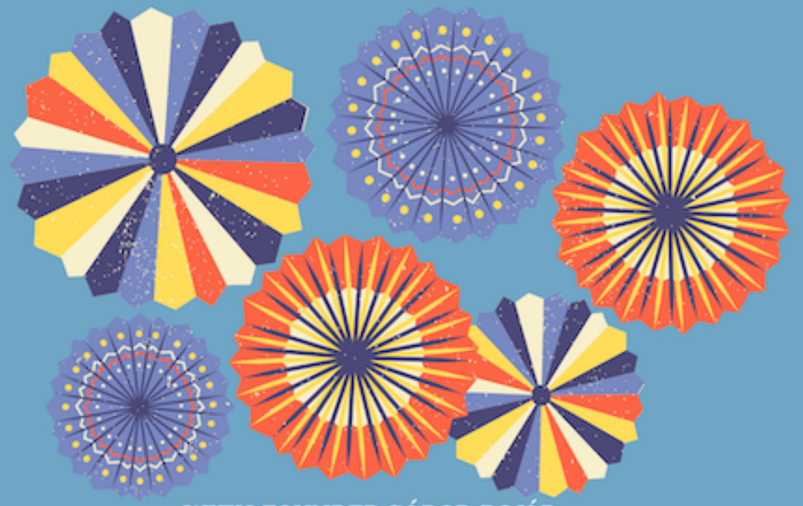
CMPU 366 · Natural Language Processing

Attention and Transformers

3 November 2026



EXPLORE AIT BUDAPEST



WITH FOUNDER,GÁBOR BOJÁR

LEARN ABOUT AIT BUDAPEST AND EXPLORING AI, DATA SCIENCE, SOFTWARE ENGINEERING, AND OTHER TECHNOLOGIES WHILE STUDYING IN ONE OF EUROPE'S LEADING INNOVATION HUBS.

NOVEMBER 5, 2026
5:30PM
SP 105
PIZZA PROVIDED

INDIVIDUALS WITH DISABILITIES REQUIRING ACCOMODATIONS OR INFROMATION ON ACCESSIBILITY SHOULD CONTACT THE CAMPUS ACTIVITIES OFFICE, (845) 437-5370

VASSAR COGNITIVE SCIENCE

Human-Robot Interaction



by Theresa Law ’18

Assistant Professor of Computer Science at Bard College

Tuesday, November 10, 2026
5:00 p.m.
New England Building 105

How do people respond to robots interacting with them? What assumptions about human interaction do we transfer to robots, and which ones do we not? What are our perceptions of robot mindedness? In this talk, Theresa Law walks through how a Vassar Cognitive Science education led her to a career in human-robot interaction. She will present on her past work on emotional perceptions of non-humanoid robots, work that stemmed from her Vassar senior thesis. Additionally, she will discuss recent work on perceptions of robot norm understanding and robot categorization.

Theresa Law is an Assistant Professor of Computer Science at Bard College. She graduated from Vassar in 2018 with a degree in Cognitive Science. Her senior thesis, on how people read emotion into the movements of a non-humanoid robot, led her into human-robot interaction. She earned her Ph.D. in Computer Science and Cognitive Science at Tufts University in 2023, where she studied how people come to trust robots. Her current work looks at how people perceive minds in artificial agents such as robots and AI.

Individuals with disabilities requiring accommodations or information on accessibility should contact the Campus Activities Office, (845) 437-5370.

Sponsored by the Cognitive Science Department.
Co-sponsored by the Computer Science Department.

First half of the semester

Breadth

Individual work

Learn and practice foundational NLP concepts

- Rule-based systems (regular expressions)
- Language models
- Logistic regression
- Word embeddings
- Deep learning

First half of the semester

Breadth

Individual work

Learn and practice foundational NLP concepts

- Rule-based systems (regular expressions)
- Language models
- Logistic regression
- Word embeddings
- Deep learning

Second half of the semester

Depth

Group work

Deep and sustained creative problem-solving to build complex projects around one topic

Special topics

This is a chance to explore a topic we didn't cover yet. It's a good idea for this to be on the general area you'd like your final project to be in – though it doesn't need to be!

If you haven't already done so, sign up for a topic/ day before the start of next class.

Projects

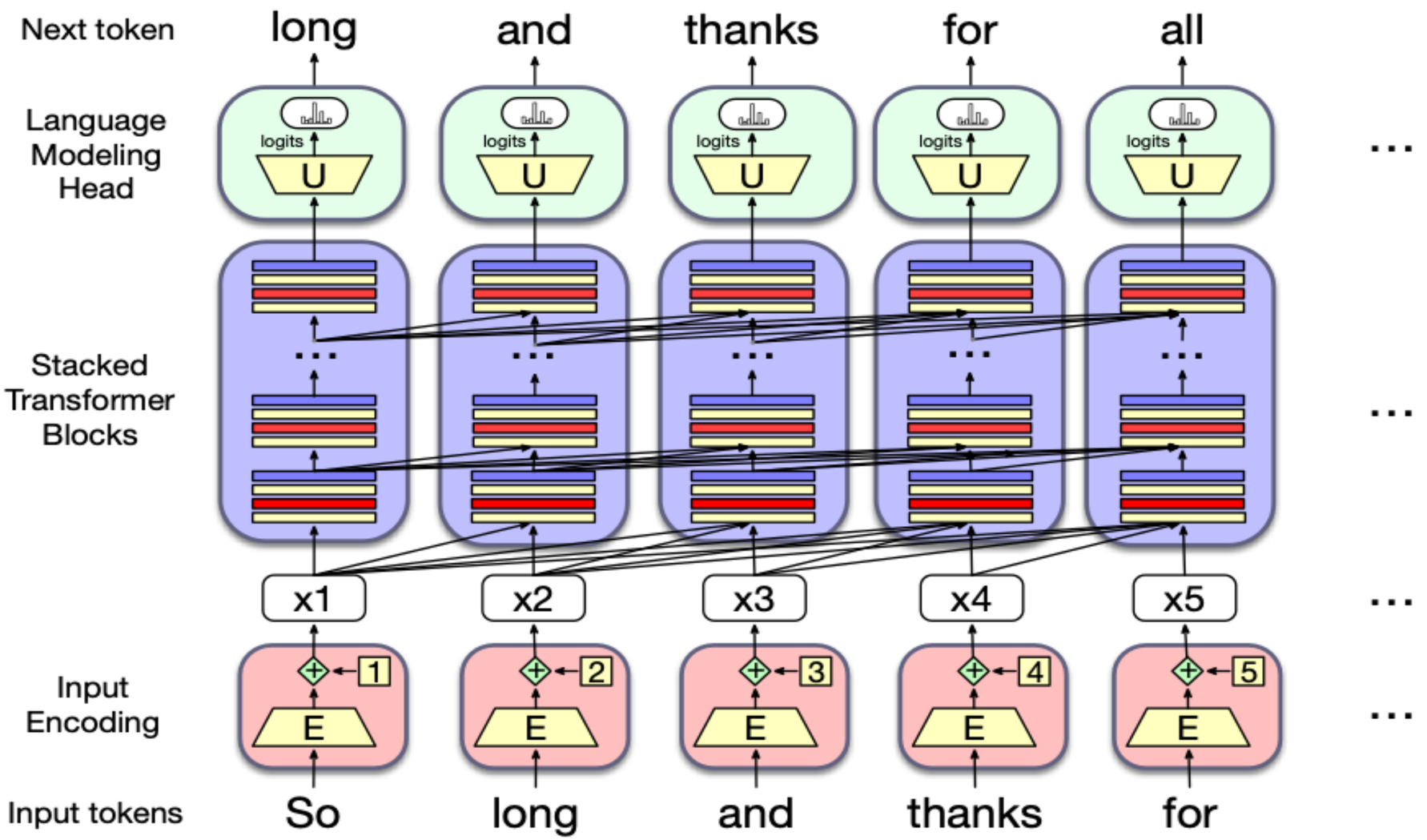
(Short) proposals will be due next Wednesday.

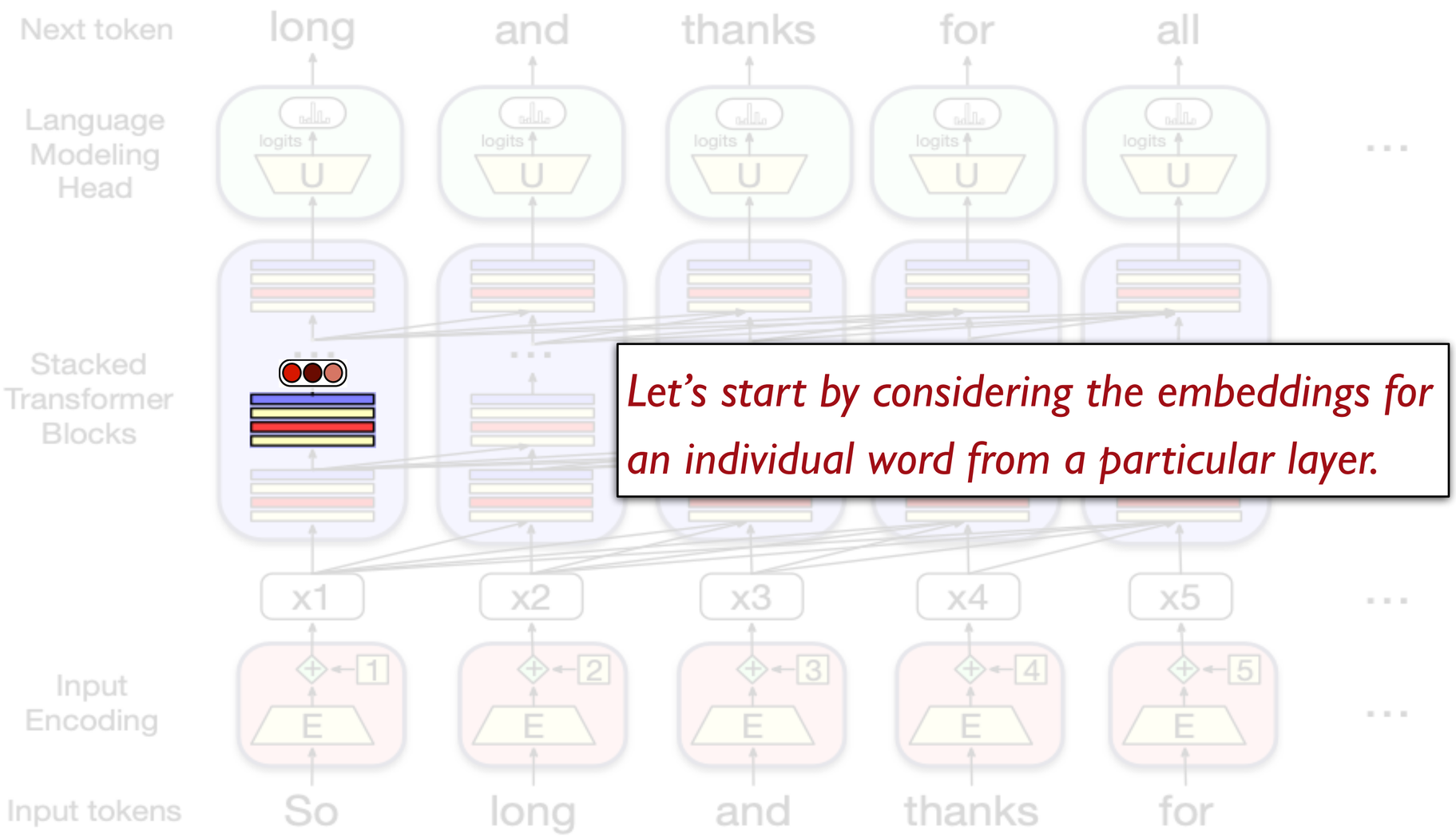
We'll use class time next Thursday to start working on projects.

LLMs are built out of transformers.

A *transformer* is a specific kind of network architecture – like a fancier feedforward network, but based on *attention*.

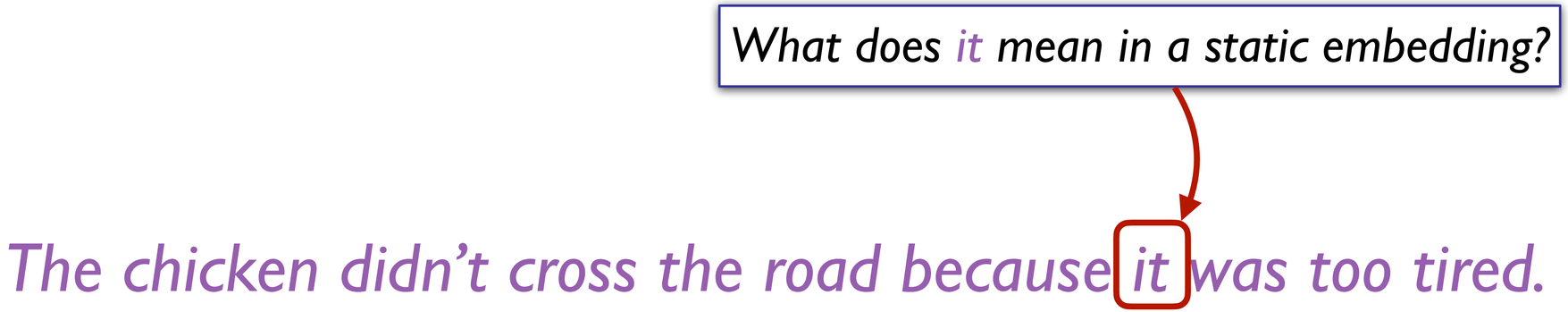
Attention


Next token
long
and
thanks
for
all
Language Modeling Head
logits
U
Stacked Transformer Blocks
x1
x2
x3
x4
x5
Input Encoding
1
2
3
4
5
E
Input tokens
So
long
and
thanks
for


Next token
long
and
thanks
for
all
Language Modeling Head
logits
U
Stacked Transformer Blocks
Let's start by considering the embeddings for an individual word from a particular layer.
x1
x2
x3
x4
x5
Input Encoding
1
2
3
4
5
E
Input tokens
So
long
and
thanks
for

The problem with embeddings like Word2vec is they're *static*; the embedding for a word doesn't reflect how its meaning changes in *context*.

The chicken didn’t cross the road because it was too tired.


What does it mean in a static embedding?
The chicken didn't cross the road because it was too tired.

A better representation of the meaning of a word would be different in different contexts!

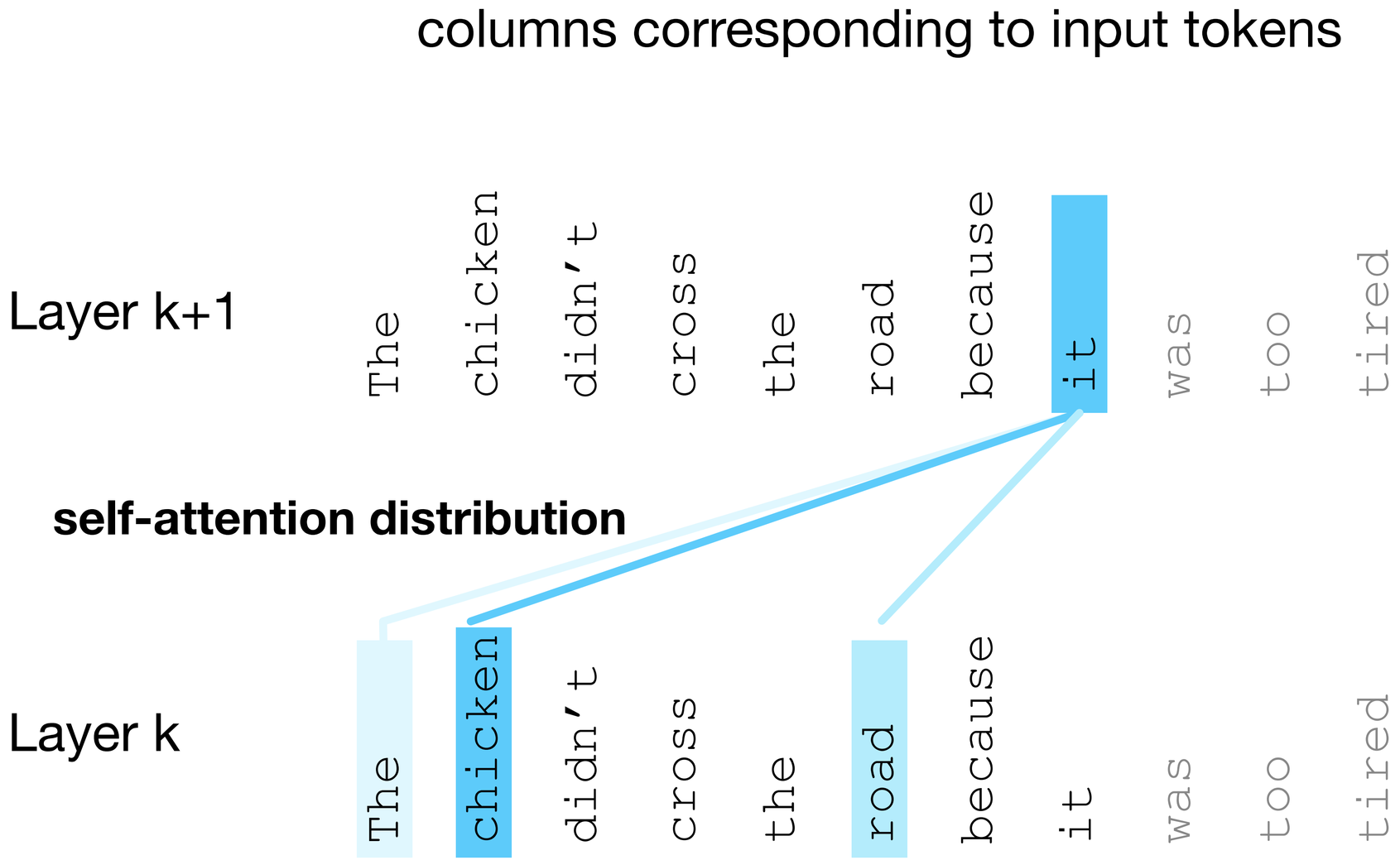
This is called a *contextual embedding*.

