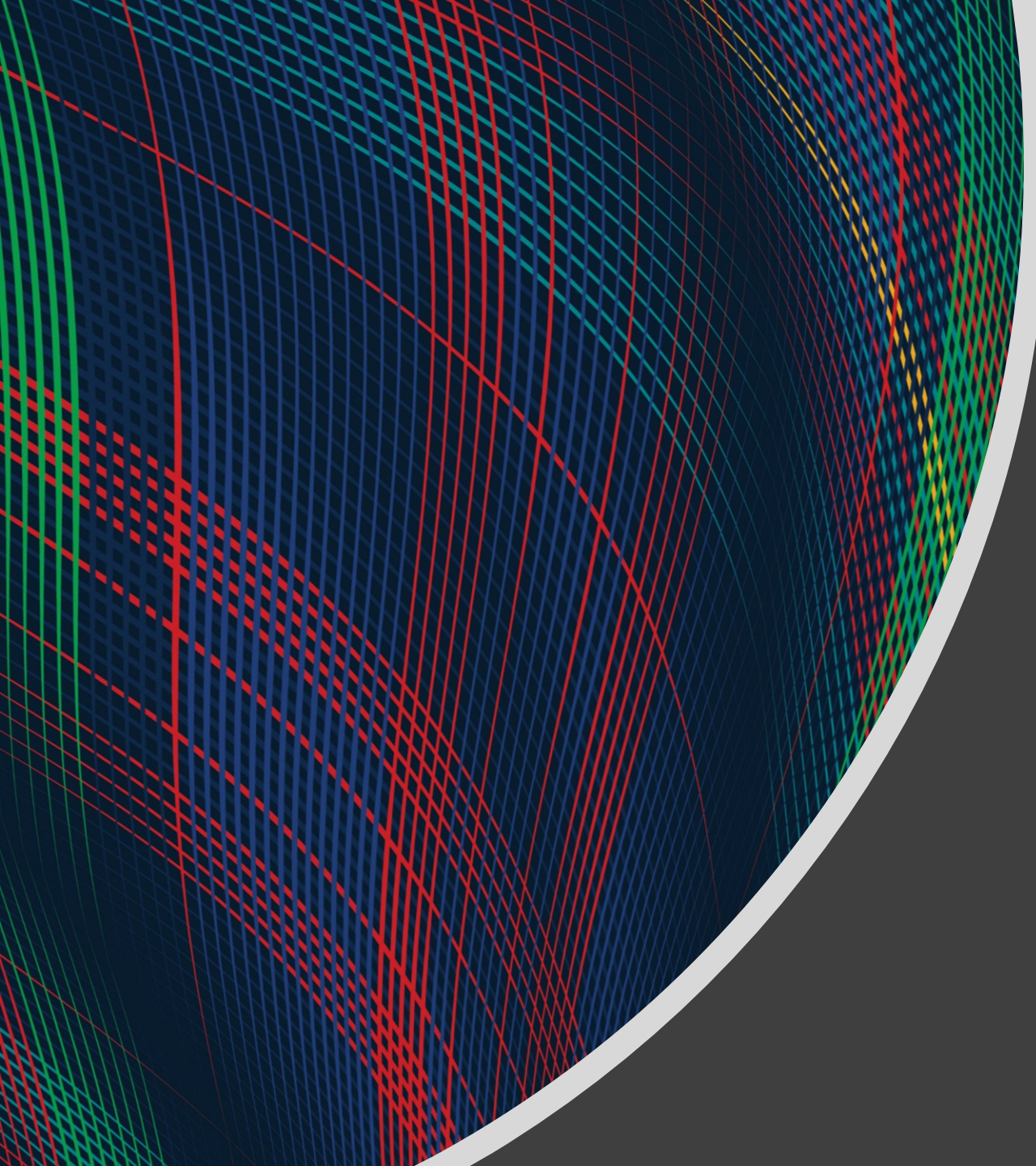


An abstract graphic on the left side of the slide, featuring a sphere-like shape composed of a dense grid of intersecting red, green, and blue lines. The lines are curved and follow the contour of the sphere, creating a complex, woven pattern. The sphere is set against a dark gray background.

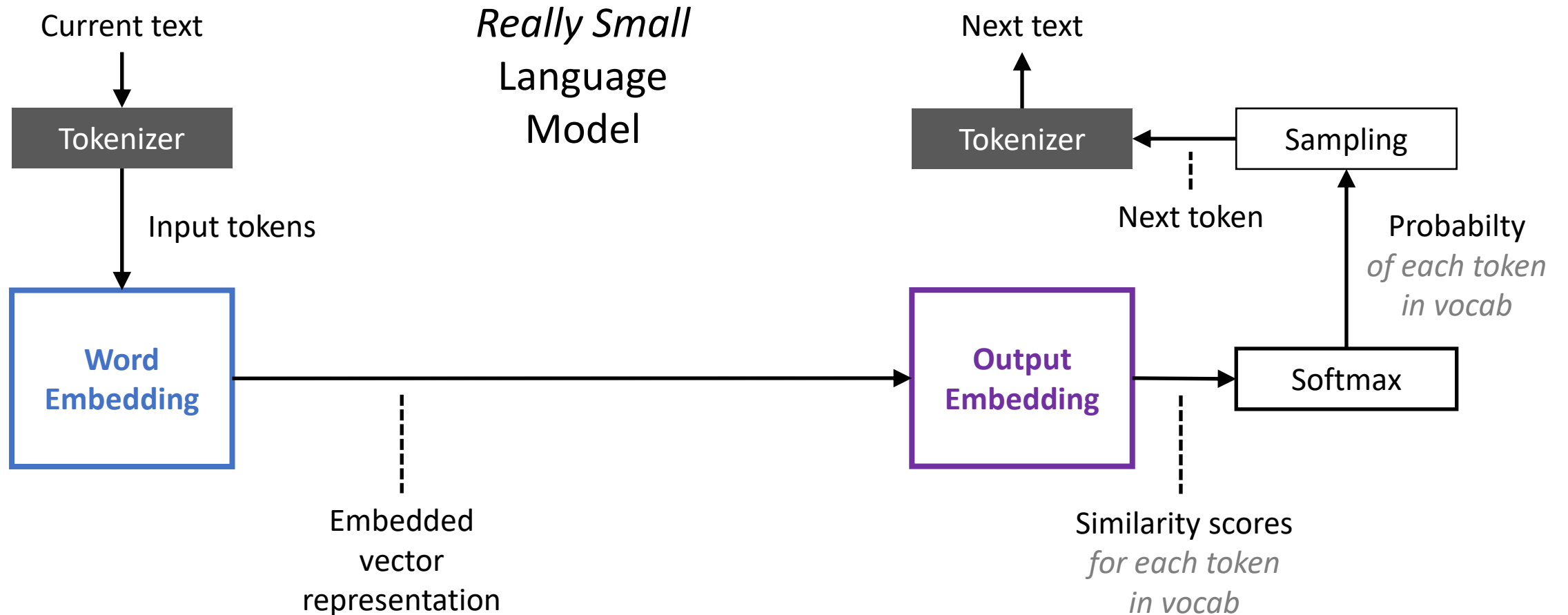
07-280
AI & ML I

Attention & Transformers

Instructors:
Pat Virtue & Nihar Shah

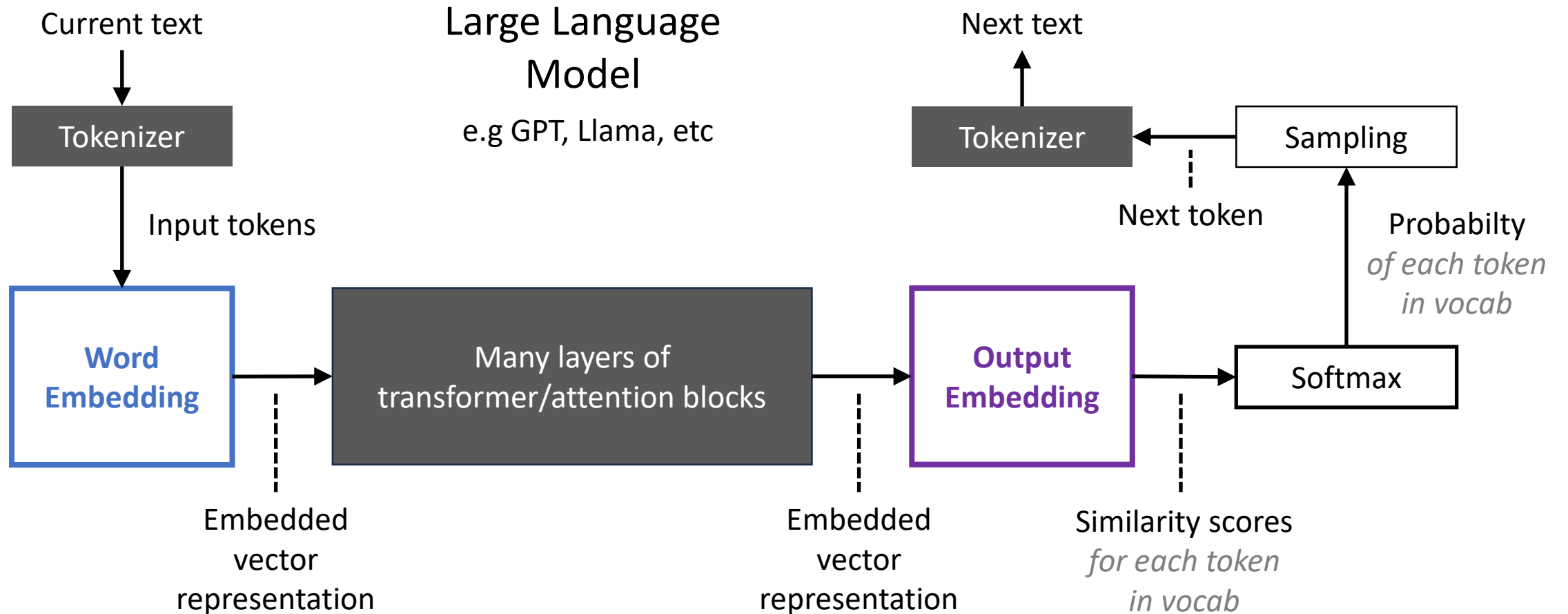
Simple Word Embedding LM

Building a language model with just word embedding layers 😊



Word (Token) Embeddings

The beginning and the end of LLM networks



Large Language Models!

Last time

Word Embeddings

Today

Increasing the context size

- Positional Encoding
- Attention

Building GPT2!

- Multiheaded attention
- Transformers



Context size > 1

Transformer Language Models

Increasing context size

- Uniform average of context vectors
- Position encoding

Attention

- Weighted average of context vectors
- Query Keys Values
- Expressive power of linear transforms

Building GPT2!

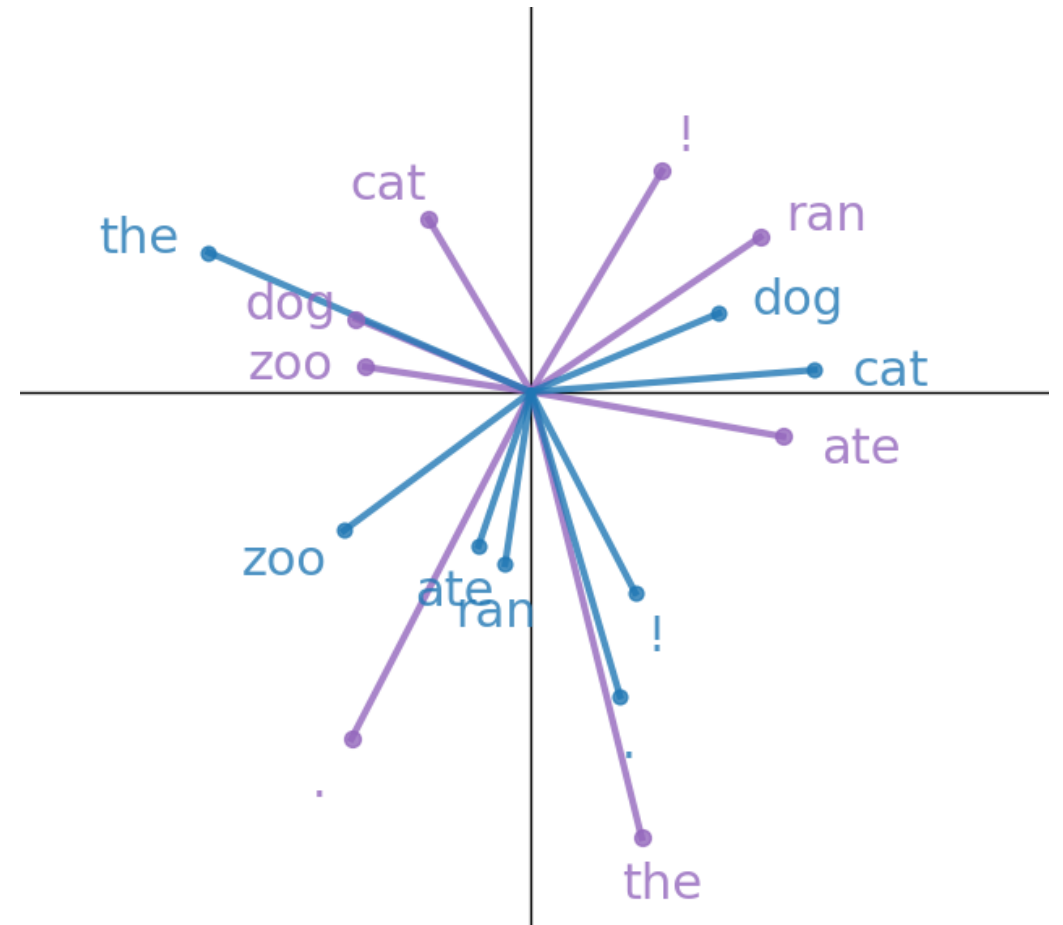
Increasing Context Size

What if we want to have more input tokens?

V : previous tokens

U : next tokens

V:		
!:	0.884,	0.196
::	0.358,	-2.343
ate:	-1.085,	0.560
cat:	0.939,	-0.978
dog:	0.503,	0.406
ran:	0.323,	-0.493
the:	-0.792,	-0.842
zoo:	-1.280,	0.246

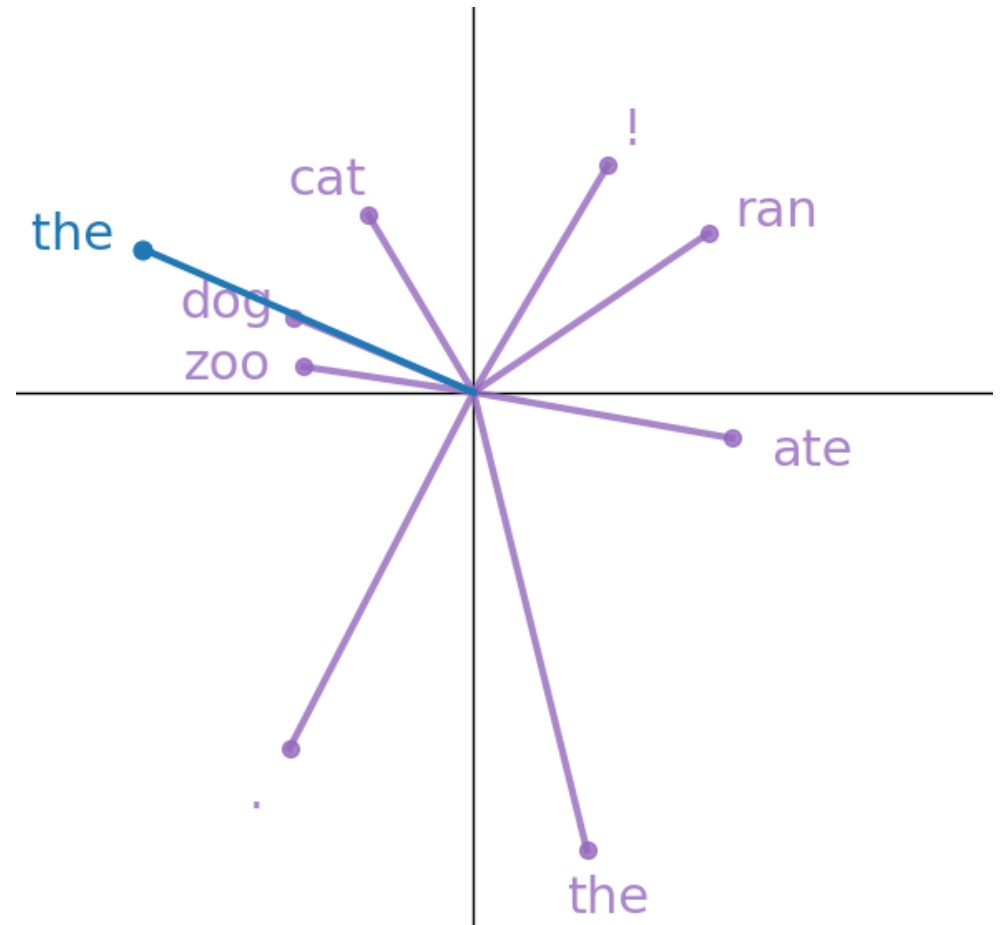


Increasing Context Size

What if we want to have more input tokens?

V: for context

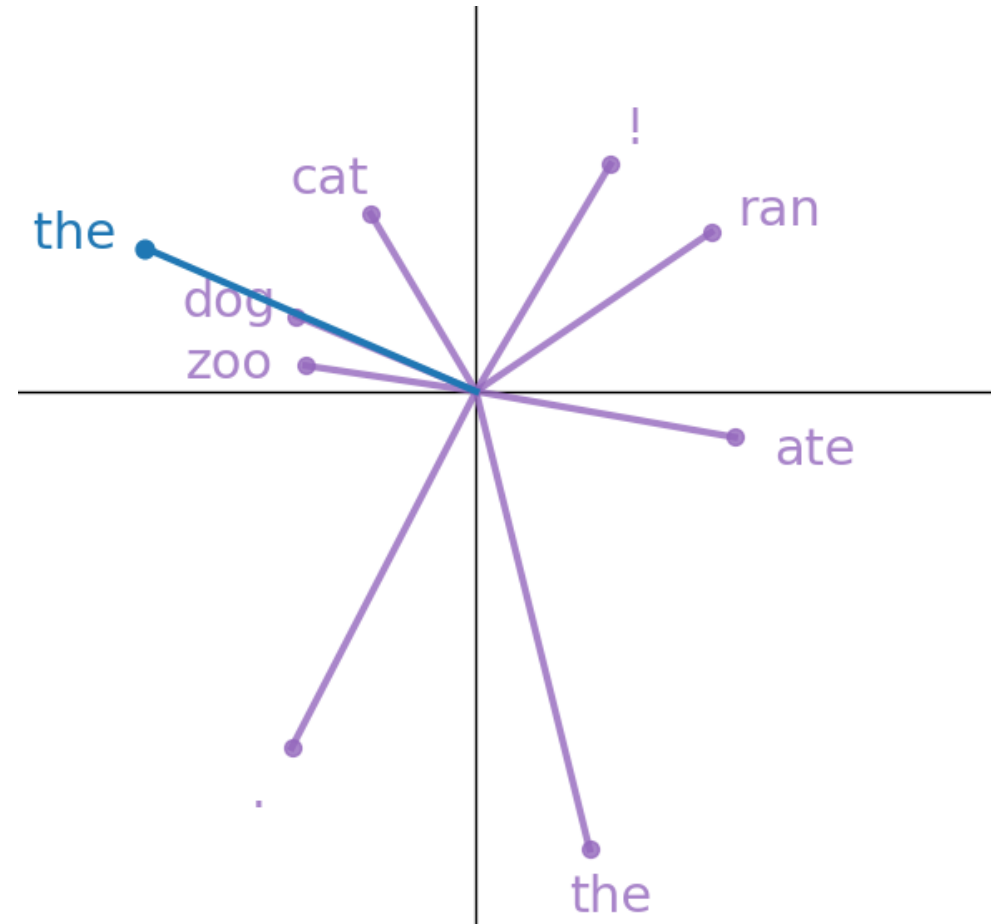
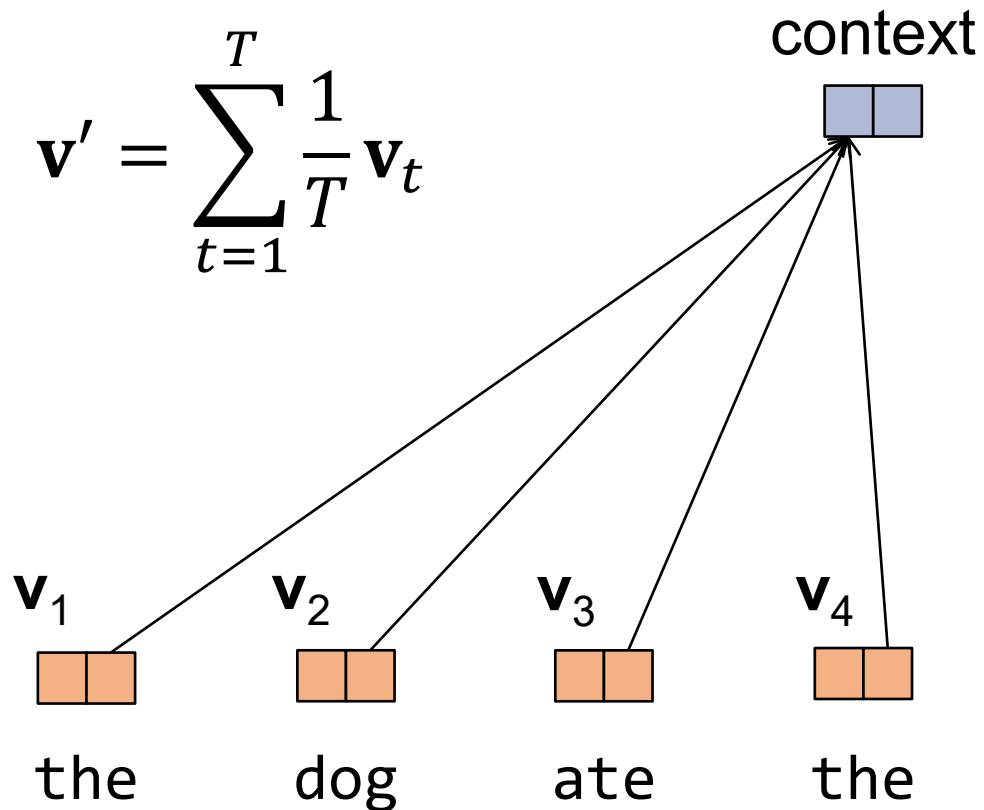
U: for next token



Increasing Context Size

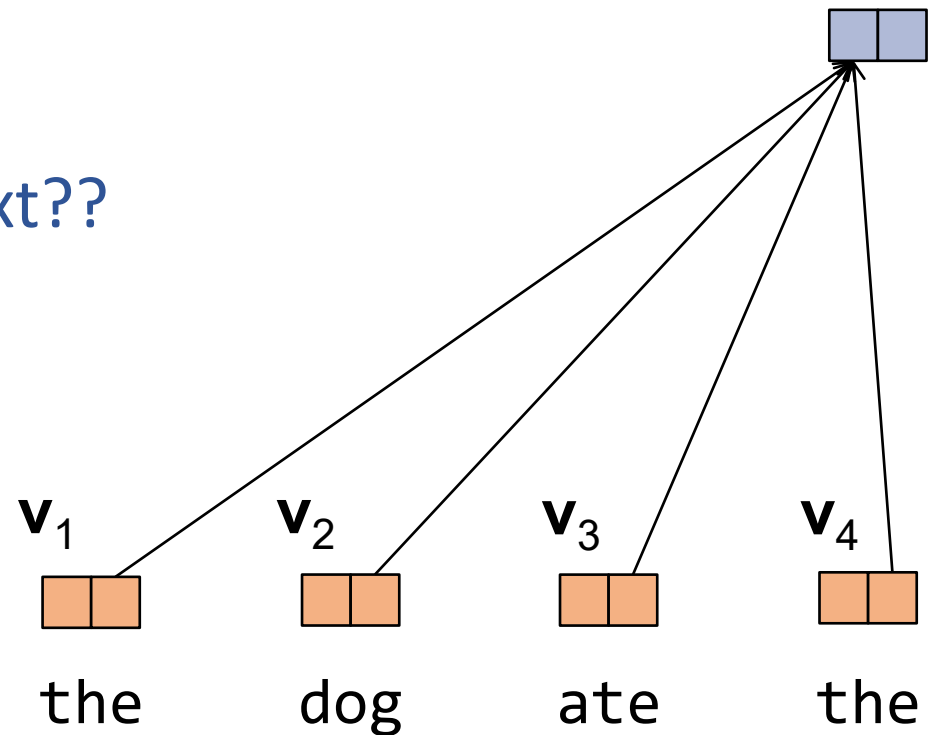
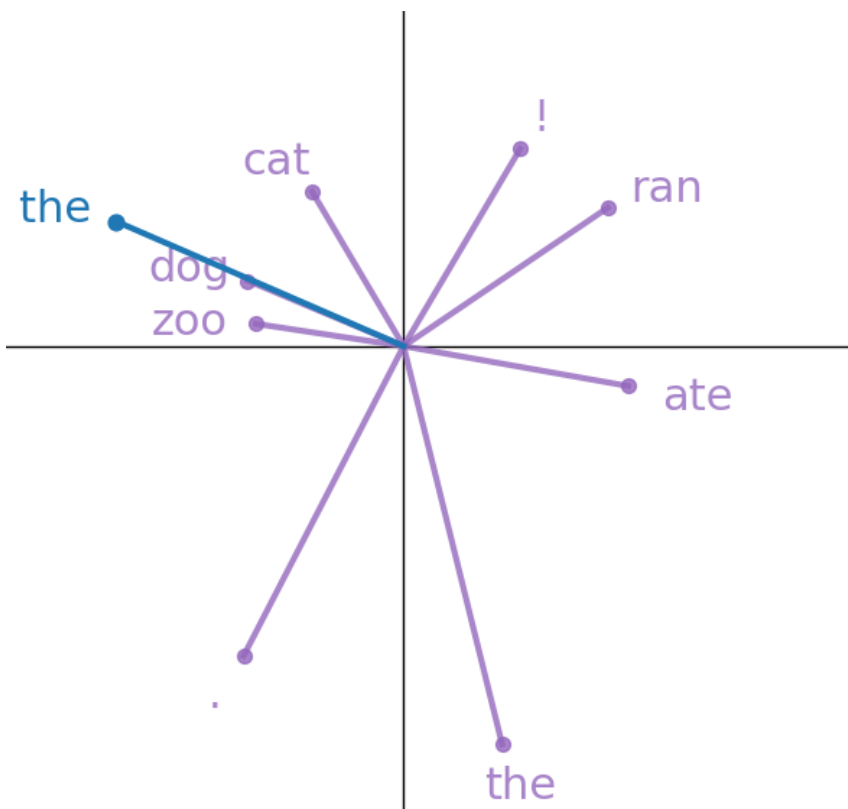
What if we want to have more input tokens?

Uniform average over T input context tokens



Position Encoding

What about position within the input context??

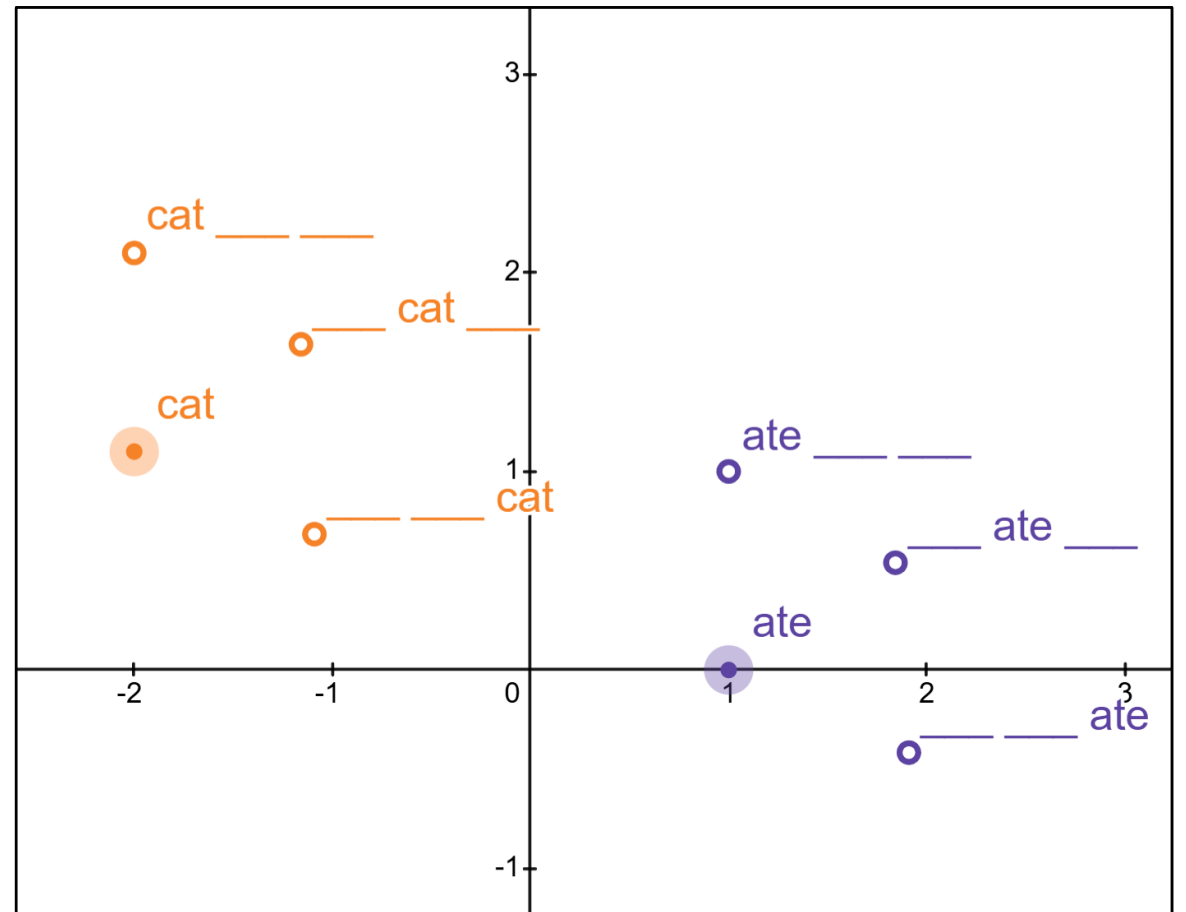


Positional Encoding

2D example of positional encoding being added to token

<https://www.desmos.com/calculator/8e1fafb2fe>

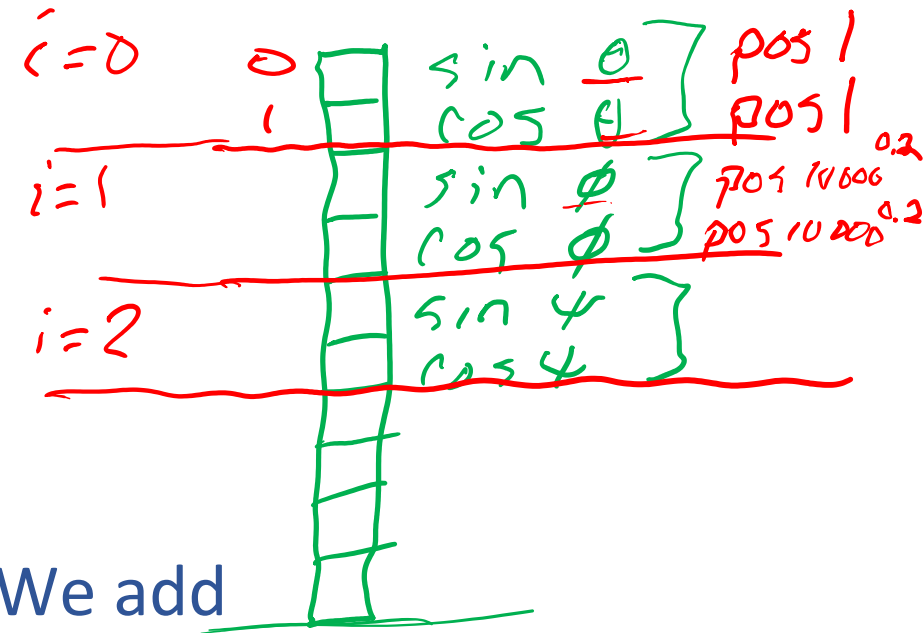
$$\begin{aligned} \mathbf{v}'_{cat_} &= \mathbf{v}_{cat} + \begin{bmatrix} \sin(pos) \\ \cos(pos) \end{bmatrix} \\ \mathbf{v}'_{_cat_} &= \mathbf{v}_{cat} + \begin{bmatrix} \sin(0) \\ \cos(0) \end{bmatrix} \\ \mathbf{v}'_{_cat_} &= \mathbf{v}_{cat} + \begin{bmatrix} \sin(1) \\ \cos(1) \end{bmatrix} \\ \mathbf{v}'_{_cat_} &= \mathbf{v}_{cat} + \begin{bmatrix} \sin(2) \\ \cos(2) \end{bmatrix} \end{aligned}$$



Positional Encoding

What about position within the input context??

