x_i$$

# Output of each input cell

These are three representations of each input

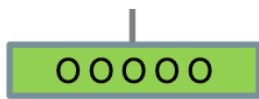
Each representation is created by multiplying the input by a weight matrix



# Self-Attention Scores

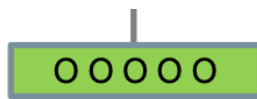
When creating a representation for  $x_i$ , how much weight/focus/attention should we give to  $x_j$

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



$x_1$

The



$x_2$

cat



$x_3$

sat



$x_4$

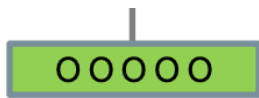
on

# 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  $score(x_i, x_j)$



$x_1$

The



$x_2$

cat



$x_3$

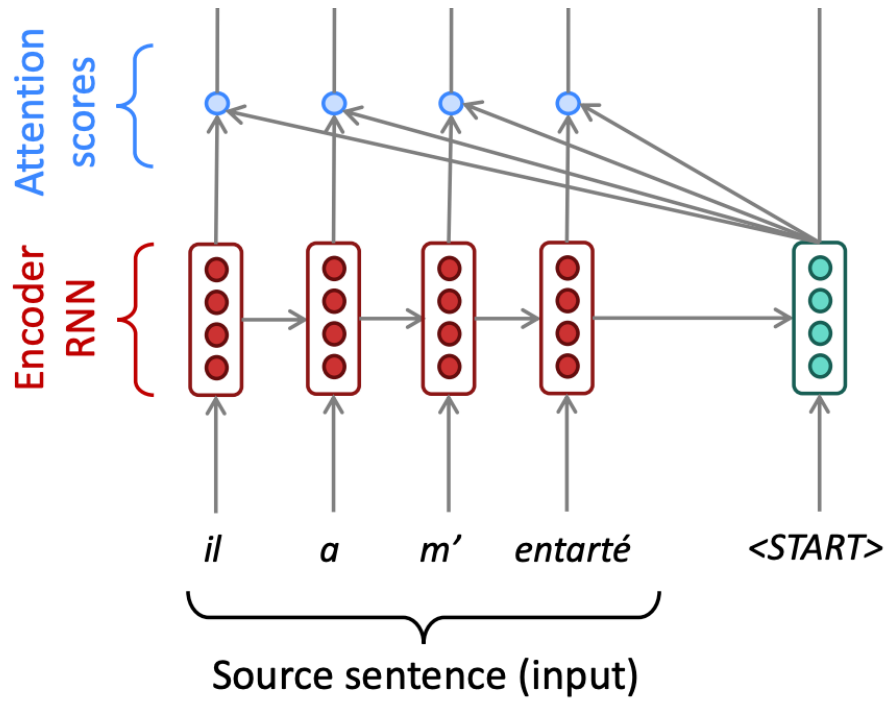
sat



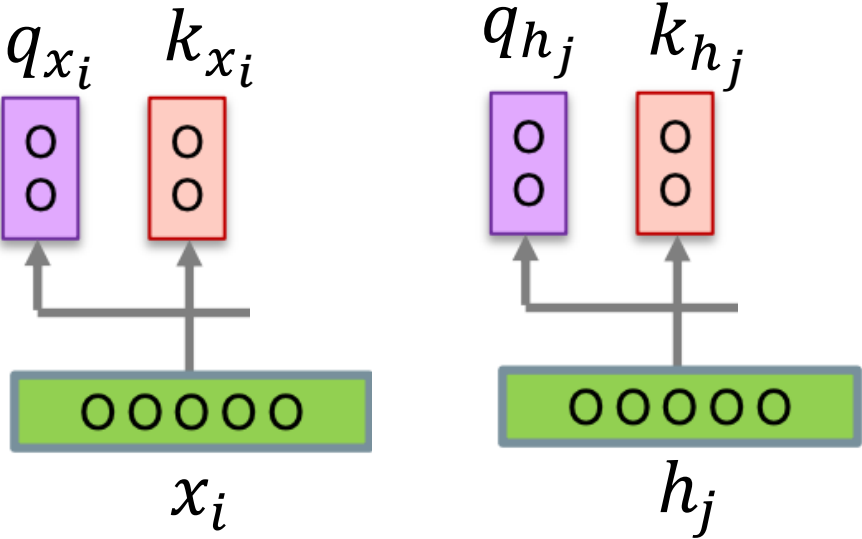
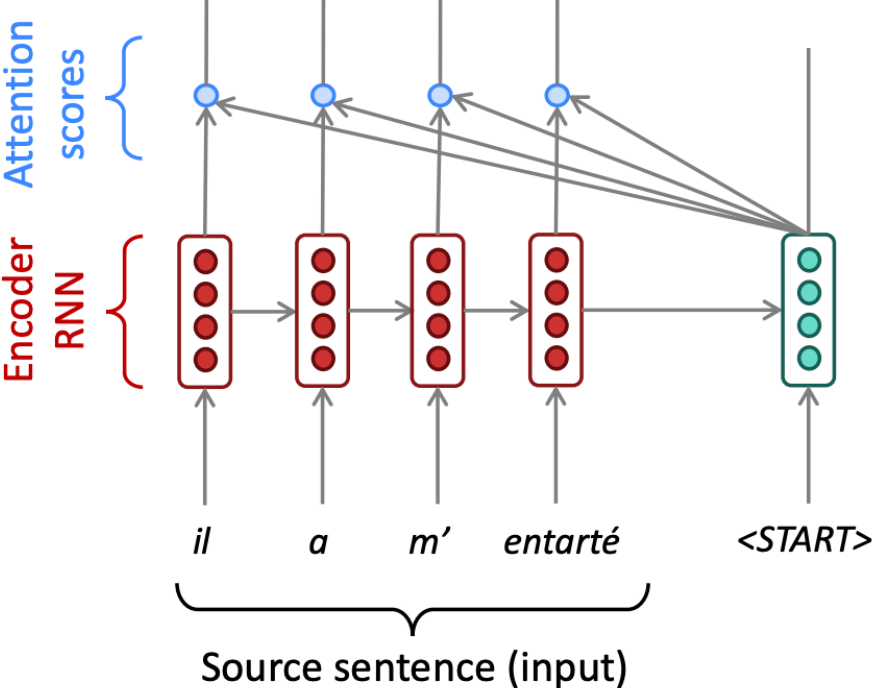
$x_4$

on

# Attention score

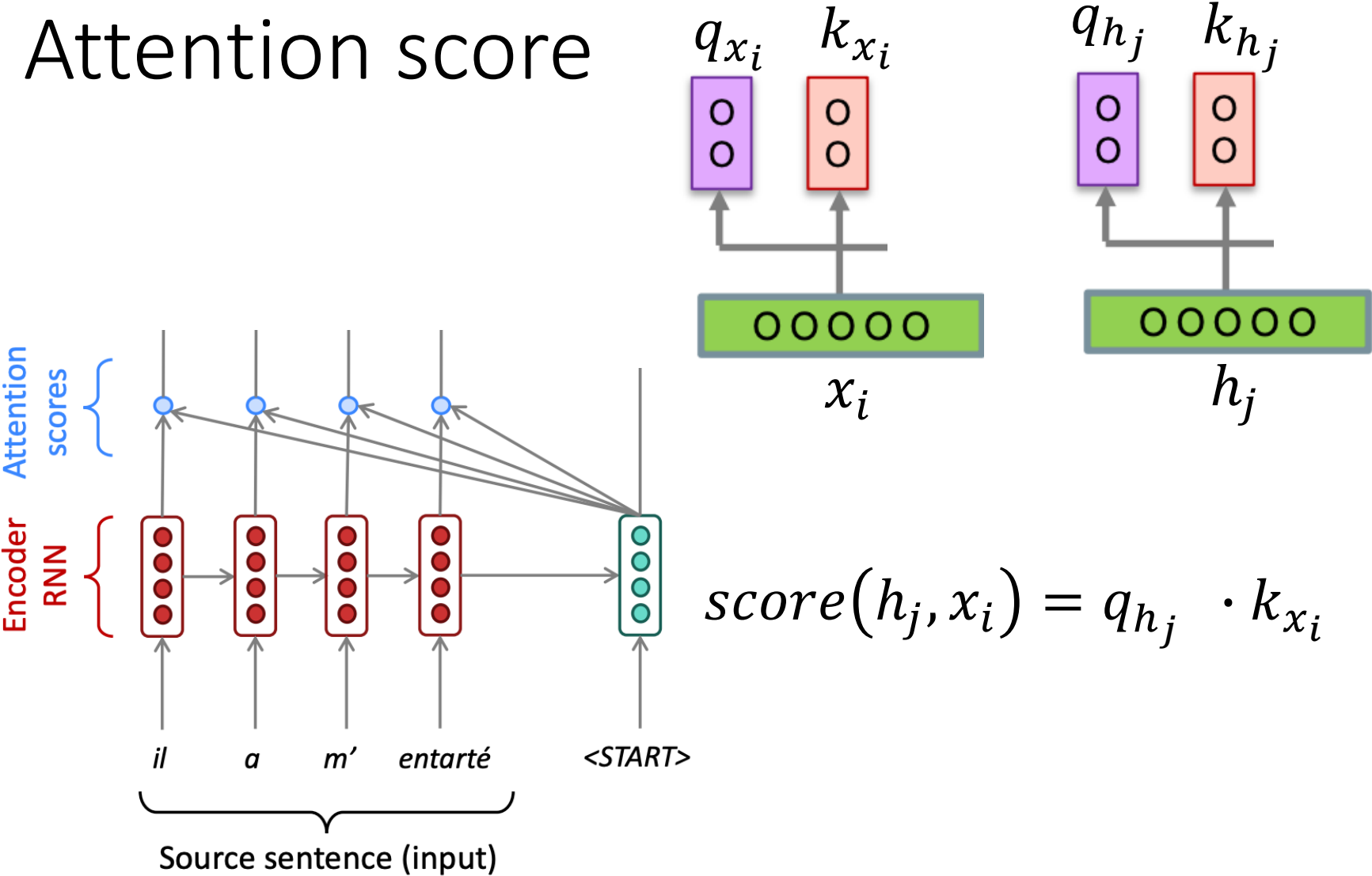


# Attention score



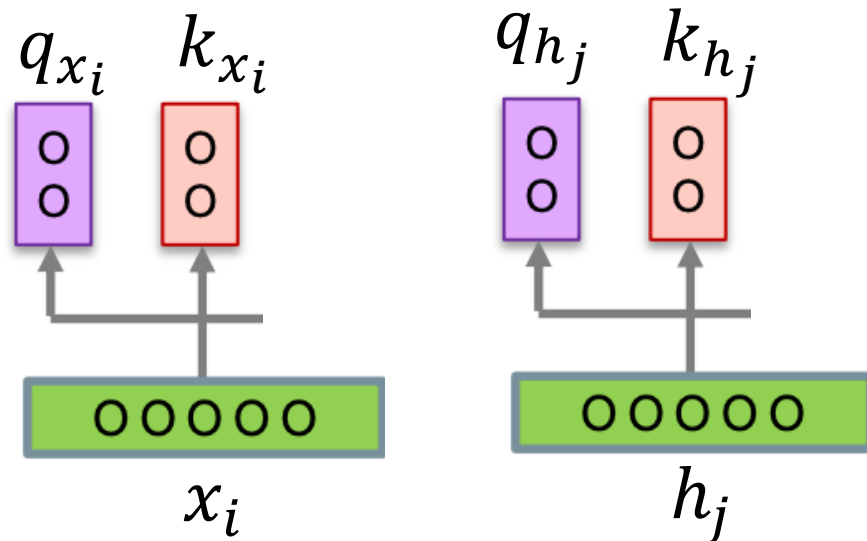
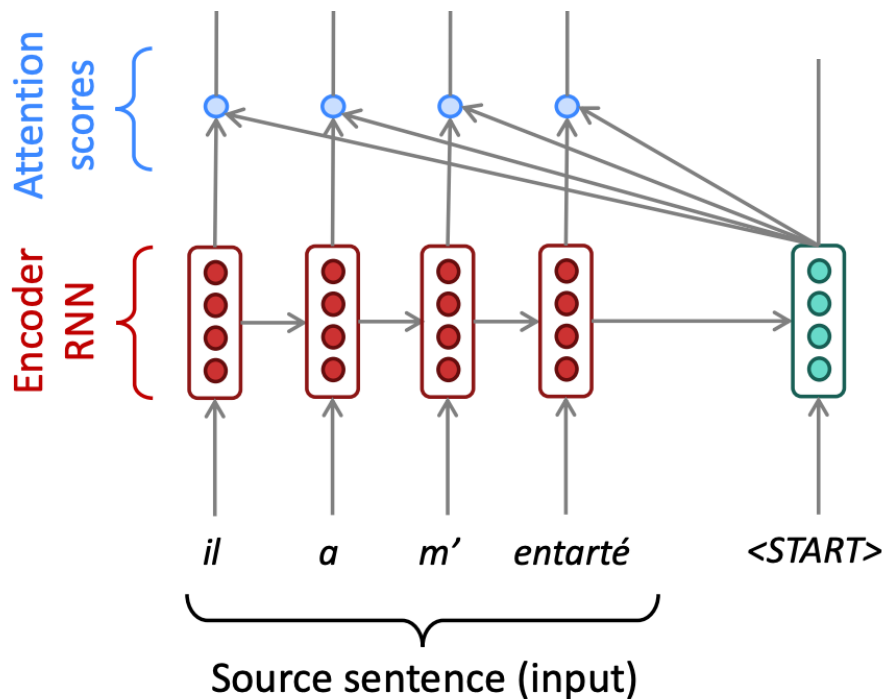
$$score(h_j, x_i) = ?$$

# Attention score





# Attention score



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

# Attention Scores

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

$$\begin{aligned} A_{score} &= \begin{bmatrix} \alpha_{1,1} & \cdots & \alpha_{1,n} \\ \vdots & \ddots & \vdots \\ \alpha_{m,1} & \cdots & \alpha_{m,n} \end{bmatrix} \\ &= \begin{bmatrix} \text{score}(h_1, x_1) & \cdots & \text{score}(h_1, x_n) \\ \vdots & \ddots & \vdots \\ \text{score}(h_m, x_1) & \cdots & \text{score}(h_m, x_n) \end{bmatrix} \\ &= \begin{bmatrix} \frac{q_{h_1} \cdot k_{x_1}}{\sqrt{d_k}} & \cdots & \frac{q_{h_1} \cdot k_{x_n}}{\sqrt{d_k}} \\ \vdots & \ddots & \vdots \\ \frac{q_{h_m} \cdot k_{x_1}}{\sqrt{d_k}} & \cdots & \frac{q_{h_m} \cdot k_{x_n}}{\sqrt{d_k}} \end{bmatrix} \end{aligned}$$