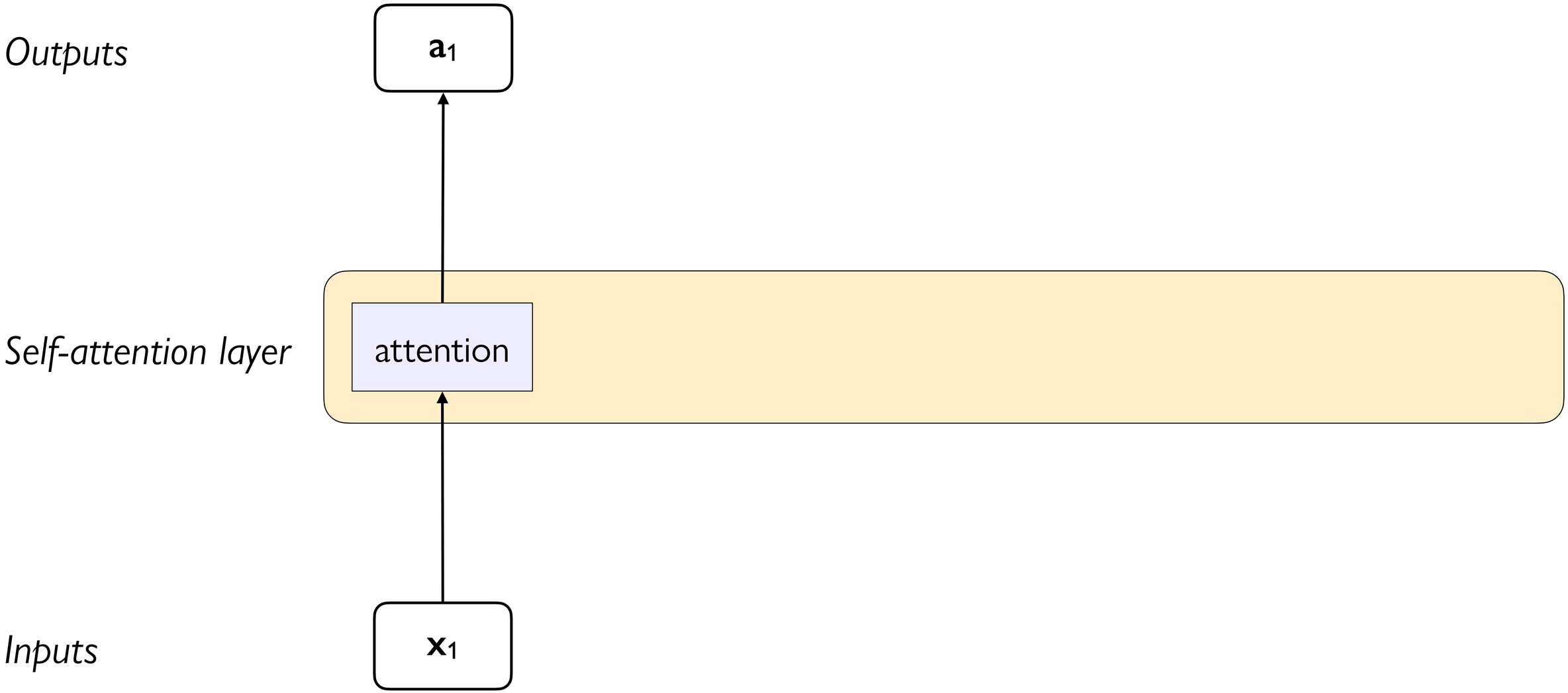
The chicken didn't cross the road because it was too tired.

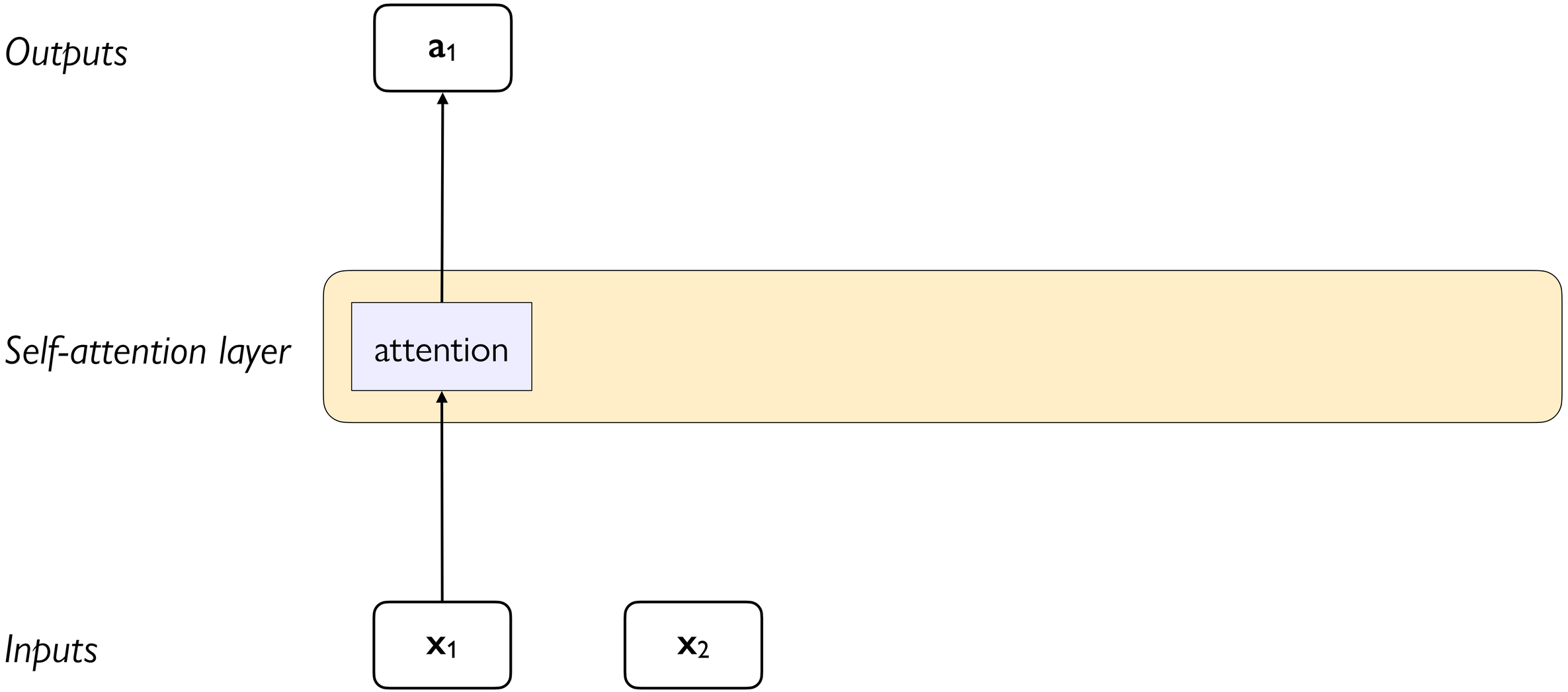
The chicken didn't cross the road because it was too wide.

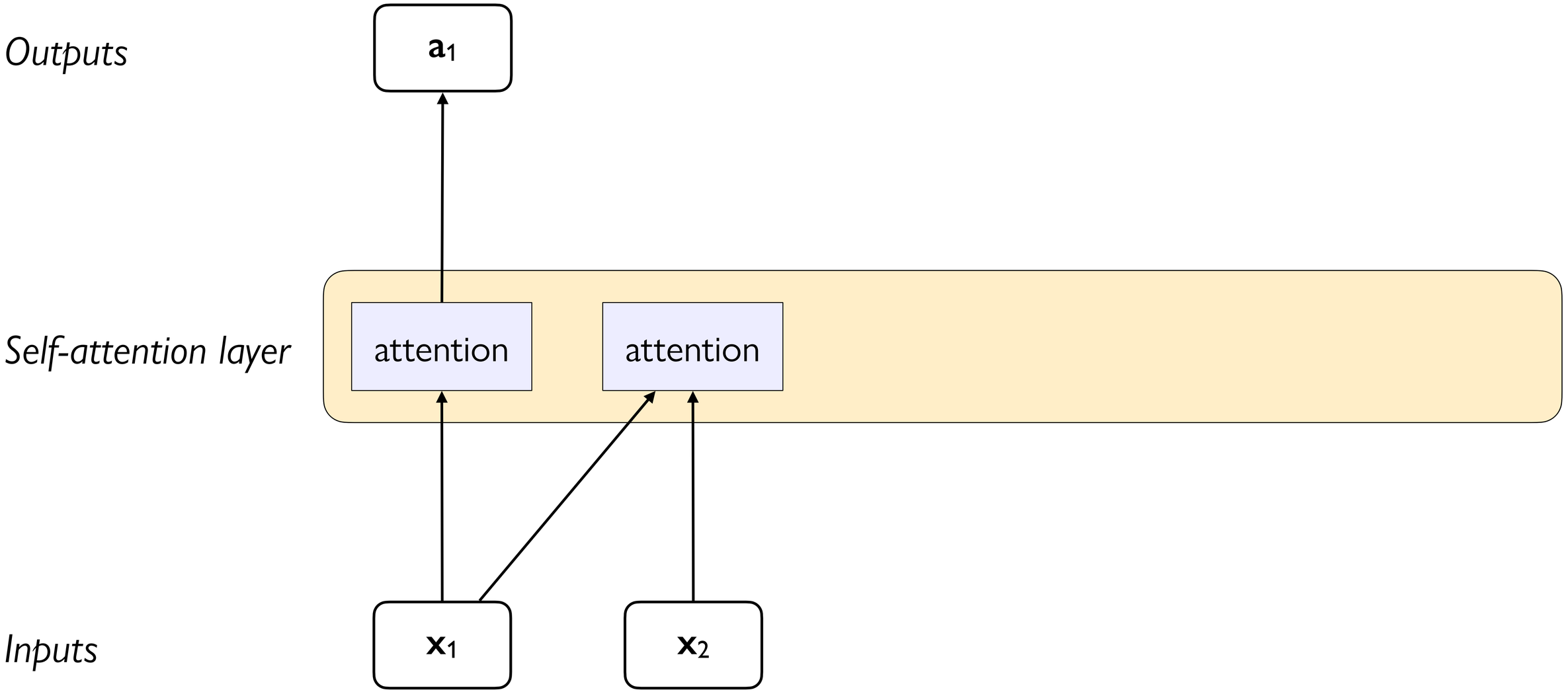
The chicken didn't cross the road because it…

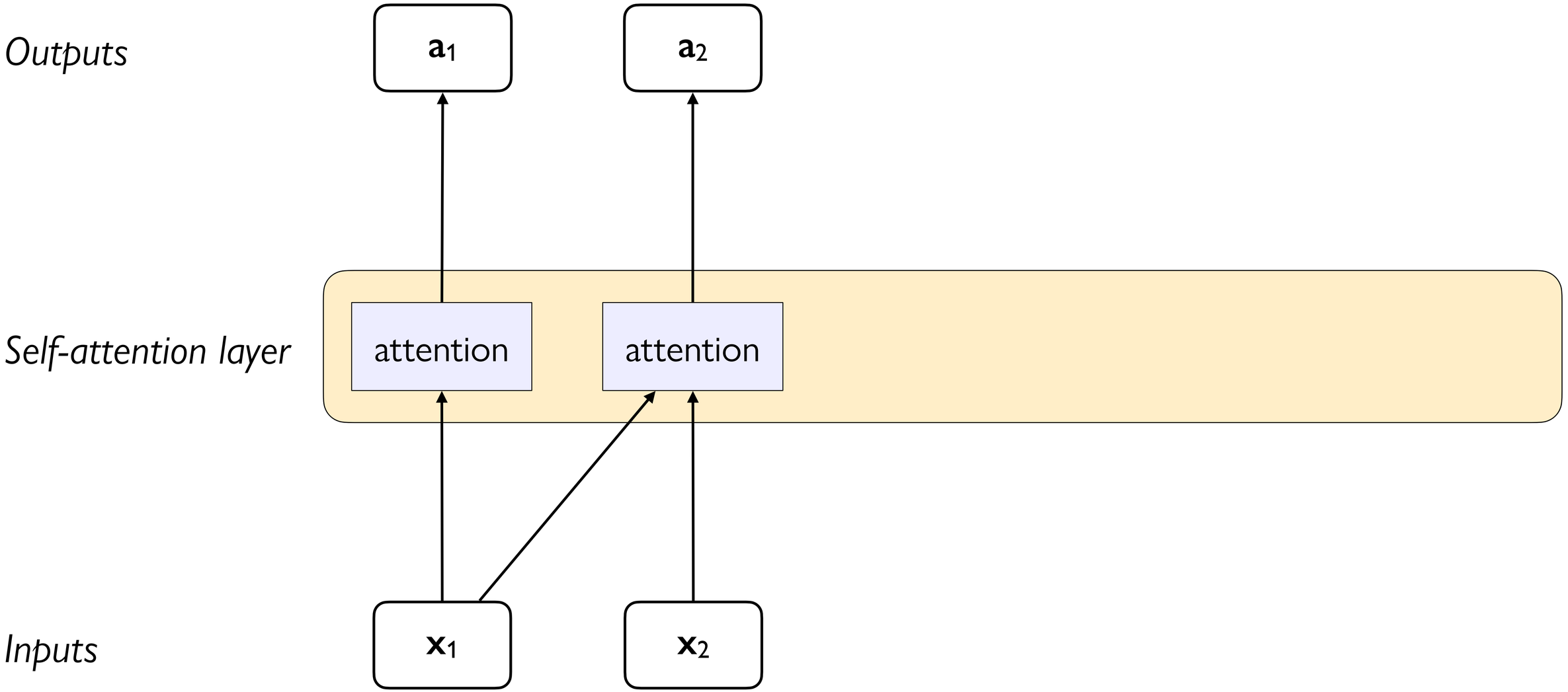
A transformer uses *attention* to build a contextual representation of a word by selectively integrating information from all the neighboring words.

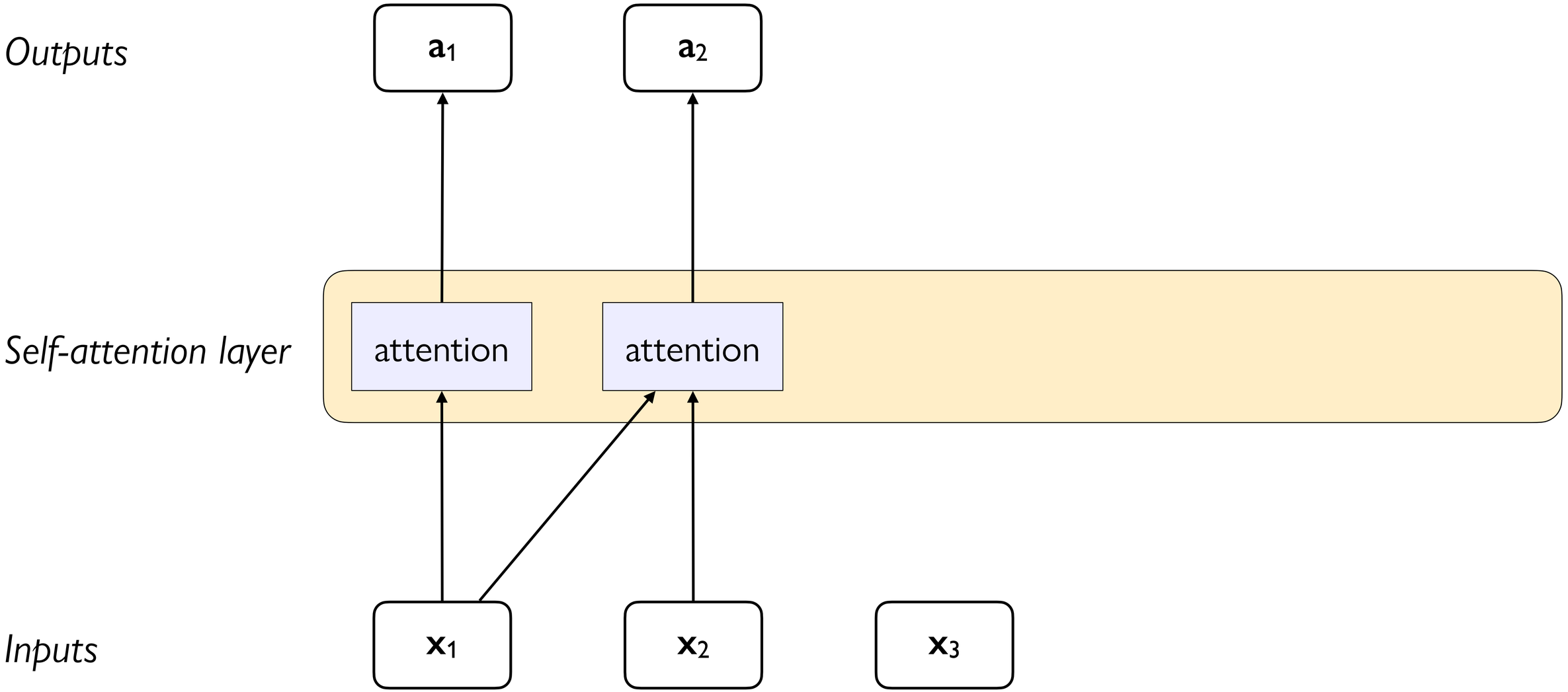

columns corresponding to input tokens
Layer k+1
The
chicken
didn't
cross
the
road
because
it
was
too
tired
self-attention distribution
Layer k
The
chicken
didn't
cross
the
road
because
it
was
too
tired