Position Encoding Desmos Demo 



Transformers have no built-in notion of position. We add **sinusoidal positional encodings** from Vaswani et al. 2017

$$d_{model} = 10$$

so each position gets a unique pattern of sines and cosines:

$$PE_{2i} = \sin\left(pos / 10000^{\frac{2i}{d_{model}}}\right)$$

$$PE_{2i+1} = \cos\left(pos / 10000^{\frac{2i}{d_{model}}}\right)$$

Handwritten notes for the equations above:

- For PE_{2i} , a red arrow points to the $2i$ in the subscript, and a red 0 is written above it.
- For PE_{2i+1} , a red arrow points to the $2i+1$ in the subscript, and a red 1 is written below it.
- Below the equations, there are four columns of handwritten calculations for $i=1, 2, 3, 4$:

$\frac{2}{10}$	$\frac{4}{10}$	$\frac{6}{10}$	$\frac{8}{10}$
$\frac{2}{10}$	$\frac{4}{10}$	$\frac{6}{10}$	$\frac{8}{10}$

Rotary Position Encoding

What about position within the input context??

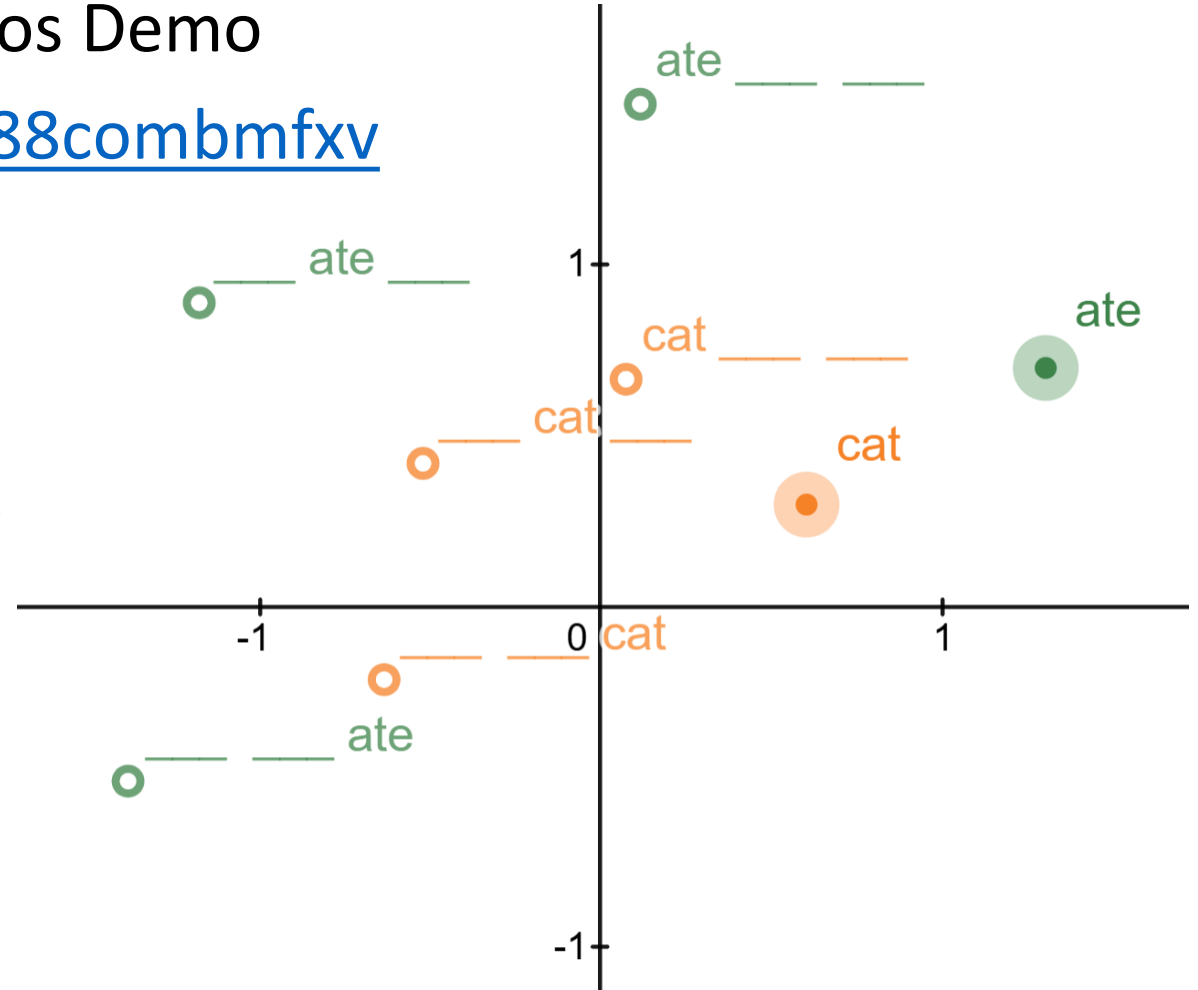
Rotary Position Encoding (RoPE) Desmos Demo

<https://www.desmos.com/calculator/88combmfxfv>

2D version:

Given a fixed base rotation angle θ_1 ,
an embedded vector $\mathbf{x}$ at integer position,
 i_{pos} , will be rotated by angle $\theta = i_{pos}\theta_1$:

$$\mathbf{x}' = \text{Rotate}(\mathbf{x}, \theta) = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \mathbf{x}$$



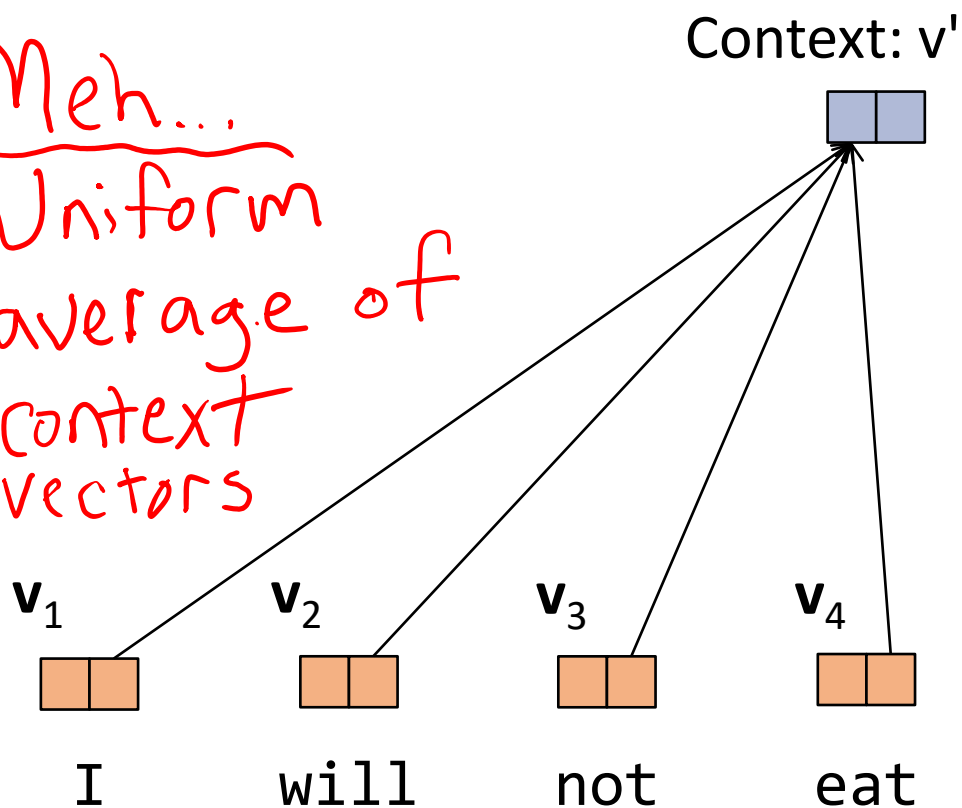
Attention

Learn to pay attention!

We can do better than uniform combination of input

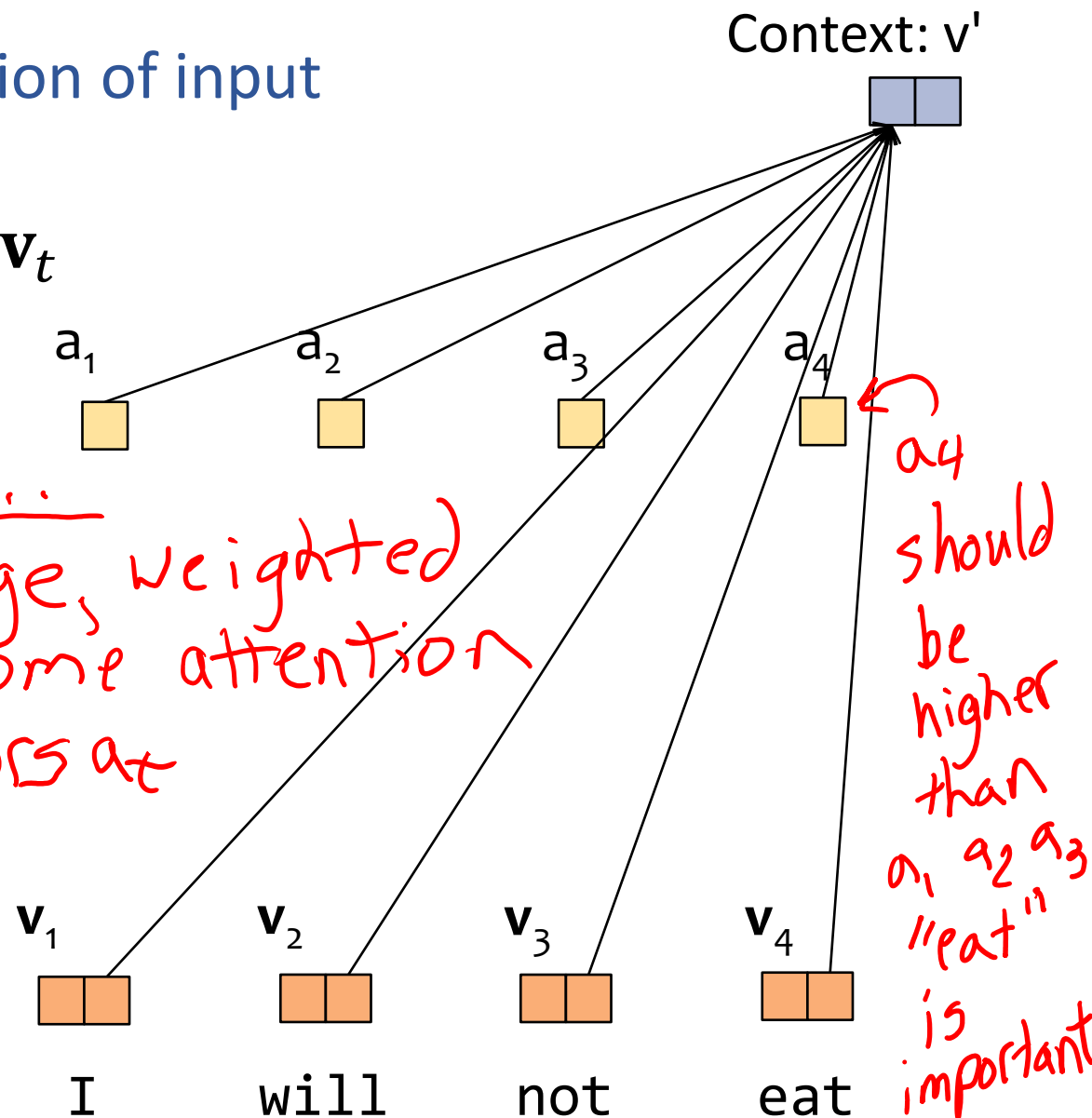
$$\mathbf{v}' = \sum_{t=1}^T \frac{1}{T} \mathbf{v}_t$$

Meh...
Uniform
average of
context
vectors



$$\mathbf{v}' = \sum_{t=1}^T a_t \mathbf{v}_t$$

Want...
Average, weighted
by some attention
factors a_t

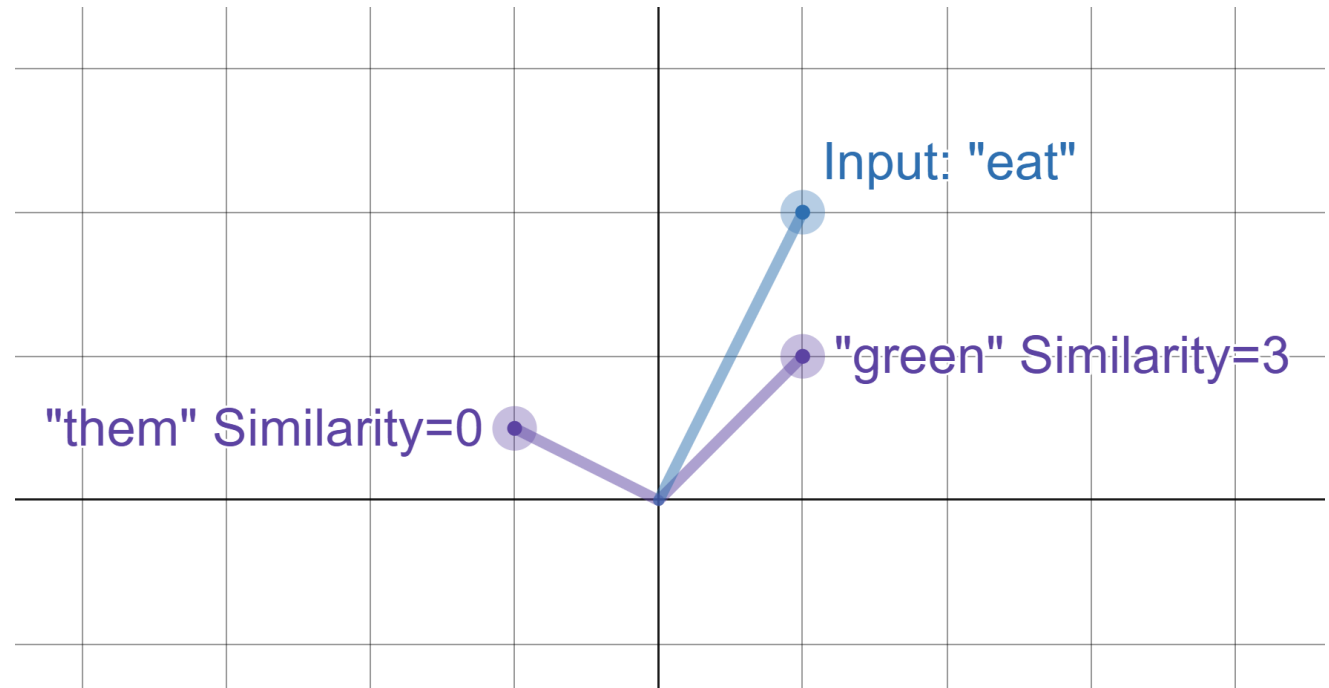


Reminder: Poll 1

Take 1: Collecting vector data (about you!)

Take 2: Compute similarity score with yourself and your neighbor

$$f(\mathbf{u}, \mathbf{v}) = \mathbf{u}^T \mathbf{v}$$



Reminder: Similarity Matrix

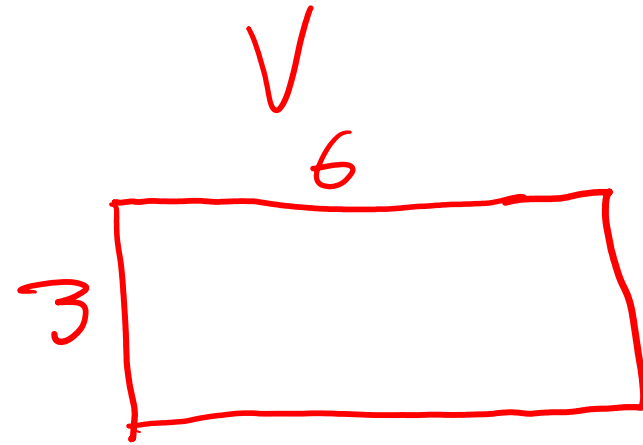


Let each row of matrix V be vector $\mathbf{v}_j^T$ representing student j in a group.

We can compare all pairs of vectors by doing matrix multiplication of V with itself.

$$f(V) = V V^T$$

3 x 3

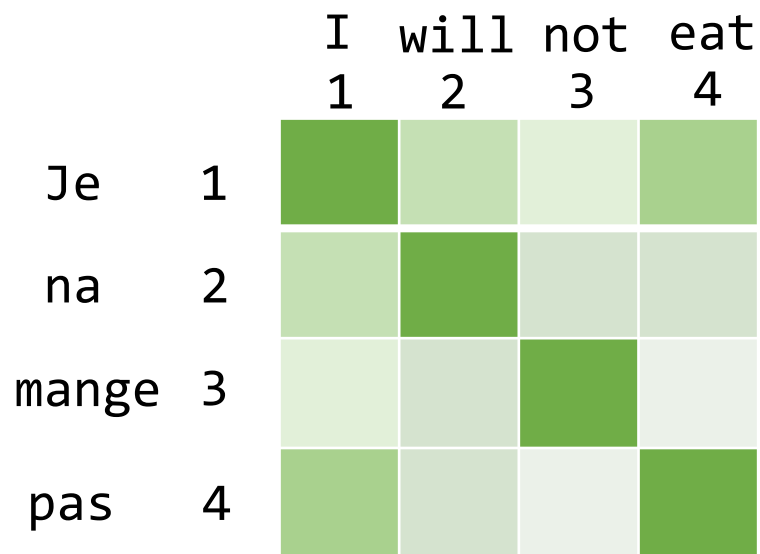


Learn to pay attention!

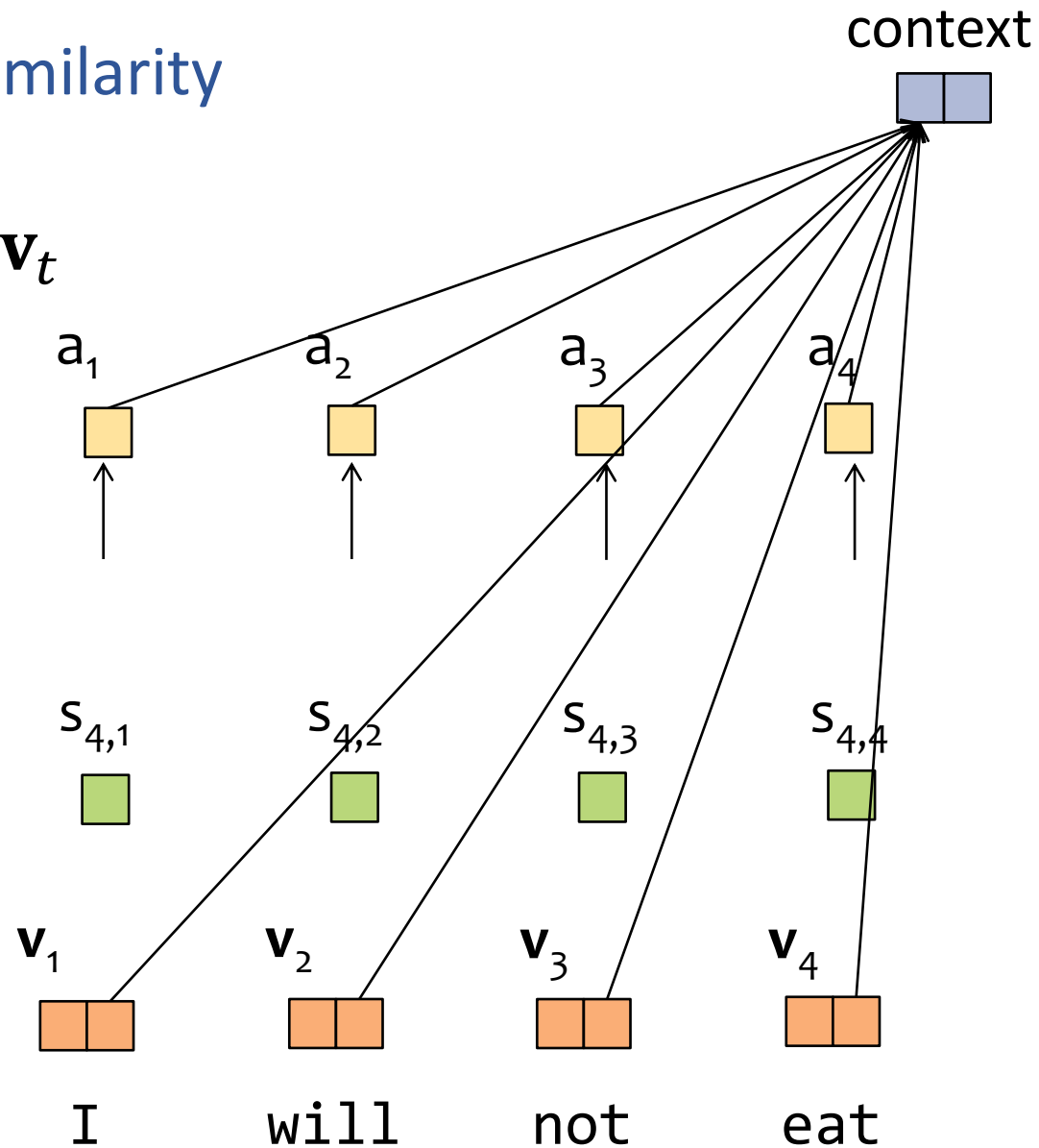
If only we had a way to measure vector similarity

Cosine similarity matrix!

$$S = VV^T$$



$$\mathbf{v}' = \sum_{t=1}^T a_t \mathbf{v}_t$$



Learn to pay attention!

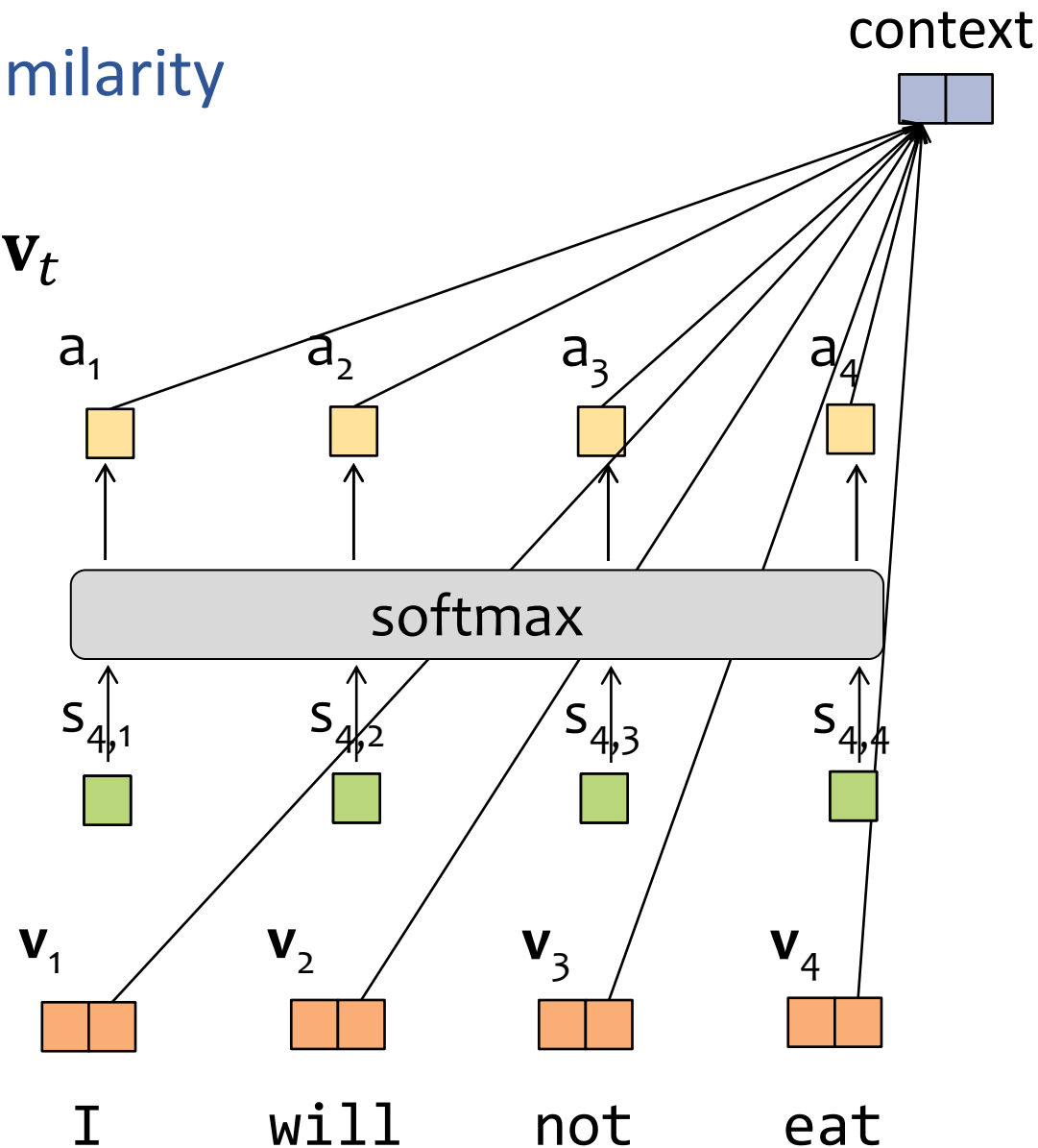
If only we had a way to measure vector similarity

Cosine similarity matrix!

$$S = VV^T$$



$$\mathbf{v}' = \sum_{t=1}^T a_t \mathbf{v}_t$$

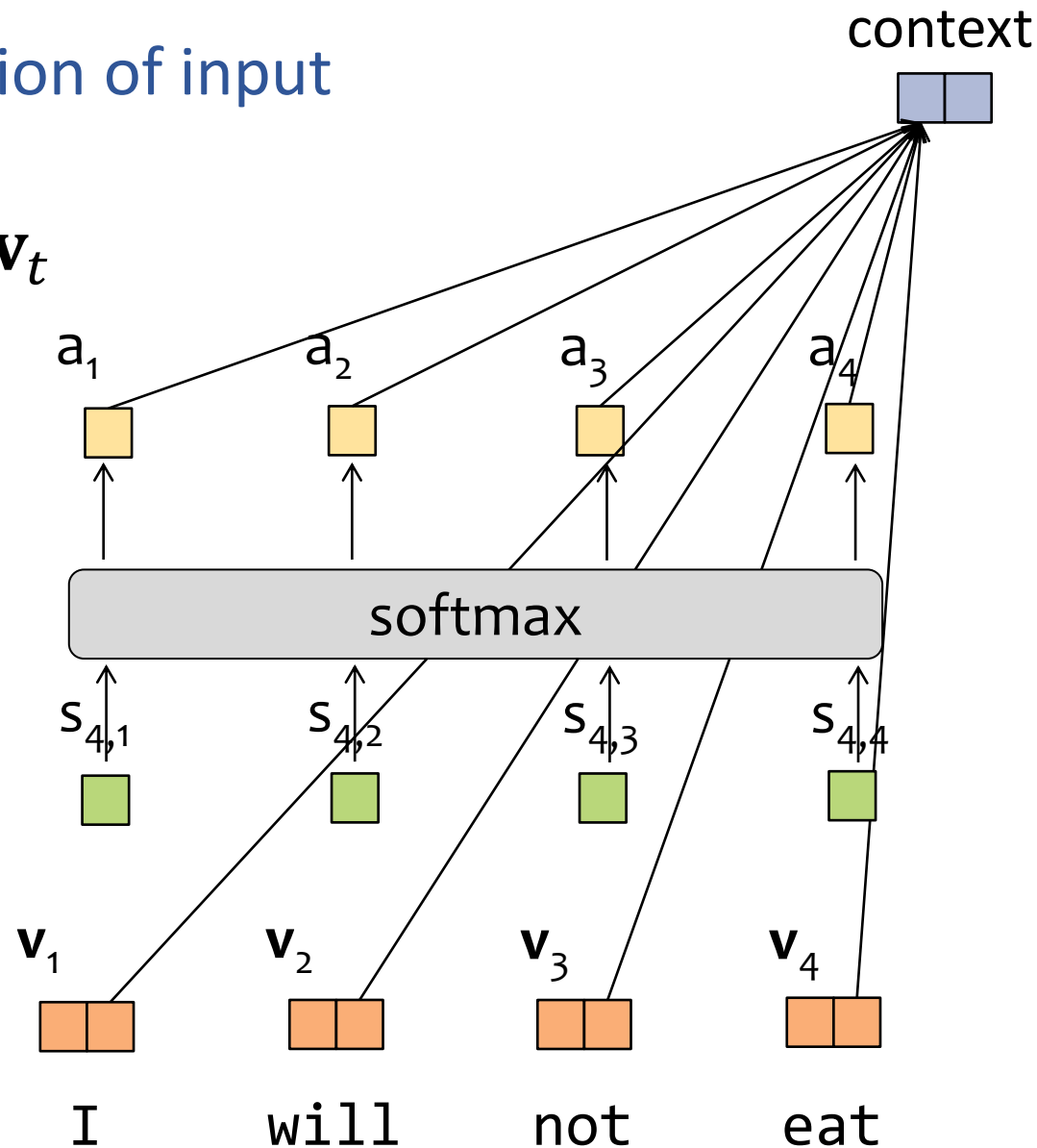
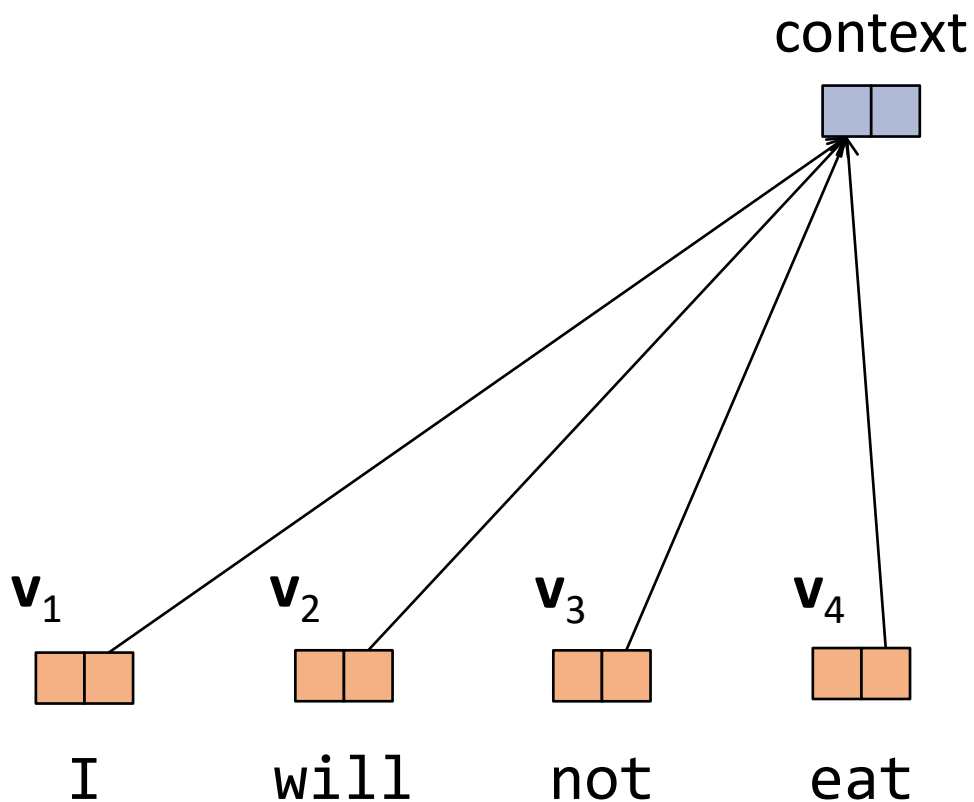


Learn to pay attention!

We can do better than uniform combination of input

$$\mathbf{v} = \sum_{t=1}^T \frac{1}{T} \mathbf{v}_t$$

$$\mathbf{v} = \sum_{t=1}^T a_t \mathbf{v}_t$$



Learn to pay attention!