# Attention Scores

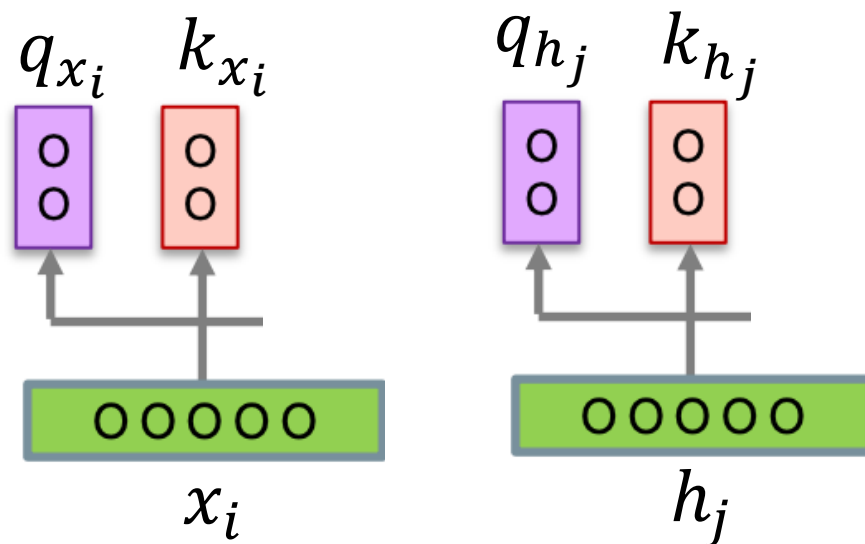
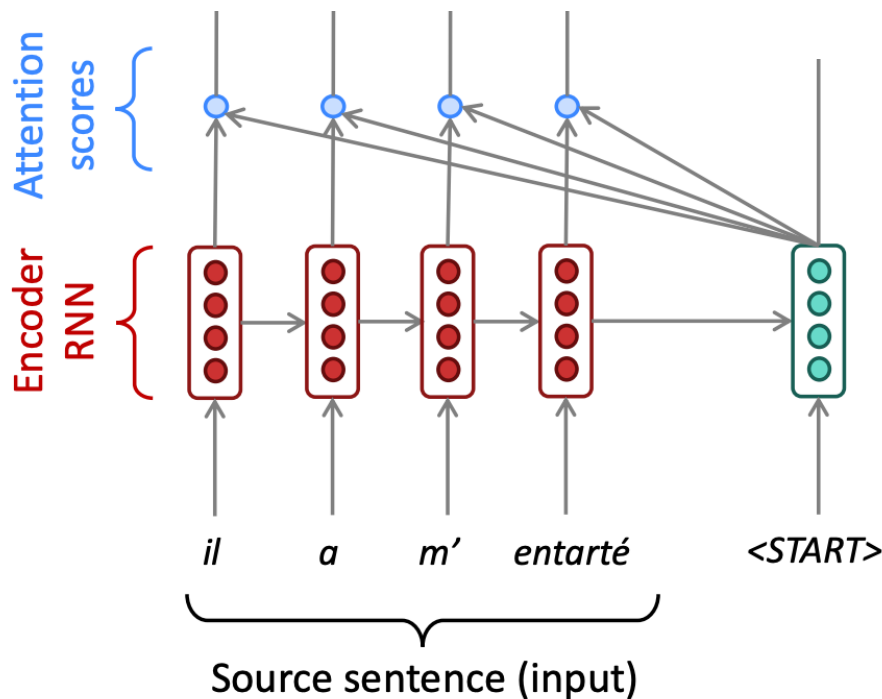
We can store all the q's  
and k's in a matrix as well

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

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

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

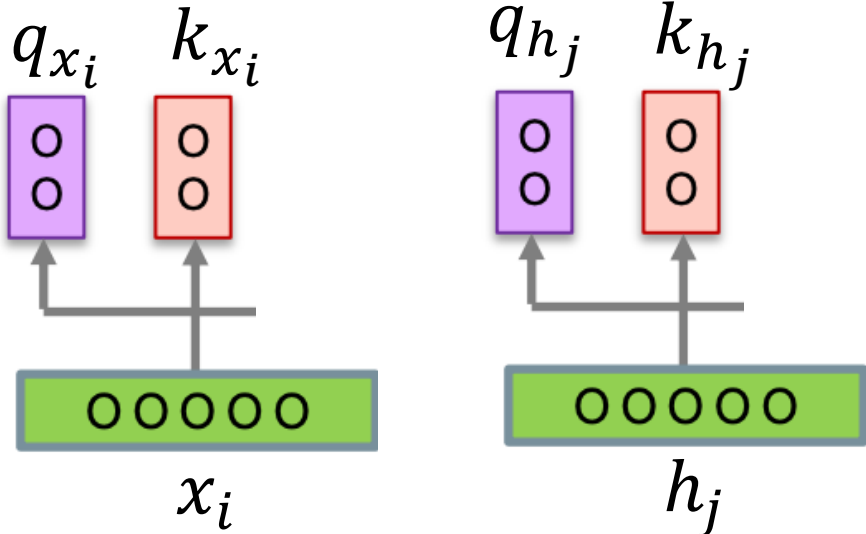
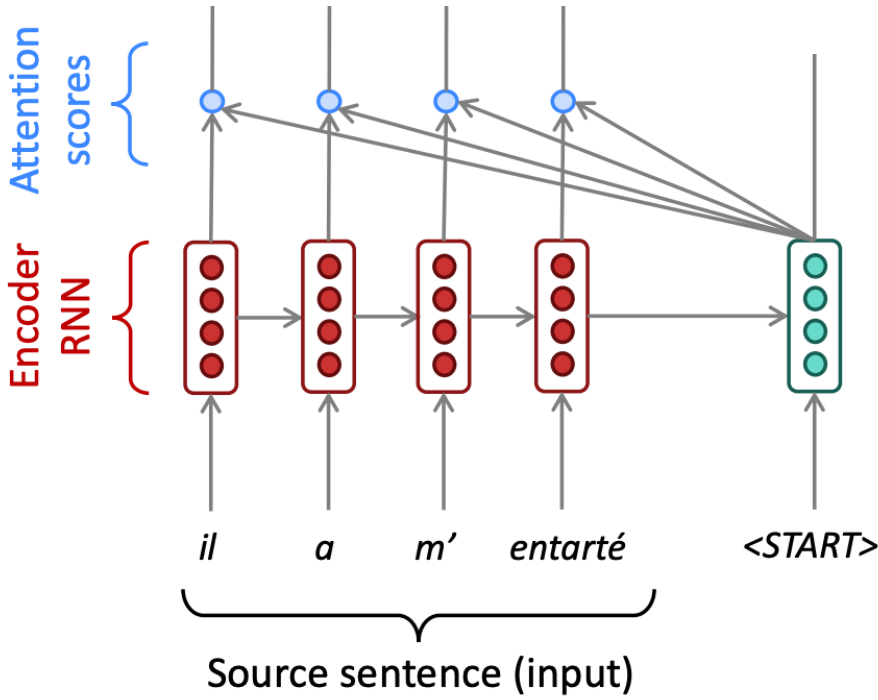
# Attention score



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

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

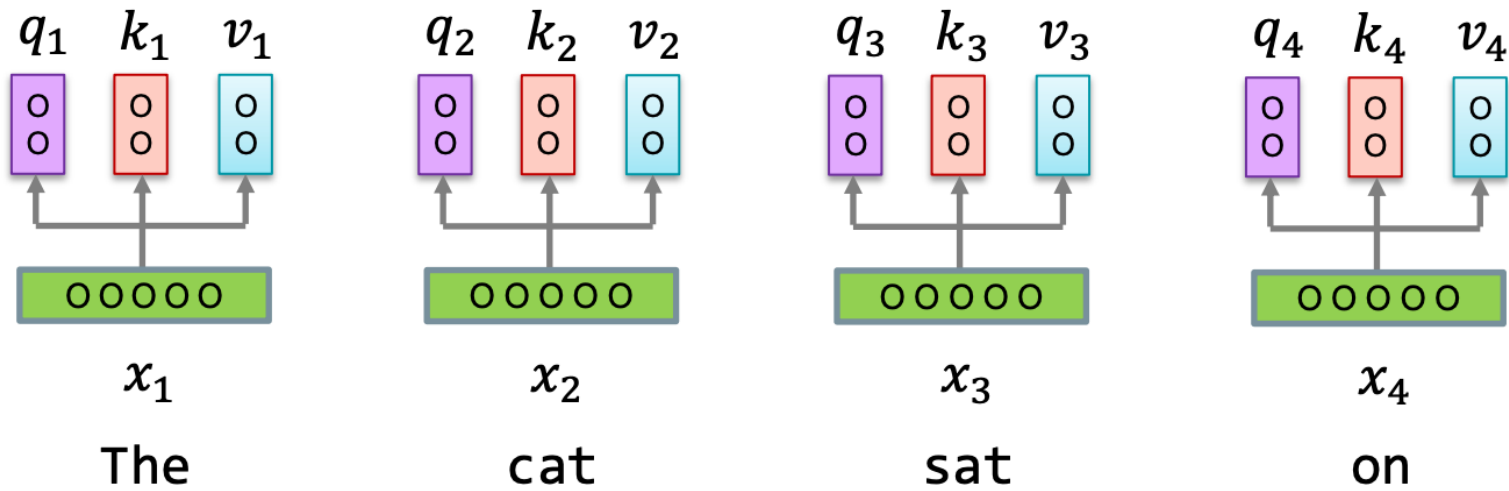
# Attention



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

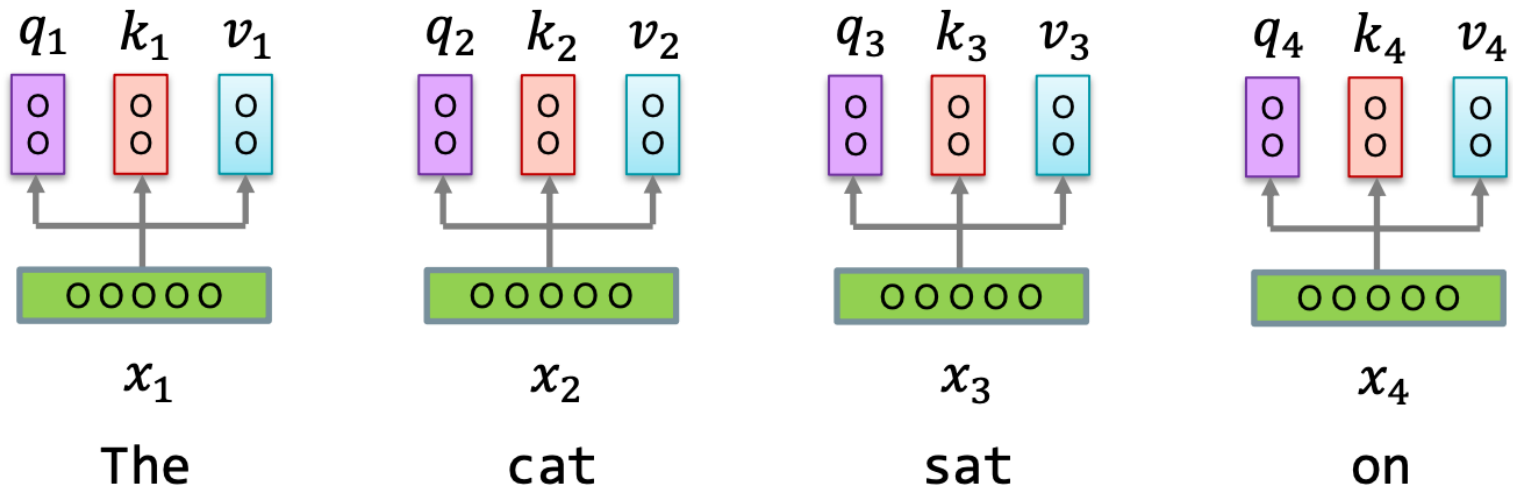
# Self-attention

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



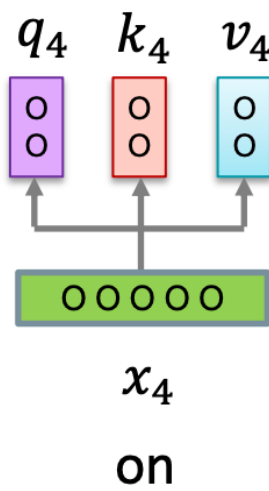
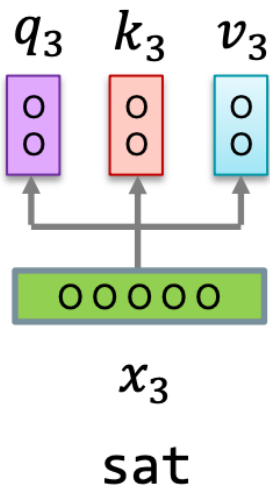
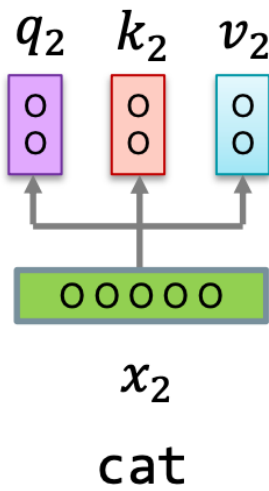
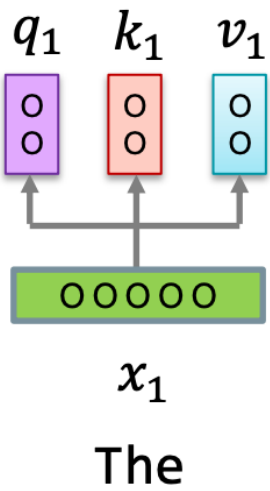
# Self-attention

