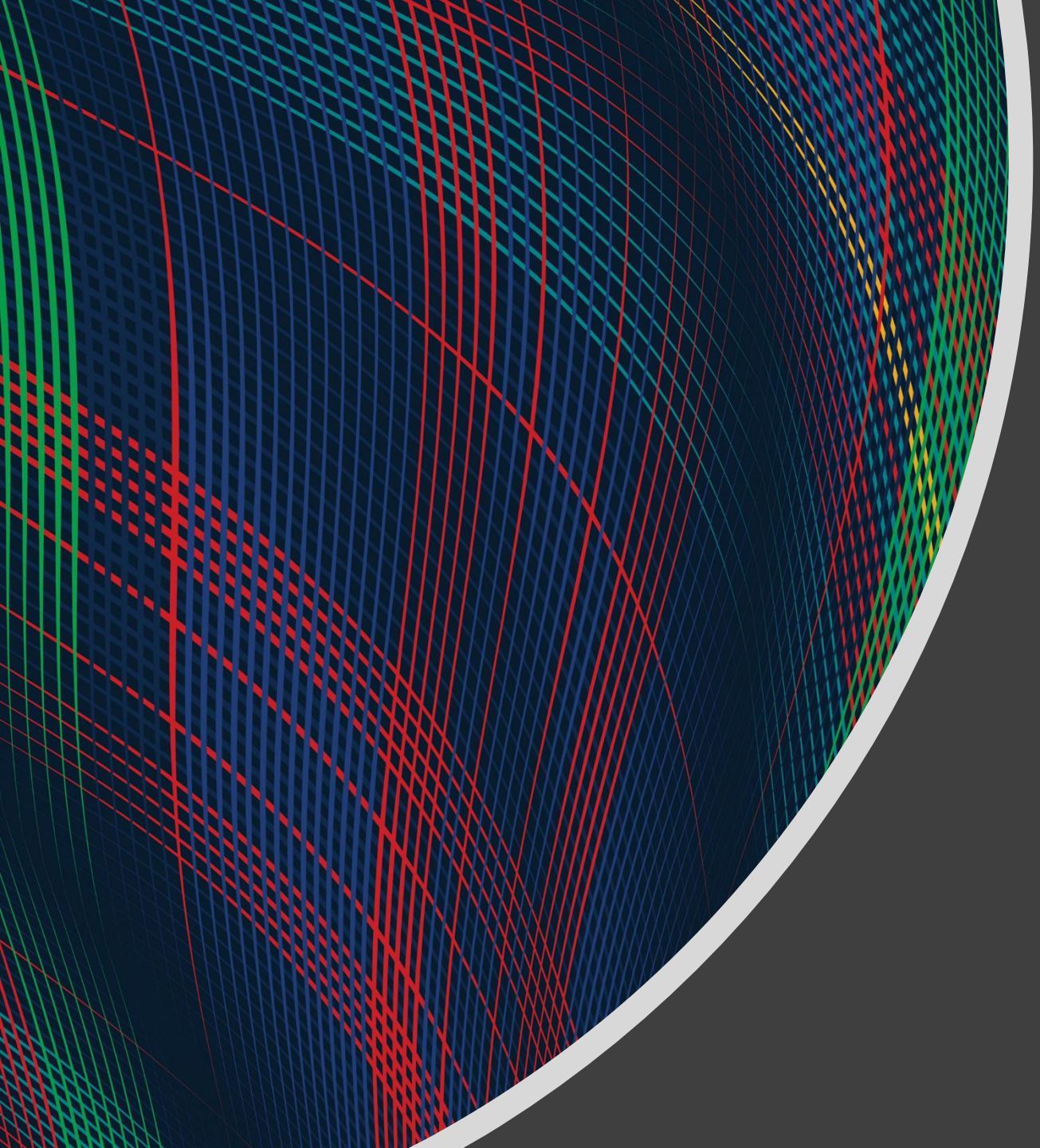


07-280
AI & ML I

NLP: Word Embeddings Attention

Instructors:

Aviral Kumar & Nihar Shah

Sampling from a Language Model

Temperature sampling

Allows us to adjust the level randomness during sampling via hyperparameter T (Temperature):

- T= 1.0: Use the original probabilities (default)
- T > 1.0: "Softer" distribution → more random, diverse outputs
- T < 1.0: "Sharper" distribution → more predictable, focused outputs
- T → 0: Effectively becomes greedy (always pick the most likely token)

Adjusted probability:

$$P_{temperature}(w_i) = \frac{e^{\log P(w_i)/T}}{\sum_j e^{\log P(w_i)/T}}$$

Feature Engineering for Text

Text Features

SPAM Classification

Text Features

One-hot vector for each token

Text Features

Bag of words: Vector of length the size of the vocabulary

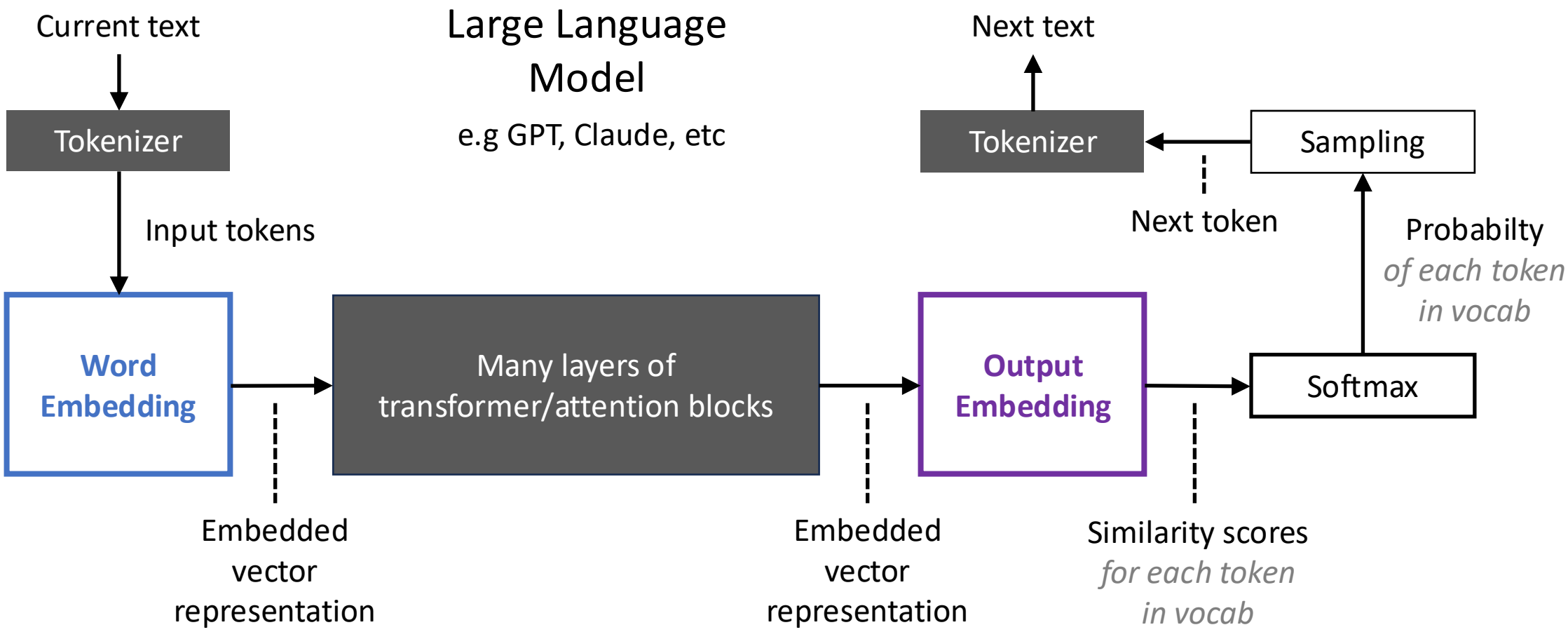
Two options

- Word occurrence: Binary: does the word exist (at least once) or not
- Word histogram: Integer: count of how many times the word appears

Word Embedding LMs

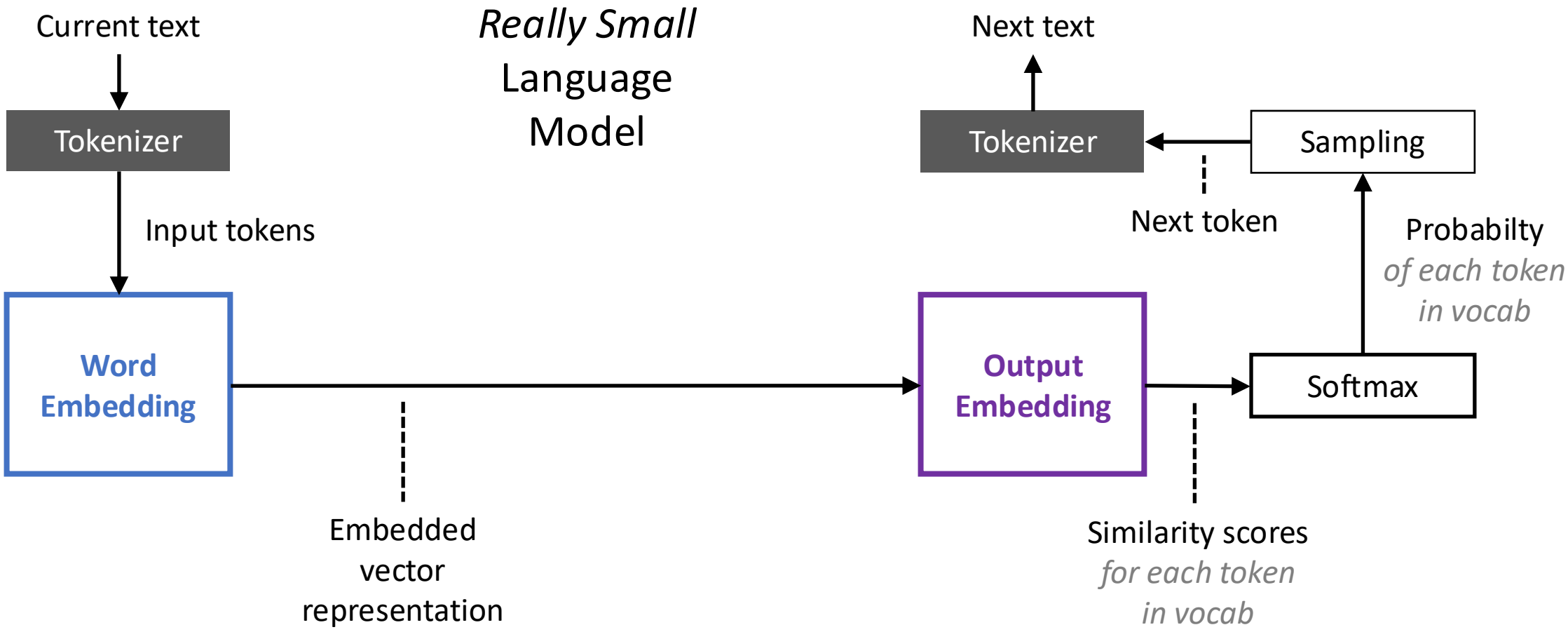
Word (Token) Embeddings

LLM networks



Simple Word Embedding LM

Building a language model with just word embedding layers ☺



Simple Word Embedding LM

Setup

Corpus:

```
The dog ran.
The dog ate.
The dog ran the zoo.
The cat ate the dog!
The cat ran the zoo.
```



(Simple) Tokenized Corpus:

```
[ 'the', 'dog', 'ran', '.', 'the',
  'dog', 'ate', '.', 'the', 'dog',
  'ran', 'the', 'zoo', '.', 'the',
  'cat', 'ate', 'the', 'dog', '!',
  'the', 'cat', 'ran', 'the',
  'zoo', '.' ]
```

Vocabulary:

```
[ '!',
  '.',
  'ate',
  'cat',
  'dog',
  'ran',
  'the',
  'zoo' ]
```

Simple Word Embedding LM

Vector representation for each token in vocabulary (initially random)

Two sets of vectors in $\mathbb{R}^M$ (we'll use $M = 2$ for better visualization)

