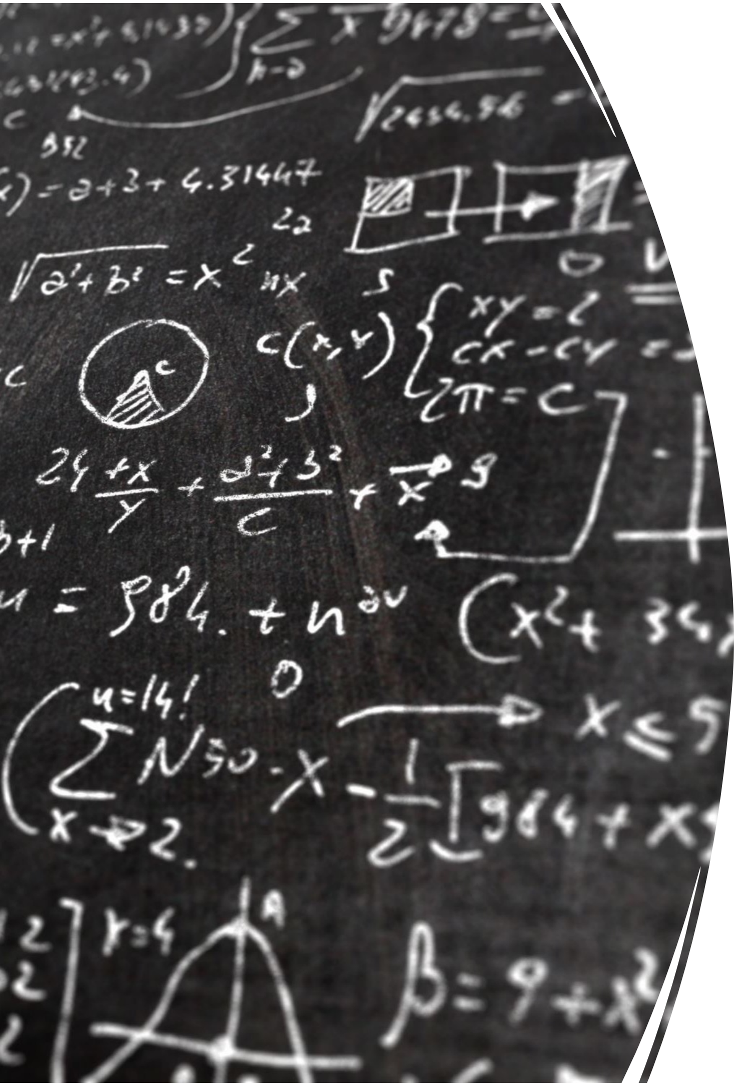


CS 383 – Computational Text Analysis

Lecture 16
Transformers

Adam Poliak

03/20/2023

Slides adapted from Daniel Khashabi, Chris Manning

Announcements

- Final project ideation
  - 13 submissions across 15 students
  - Mandating partner, must work with a partner!
  - Due this Friday 03/25

- HW05:
  - Due tonight
  - Ignore test.py
  - Get the driver working

Twitter API for next HW

1. Create a Twitter developer account https://developer.twitter.com/
2. Go to https://developer.twitter.com/en/apps and log in with your Twitter user account.
3. Click "Create an app"
4. Fill out the form, and click "Create"
5. A pop up window will appear for reviewing Developer Terms. Click the "Create" button again.

Instructions from http://socialmedia-class.org/twittertutorial.html

Look here for instructions on how to use Tweepy: https://github.com/BC-COMS-2710/summer21-material/blob/master/demo/Demo13.ipynb

Outline

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

Attention

What problem does it solve?

- Bottleneck from having single sentence representation

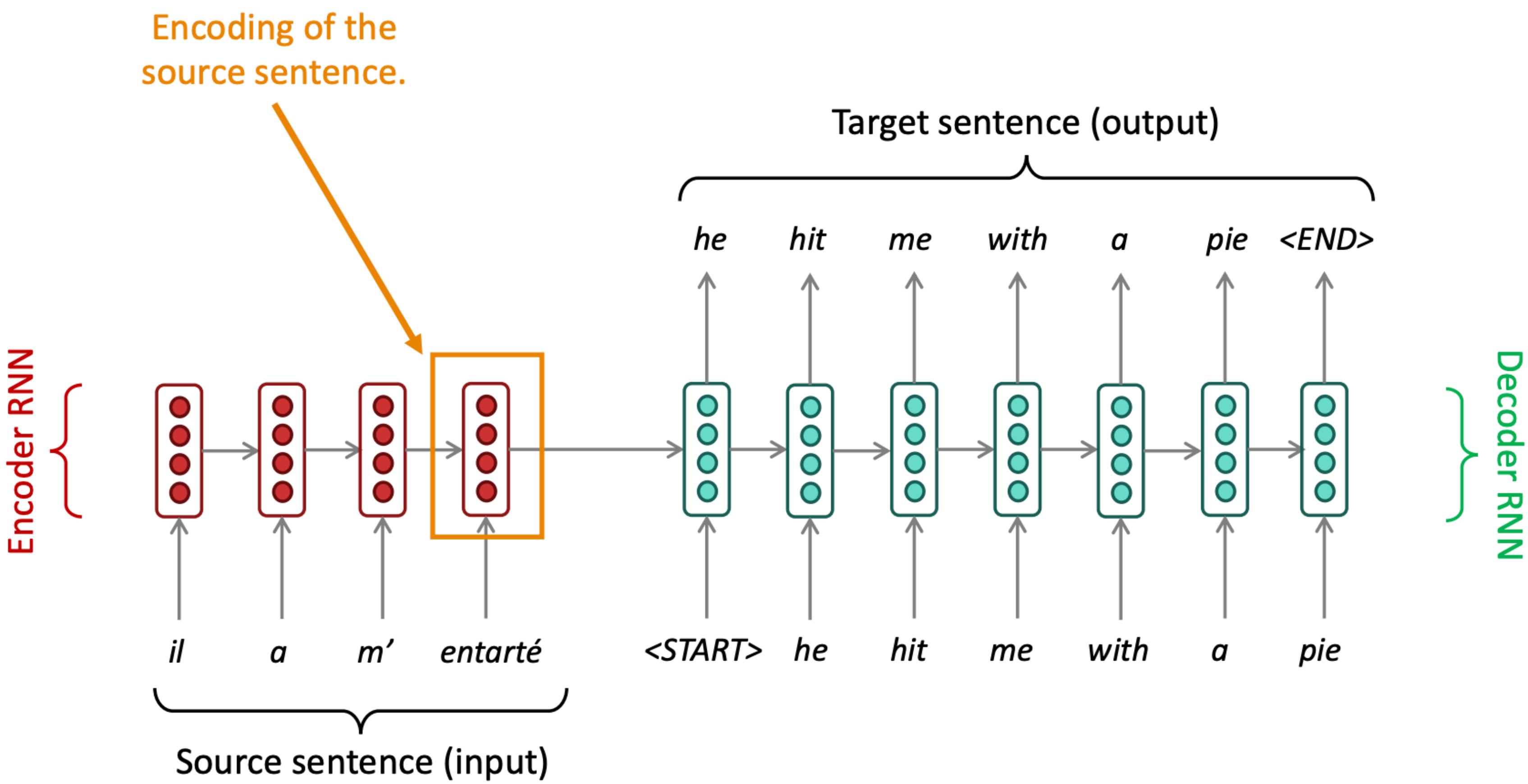
How does it work?

- Instead of looking at just the sentence representation, combine it with a new attention vector for each prediction

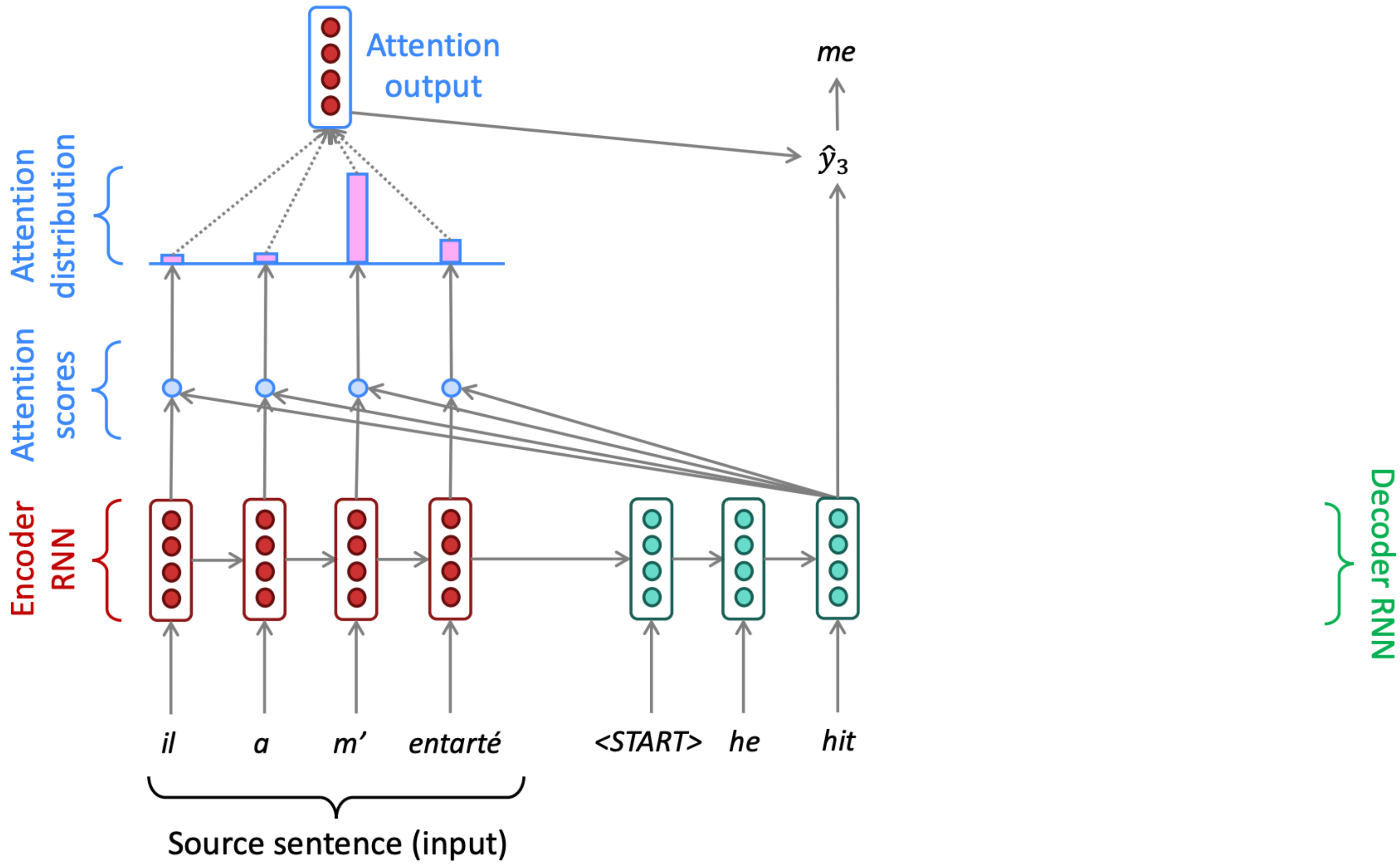
Attention vector/output:

- A weighted average off the outputs of the hidden layer in the encoder

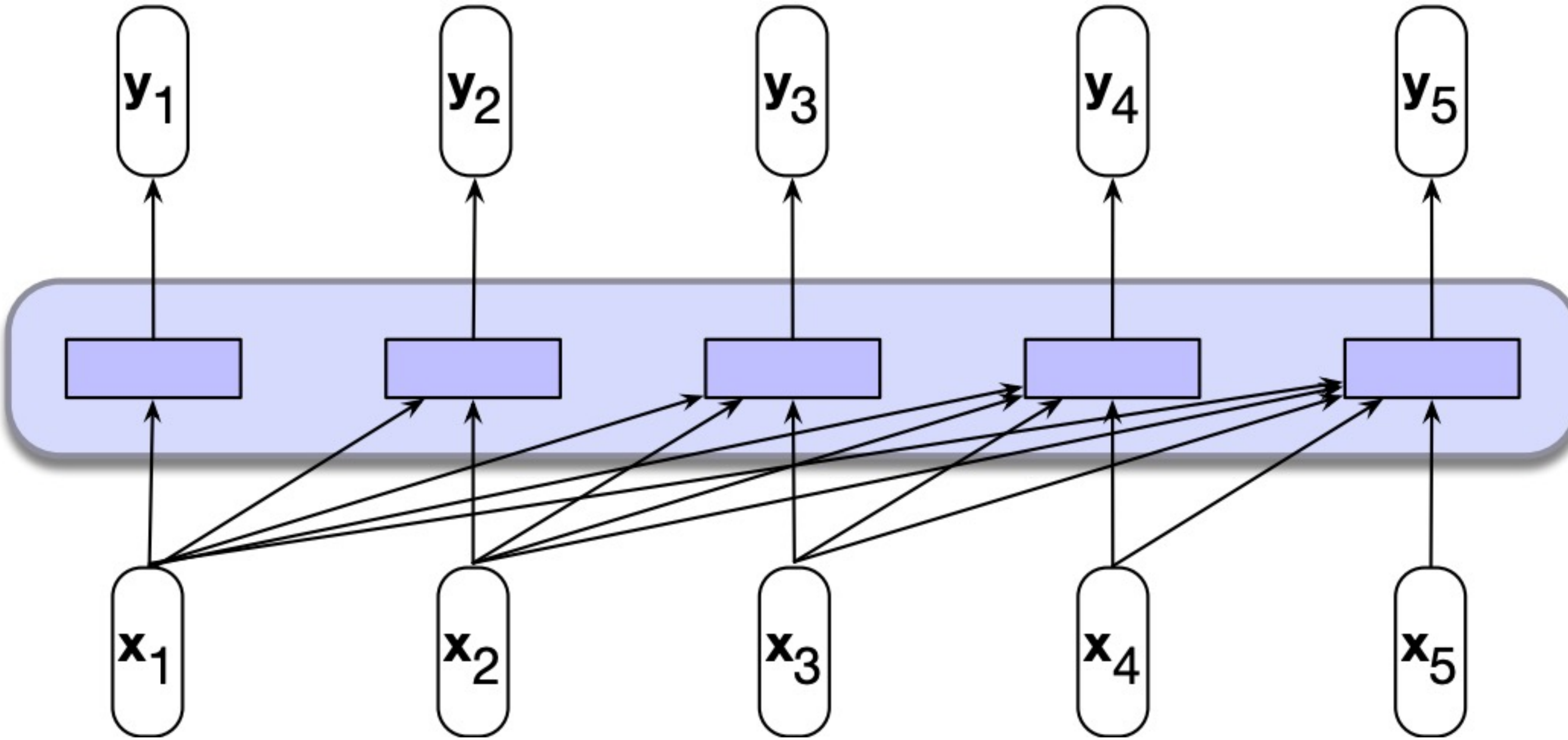
Seq2Seq Model



Seq2Seq w/ Attention



Self-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
- 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

Self-attention in a nutshell

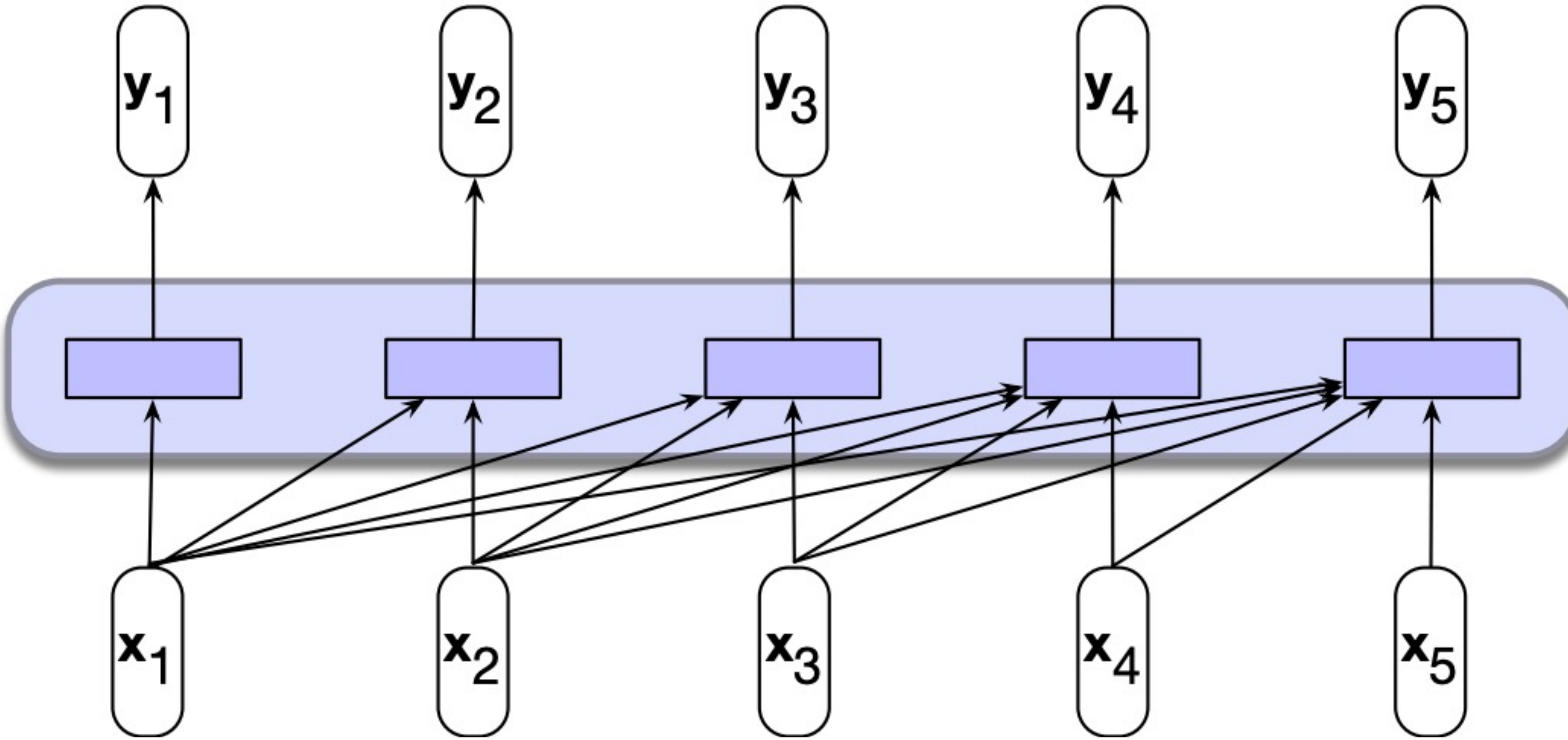
Attention:

- 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:

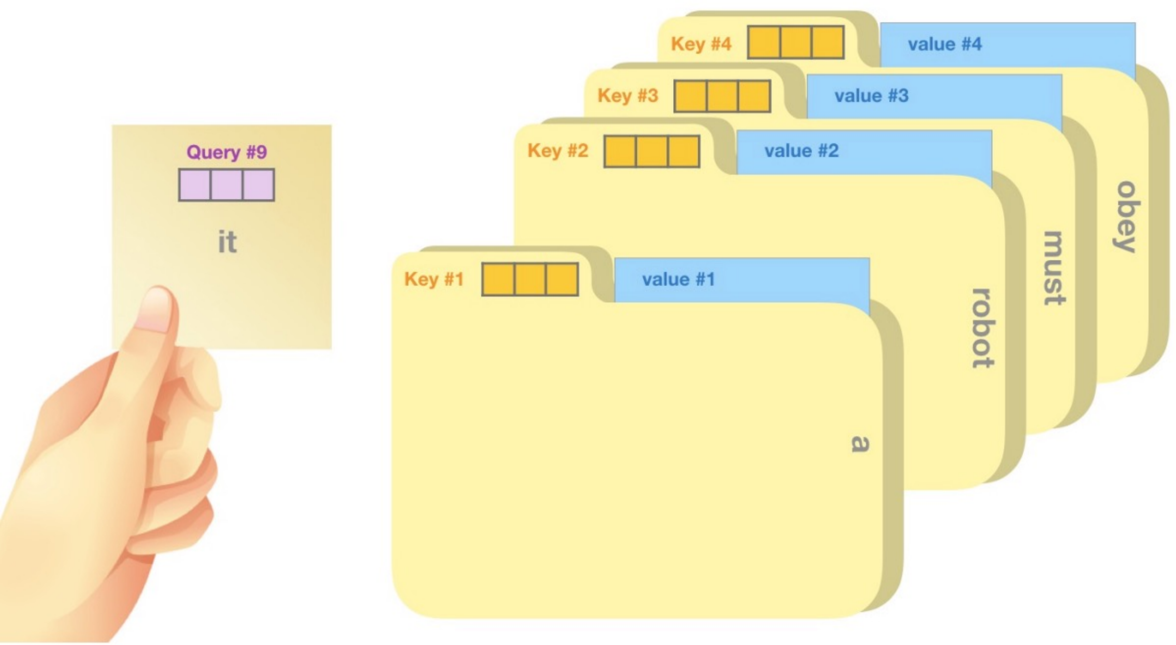
Value:

Terminology

Query: what to match

Key: the thing to match

Value: what to be extracted from the match



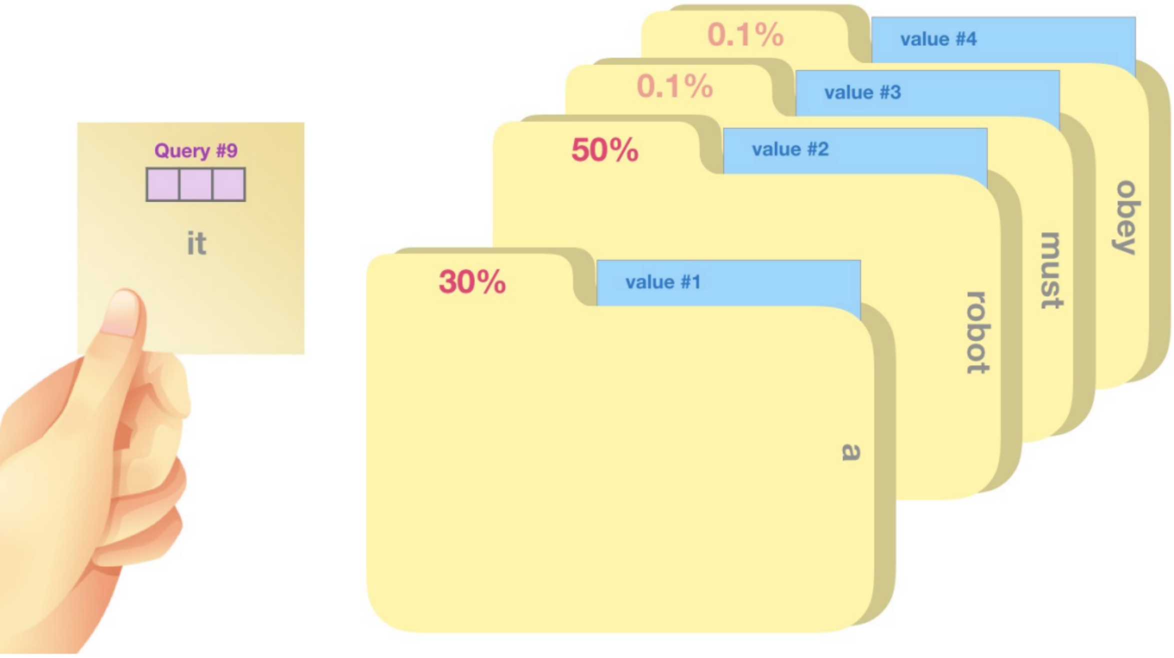
Daniel Khashabi

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:

$$q_i = W^q x_i$$

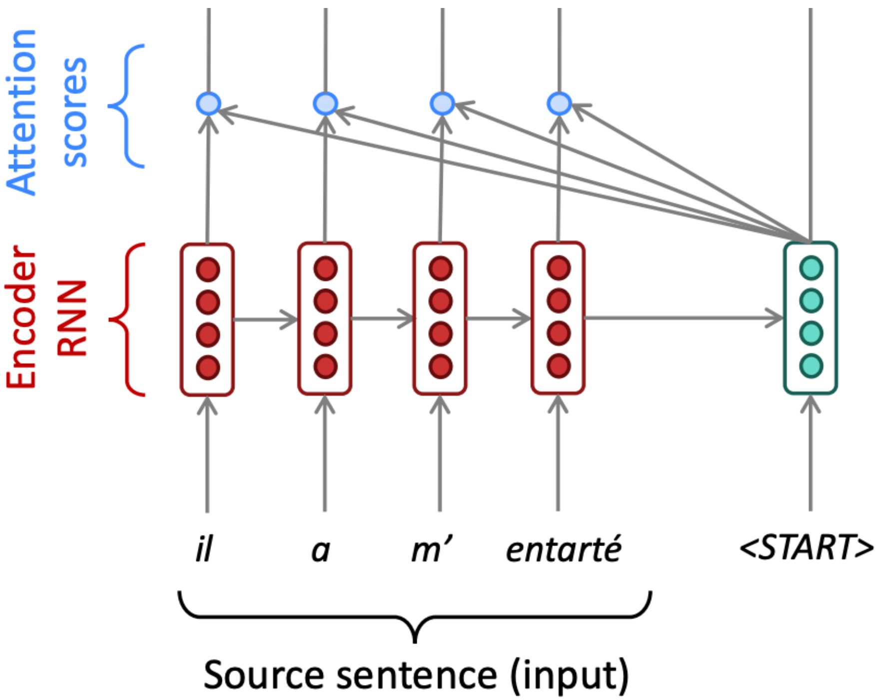
Key: the thing to match:

$$k_i = W^k x_i$$

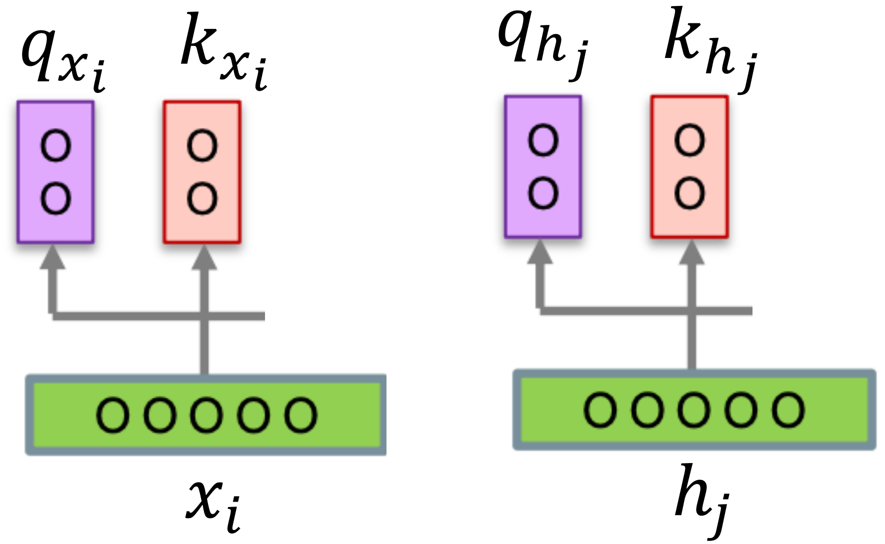
Value: what to be extracted from the match

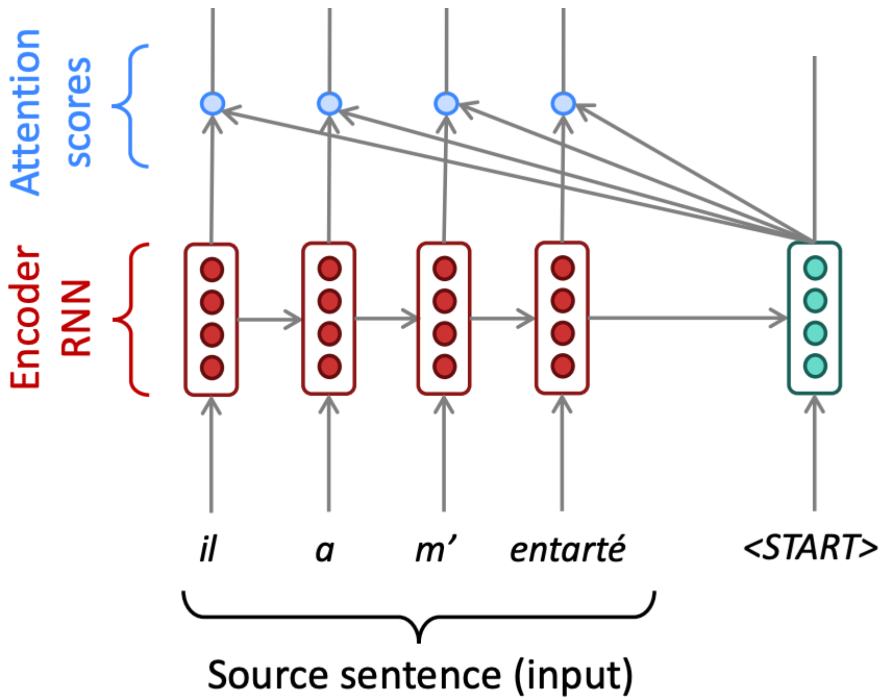
$$v_i = W^v x_i$$

Attention score



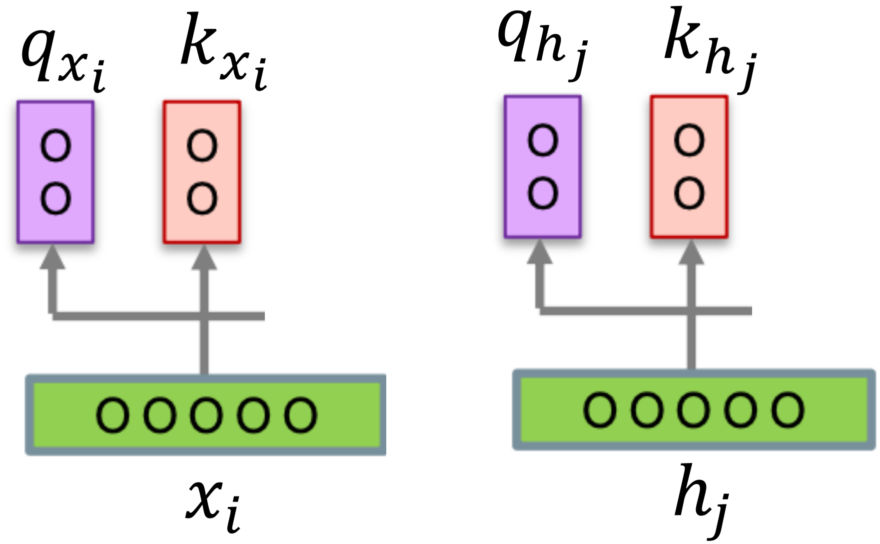
Attention score

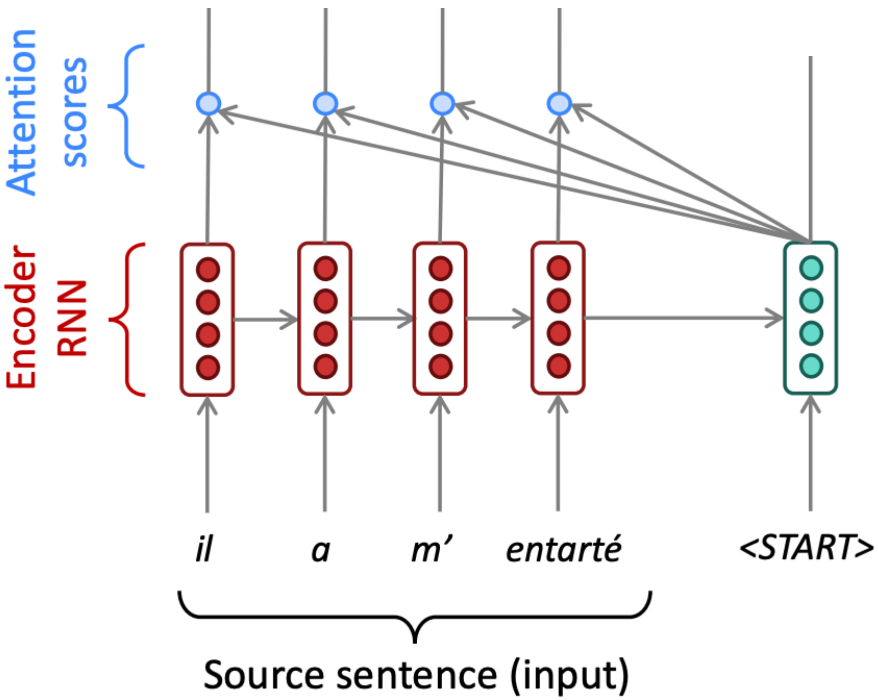




$$score(h_j, x_i) = ?$$

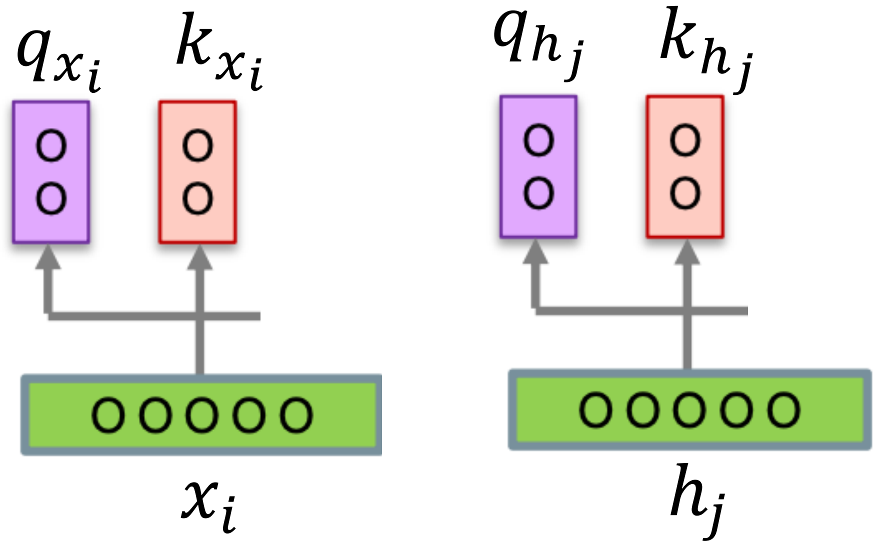
Attention score

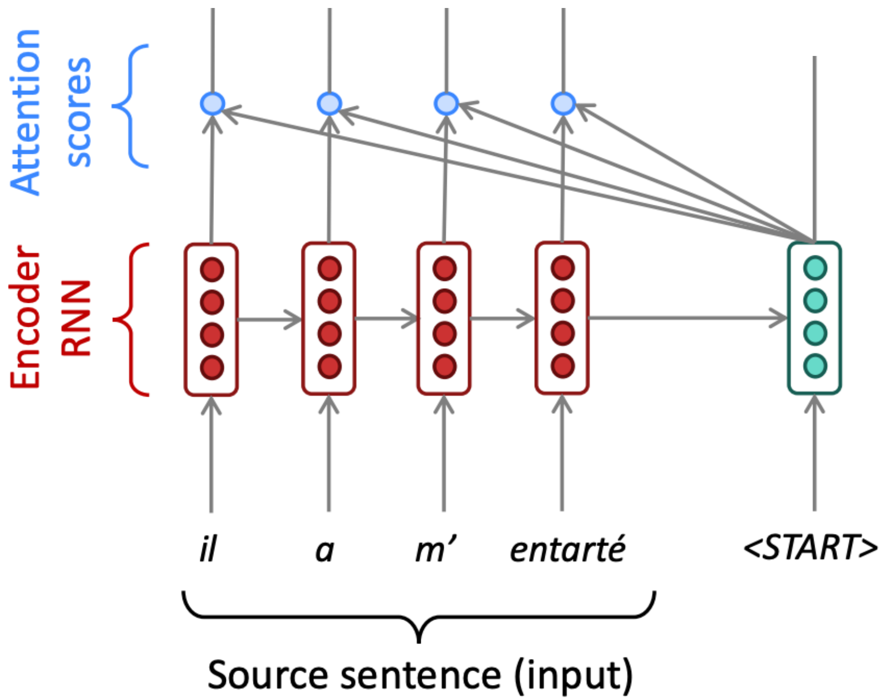




$$score(h_j, x_i) = q_{h_j} \cdot k_{x_i}$$

Attention score





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

Attention Scores

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

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

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

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

Attention Scores

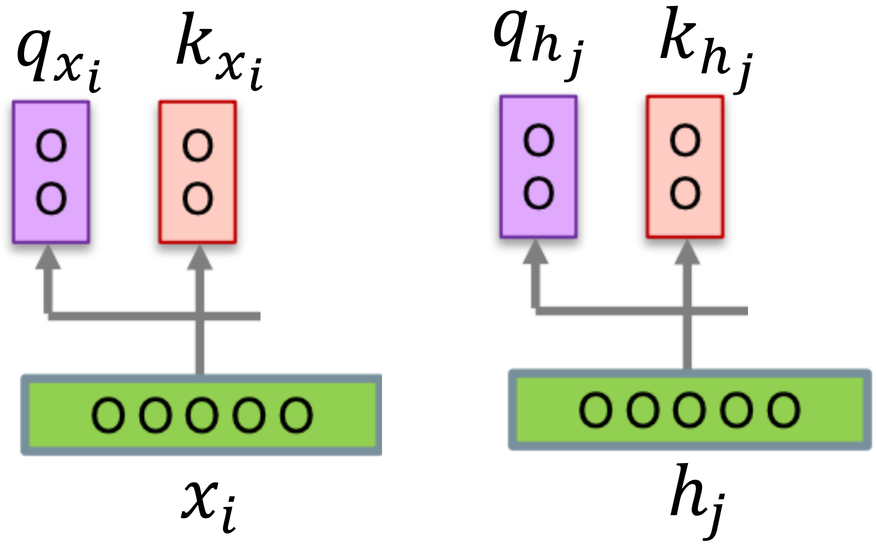
We can store all the q's and k's in a matrix as well