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



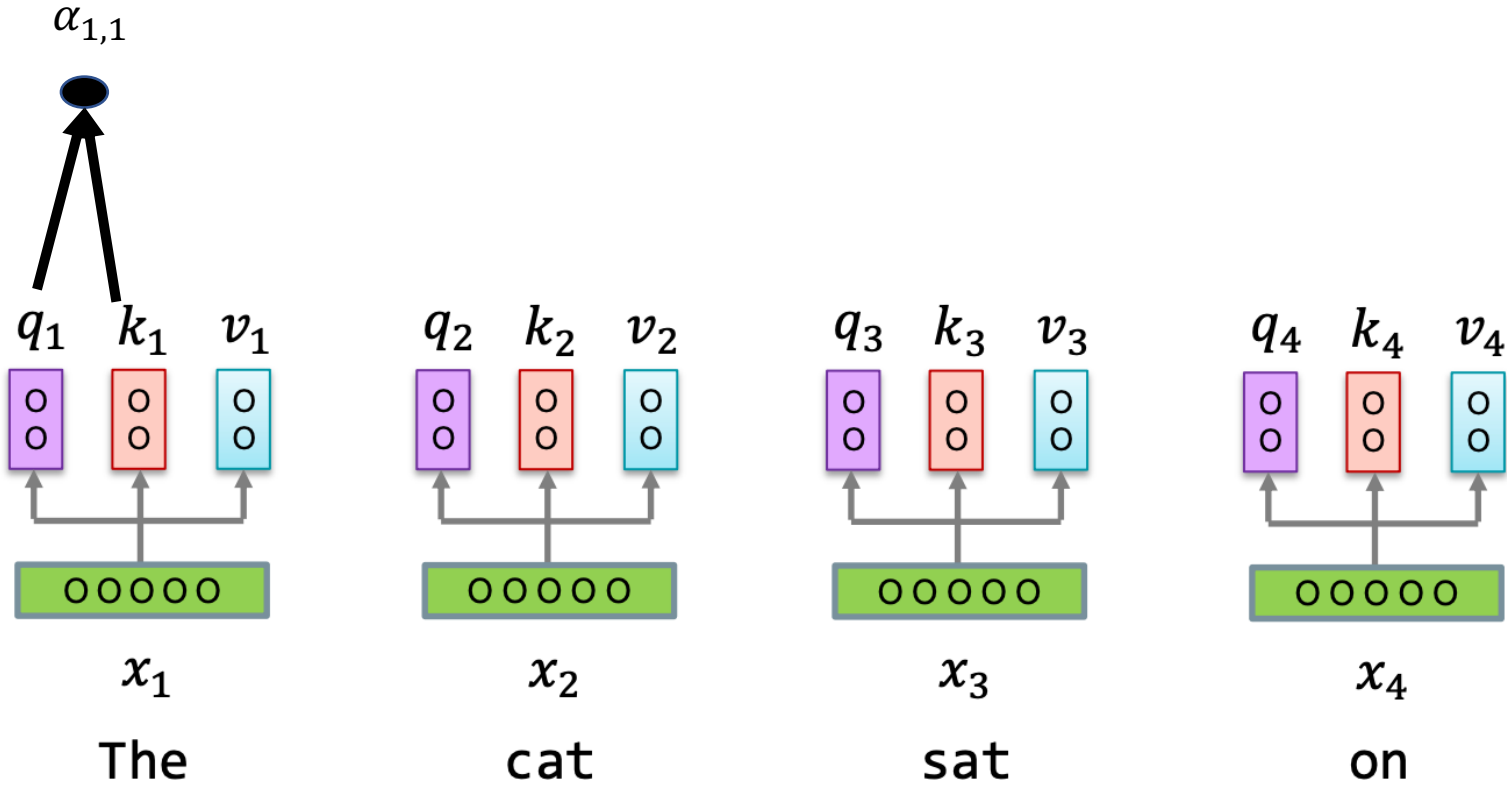
# Self-attention

$$\alpha_{1,1}$$

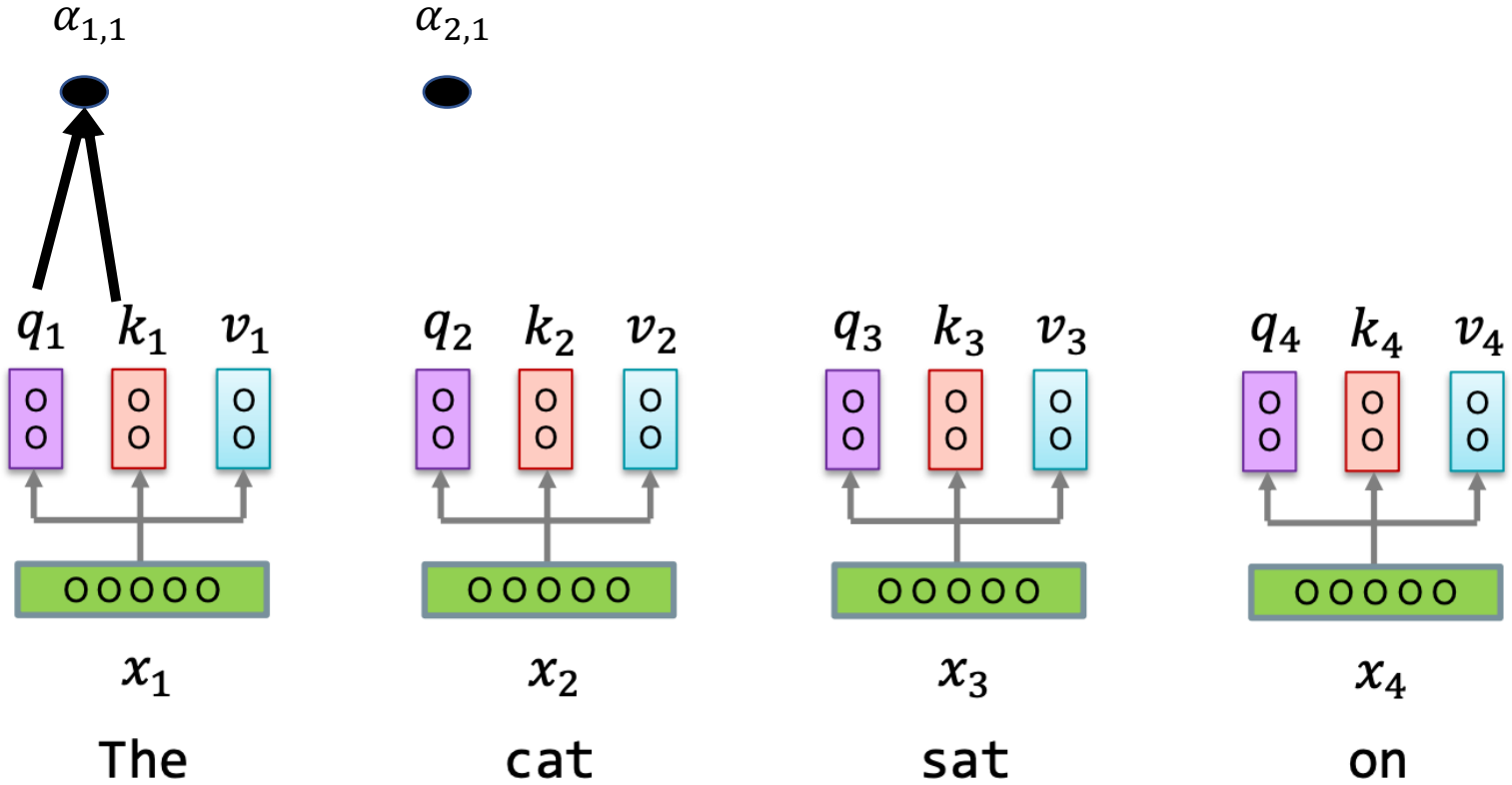





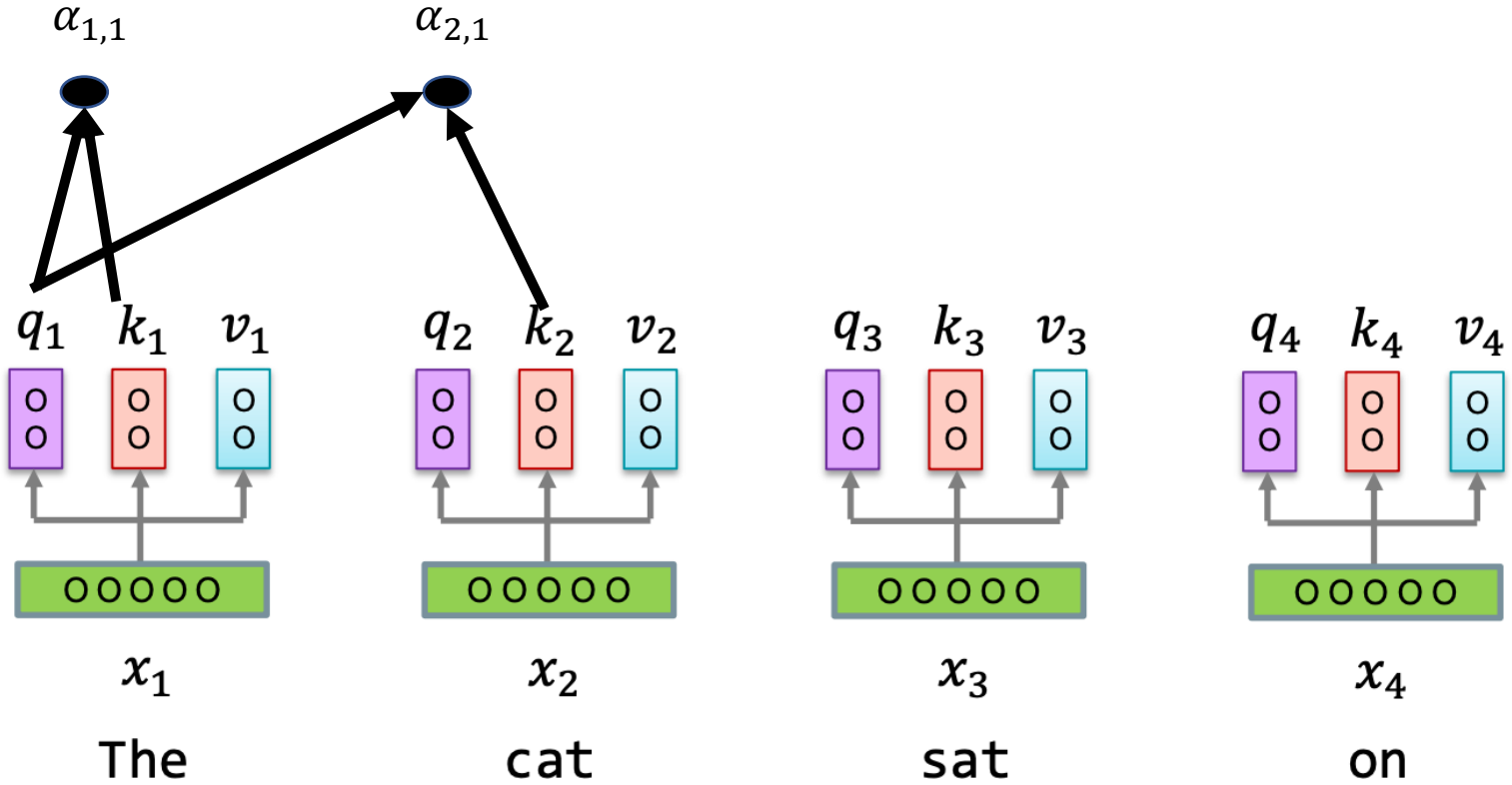
# Self-attention



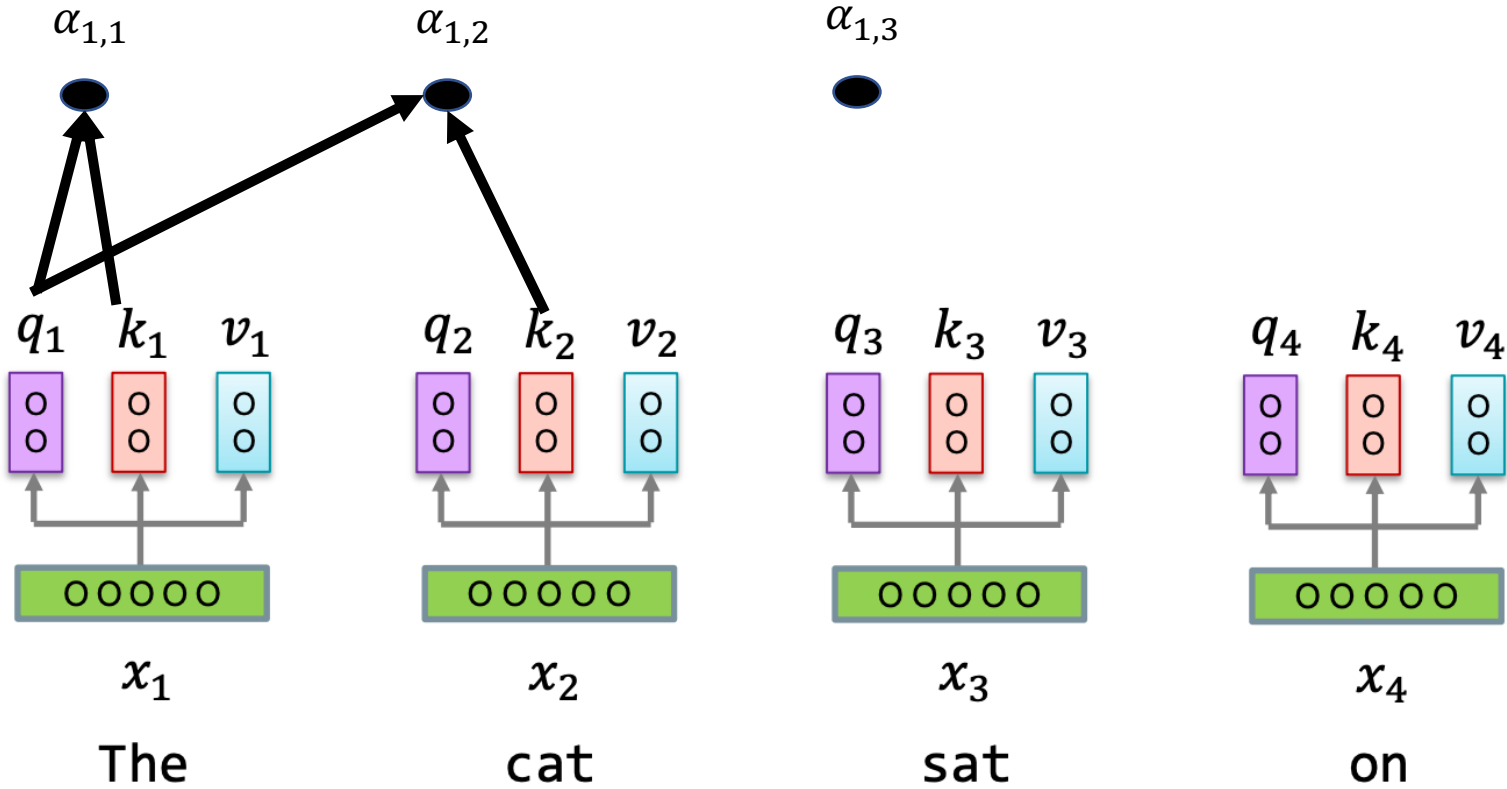
# Self-attention



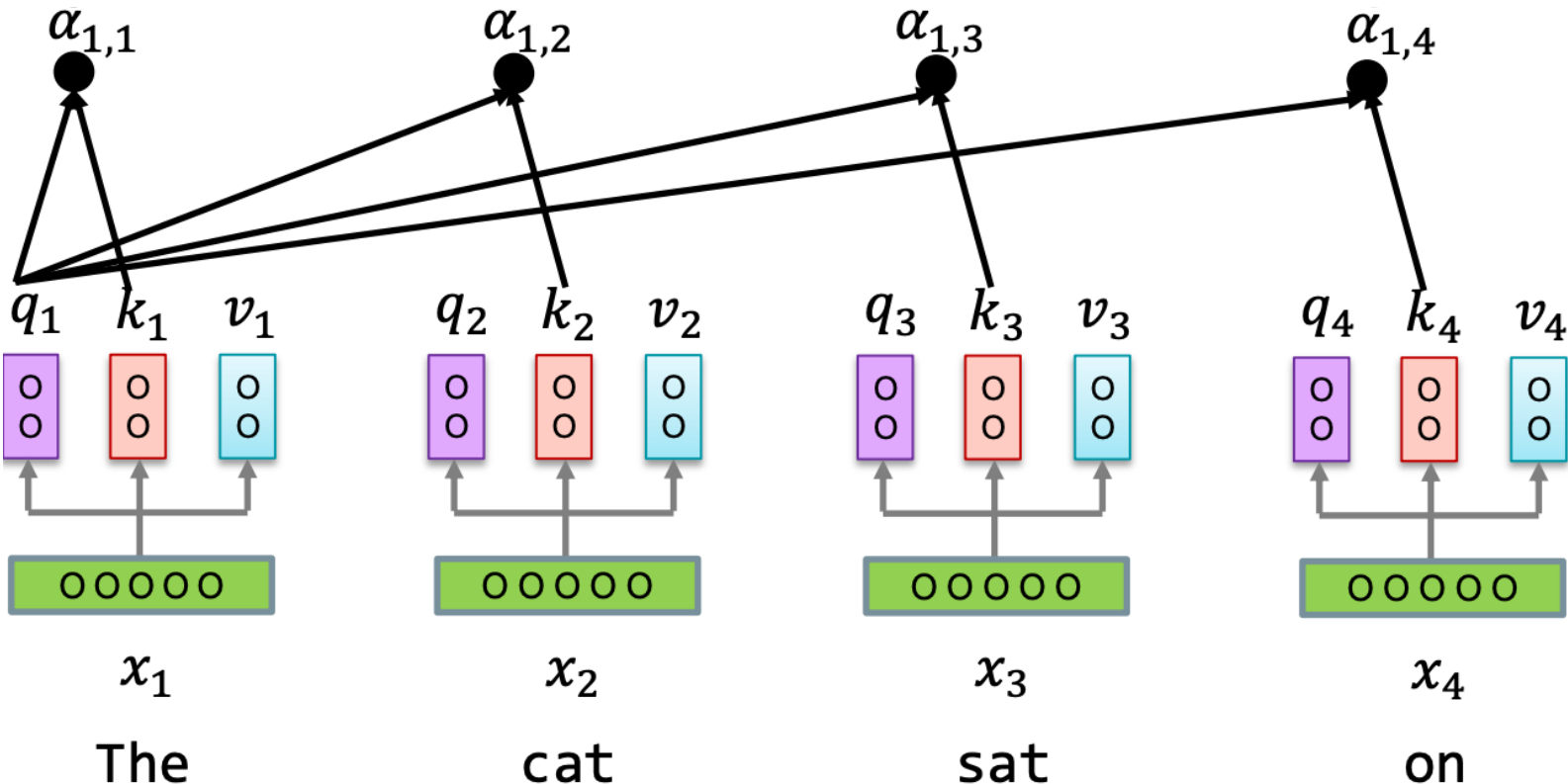
# Self-attention



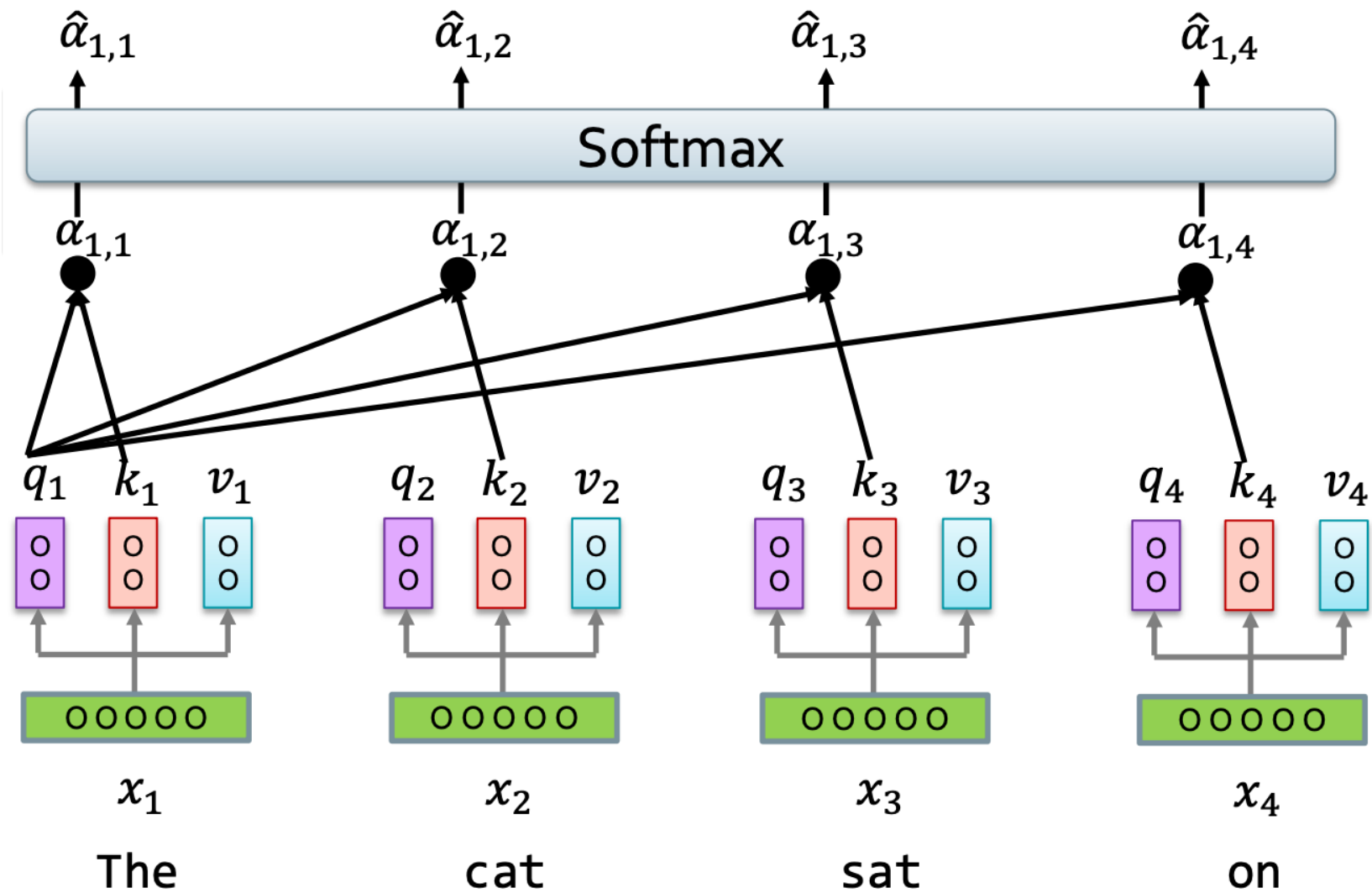