V: to represent previous tokens

U: to represent 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
```

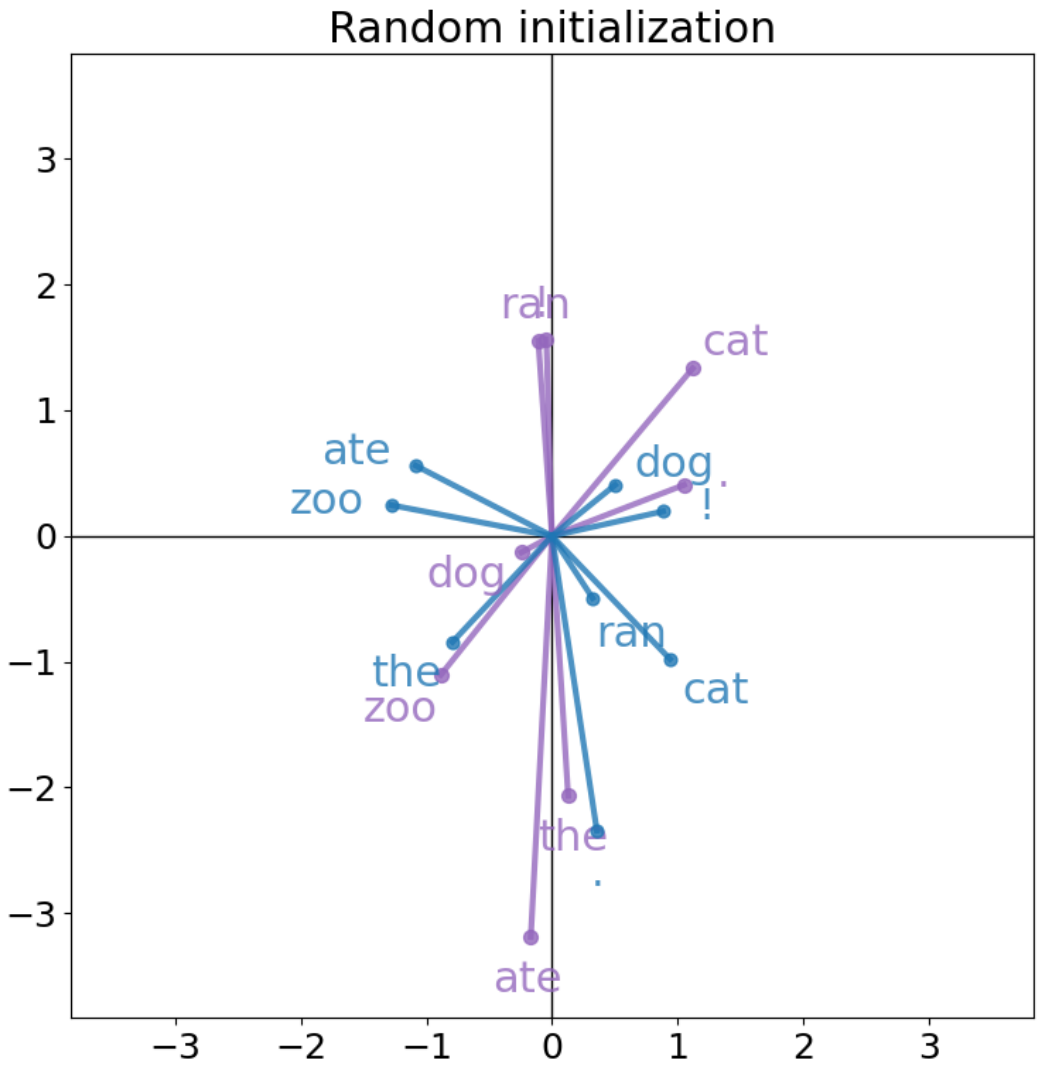
```
U:
!:     -0.044,   1.568
.:      1.051,   0.406
ate:   -0.169,  -3.190
cat:    1.120,   1.333
dog:   -0.243,  -0.130
ran:   -0.109,   1.556
the:    0.129,  -2.067
zoo:   -0.885,  -1.105
```

Simple Word Embedding LM

Vector representation for each token in vocabulary (initially random)

V: Previous

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

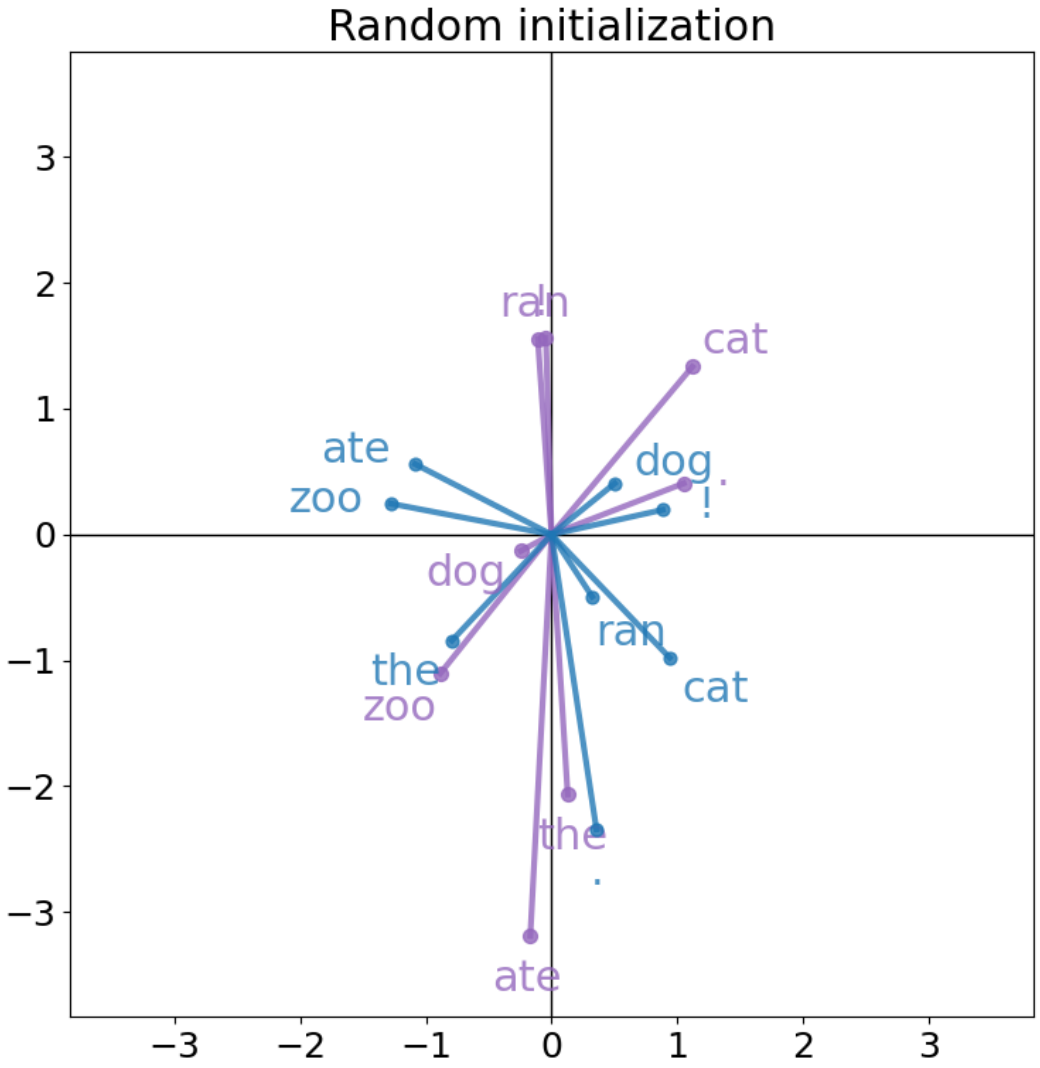


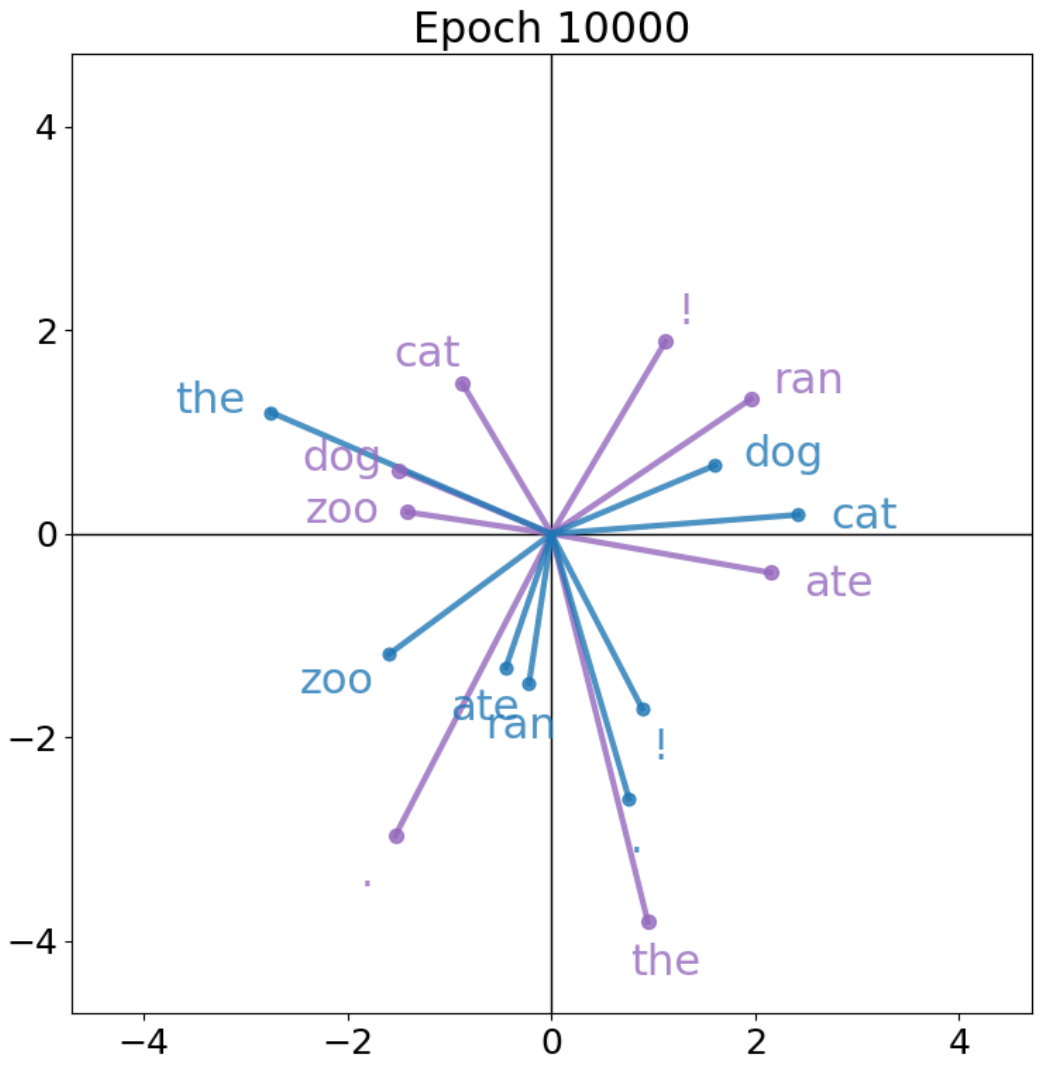
U: Next

U:		
!:	-0.044,	1.568
.:	1.051,	0.406
ate:	-0.169,	-3.190
cat:	1.120,	1.333
dog:	-0.243,	-0.130
ran:	-0.109,	1.556
the:	0.129,	-2.067
zoo:	-0.885,	-1.105

Simple Word Embedding LM

After training our LM, we'll learn more organized vectors (details later)

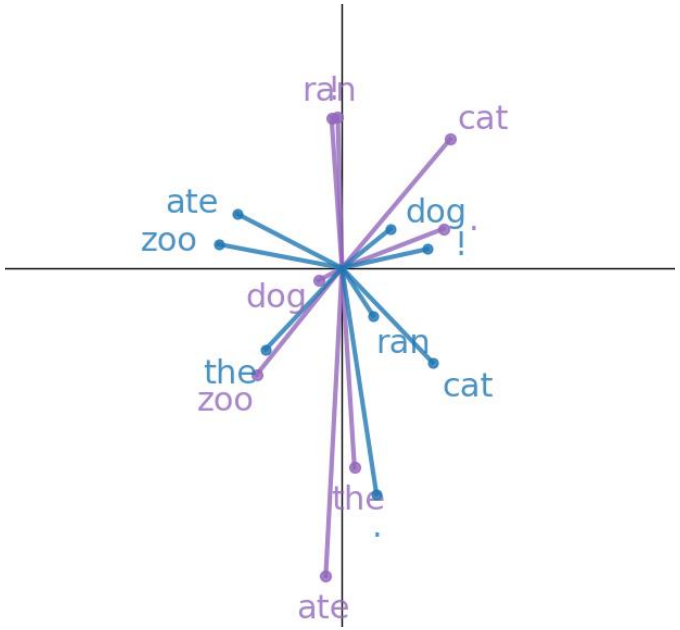




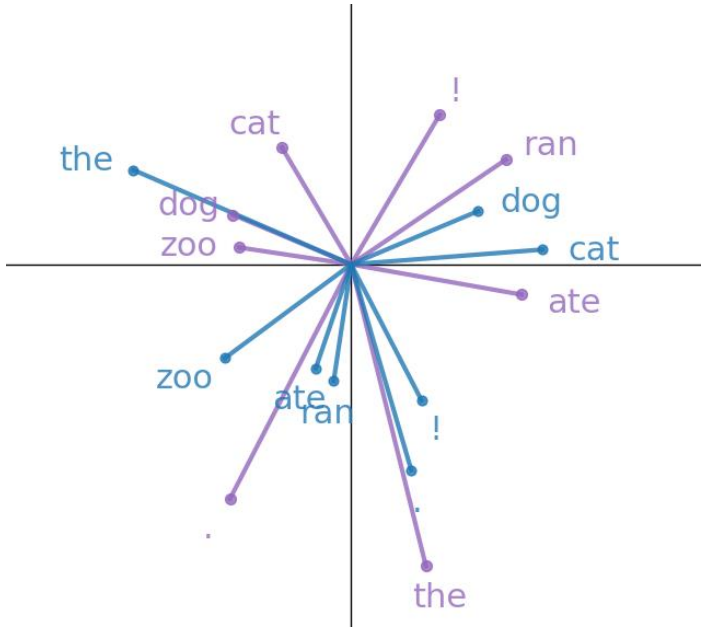
Simple Word Embedding LM

We can use either of these models to generate text, staring with "the dog"

Random vectors



Trained vectors



Generated tokens from random and trained models

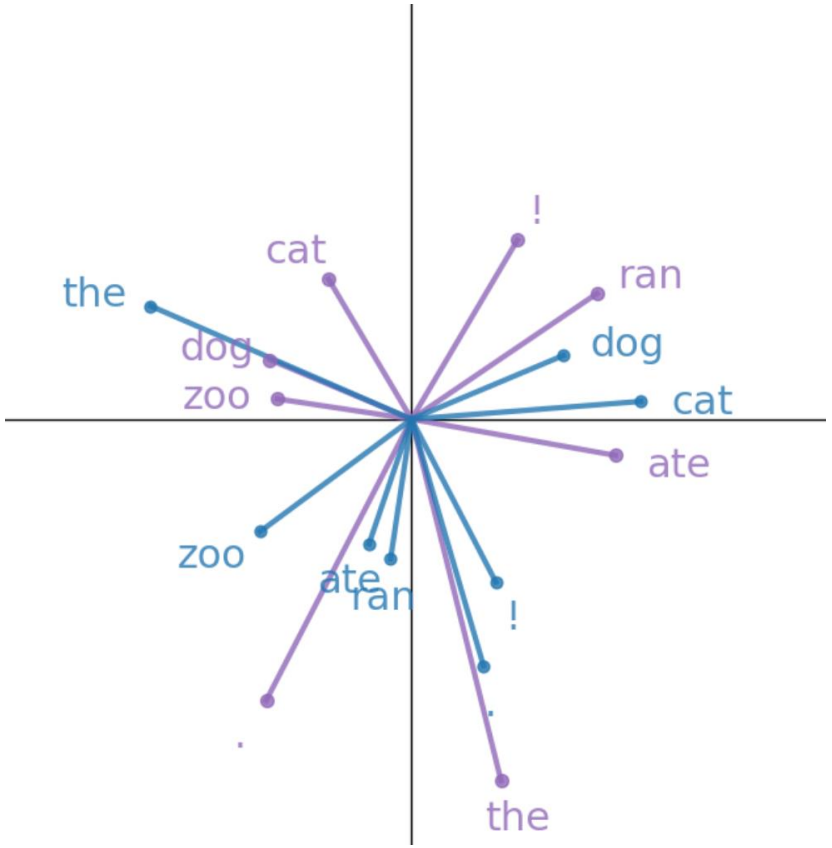
the dog cat ate ran zoo zoo
zoo dog dog cat cat ate ran .
ate . ate ate ! the ate

the dog ate . the dog ! the
zoo . the dog ran the zoo .
the dog ate the dog !

Outline: Word Embedding LM

Vector representation of vocab tokens

- Set of vectors for both previous and next token



Sampling next token

- Cosine similarity
- Softmax
- Sample from categorical distribution

Learning better vectors

- Cross-entropy loss
- SGD: looping through pairs of tokens in our corpus

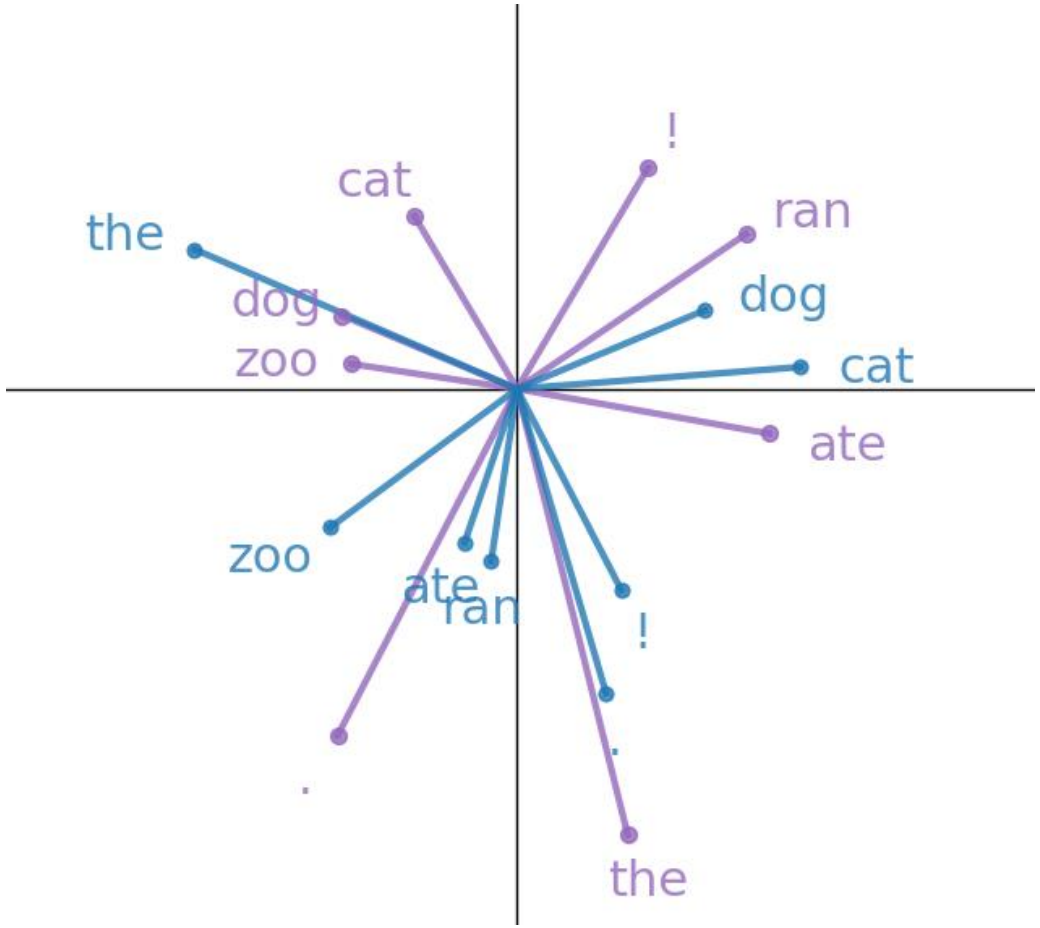
Sampling from Word Embeddings

Suppose we have a trained set of embedded vectors and we want to generate a the next token after the previous token 'the'.

V: previous tokens

U: next tokens

Which token should be our next token?



Sampling from Word Embeddings

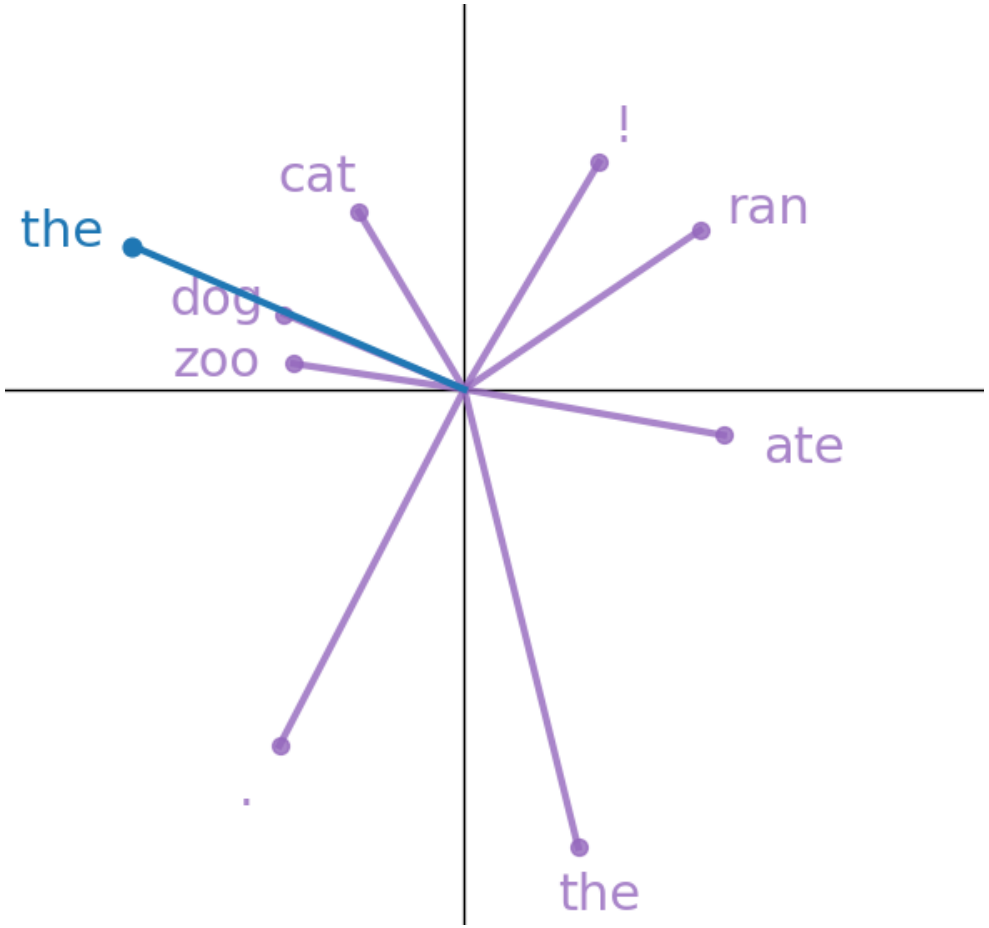
Suppose we have a trained set of embedded vectors and we want to generate a the next token after the previous token 'the'.

V: previous tokens

U: next tokens

Which token should be our next token?

1. Lookup the index i for the vocab token 'the'
2. Access the i-th row of V, $\mathbf{v}$
3. Compare $\mathbf{v}$ to all vectors in U



(Unnormalized) Cosine Similarity Metric