But...there is an issue with just doing VV^T ☹️

We're really just comparing input to input

→ Symmetric with strong diagonal ☹️

$$S = VV^T$$



x_1

I

x_2

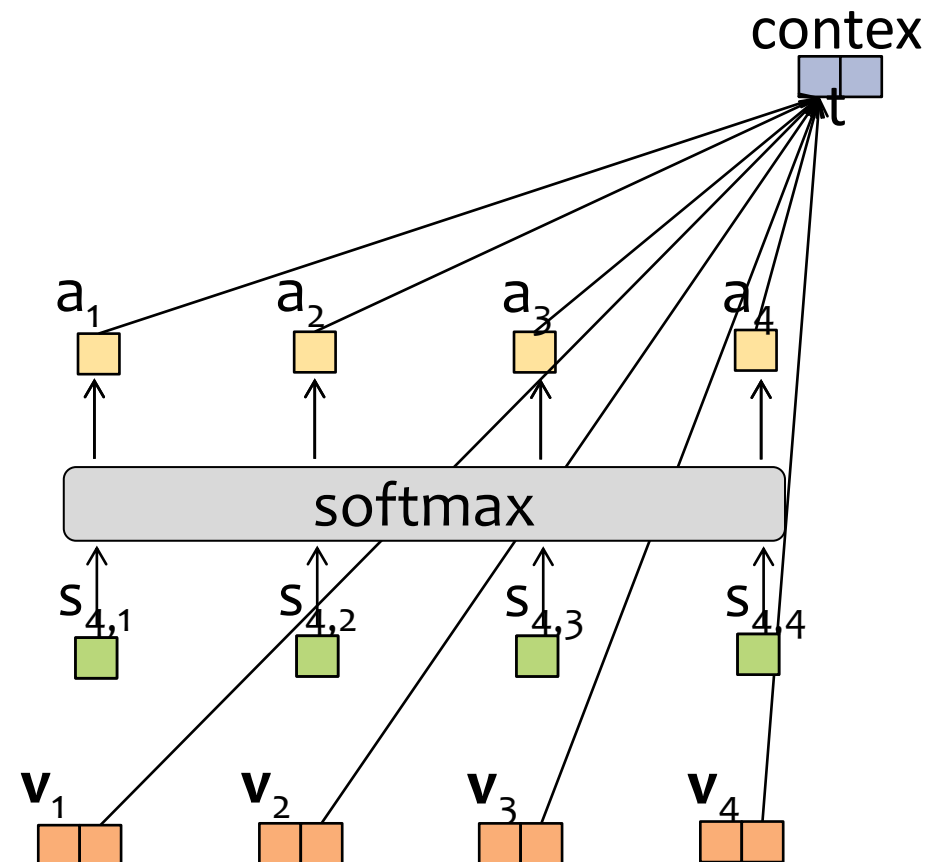
will

x_3

not

x_4

eat

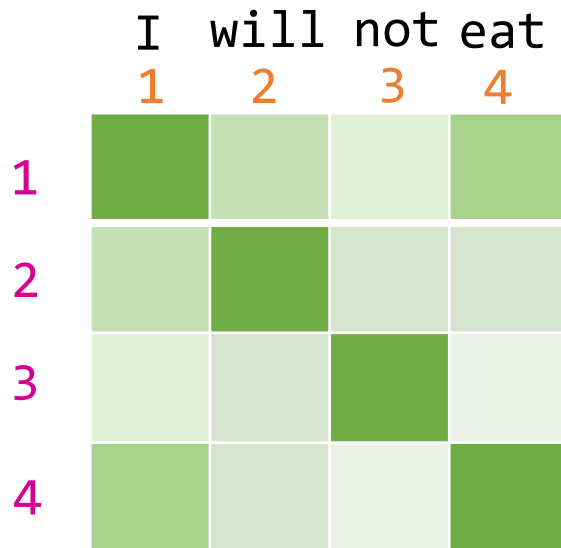


$$V = XW_V$$

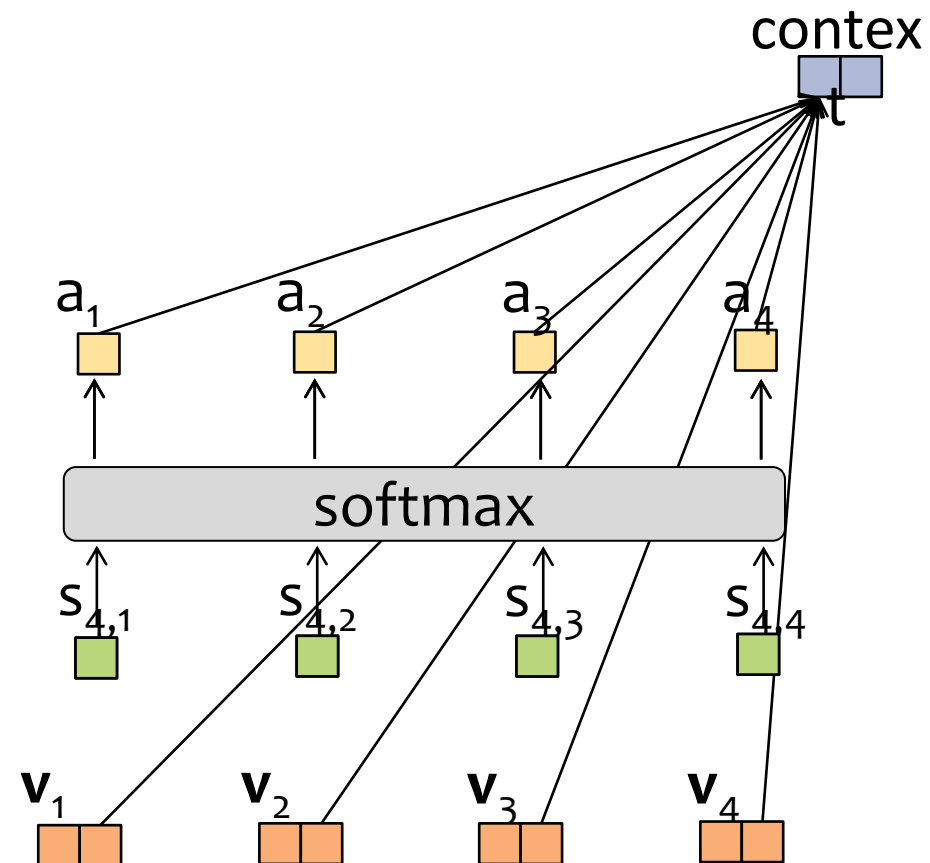
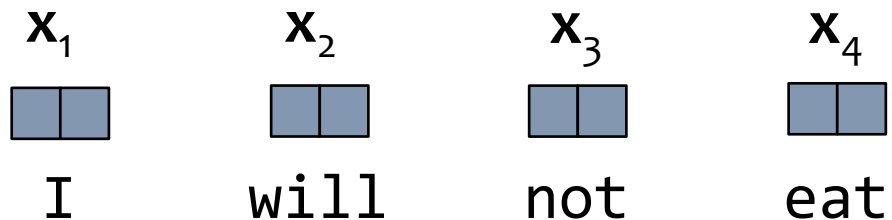
Learn to pay attention!

Instead learn a query vectors $\mathbf{q}_t$ to represent the output

$$S = QV^T$$



$$Q = XW_Q$$



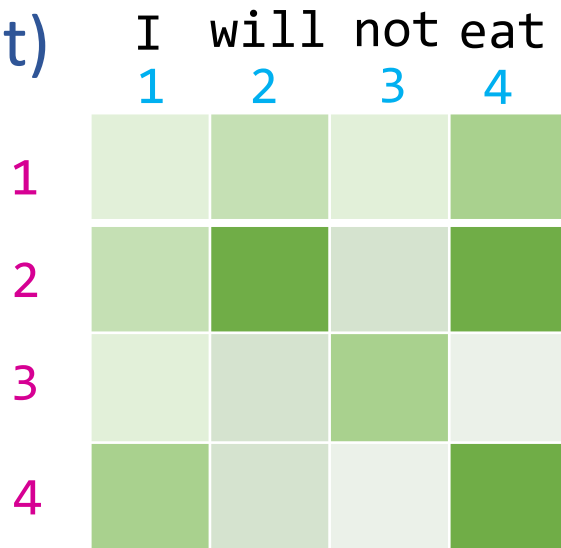
$$V = XW_V$$

Learn to pay attention!

Instead learn a query vectors $\mathbf{q}_t$ to represent the output

(And also $\mathbf{k}_t$ for the input)

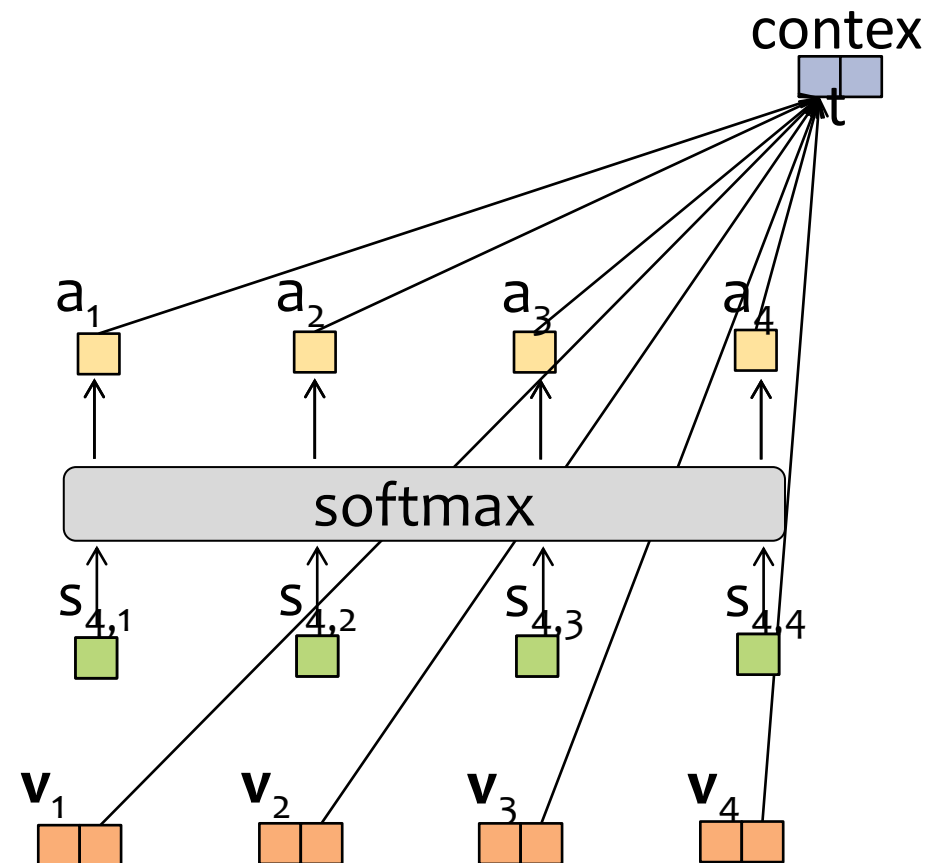
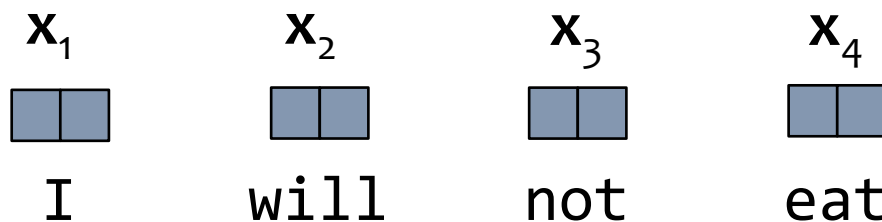
$$S = QK^T / \sqrt{d_k}$$



$$Q = XW_Q$$

$$K = XW_K$$

$$V = XW_V$$



Attention

Vector representation

$$\mathbf{o}_t = W_O^T \mathbf{z}_t$$

$$\mathbf{z}_t = V^T \mathbf{a}_t$$

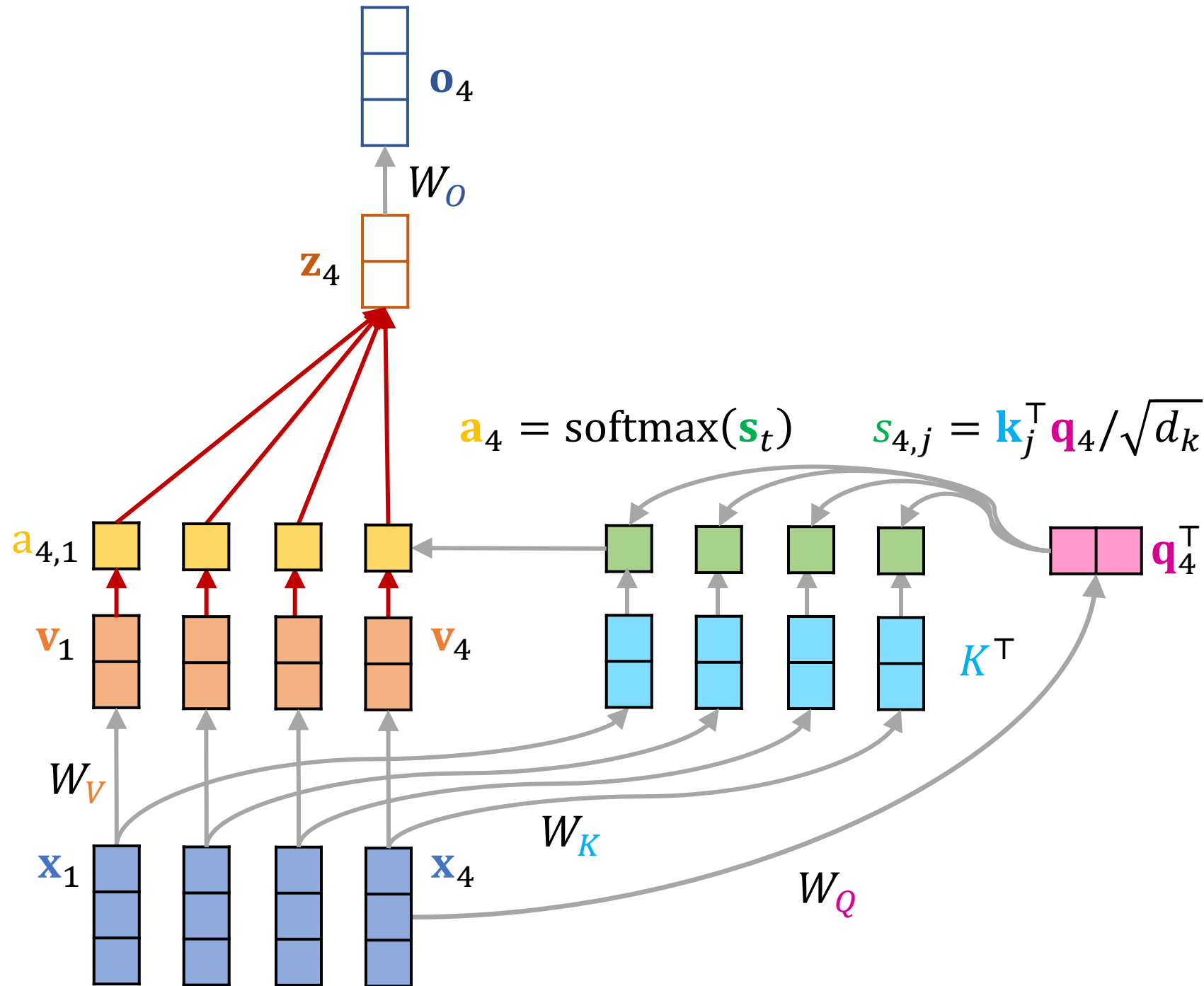
$$\mathbf{a}_t = \text{softmax}(\mathbf{s}_t)$$

$$\mathbf{s}_t = K \mathbf{q}_t / \sqrt{d_k}$$

$$\mathbf{q}_t = W_Q^T \mathbf{x}_t$$

$$\mathbf{k}_t = W_K^T \mathbf{x}_t$$

$$\mathbf{v}_t = W_V^T \mathbf{x}_t$$



(Full) Attention

Matrix representation

$$X' = X \oplus O$$

$$O = ZW_O$$

$$Z = AV$$

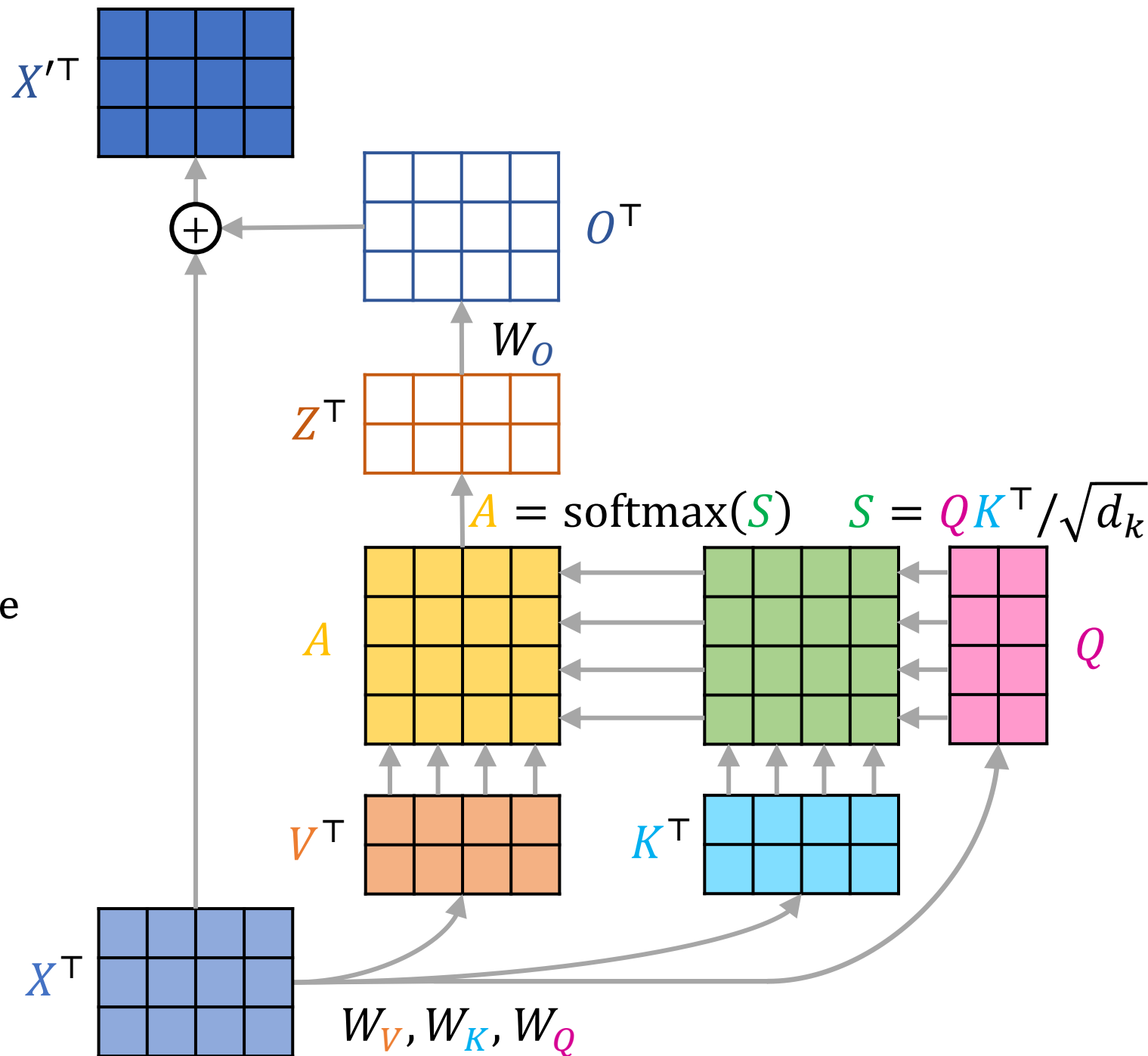
$$A = \text{softmax}(S)_{\text{row-wise}}$$

$$S = QK^T / \sqrt{d_k}$$

$$Q = XW_Q$$

$$K = XW_K$$

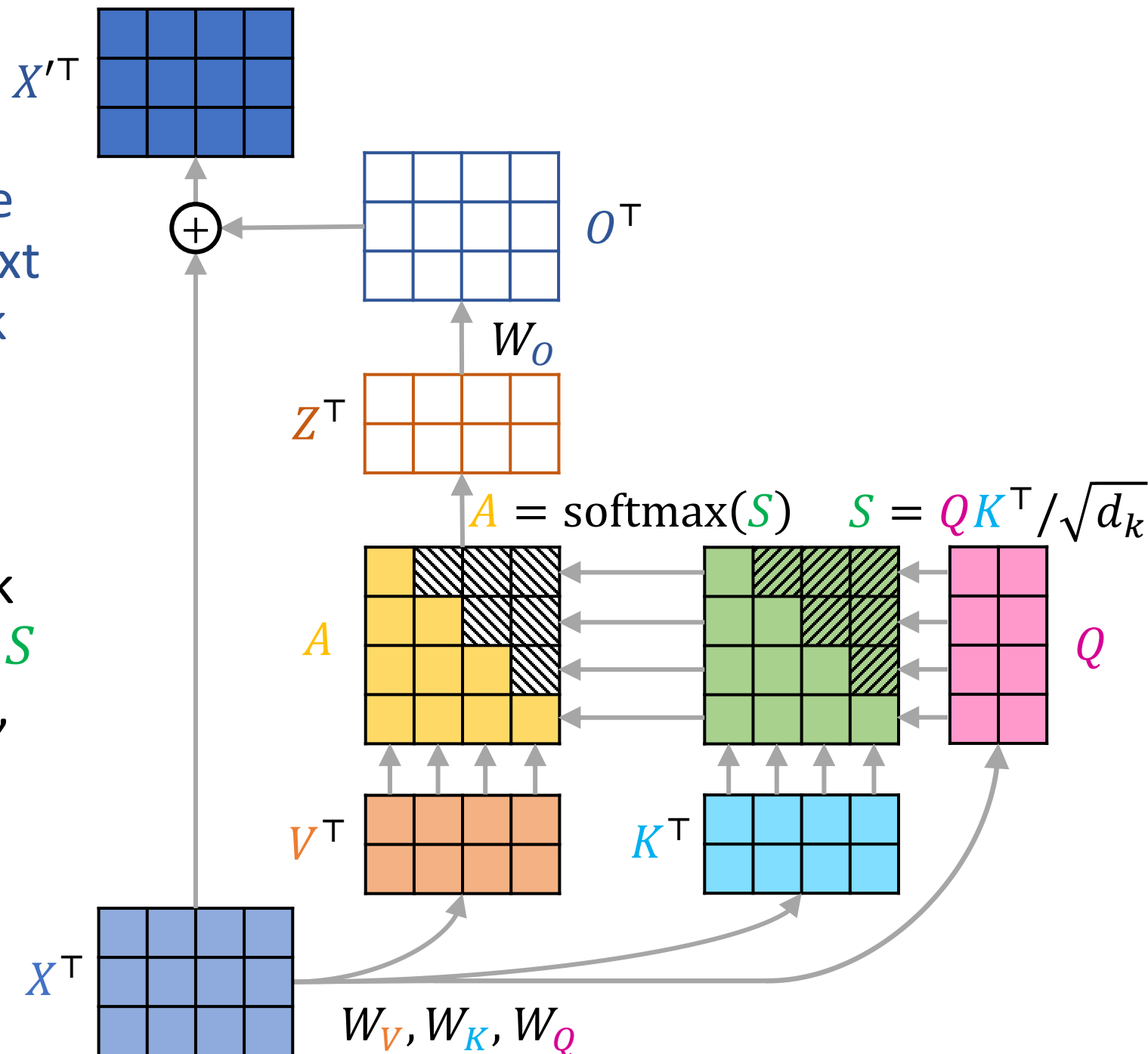
$$V = XW_V$$



Causal Attention

When learning to generate the next token from context 1: t , we don't want to look ahead and use any information from $t + 1$ or greater

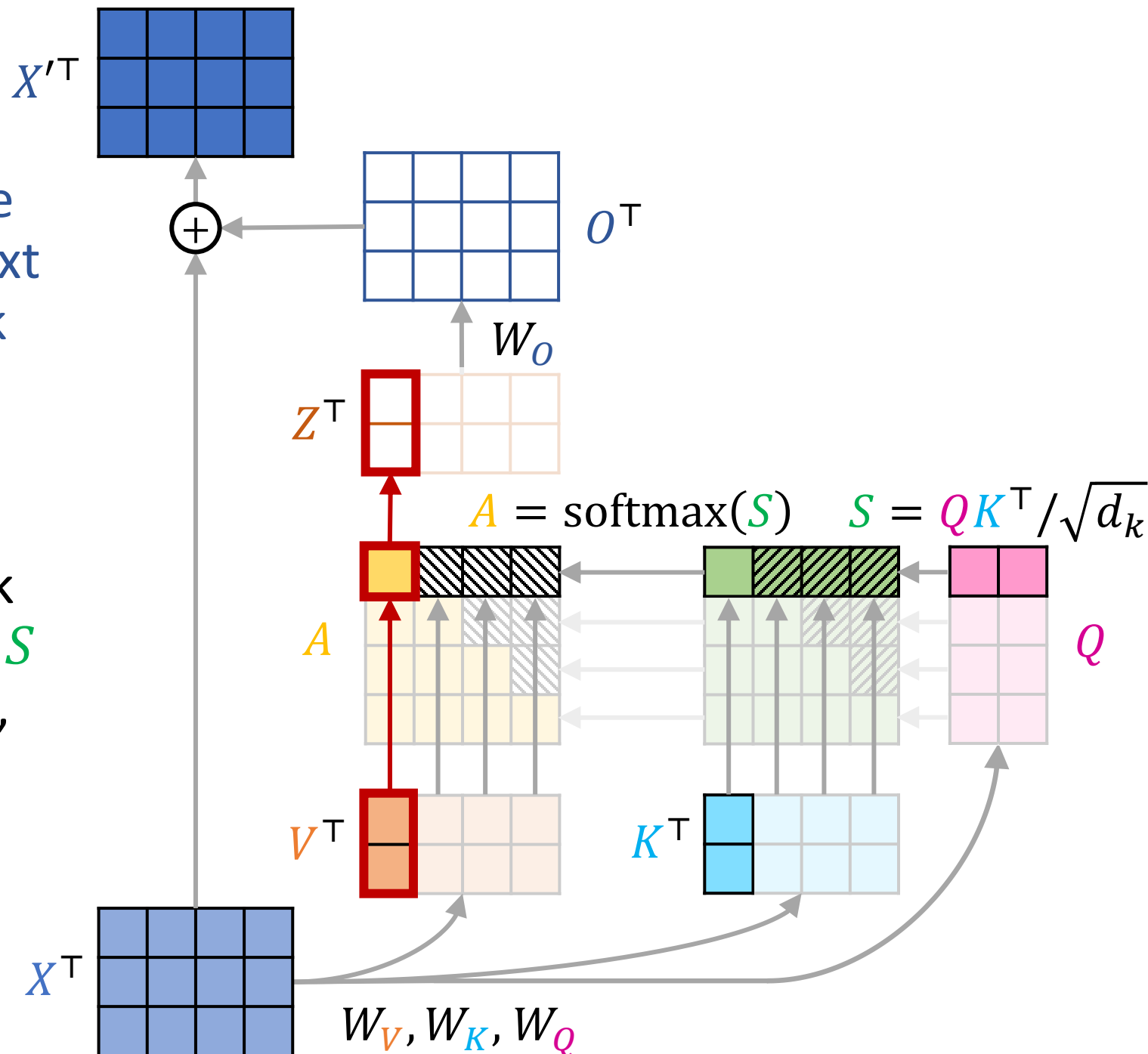
- We apply a causal mask to the attention scores S (filled with $-\infty$ values), which zeros out the appropriate attention weights in A



Causal Attention

When learning to generate the next token from context 1: t , we don't want to look ahead and use any information from $t + 1$ or greater

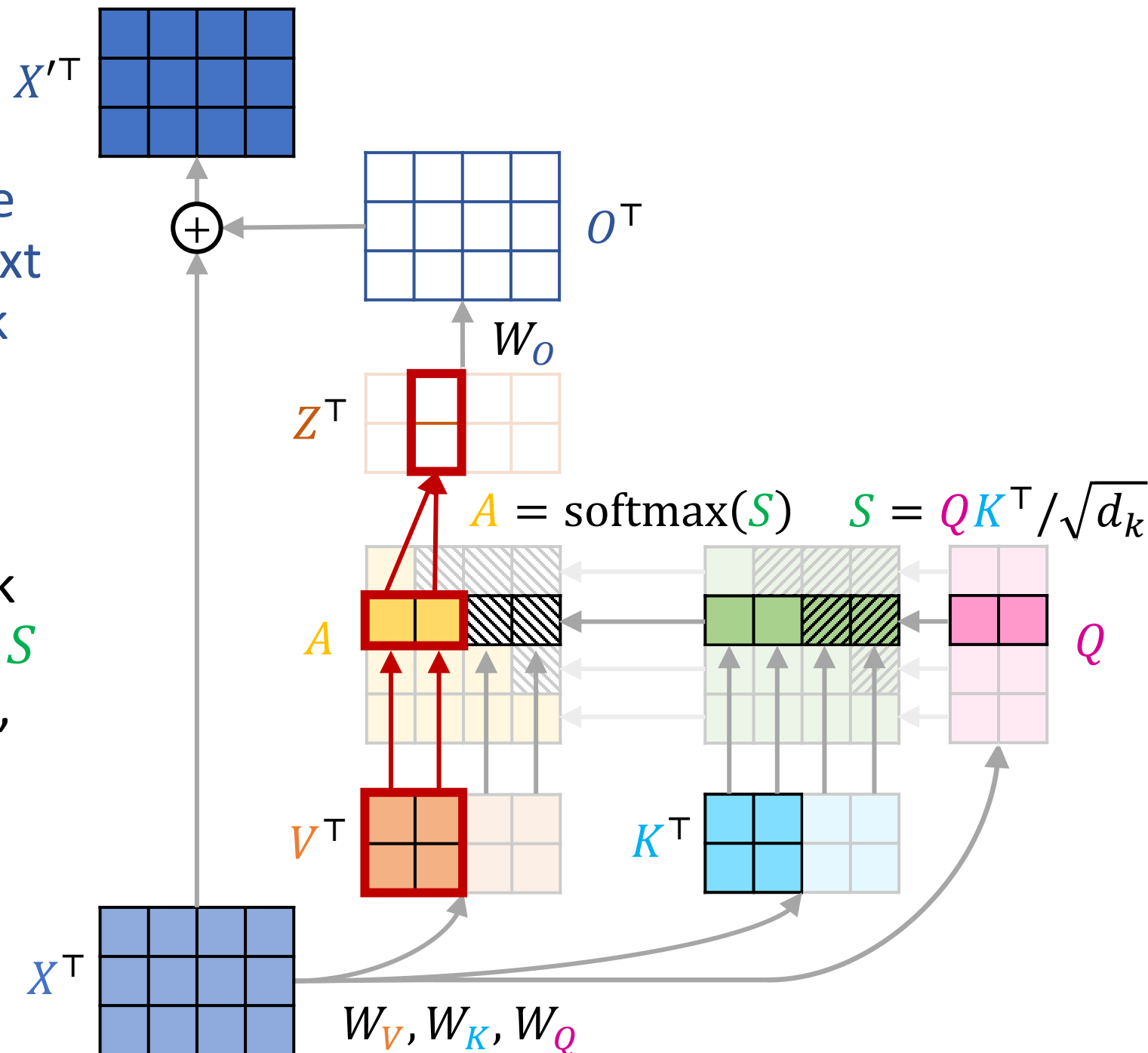
- We apply a causal mask to the attention scores S (filled with $-\infty$ values), which zeros out the appropriate attention weights in A



Causal Attention

When learning to generate the next token from context 1: t , we don't want to look ahead and use any information from $t + 1$ or greater