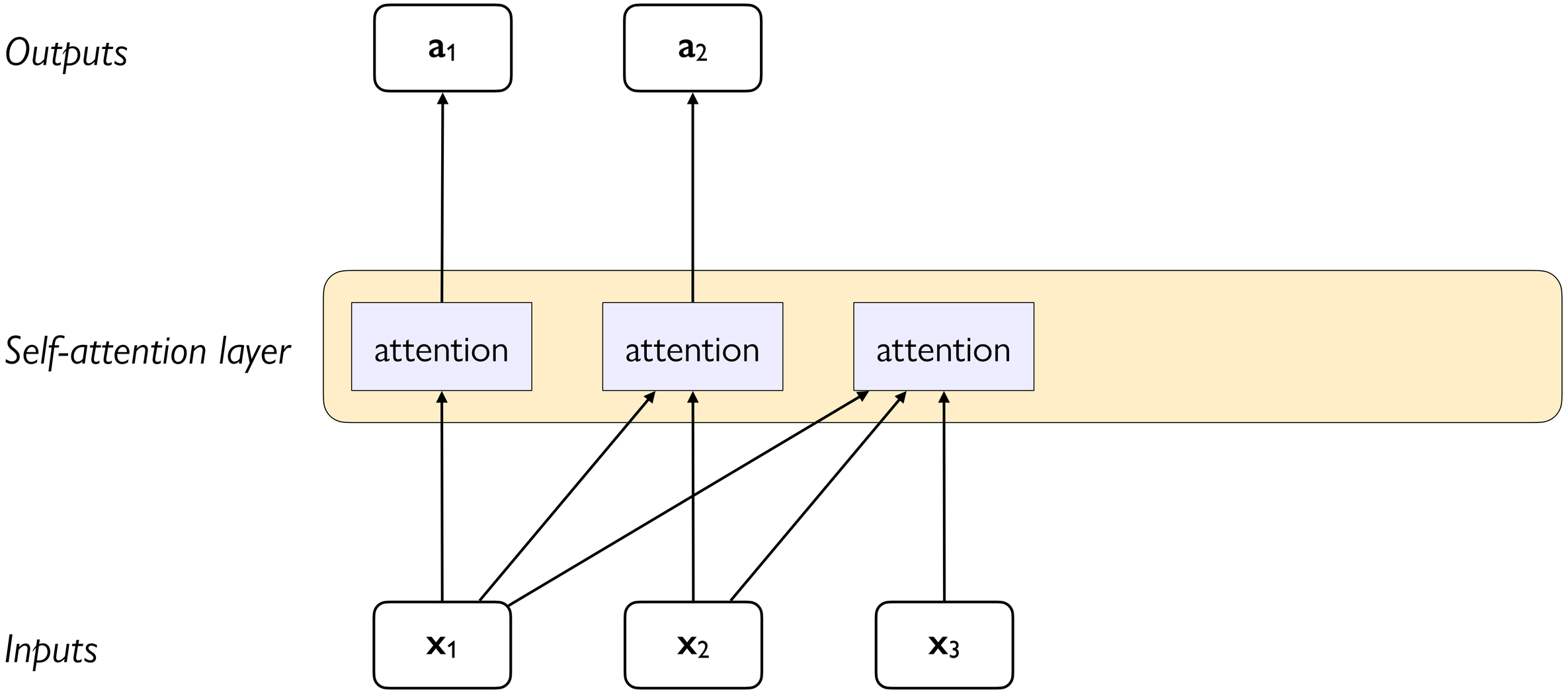

Outputs
$\mathbf{a}_1$
Self-attention layer
attention
Inputs
$\mathbf{x}_1$

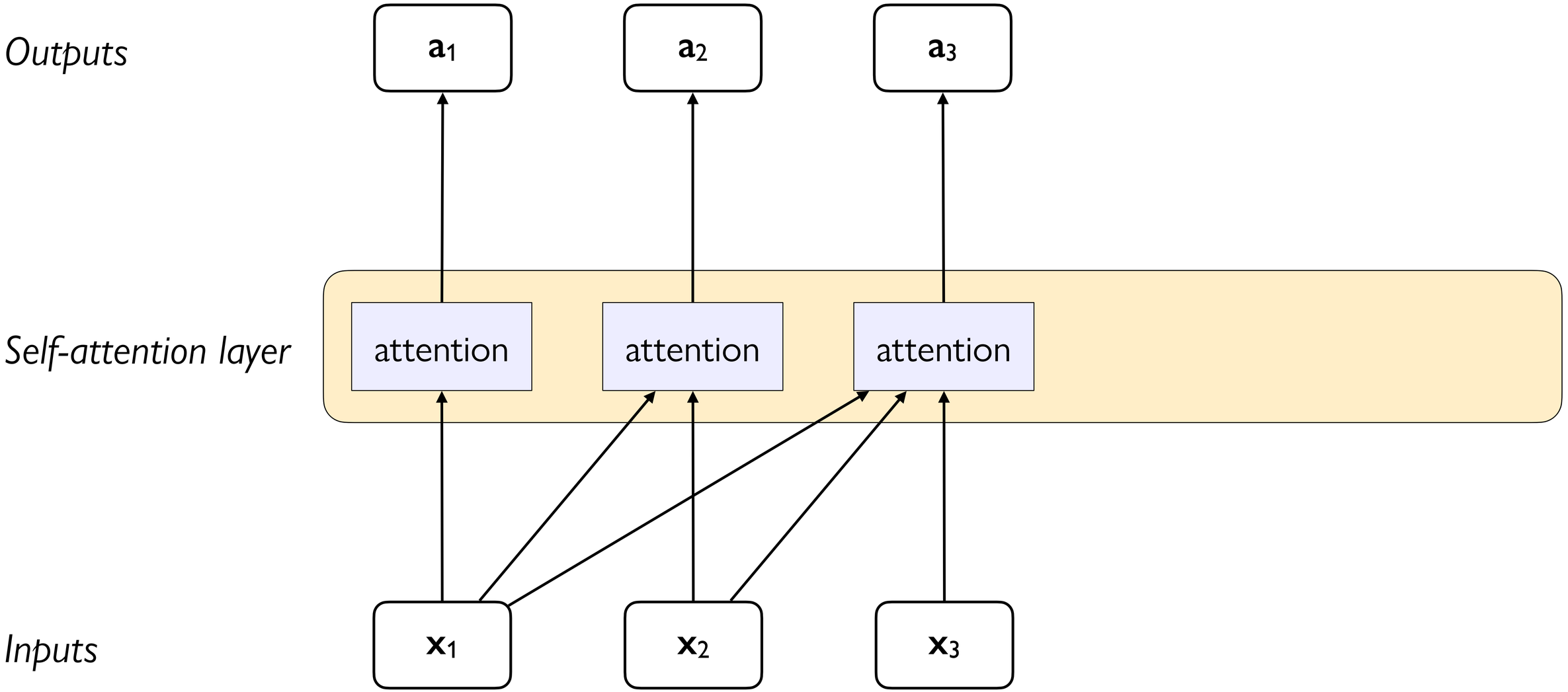

Outputs
$\mathbf{a}_1$
Self-attention layer
attention
Inputs
$\mathbf{x}_1$
$\mathbf{x}_2$

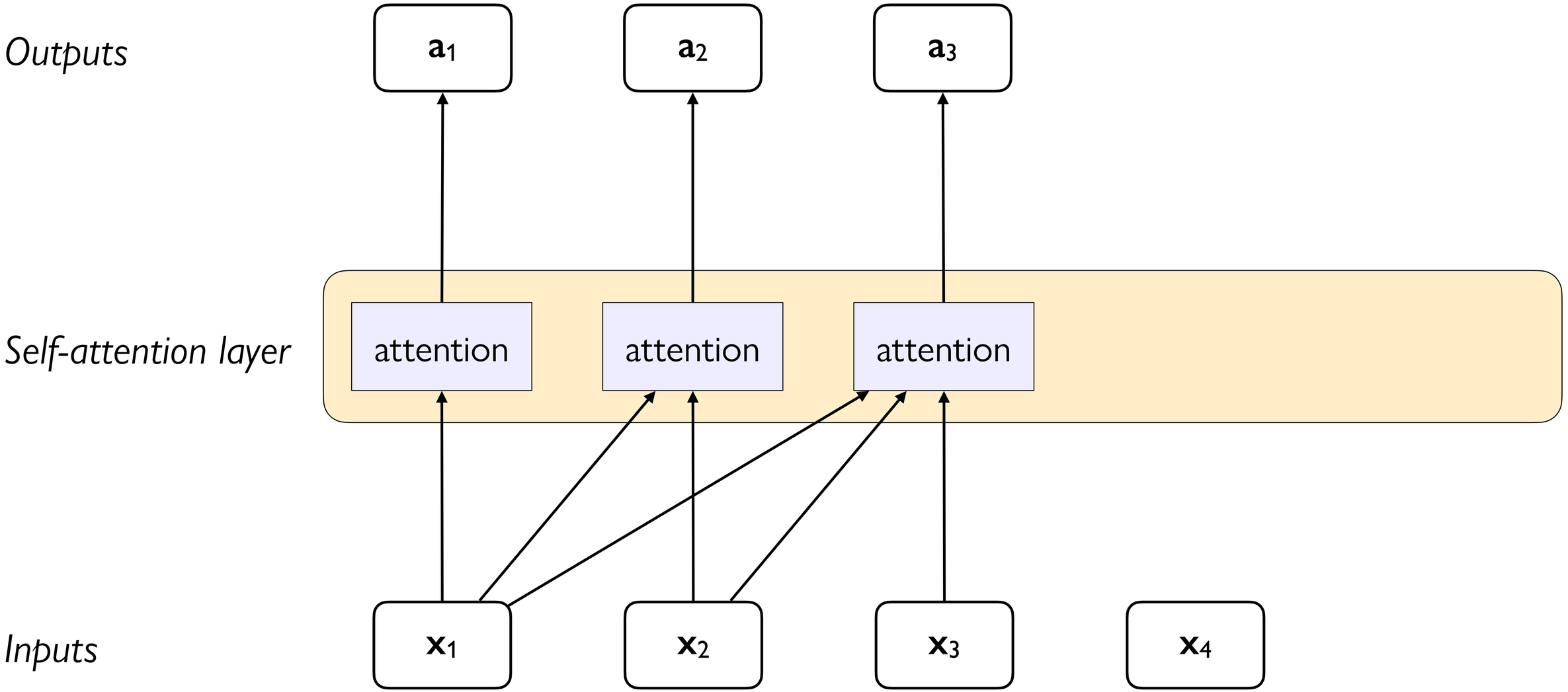

Outputs
$\mathbf{a}_1$
Self-attention layer
attention
attention
Inputs
$\mathbf{x}_1$
$\mathbf{x}_2$

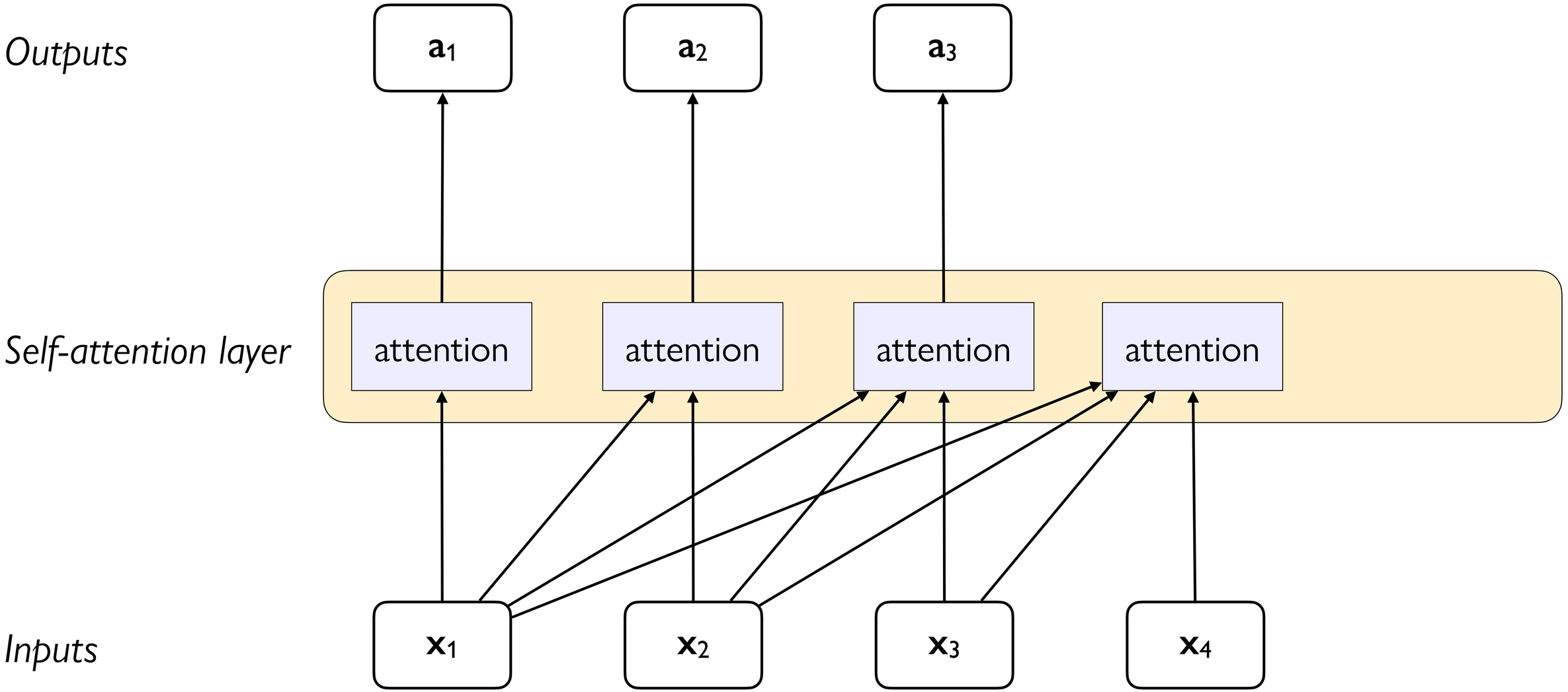

Outputs
$\mathbf{a}_1$
$\mathbf{a}_2$
Self-attention layer
attention
attention
Inputs
$\mathbf{x}_1$
$\mathbf{x}_2$

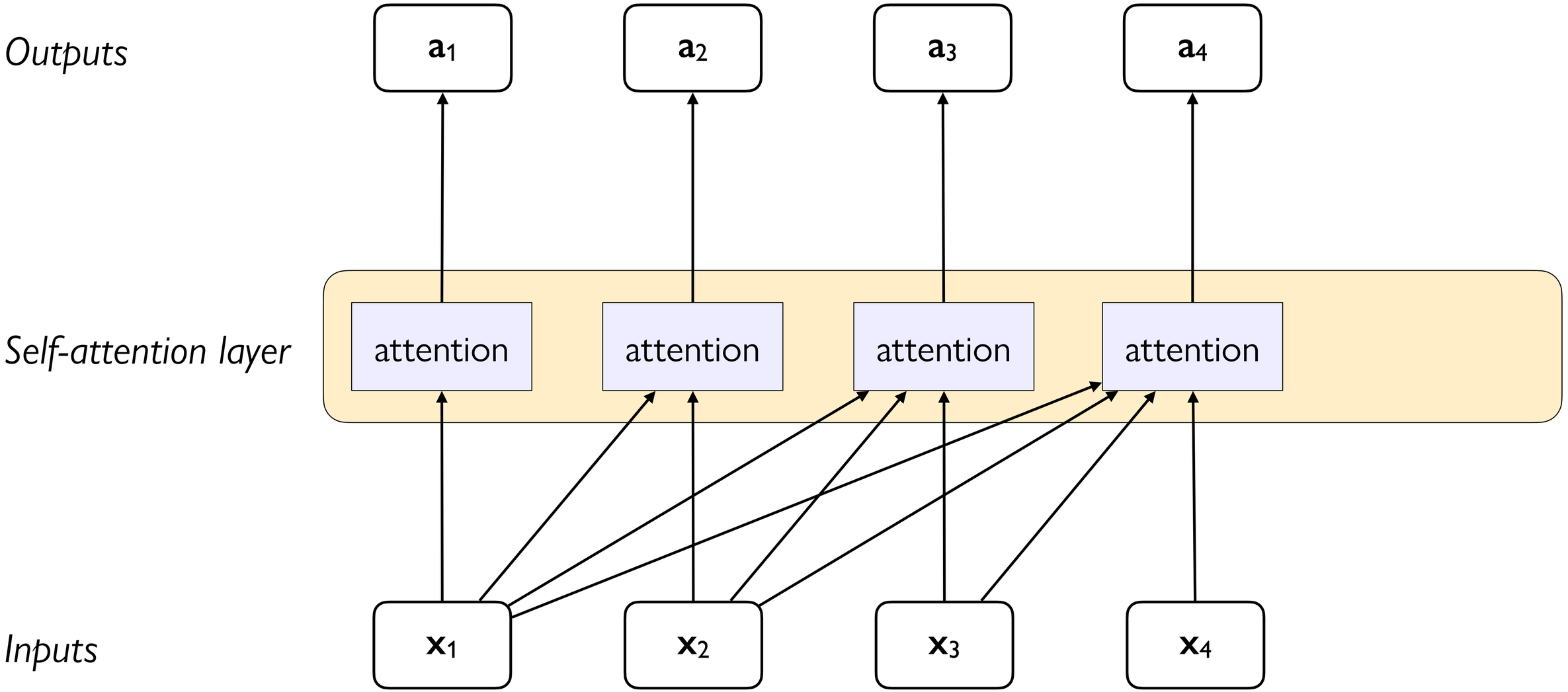

Outputs
$\mathbf{a}_1$
$\mathbf{a}_2$
Self-attention layer
attention
attention
Inputs
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$

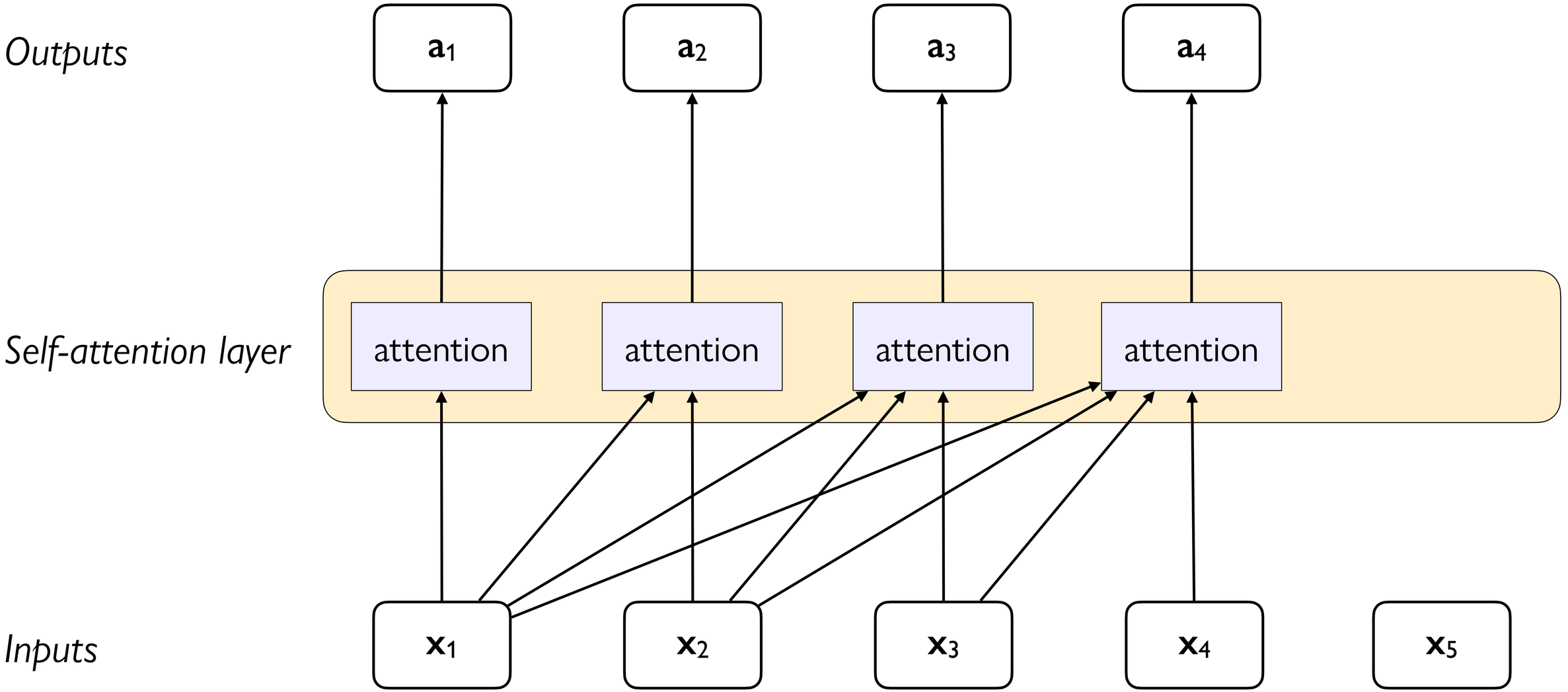

Outputs
$\mathbf{a}_1$
$\mathbf{a}_2$
Self-attention layer
attention
attention
attention
Inputs
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$