We've been using Euclidean distance

- $d(\mathbf{u}, \mathbf{v}) = \|\mathbf{u} - \mathbf{v}\|_2$

Cosine similarity

- Two vectors are similar if their dot product is positive and big
  - $f(\mathbf{u}, \mathbf{v}) = \mathbf{u}^T\mathbf{v}$
- (Why cosine?)
  - Two vectors are similar if the angle between them is small (small angle → large $\cos\theta$)
    - $f(\mathbf{u}, \mathbf{v}) = \mathbf{u}^T\mathbf{v} = \|\vec{u}\|_2 \|\vec{v}\|_2 \cos\theta$

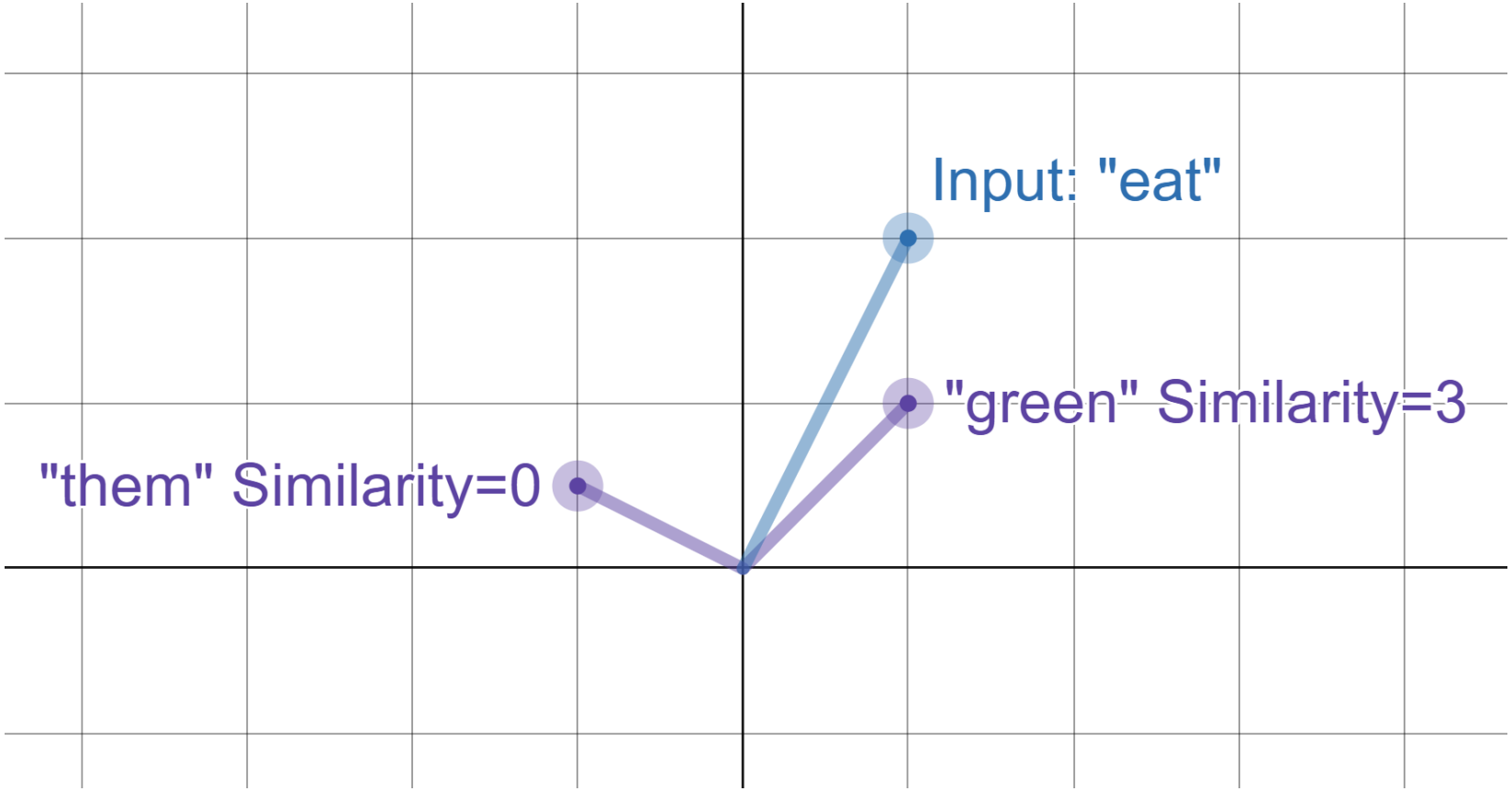
$$\cos\theta = \frac{\mathbf{u}^T\mathbf{v}}{\|\mathbf{u}\|_2\|\mathbf{v}\|_2}$$

(Unnormalized) Cosine Similarity Metric

Cosine similarity Desmos demo

https://www.desmos.com/calculator/82m4zkjlkc

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



Similarity Matrix

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

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

$f(\boldsymbol{V}) =$

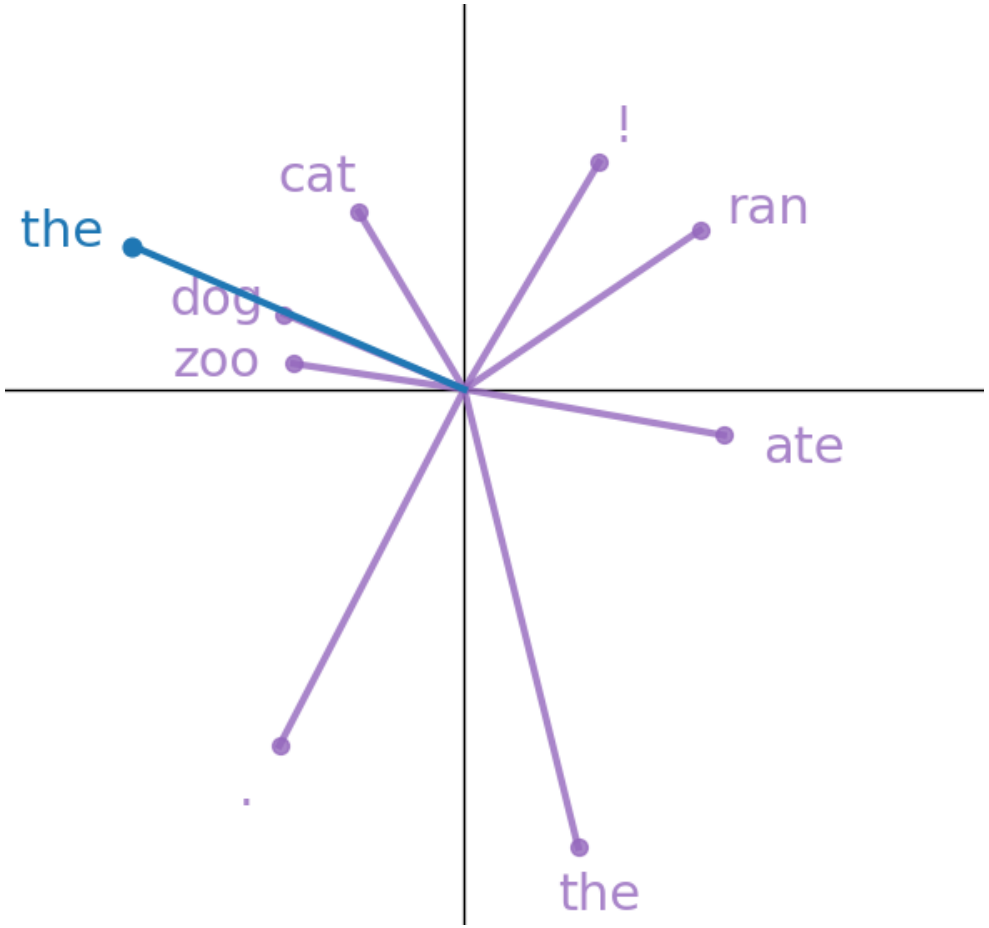
Sampling from Word Embeddings

Suppose we have a trained set of embedded vectors and we want to generate a the next token after the previous token 'the'.

V: previous tokens

U: next tokens

1. Compute similarity scores
   $\mathbf{s} = U\mathbf{v}$

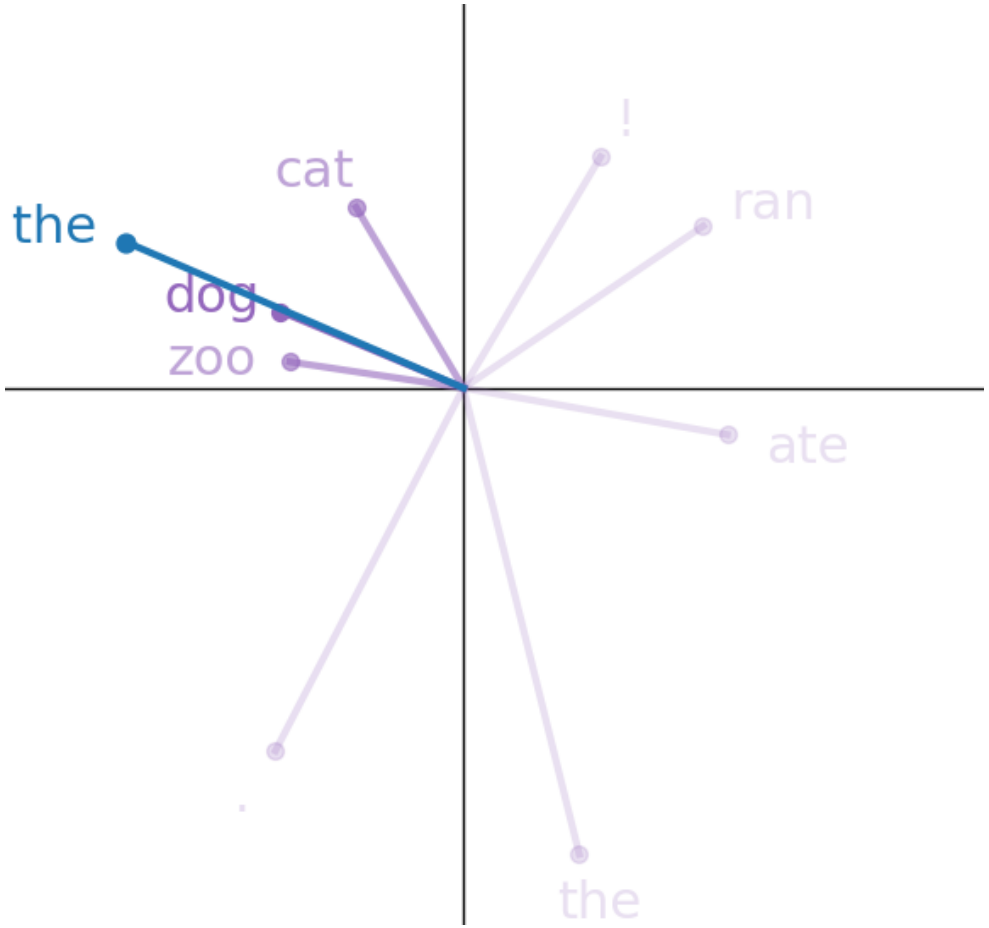


Sampling from Word Embeddings

Suppose we have a trained set of embedded vectors and we want to generate a the next token after the previous token 'the'.

V: previous tokens

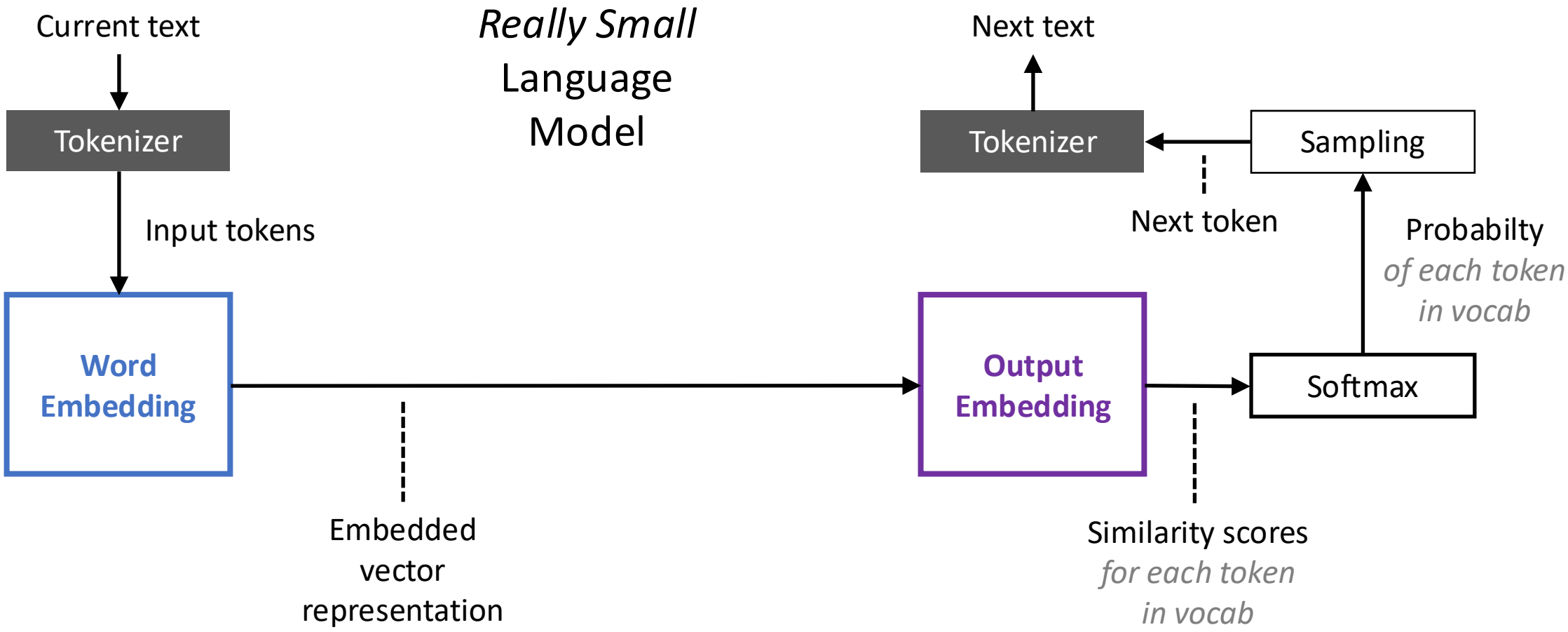
U: next tokens



1. Compute similarity scores $\mathbf{s} = U\mathbf{v}$
2. Convert to probabilities $\hat{\mathbf{y}} = g_{softmax}(\mathbf{s})$
3. Sample from Categorical distribution defined by $\hat{\mathbf{y}}$
4. LOOP: next token → prev

Simple Word Embedding LM

Building a language model with just word embedding layers ☺



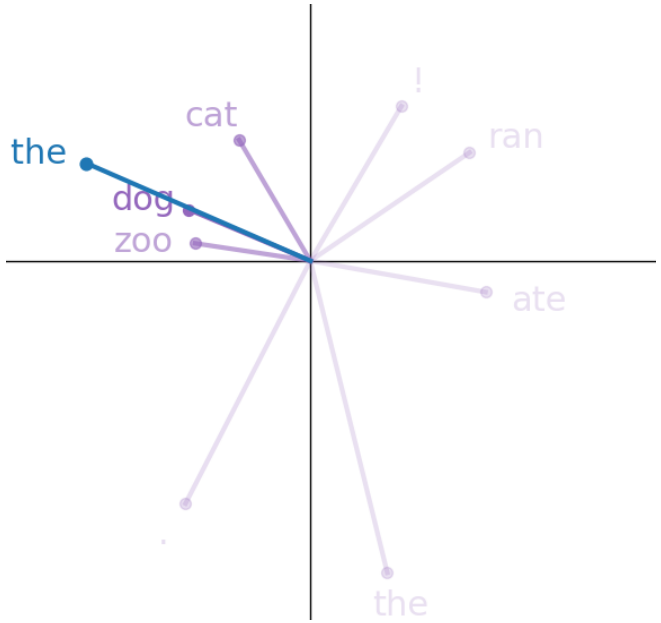
Poll

Given a vocab size of 10 and an embedded dimension of size 2 how many parameters do have in a word embedding language model?

V: previous tokens

U: next tokens

1. Compute similarity scores $\mathbf{s} = U\mathbf{v}$
2. Convert to probabilities $\hat{\mathbf{y}} = g_{softmax}(\mathbf{s})$
3. Sample from Categorical distribution defined by $\hat{\mathbf{y}}$
4. LOOP: next token → prev



PyTorch for Word Embedding LM

Two matrices of features vectors:

W_1: for context

W_2: for next token

Using PyTorch