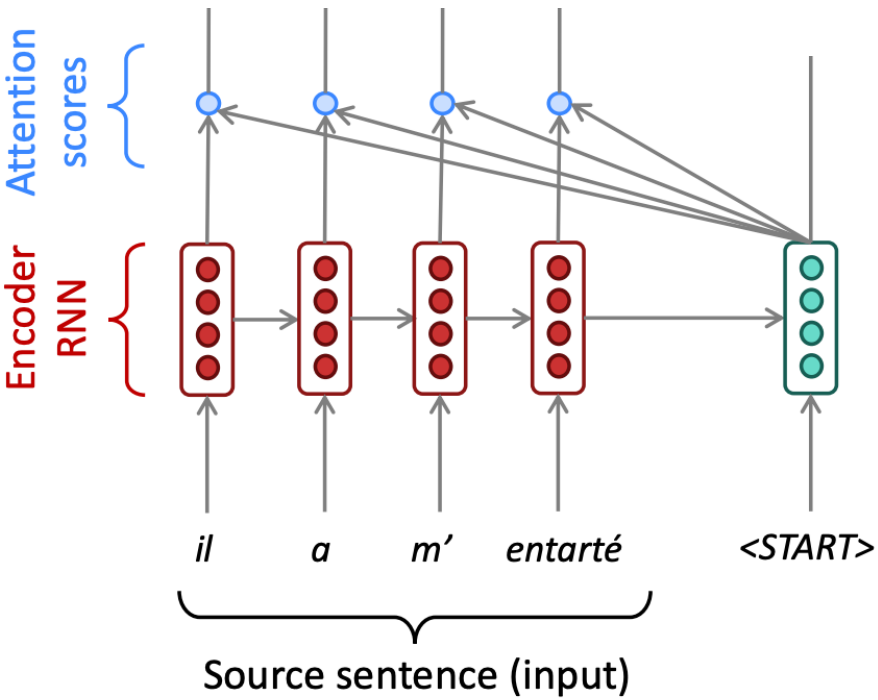
$$A_{score} = \begin{bmatrix} \alpha_{1,1} & \cdots & \alpha_{1,n} \\ \vdots & \ddots & \vdots \\ \alpha_{m,1} & \cdots & \alpha_{m,n} \end{bmatrix}$$

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

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

Attention score

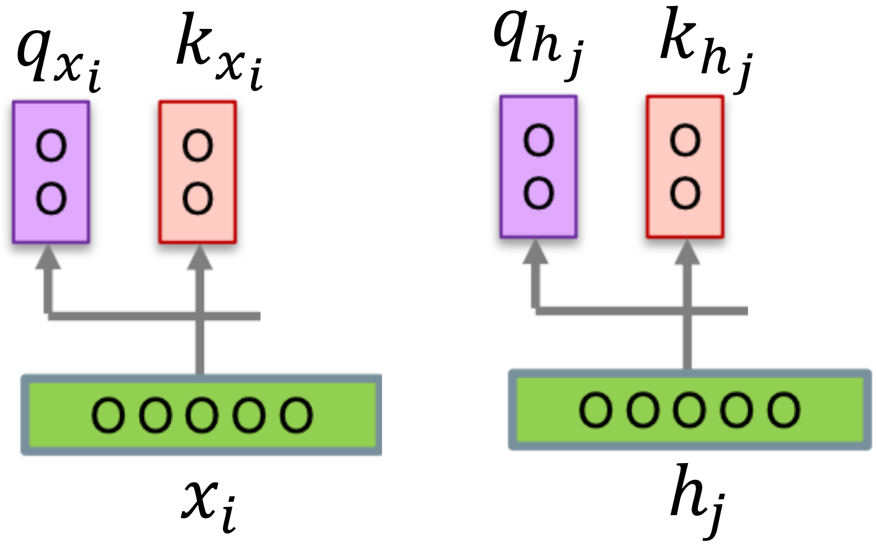


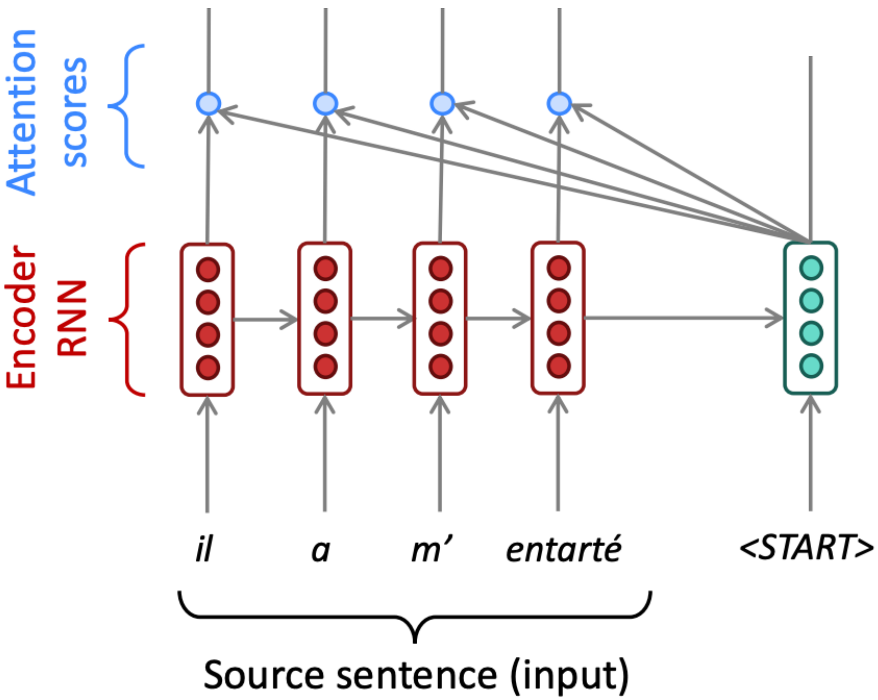


$$score(h_j, x_i) = \frac{q_{h_j} \cdot k_{x_i}}{\sqrt{d_k}}$$

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

Attention

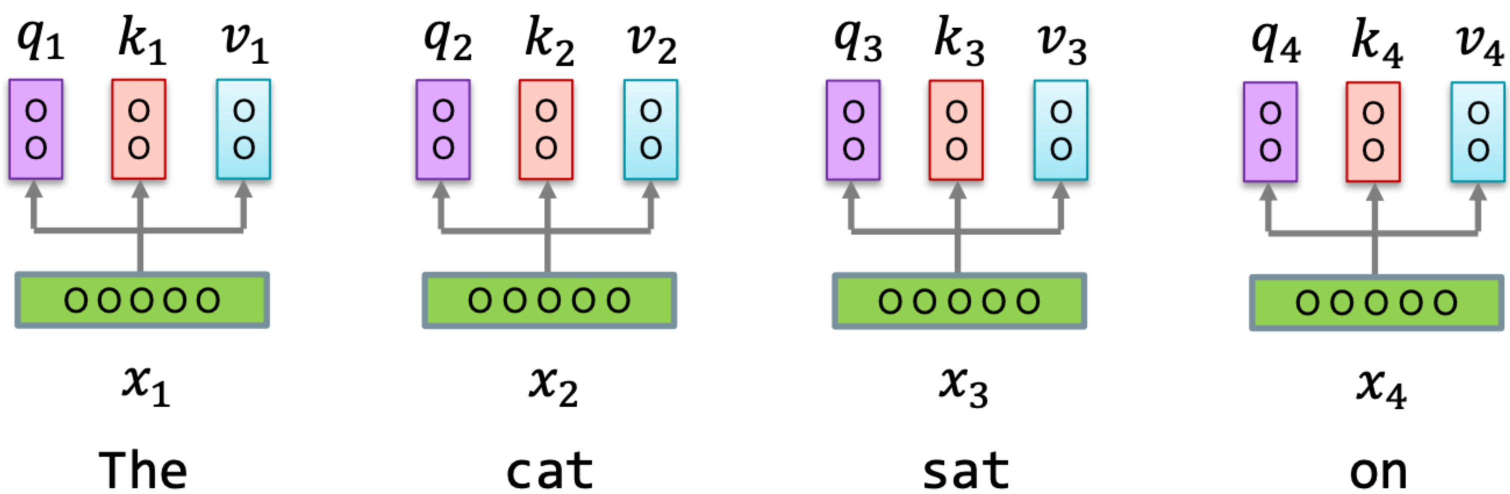




$$A = softmax(\frac{QK^T}{\sqrt{d_k}})$$

Self-attention

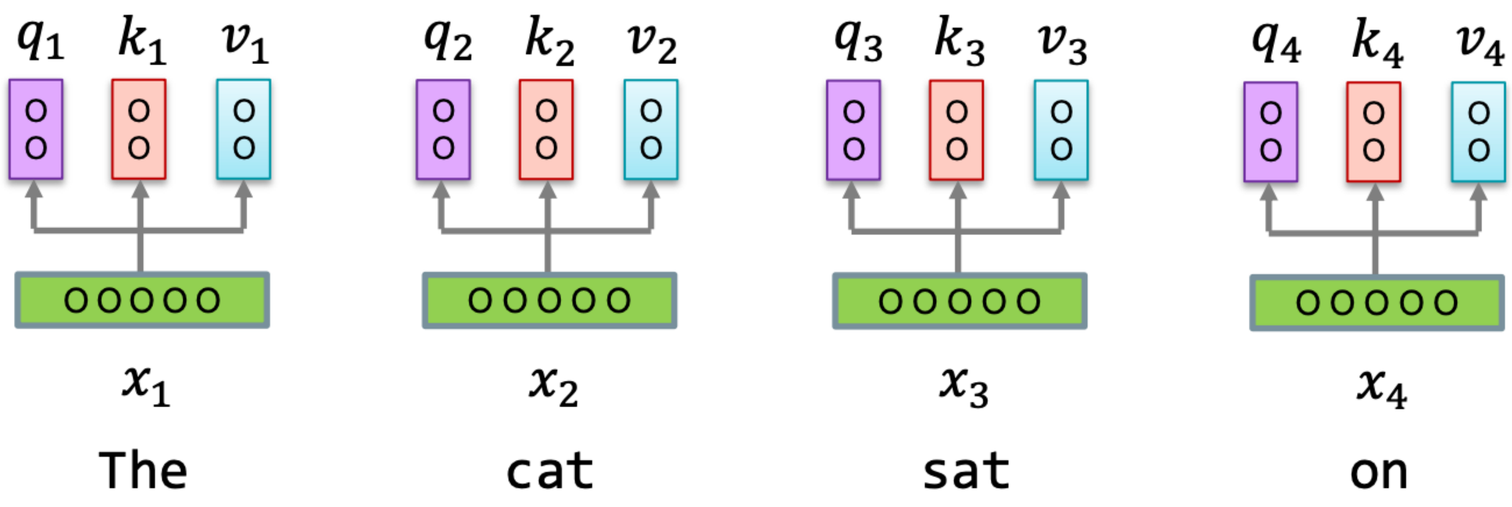
When creating a representation for x_i, how much weight/focus/attention should we give to x_j



Output of each input cell

These are three representations of each input

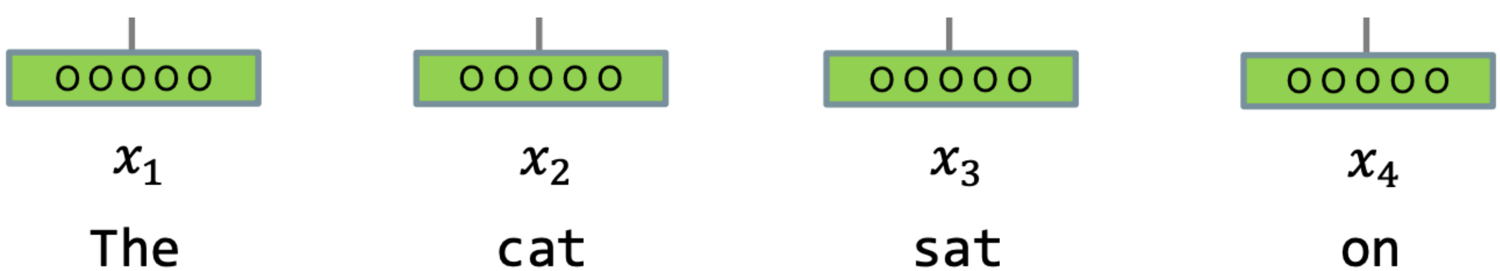
Each representation is created by multiplying the input by a weight matrix



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 $score(x_i, x_j)$

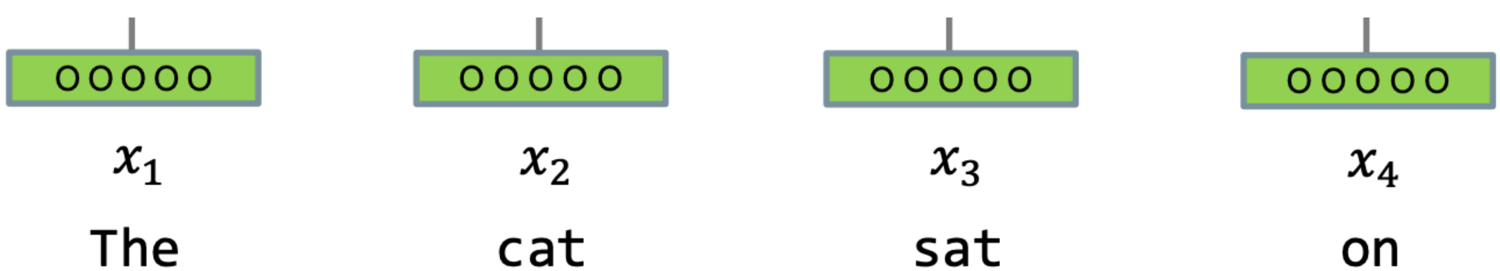


Self-Attention Scores

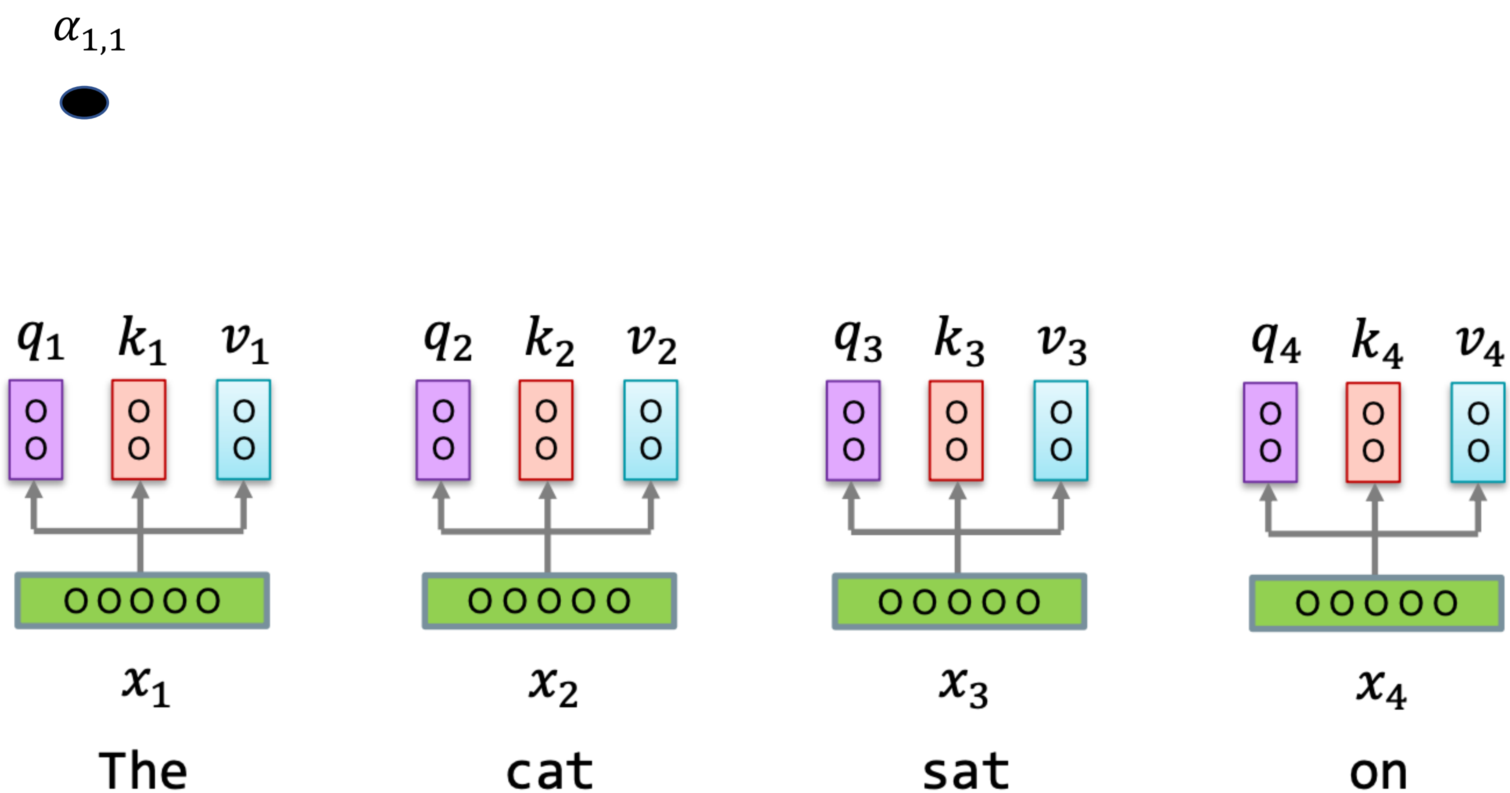
$\forall i, j \in |x|$ we must compute $score(x_i, x_j)$

Question: are these scores distance functions?