# Self-attention



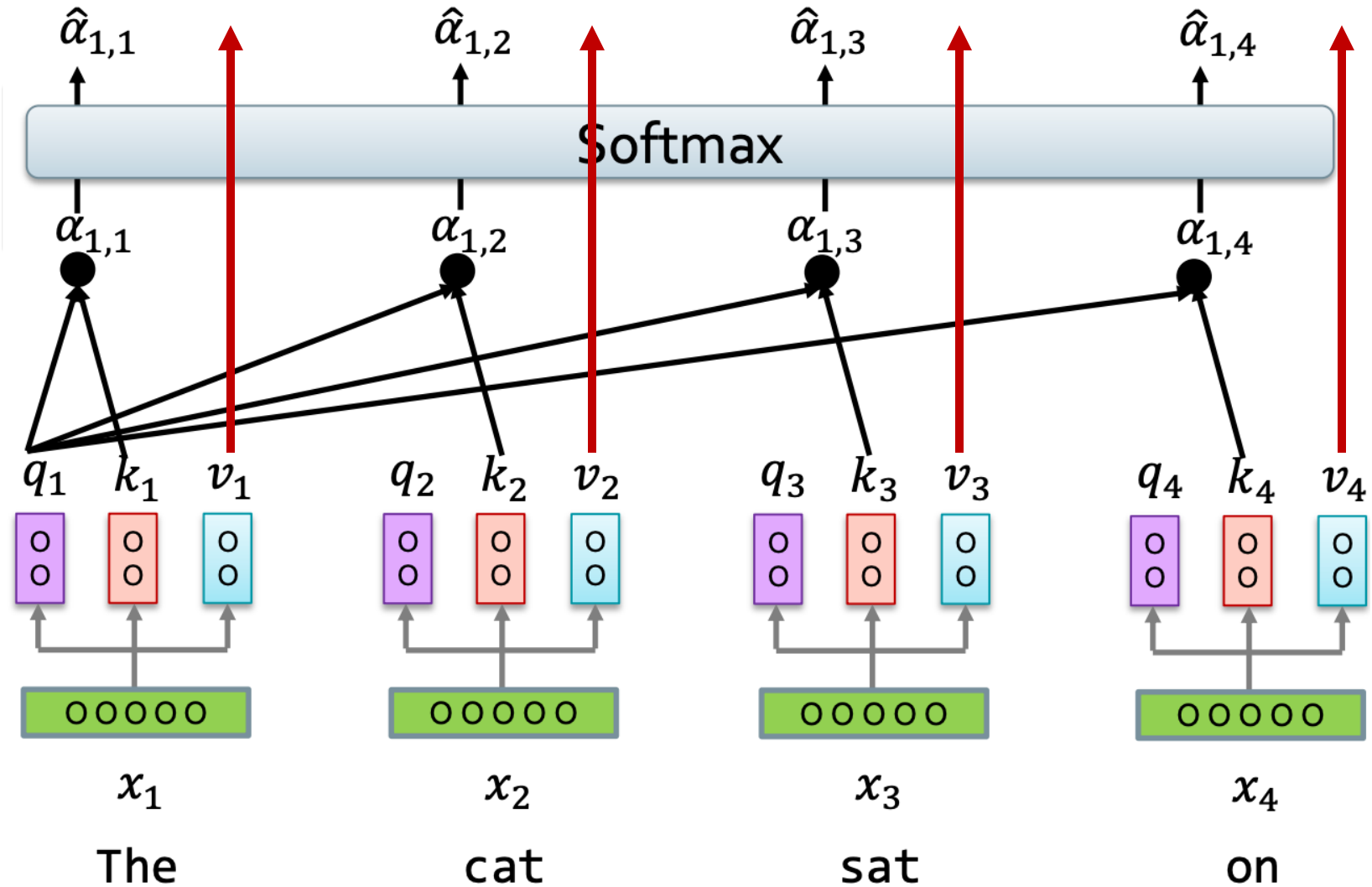
# Self-attention



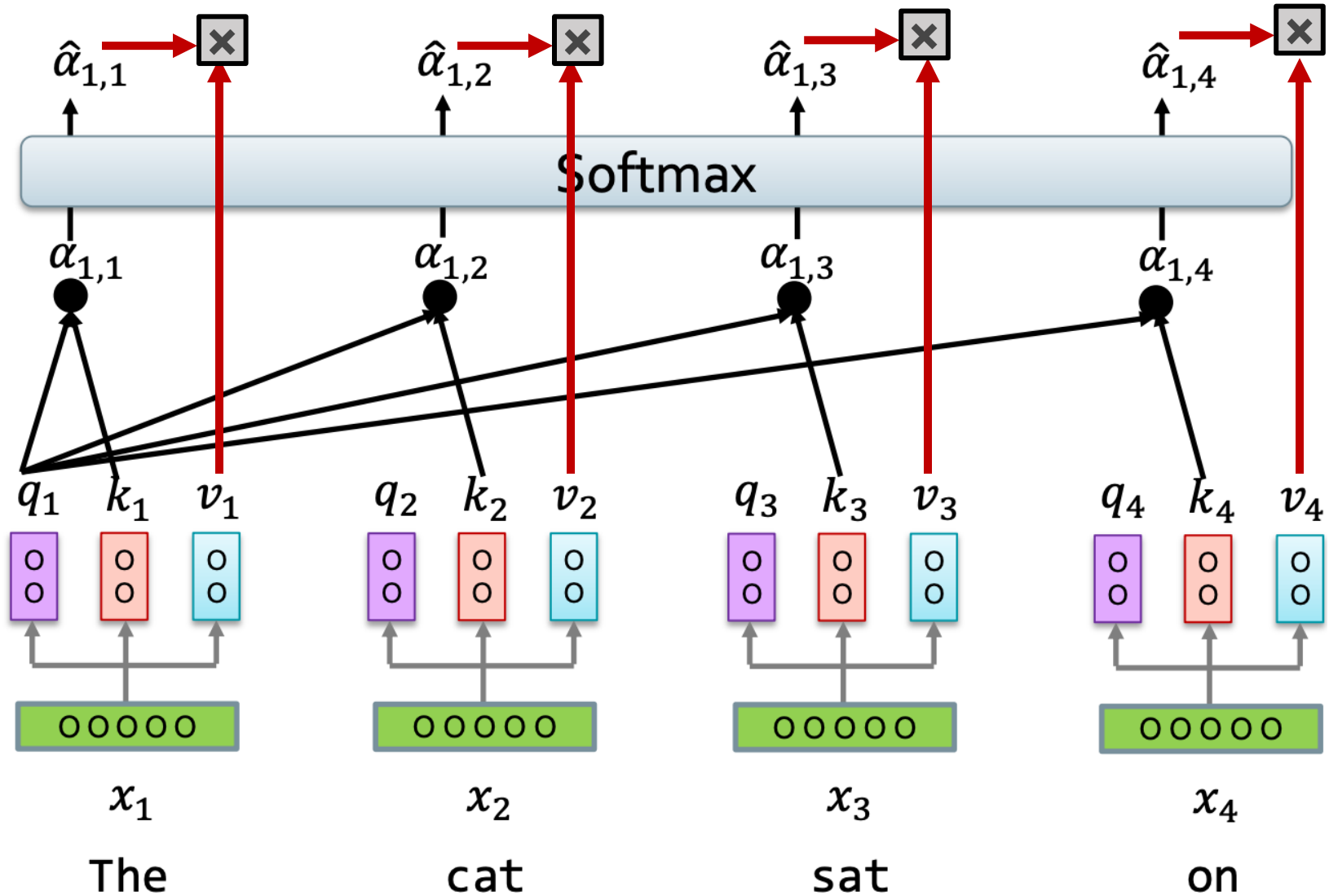
# Self-attention



# Self-attention

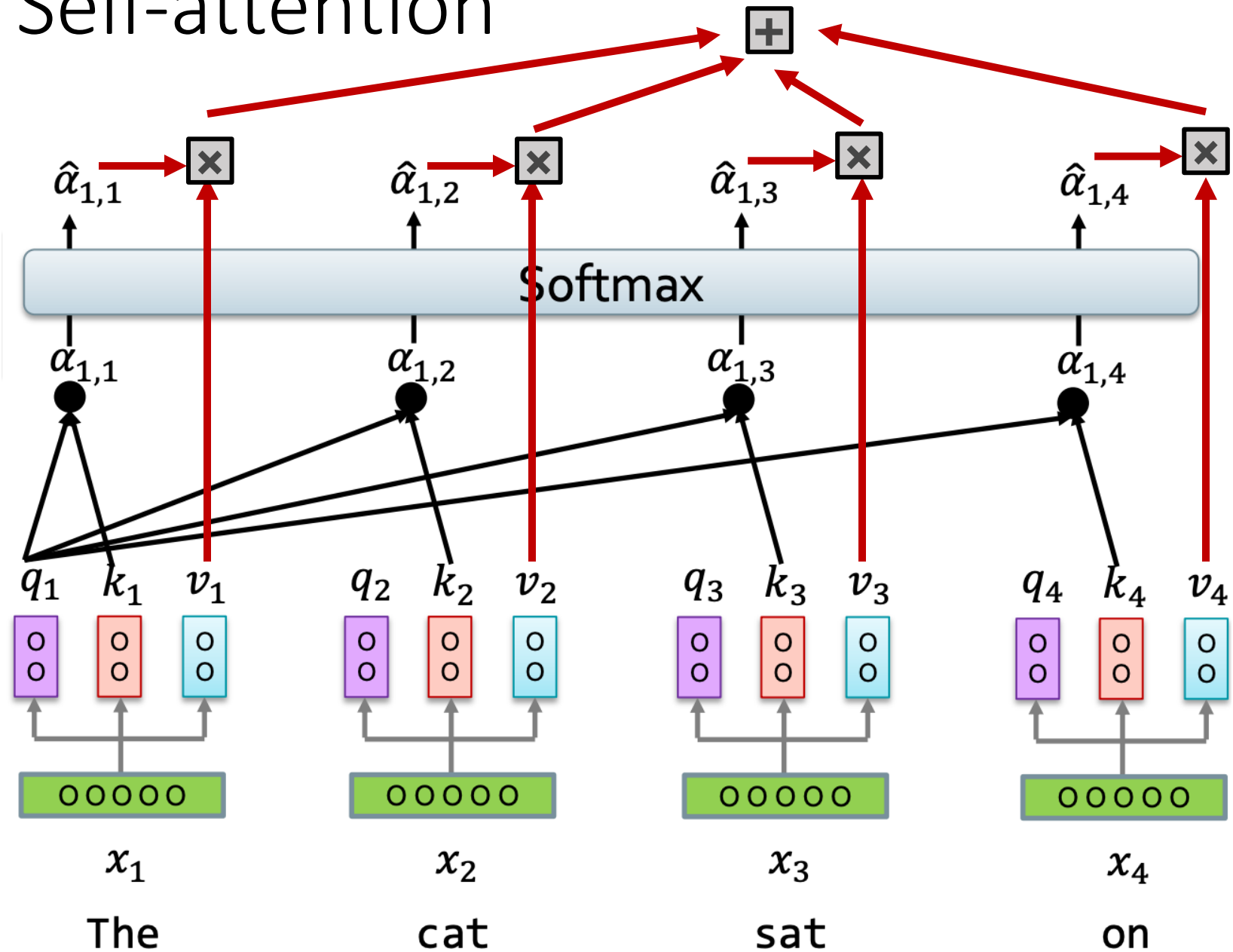


# Self-attention

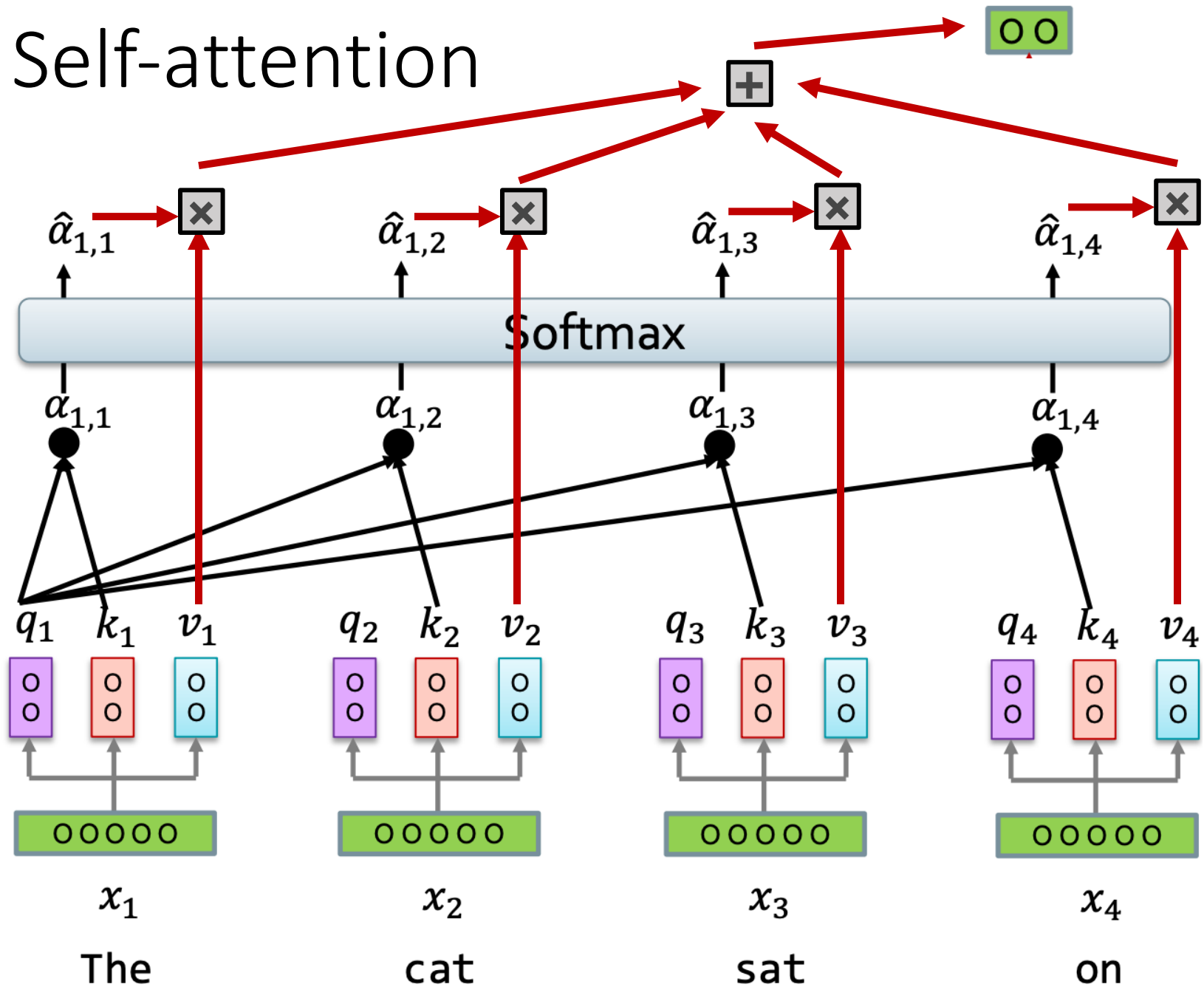




# Self-attention

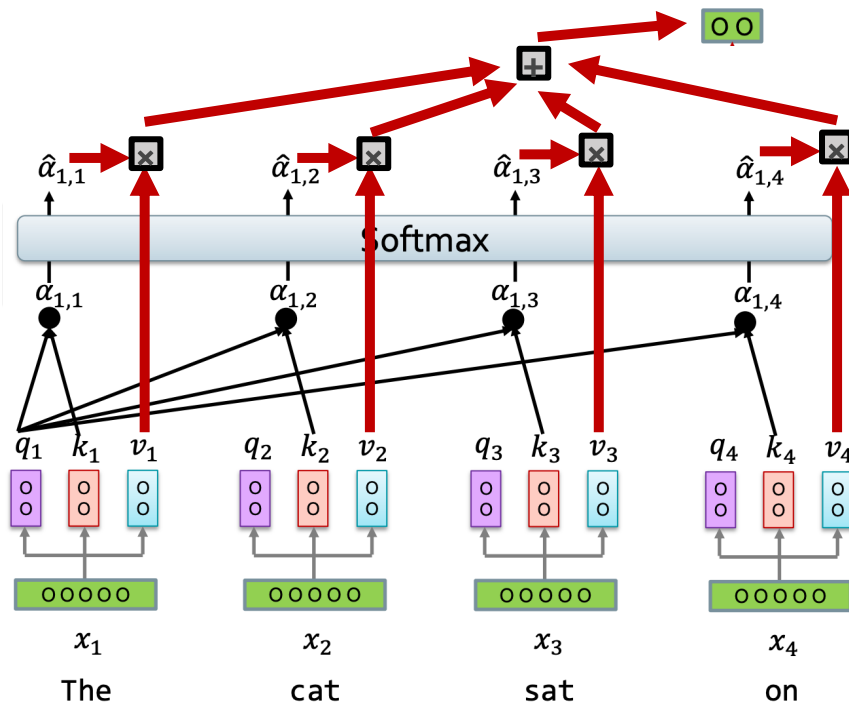


# Self-attention



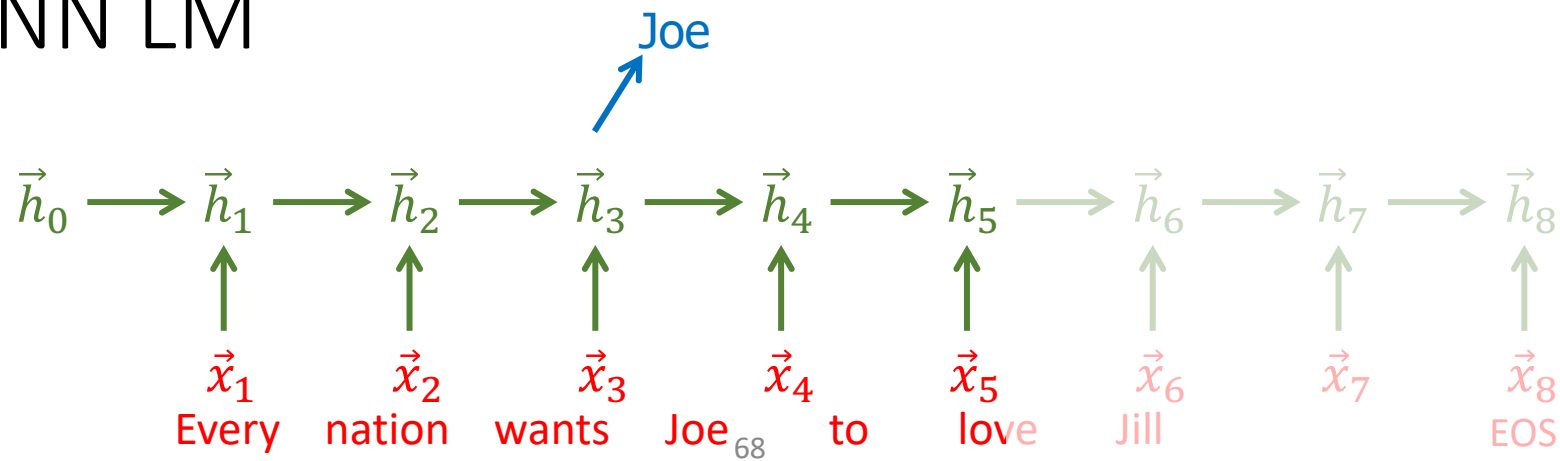