```
WordEmbedLM(
  (encode): Linear(in_features=vocab_size, out_features=2)
  (decode): Linear(in_features=2, out_features=vocab_size)
)
```

```
F.softmax(model.forward(x_onehot))
```

PyTorch for Word Embedding LM

Two matrices of features vectors:

W_1: for context

W_2: for next token

Using PyTorch

`torch.nn.Embedding(num_embeddings, embedding_dim)`

```
WordEmbedLM(
  (encode): Linear(in_features=vocab_size, out_features=2)
  (decode): Linear(in_features=2, out_features=vocab_size)
)
```

`x_index`

```
F.softmax(model.forward(x_onehot))
```

Learning Better Vectors

(Simple) Tokenized Corpus:

```
[ 'the', 'dog', 'ran', '.', 'the',
  'dog', 'ate', '.', 'the', 'dog',
  'ran', 'the', 'zoo', '.', 'the',
  'cat', 'ate', 'the', 'dog', '!',
  'the', 'cat', 'ran', 'the',
  'zoo', '.' ]
```

Classic ML recipe:

1. Training data

2. Hypothesis function

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

3. Formulate objective

   Loss:

   Objective: $\frac{1}{\mathrm{N}}\sum_i^N J^{(i)}(U,V)$ $J^{(i)}(U,V)$

4. SGD to find parameters that optimize the objective

Feature Learning
Image and Audio

Feature Learning

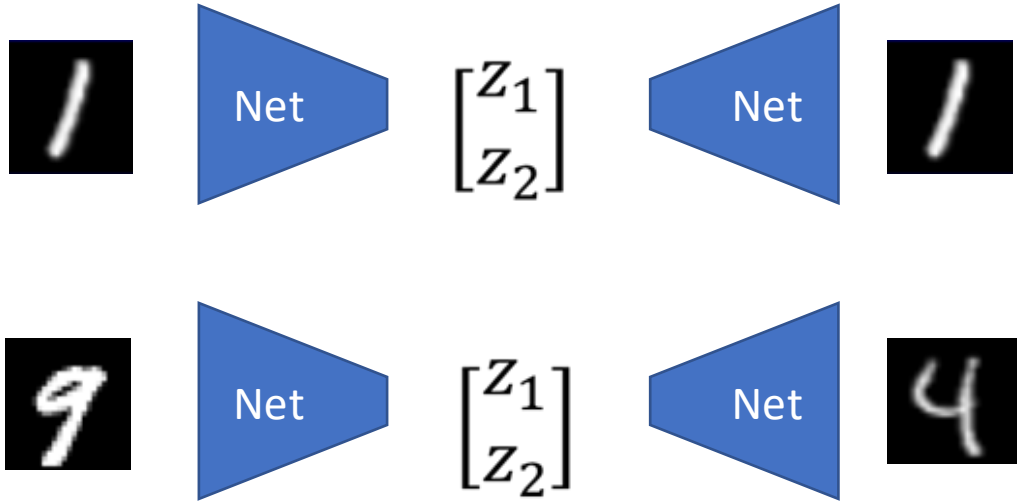
Learning a lower dimensional representation of our data rather than doing feature engineering to represent the data

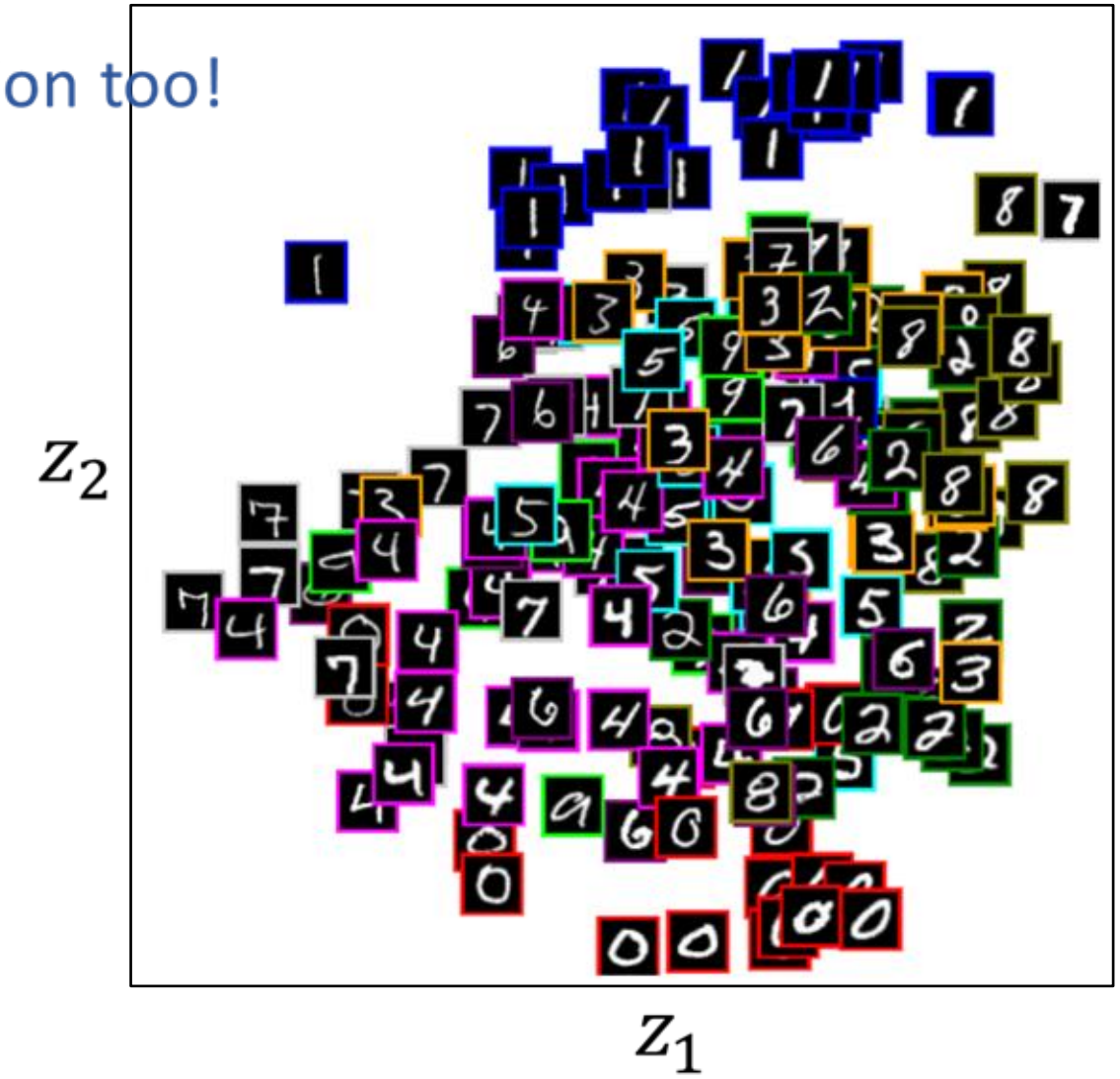
Also called feature embedding

(embedding data in a lower/different dimensional space)

Learning to Organize Data