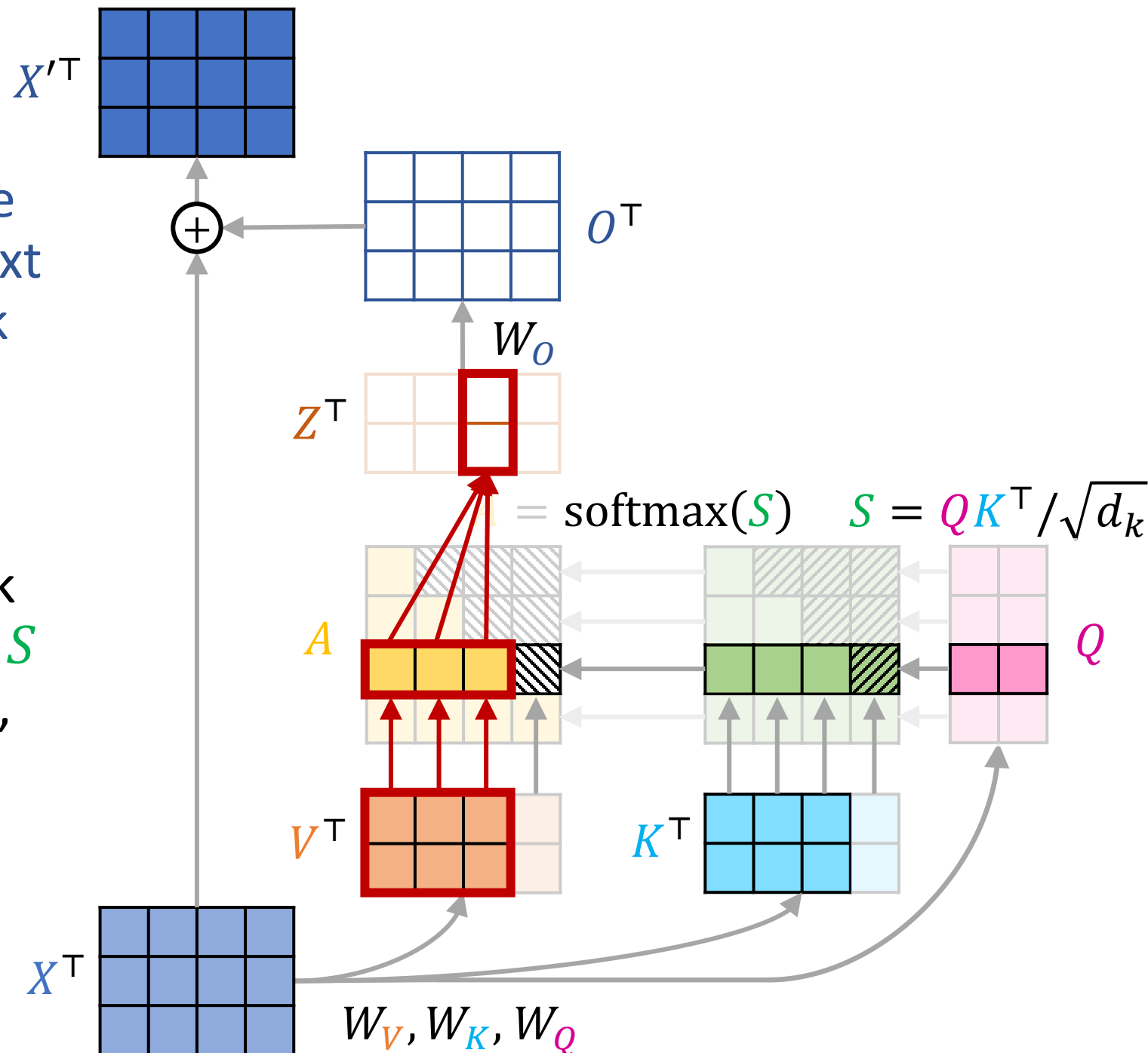
- We apply a causal mask to the attention scores S (filled with $-\infty$ values), which zeros out the appropriate attention weights in A



Causal Attention

When learning to generate the next token from context 1: t , we don't want to look ahead and use any information from $t + 1$ or greater

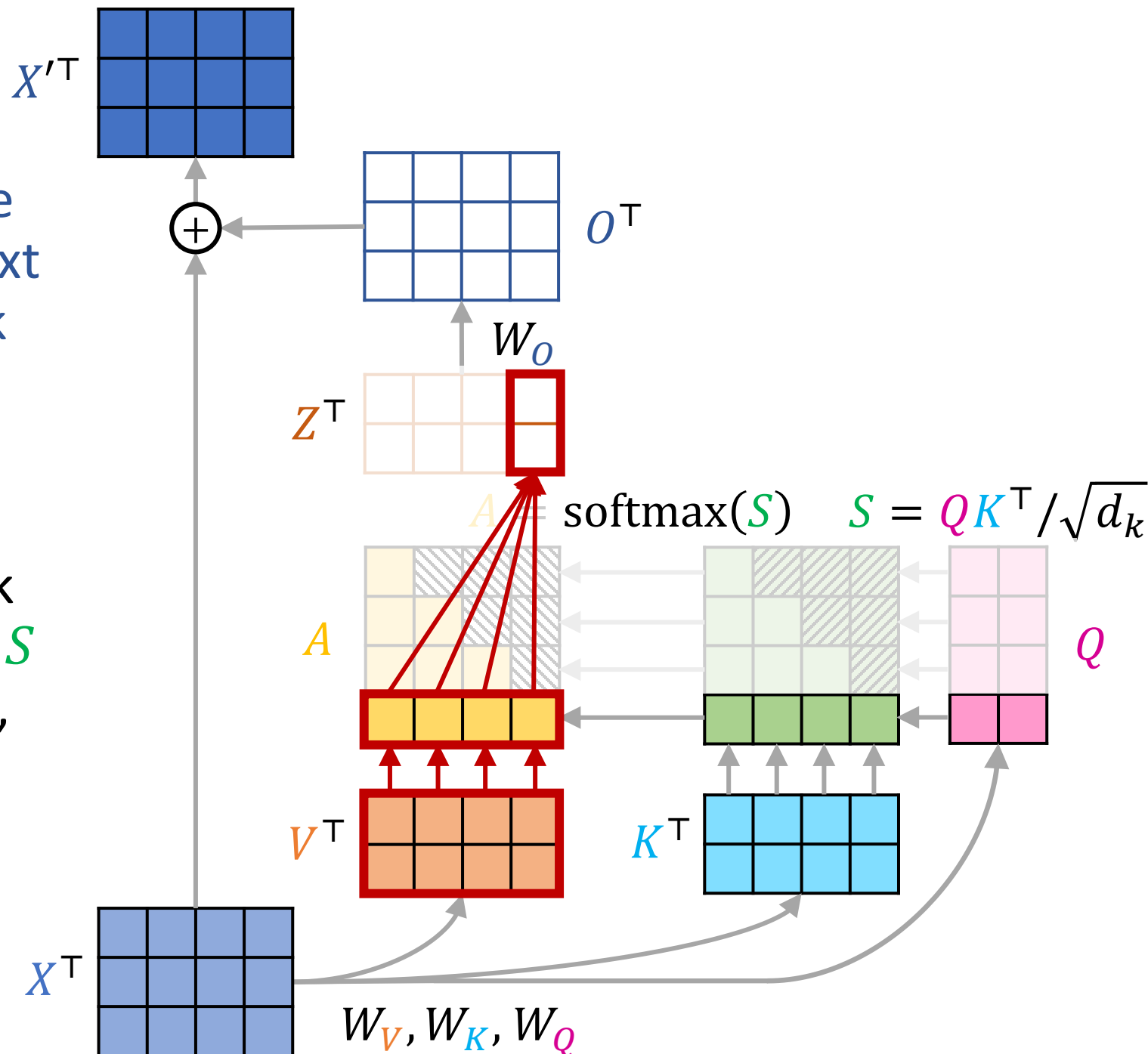
- We apply a causal mask to the attention scores S (filled with $-\infty$ values), which zeros out the appropriate attention weights in A



Causal Attention

When learning to generate the next token from context 1: t , we don't want to look ahead and use any information from $t + 1$ or greater

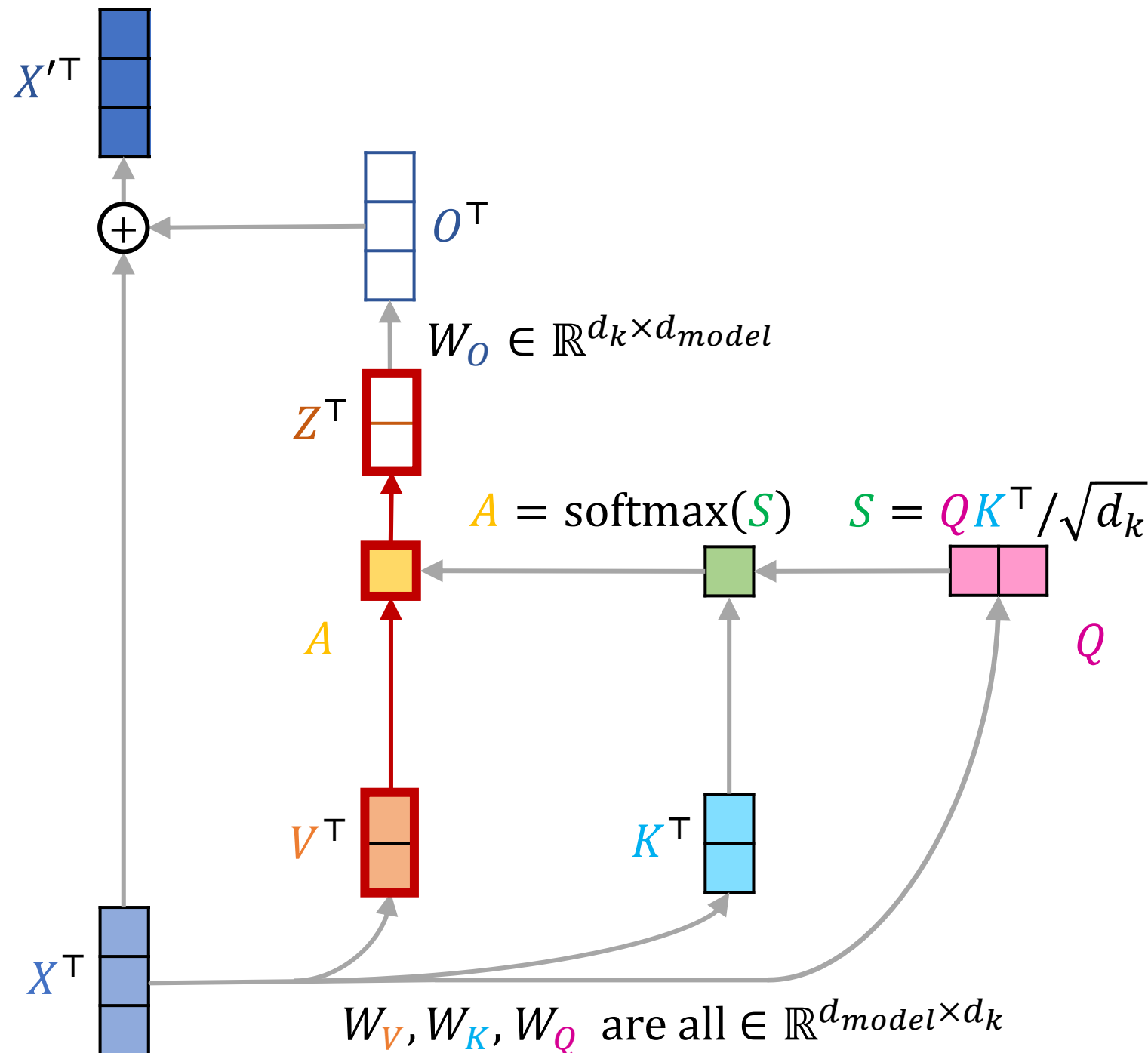
- We apply a causal mask to the attention scores S (filled with $-\infty$ values), which zeros out the appropriate attention weights in A



Causal Attention

Inference time

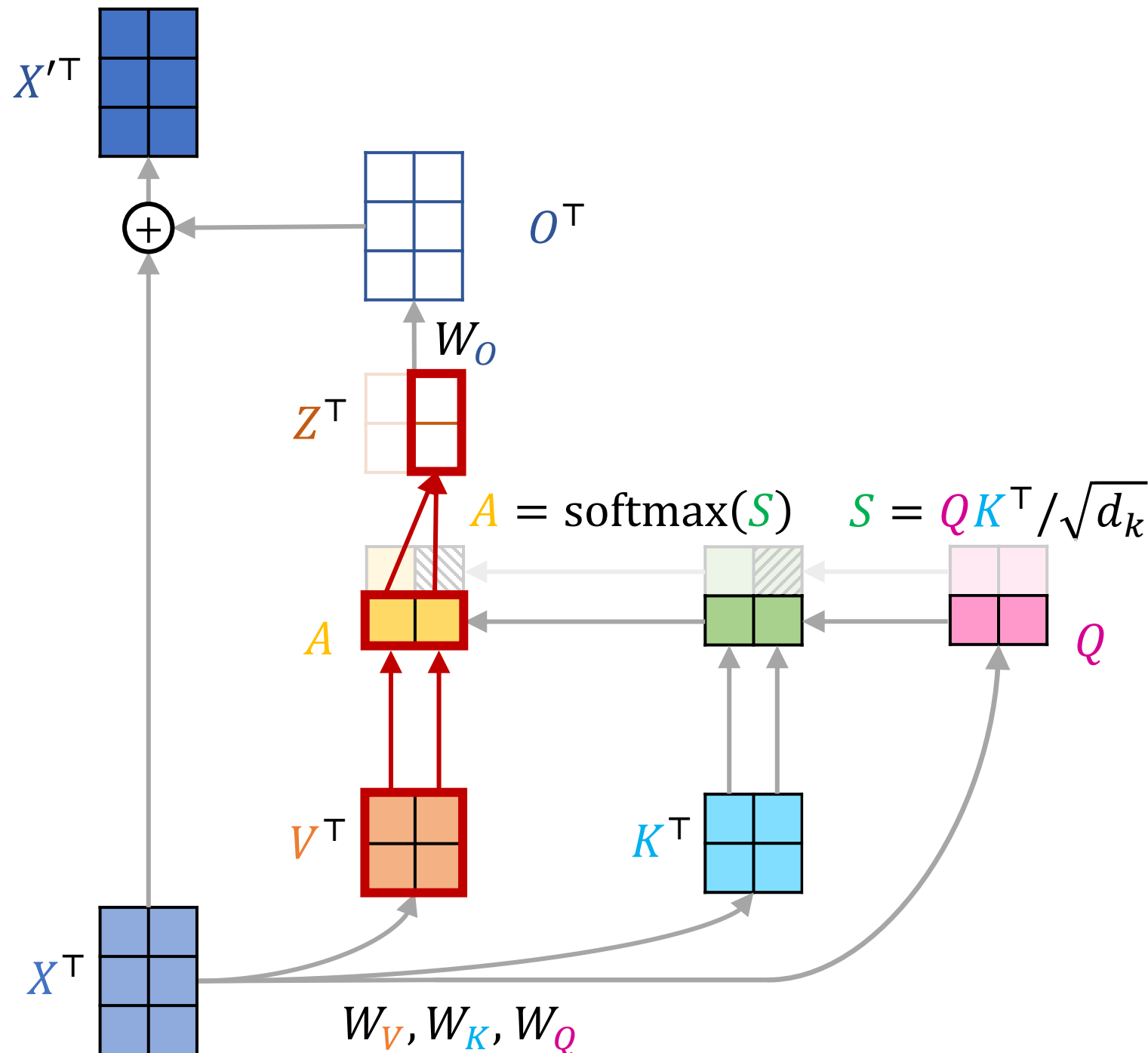
- Done training. Watch this attention block as the context size increases as we generate more and more tokens
- Note how different components build up as the context grows
- But the size of the parameters don't change!



Causal Attention

Inference time

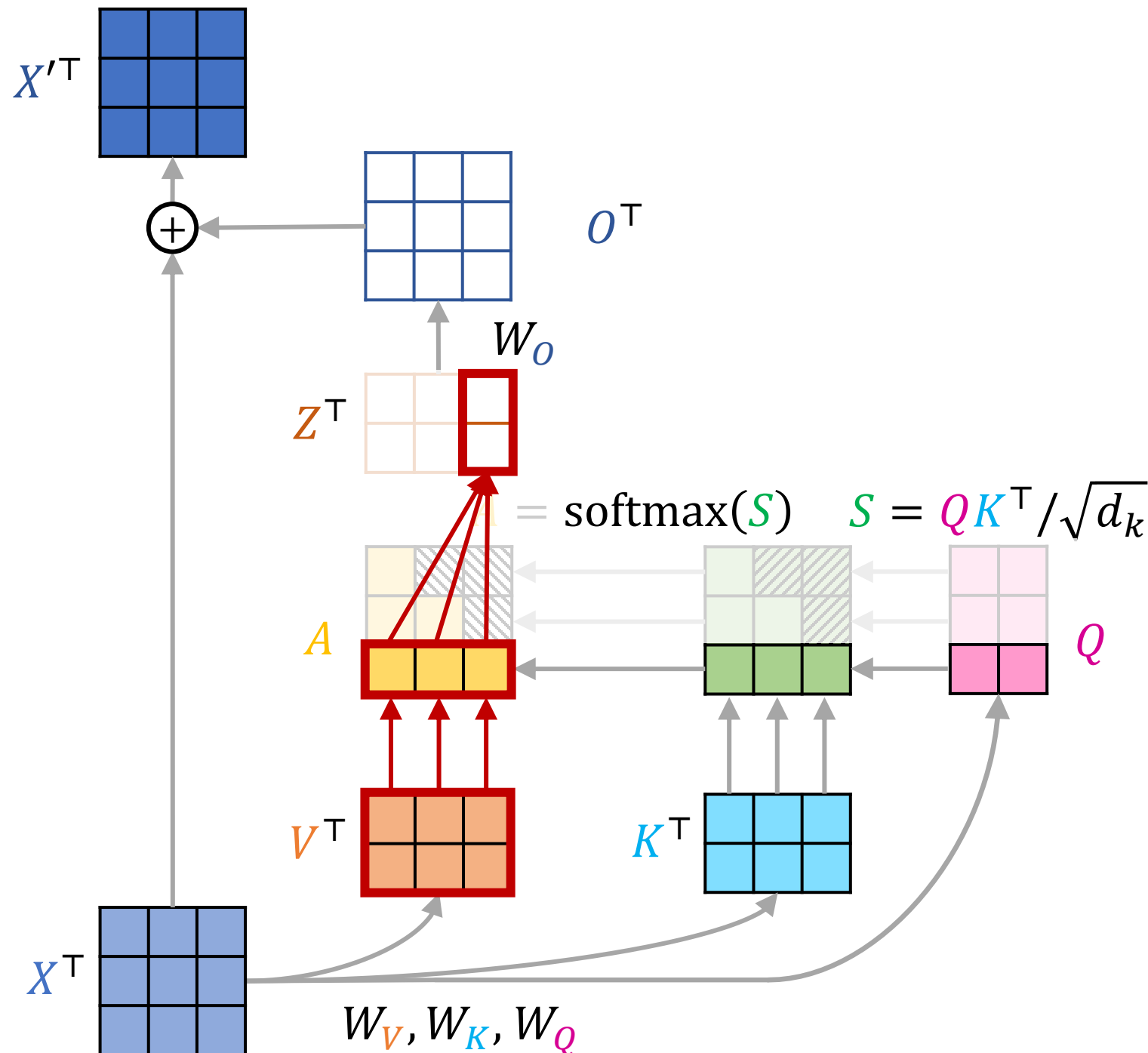
- Done training. Watch this attention block as the context size increases as we generate more and more tokens
- Note how different components build up as the context grows
- But the size of the parameters don't change!



Causal Attention

Inference time

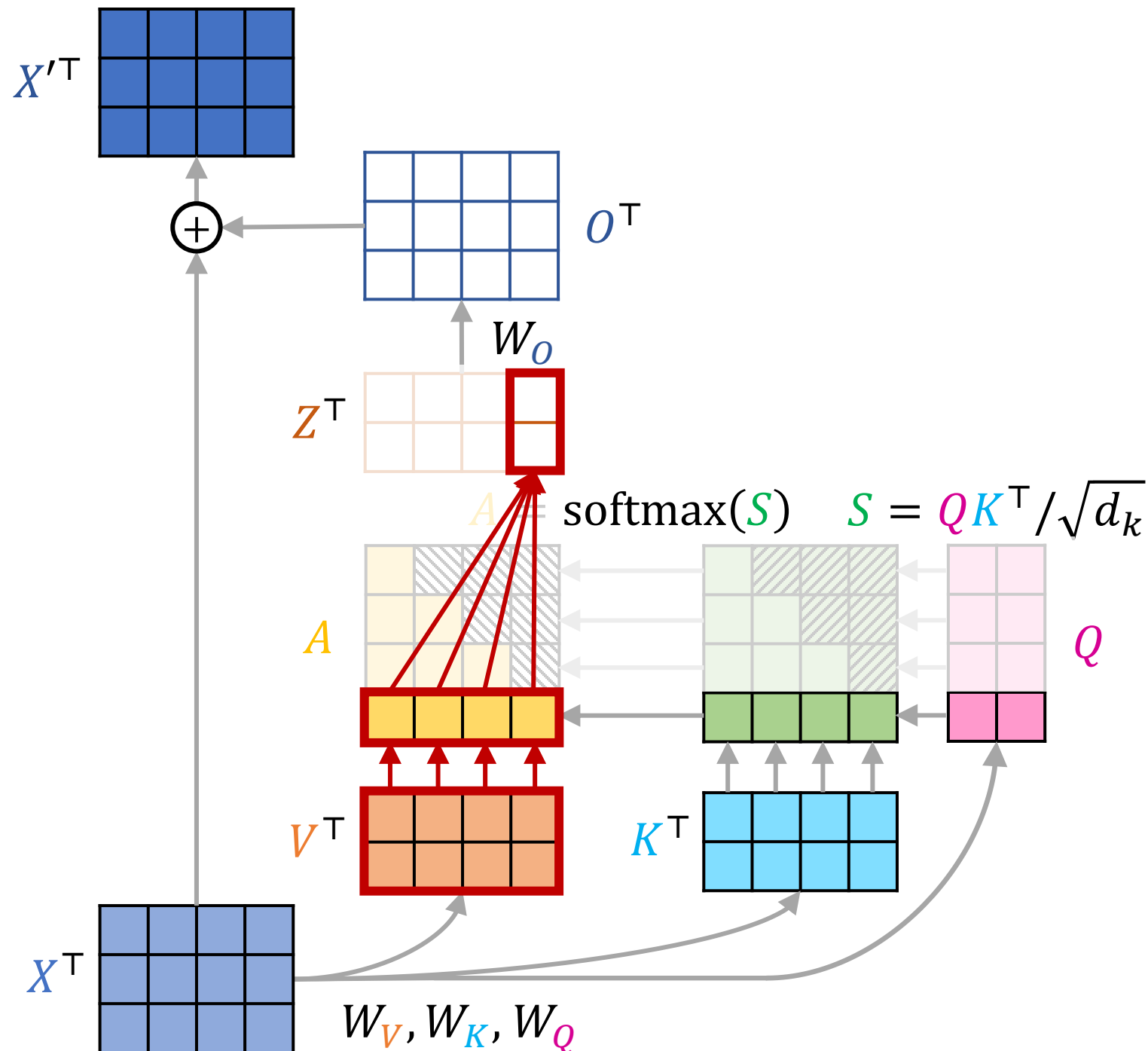
- Done training. Watch this attention block as the context size increases as we generate more and more tokens
- Note how different components build up as the context grows
- But the size of the parameters don't change!



Causal Attention

Inference time

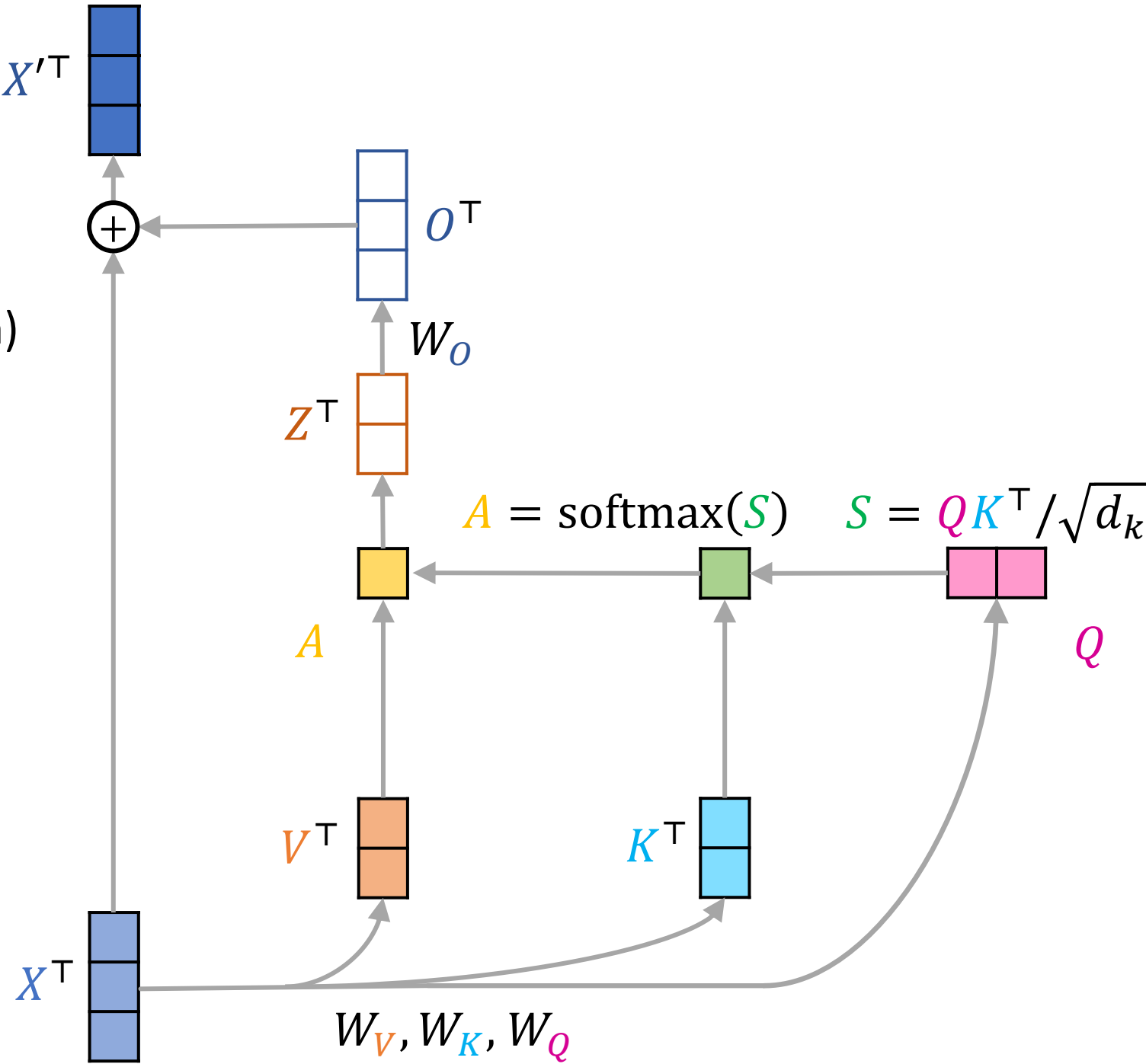
- Done training. Watch this attention block as the context size increases as we generate more and more tokens
- Note how different components build up as the context grows
- But the size of the parameters don't change!



Causal Attention

Inference time

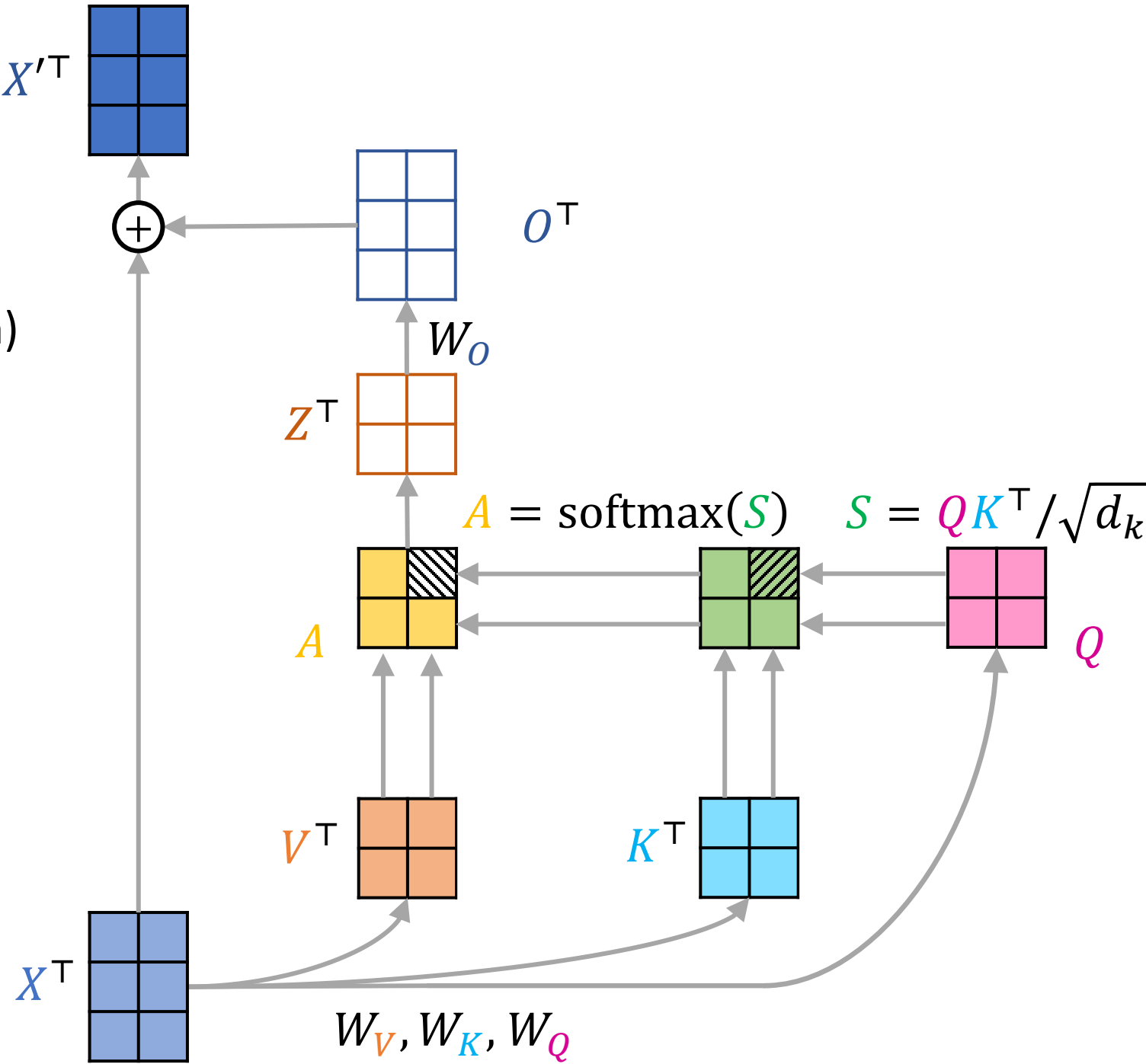
(Repeated without red arrows showing attention combination)



Causal Attention

Inference time

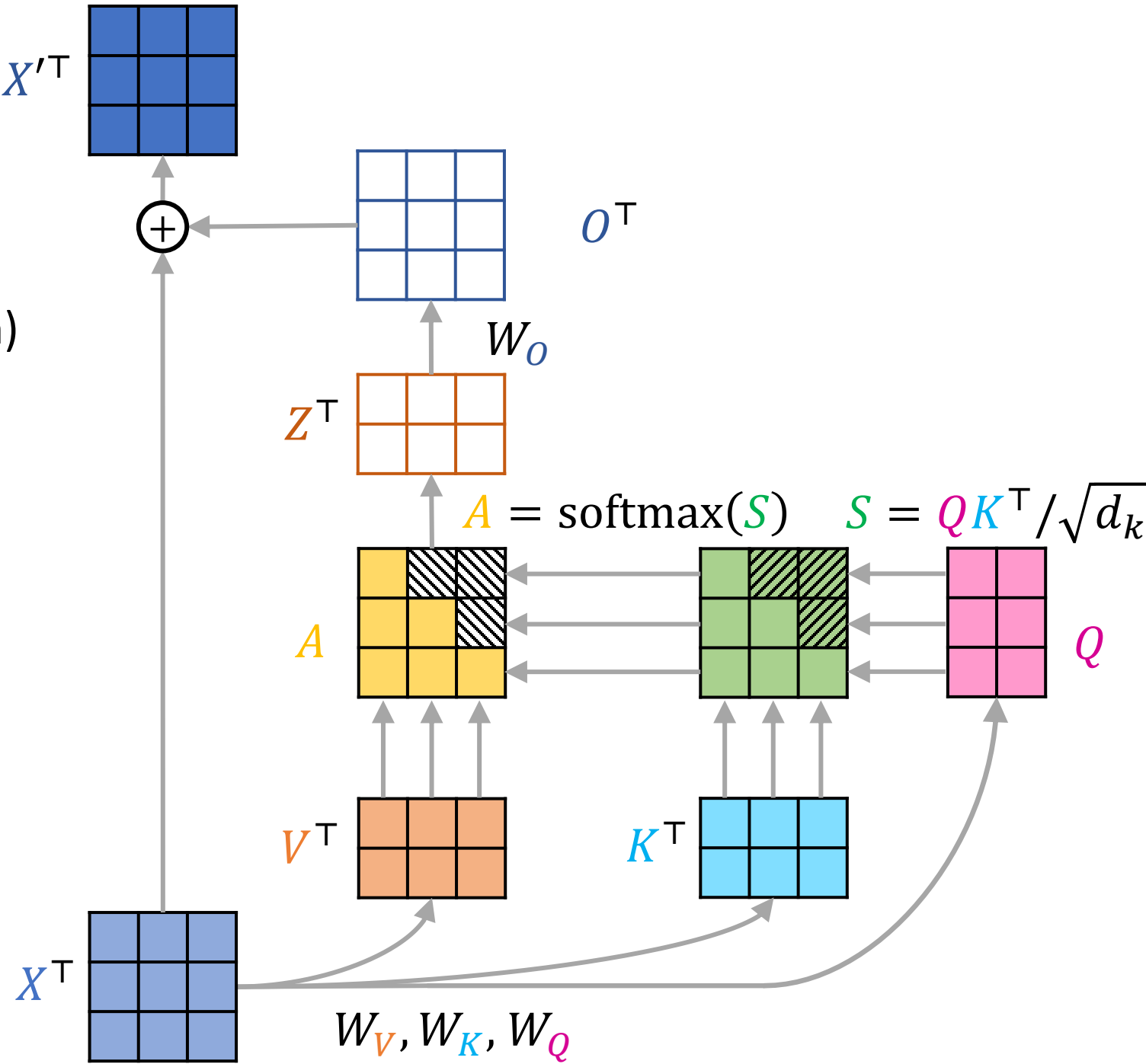
(Repeated without red arrows showing attention combination)