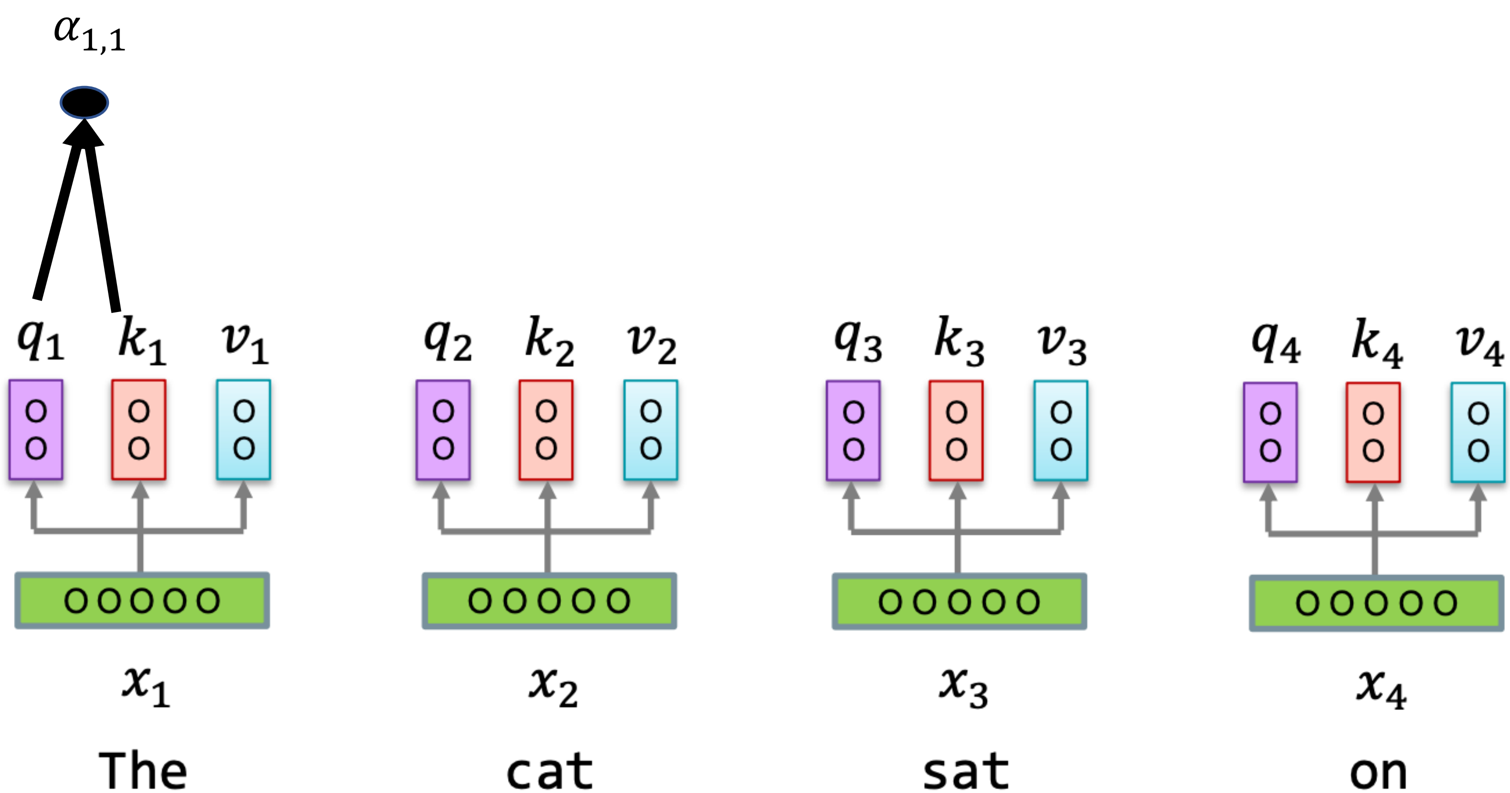
No! $score(x_i, x_j)$ shouldn't be equal to $score(x_i, x_j)$