# Self-attention

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

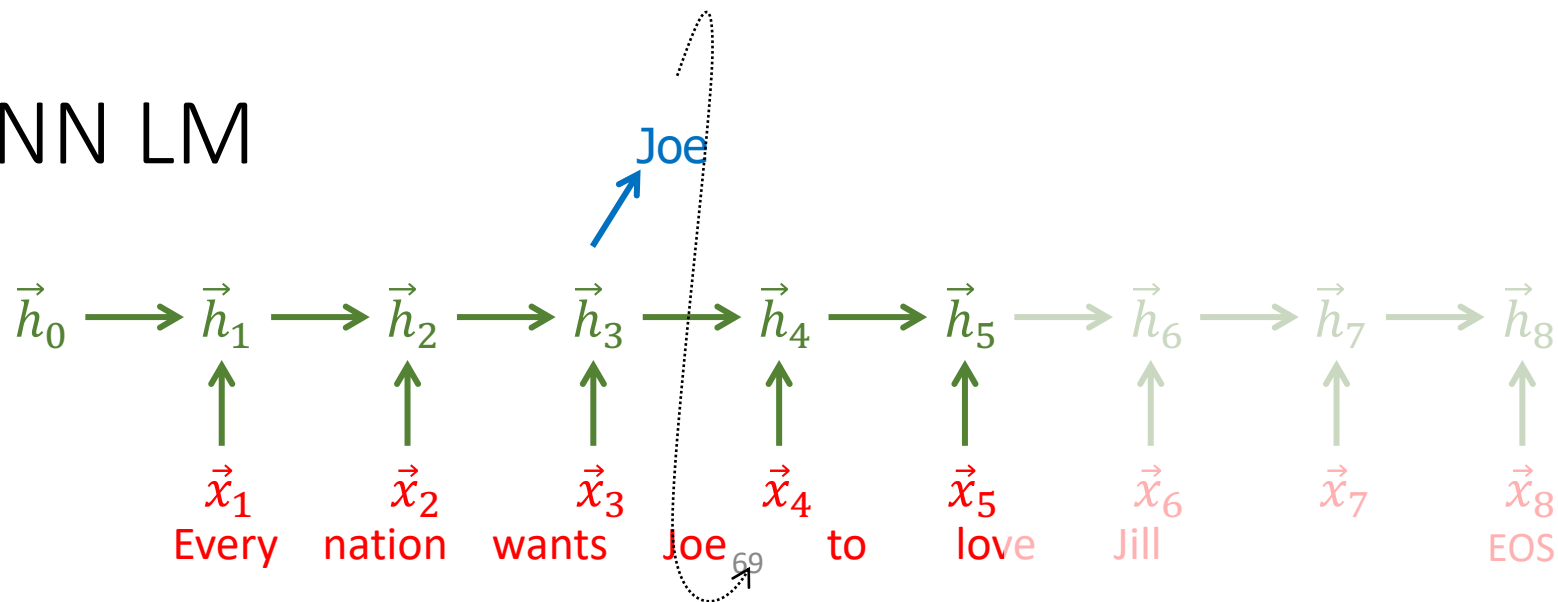


This is the main idea behind a **transformer**

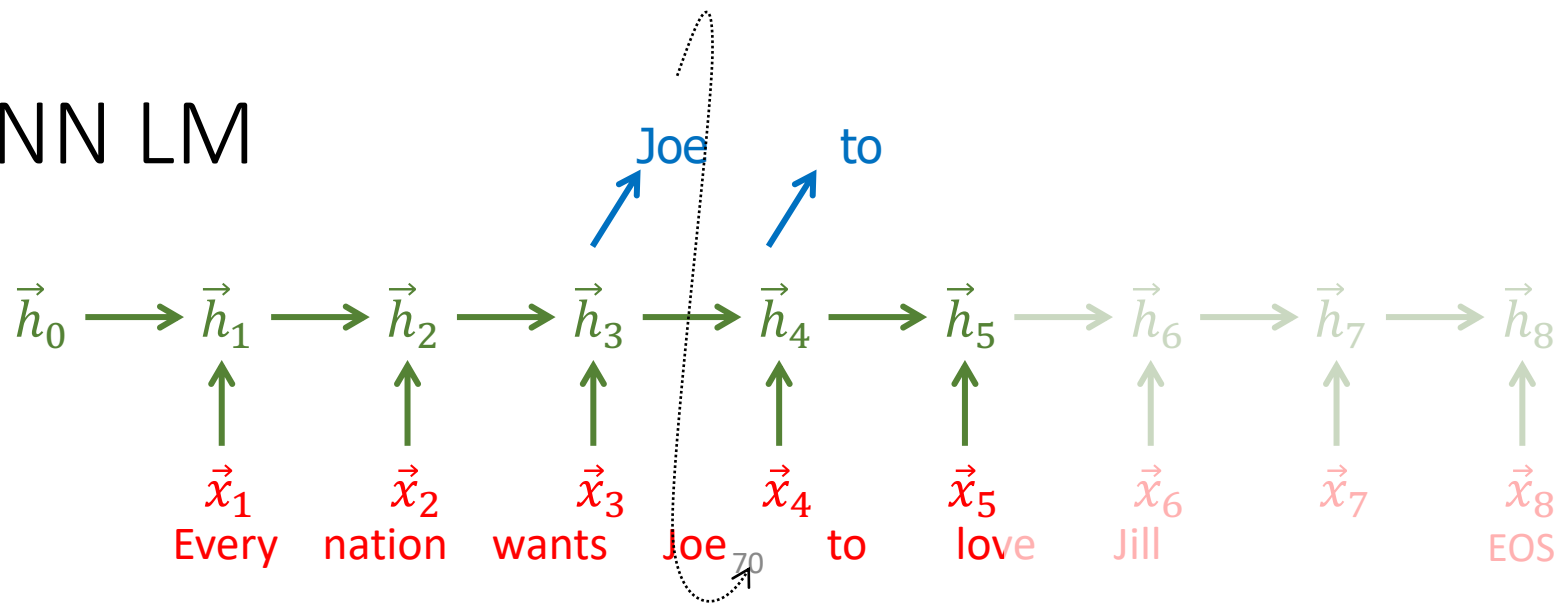
# RNN LM



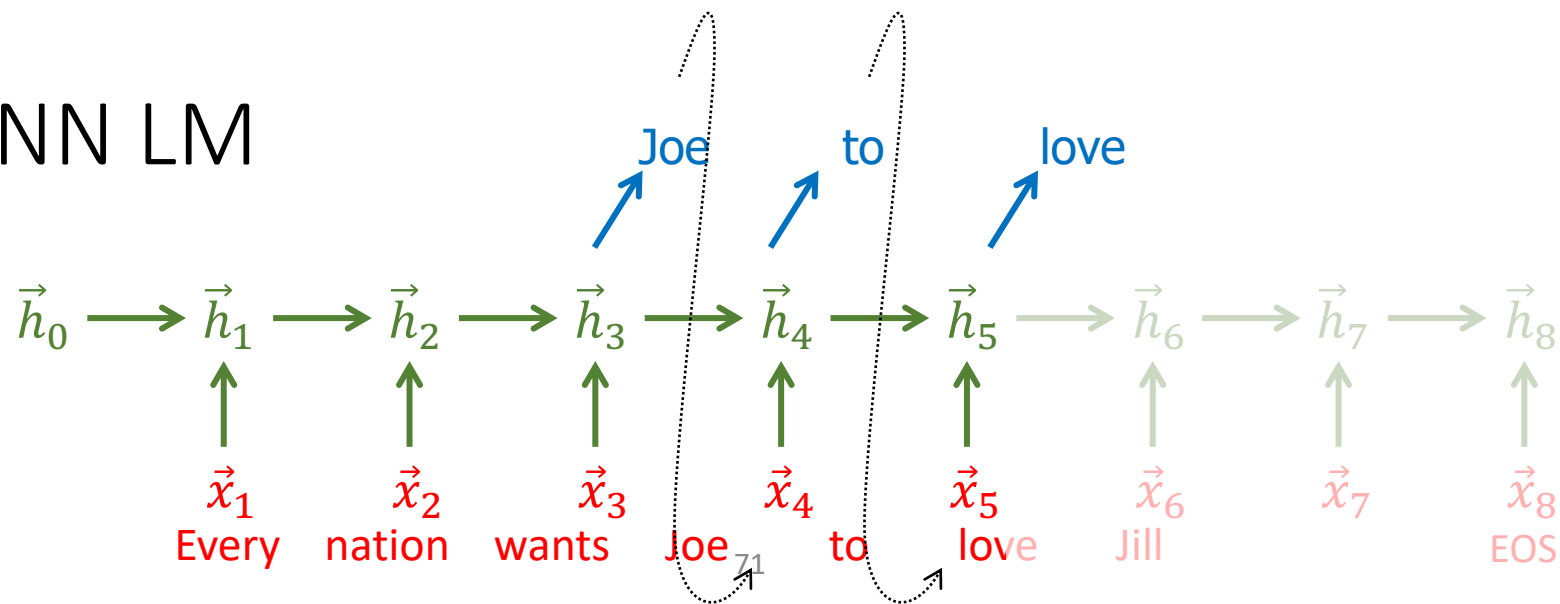
# RNN LM



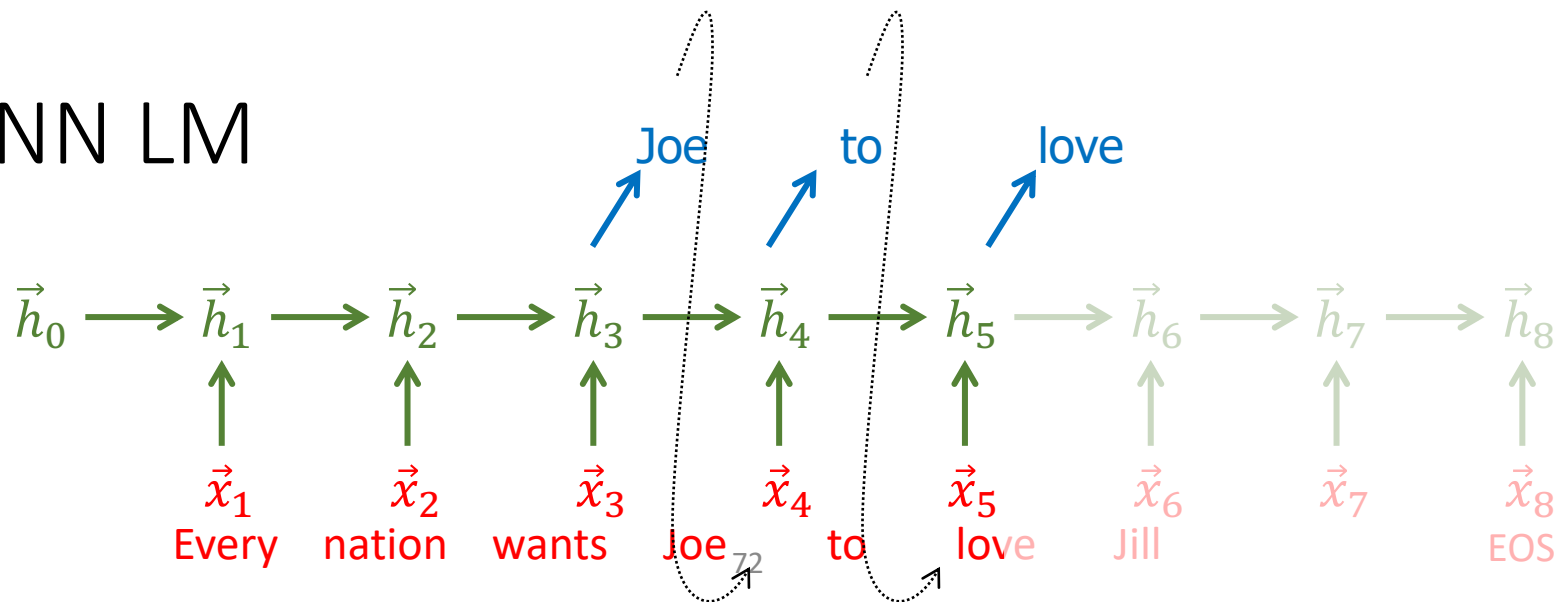
# RNN LM



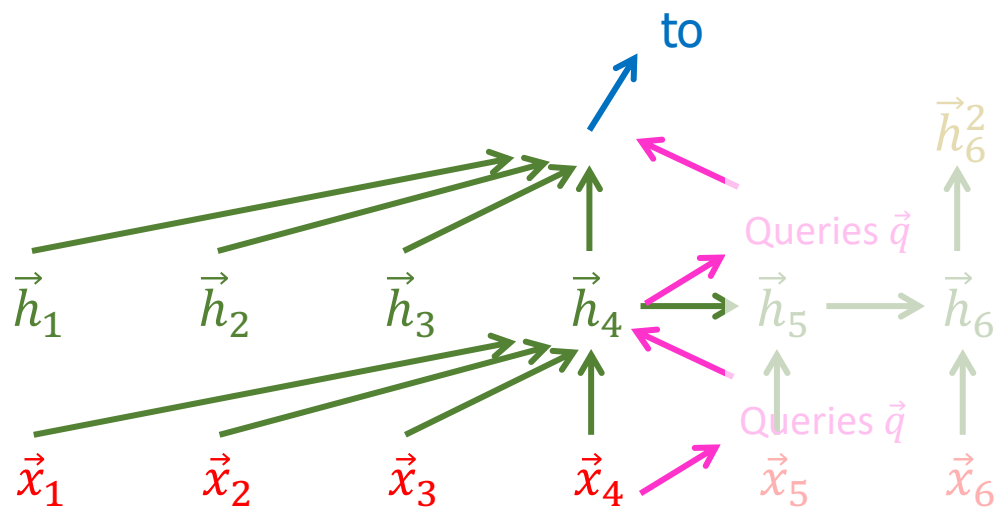
# RNN LM



# RNN LM

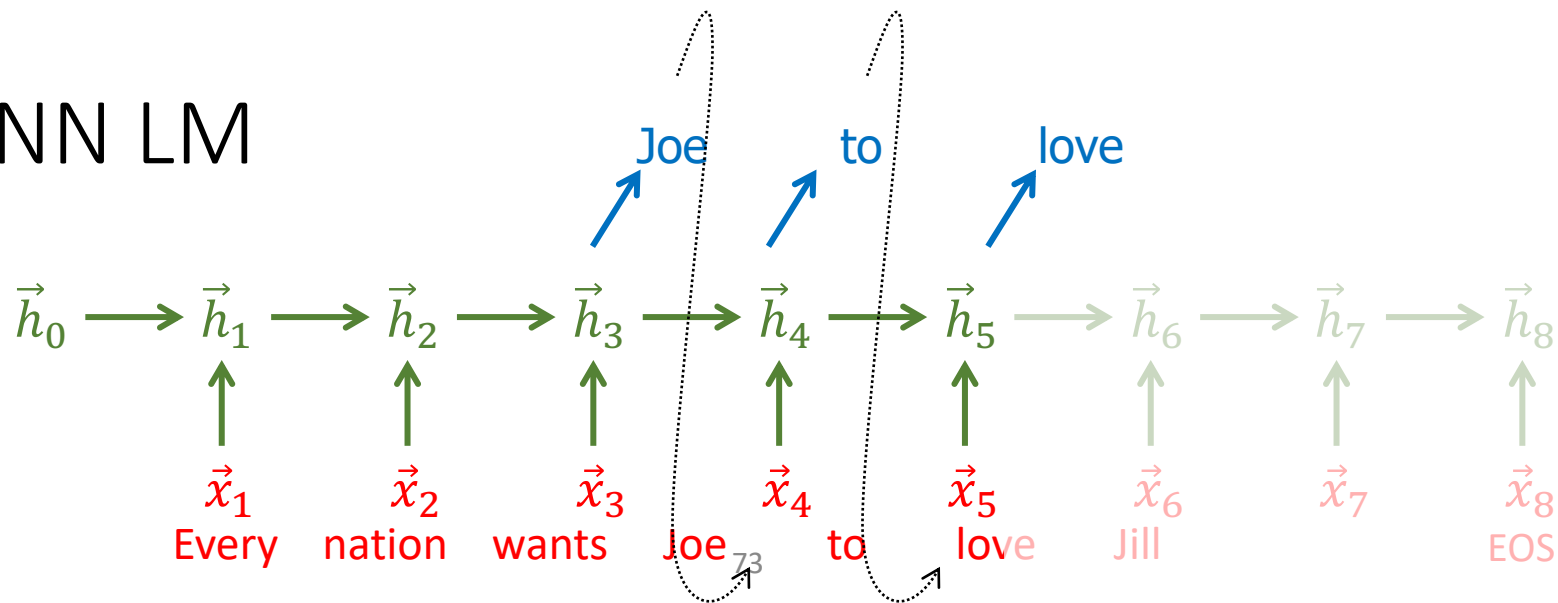