Causal Attention

Inference time

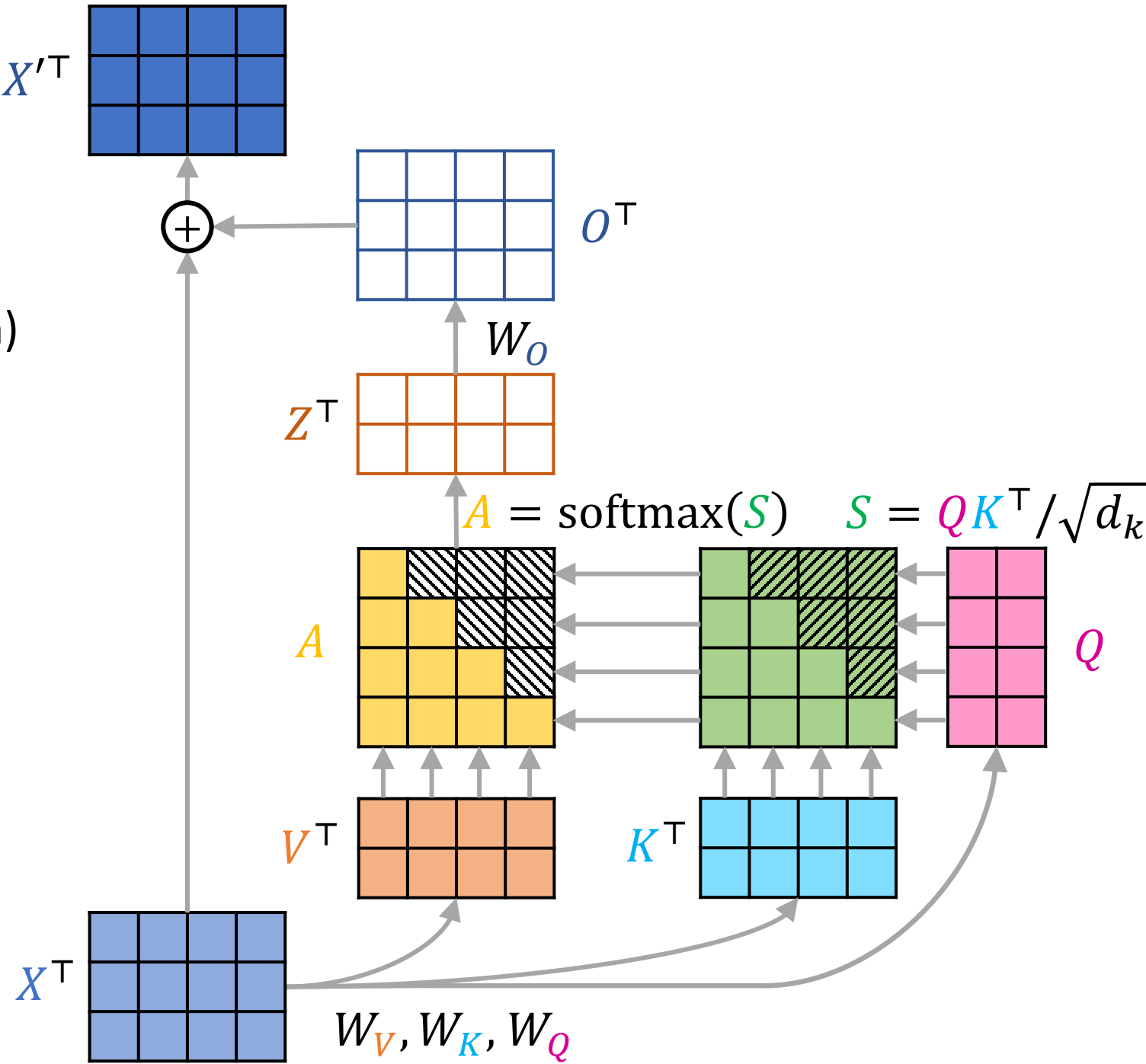
(Repeated without red arrows showing attention combination)



Causal Attention

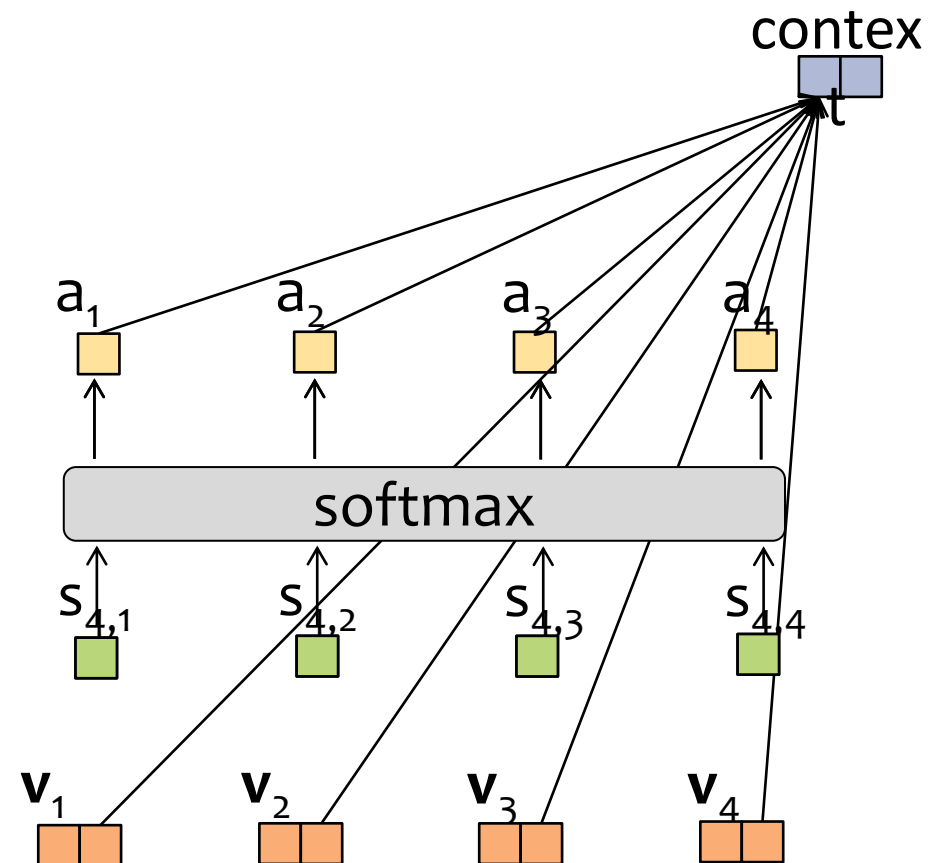
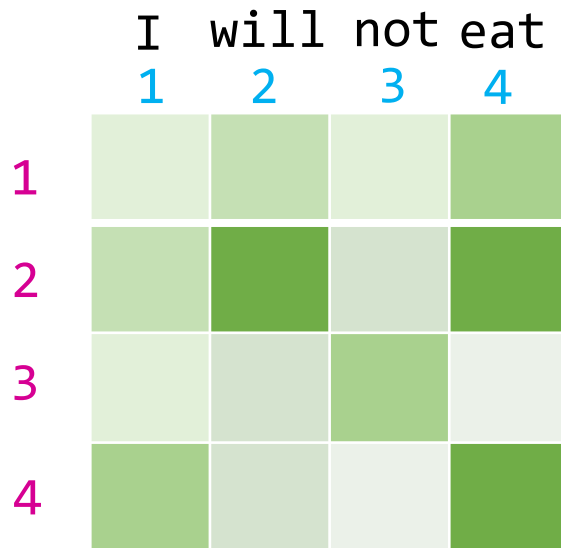
Inference time

(Repeated without red arrows showing attention combination)



Full Attention

What if we want to translate an input context to another language?



$$S = QK^T / \sqrt{d_k}$$

$$Q = XW_Q$$

$$K = XW_K$$

$$V = XW_V$$

x_1
I

x_2
will

x_3
not

x_4
eat

(Full) Attention

Matrix representation

$$X' = X \oplus O$$

$$O = ZW_O$$

$$Z = AV$$

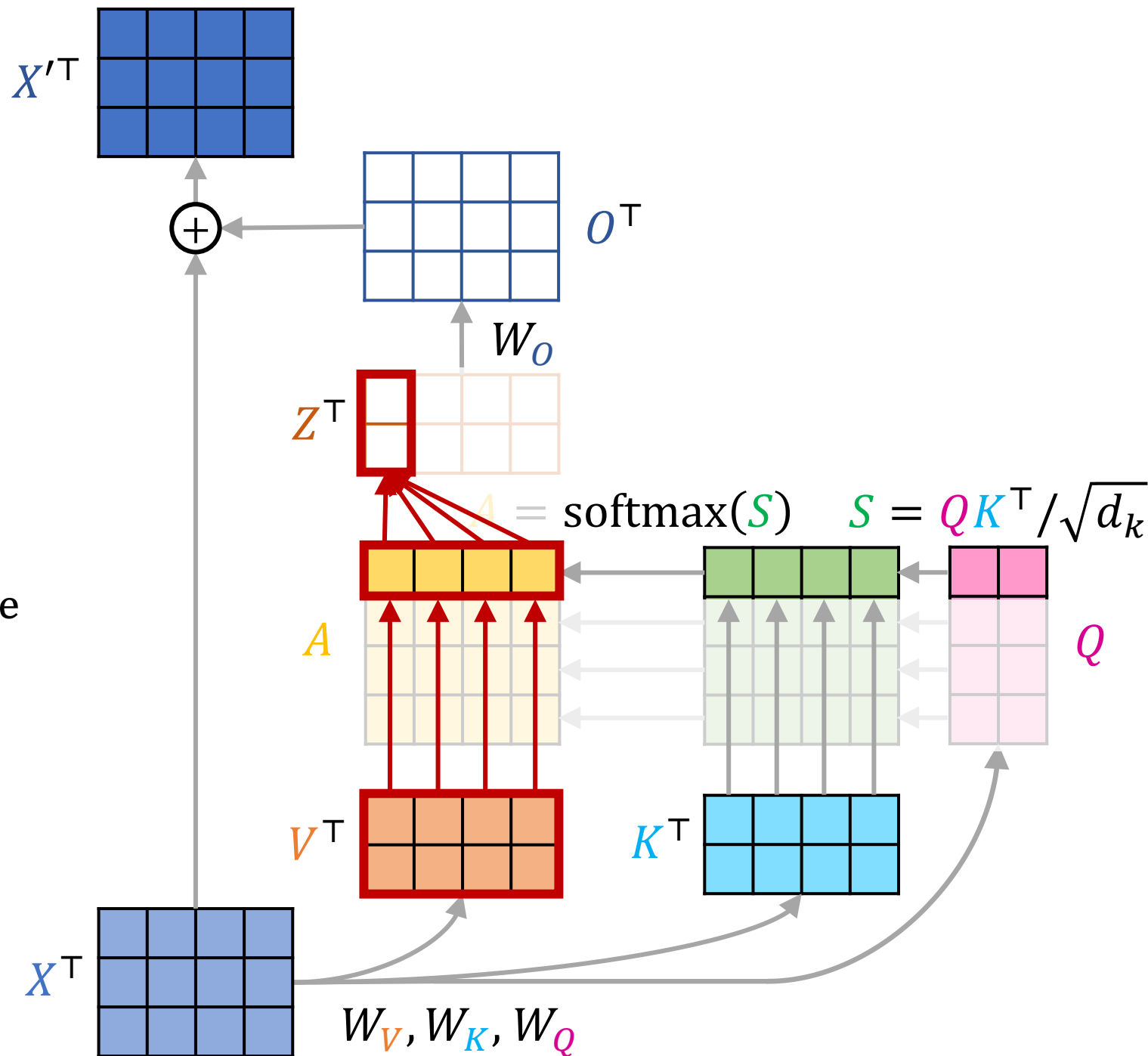
$$A = \text{softmax}(S)_{\text{row-wise}}$$

$$S = QK^T / \sqrt{d_k}$$

$$Q = XW_Q$$

$$K = XW_K$$

$$V = XW_V$$



(Full) Attention

Matrix representation

$$X' = X \oplus O$$

$$O = ZW_O$$

$$Z = AV$$

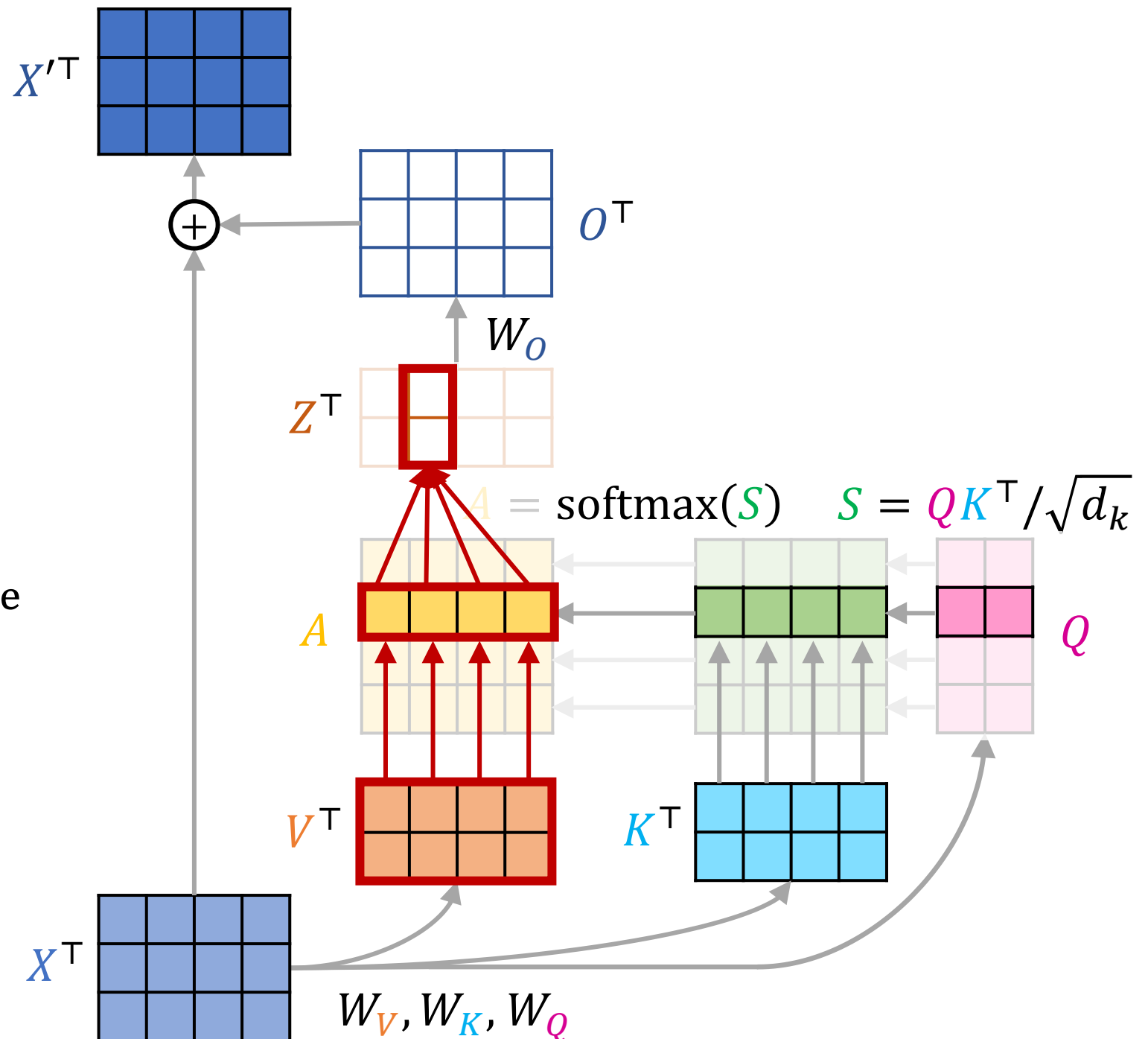
$$A = \text{softmax}(S)_{\text{row-wise}}$$

$$S = QK^T / \sqrt{d_k}$$

$$Q = XW_Q$$

$$K = XW_K$$

$$V = XW_V$$



(Full) Attention

Matrix representation

$$X' = X \oplus O$$

$$O = ZW_O$$

$$Z = AV$$

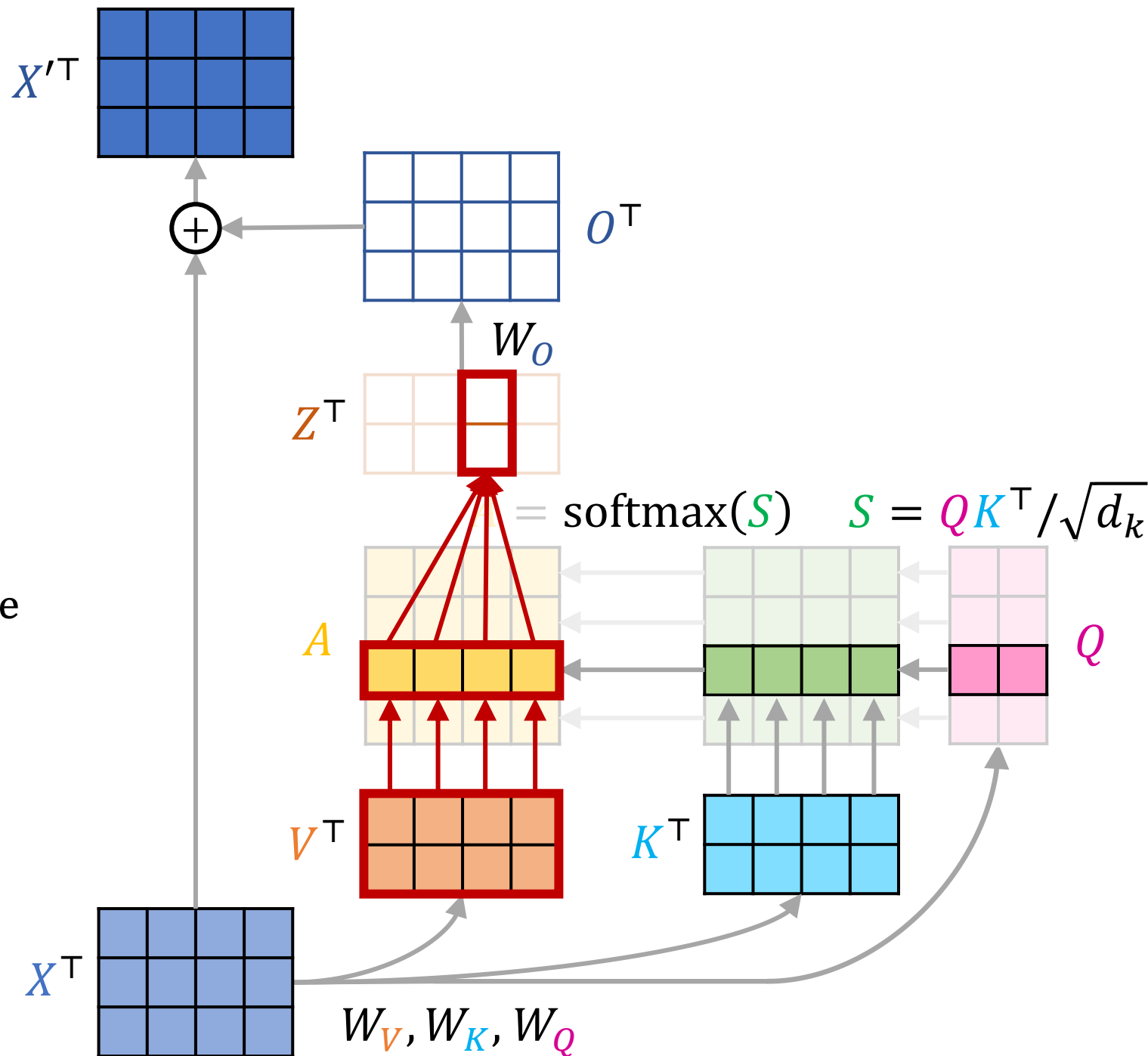
$$A = \text{softmax}(S)_{\text{row-wise}}$$

$$S = QK^T / \sqrt{d_k}$$

$$Q = XW_Q$$

$$K = XW_K$$

$$V = XW_V$$



(Full) Attention

Matrix representation

$$X' = X \oplus O$$

$$O = ZW_O$$

$$Z = AV$$

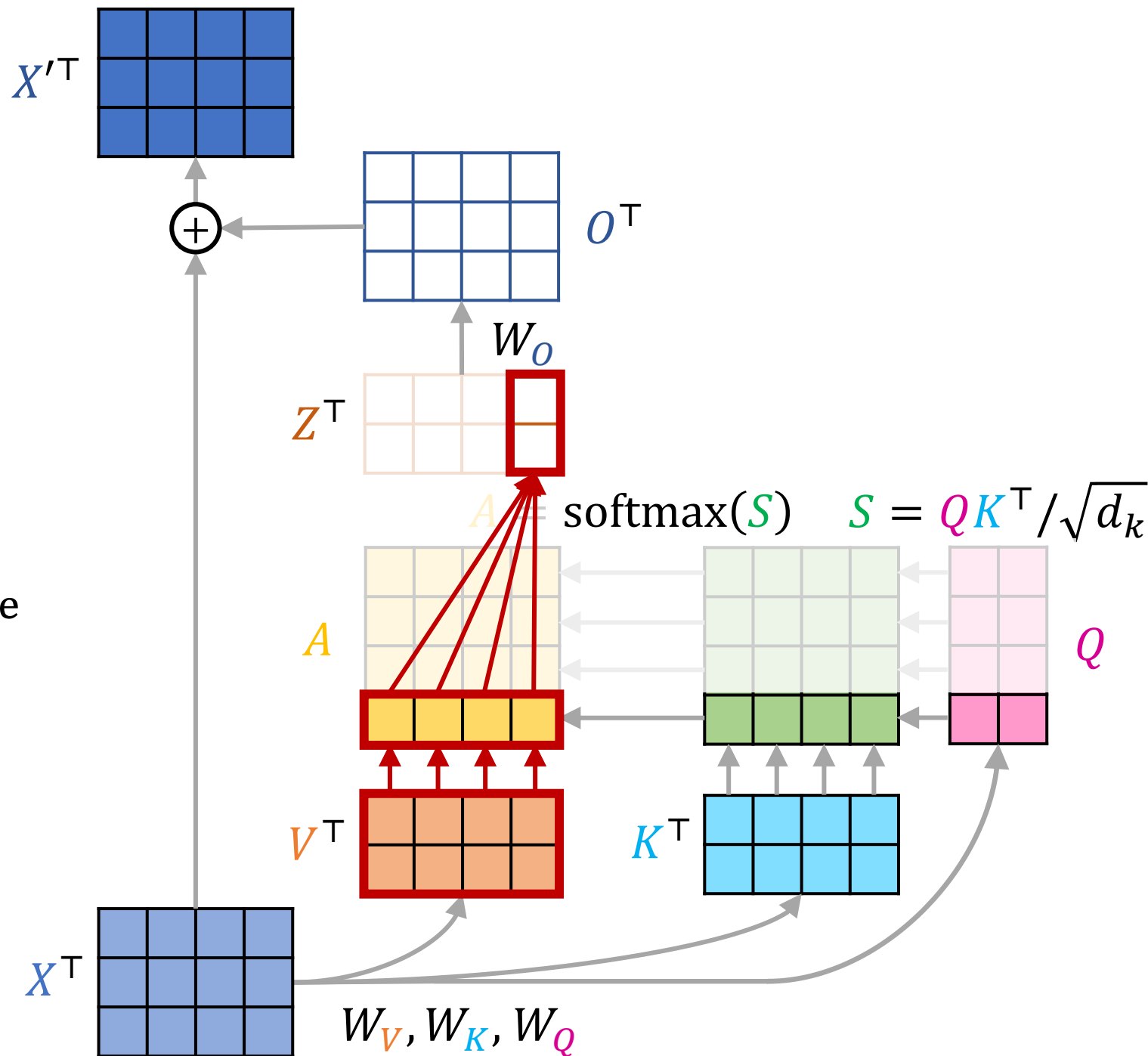
$$A = \text{softmax}(S)_{\text{row-wise}}$$

$$S = QK^T / \sqrt{d_k}$$

$$Q = XW_Q$$

$$K = XW_K$$

$$V = XW_V$$



(Full) Attention

Matrix representation

$$X' = X \oplus O$$

$$O = ZW_O$$

$$Z = AV$$

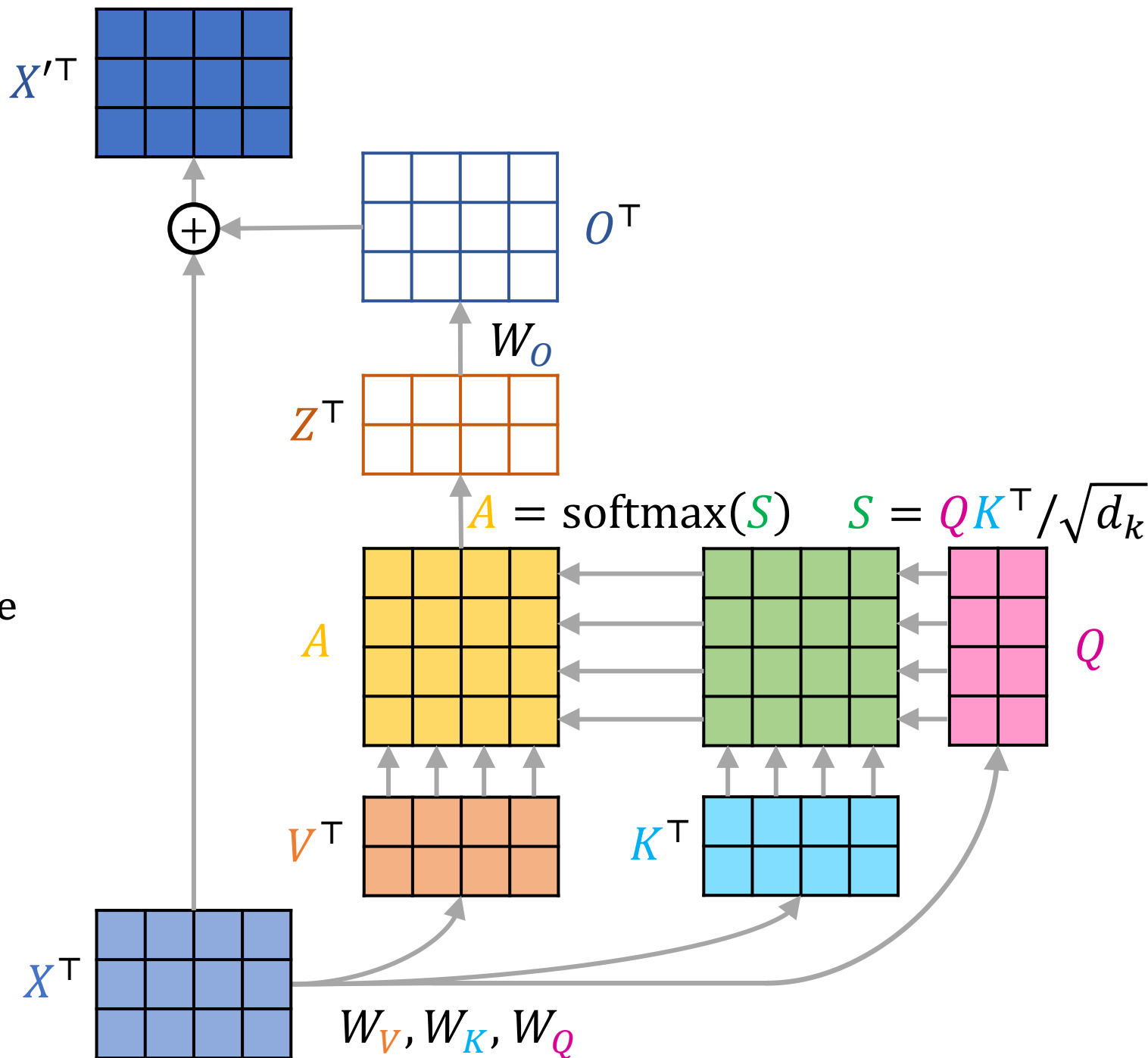
$$A = \text{softmax}(S)_{\text{row-wise}}$$

$$S = QK^T / \sqrt{d_k}$$

$$Q = XW_Q$$

$$K = XW_K$$

$$V = XW_V$$



Attention Takeaways

Size of weights doesn't change when the context size changes!

- Weights have to learn a good way to convert input tokens to q , k , v regardless of context length
- Better than having different copies of weights for every context size

Similarity matrix in attention is calculated (not a parameter matrix)!

- S does change size based on context but it is generated from two generated matrices Q and K
- Attention is learned from Q and K via W_Q and W_K

Linear operations!

- GPUs already do this fast

Transformer Language Models

Increasing context size

- Uniform average of context vectors
- Position encoding

Attention

- Weighted average of context vectors
- Query Keys Values
- **Expressive power of linear transforms**

Building GPT2!