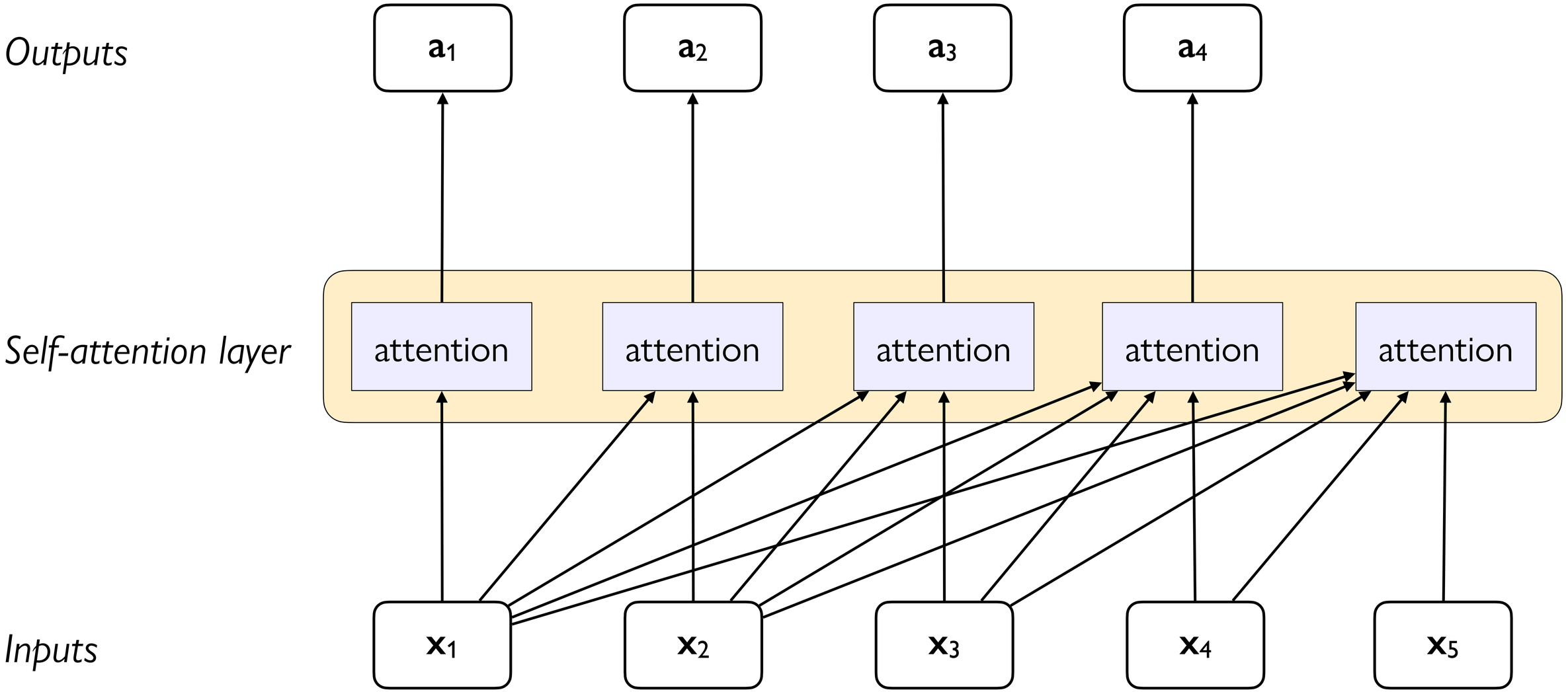

Outputs
$\mathbf{a}_1$
$\mathbf{a}_2$
$\mathbf{a}_3$
Self-attention layer
attention
attention
attention
Inputs
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$

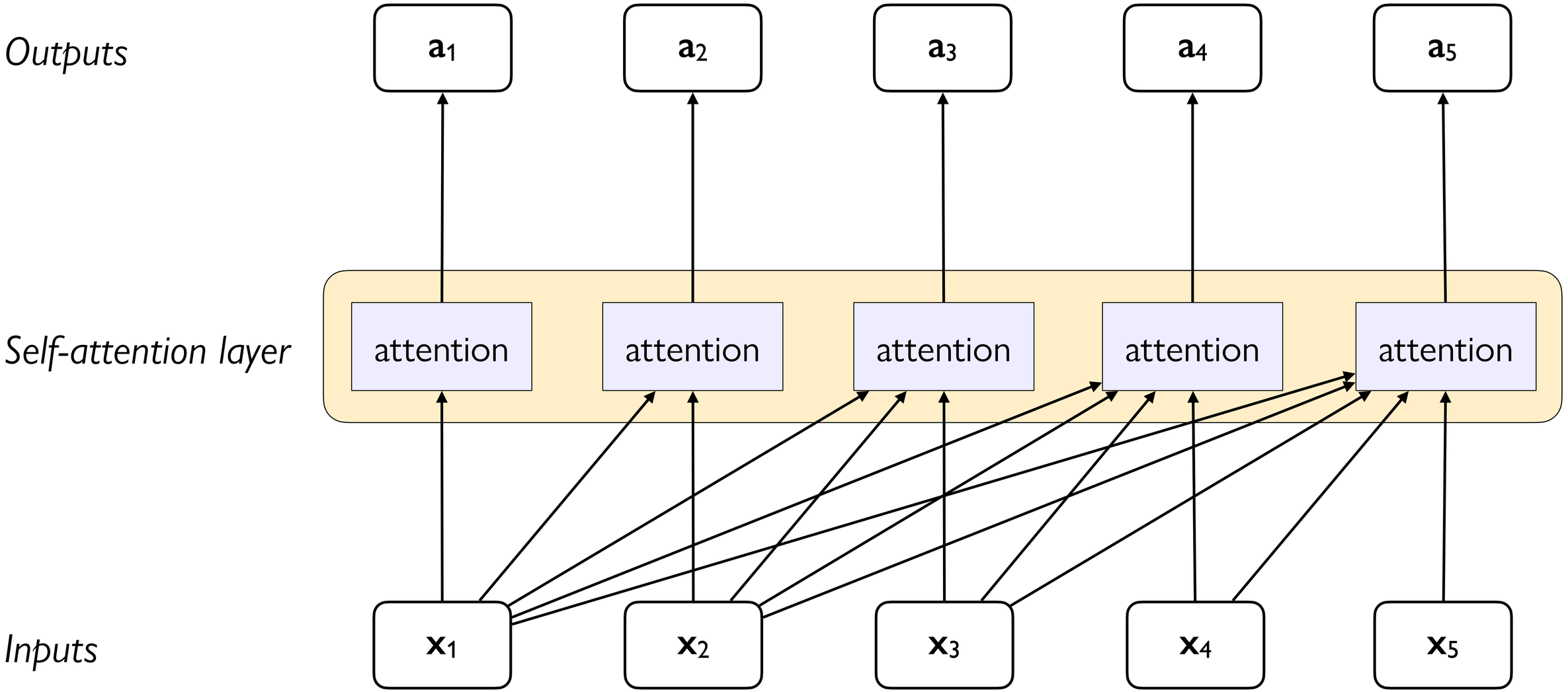

Outputs
$\mathbf{a}_1$
$\mathbf{a}_2$
$\mathbf{a}_3$
Self-attention layer
attention
attention
attention
Inputs
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$


Outputs
$\mathbf{a}_1$
$\mathbf{a}_2$
$\mathbf{a}_3$
Self-attention layer
attention
attention
attention
attention
Inputs
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$


Outputs
Self-attention layer
Inputs
a1
a2
a3
a4
attention
attention
attention
attention
x1
x2
x3
x4


Outputs
$\mathbf{a}_1$
$\mathbf{a}_2$
$\mathbf{a}_3$
$\mathbf{a}_4$
Self-attention layer
attention
attention
attention
attention
Inputs
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
$\mathbf{x}_5$


Outputs
$\mathbf{a}_1$
$\mathbf{a}_2$
$\mathbf{a}_3$
$\mathbf{a}_4$
Self-attention layer
attention
attention
attention
attention
attention
Inputs
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
$\mathbf{x}_5$


Outputs
$\mathbf{a}_1$
$\mathbf{a}_2$
$\mathbf{a}_3$
$\mathbf{a}_4$
$\mathbf{a}_5$
Self-attention layer
attention
attention
attention
attention
attention
Inputs
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
$\mathbf{x}_5$

Given a sequence of token embeddings

$$\mathbf{x}_1 \quad \mathbf{x}_2 \quad \mathbf{x}_3 \quad \mathbf{x}_4 \quad \mathbf{x}_5 \quad \mathbf{x}_6 \quad \mathbf{x}_7 \quad \mathbf{x}_i$$

we can produce a sum of the embeddings weighted by their similarity to $\mathbf{x}_i$:

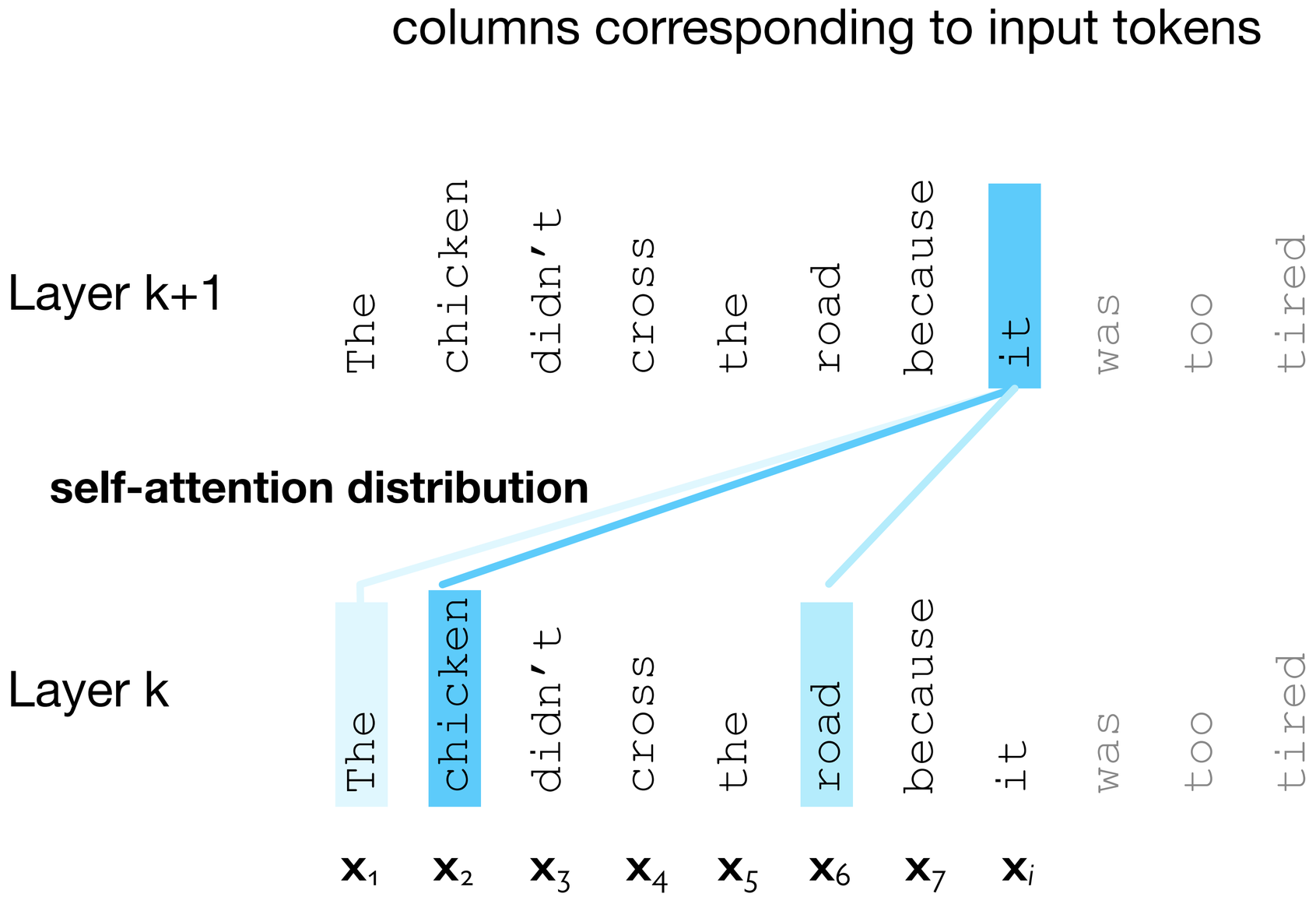
$$\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{x}_j$$

where the weight

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

and we can simply define

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


columns corresponding to input tokens
Layer k+1
The
chicken
didn't
cross
the
road
because
it
was
too
tired
self-attention distribution
Layer k
The
chicken
didn't
cross
the
road
because
it
was
too
tired
x1
x2
x3
x4
x5
x6
x7
xi

An actual attention head is slightly more complicated.

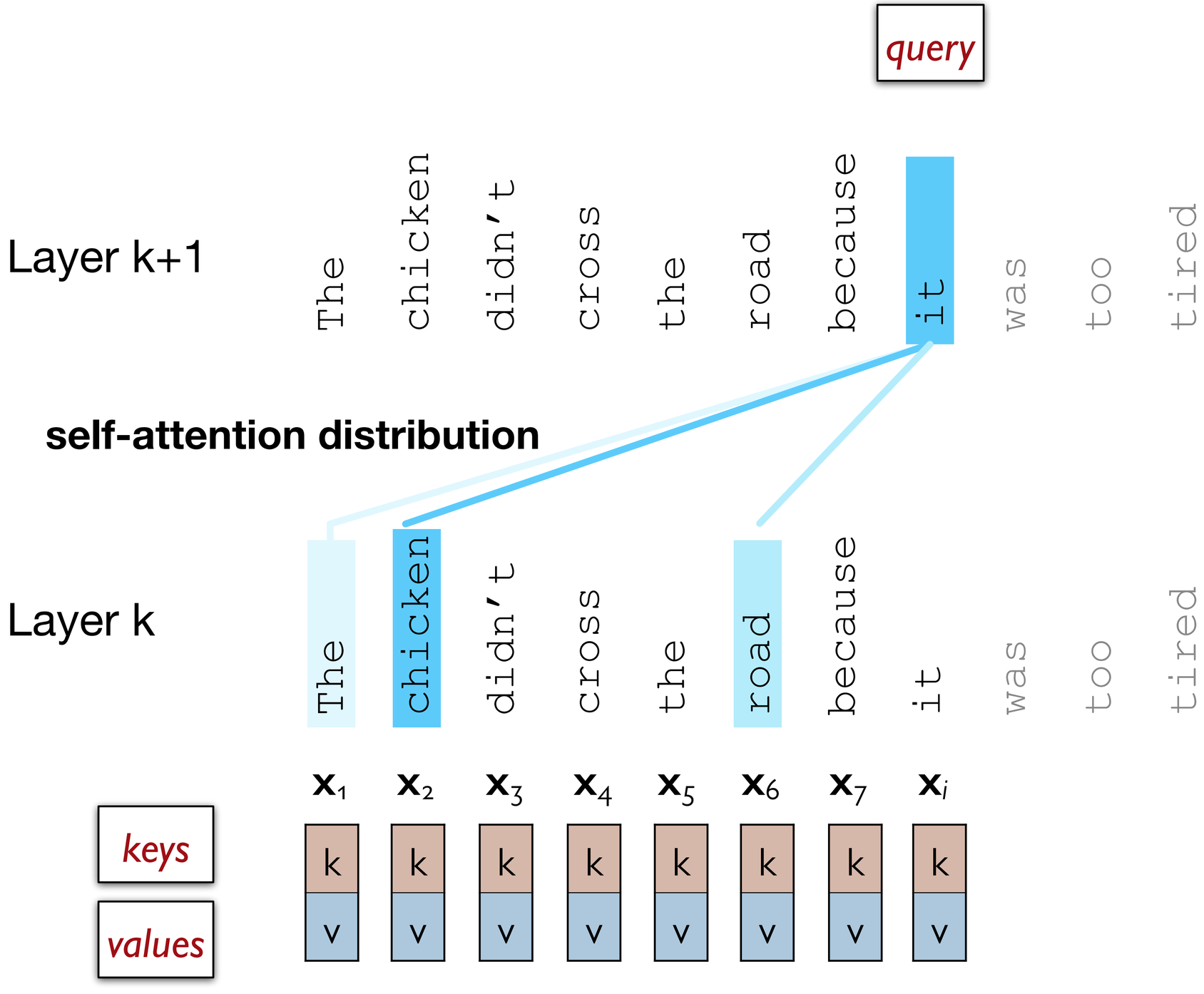
Instead of using vectors like $\mathbf{x}_i$ and $\mathbf{x}_4$ directly, we'll represent three separate *roles* each vector $\mathbf{x}_i$ plays:

Query: As the *current element* being compared to the preceding inputs

Key: As a *preceding input* that is being compared to the current element to determine a similarity

Value: a value of a preceding element that gets *weighted and summed*.

query
Layer k+1
The
chicken
didn't
cross
the
road
because
it
was
too
tired
self-attention distribution
Layer k
The
chicken
didn't
cross
the
road
because
it
was
too
tired
x1
x2
x3
x4
x5
x6
x7
xi
values


query
Layer k+1
The
chicken
didn't
cross
the
road
because
it
was
too
tired
self-attention distribution
Layer k
The
chicken
didn't
cross
the
road
because
it
was
too
tired
x1
x2
x3
x4
x5
x6
x7
xi
keys
values
k
v

We'll use weight matrices $\mathbf{W^Q}$, $\mathbf{W^K}$, and $\mathbf{W^V}$ to project each vector $\mathbf{x}_i$ into a representation of its role as a

query: $\mathbf{q}_i = \mathbf{x}_i\mathbf{W^Q}$

key: $\mathbf{k}_i = \mathbf{x}_i\mathbf{W^K}$

value: $\mathbf{v}_i = \mathbf{x}_i\mathbf{W^V}$

To compute the similarity of the current element $\mathbf{x}_i$ with some prior element $\mathbf{x}_j$, we'll use the dot product between the projections $\mathbf{q}_i$ and $\mathbf{k}_j$.