# Transformer (self-attention) LM



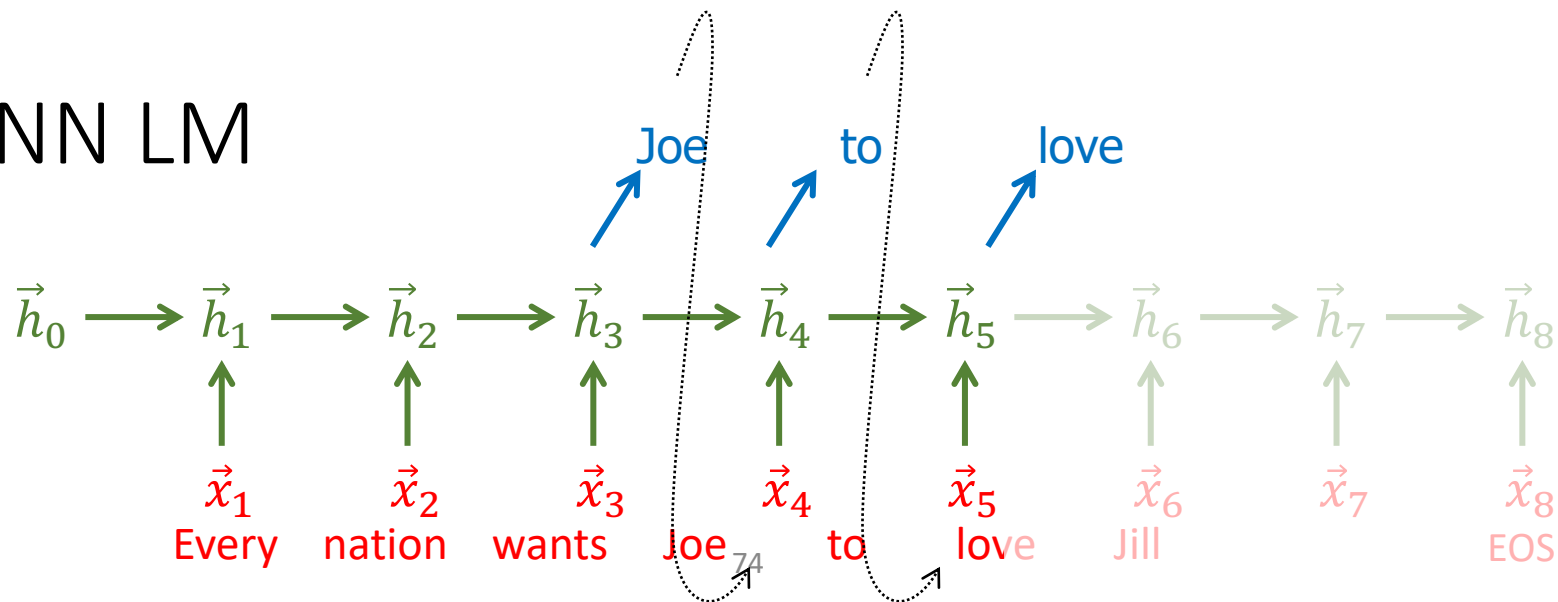


# RNN LM

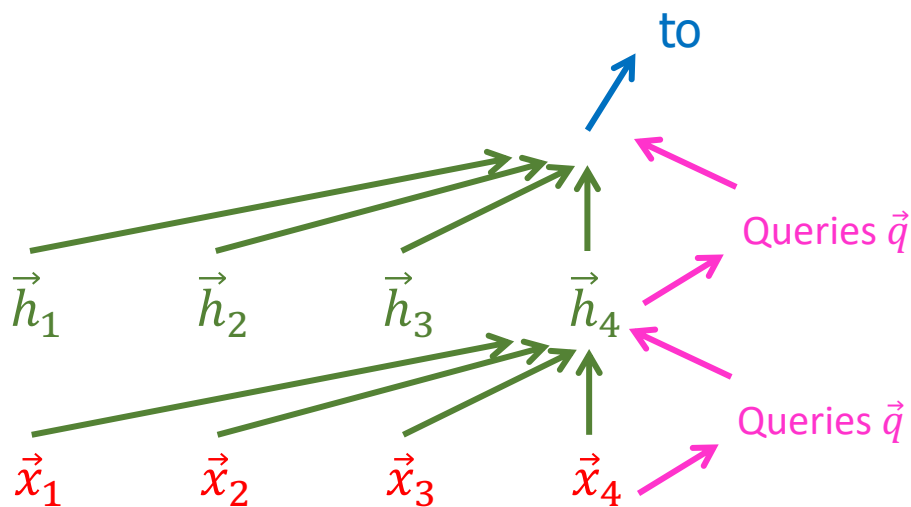


# Transformer (self-attention) LM

# RNN LM



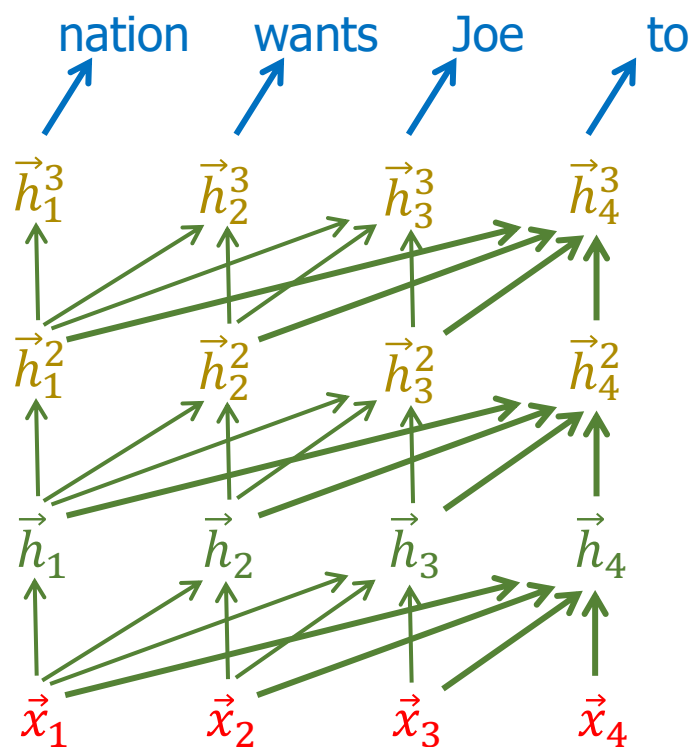
# Transformer (self-attention) LM



# 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  $\vec{x}_0 = \langle s \rangle$ !  
It's missing from our diagrams.)