Neural networks can learn to organization too!

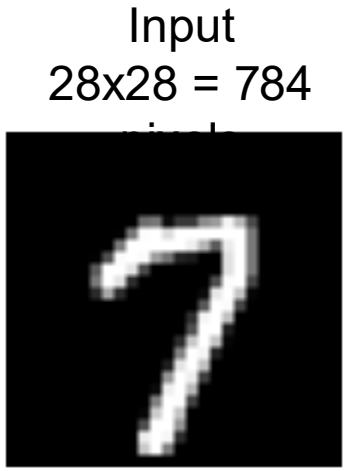


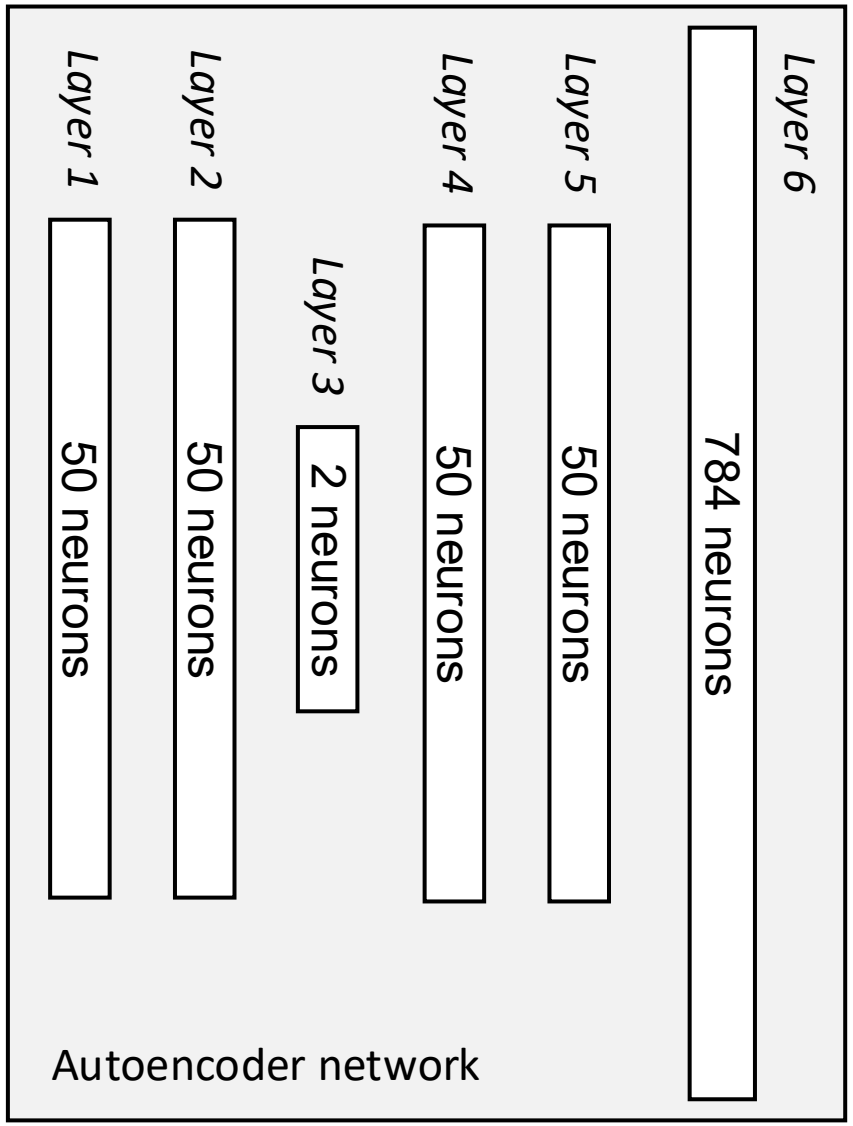


https://cs.stanford.edu/people/karpathy/convnetjs/demo/autoencoder.html

Digit Autoencoder

https://cs.stanford.edu/people/karpathy/convnetjs/demo/autoencoder.html

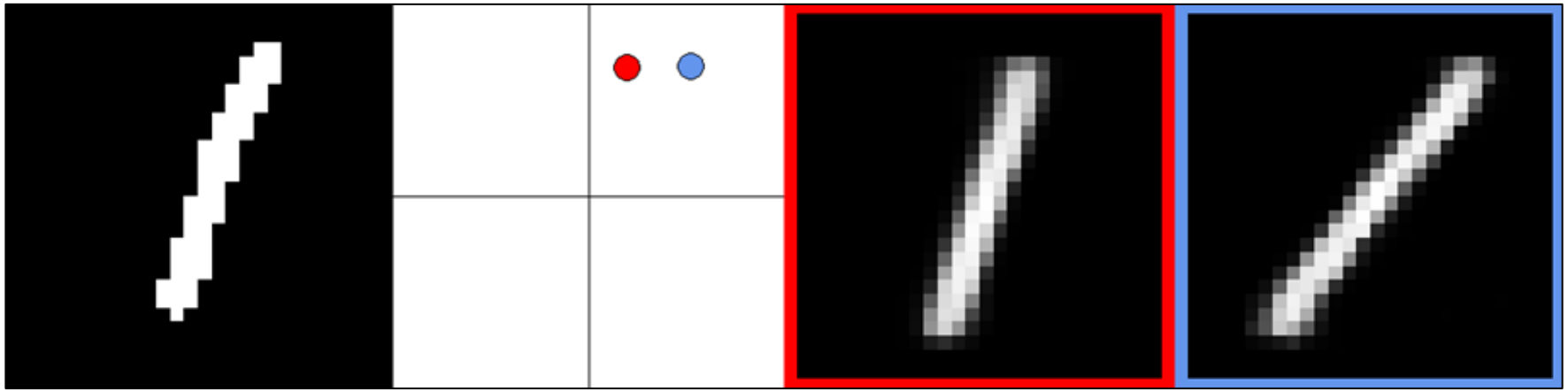






Digit Autoencoder

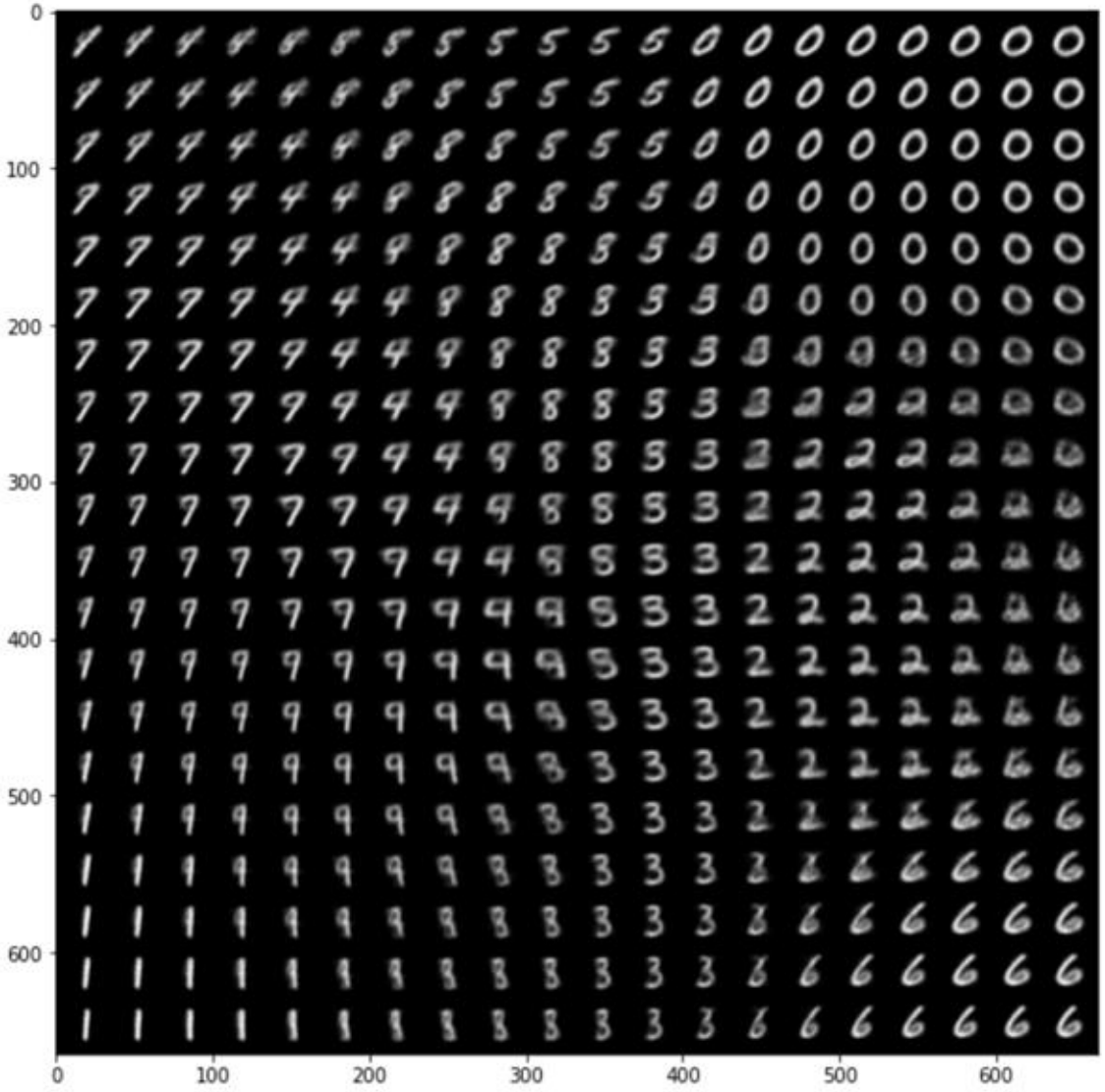
Demo: Using a learned feature space



Variational Autoencoder Demo

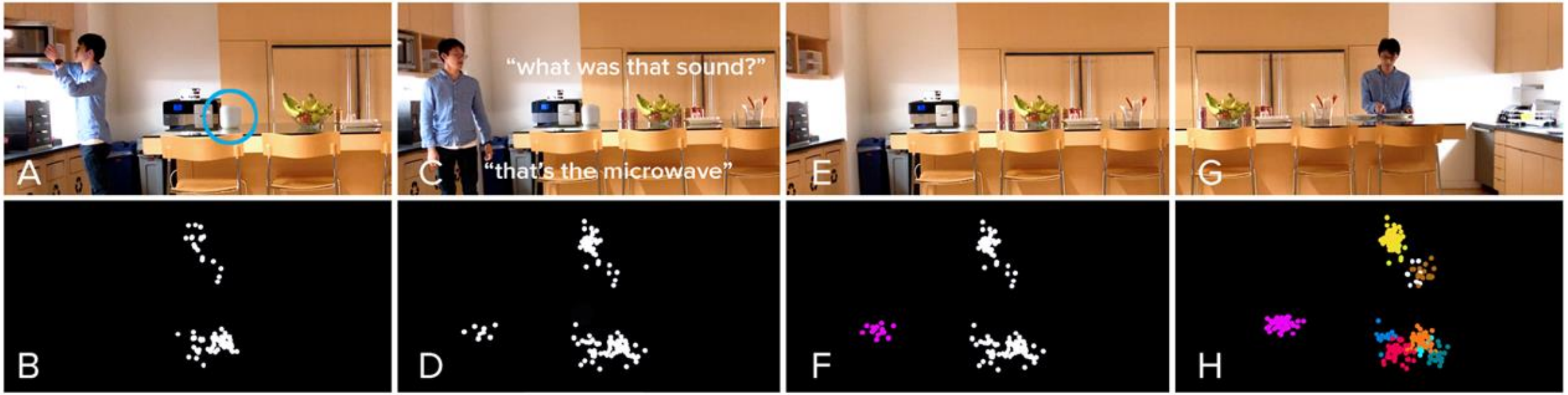
Zhuoyue Lyu, Safinah Ali, and Cynthia Breazeal. EAAI 2022.

https://colab.research.google.com/gist/ZhuoyueLyu/5046225a9ae3675cf633c1df5f63be06/digits-interpolation-notebook-eaai.ipynb



Feature Learning

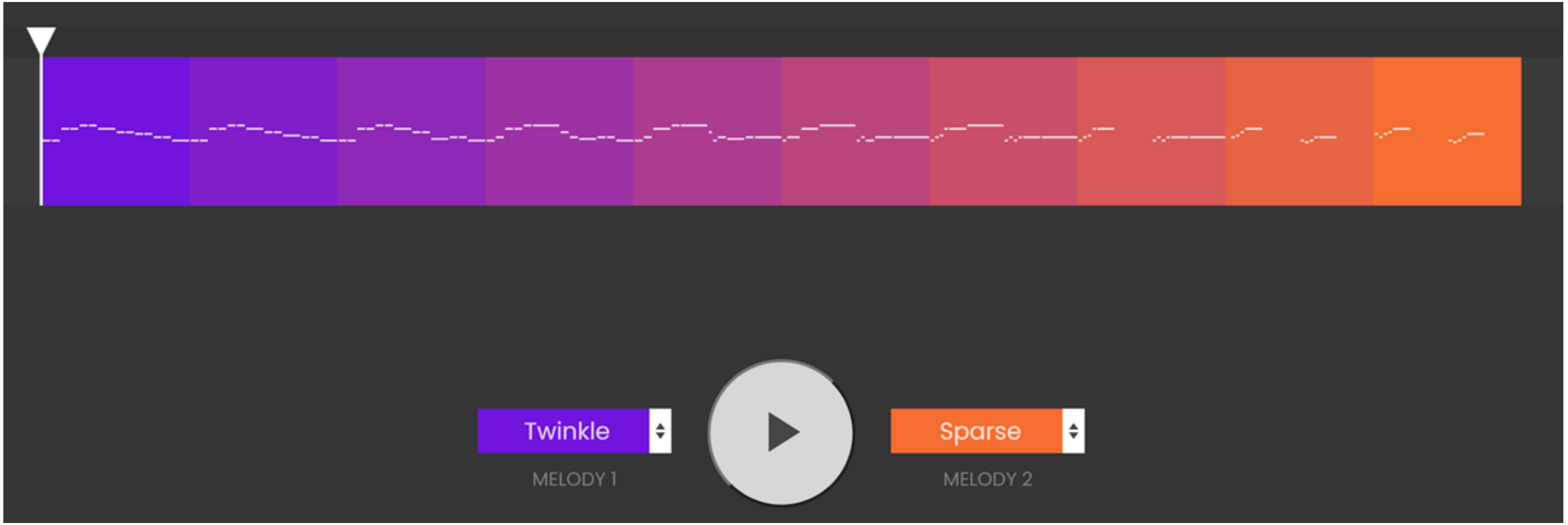
Listen Learner



https://chrisharrison.net/index.php/Research/ListenLearner

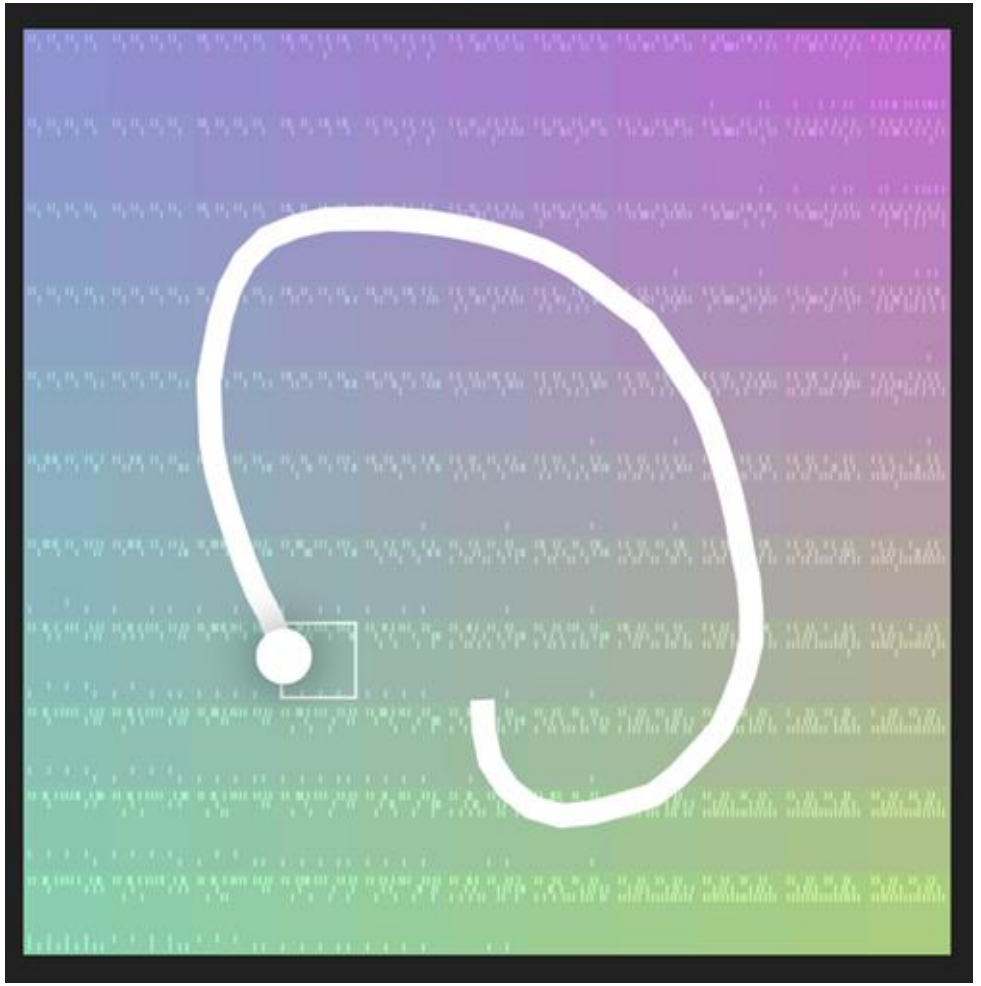
Exploring Feature Space

https://experiments.withgoogle.com/ai/melody-mixer/view/



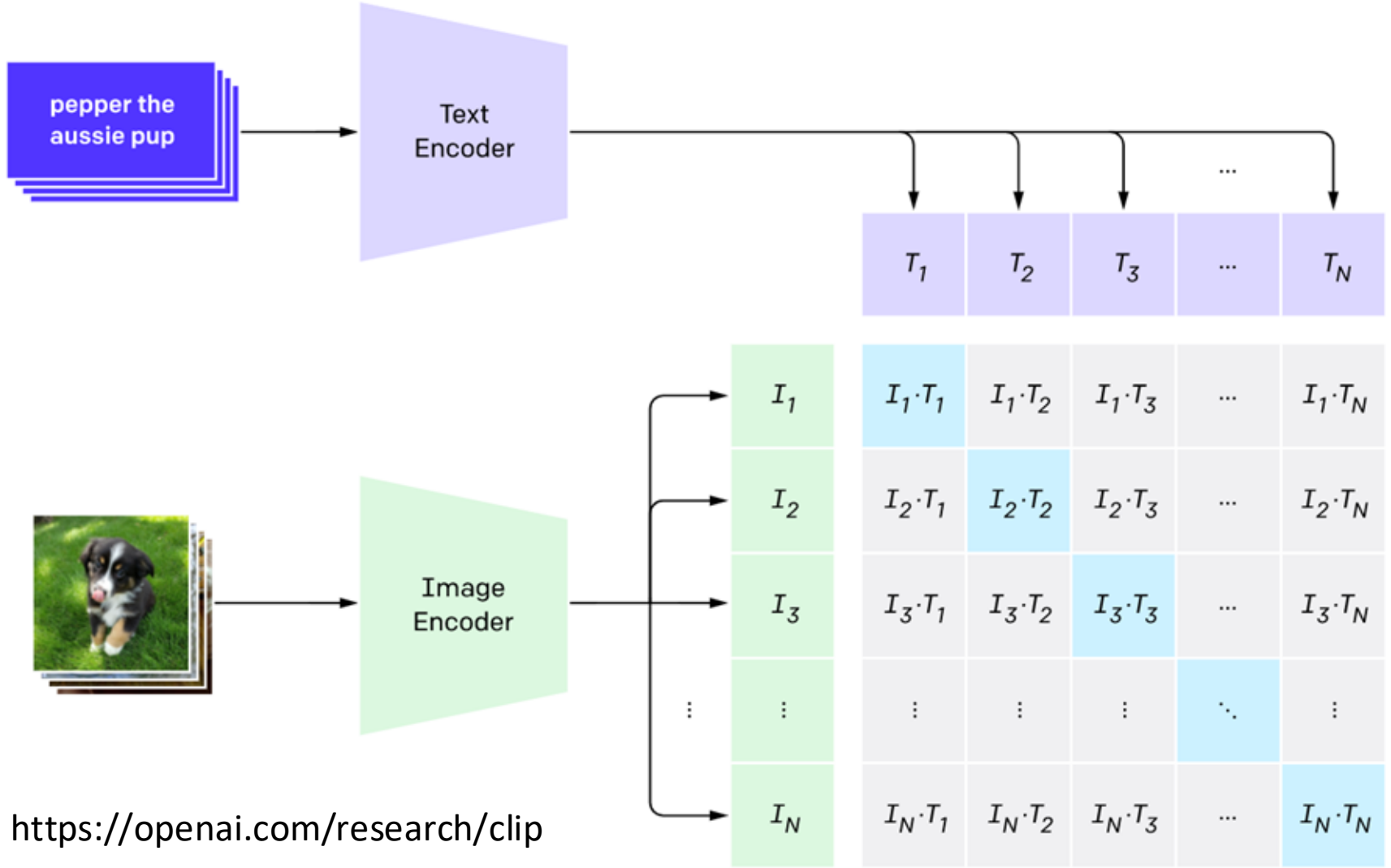
Exploring Feature Space

https://experiments.withgoogle.com/ai/beat-blender/view/



Feature Learning

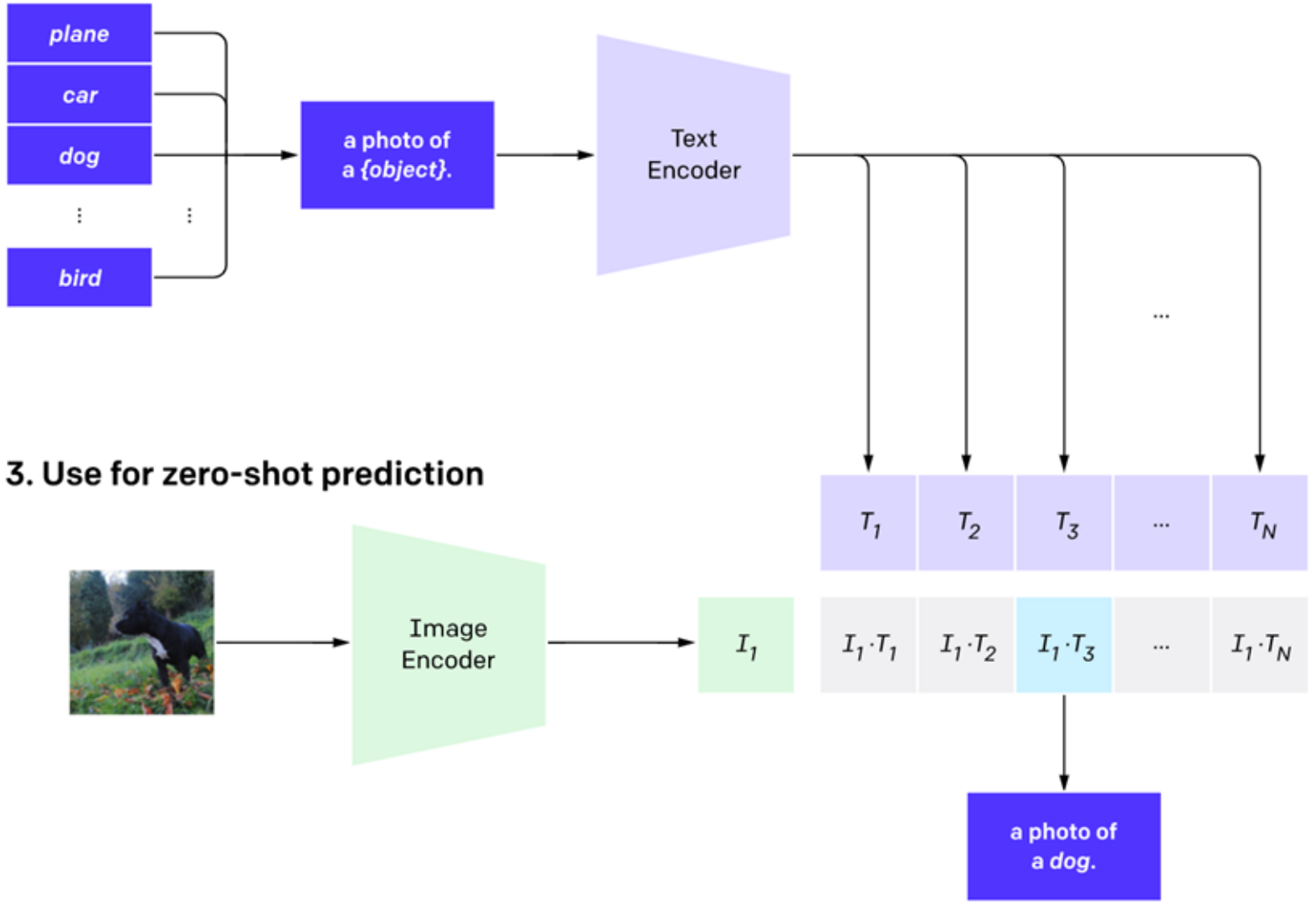
CLIP: Connecting text and images



https://openai.com/research/clip

Feature Learning

CLIP: Connecting text and images



https://openai.com/research/clip

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

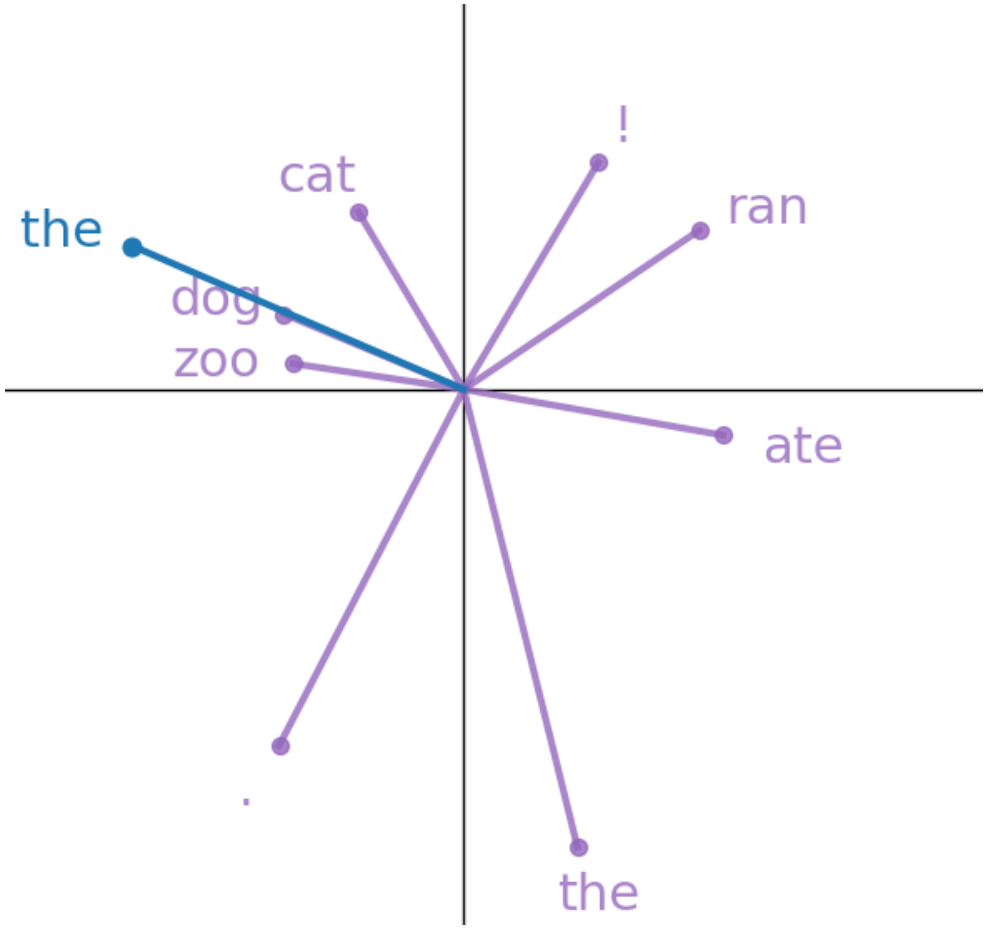
Transformer blocks

Increasing Context Size

What if we want to have more input tokens?