And instead of summing up $\mathbf{x}_j$, we'll sum up the projection $\mathbf{v}_j$.

Simplified

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

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

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

De-simplified

$$\mathbf{q}_i = \mathbf{x}_i \mathbf{W}^{\mathbf{Q}}$$

$$\mathbf{k}_j = \mathbf{x}_j \mathbf{W}^{\mathbf{K}}$$

$$\mathbf{v}_j = \mathbf{x}_j \mathbf{W}^{\mathbf{V}}$$

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

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

$$\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{v}_j$$

$$\mathbf{q}_i = \mathbf{x}_i \mathbf{W}^{\mathbf{Q}}$$

$$\mathbf{k}_j = \mathbf{x}_j \mathbf{W}^{\mathbf{K}}$$

$$\mathbf{v}_j = \mathbf{v}_j \mathbf{W}^{\mathbf{V}}$$

d_k is the dimensionality of the query and key vectors

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

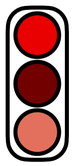
$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{q}_i \cdot \mathbf{k}_j$$

This is a practical change to avoid numerical issues during training.

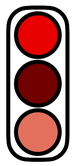
$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \;\; \forall j \leq i$$

$$\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{v}_j$$

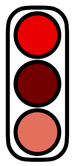
Project each $\mathbf{x}_i$ into key, query, and value vectors



$\mathbf{x}_1$

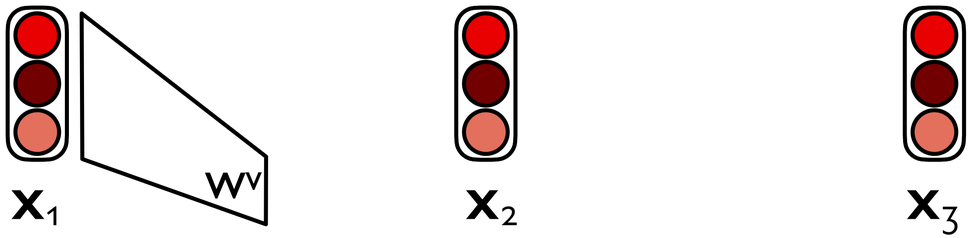


$\mathbf{x}_2$

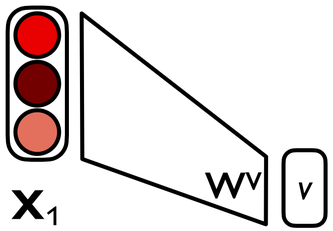


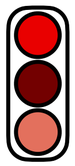
$\mathbf{x}_3$

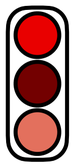
Project each $\mathbf{x}_i$ into key, query, and value vectors



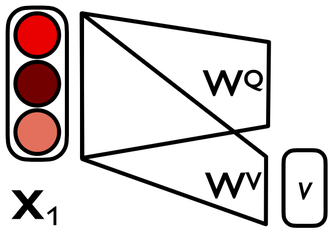
Project each $\mathbf{x}_i$ into key, query, and value vectors

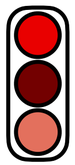


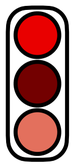




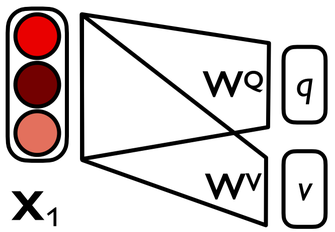
Project each $\mathbf{x}_i$ into key, query, and value vectors

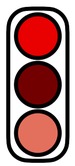


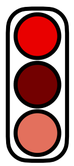




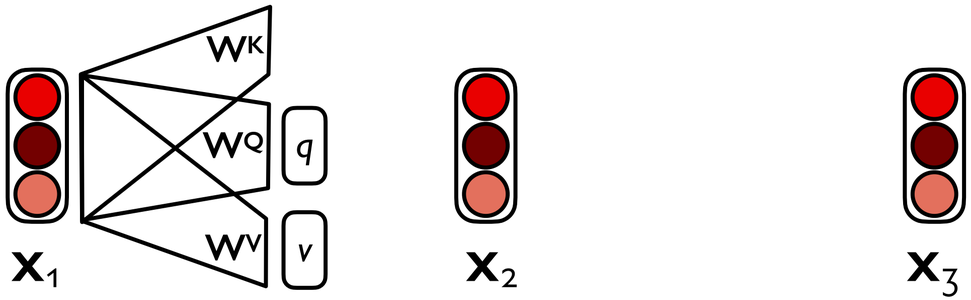
Project each $\mathbf{x}_i$ into key, query, and value vectors



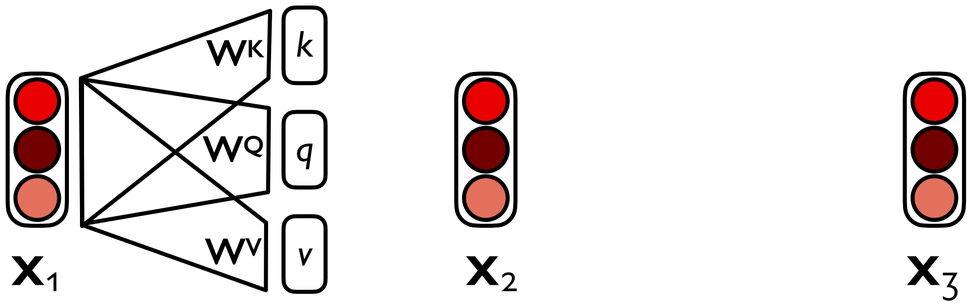




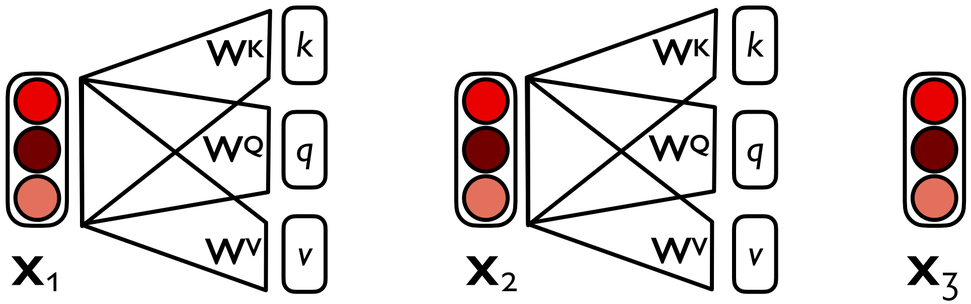
Project each $\mathbf{x}_i$ into key, query, and value vectors



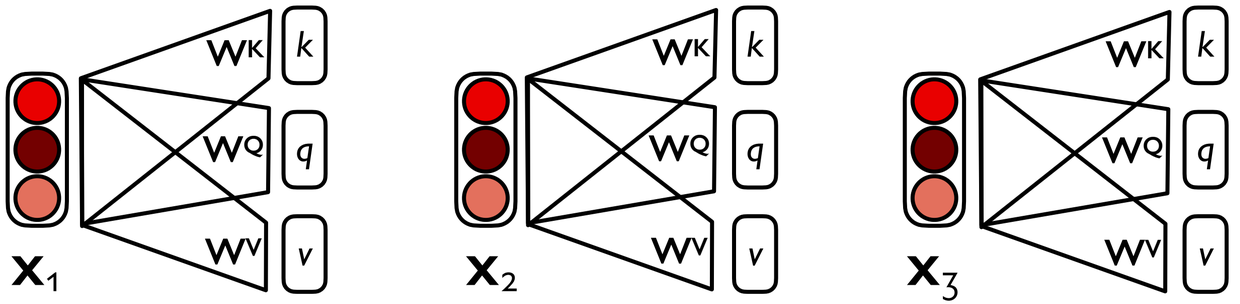
Project each $\mathbf{x}_i$ into key, query, and value vectors



Project each $\mathbf{x}_i$ into key, query, and value vectors

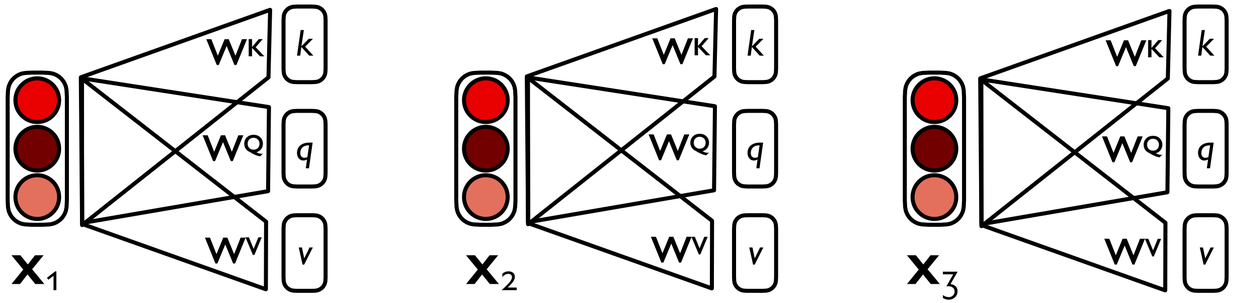


Project each $\mathbf{x}_i$ into key, query, and value vectors



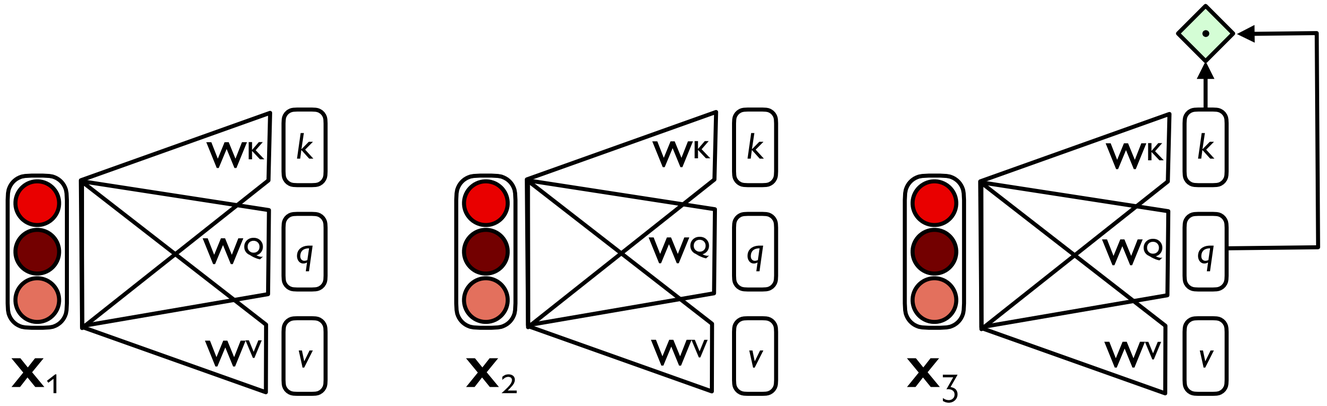
Compare $\mathbf{x}_3$*'s query with the keys for* $\mathbf{x}_1$, $\mathbf{x}_2$, *and* $\mathbf{x}_3$.

Project each $\mathbf{x}_i$ *into key, query, and value vectors*



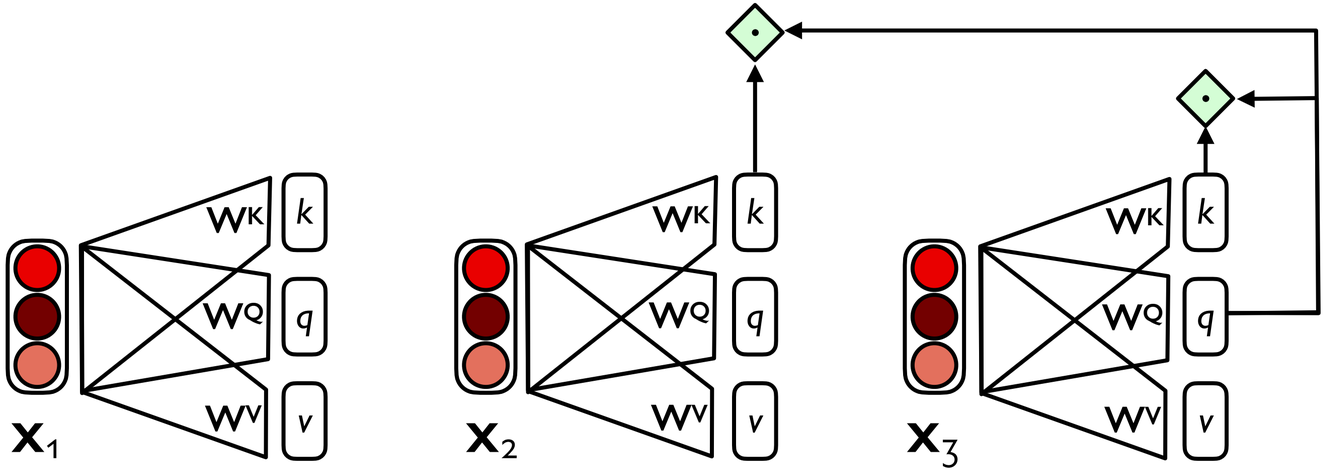
Compare $\mathbf{x}_3$*'s query with the keys for* $\mathbf{x}_1$, $\mathbf{x}_2$, *and* $\mathbf{x}_3$.

Project each $\mathbf{x}_i$ *into key, query, and value vectors*



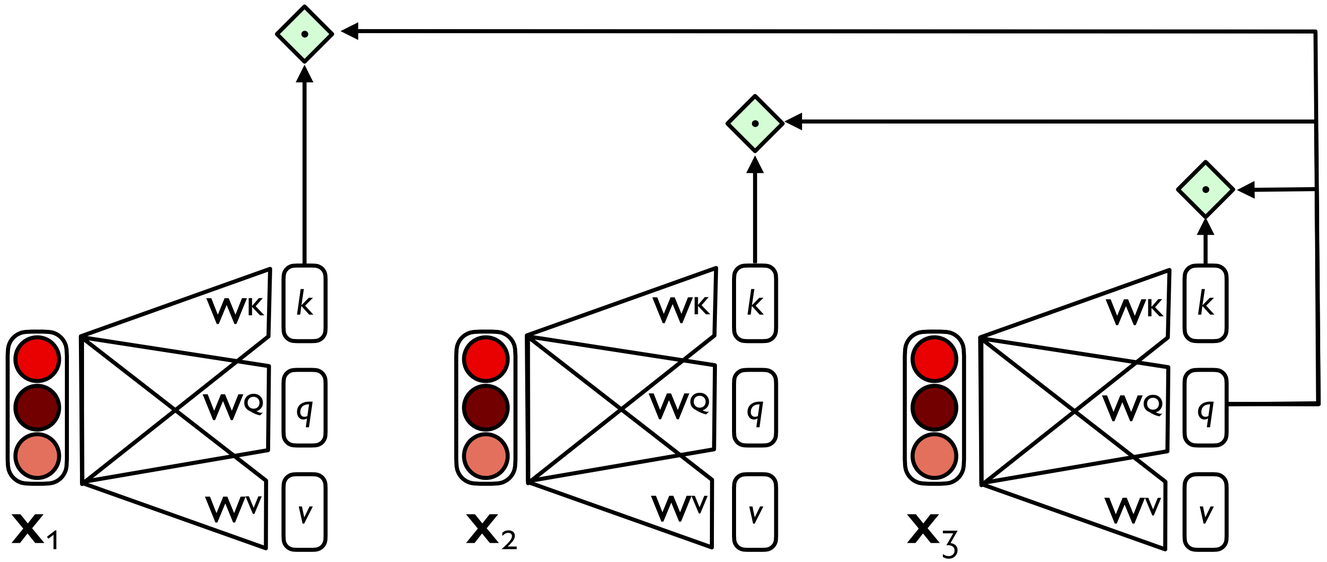
Compare $\mathbf{x}_3$*'s query with the keys for* $\mathbf{x}_1$, $\mathbf{x}_2$, *and* $\mathbf{x}_3$.

Project each $\mathbf{x}_i$ *into key, query, and value vectors*



Compare $\mathbf{x}_3$*'s query with the keys for* $\mathbf{x}_1$, $\mathbf{x}_2$, *and* $\mathbf{x}_3$.