Training,  
on GPU,  
per layer

# RNN vs. Transformer

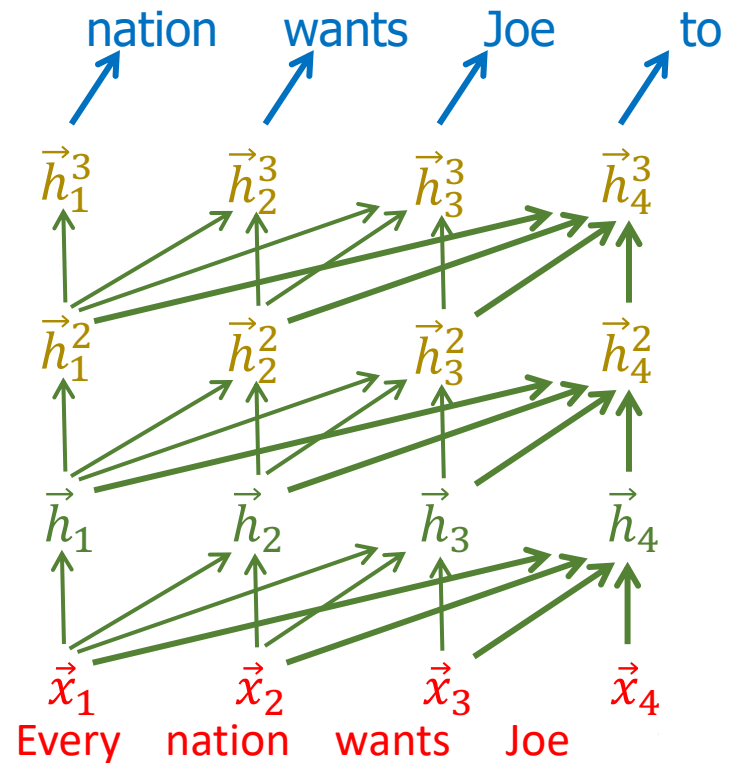
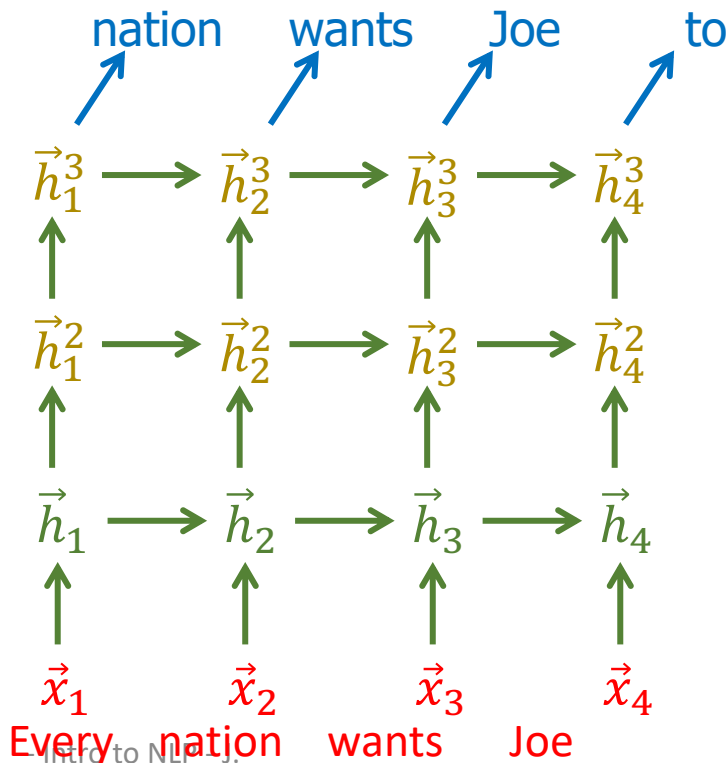


Computations: ☺  $O(n)$

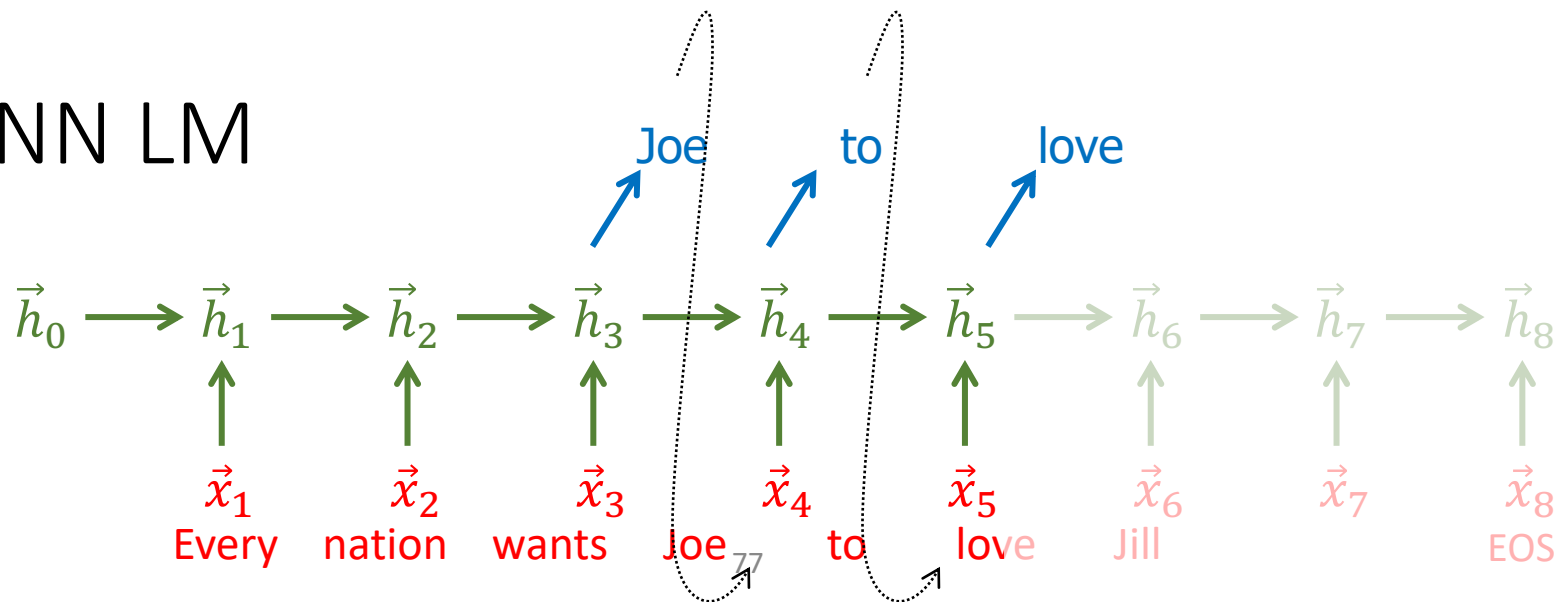
# serial steps: ☹  $O(n)$  due to  $\longrightarrow$

☹  $O(n^2)$

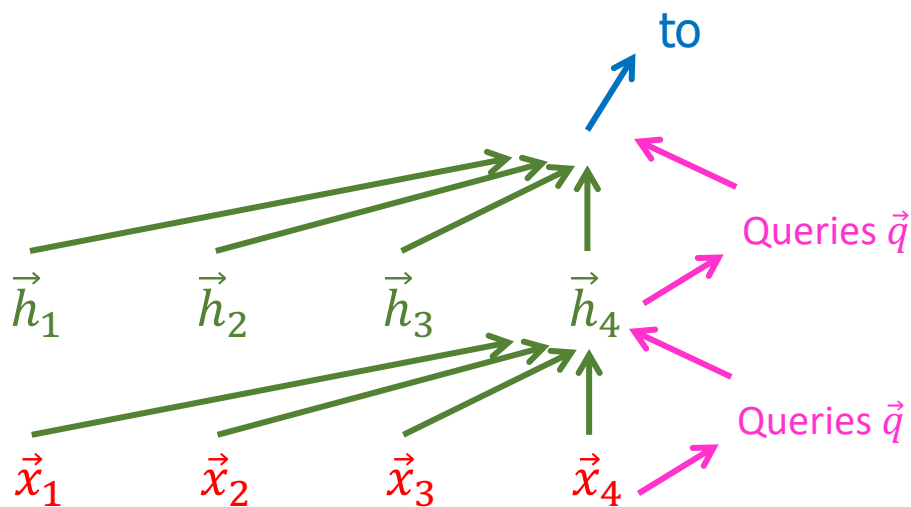
☺  $O(1)$ : all  $\nearrow$  in parallel  
+  $O(\log n)$  to sum  $n$  inputs