V: for context

U: for next token

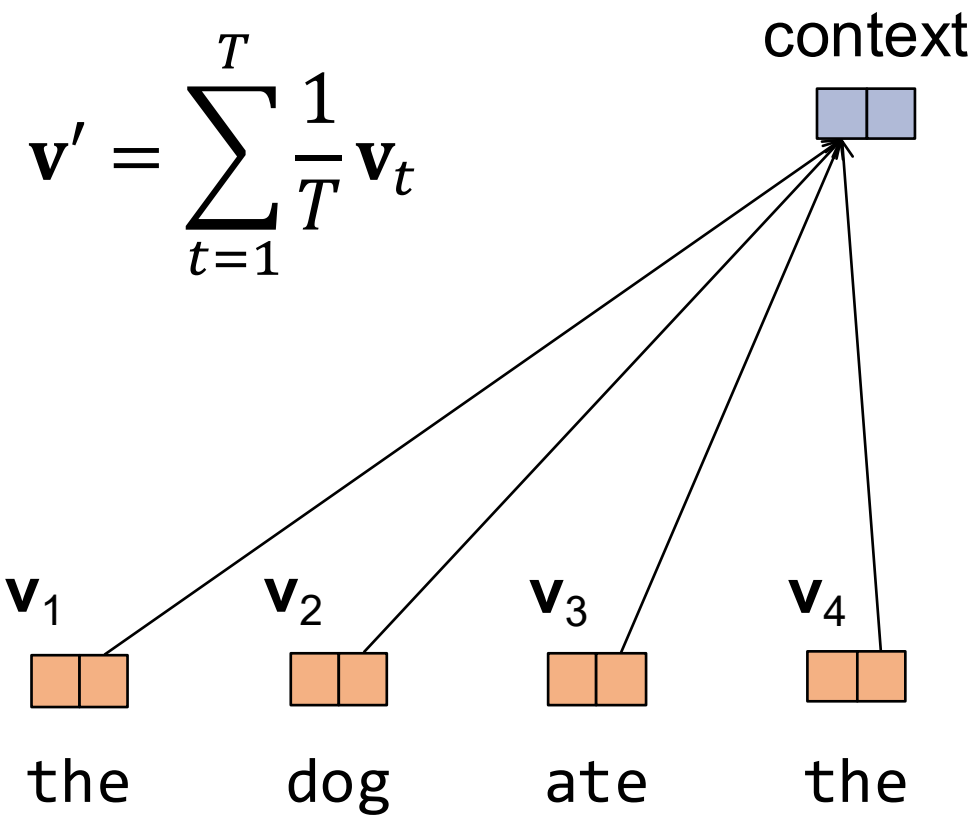


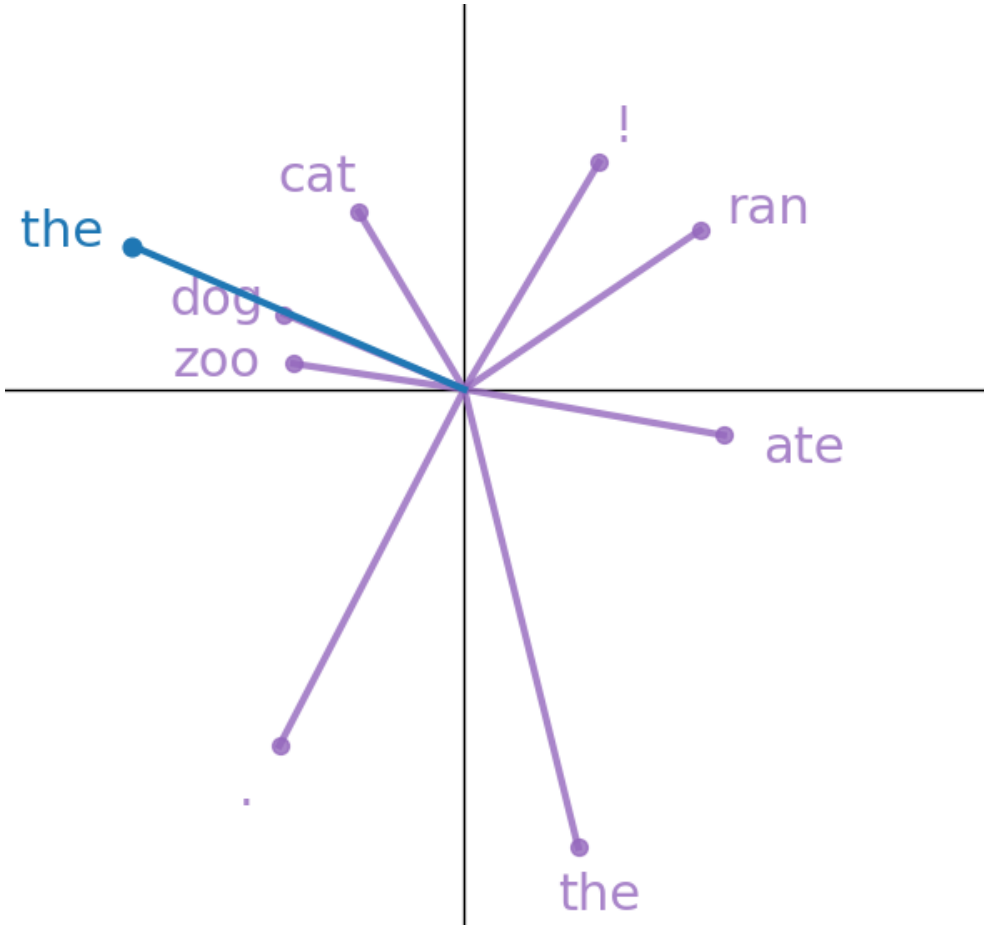
Increasing Context Size

What if we want to have more input tokens?

Uniform average over T input context tokens

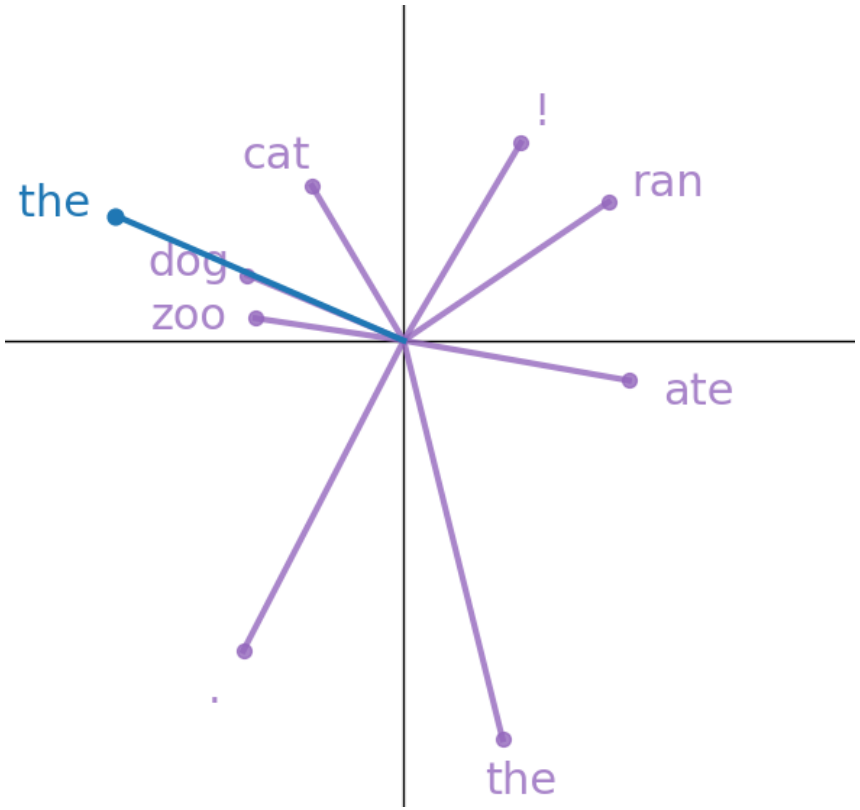
$$\mathbf{v}' = \sum_{t=1}^{T} \frac{1}{T} \mathbf{v}_t$$

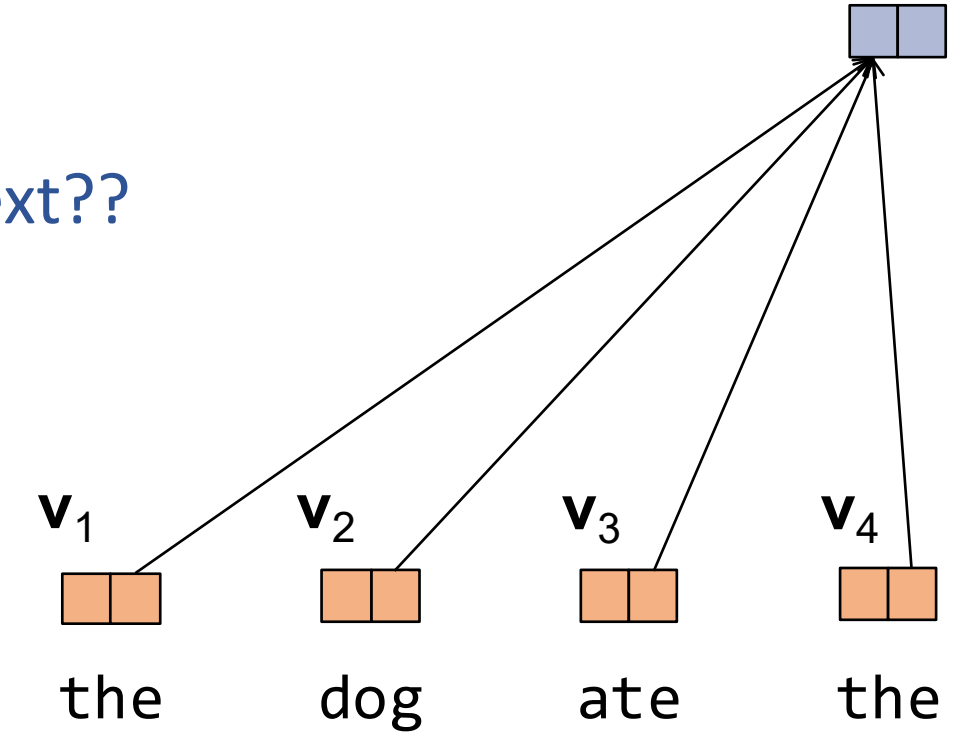




Position Encoding

What about position within the input context??





Position Encoding

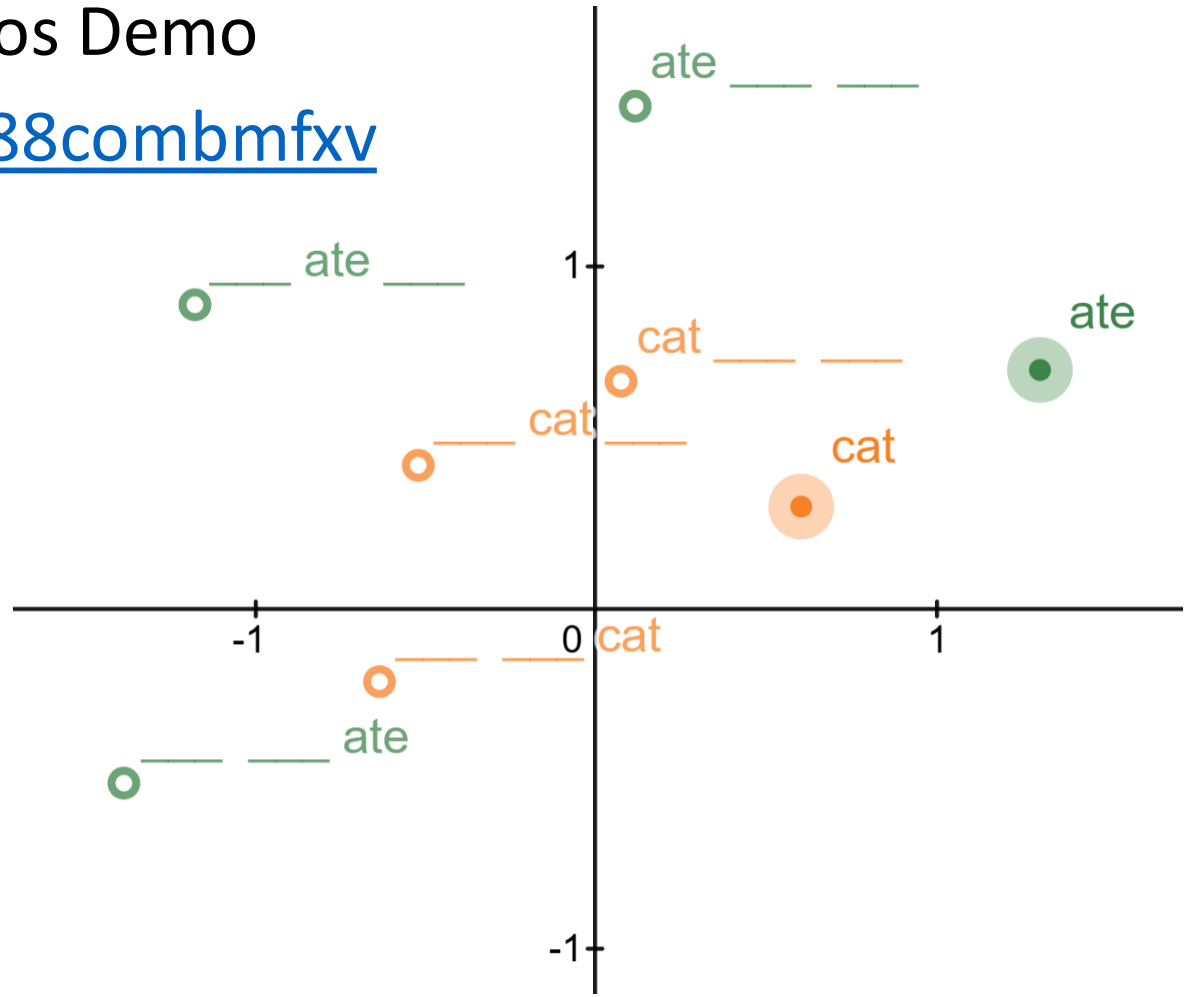
What about position within the input context??

Rotary Position Encoding

What about position within the input context??

Rotary Position Encoding (RoPE) Desmos Demo

https://www.desmos.com/calculator/88combmfxv



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 $\theta = i_{pos}\theta_1$:

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

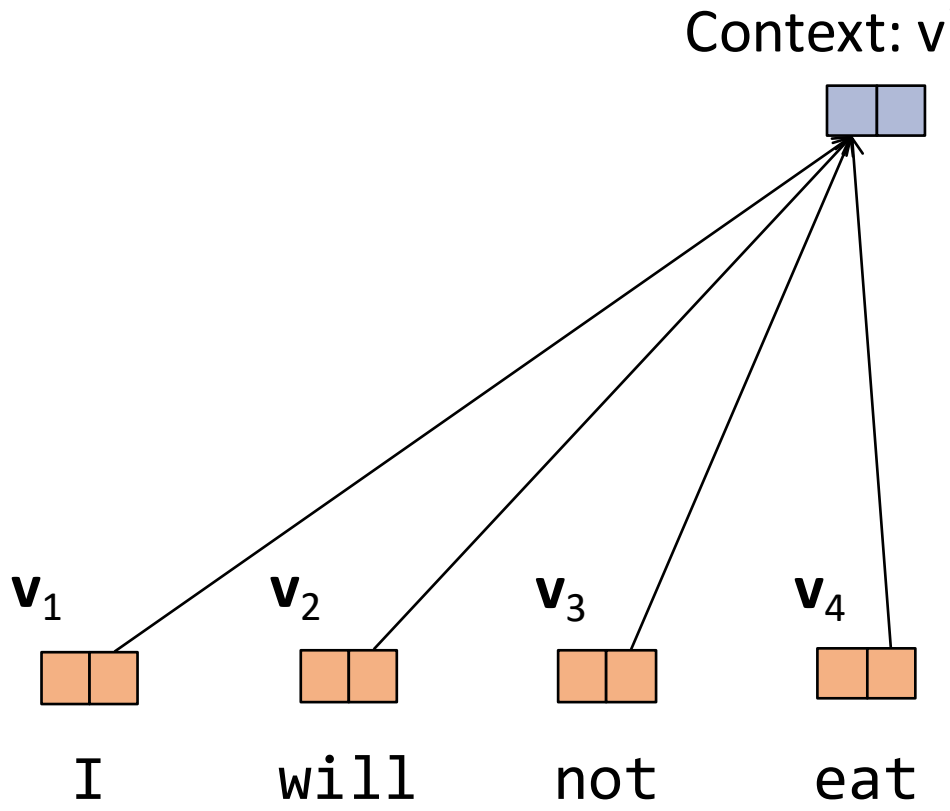
Attention

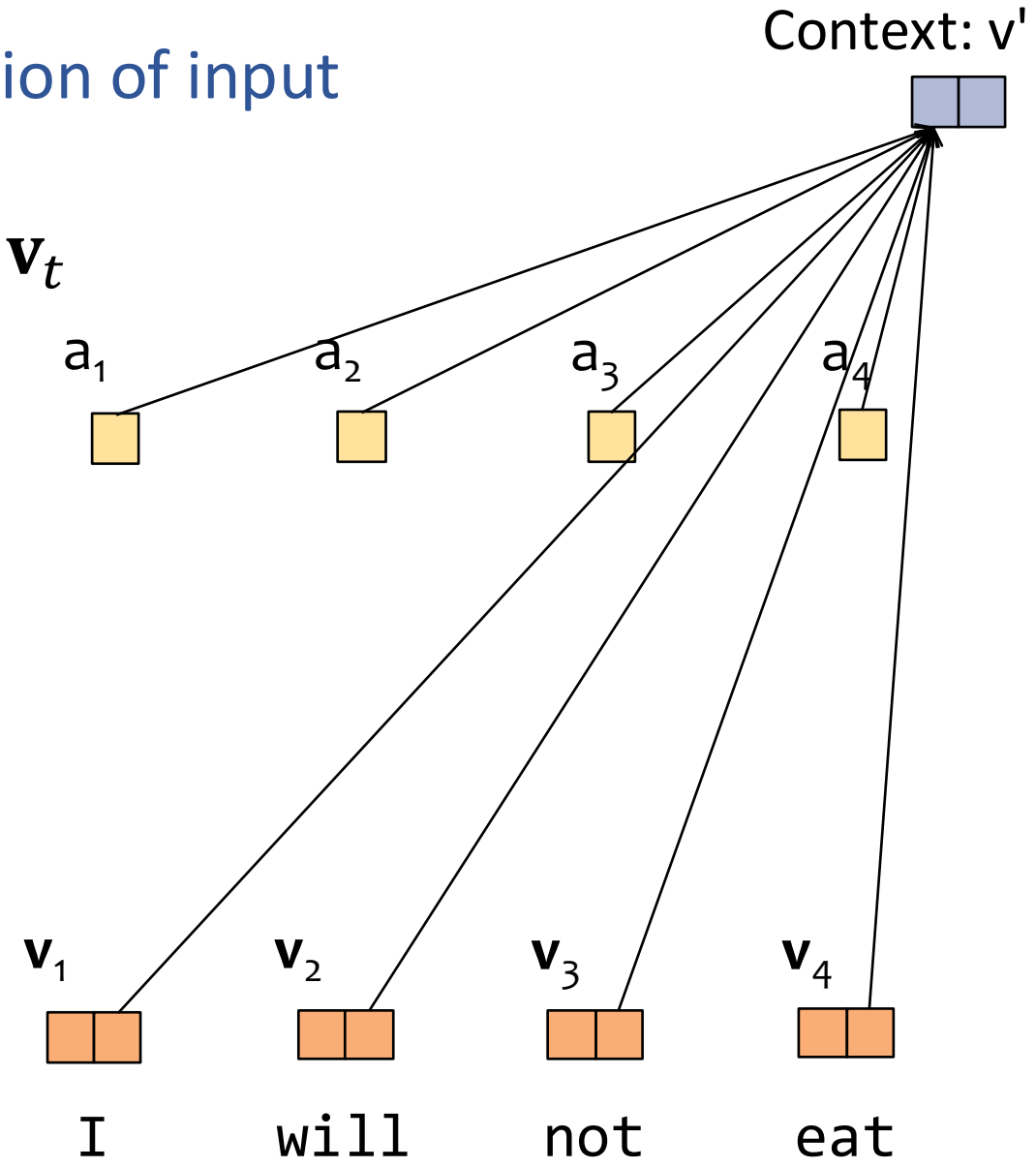
Learn to pay attention!

We can do better than uniform combination of input

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

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





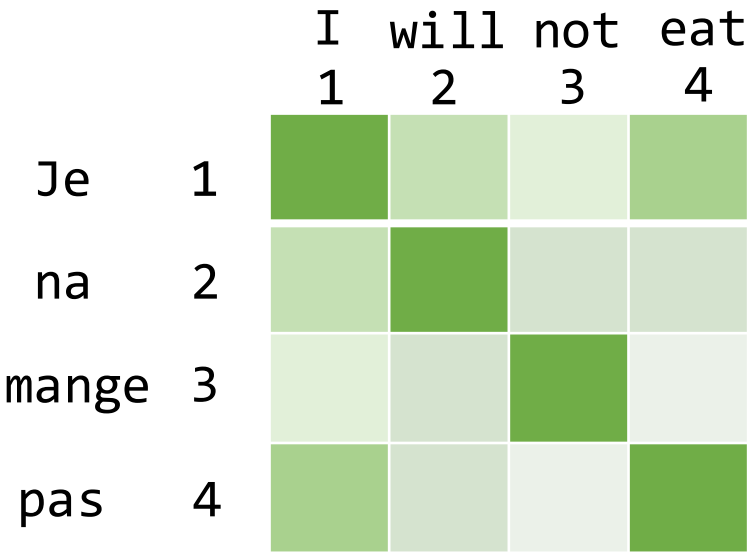
Learn to pay attention!

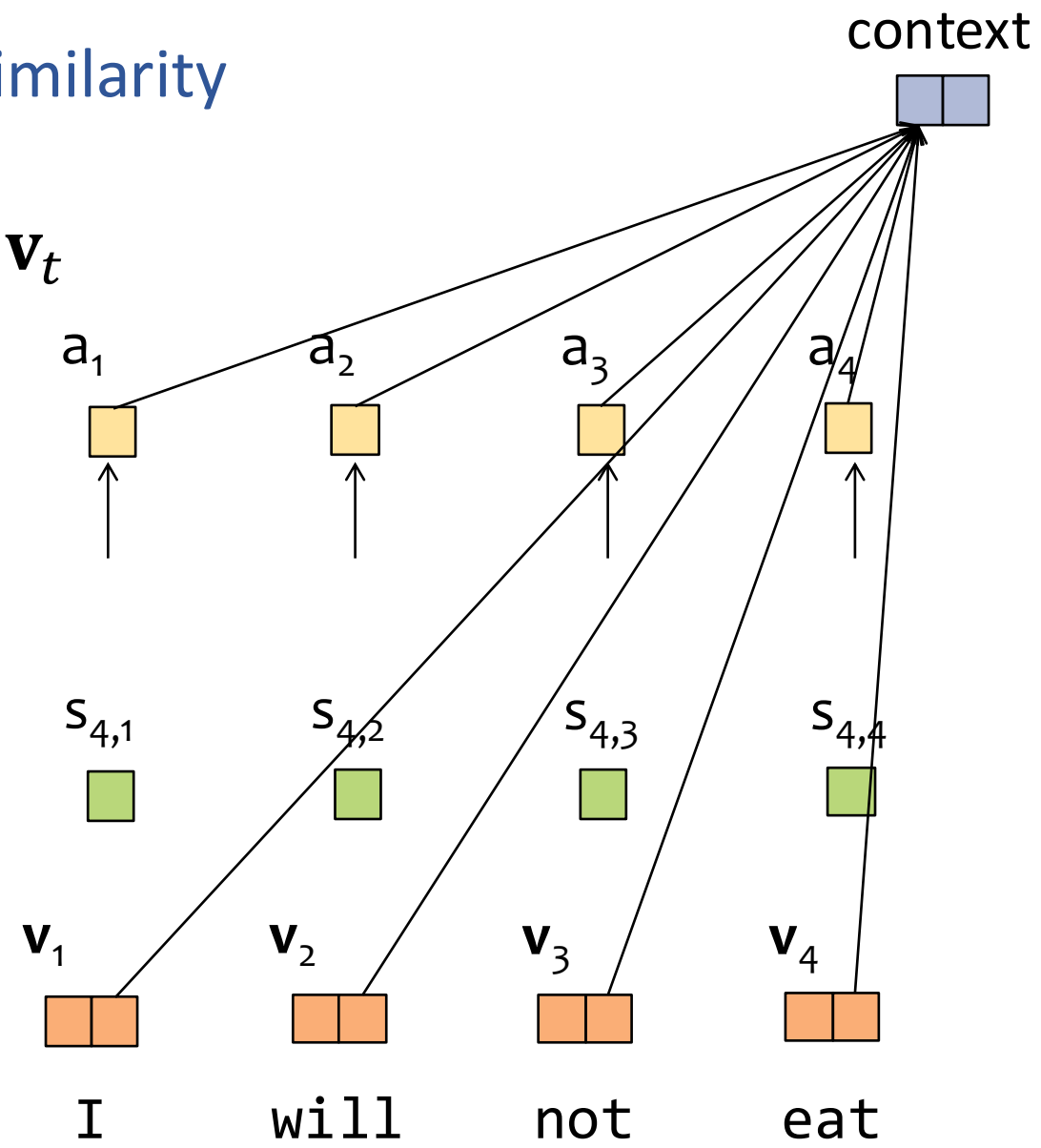
If only we had a way to measure vector similarity

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

Cosine similarity matrix!

$$S = VV^T$$





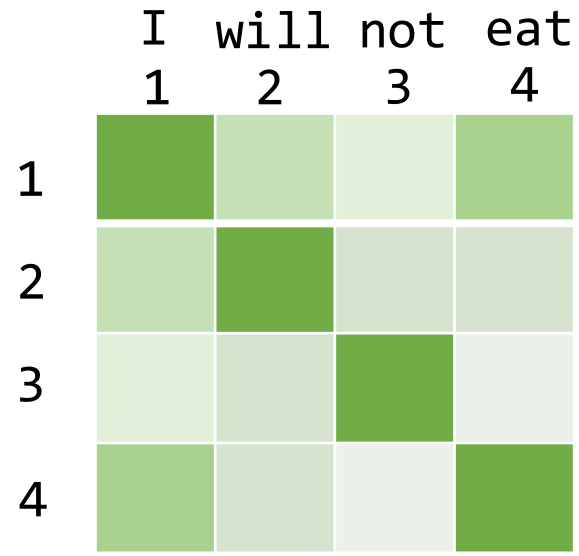