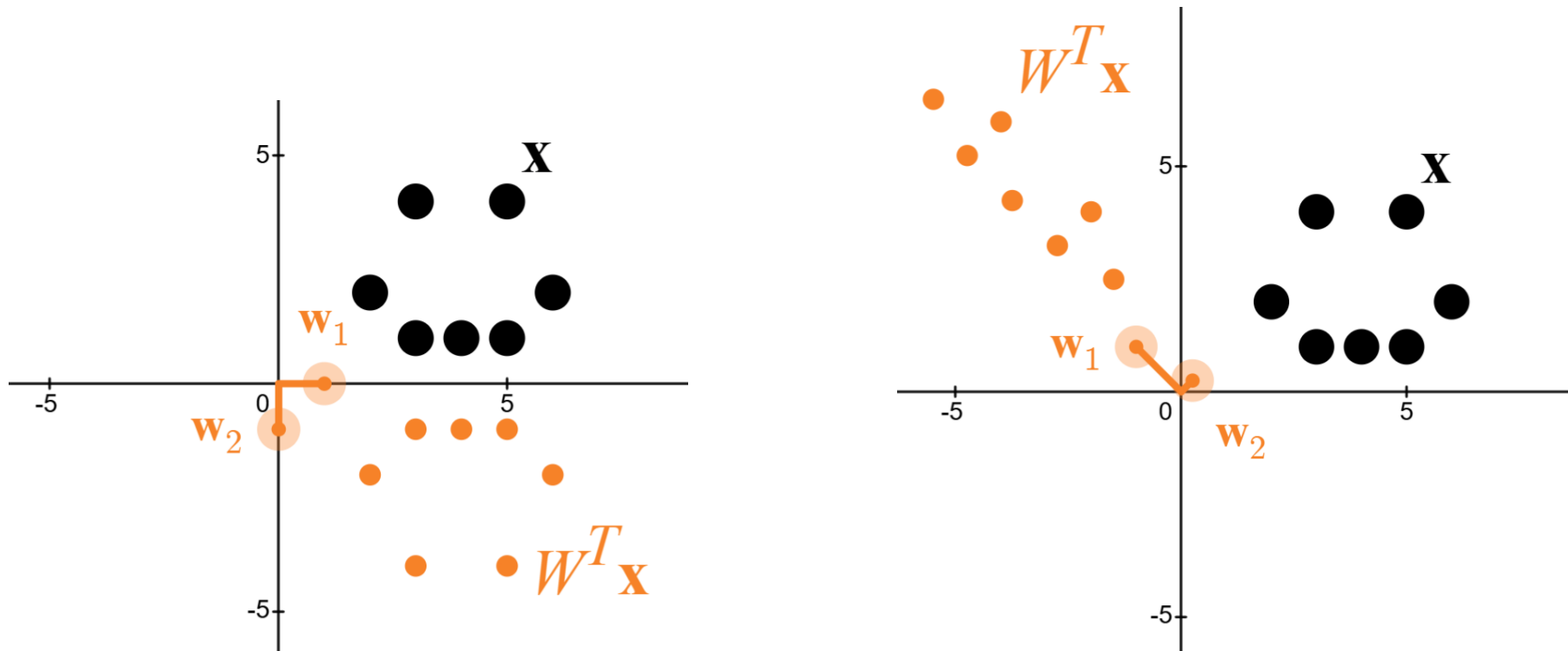
Linear Transforms: Graphical Intuition

In Transformer models, we see quite a few linear transforms

A simple $\mathbf{z} = W^T \mathbf{x}$ can move points quite a bit

Desmos example for $\mathbf{x}$ and $\mathbf{z}$ in $\mathbb{R}^2$

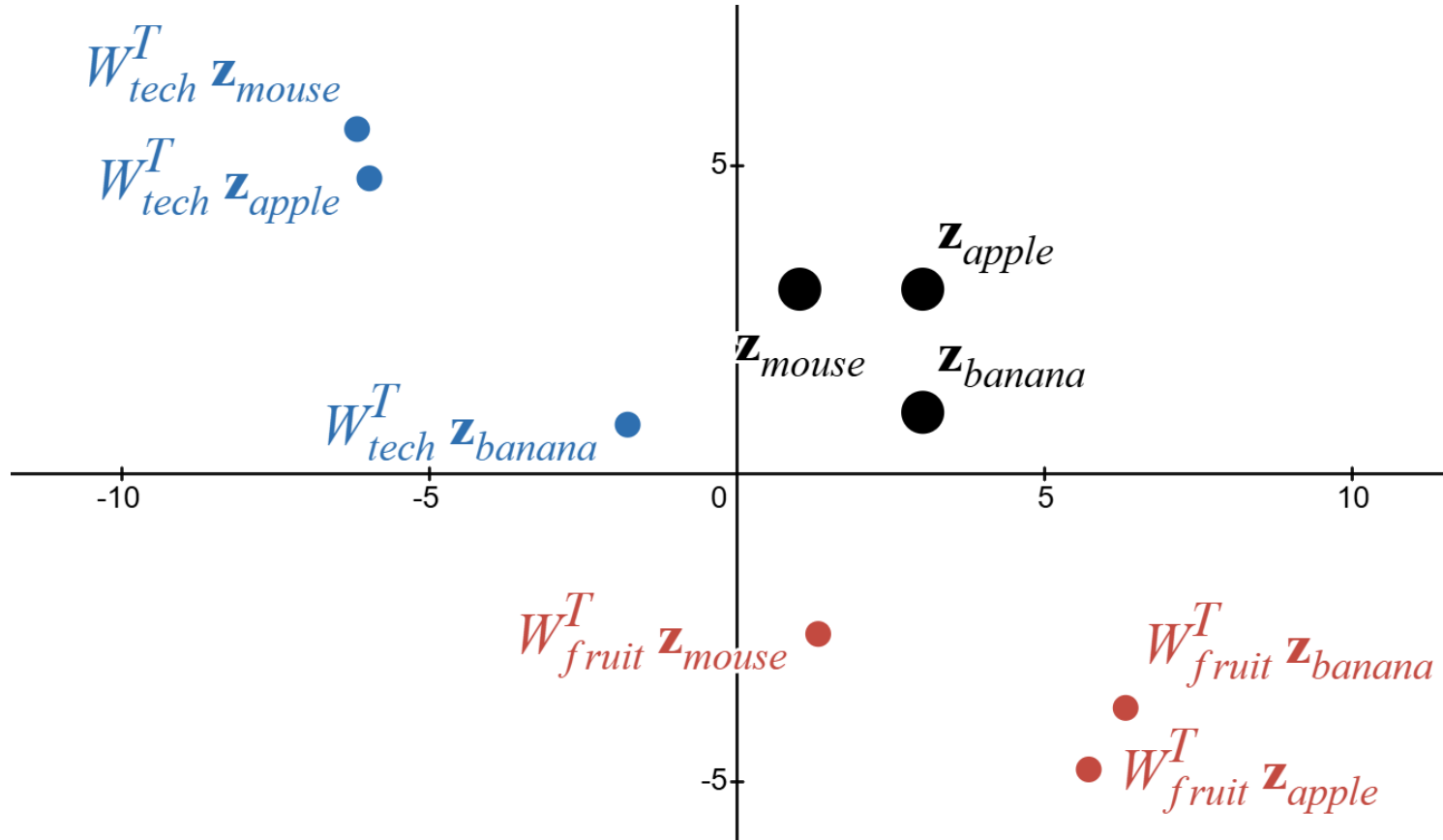
<https://www.desmos.com/calculator/gl5ljvorcy>



Linear Transforms: Graphical Intuition

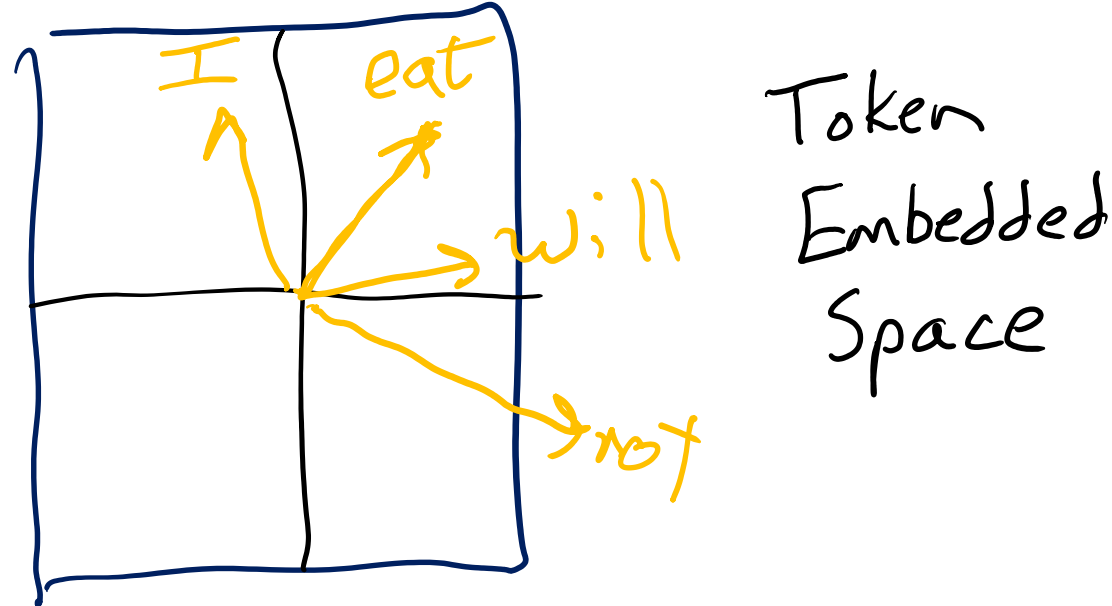
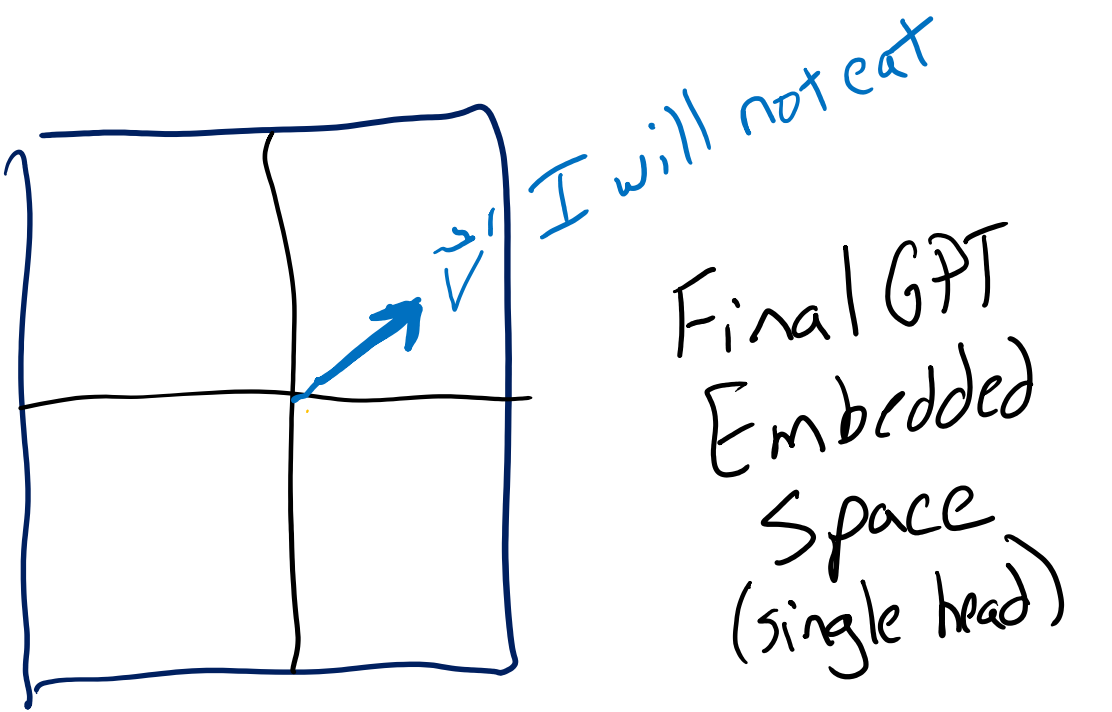
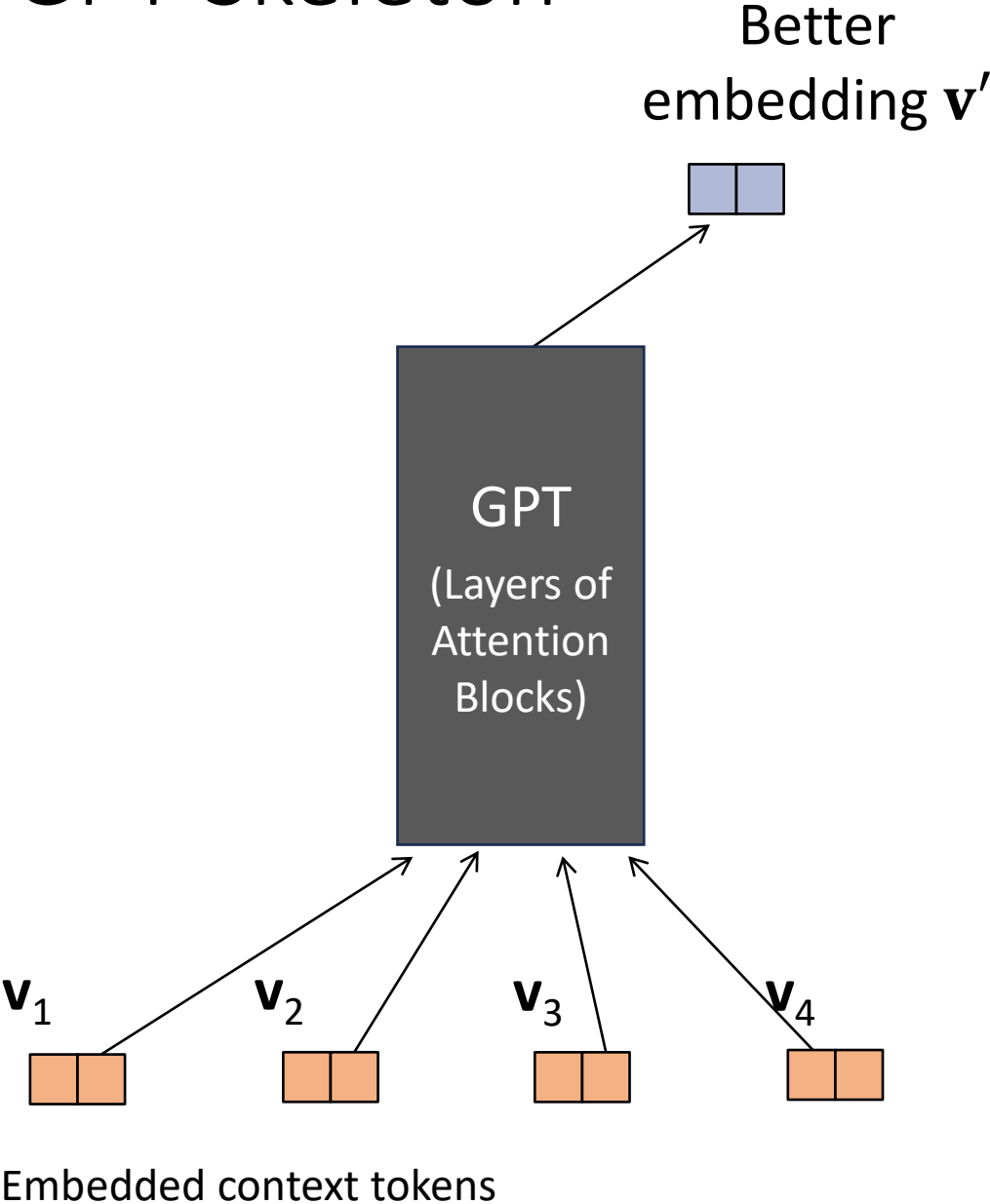
Two different transforms $W_{tech}^T \mathbf{z}$ and $W_{fruit}^T \mathbf{z}$ can create two different meaningful embeddings for the input vectors $\mathbf{z}$

Desmos example for W in $\mathbb{R}^{2 \times 2}$ <https://www.desmos.com/calculator/tbec1bo83h>

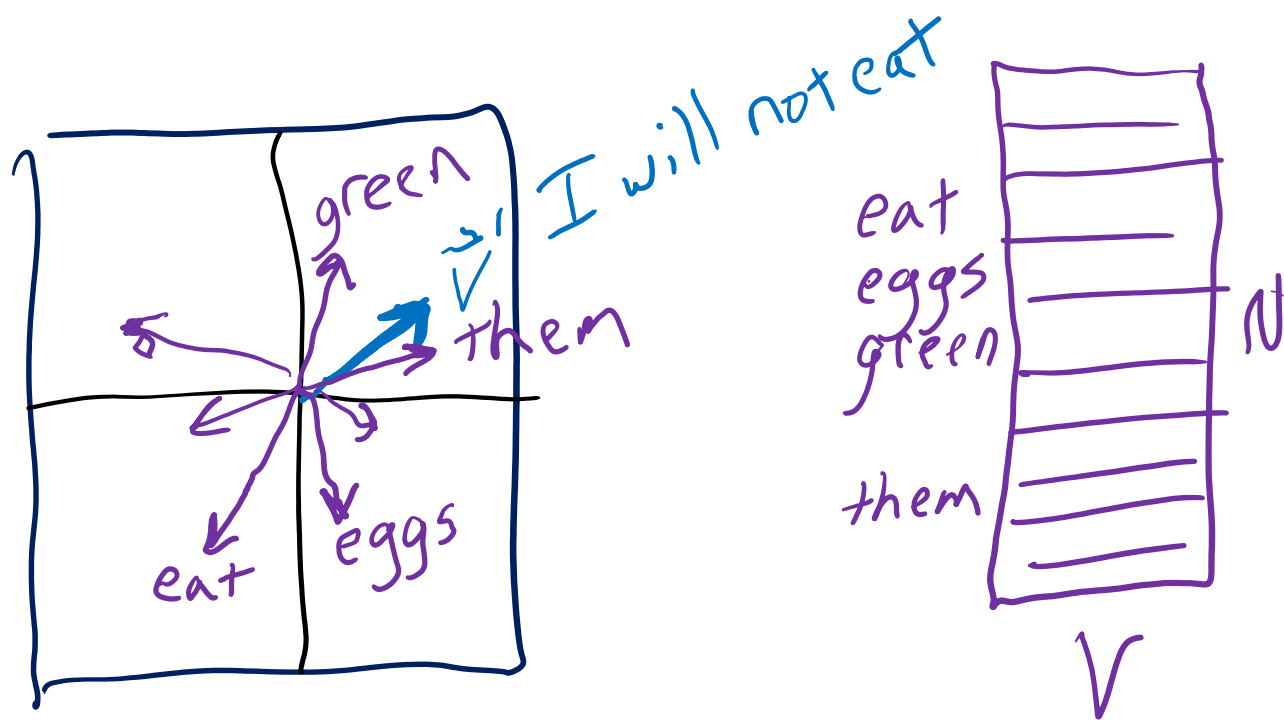
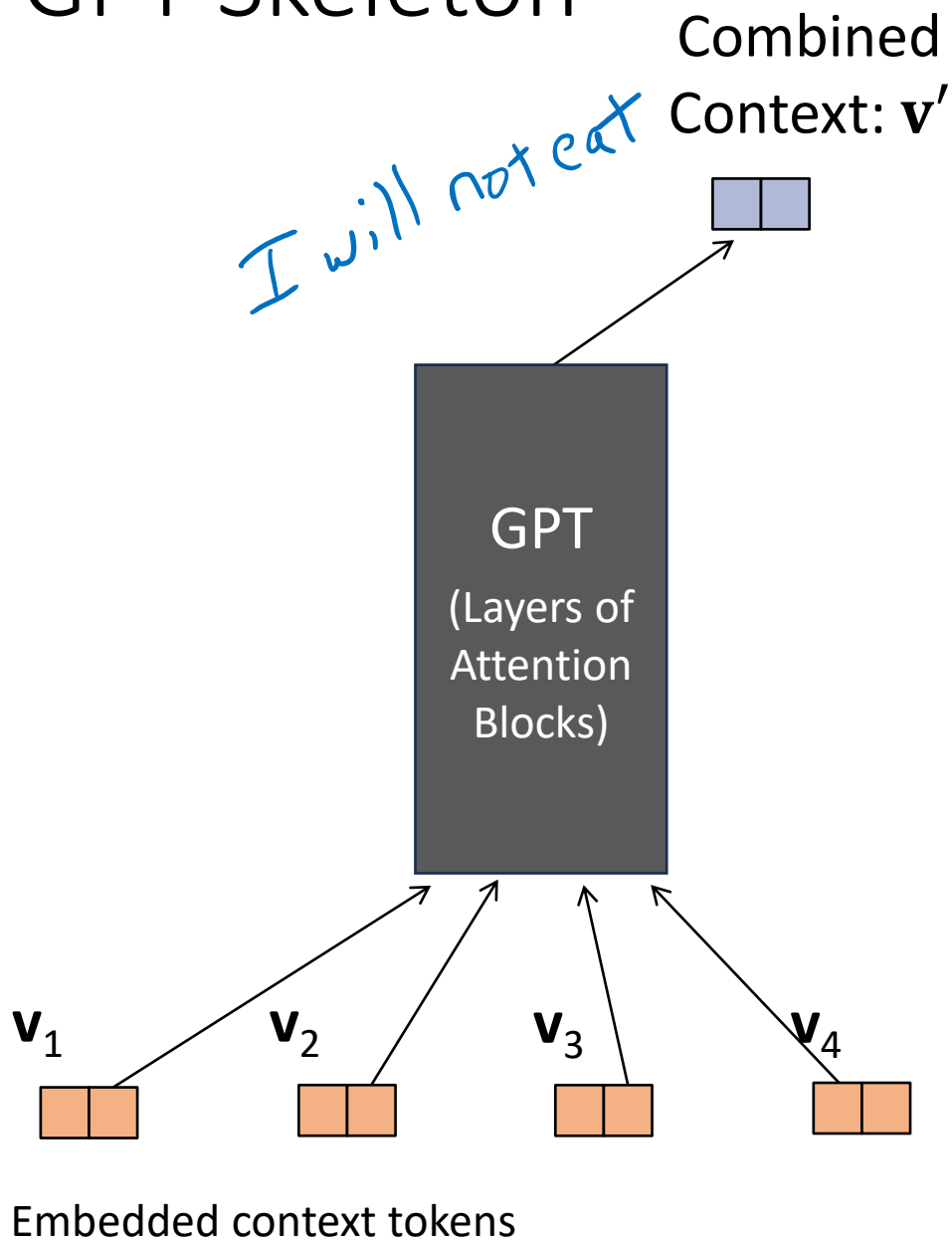


Building GPT2!

GPT Skeleton



GPT Skeleton



Final GPT
linear layer

$$\hat{\mathbf{y}} = g_{\text{softmax}}(U\mathbf{v}')$$

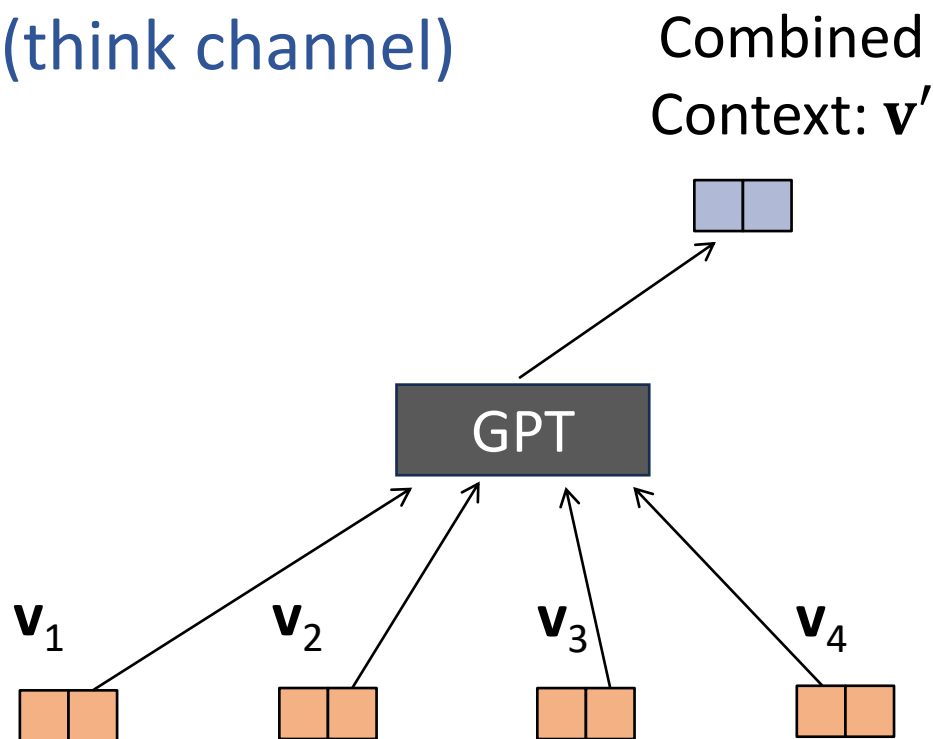
$$\text{next_idx} = \underset{j}{\operatorname{argmax}} \hat{\mathbf{y}}$$

MinGPT Femto

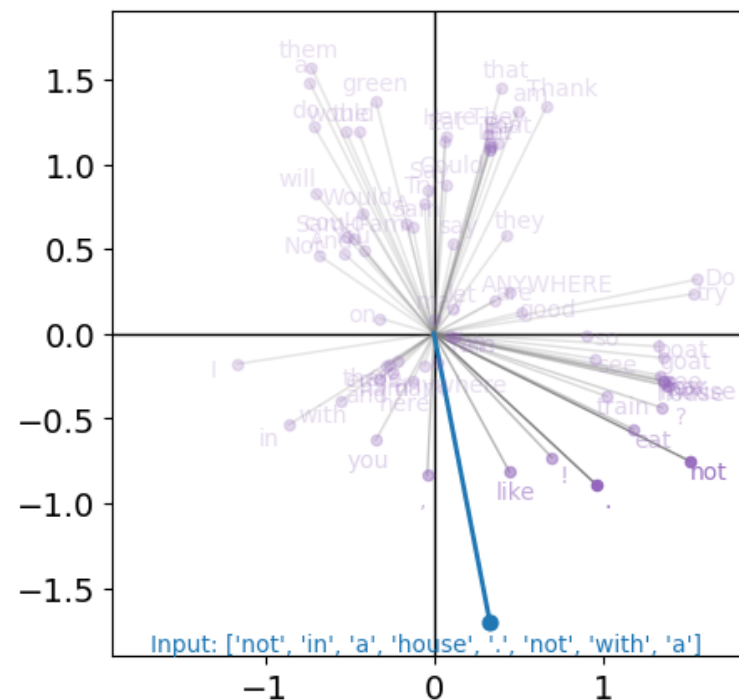
2-D embedded
space

1 attention layer

1 attention head
(think channel)



Embedded words/tokens



$$\hat{\mathbf{y}} = g_{softmax}(U\mathbf{v}')$$

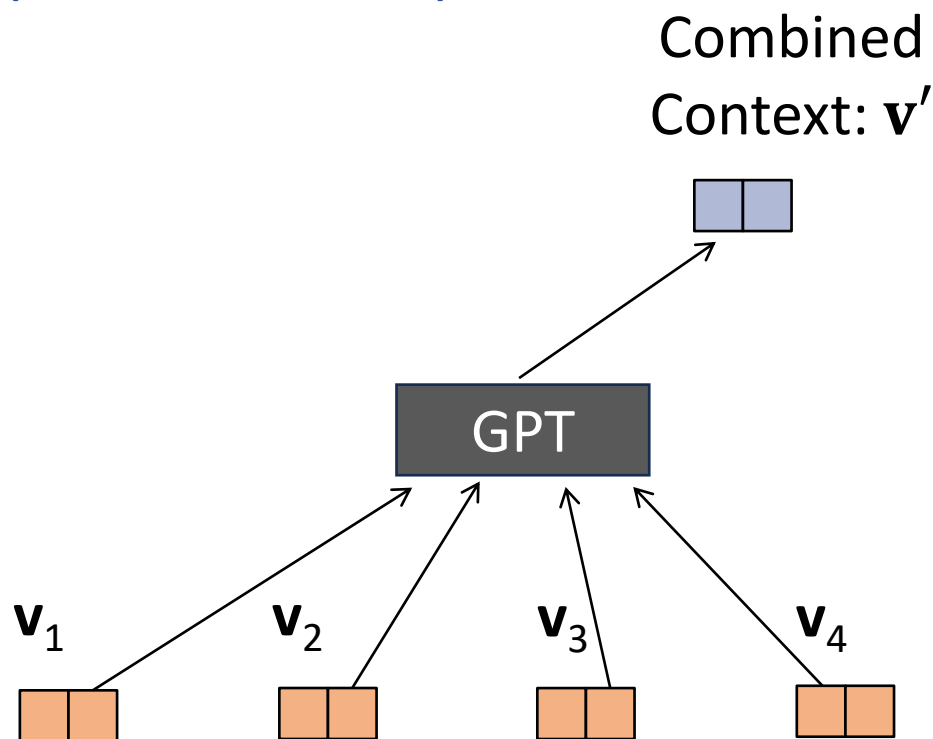
$$next_idx = \underset{j}{\operatorname{argmax}} \hat{\mathbf{y}}$$

MinGPT Femto

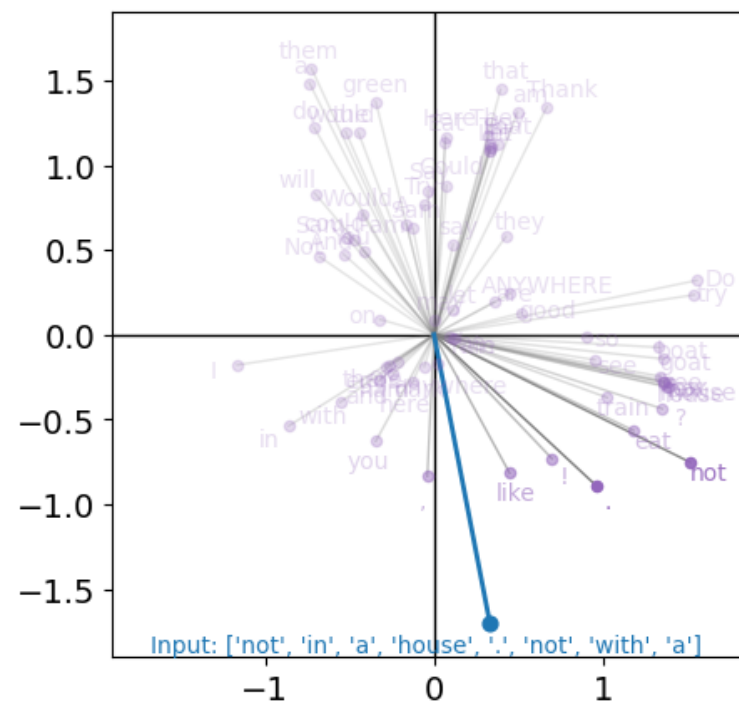
2-D embedded space

1 attention layer

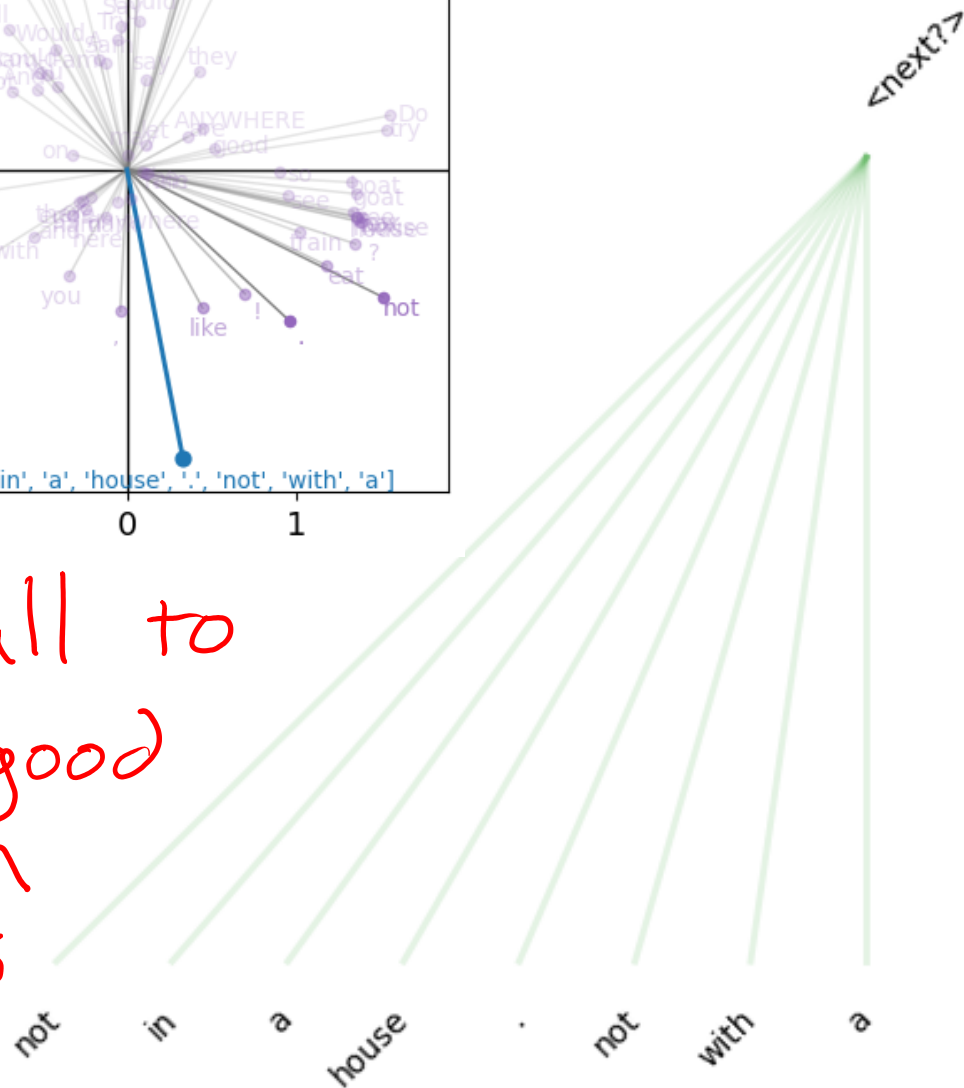
1 attention head
(think channel)