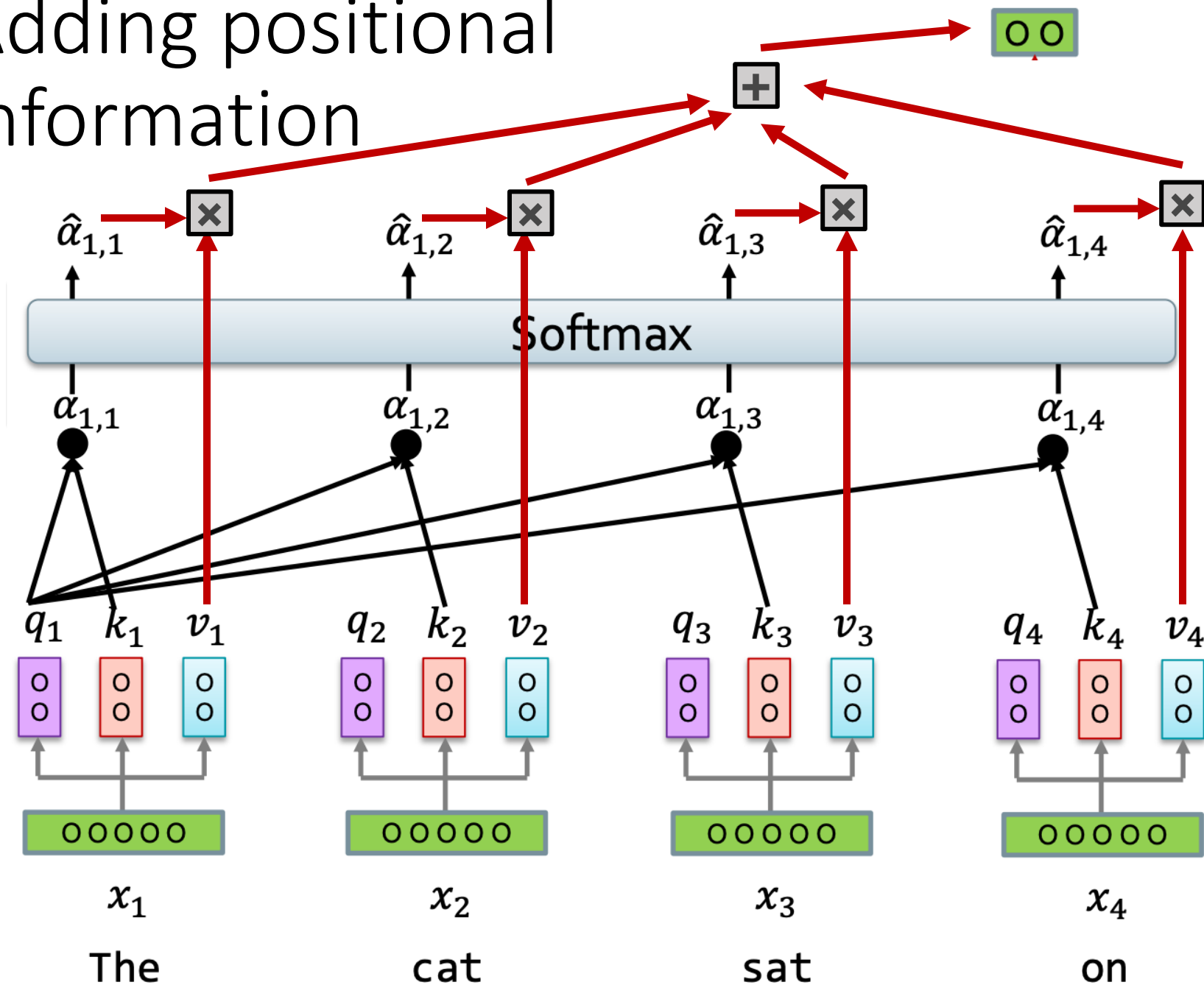
# RNN LM



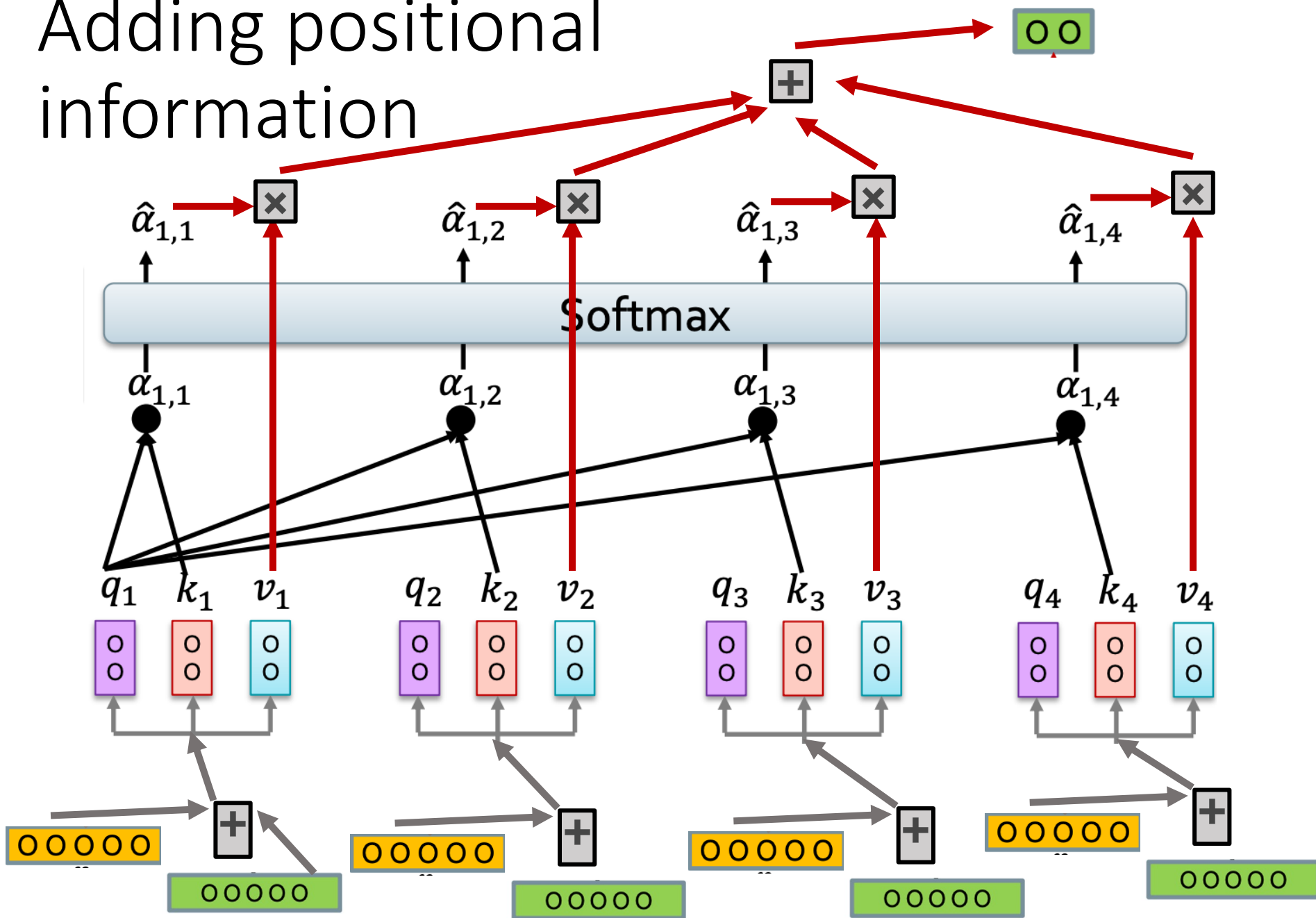
# Transformer (self-attention) LM



# Adding positional information

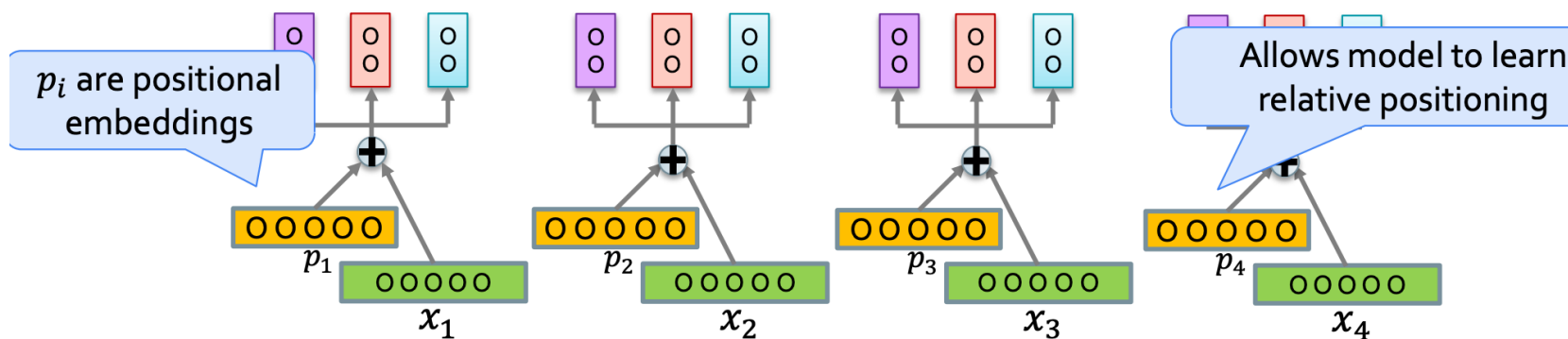
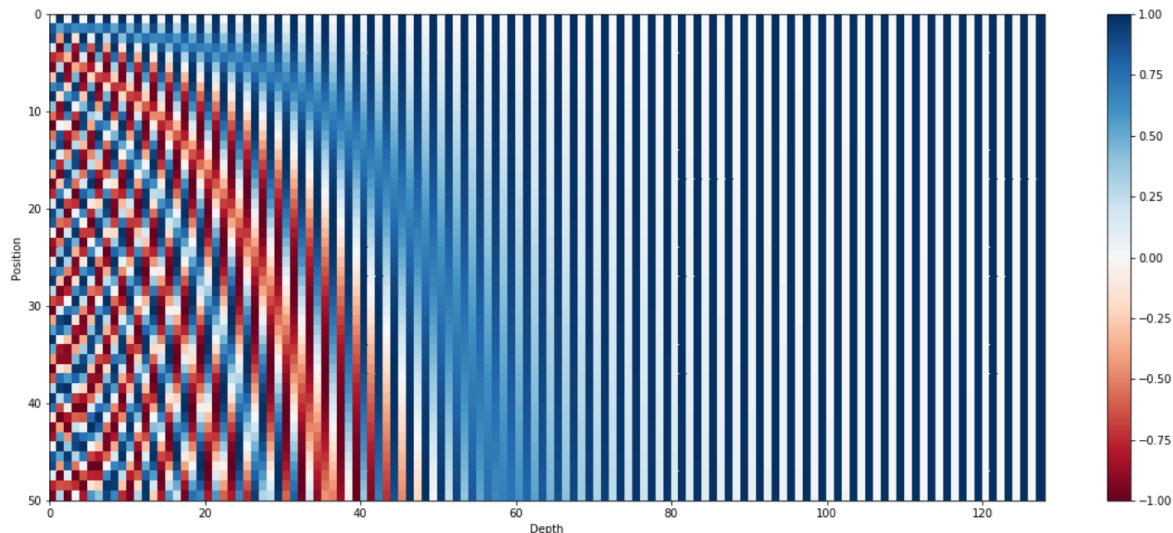


# Adding positional information



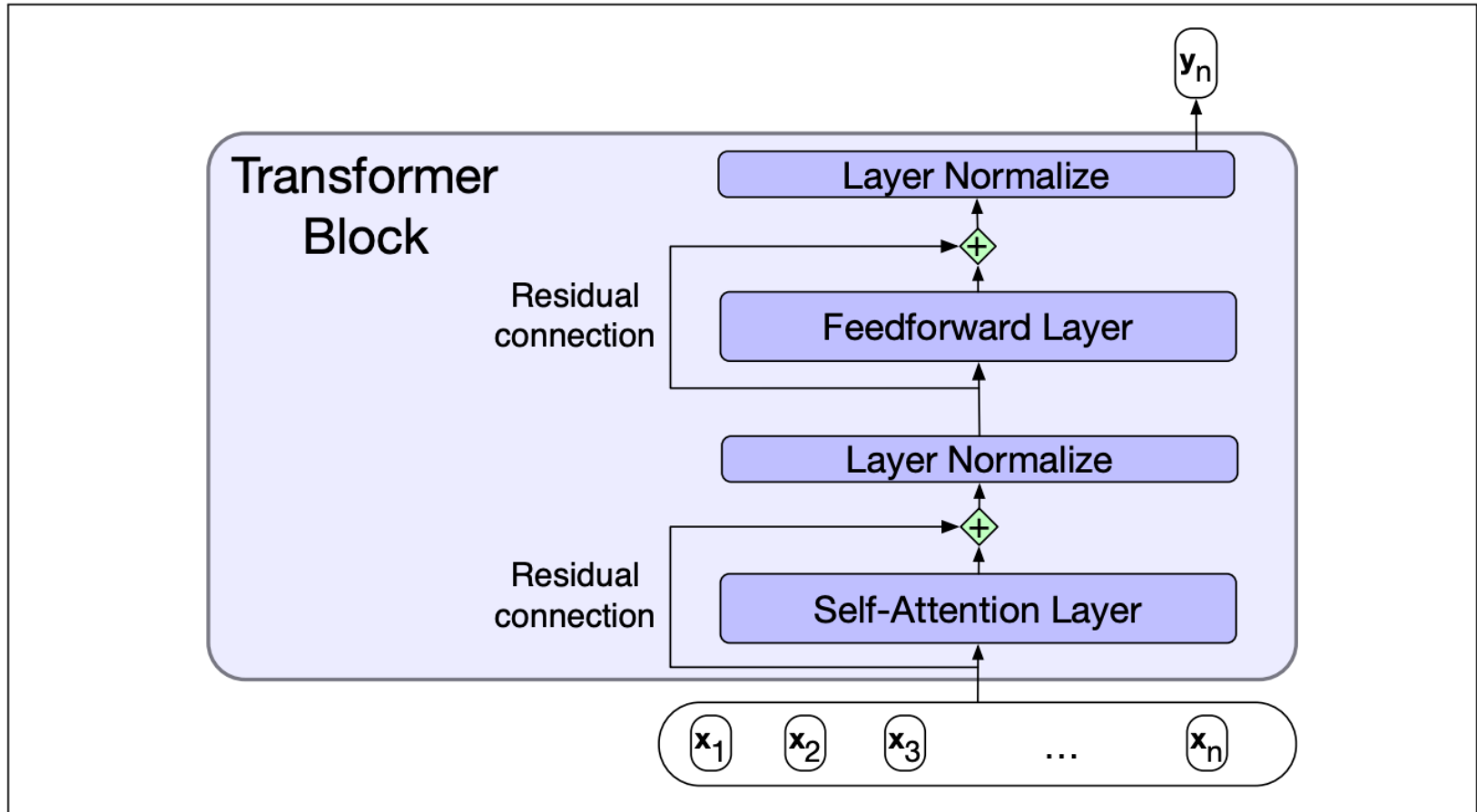
# Adding positional information

An approach:  
Sine/Cosine encoding





# Transformer block



# Transformer block

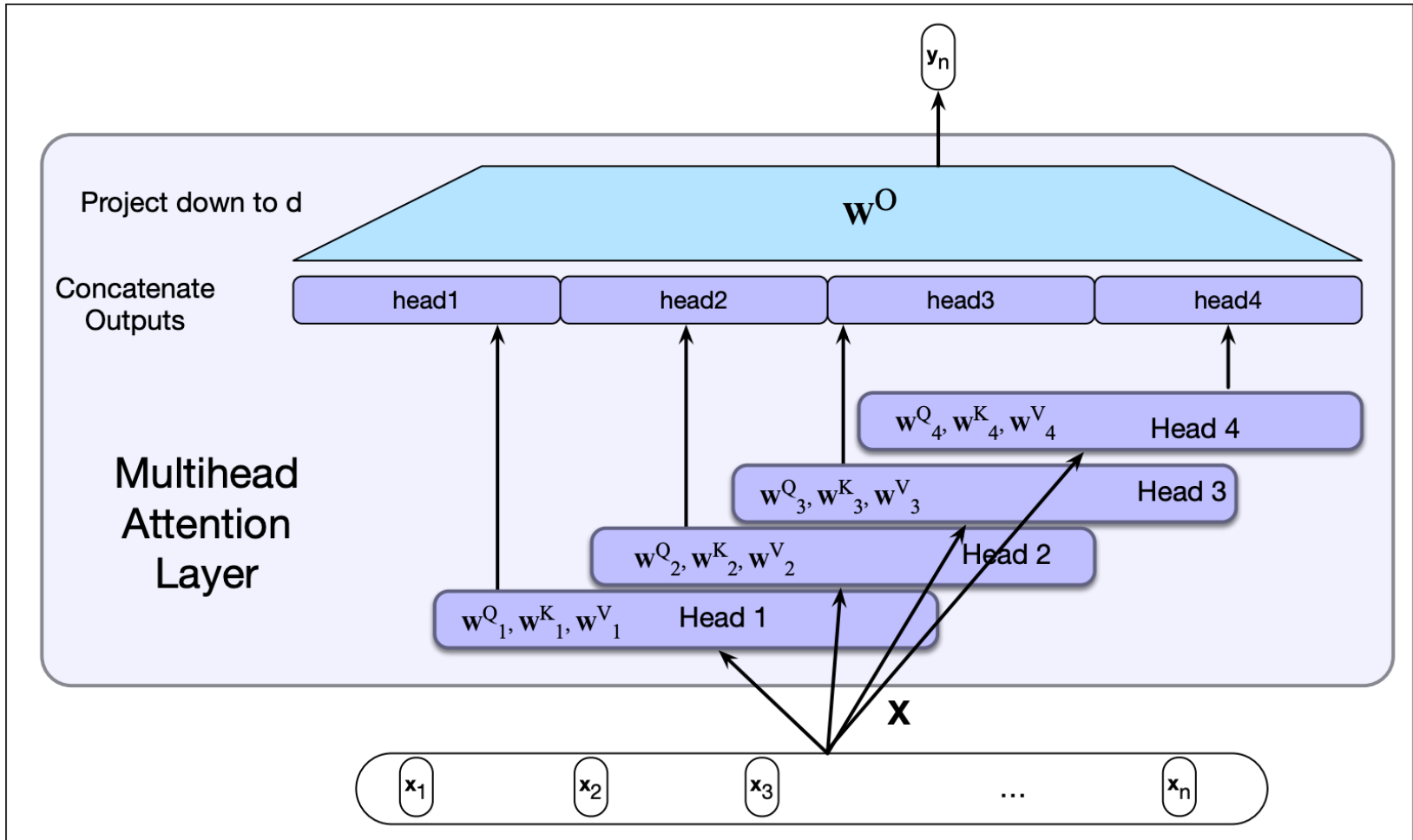
## Residual connection:

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

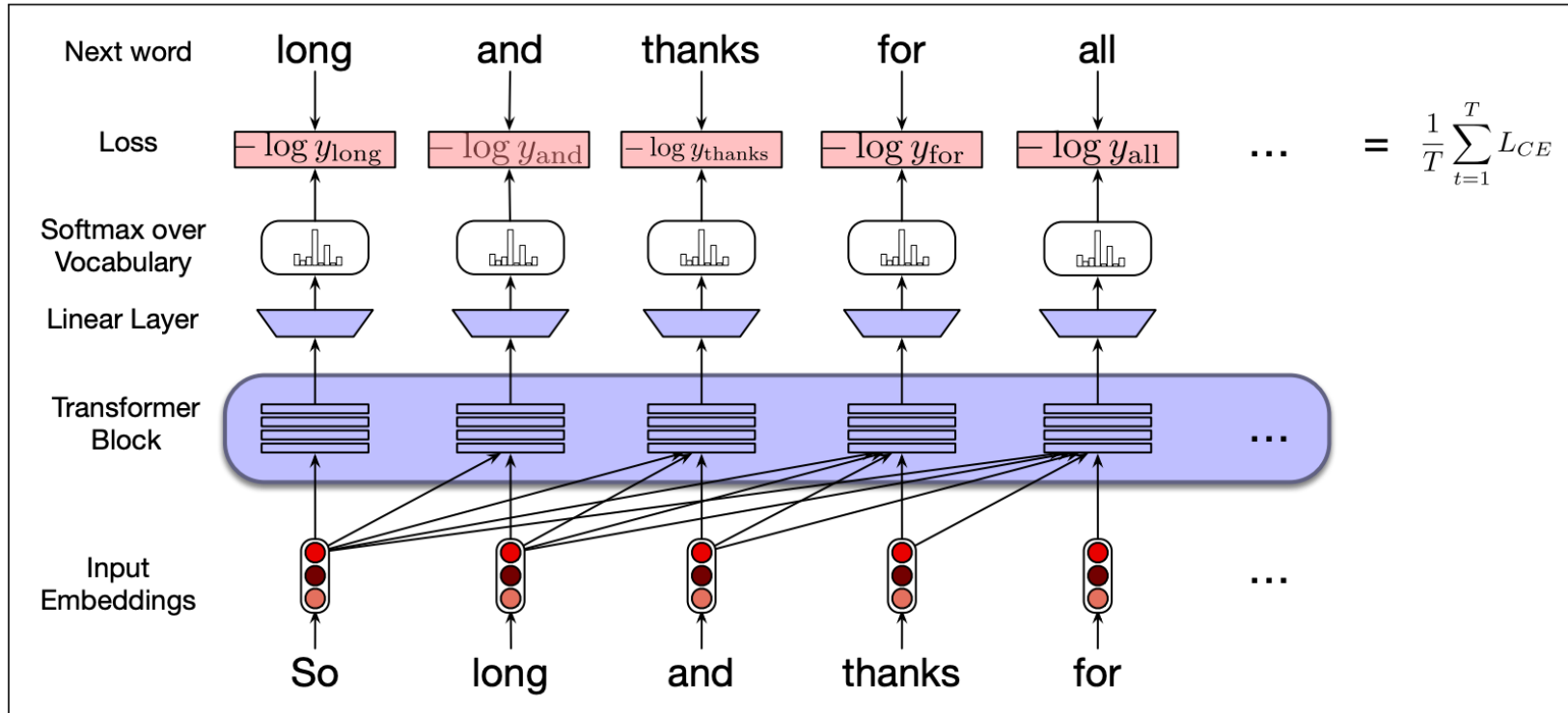
## Layer normalization

- 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

So, training requires a lot of data

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

Like in the probing work

Led to SoTA models

# Next class

- Pre-training and fine-tuning
- Examples of popular transformer models:
  - BERT: **B**idirectional **E**ncoder **R**epresentations from **T**ransformers (Google)
  - RoBERTa: **R**obustly **O**ptimized BERT (Facebook)
  - GPT: **G**enerative **P**re-trained **T**ransformer (OpenAI)

# Outline

Recap – RNNs, Seq2Seq

Attention

Self-attention

Transformer

**Pytorch demo (if time)**