Learn to pay attention!

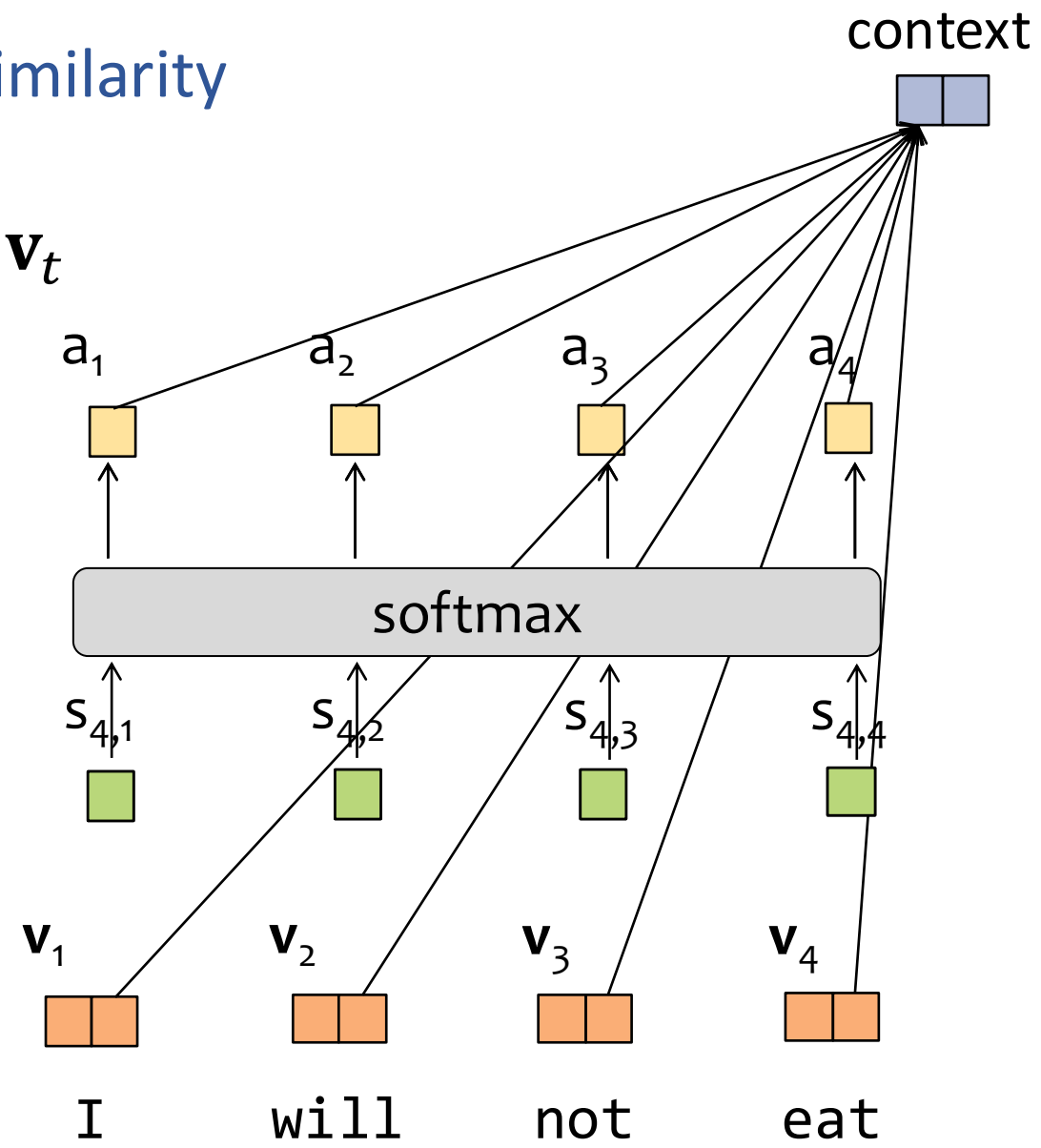
If only we had a way to measure vector similarity

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

Cosine similarity matrix!

$$S = VV^T$$



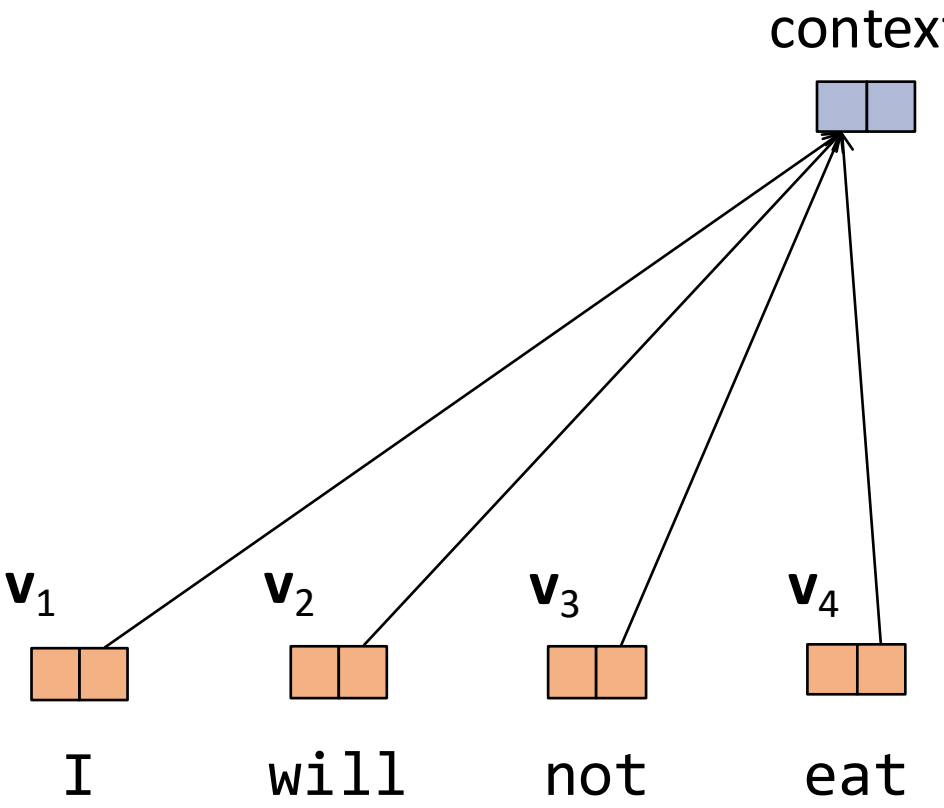


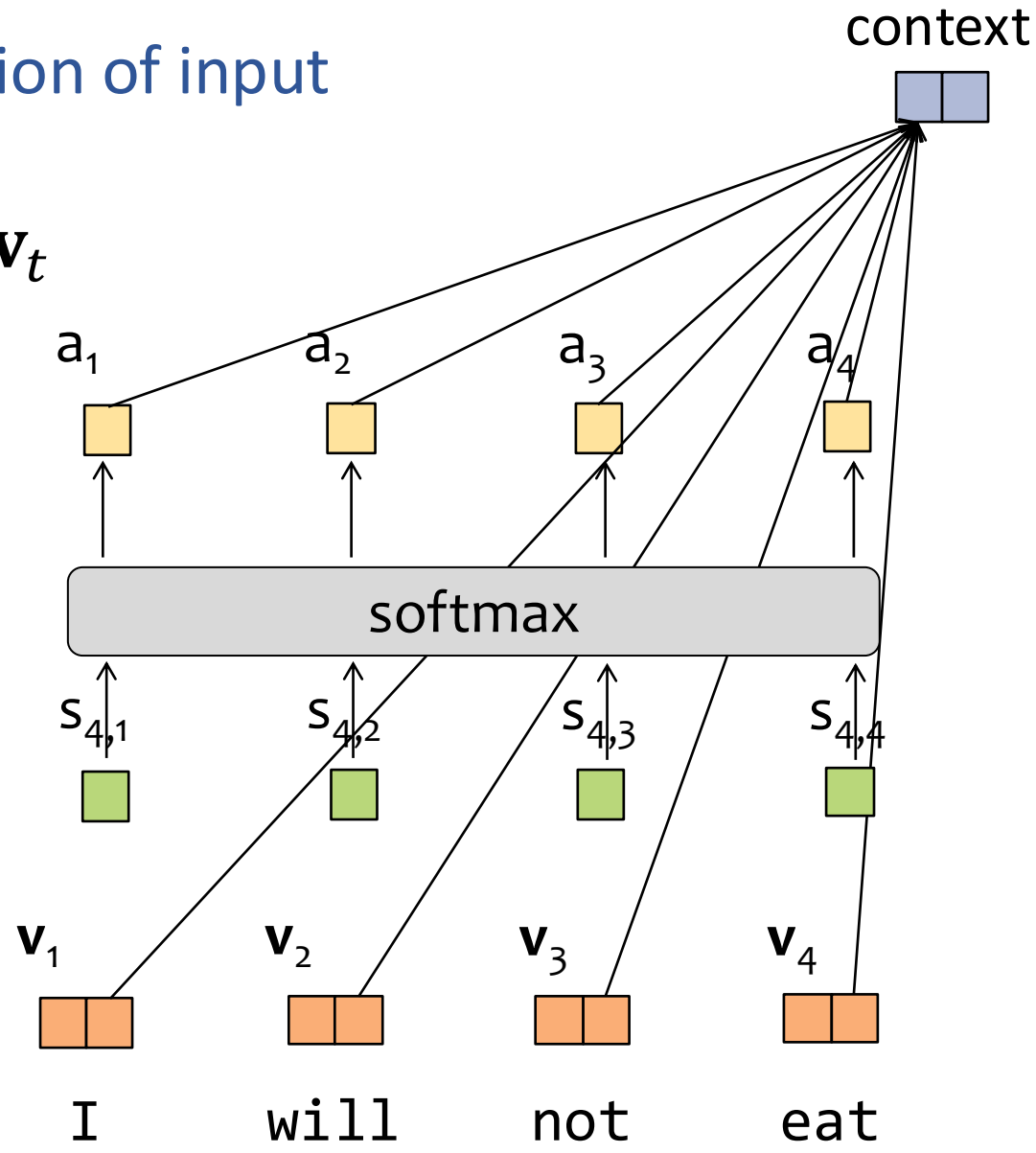
Learn to pay attention!

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!

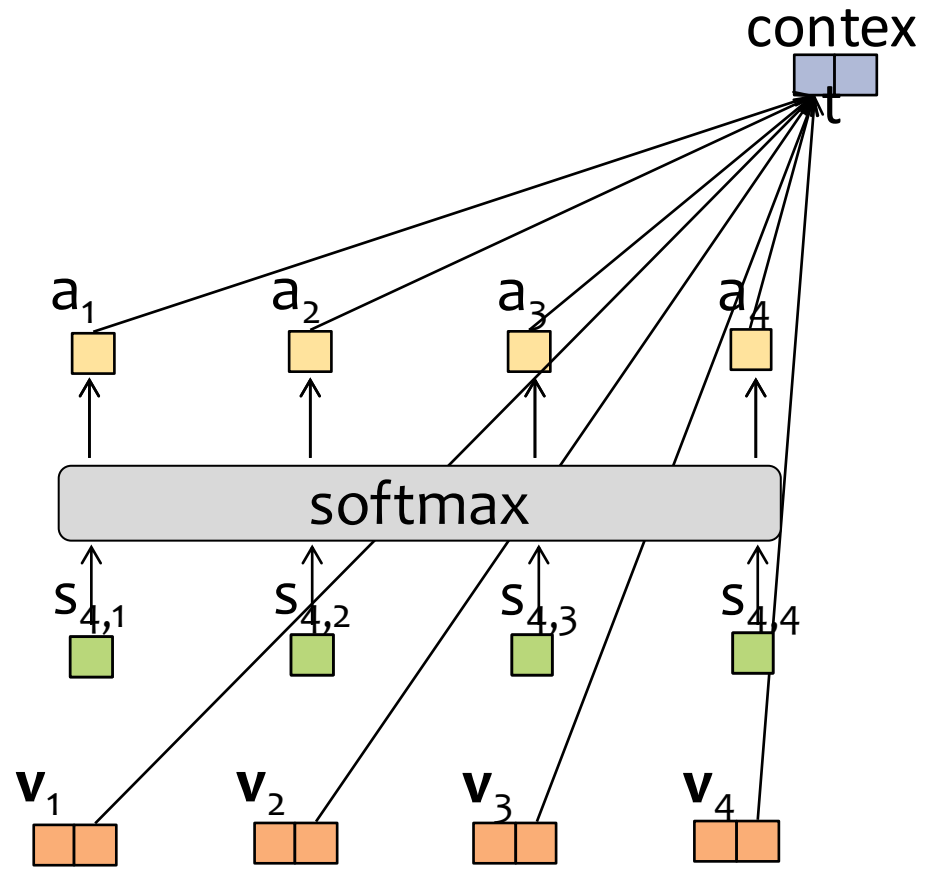
But...there is an issue with just doing VV^T ☹

We're really just comparing input to input

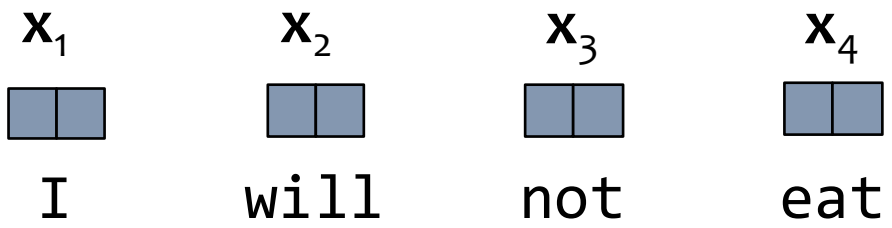
→ Symmetric with strong diagonal ☹

$$S = VV^T$$





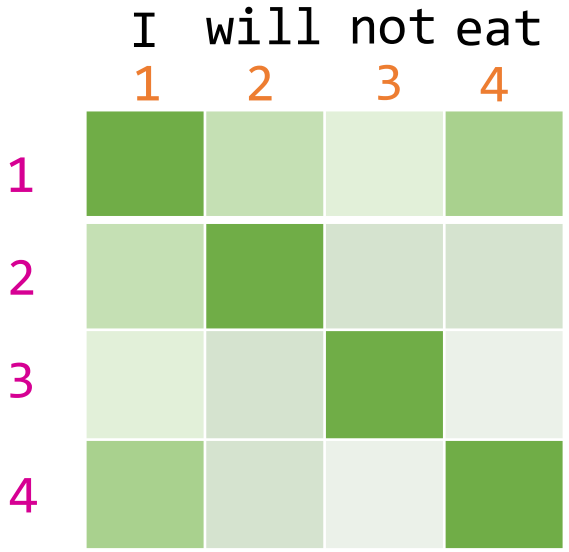
$$V = XW_V$$

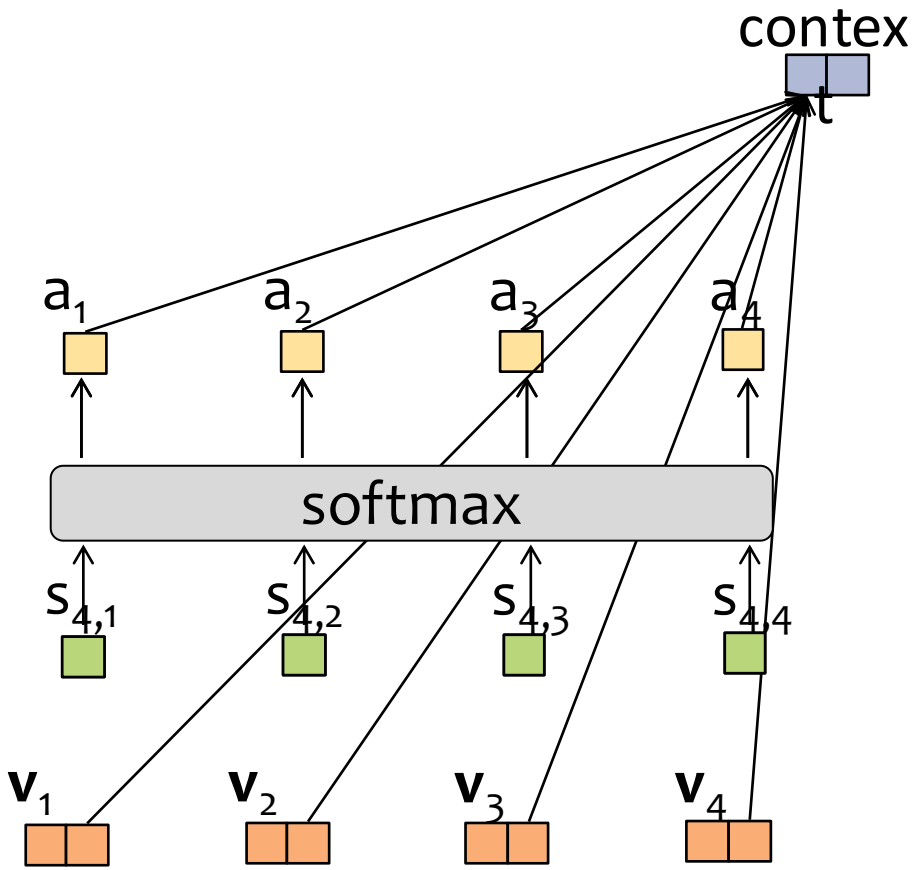


Learn to pay attention!

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

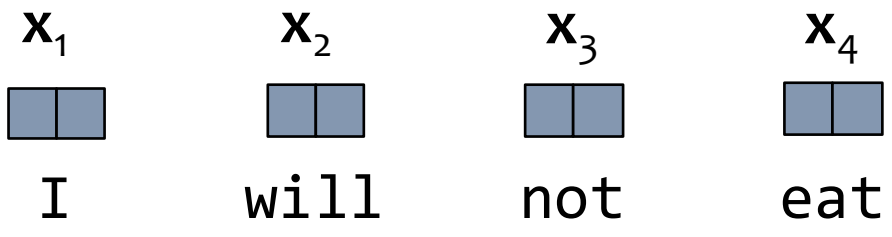
$$S = QV^T$$





$$Q = XW_Q$$

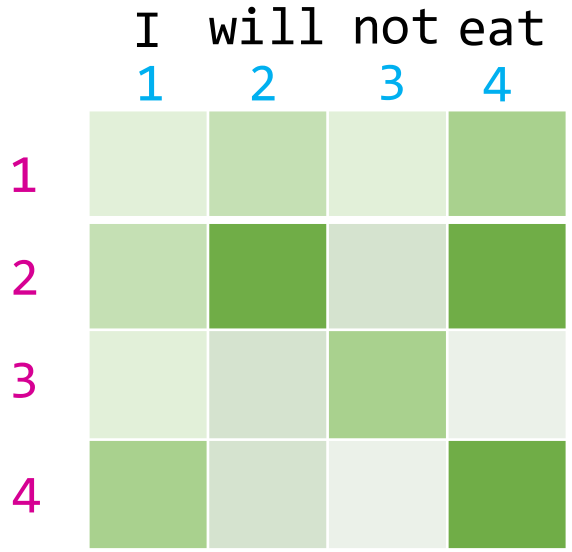
$$V = XW_V$$

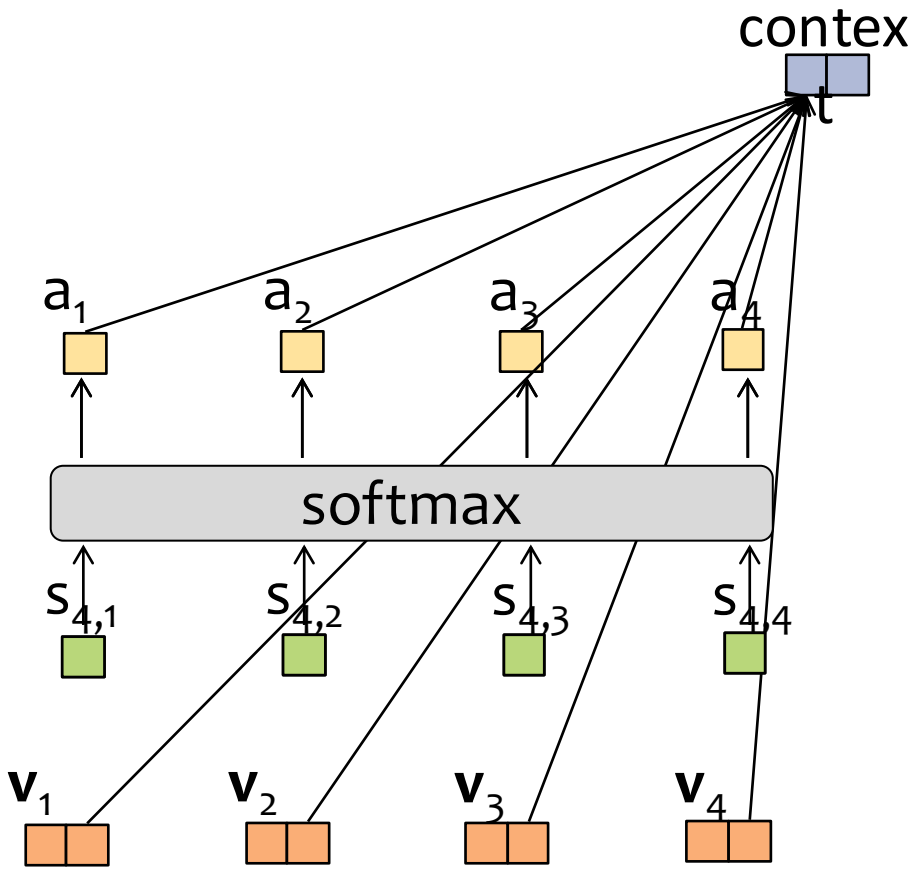


Learn to pay attention!

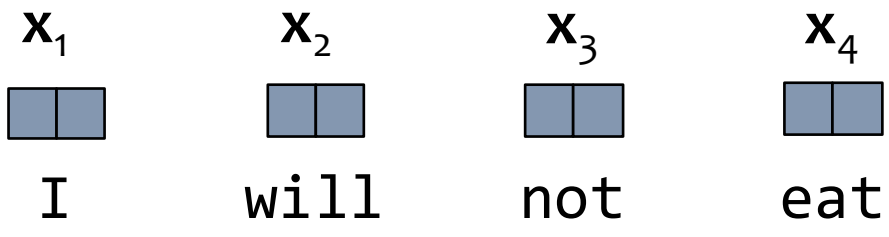
Instead learn a query vectors $\mathbf{q}_t$ to represent the output

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





$$Q = XW_Q \qquad K = XW_K \qquad V = XW_V$$



Learn to pay attention!

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

Attention:

Query, Key, Value

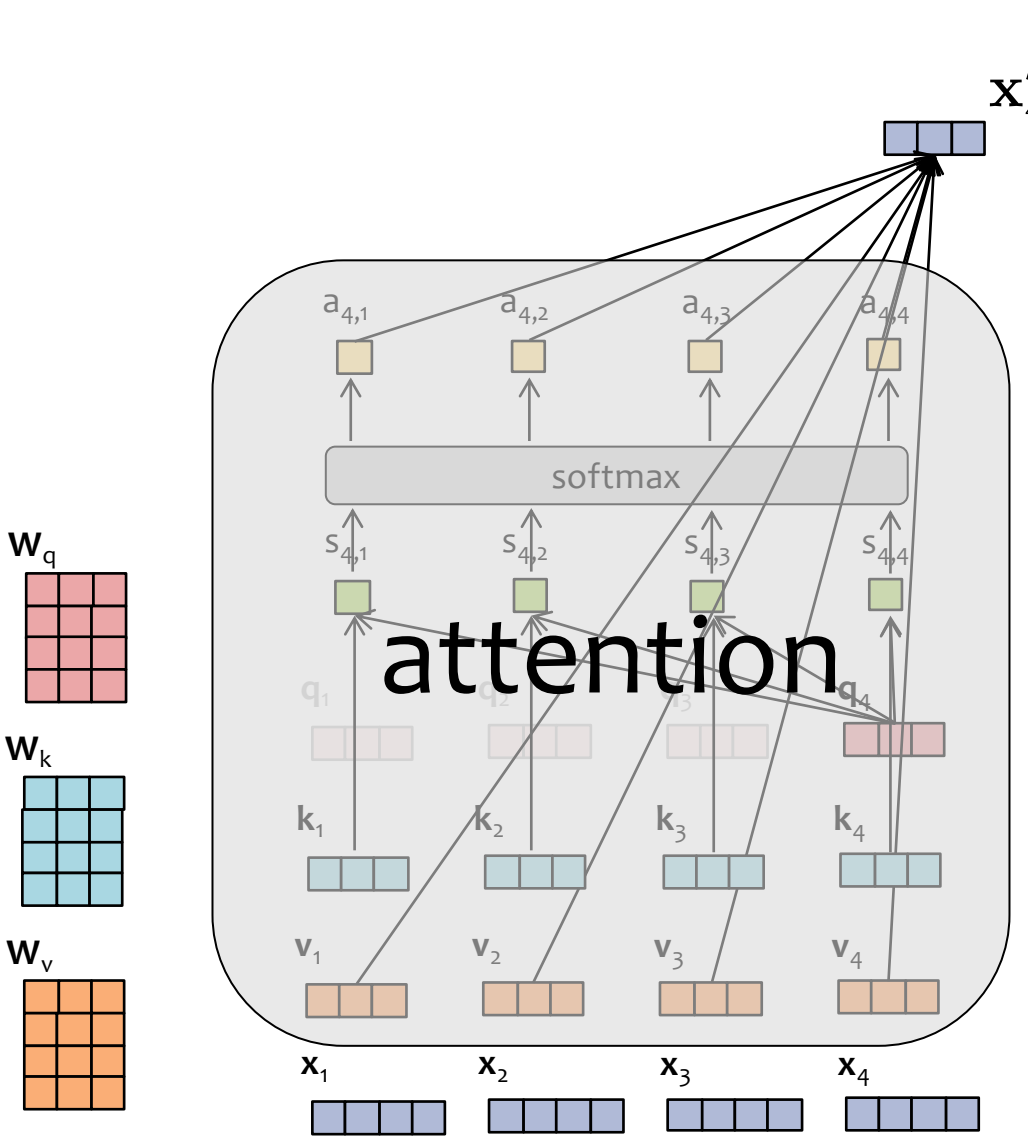
$$Q = XW_Q$$

$$K = XW_K$$

$$V = XW_V$$

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