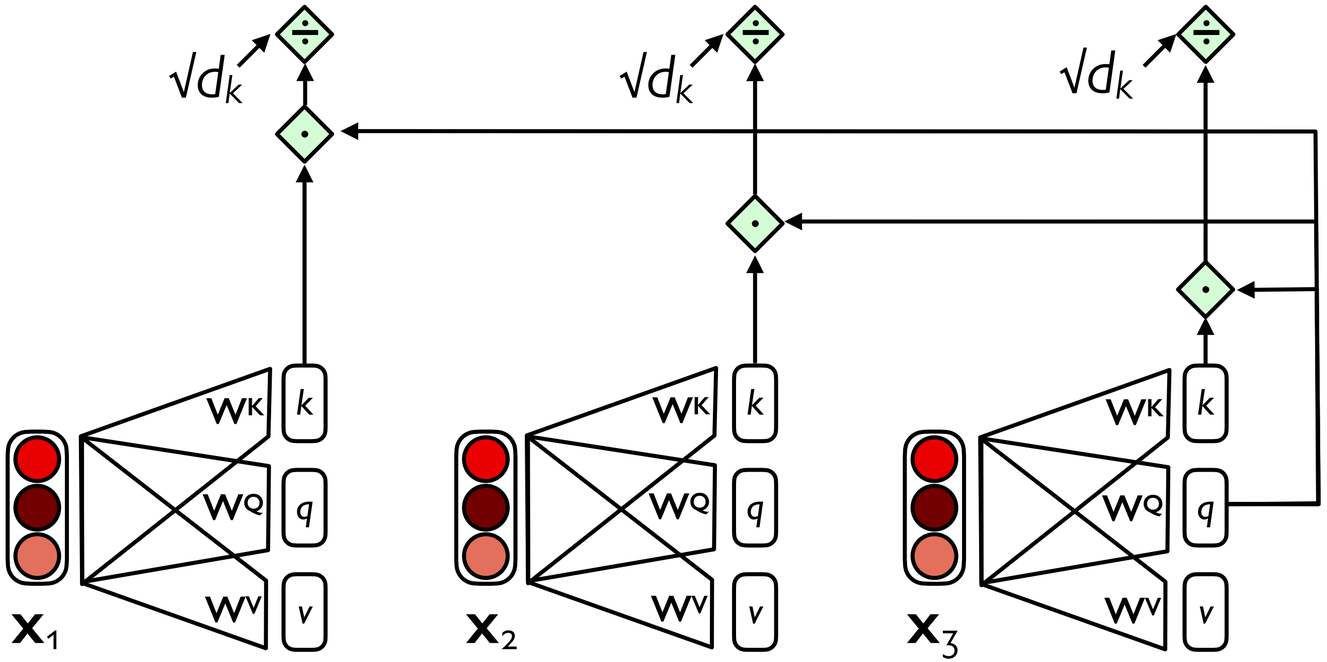
Project each $\mathbf{x}_i$ *into key, query, and value vectors*



Divide score by $\sqrt{d_k}$

Compare $\mathbf{x}_3$'s query with the keys for $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$.

Project each $\mathbf{x}_i$ into key, query, and value vectors

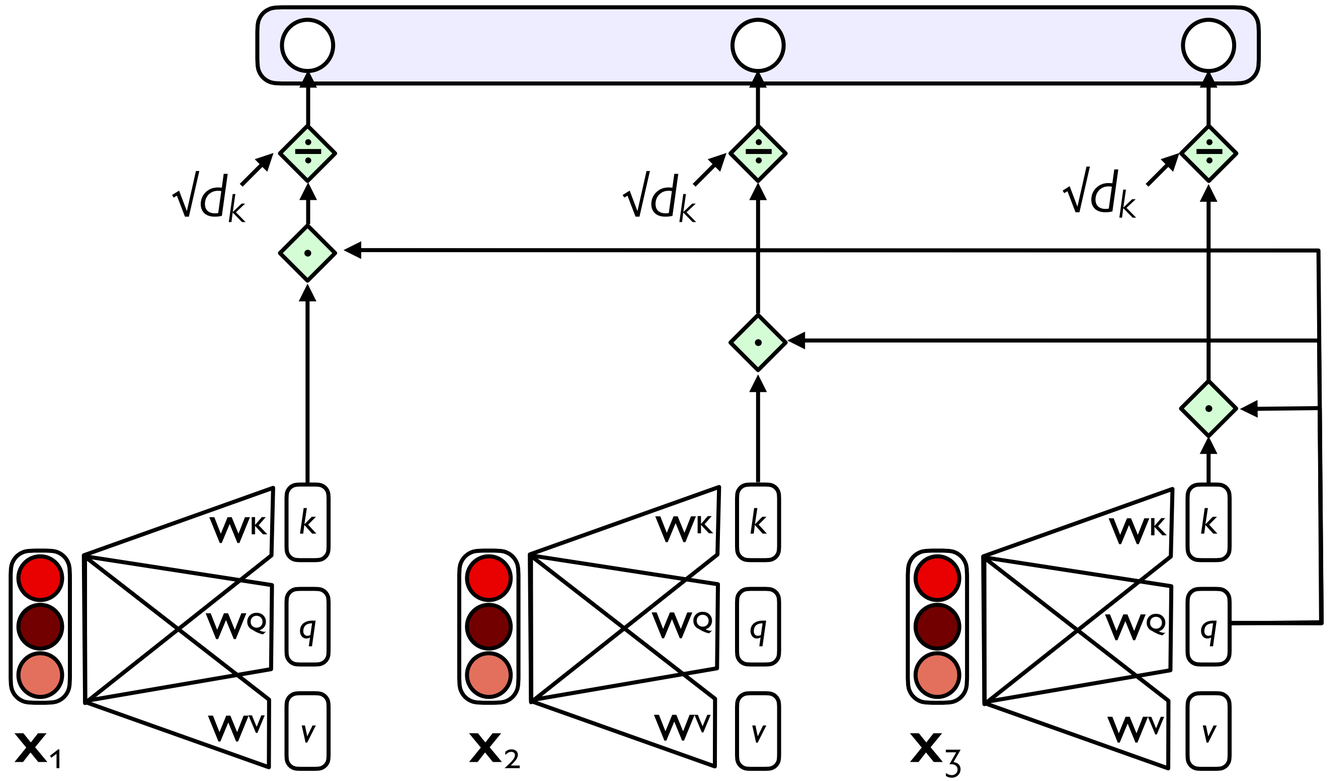


Turn into a_{ij} weights via softmax

Divide score by $\sqrt{d_k}$

Compare $\mathbf{x}_3$'s query with the keys for $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$.

Project each $\mathbf{x}_i$ into key, query, and value vectors



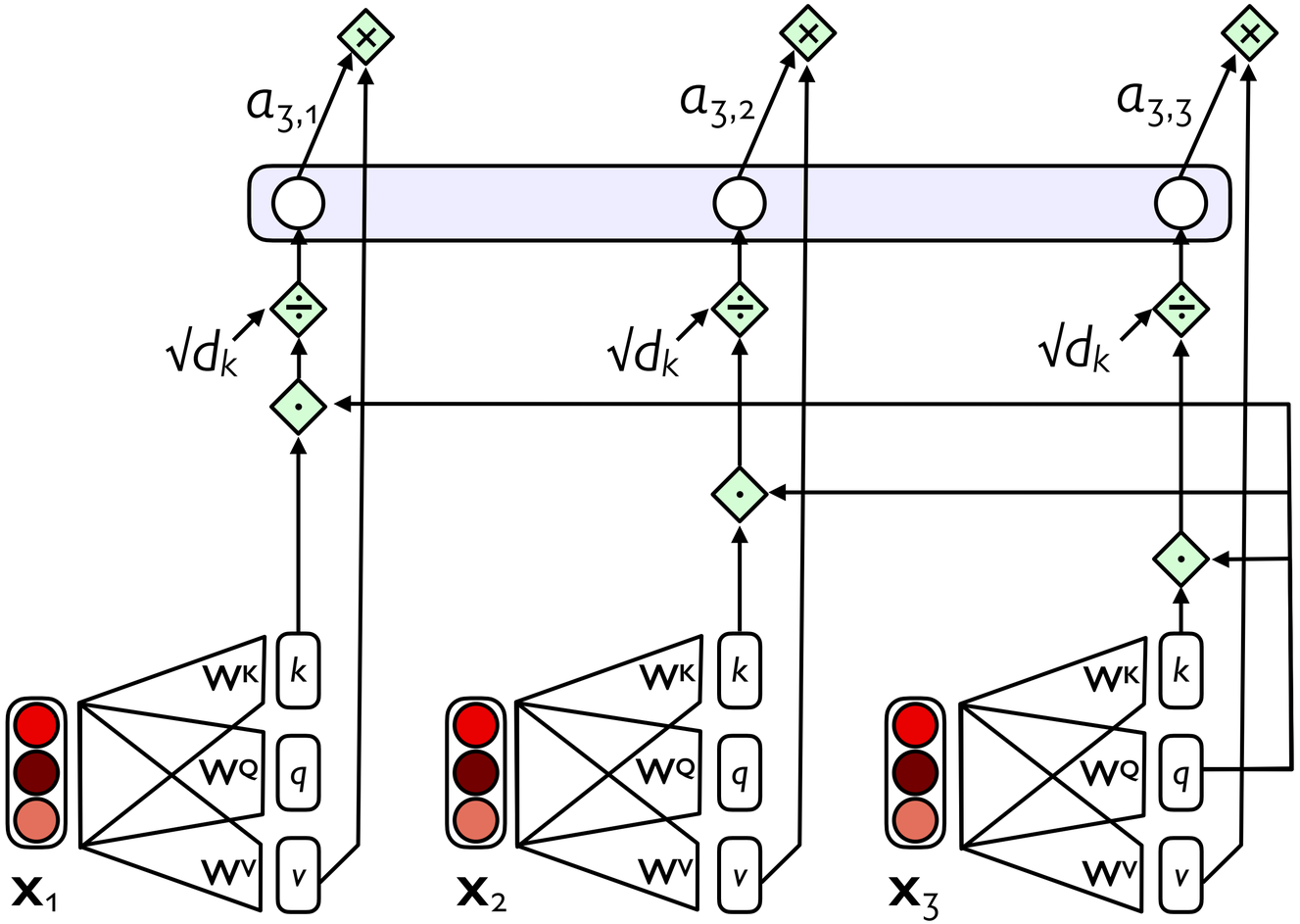
Weigh each value vector

Turn into a_{ij} weights via softmax

Divide score by $\sqrt{d_k}$

Compare $\mathbf{x}_3$'s query with the keys for $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$.

Project each $\mathbf{x}_i$ into key, query, and value vectors



Sum the weighted value vectors

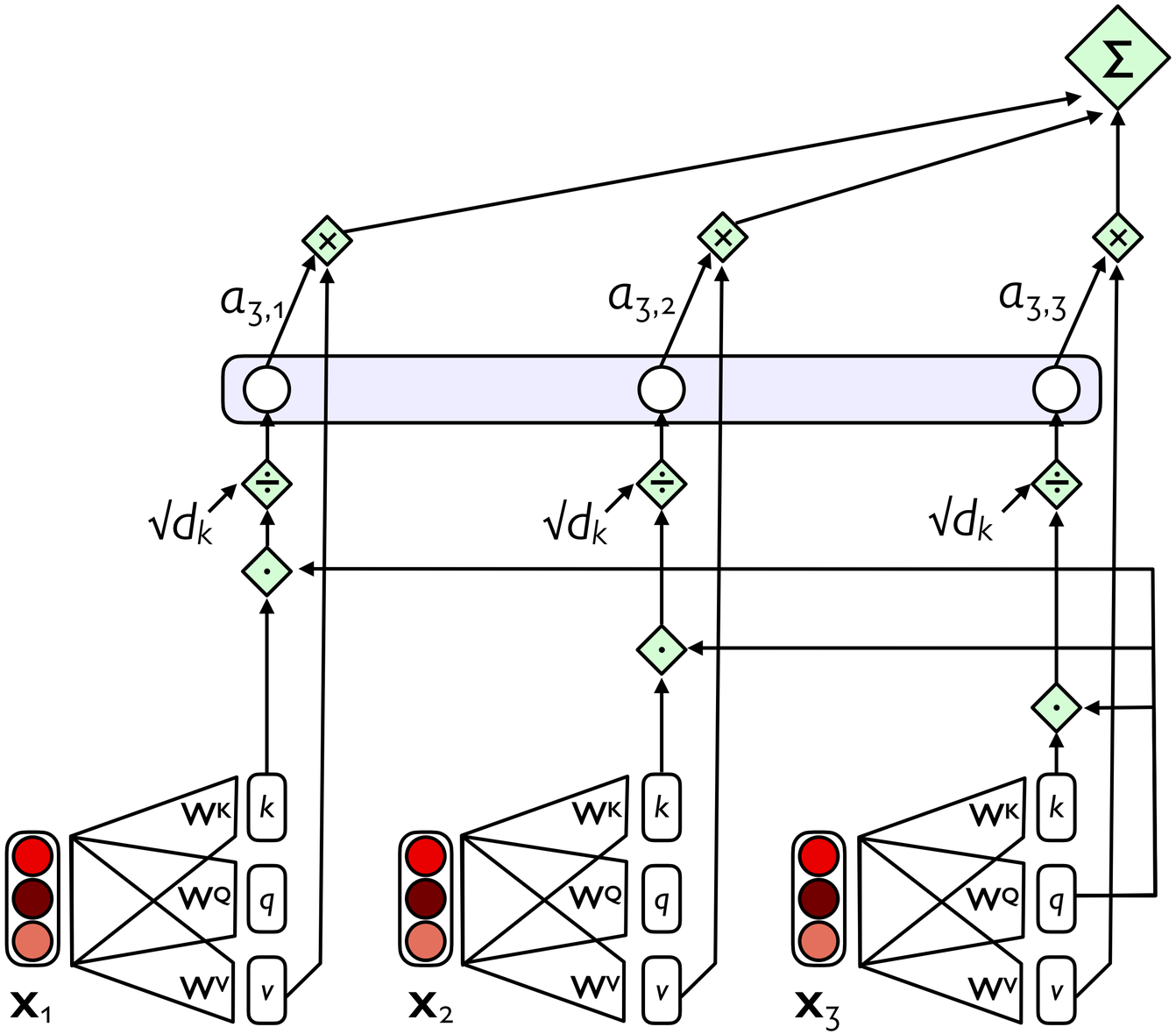
Weigh each value vector

Turn into a_{ij} weights via softmax

Divide score by $\sqrt{d_k}$

Compare $\mathbf{x}_3$'s query with the keys for $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$.

Project each $\mathbf{x}_i$ into key, query, and value vectors



Output of self-attention

Sum the weighted value vectors

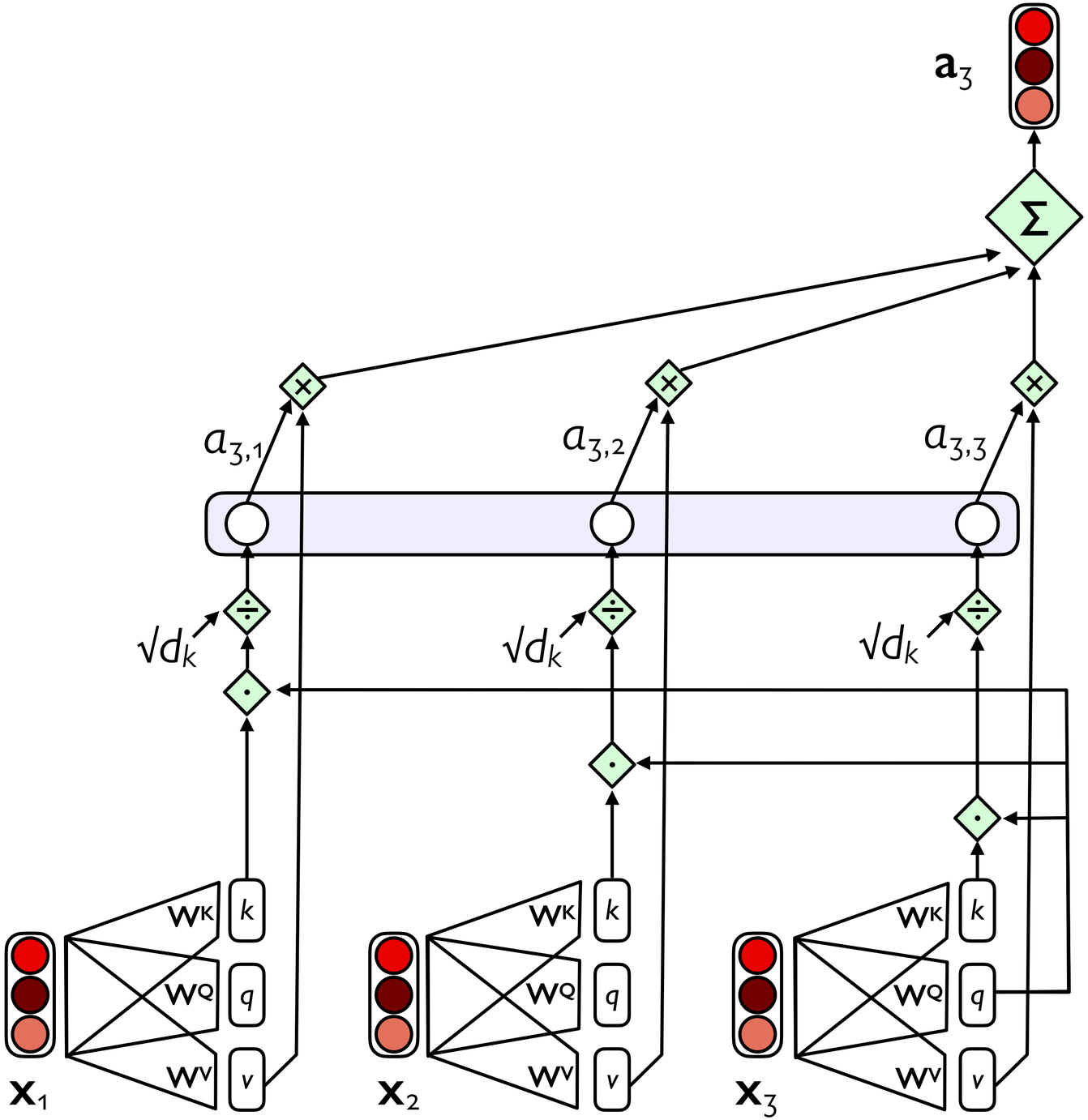
Weigh each value vector

Turn into a_{ij} weights via softmax

Divide score by $\sqrt{d_k}$

Compare $\mathbf{x}_3$'s query with the keys for $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$.

Project each $\mathbf{x}_i$ into key, query, and value vectors




vicki
@vboykis
They don’t tell you this in the paper (well they do but you have to read it like 15 times)
Multiplying
a lot of vectors
a lot of times
with scaled softmax
Attention
imgflip.com
6:20 PM · Feb 22, 2023 · 88.3K Views

In practice, instead of one attention head, we'll have lots of them: *multi-head attention*.