Embedded words/tokens



Too small to
learn good
attention
weights

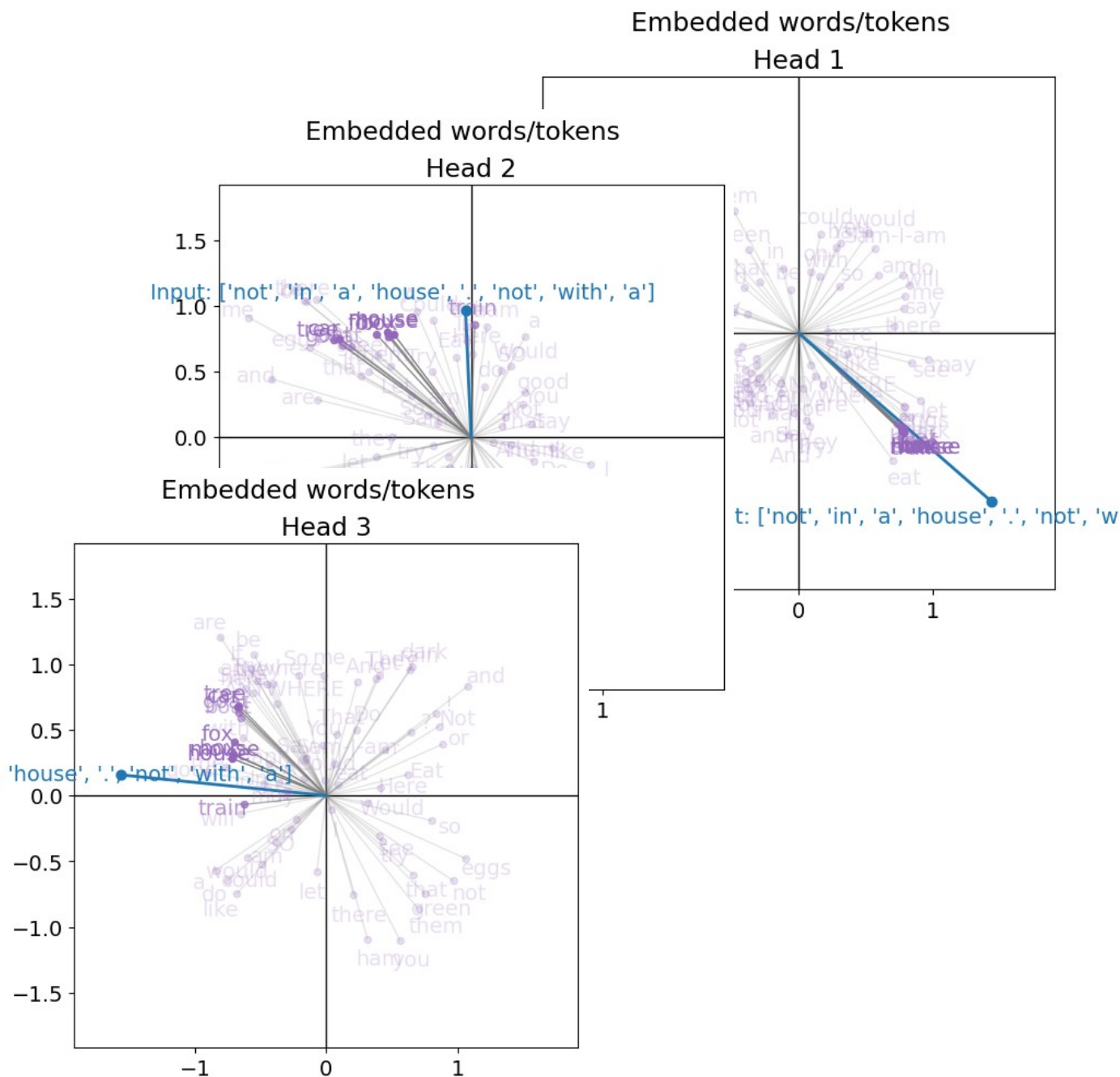
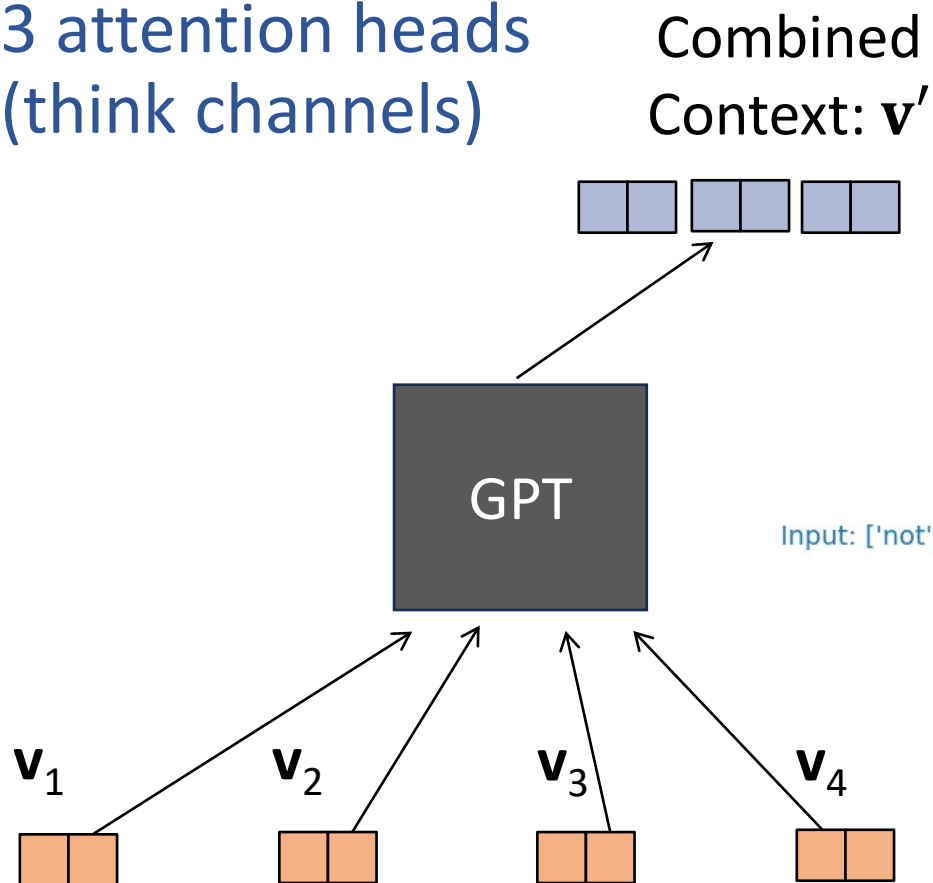


MinGPT Pico

2-D embedded space

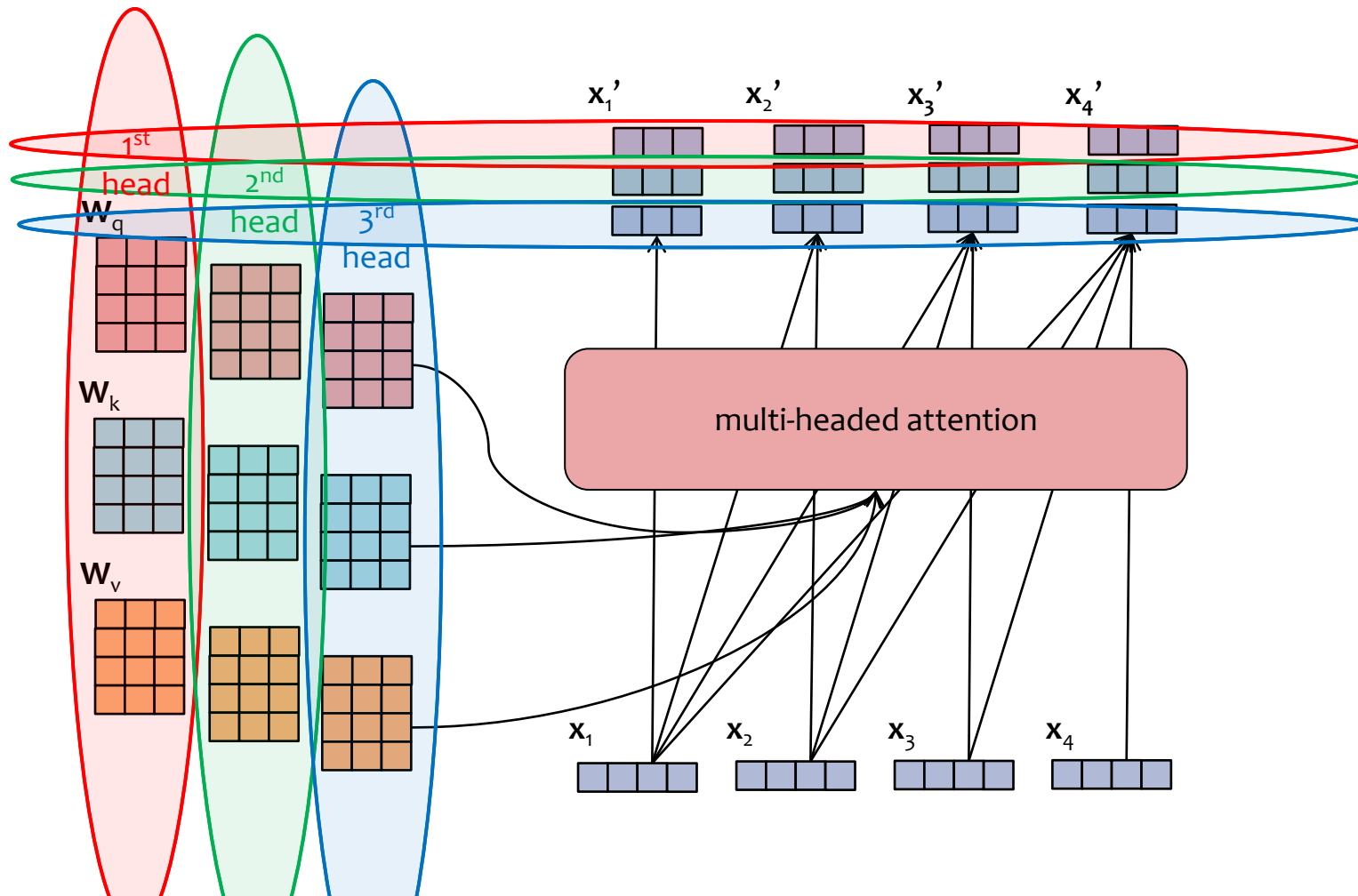
3 attention layer

3 attention heads
(think channels)



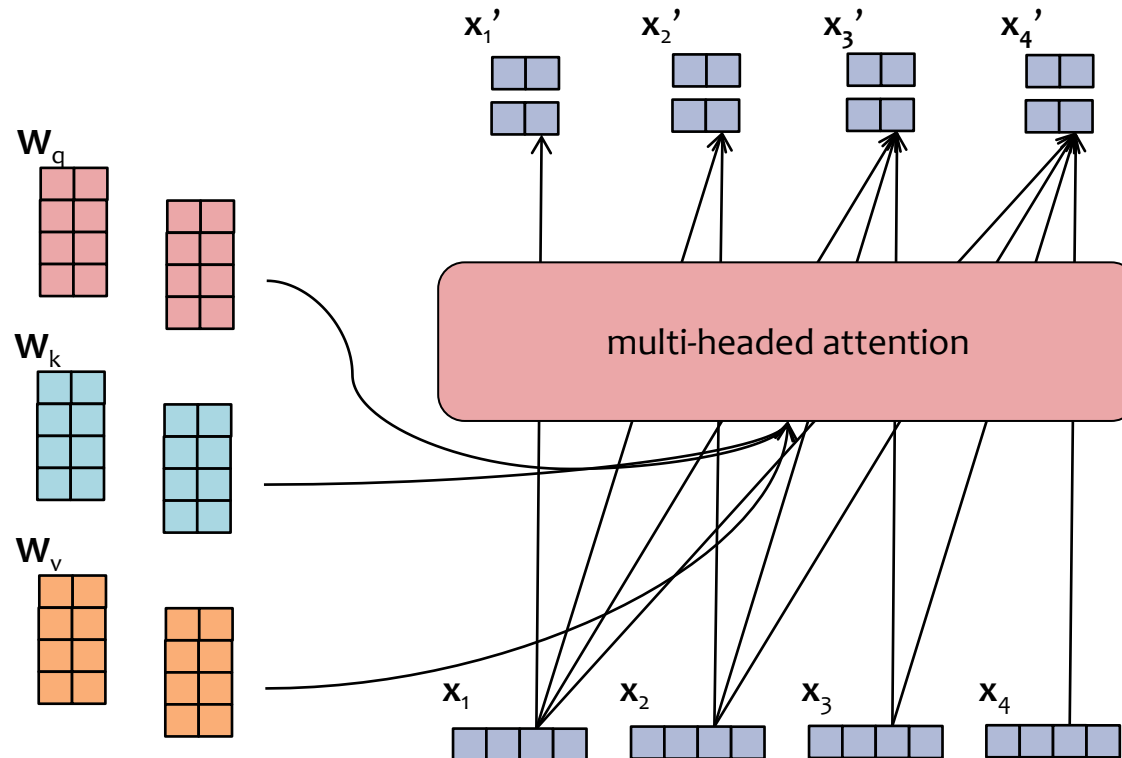
Multi-headed Attention

Multi-headed Attention



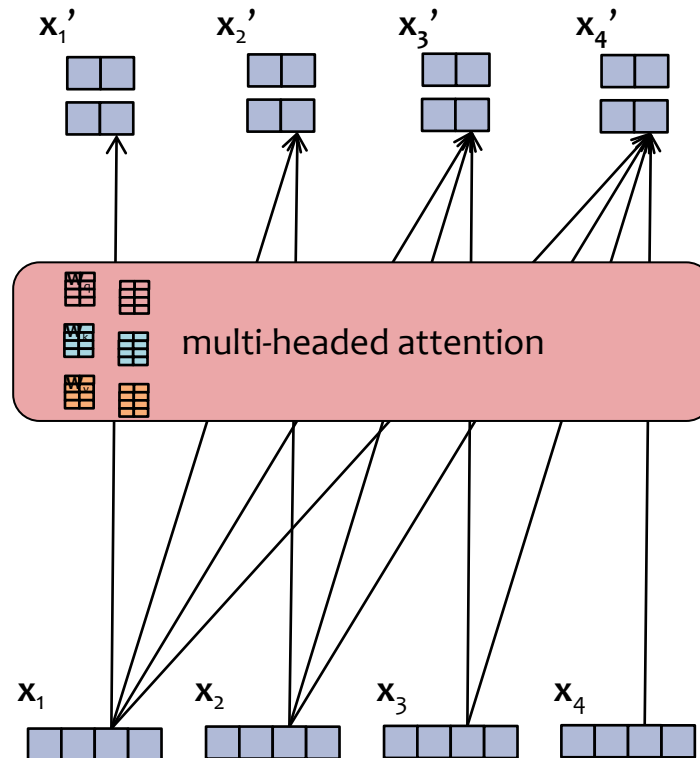
- Just as we can have **multiple channels** in a **convolution** layer, we can use **multiple heads** in an **attention** layer
- Each head gets **its own parameters**
- We can **concatenate** all the outputs to get a single vector for each time step

Multi-headed Attention



- To ensure the dimension of the **input** embedding x_t is the same as the **output** embedding x'_t , Transformers usually choose the embedding sizes and number of heads appropriately:
 - $d_{\text{model}} = \text{dim. of inputs}$
 - $d_k = \text{dim. of each output}$
 - $h = \# \text{ of heads}$
 - Choose $d_k = d_{\text{model}} / h$
- Then concatenate the outputs

Multi-headed Attention



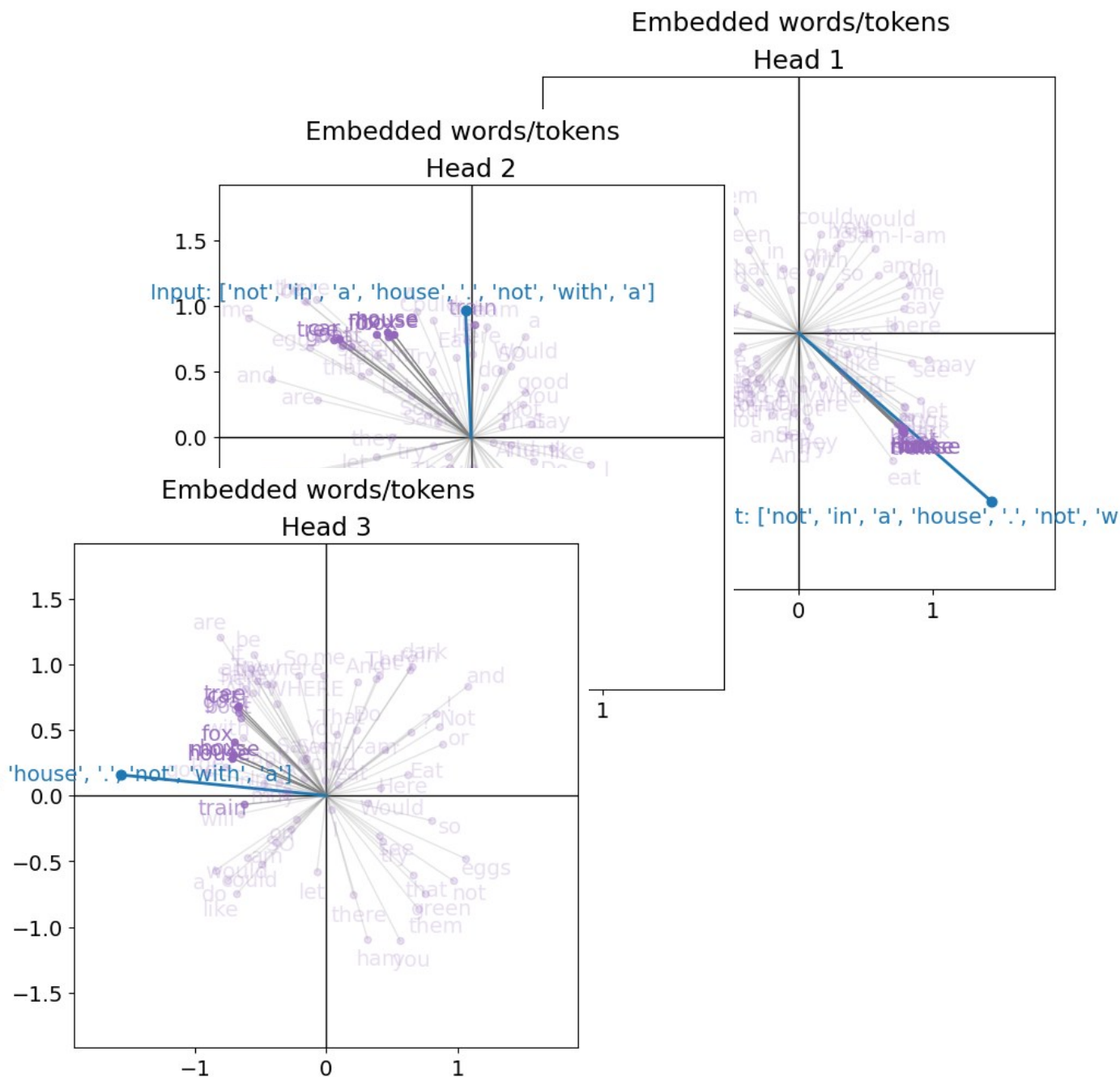
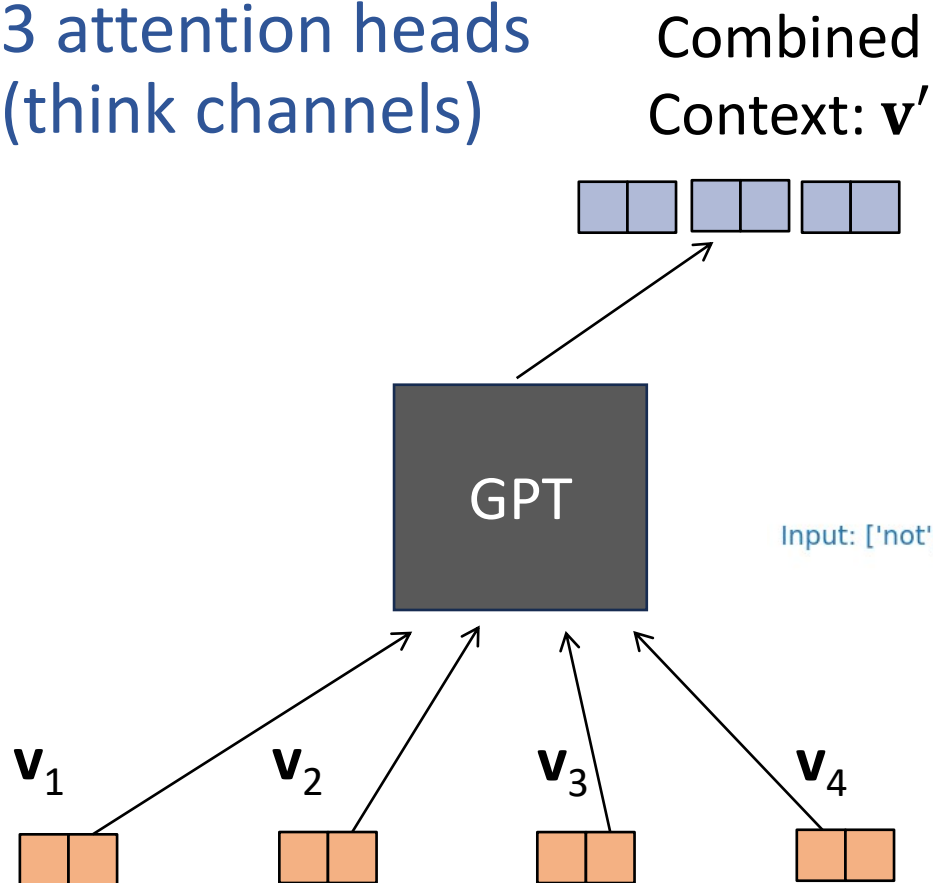
- To ensure the dimension of the **input** embedding x_t is the same as the **output** embedding x'_t , Transformers usually choose the embedding sizes and number of heads appropriately:
 - $d_{\text{model}} = \text{dim. of inputs}$
 - $d_k = \text{dim. of each output}$
 - $h = \# \text{ of heads}$
 - Choose $d_k = d_{\text{model}} / h$
- Then concatenate the outputs

MinGPT Pico

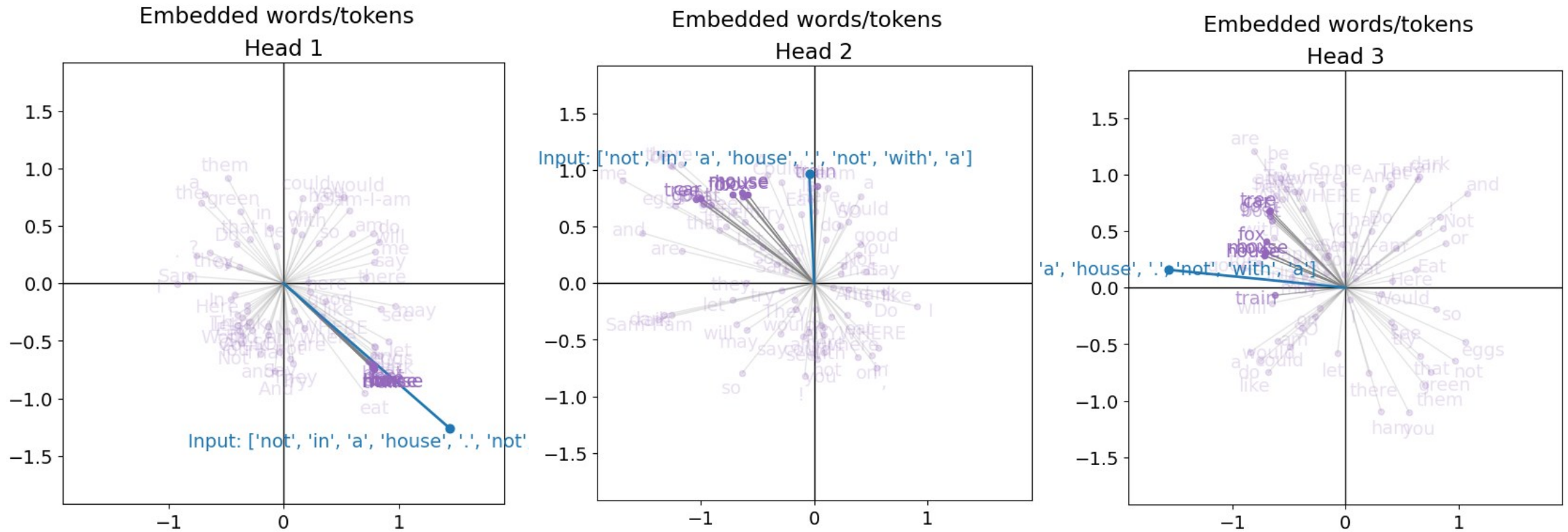
2-D embedded space

3 attention layer

3 attention heads
(think channels)

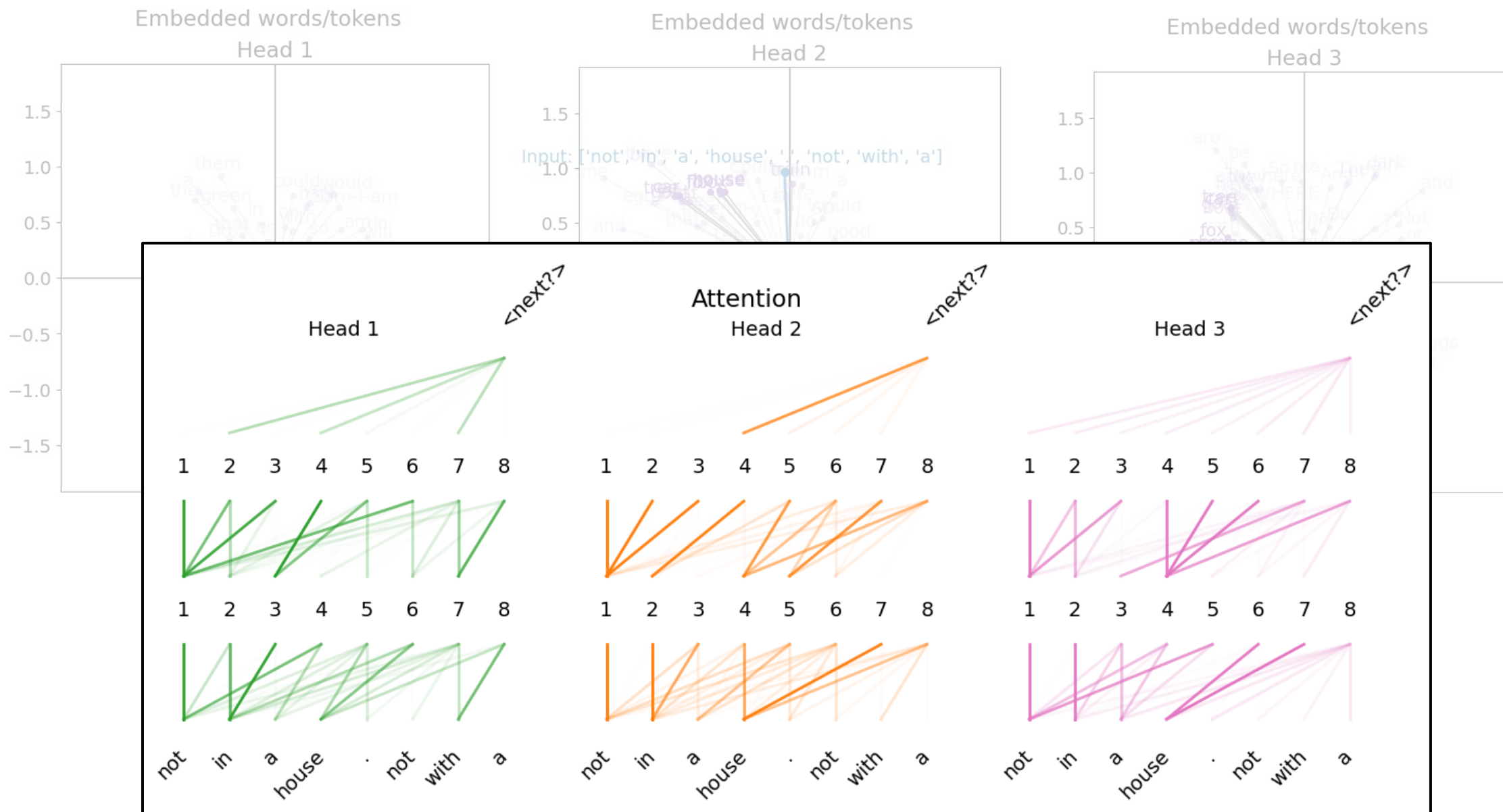


MinGPT Pico: Output embedded space - 3 heads



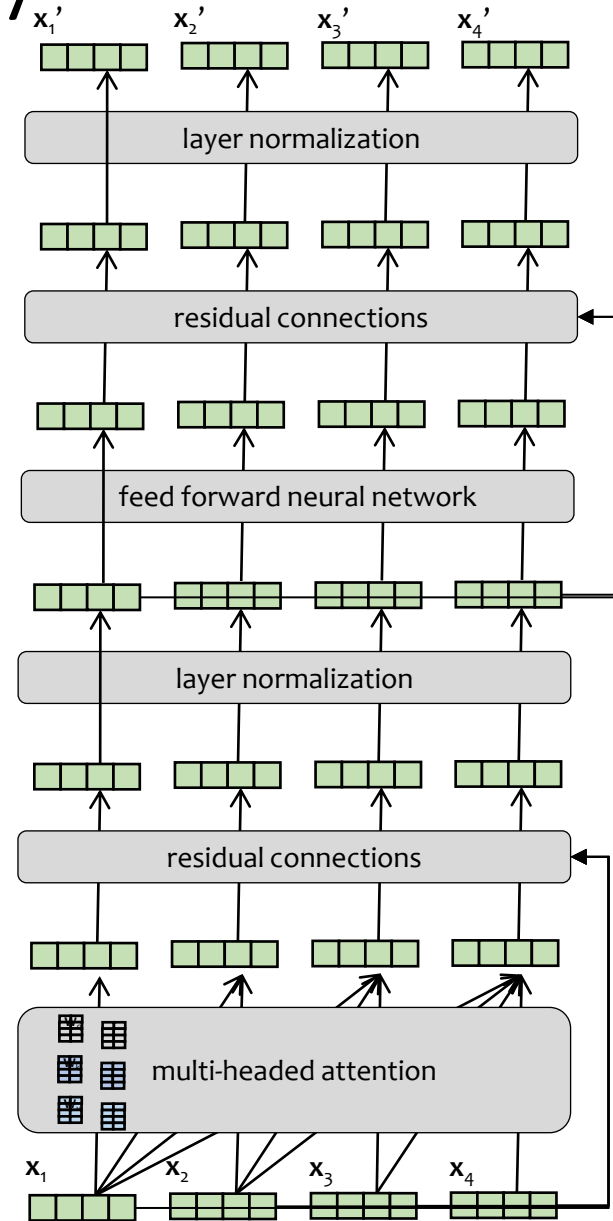
Three heads allows more room to learn different feature representations

MinGPT Pico: Attention Weights – 3 layers, 3 heads



Transformer Layer

Transformer Layer



- Each layer** of a Transformer LM consists of several **sublayers**:
1. attention
 2. feed-forward neural network
 3. layer normalization
 4. residual connections

Layer Normalization

The Problem: **internal covariate shift** occurs during training of a deep network when a small change in the low layers amplifies into a large change in the high layers

One Solution: **Layer normalization** normalizes each layer and learns elementwise gain/bias

- Such normalization allows for higher learning rates (for **faster convergence**) without issues of diverging gradients

Layer Normalization

The Problem: **internal covariate shift** occurs during training of a deep network when a small change in the low layers amplifies into a large change in the high layers

One Solution: **Layer normalization** normalizes each layer and learns elementwise gain/bias

- Such normalization allows for higher learning rates (for **faster convergence**) without issues of diverging gradients

Given input $\mathbf{a} \in \mathbb{R}^K$, LayerNorm computes output $\mathbf{b} \in \mathbb{R}^K$:

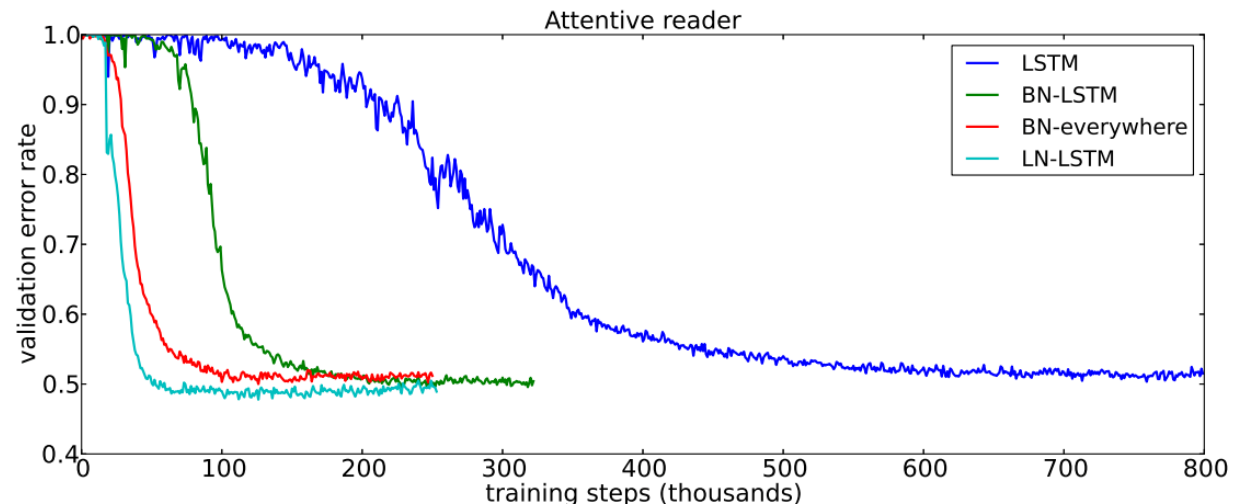
$$\mathbf{b} = \gamma \odot \frac{\mathbf{a} - \mu}{\sigma} \oplus \beta$$

where we have mean $\mu = \frac{1}{K} \sum_{k=1}^K a_k$,

standard deviation $\sigma = \sqrt{\frac{1}{K} \sum_{k=1}^K (a_k - \mu)^2}$,

and parameters $\gamma \in \mathbb{R}^K, \beta \in \mathbb{R}^K$.

$\odot$ and $\oplus$ denote elementwise multiplication and addition.