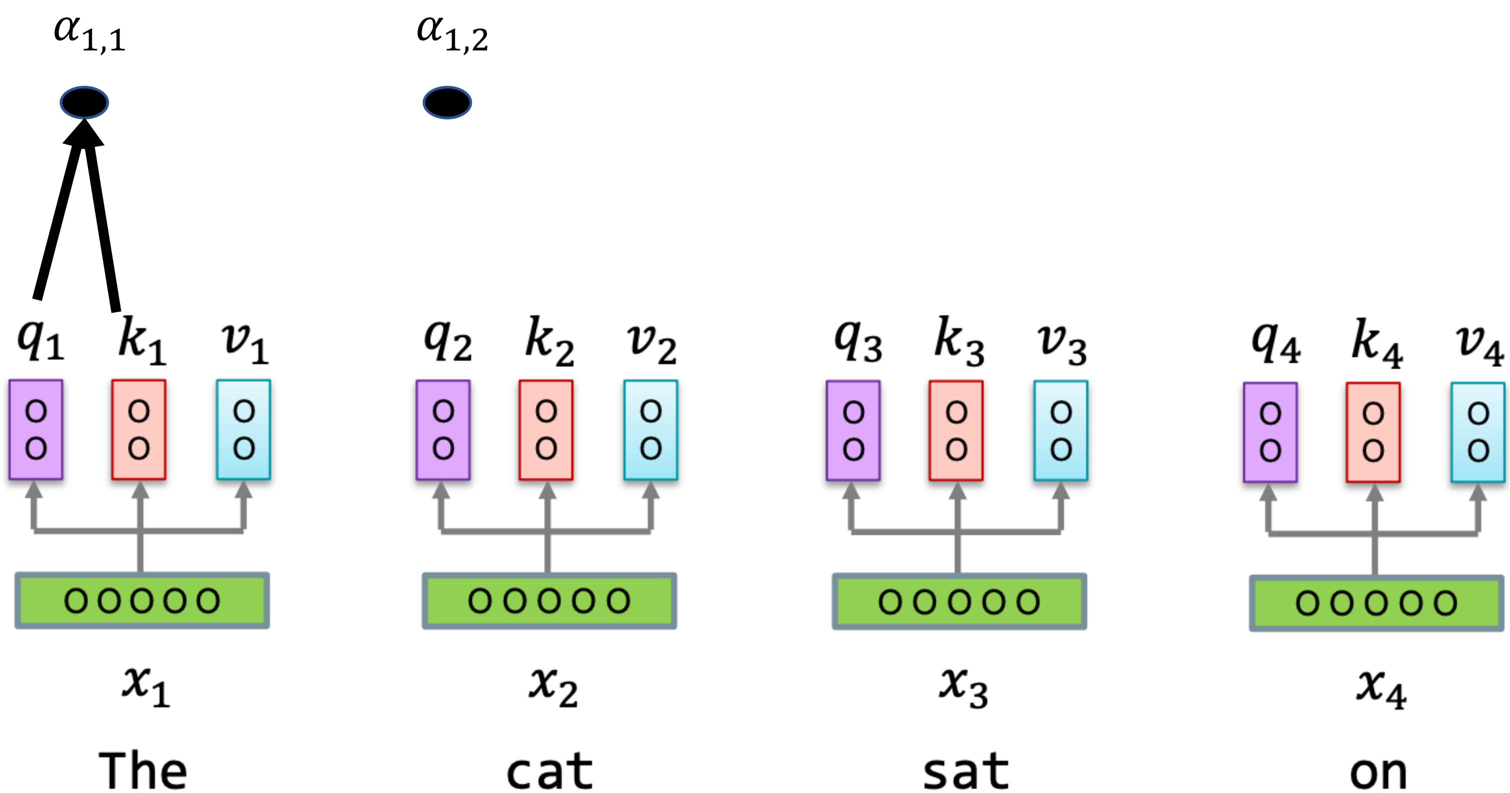
Self-attention



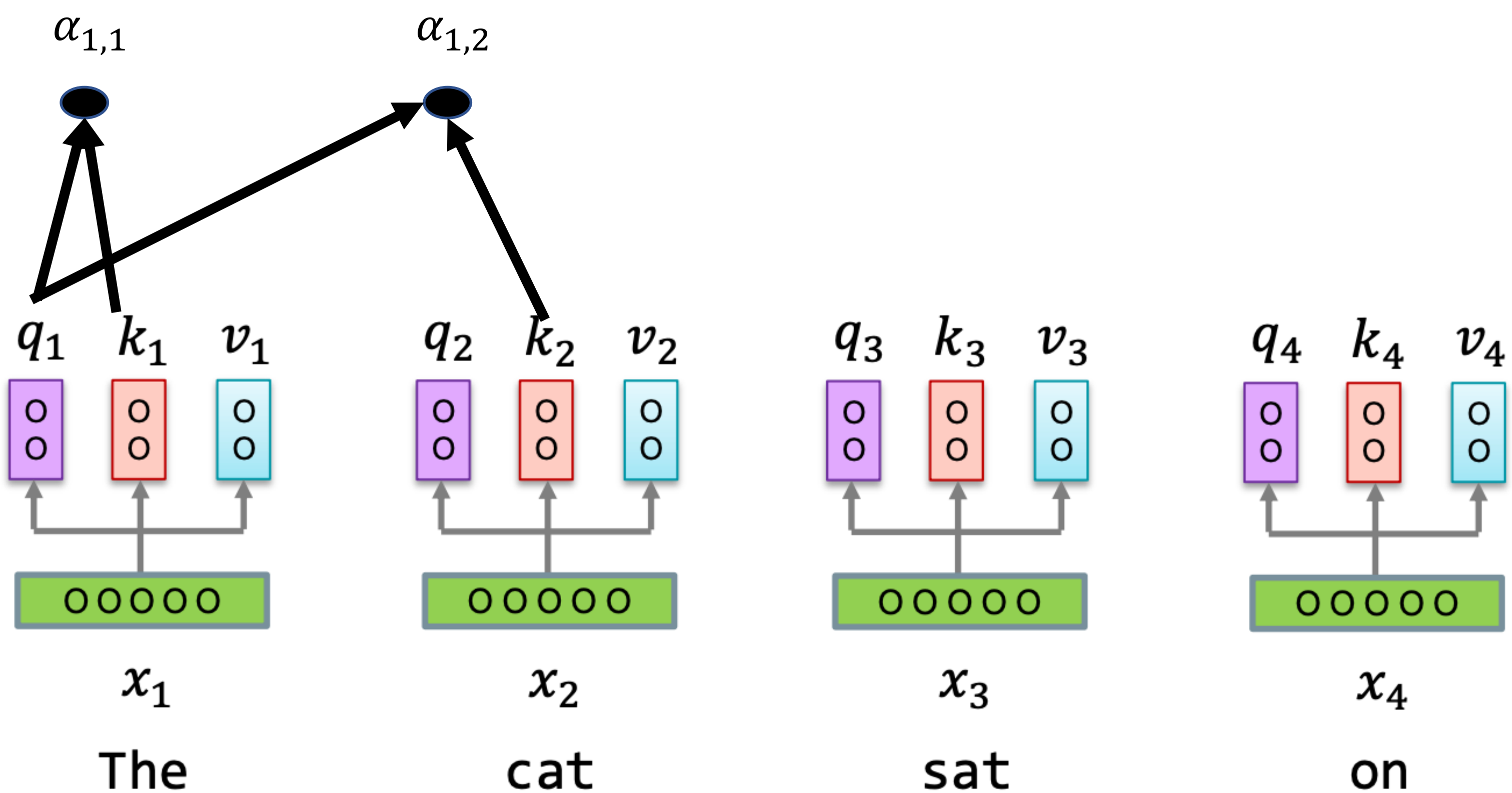
Self-attention



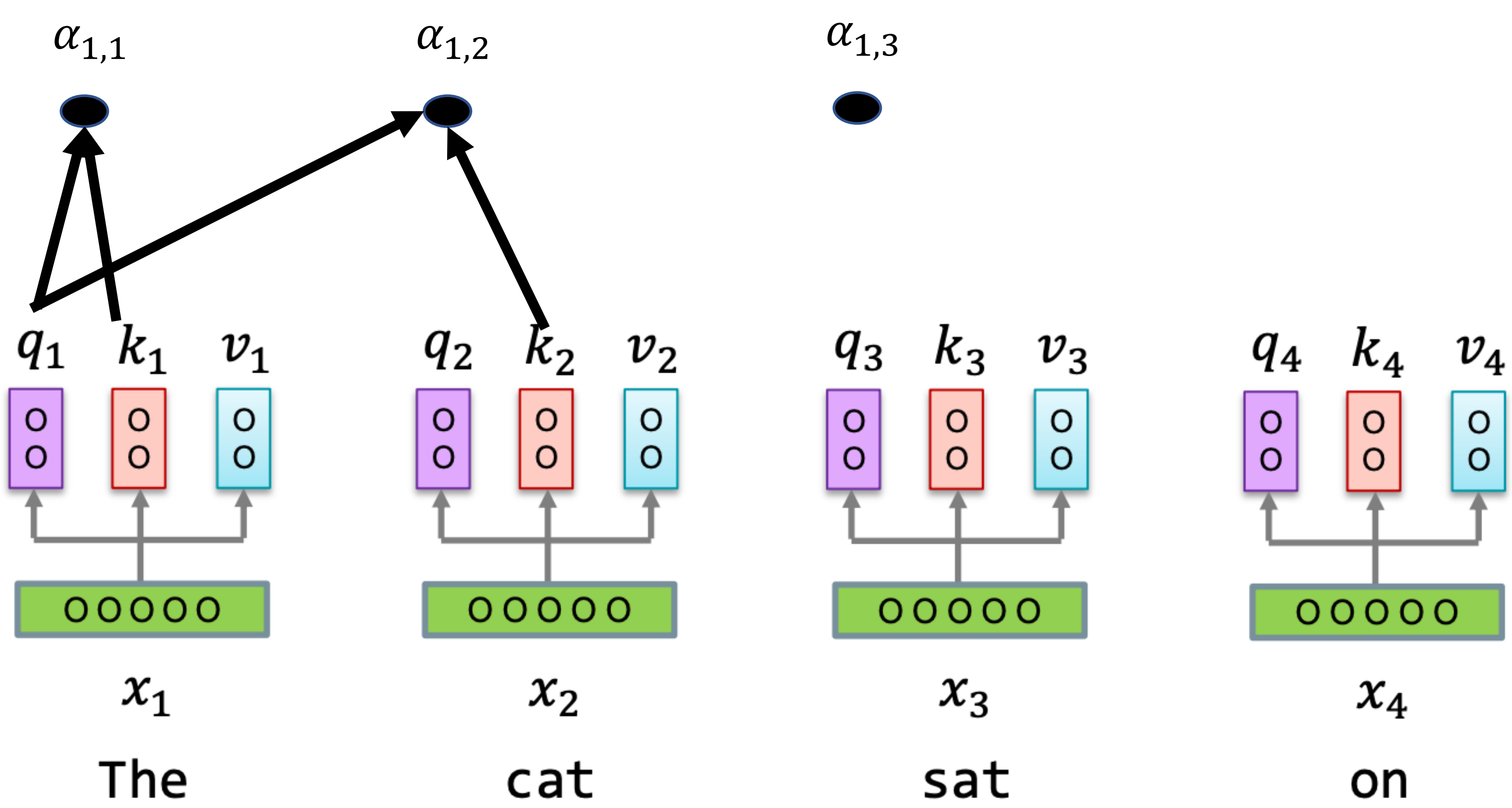
Self-attention



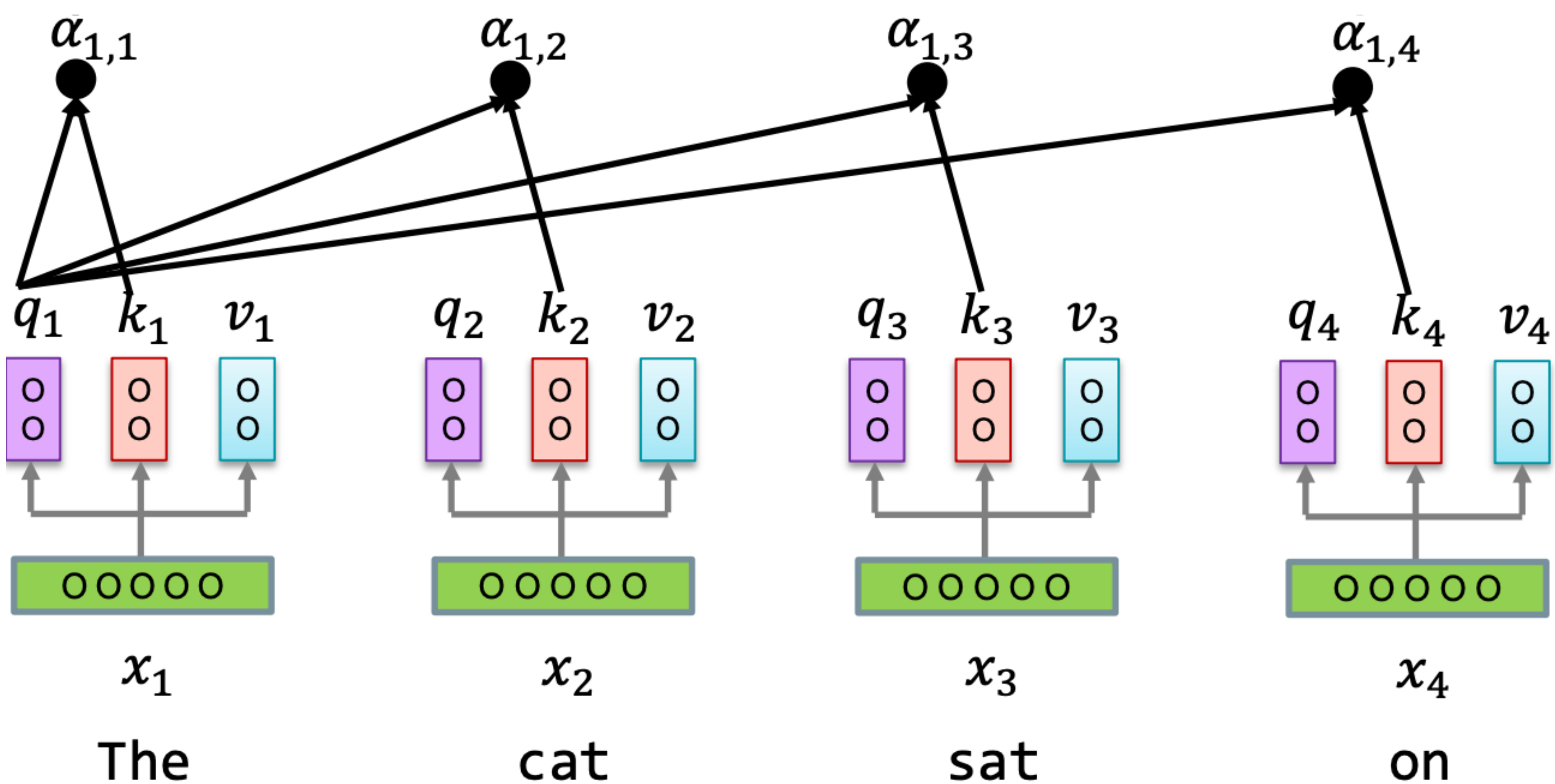
Self-attention



Self-attention

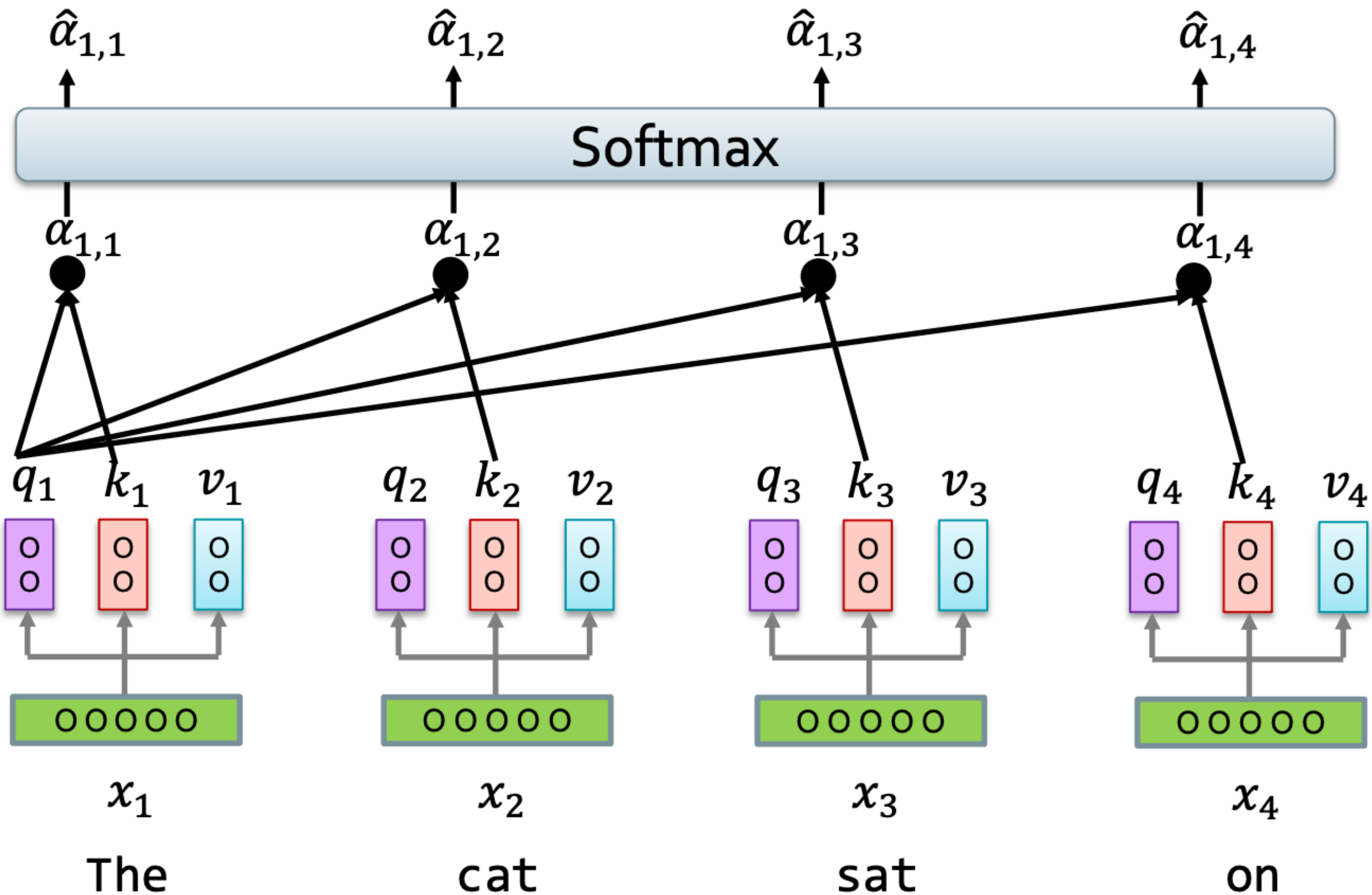


Self-attention

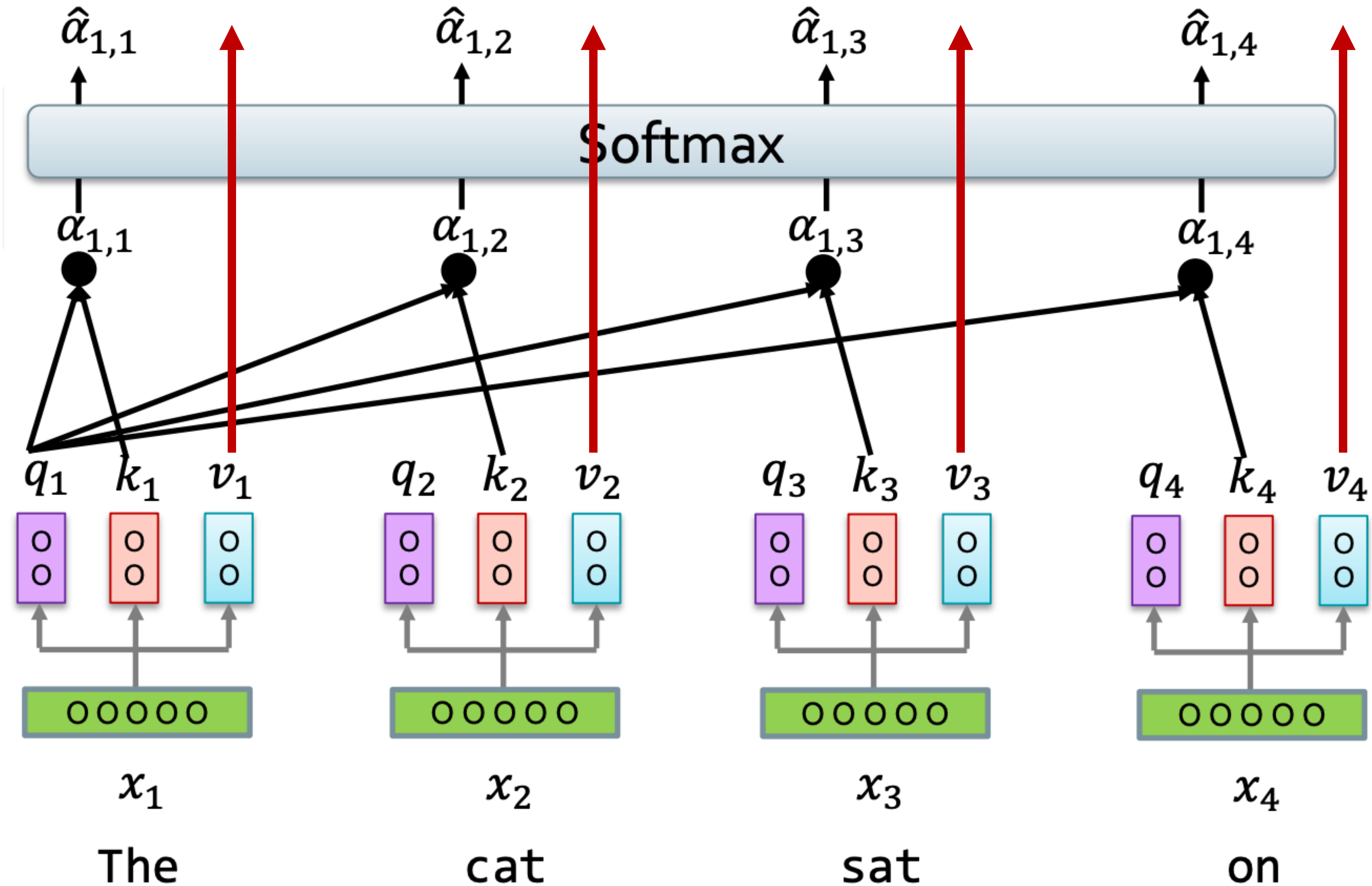


Daniel Khashabi

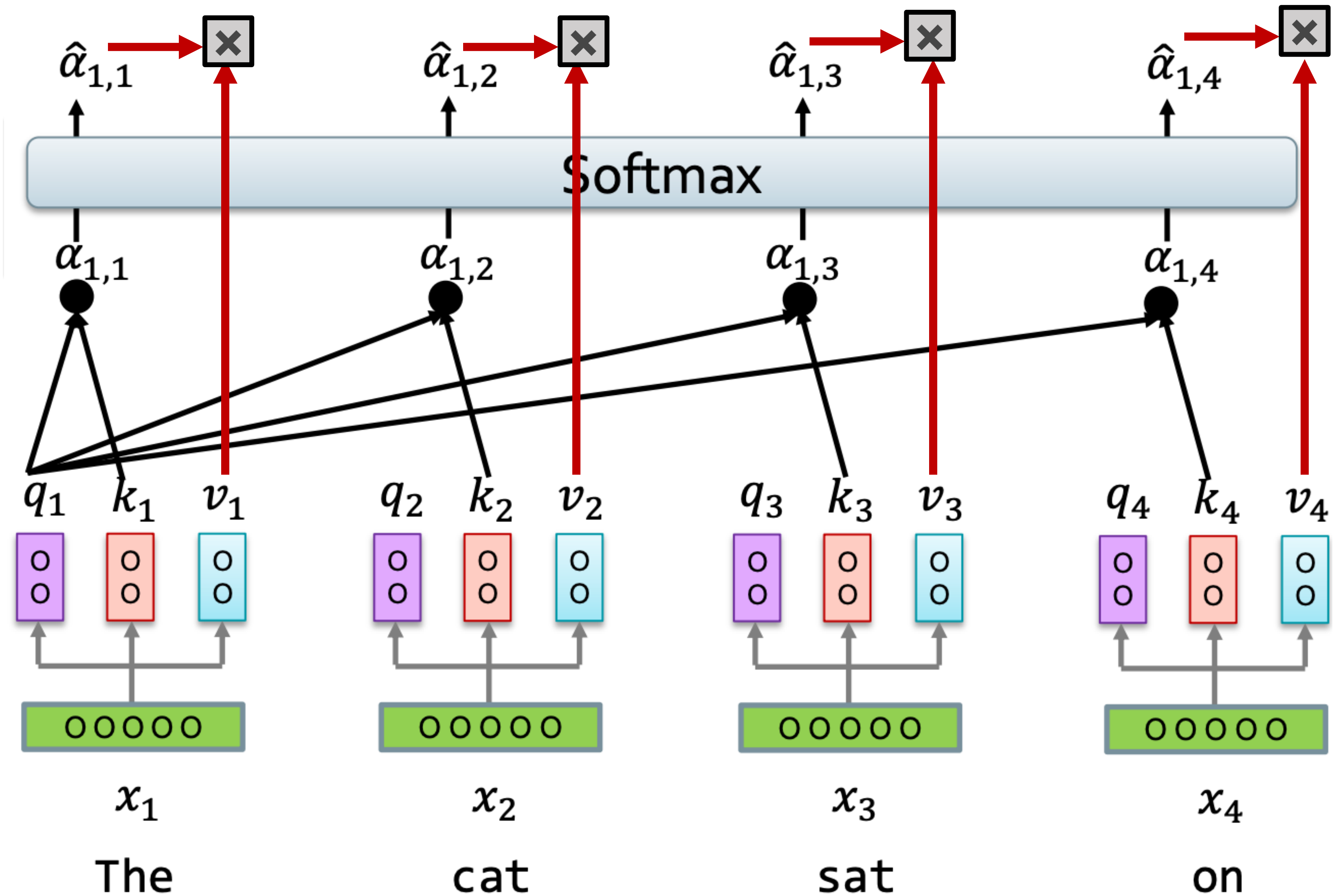
Self-attention



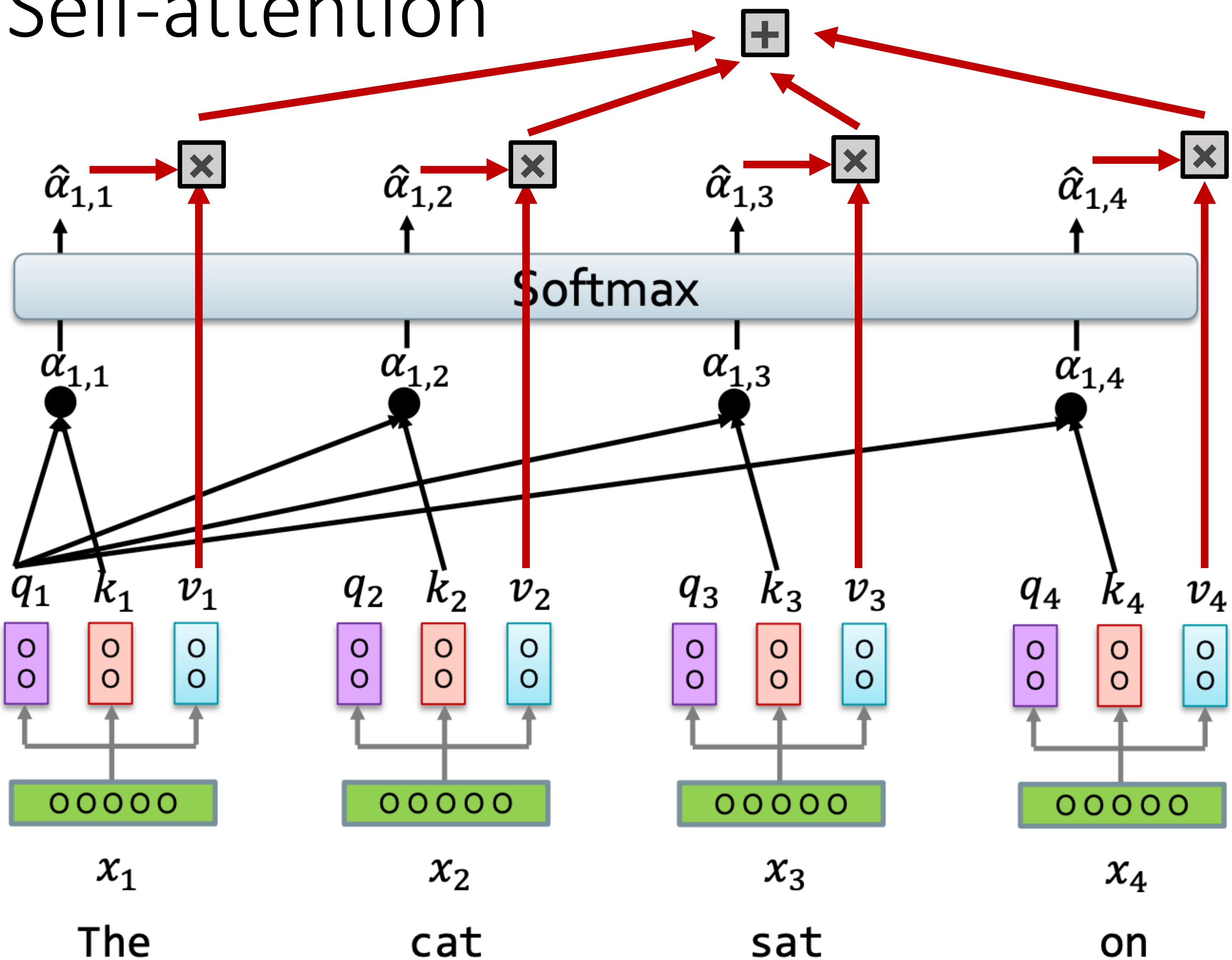
Self-attention



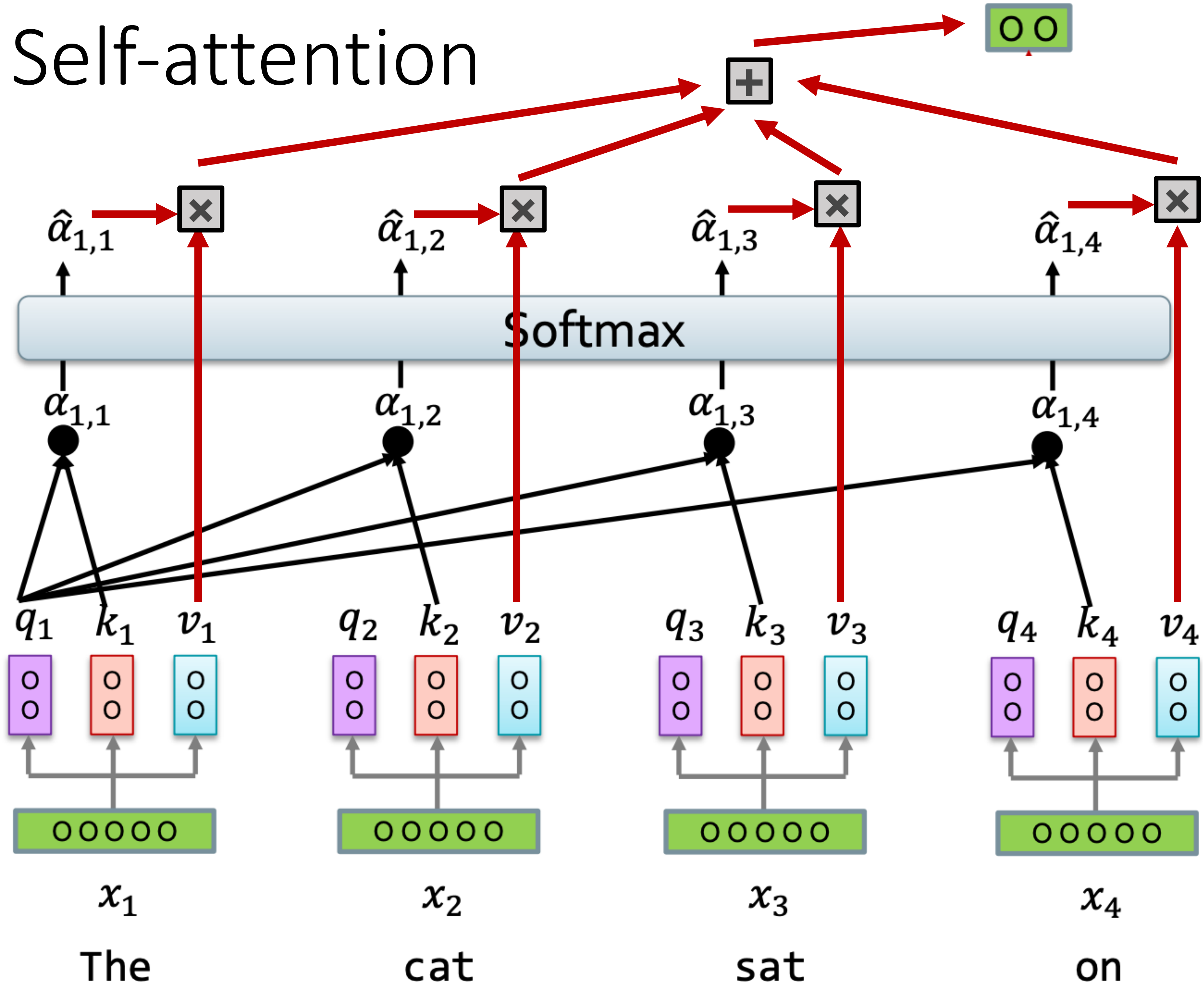
Self-attention



Self-attention

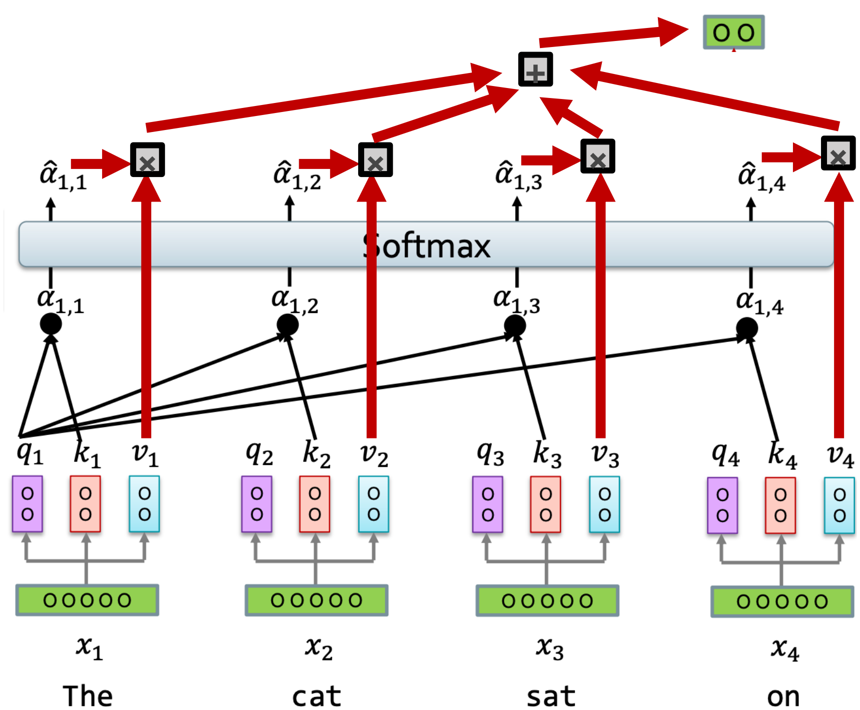


Self-attention



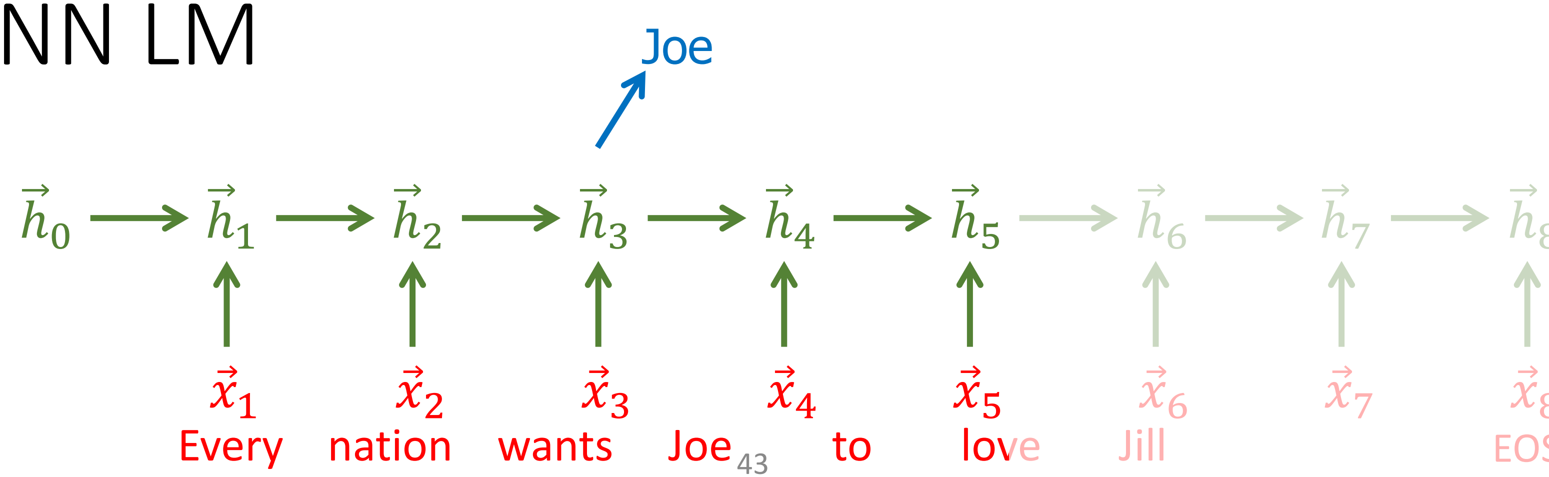
Self-attention

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



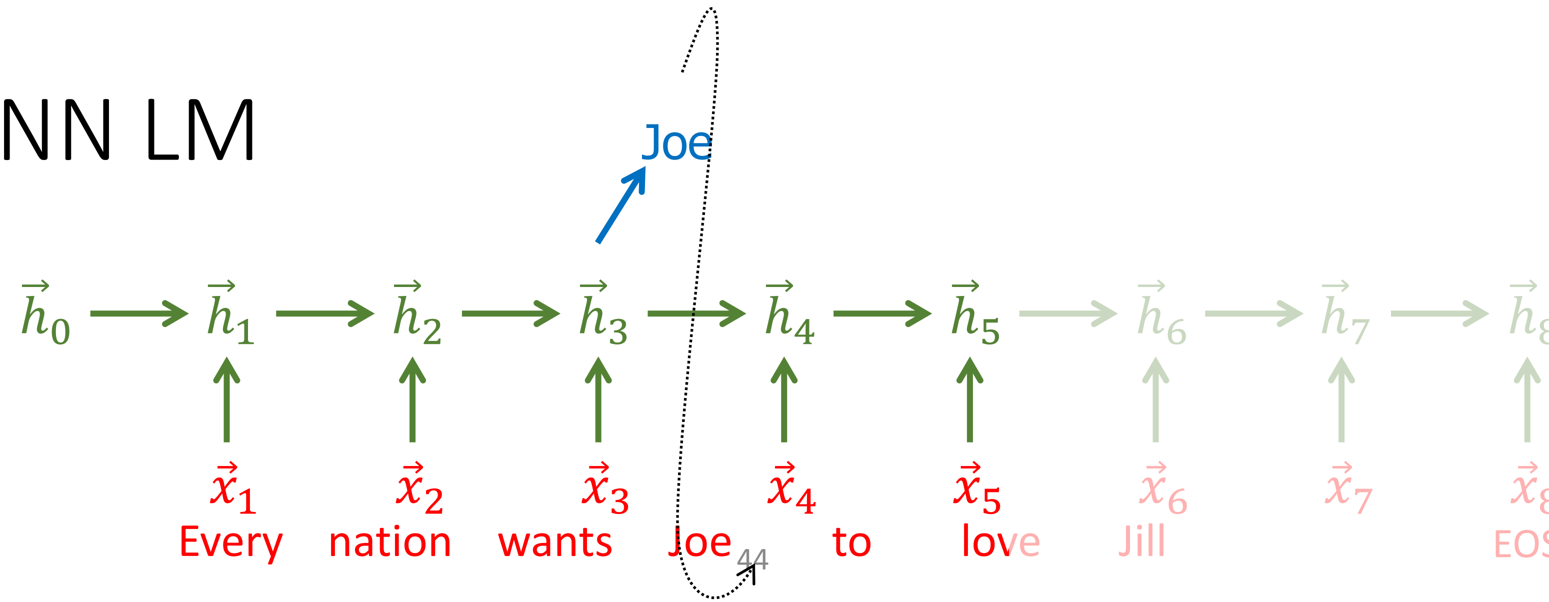
This is the main idea behind a **transformer**

RNN LM



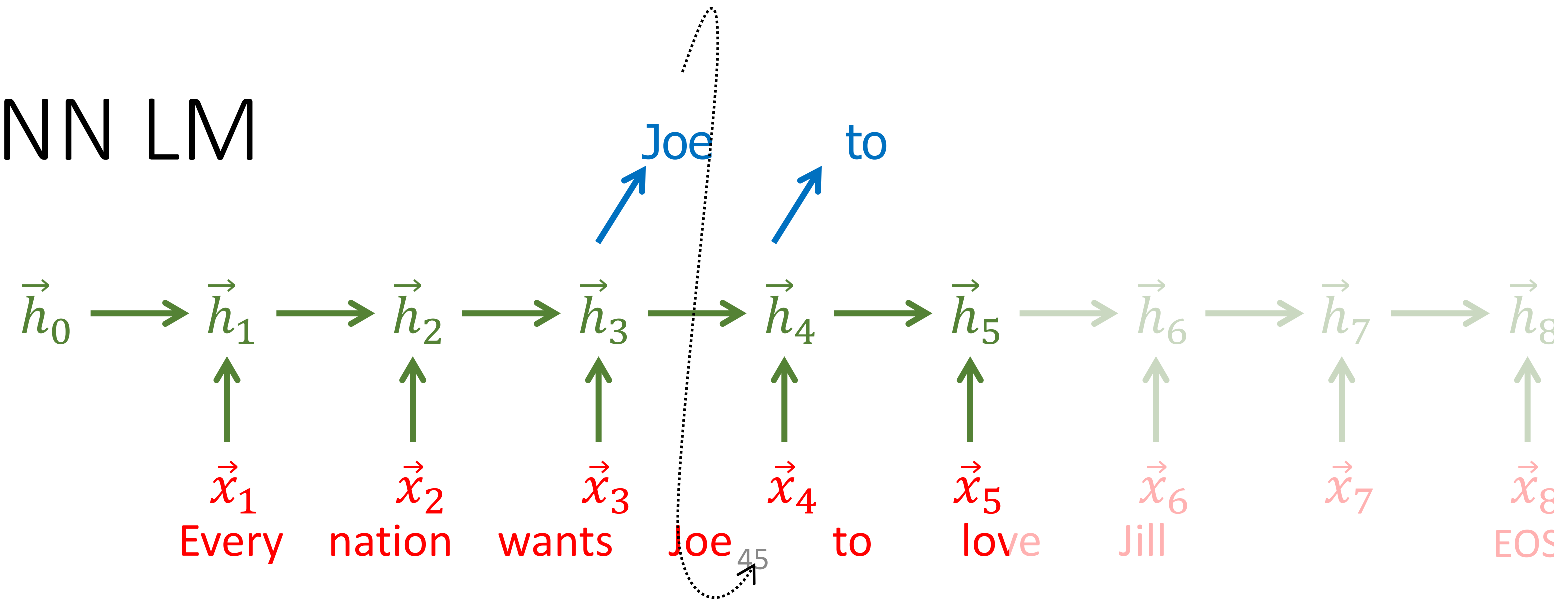
Joe
$\vec{h}_0$ $\vec{h}_1$ $\vec{h}_2$ $\vec{h}_3$ $\vec{h}_4$ $\vec{h}_5$ $\vec{h}_6$ $\vec{h}_7$ $\vec{h}_8$