Why? Each head might be attending to the context for different purposes – different linguistic relationships or patterns in the context

Summary

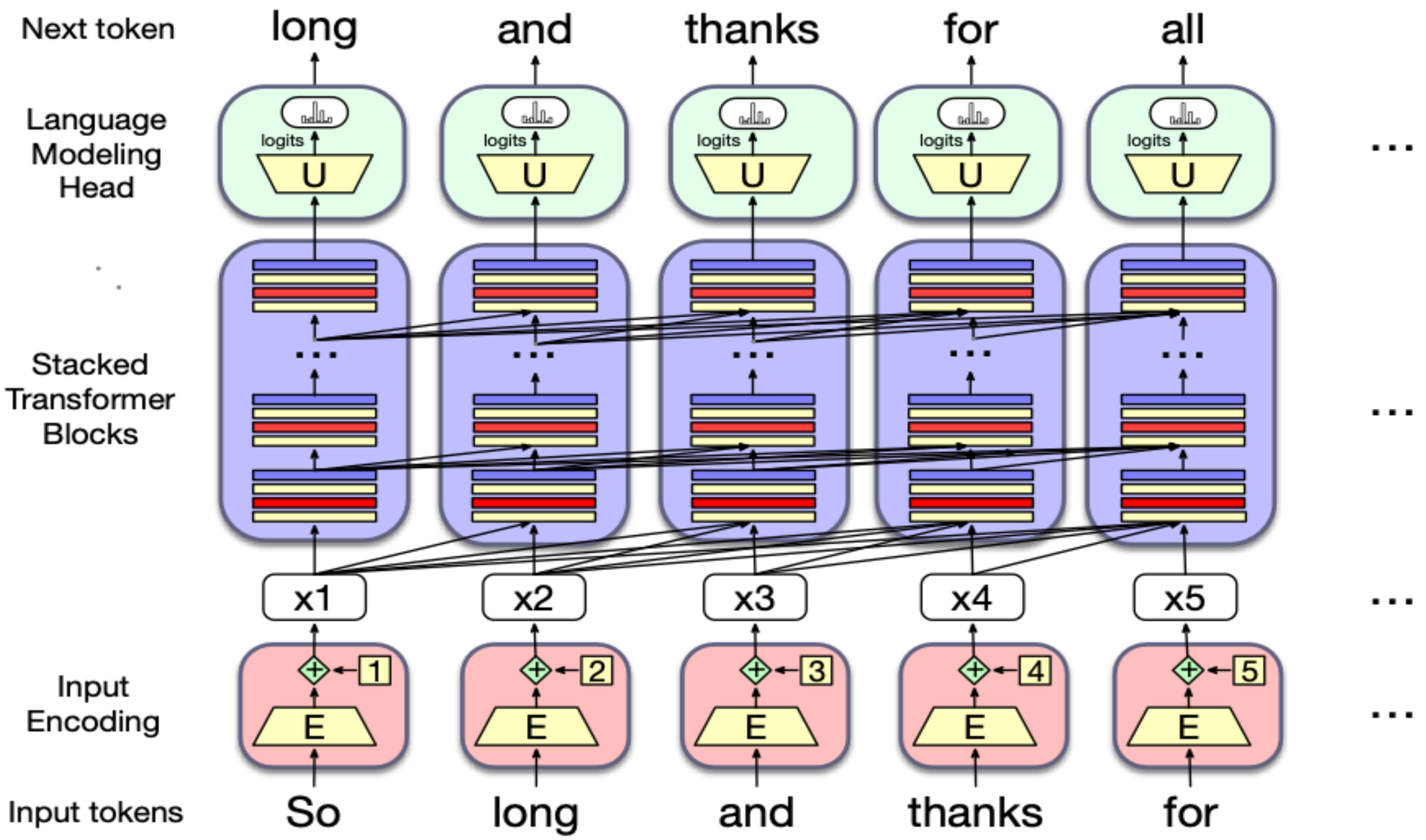
Attention is a method for enriching the representation of a token by incorporating contextual information

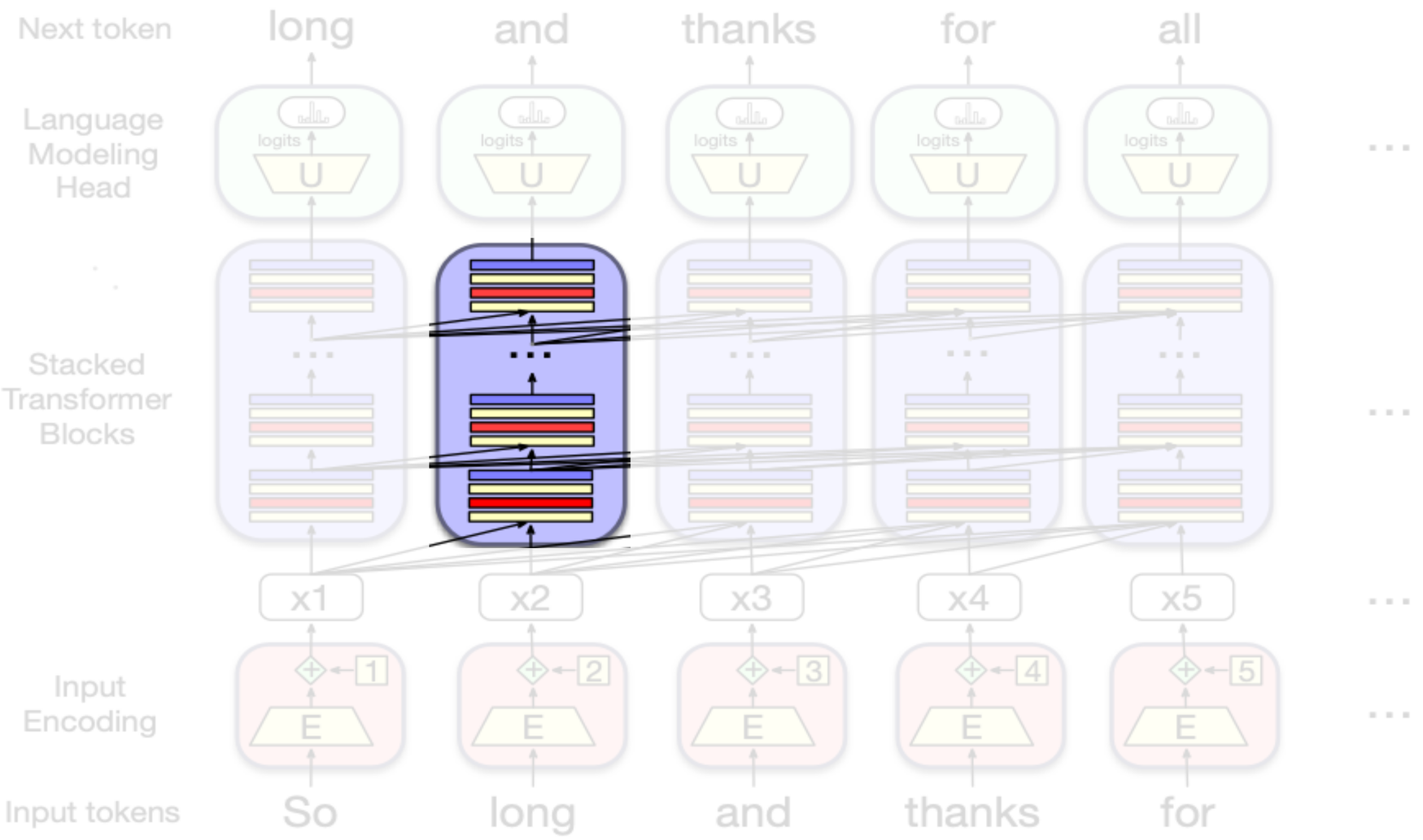
The result: the embedding for each word will be different in different contexts!

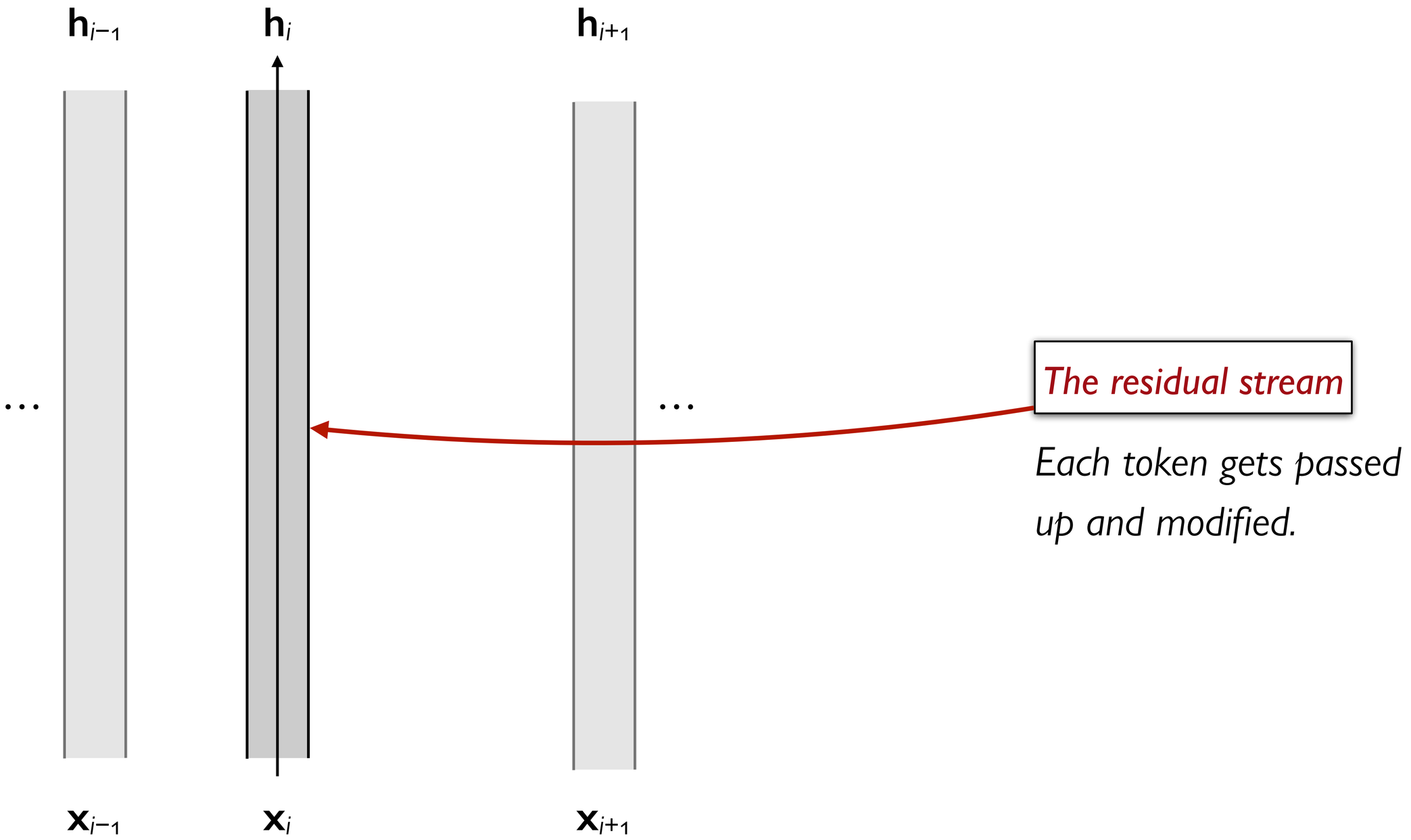
Contextual embeddings: a representation of word meaning in its context.

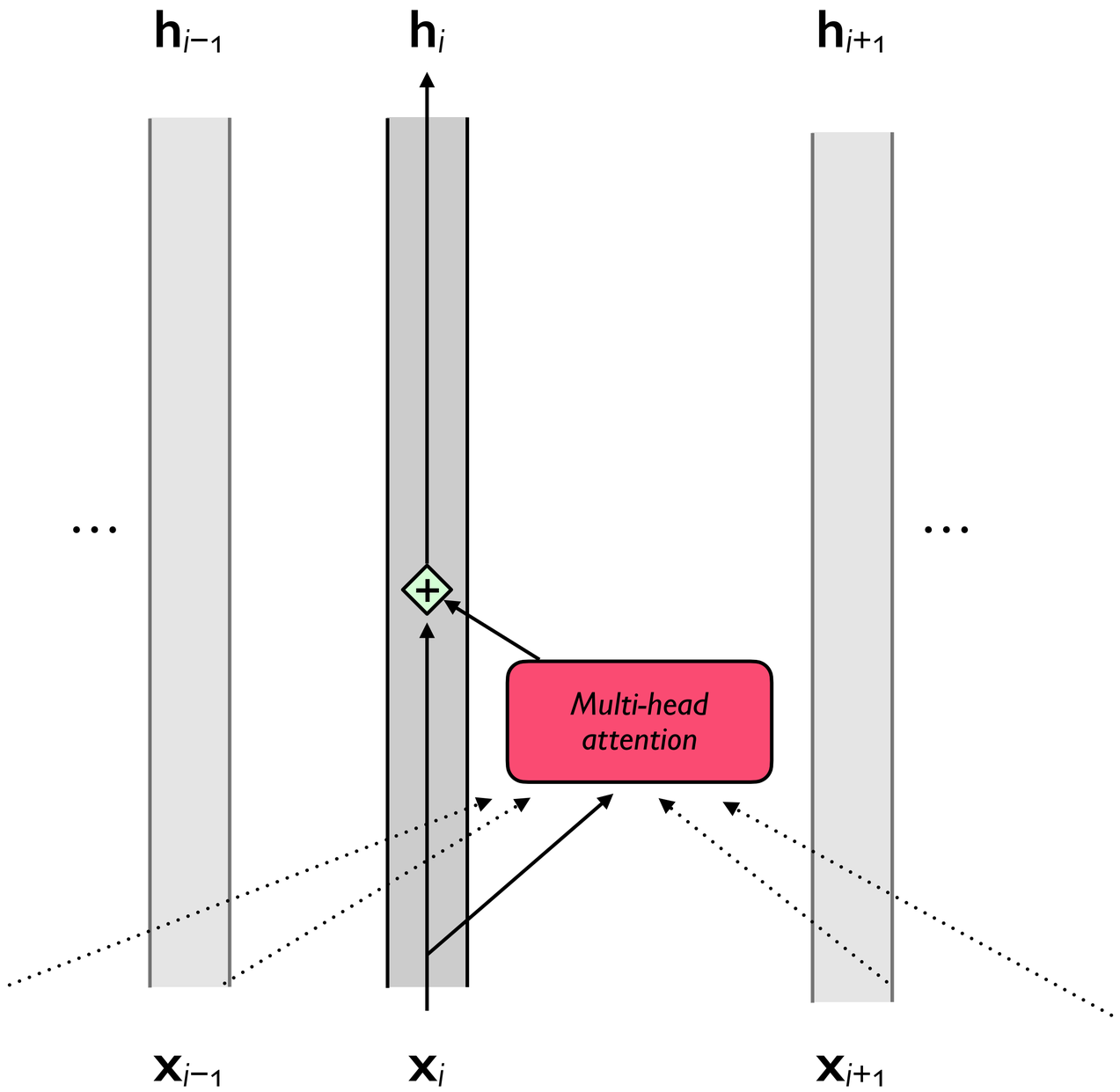
We'll see next that attention can also be viewed as a way to move information from one token to another.