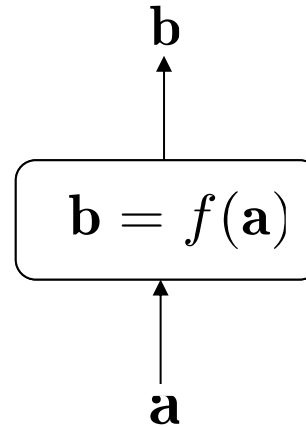
Residual Connections

The Problem: as network depth grows very large, a **performance degradation** occurs that is not explained by overfitting (i.e. train / test error both worsen)

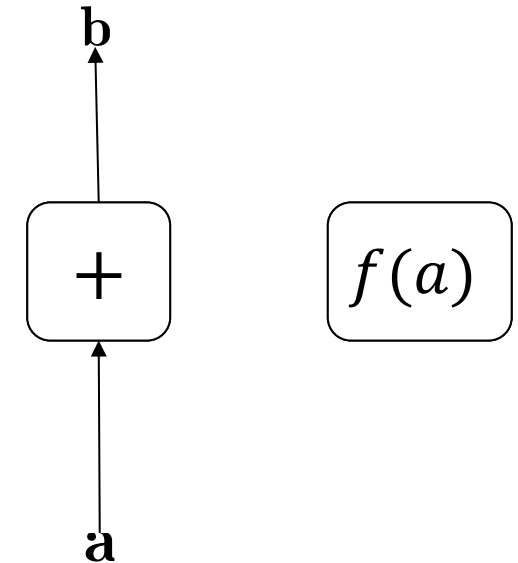
One Solution: Residual connections pass a copy of the input alongside another function so that information can flow more directly

- These residual connections allow for **effective training of very deep networks** that perform better than their shallower (though still deep) counterparts

Plain Connection



Residual Connection



Why are residual connections helpful?

Instead of $f(a)$ having to learn a full transformation of a , $f(a)$ only needs to learn an additive modification of a (i.e. the residual).

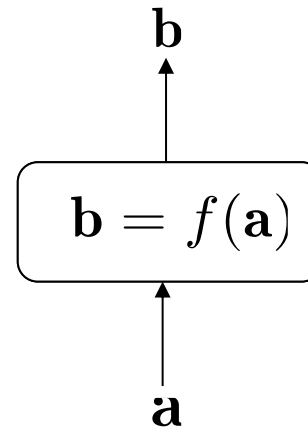
Residual Connections

The Problem: as network depth grows very large, a **performance degradation** occurs that is not explained by overfitting (i.e. train / test error both worsen)

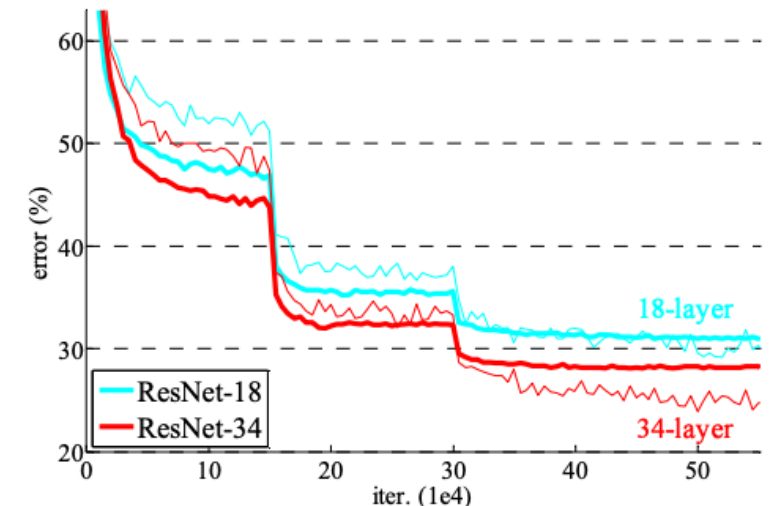
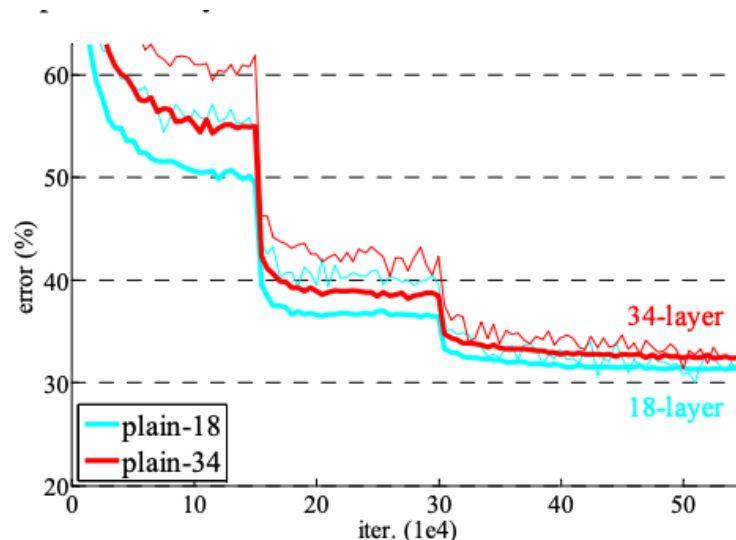
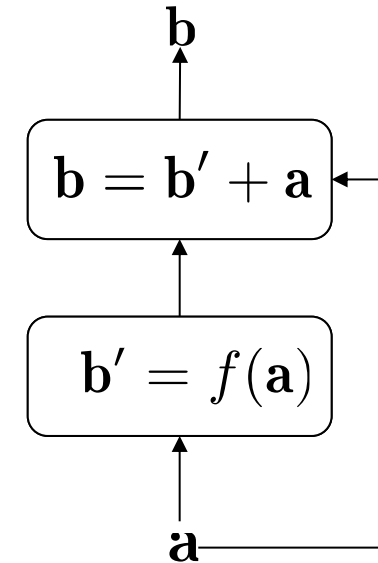
One Solution: **Residual connections** pass a copy of the input alongside another function so that information can flow more directly

- These residual connections allow for **effective training of very deep networks** that perform better than their shallower (though still deep) counterparts

Plain Connection



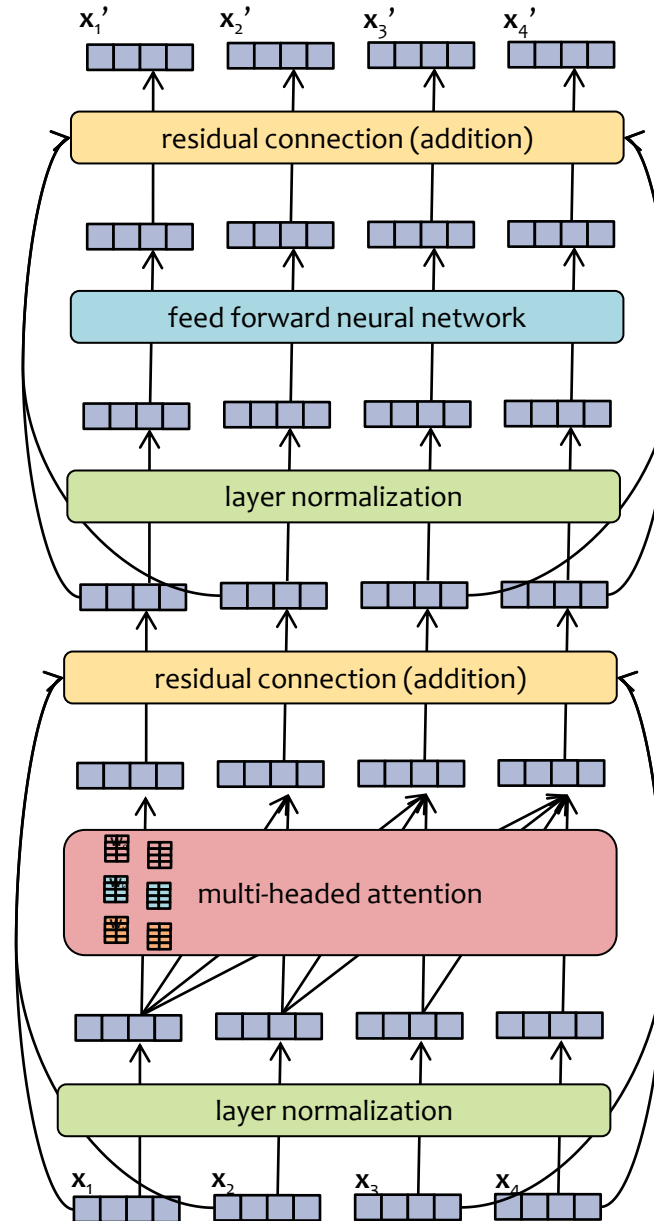
Residual Connection



Transformer Layer

Pre-LN Version:

However, subsequent work found that reordering such that the LayerNorm's came at the beginning of each set of 3 layers, the multi-headed attention and feed-forward NN layers tend to be better behaved (i.e. tricks like warm-up are less important).



Each **layer** of a Transformer LM consists of several **sublayers**:

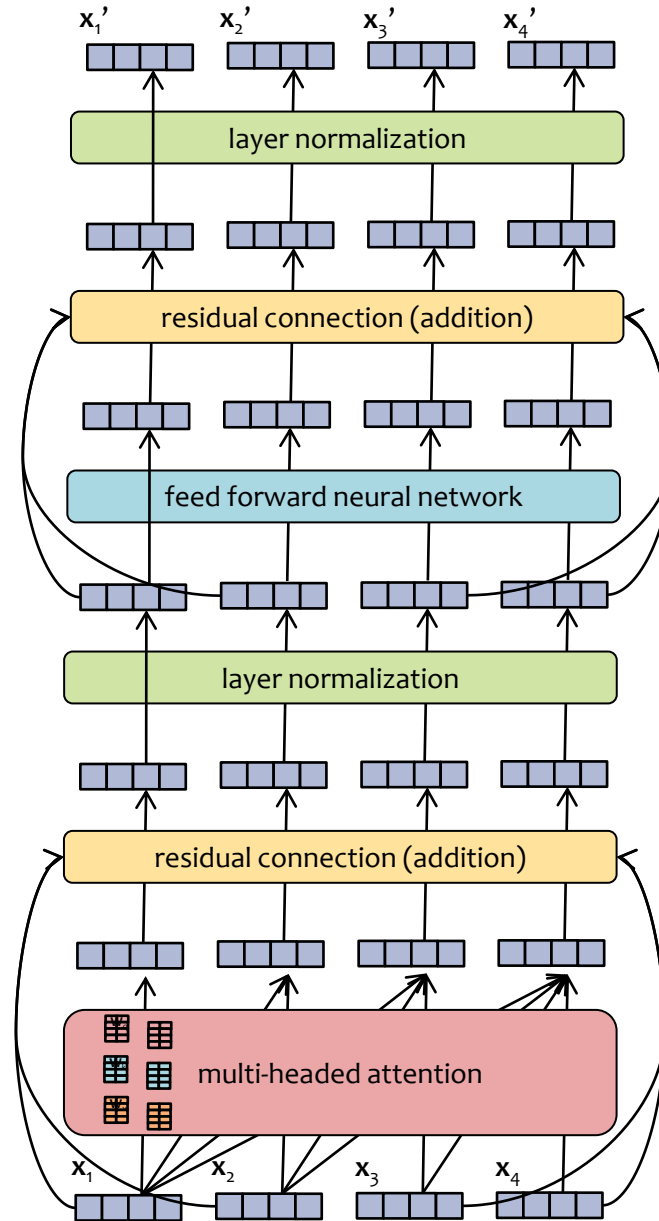
1. attention
2. feed-forward neural network
3. layer normalization
4. residual connections

Transformer Layer

Post-LN Version:

This is the version of the Transformer Layer that was introduced in the original paper in 2017.

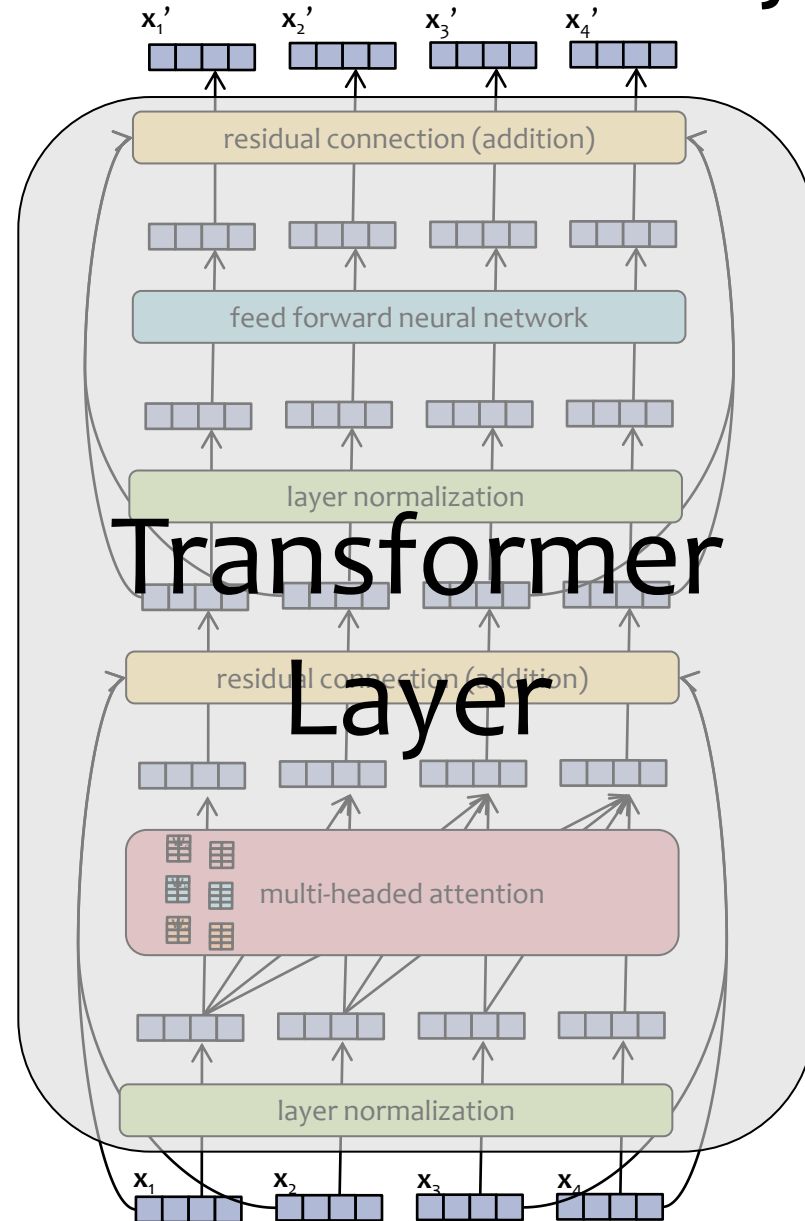
The LayerNorm modules occur at the end of each set of 3 layers.



Each **layer** of a Transformer LM consists of several **sublayers**:

1. attention
2. feed-forward neural network
3. layer normalization
4. residual connections

Transformer Layer



Each layer of a Transformer LM consists of several **sublayers**:

1. attention
2. feed-forward neural network
3. layer normalization
4. residual connections

Transformer Layer



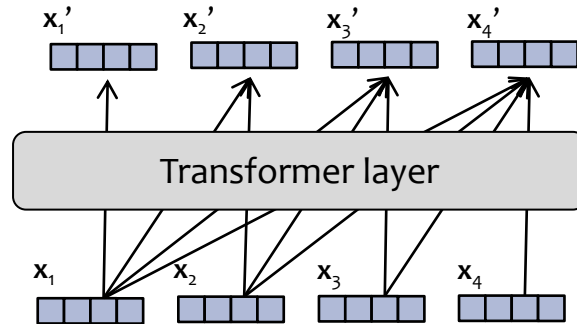
Each layer of a Transformer LM consists of several **sublayers**:

1. attention
2. feed-forward neural network
3. layer normalization
4. residual connections

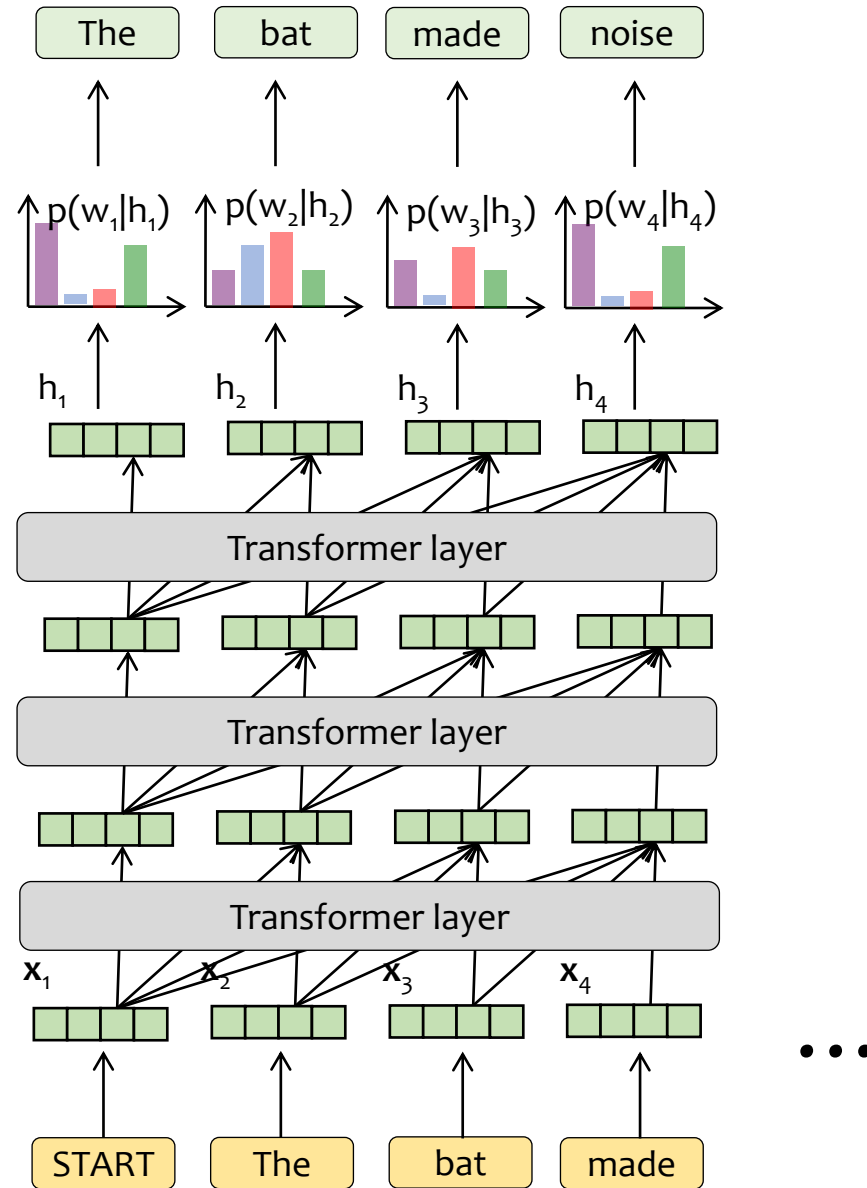
Transformer Layer

Each layer of a Transformer LM consists of several **sublayers**:

1. attention
2. feed-forward neural network
3. layer normalization
4. residual connections



Transformer Language Model



Each layer of a Transformer LM consists of several **sublayers**:

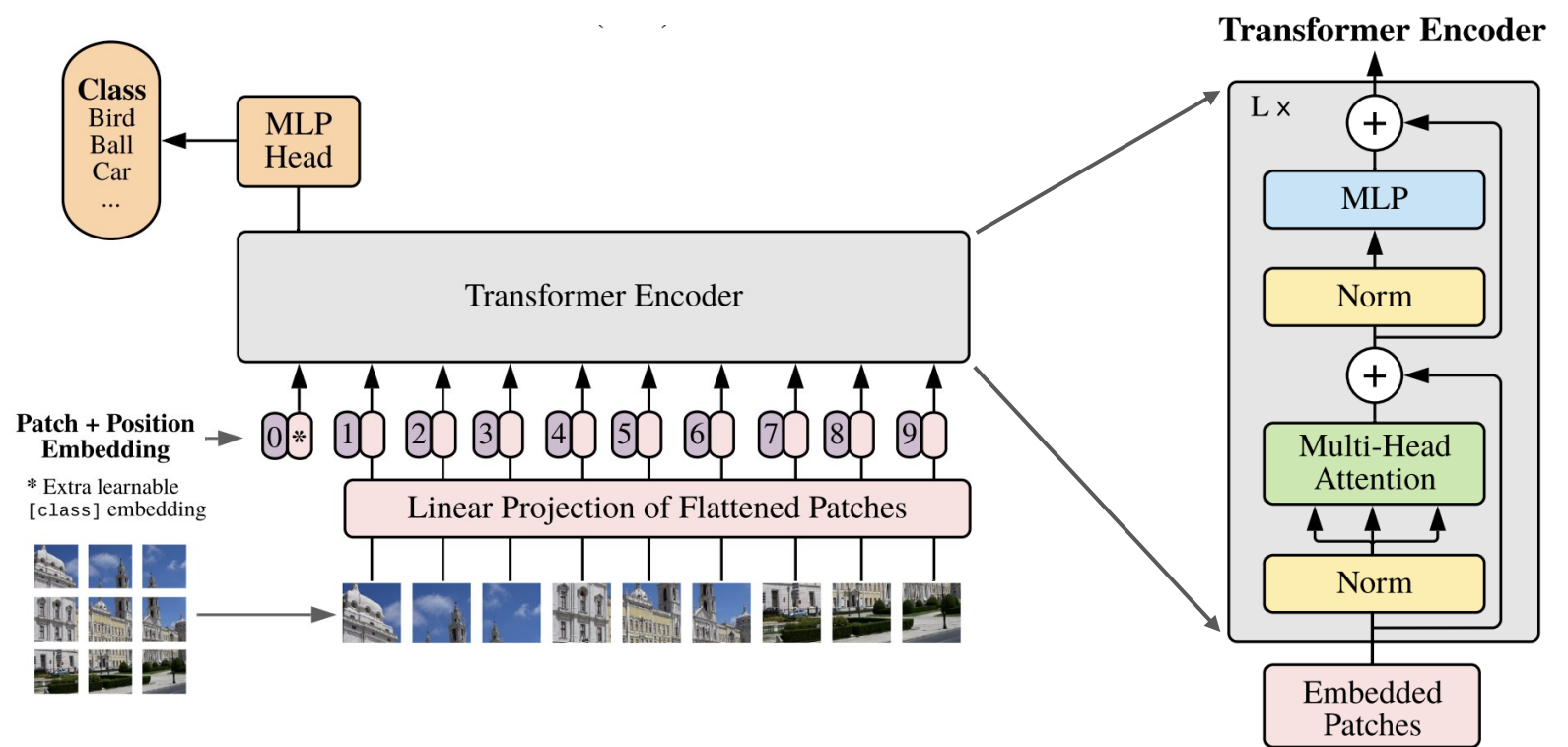
1. attention
2. feed-forward neural network
3. layer normalization
4. residual connections

Each hidden vector looks back at the hidden vectors of the **current and previous timesteps in the previous layer**.

The language model part is just like an RNN-LM!

Vision Transformers

Vision Transf



- Instead of words as input, the inputs are $P \times P$ pixel *patches*
- Each patch is embedded linearly into a vector of size 1024
- Uses 1D positional embeddings
- Pre-trained on a large, supervised dataset (e.g., ImageNet 21K, JFT-300M)

From LM to Chat

We trained a GPT LM!

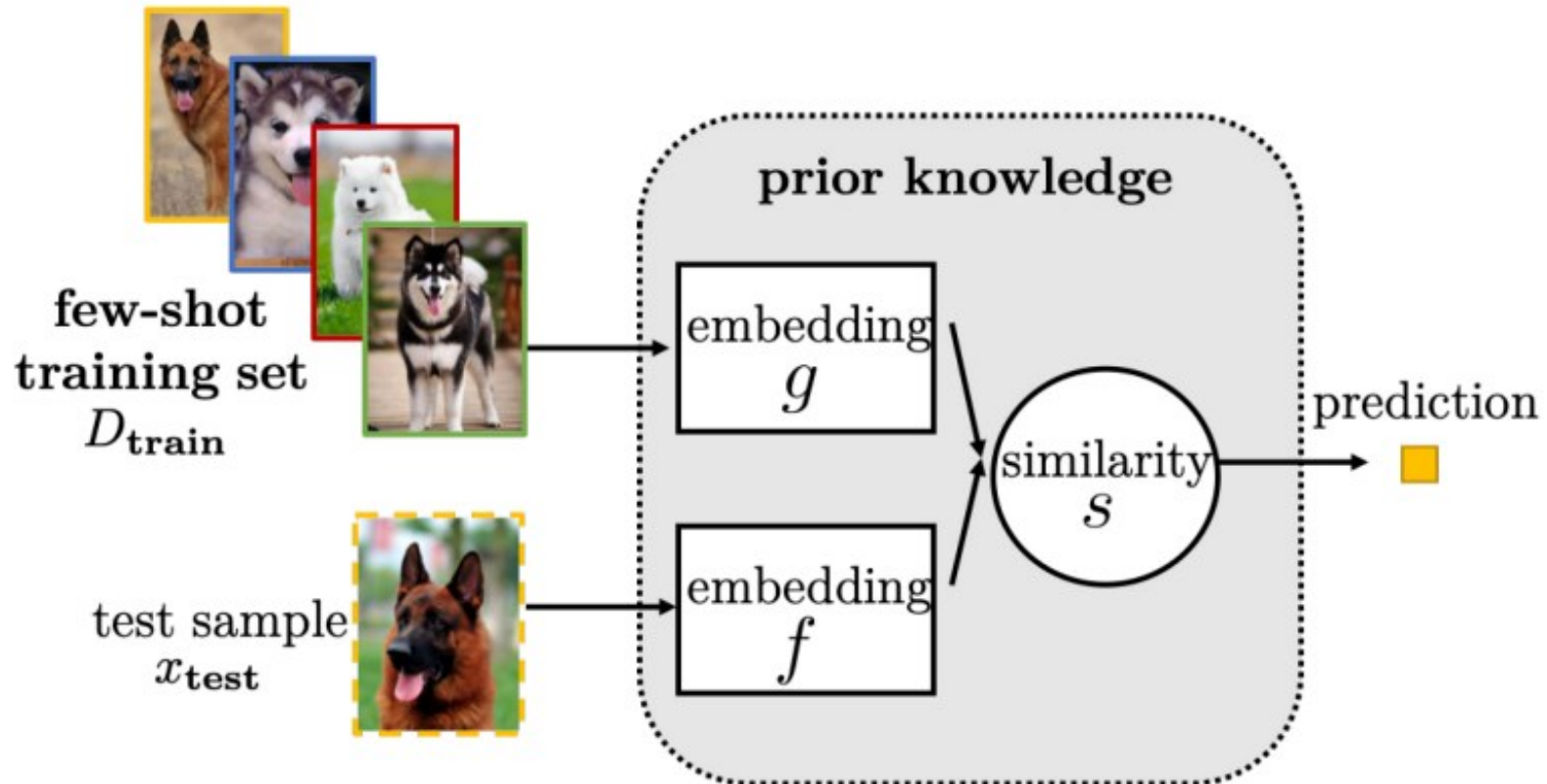
But all it can do is auto-complete sentences 😞

How can we get it to do more?

In-Context Learning

Few-shot Learning

Definition: in **few-shot learning** we assume that training data contains a handful (maybe two, three, or four) examples of each label



Few-shot Learning with LLMs

Suppose you have...

- a dataset $D = \{(x_i, y_i)\}_{i=1}^N$ and N is rather small (i.e. few-shot setting)
- a very large (billions of parameters) pre-trained language model

There are two ways to “learn”

This section!



Option A: Supervised fine-tuning

- **Definition:** fine-tune the LLM on the training data using...
 - a standard supervised objective
 - backpropagation to compute gradients
 - your favorite optimizer (e.g. Adam)
- **Pro:** fits into the standard ML recipe
- **Pro:** still works if N is large
- **Con:** backpropagation requires $\sim 3x$ the memory and computation time as the forward computation
- **Con:** you might not have access to the model weights at all (e.g. because the model is proprietary)

Option B: In-context learning

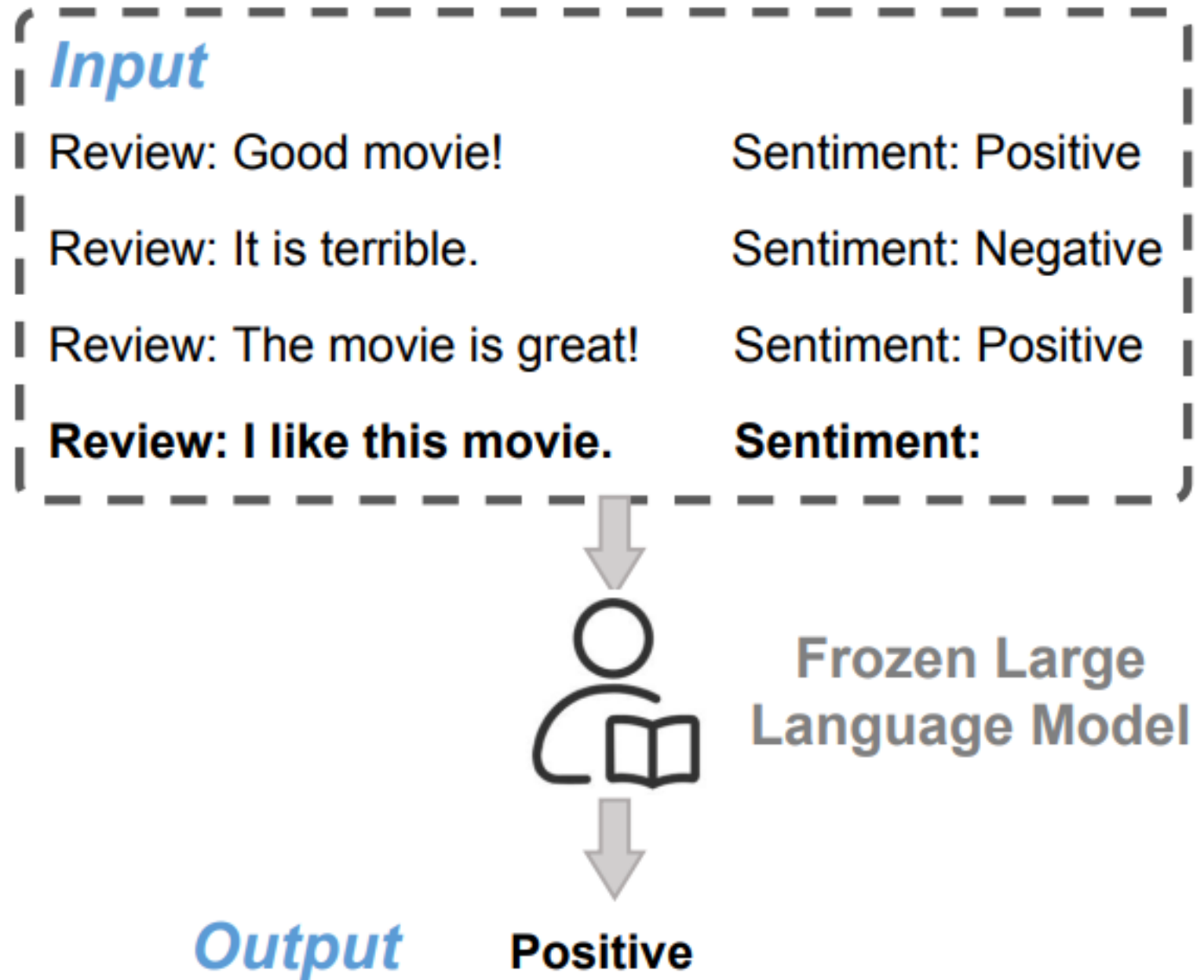
- **Definition:**
 1. feed training examples to the LLM as a prompt
 2. allow the LLM to infer patterns in the training examples during inference (i.e. decoding)
 3. take the output of the LLM following the prompt as its prediction
- **Con:** the prompt may be very long and Transformer LMs require $O(N^2)$ time/space where N = length of context
- **Pro:** no backpropagation required and only one pass through the training data
- **Pro:** does not require model weights, only API access

Few-shot In-context Learning

Few-shot learning can
be done via in-context
learning

Typically, a task
description is presented
first

Then a sequence of
input/output pairs from
a training dataset are
presented in sequence



Few-shot In-context Learning

Few-shot learning can
be done via in-context
learning

Typically, a task
description is presented
first

Then a sequence of
input/output pairs from
a training dataset are
presented in sequence

The three settings we explore for in-context learning

Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
1 Translate English to French: ← task description
2 cheese => ..... ← prompt
```

One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
1 Translate English to French: ← task description
2 sea otter => loutre de mer ← example
3 cheese => ..... ← prompt
```

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
1 Translate English to French: ← task description
2 sea otter => loutre de mer ← examples
3 peppermint => menthe poivrée ←
4 plush girafe => girafe peluche ←
5 cheese => ..... ← prompt
```

Traditional fine-tuning (not used for GPT-3)

Fine-tuning

The model is trained via repeated gradient updates using a large corpus of example tasks.