$\vec{x}_1$ $\vec{x}_2$ $\vec{x}_3$ $\vec{x}_4$ $\vec{x}_5$ $\vec{x}_6$ $\vec{x}_7$ $\vec{x}_8$
Every nation wants Joe to love Jill EOS

RNN LM

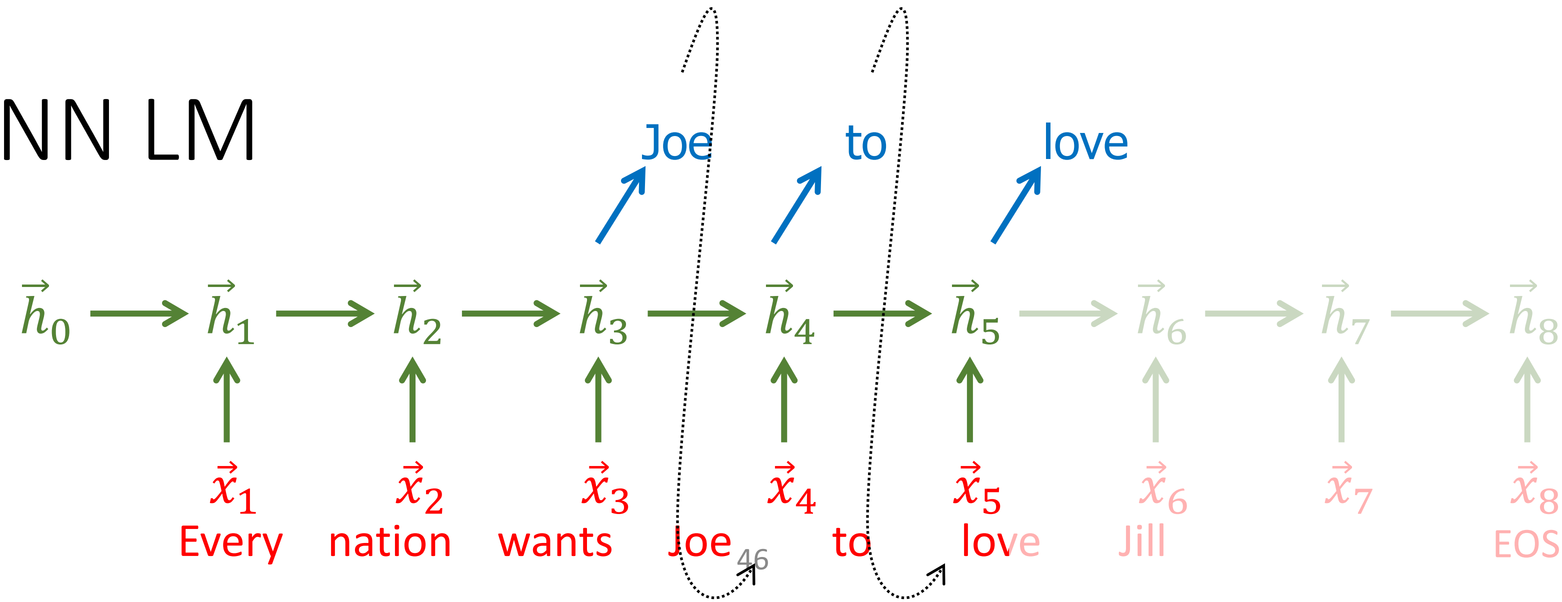


Joe
$\vec{h}_0$ $\vec{h}_1$ $\vec{h}_2$ $\vec{h}_3$ $\vec{h}_4$ $\vec{h}_5$ $\vec{h}_6$ $\vec{h}_7$ $\vec{h}_8$
$\vec{x}_1$ $\vec{x}_2$ $\vec{x}_3$ $\vec{x}_4$ $\vec{x}_5$ $\vec{x}_6$ $\vec{x}_7$ $\vec{x}_8$
Every nation wants Joe to love Jill EOS

RNN LM

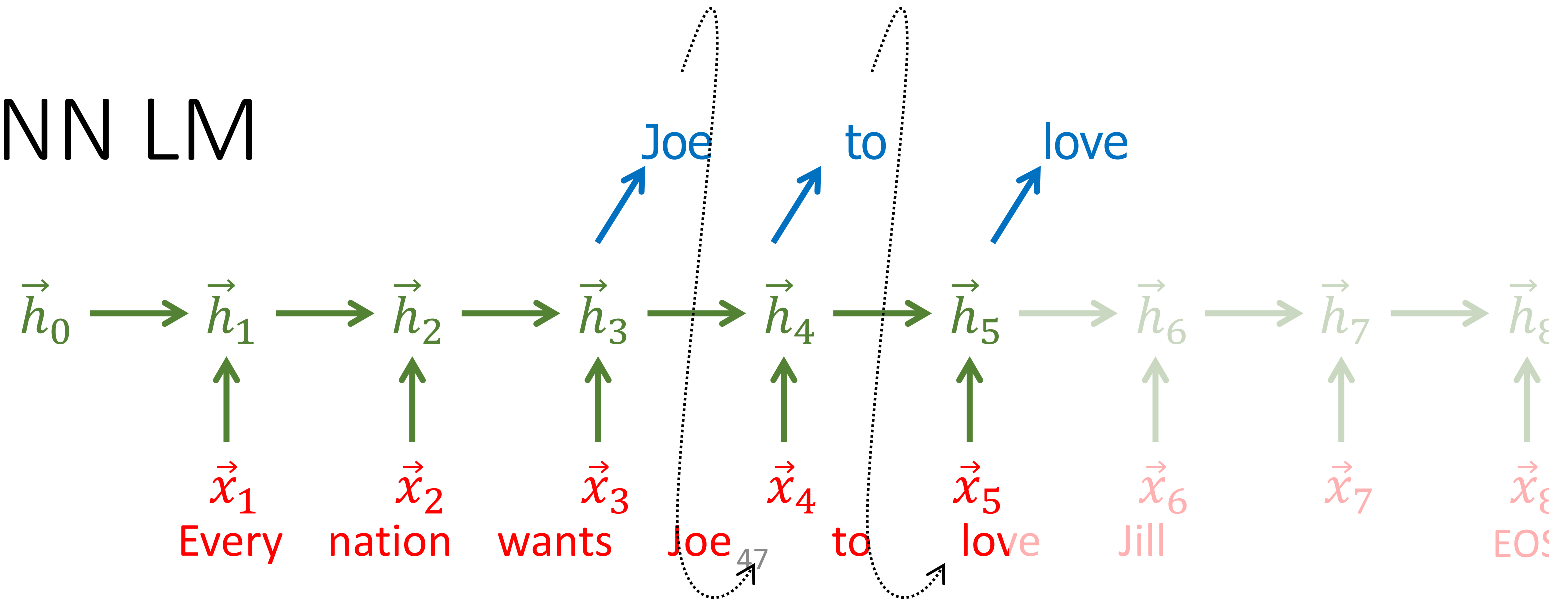

Joe
to
$\vec{h}_0$ $\vec{h}_1$ $\vec{h}_2$ $\vec{h}_3$ $\vec{h}_4$ $\vec{h}_5$ $\vec{h}_6$ $\vec{h}_7$ $\vec{h}_8$
$\vec{x}_1$ $\vec{x}_2$ $\vec{x}_3$ $\vec{x}_4$ $\vec{x}_5$ $\vec{x}_6$ $\vec{x}_7$ $\vec{x}_8$
Every nation wants Joe to love Jill EOS

RNN LM

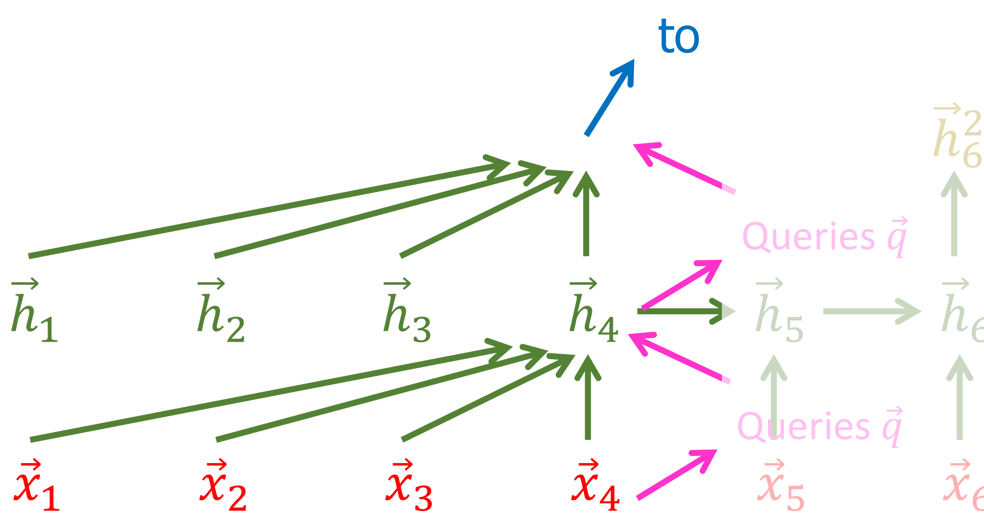


Joe
to
love
$\vec{h}_0$
$\vec{h}_1$
$\vec{h}_2$
$\vec{h}_3$
$\vec{h}_4$
$\vec{h}_5$
$\vec{h}_6$
$\vec{h}_7$
$\vec{h}_8$
$\vec{x}_1$
$\vec{x}_2$
$\vec{x}_3$
$\vec{x}_4$
$\vec{x}_5$
$\vec{x}_6$
$\vec{x}_7$
$\vec{x}_8$
Every
nation
wants
Joe
to
love
Jill
EOS