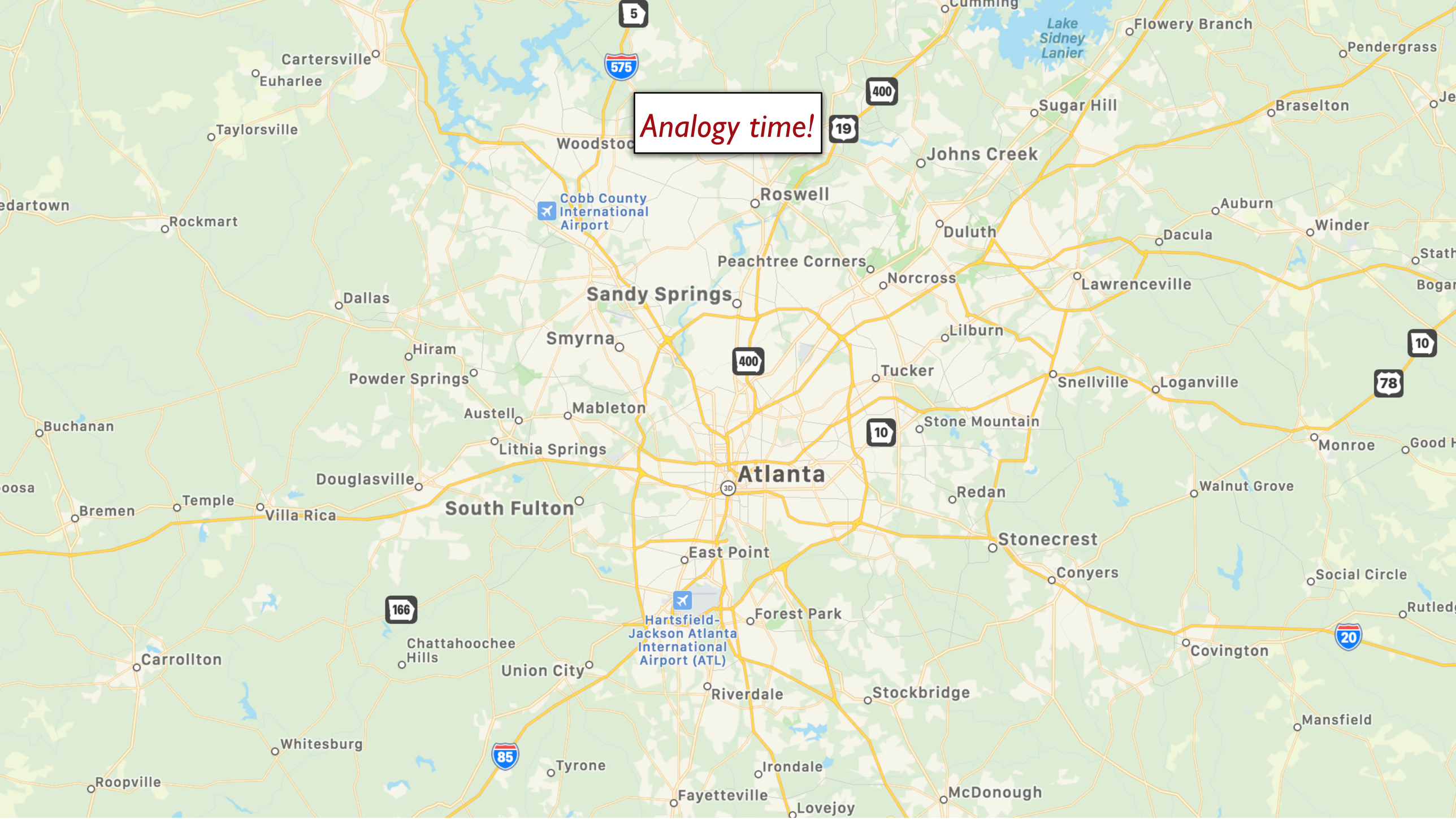
The transformer block

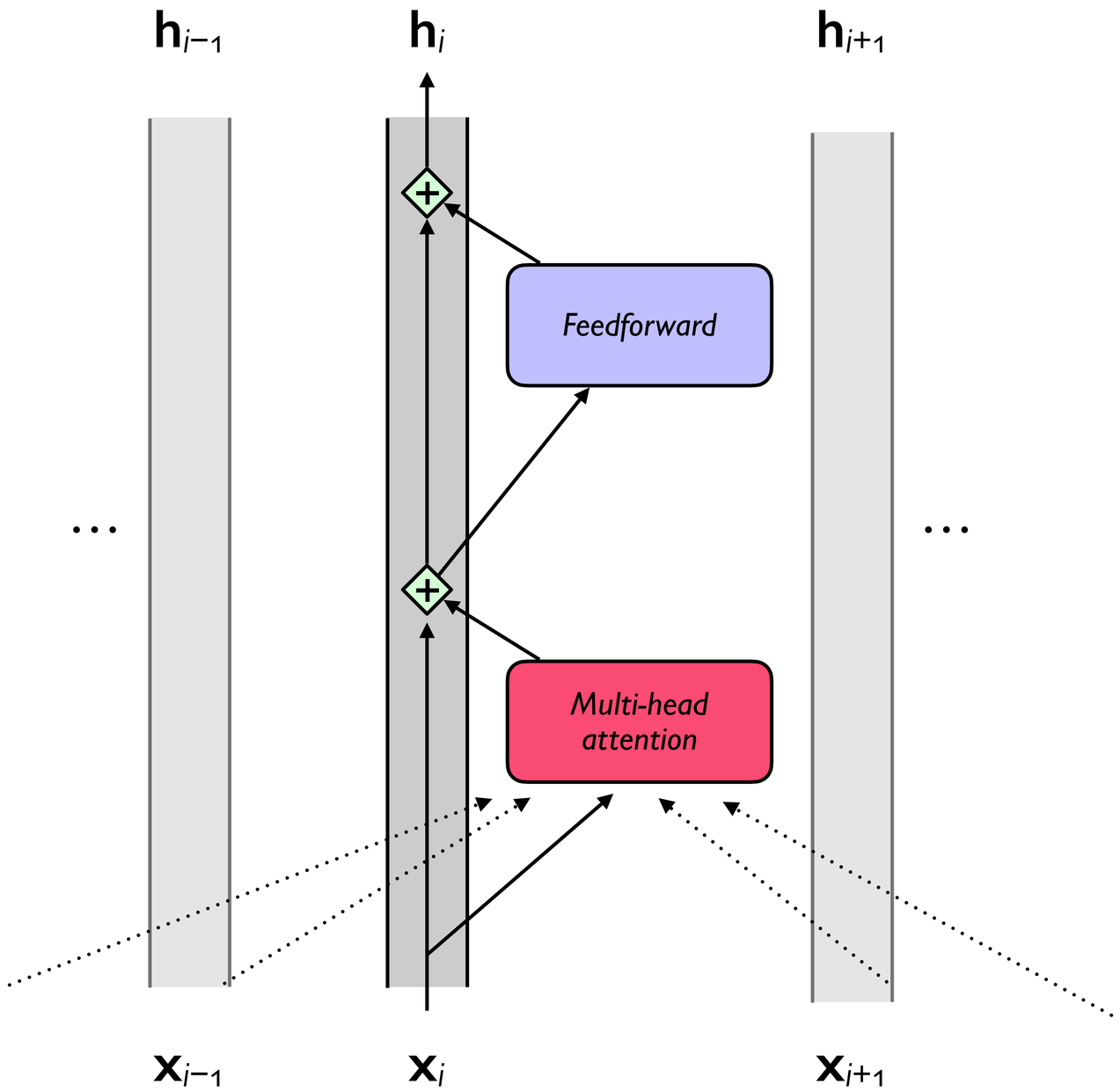

Next token
long
and
thanks
for
all
Language Modeling Head
logits
U
Stacked Transformer Blocks
x1
x2
x3
x4
x5
Input Encoding
1
2
3
4
5
E
Input tokens
So
long
and
thanks
for


Next token
long
and
thanks
for
all
Language Modeling Head
logits
U
Stacked Transformer Blocks
x1
x2
x3
x4
x5
Input Encoding
1
2
3
4
5
E
Input tokens
So
long
and
thanks
for


$\mathbf{h}_{i-1}$
$\mathbf{h}_i$
$\mathbf{h}_{i+1}$
...
...
$\mathbf{x}_{i-1}$
$\mathbf{x}_i$
$\mathbf{x}_{i+1}$
The residual stream
Each token gets passed up and modified.

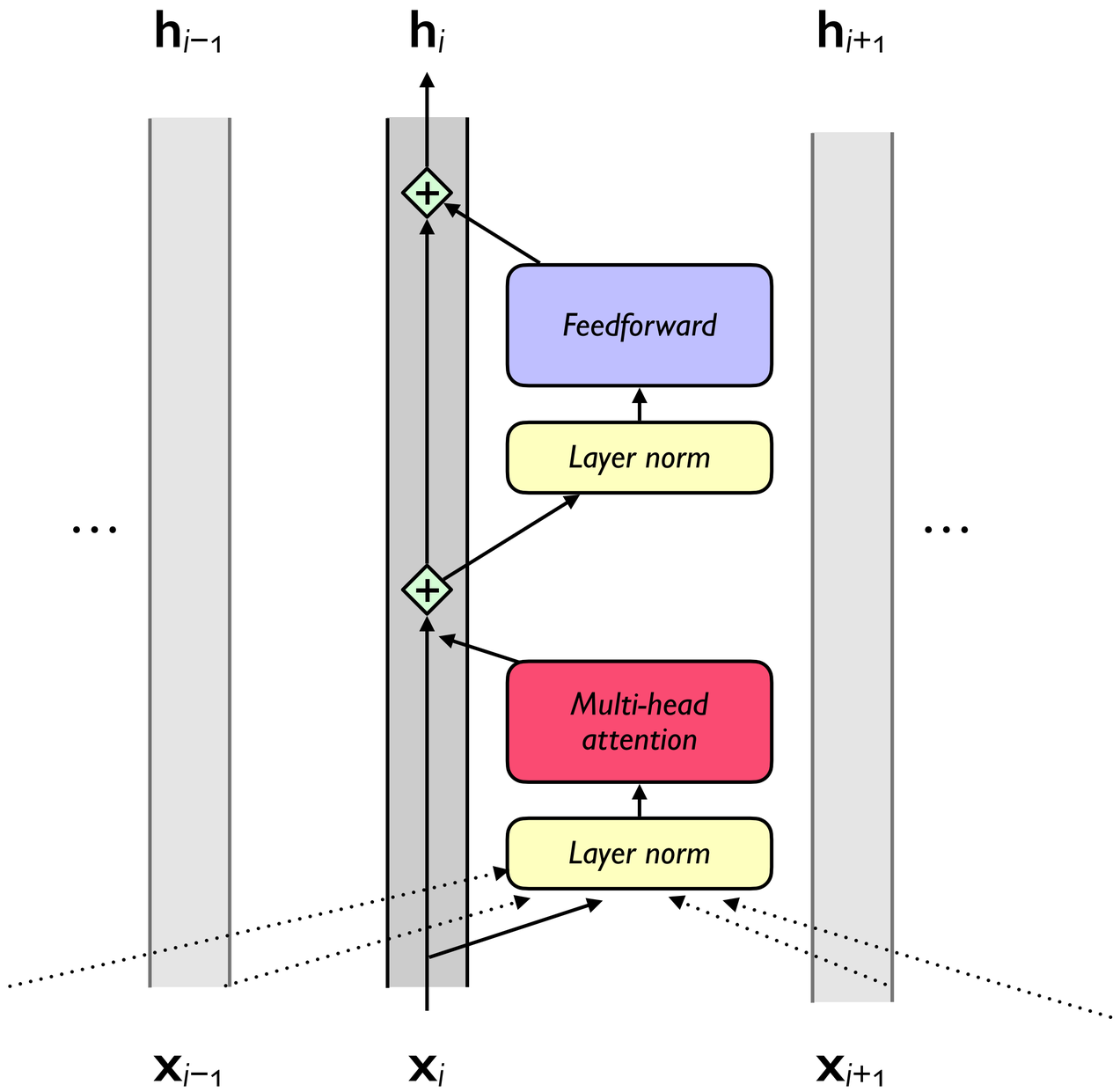

$\mathbf{h}_{i-1}$
$\mathbf{h}_i$
$\mathbf{h}_{i+1}$
...
...
Multi-head attention
$\mathbf{x}_{i-1}$
$\mathbf{x}_i$
$\mathbf{x}_{i+1}$


Analogy time!
Cartersville
Euharlee
Taylorsville
Woodstock
Cobb County International Airport
Roswell
Johns Creek
Sugar Hill
Lake Sidney Lanier
Flowery Branch
Pendergrass
Braselton
Auburn
Winder
Dacula
Duluth
Peachtree Corners
Norcross
Lawrenceville
Rockmart
Dallas
Sandy Springs
Smyrna
Hiram
Powder Springs
Lilburn
Tucker
Snellville
Loganville
Austell
Mableton
Lithia Springs
Stone Mountain
Monroe
Buchanan
Douglasville
Atlanta
Redan
Walnut Grove
Bremen
Temple
Villa Rica
South Fulton
Stonecrest
East Point
Conyers
Social Circle
Hartsfield-Jackson Atlanta International Airport (ATL)
Forest Park
Chattahoochee Hills
Carrollton
Union City
Covington
Riverdale
Stockbridge
Mansfield
Whitesburg
Tyrone
Irondale
Roopville
Fayetteville
Lovejoy
McDonough
5
575
400
19
10
78
166
85
20



We'll need nonlinearities, so add a feedforward layer

$$\text{FFN}(\mathbf{x}_i) = \text{ReLU}(\mathbf{x}_i\mathbf{W}_1 + b_1)\mathbf{W}_2 + b_2$$



Layer norm

The vector $\mathbf{x}_i$ is normalized twice

Layer norm is a variation of the z-score from statistics, applied to a single vector in a hidden layer:

mean

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

standard deviation

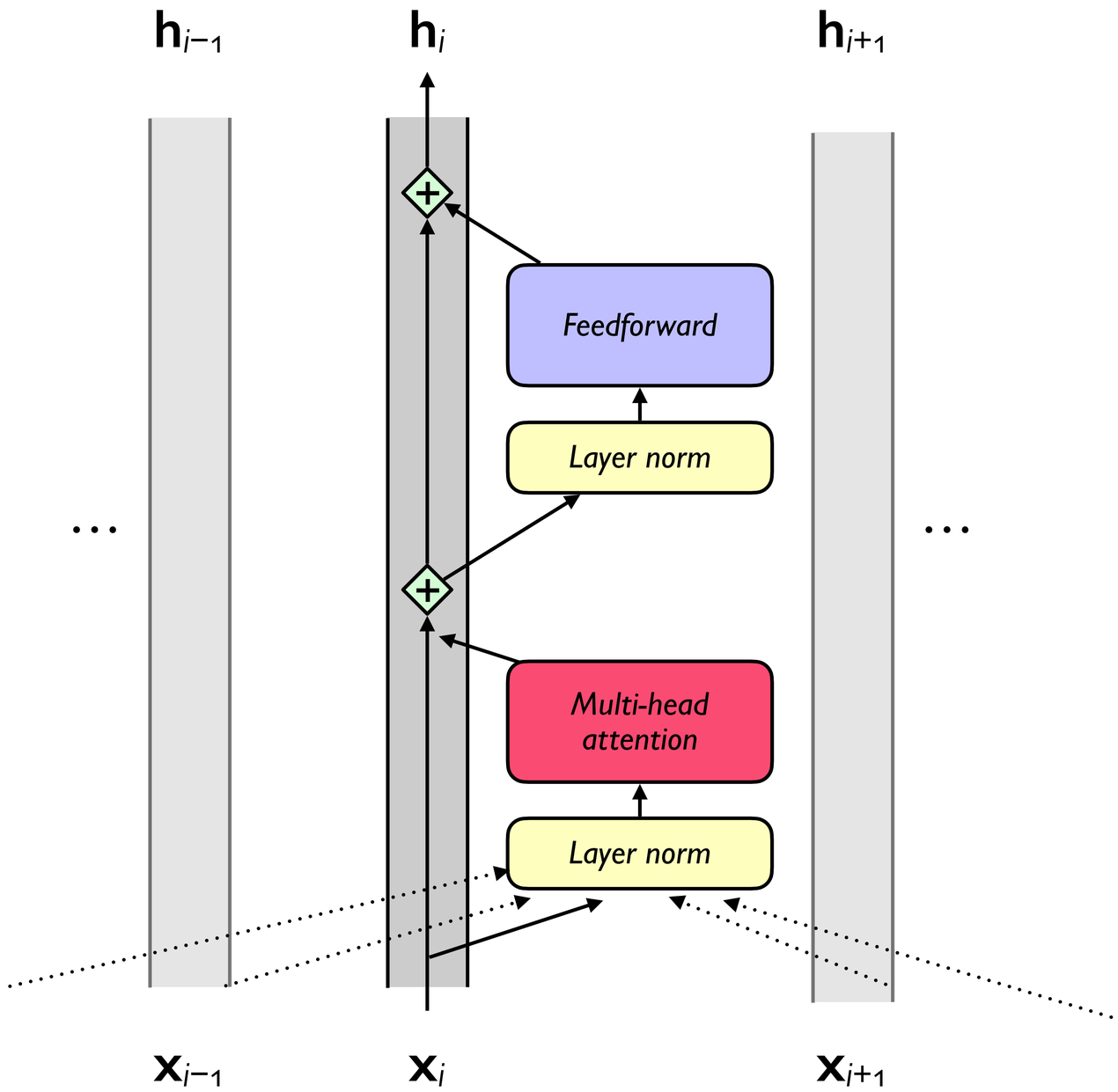
$$\sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2}$$

normalized

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

normalized in a tunable way

$$\text{LayerNorm}(\mathbf{x}) = \gamma \frac{(\mathbf{x} - \mu)}{\sigma} + \beta$$



$$\mathbf{h}_i = \mathbf{t}_i^5 + \mathbf{t}_i^3$$

$$\mathbf{t}_i^5 = \text{FFN}(\mathbf{t}_i^4)$$

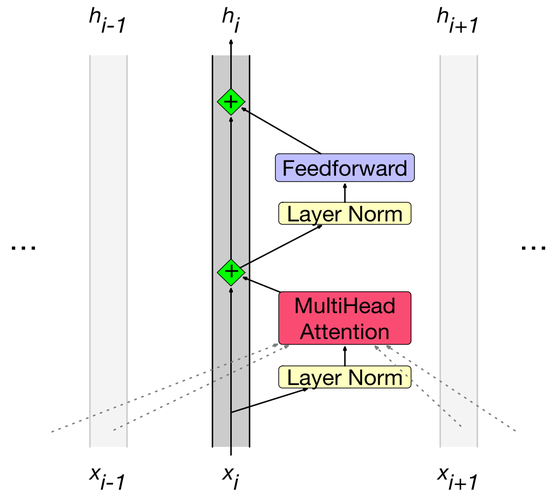
$$\mathbf{t}_i^4 = \text{LayerNorm}(\mathbf{t}_i^3)$$

$$\mathbf{t}_i^3 = \mathbf{t}_i^2 + \mathbf{x}_i$$

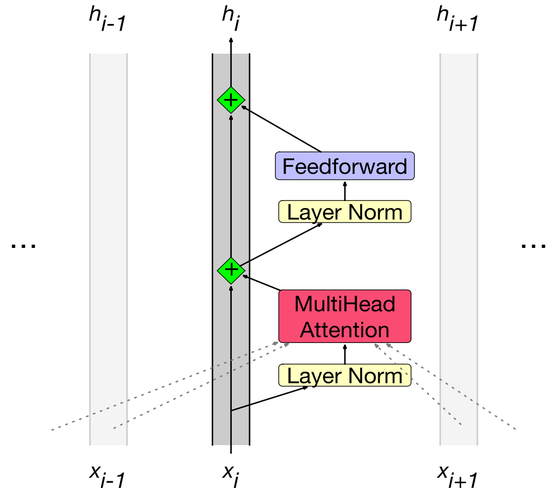
$$\mathbf{t}_i^2 = \text{MultiHeadAttention}(\mathbf{t}_i^1, [\mathbf{t}_1^1, \ldots, \mathbf{t}_N^1])$$

$$\mathbf{t}_i^1 = \text{LayerNorm}(\mathbf{x}_i)$$

Block 2



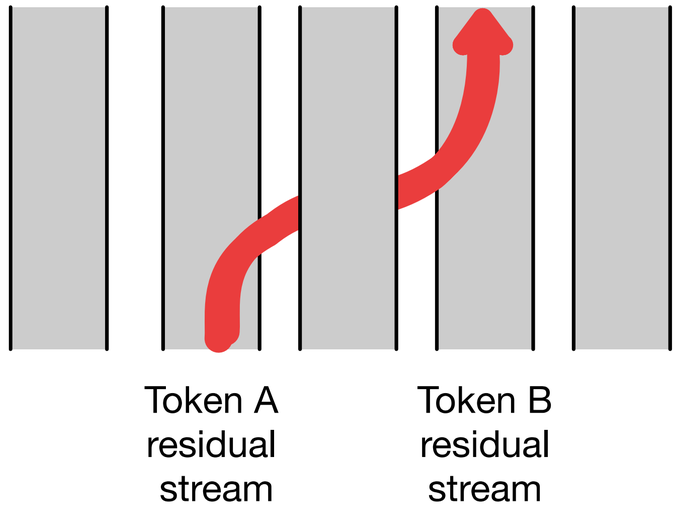
Block 1



A transformer is a stack of these blocks – so all the vectors are of the same dimensionality d

Notice that all parts of the transformer block apply to 1 residual stream (1 token) – except attention, which takes information from other tokens.

Elhage et al. (2021) show that we can view attention heads as literally moving information from the residual stream of a neighboring token into the current stream:



Acknowledgments

The lecture incorporates material from:

Jurafsky & Martin, *Speech and Language Processing*, 3rd ed. draft