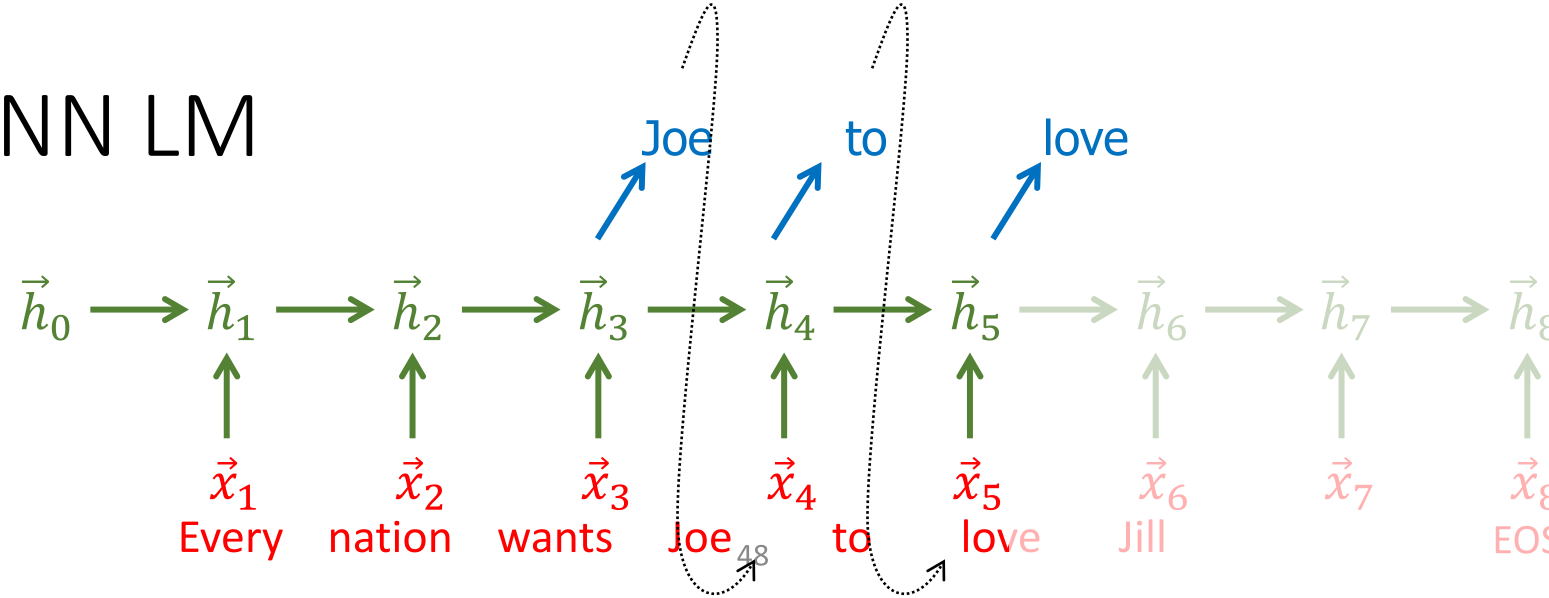
RNN LM



Transformer (self-attention) LM

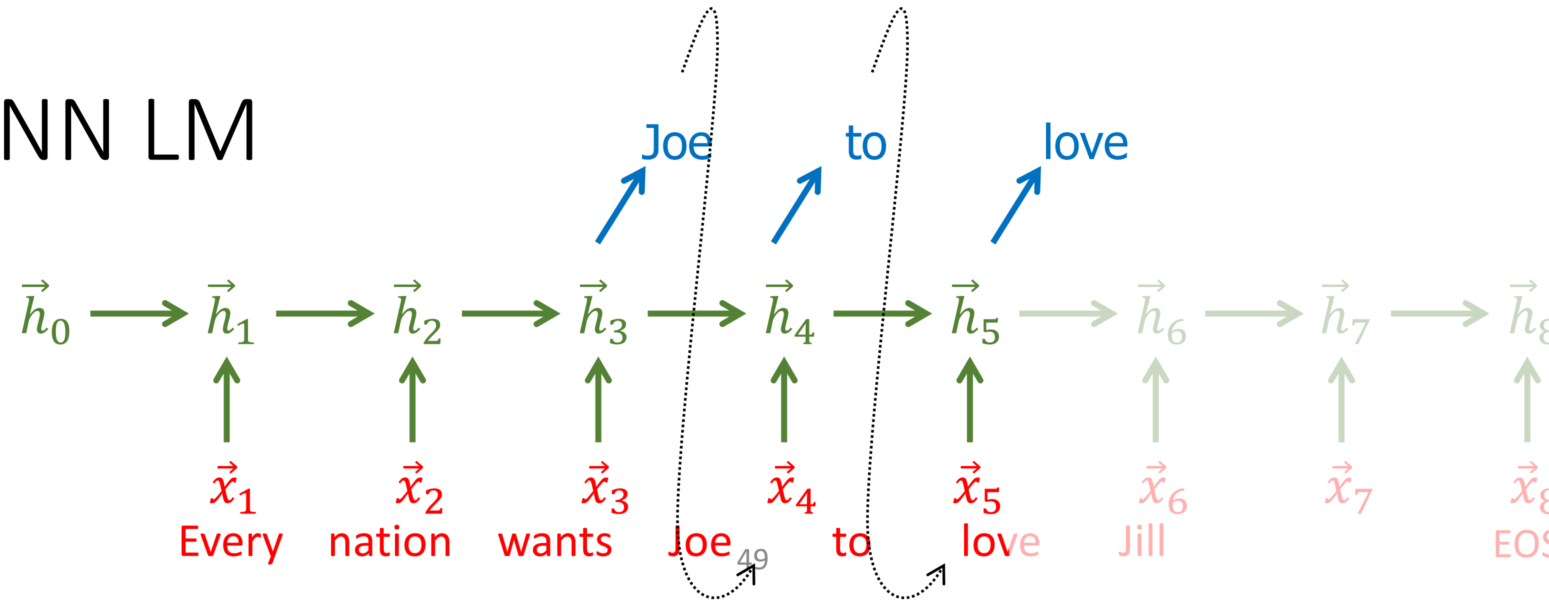


RNN LM

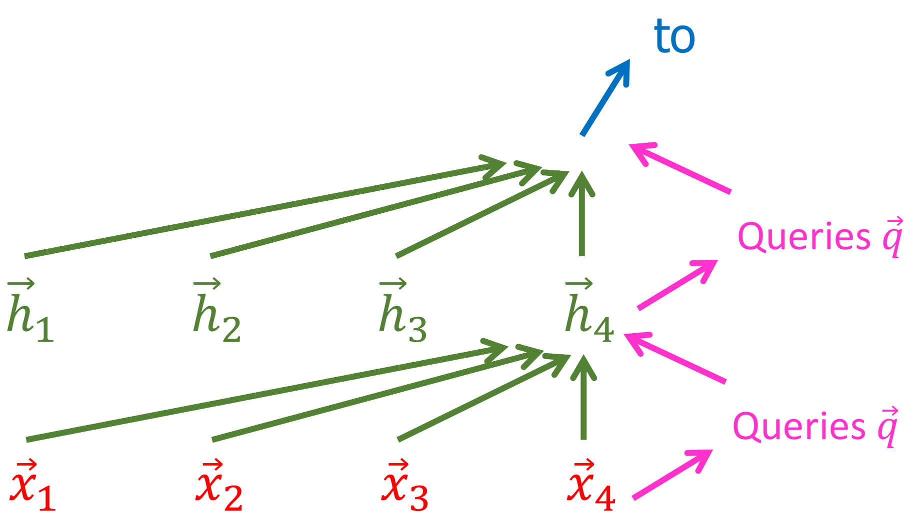


Transformer (self-attention) LM

RNN LM

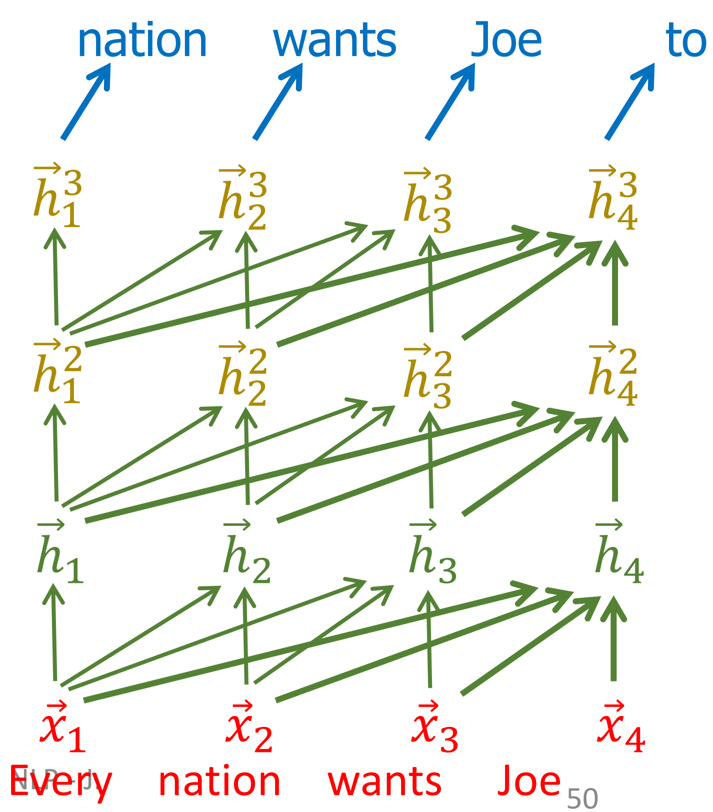


Transformer (self-attention) LM



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

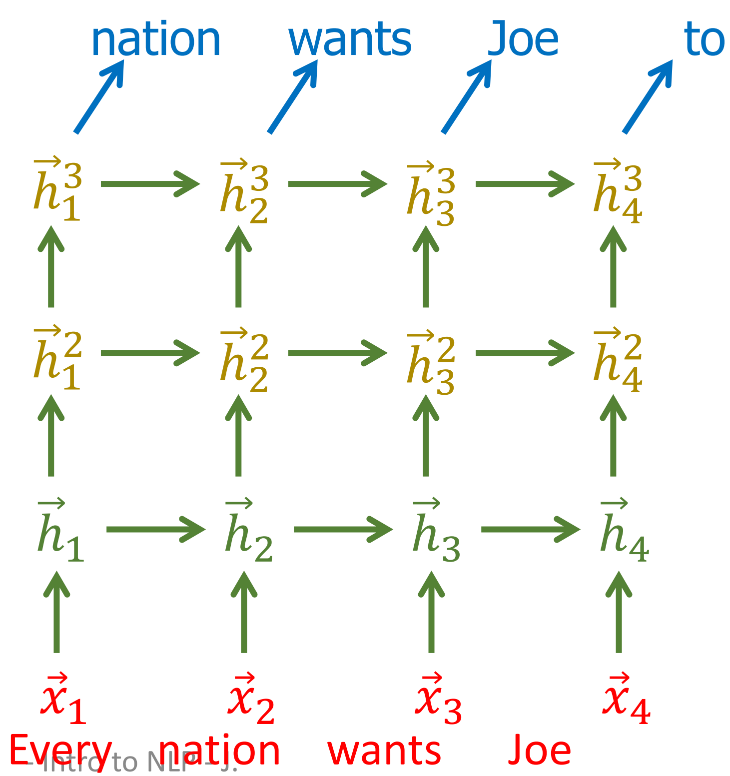


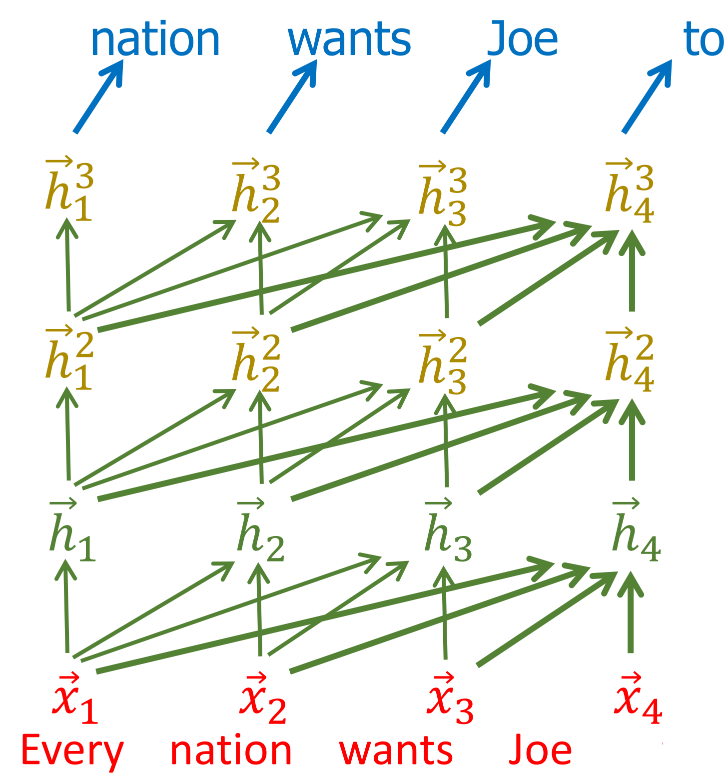
(oops, to predict the very first word, we needed $\vec{x}_0 = < s >$! It's missing from our diagrams.)

RNN vs. Transformer

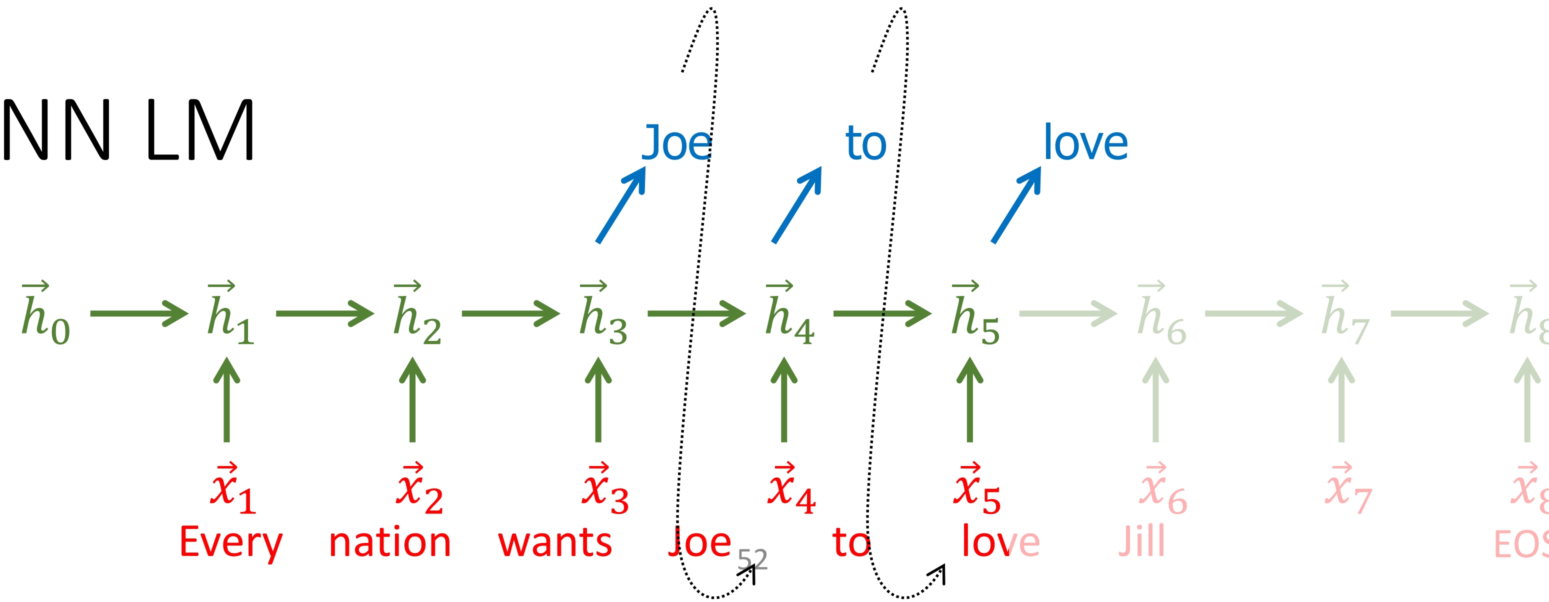
Training, on GPU, per layer ↓

	RNN	Transformer
Computations:	☺ O(n)	☹ O(n^2)
# serial steps:	☹ O(n) due to →	☺ O(1): all ↗↑ in parallel + O(log n) to sum n inputs

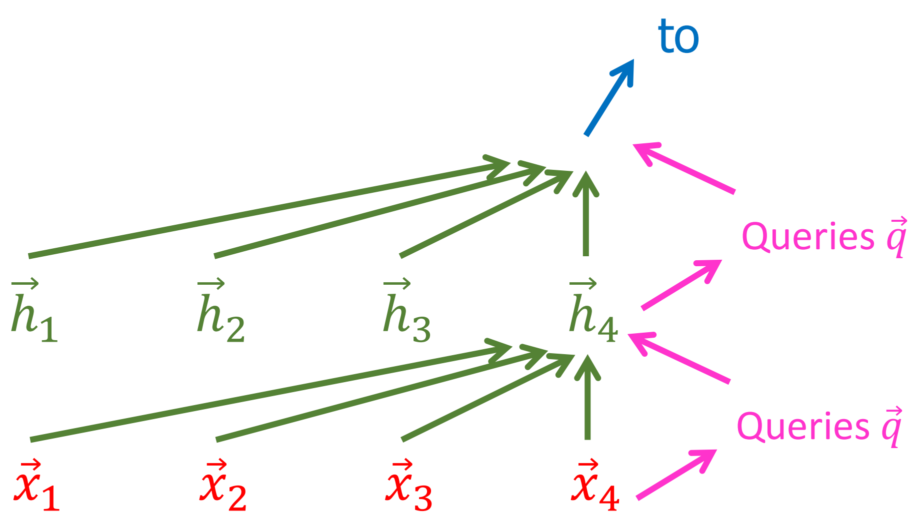




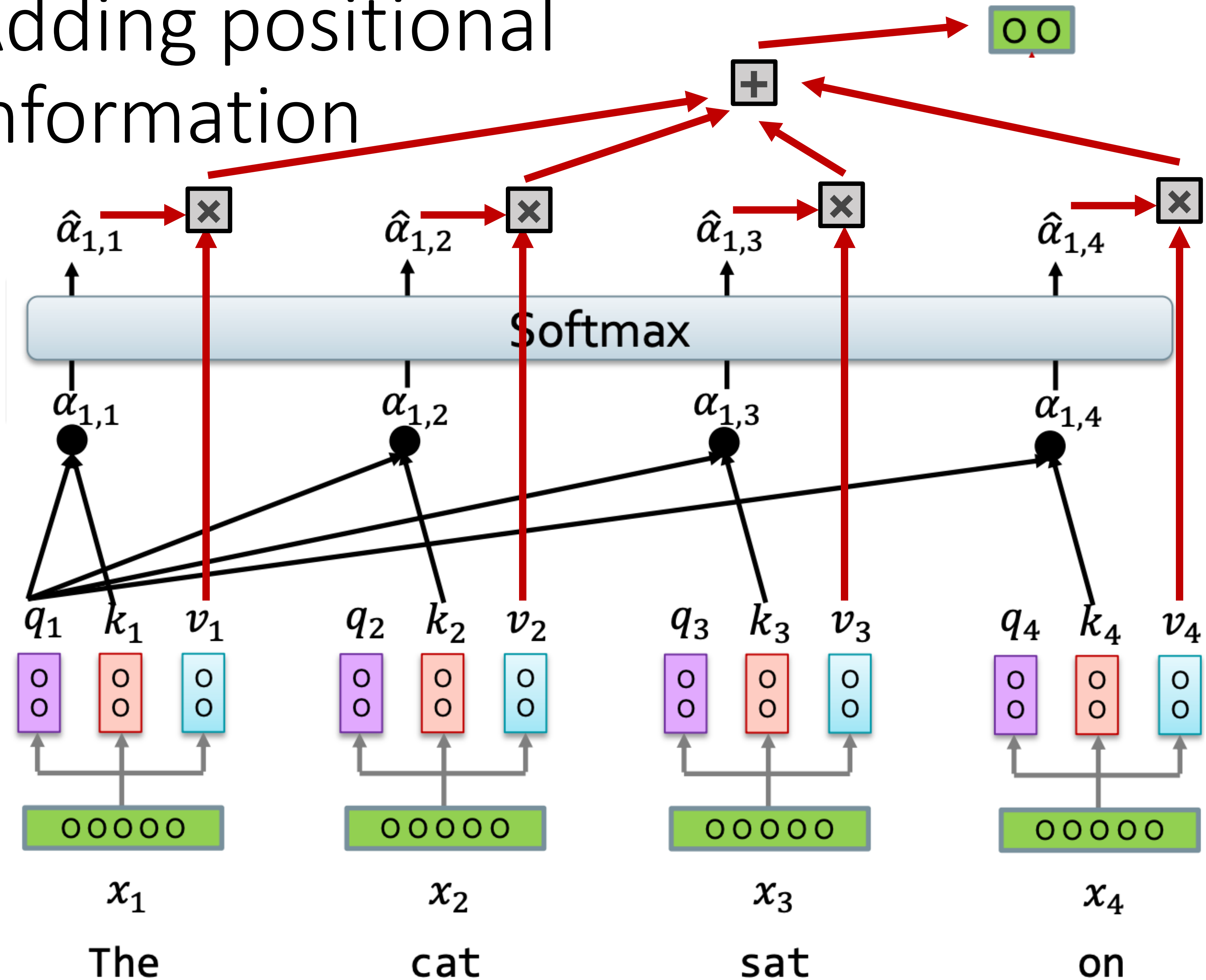
RNN LM



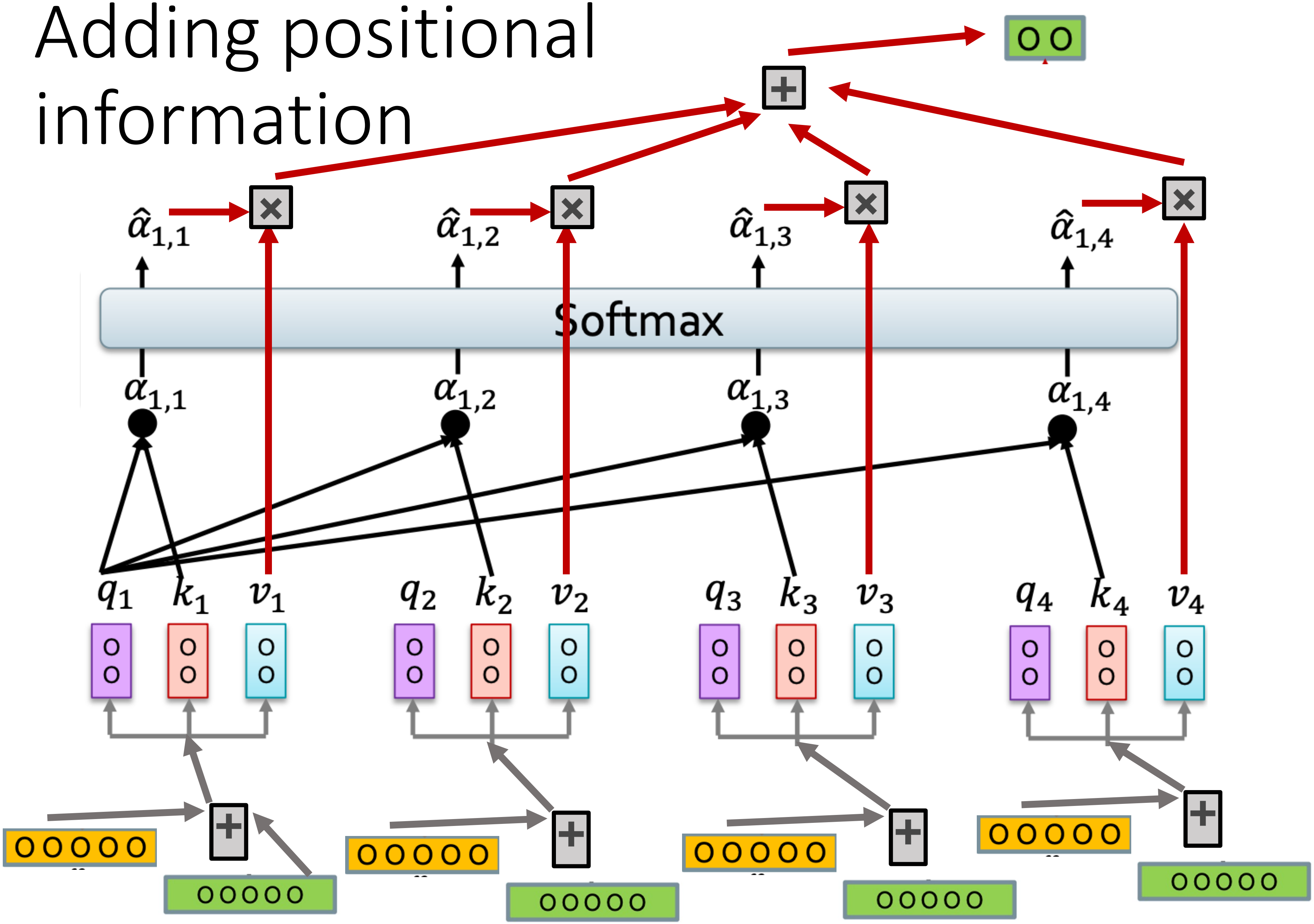
Transformer (self-attention) LM



Adding positional information

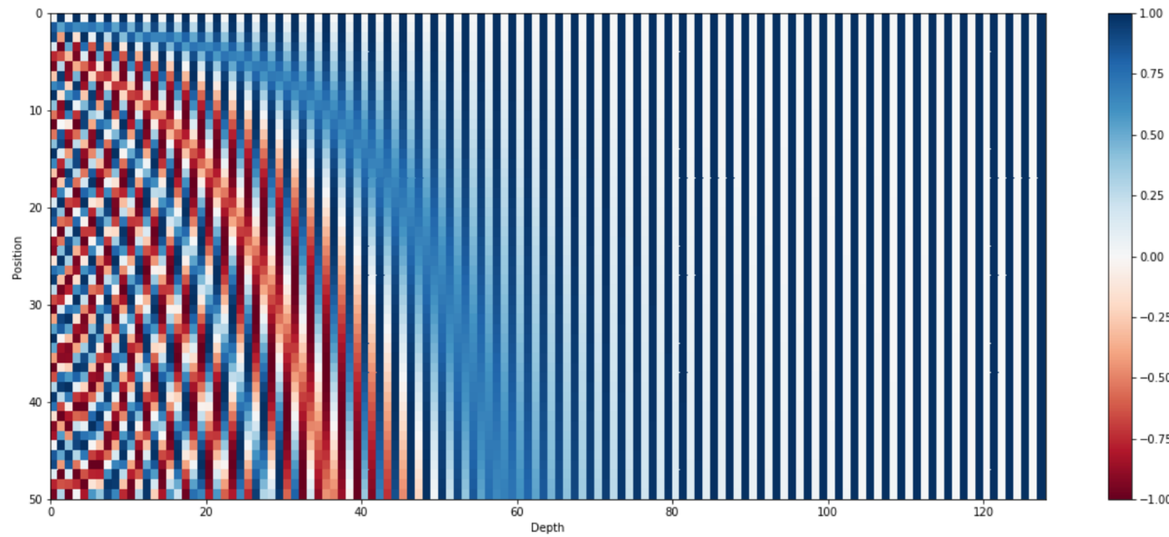


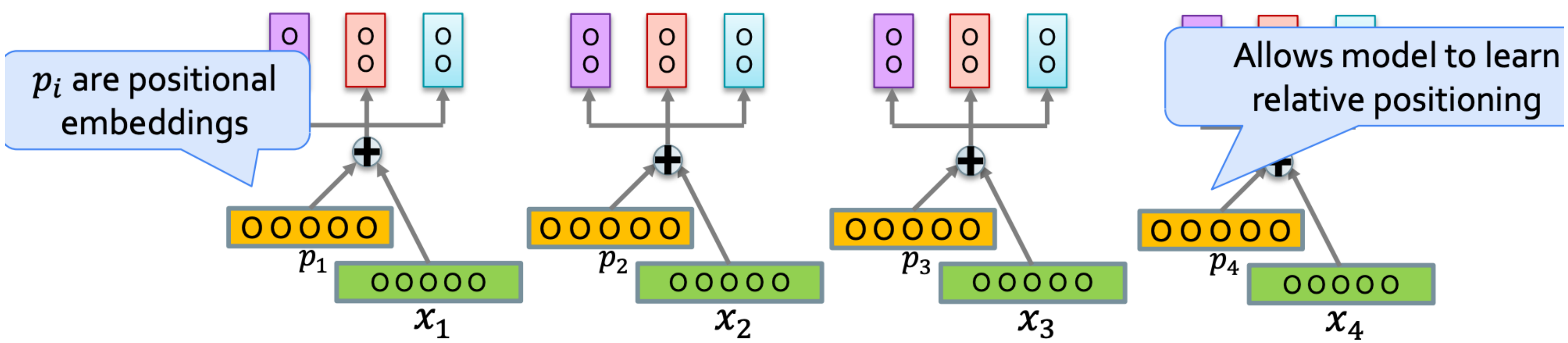
Adding positional information



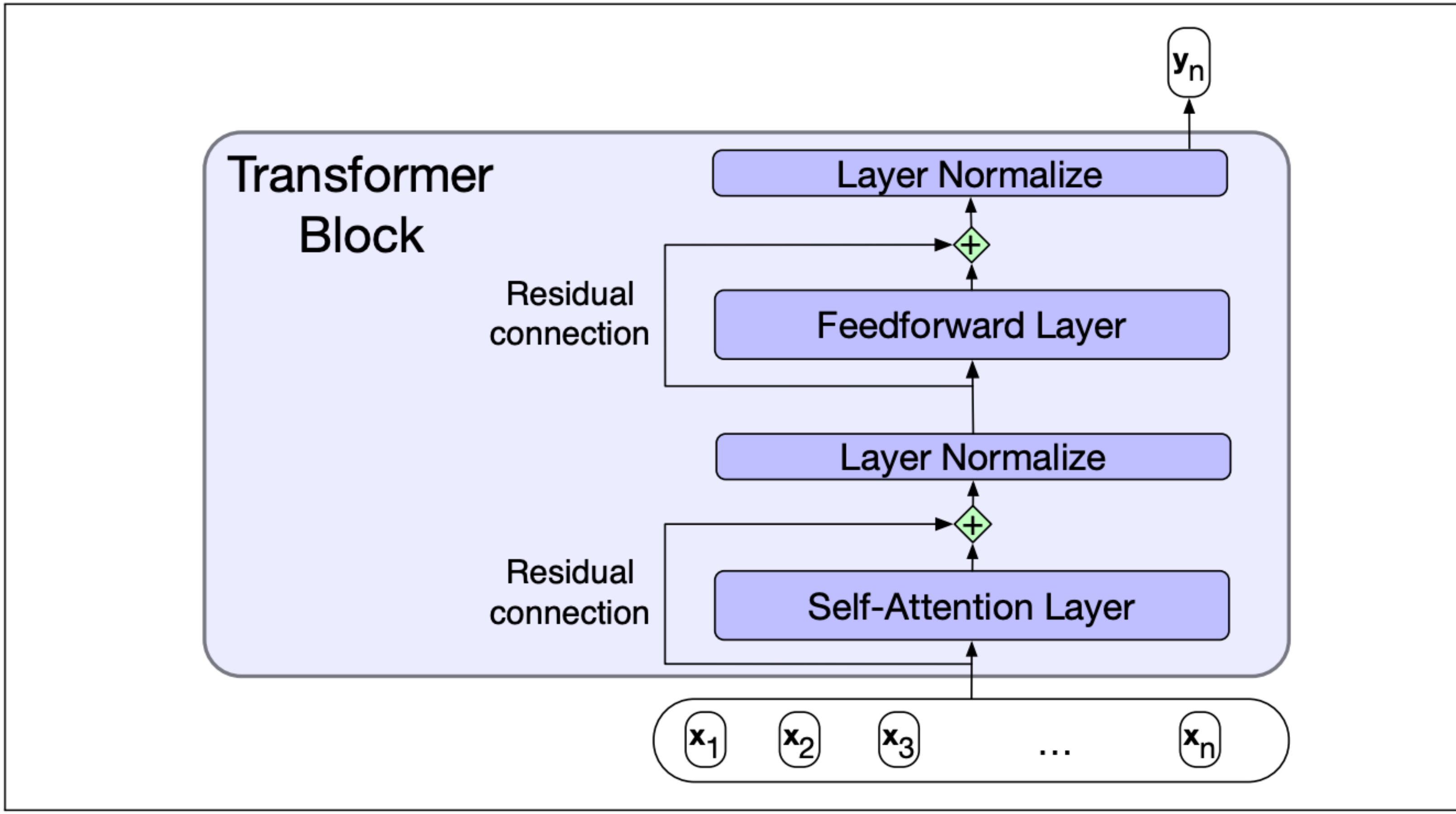
Adding positional information

An approach:
Sine/Cosine encoding





Transformer block



Transformer block

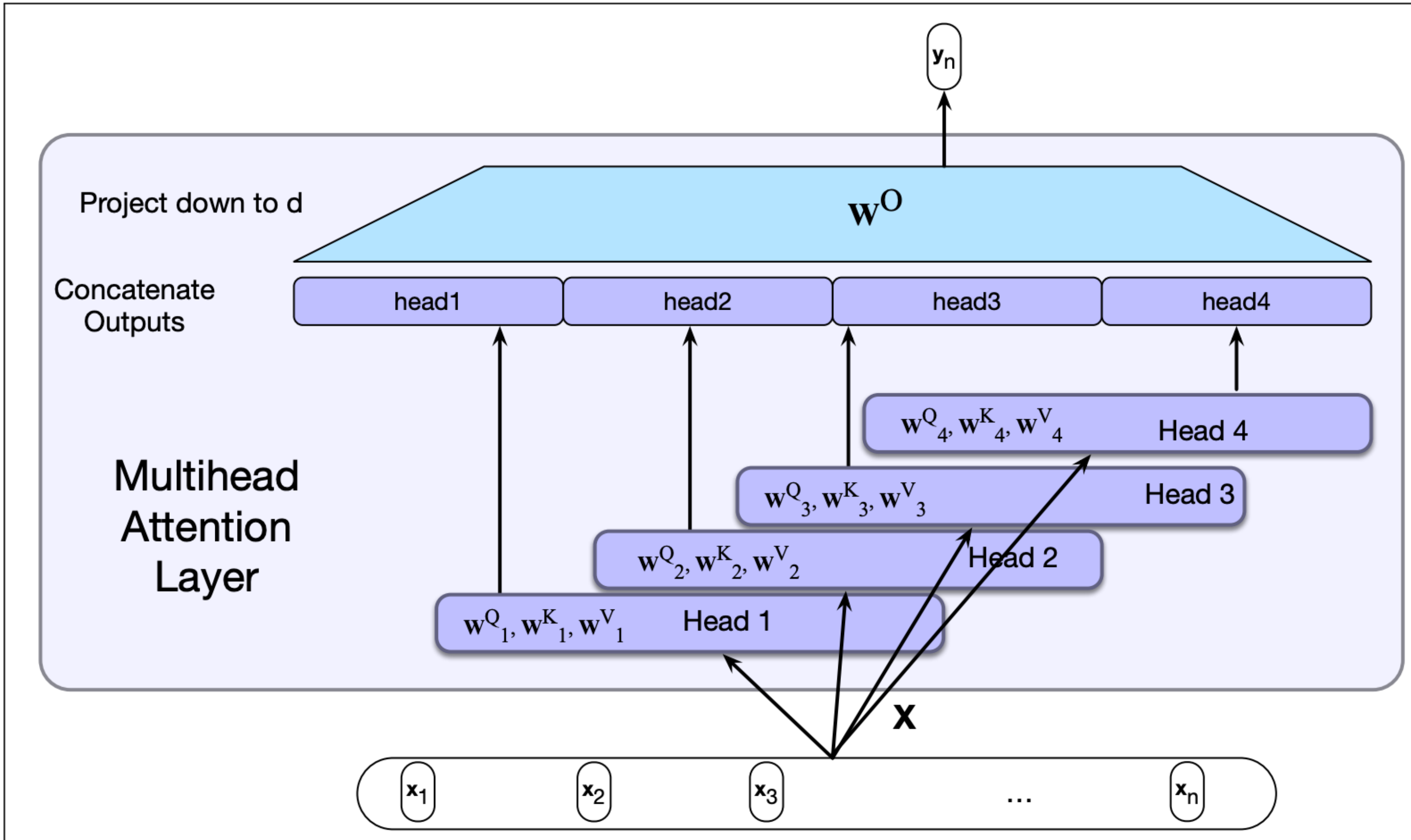
Residual connection:

- 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

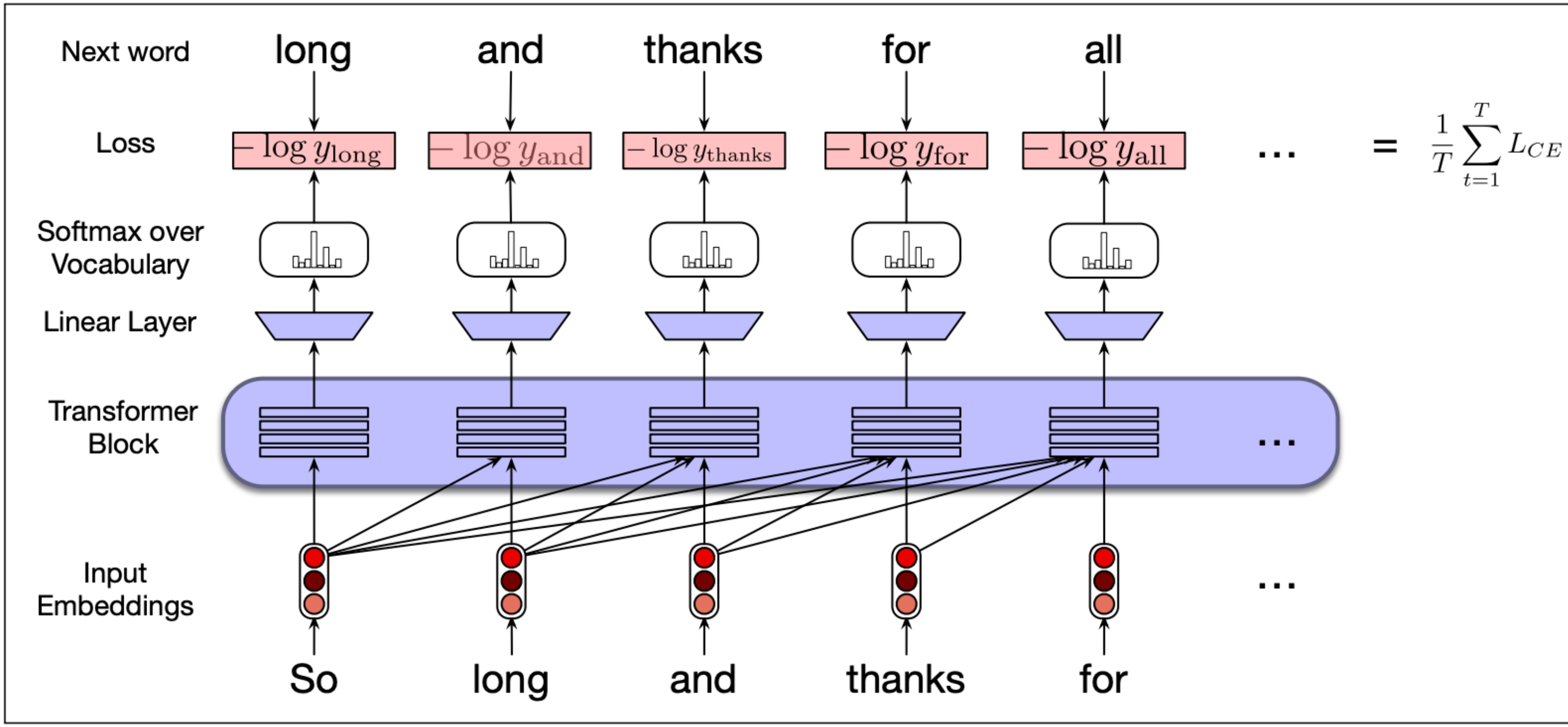


Figure 10.7 Training a transformer as a language model.

Training transformers

parameters in transformer >> # parameters in LSTM

So, training requires a lot of data

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

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**ransformersrmers (Google)
  - RoBERTa: **R**obustly **O**ptimized BERT (Facebook)
  - GPT: **G**enerative **P**re-trained **T**ransformer (OpenAI)