Scaled Dot-Product Attention



$\mathbf{a}_4 = \text{softmax}(\mathbf{s}_4)$ attention weights

$s_{4,j} = \mathbf{k}_j^T \mathbf{q}_4 / \sqrt{d_k}$ scores

$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j$ queries

$\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j$ keys

$\mathbf{v}_j = \mathbf{W}_v^T \mathbf{x}_j$ values

Attention

Vector representation

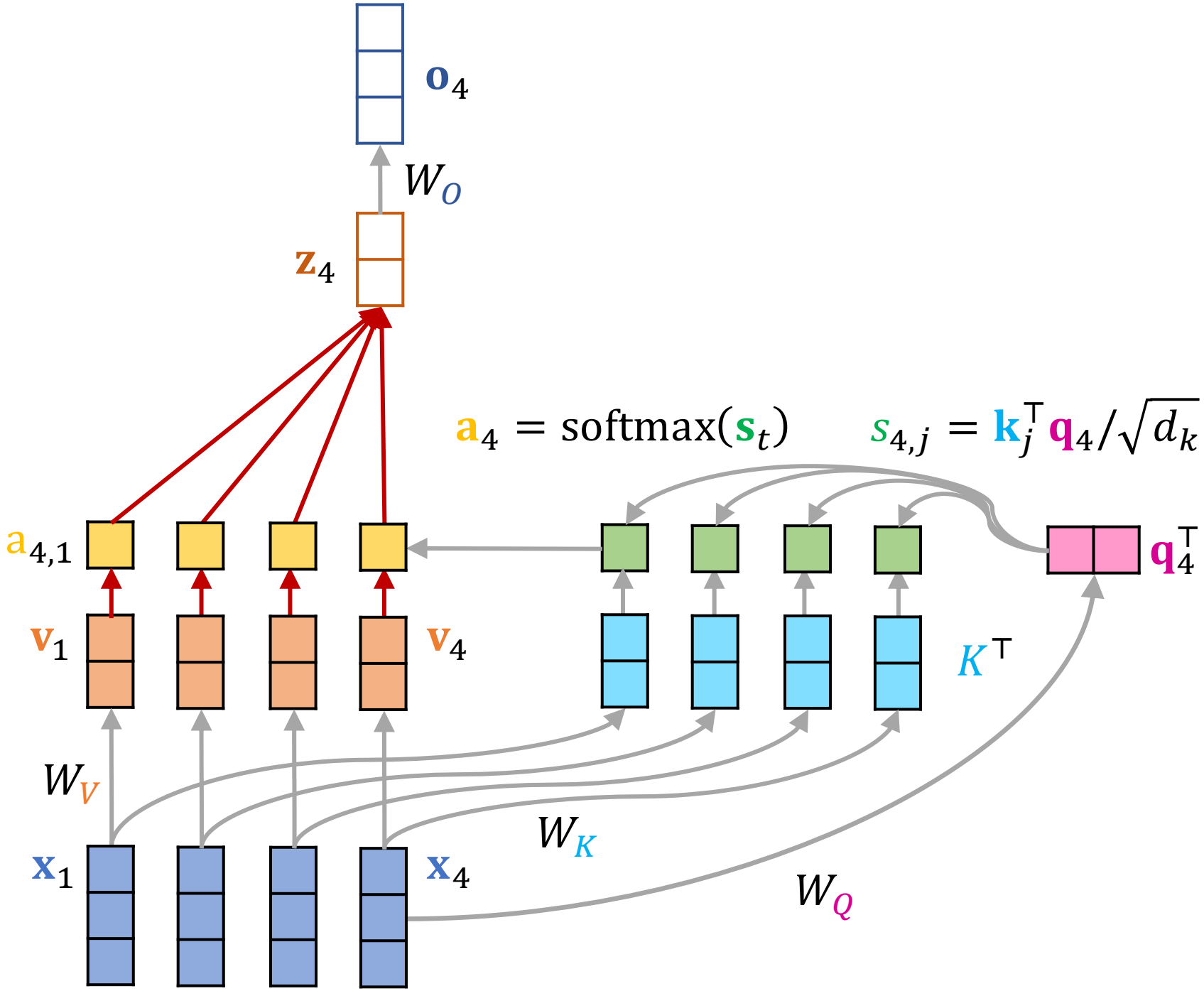
$$\mathbf{o}_t = W_O^\top \mathbf{z}_t$$

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

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

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

$$\mathbf{q}_t = W_Q^\top \mathbf{x}_t$$
$$\mathbf{k}_t = W_K^\top \mathbf{x}_t$$
$$\mathbf{v}_t = W_V^\top \mathbf{x}_t$$



Attention

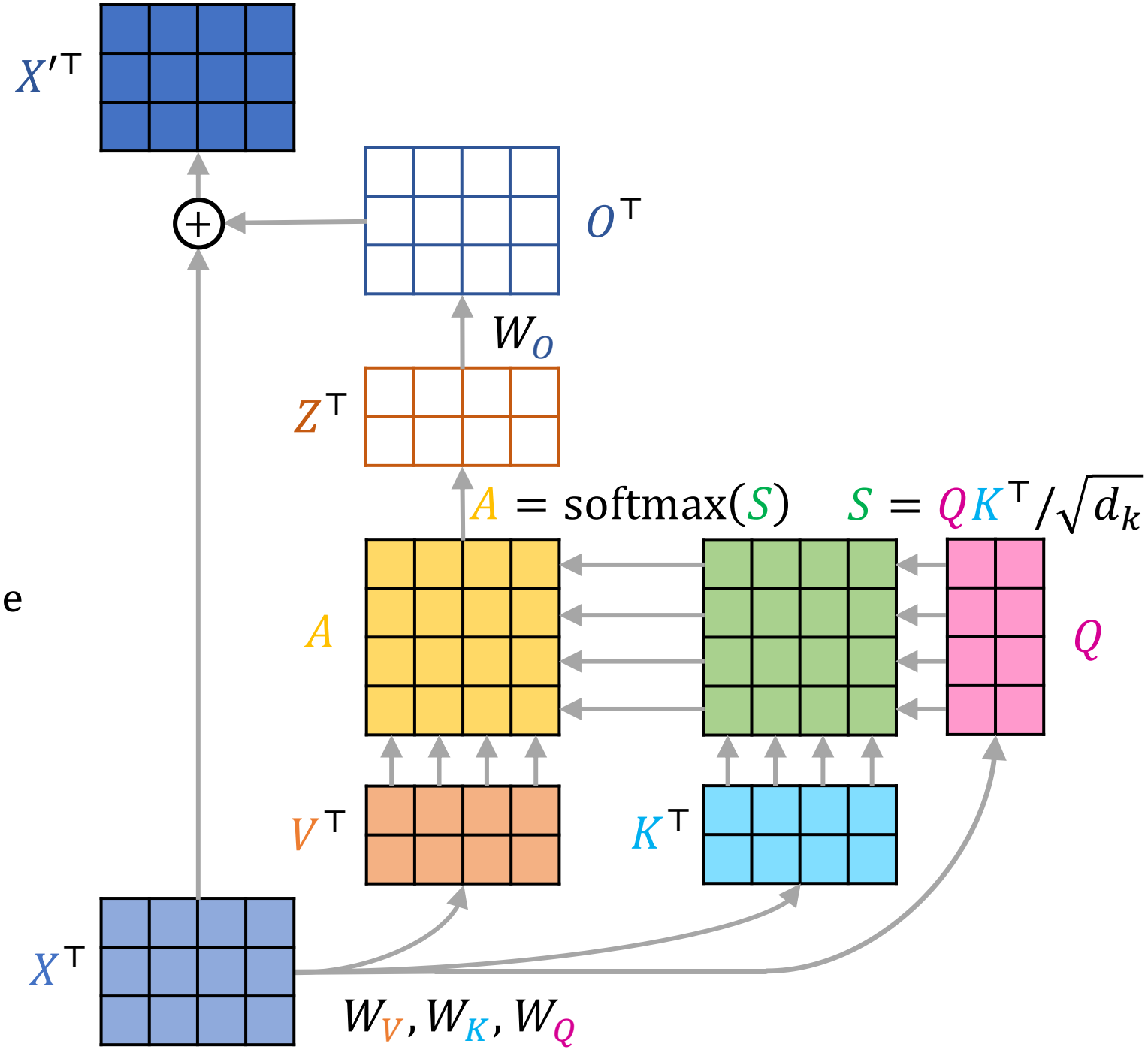
Matrix representation

$$X' = X \oplus O$$
$$O = ZW_O$$

$$Z = AV$$

$$A = \mathrm{softmax}(S)_{\mathrm{row-wise}}$$
$$S = QK^\top / \sqrt{d_k}$$

$$Q = XW_Q$$
$$K = XW_K$$
$$V = XW_V$$



Attention

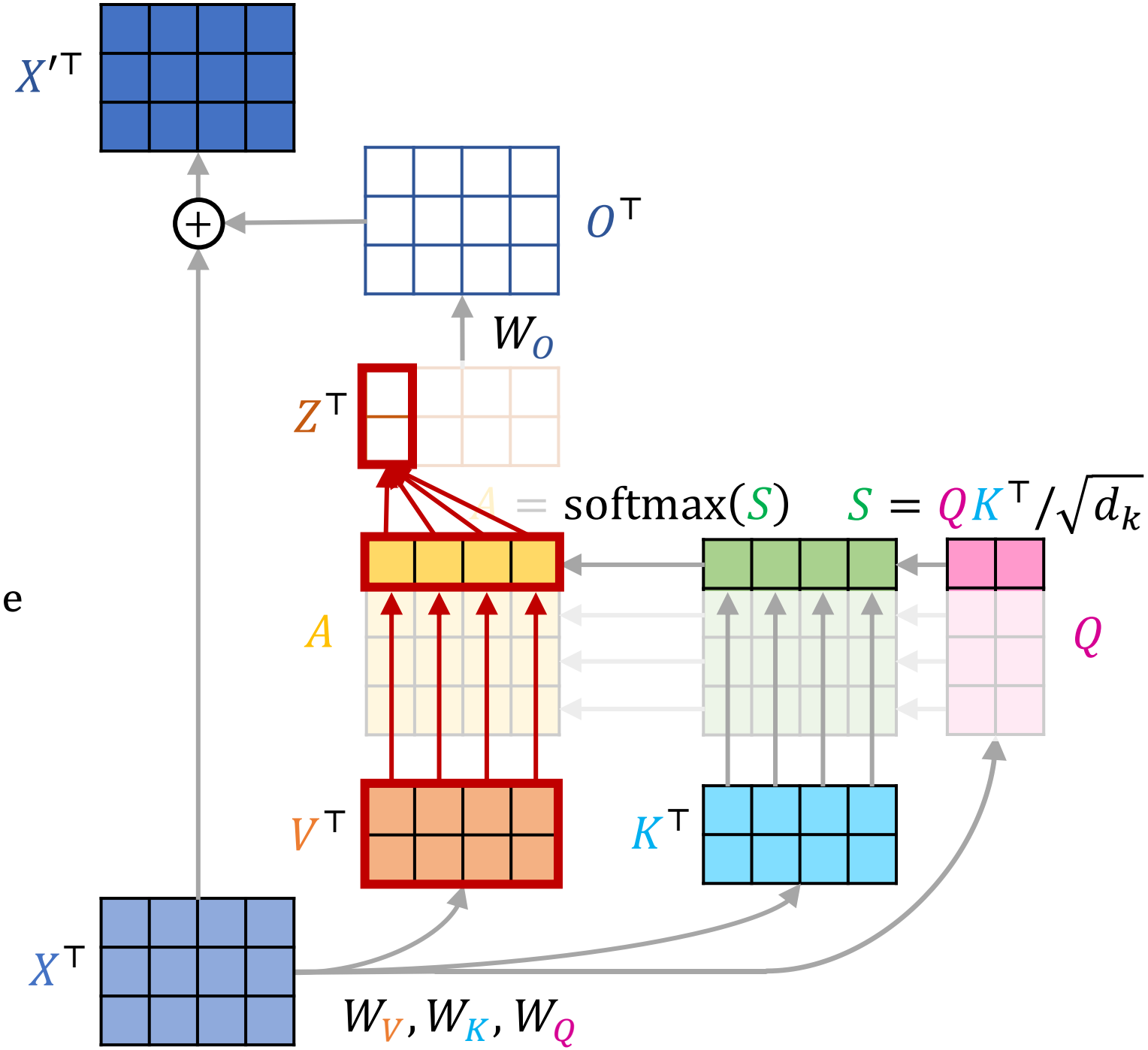
Matrix representation

$$X' = X \oplus O$$
$$O = ZW_O$$

$$Z = AV$$

$$A = \mathrm{softmax}(S)_{\mathrm{row-wise}}$$
$$S = QK^\top/\sqrt{d_k}$$

$$Q = XW_Q$$
$$K = XW_K$$
$$V = XW_V$$



Attention

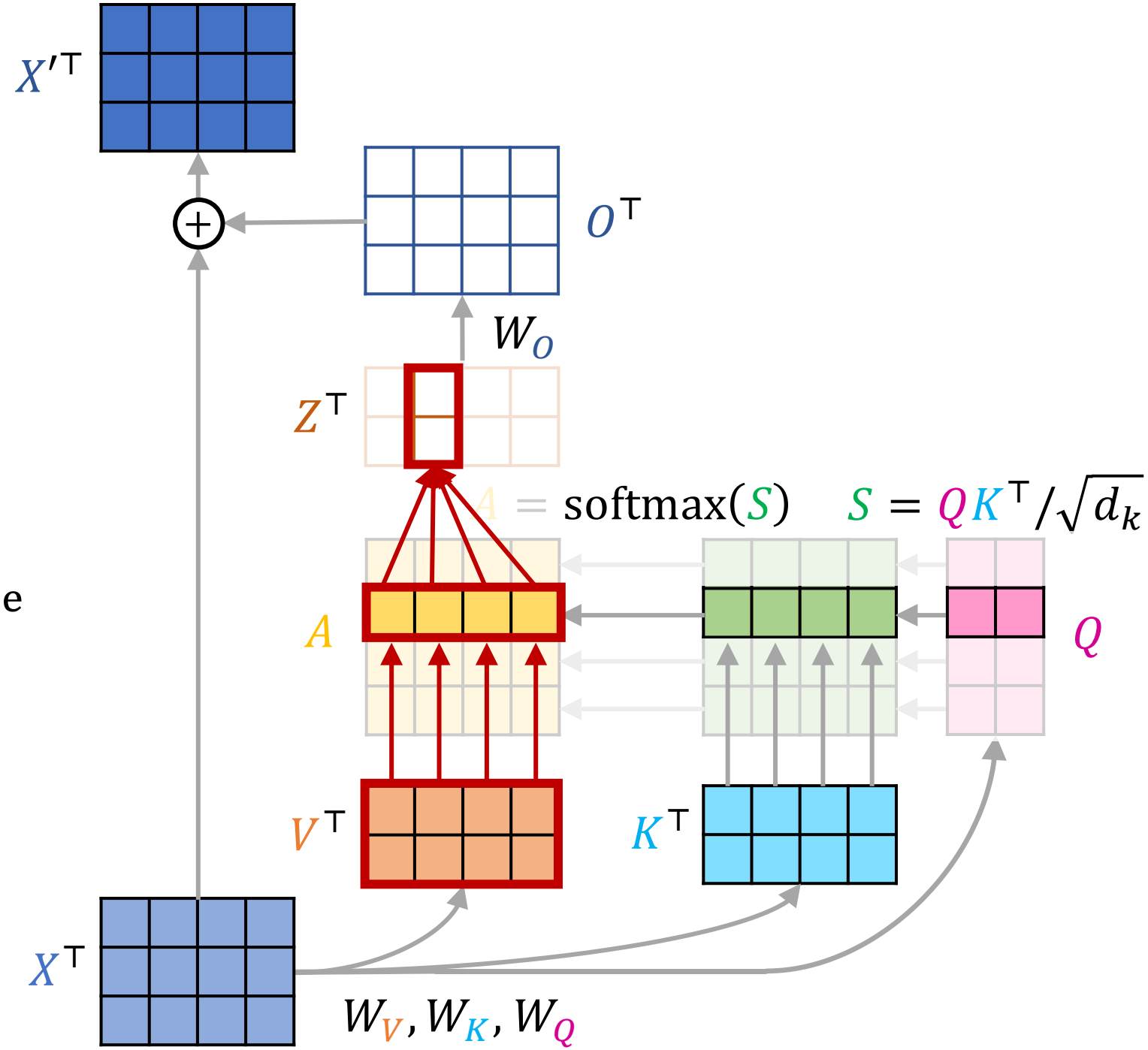
Matrix representation

$$X' = X \oplus O$$
$$O = ZW_O$$

$$Z = AV$$

$$A = \text{softmax}(S)_{\text{row-wise}}$$
$$S = QK^\top / \sqrt{d_k}$$

$$Q = XW_Q$$
$$K = XW_K$$
$$V = XW_V$$



Attention

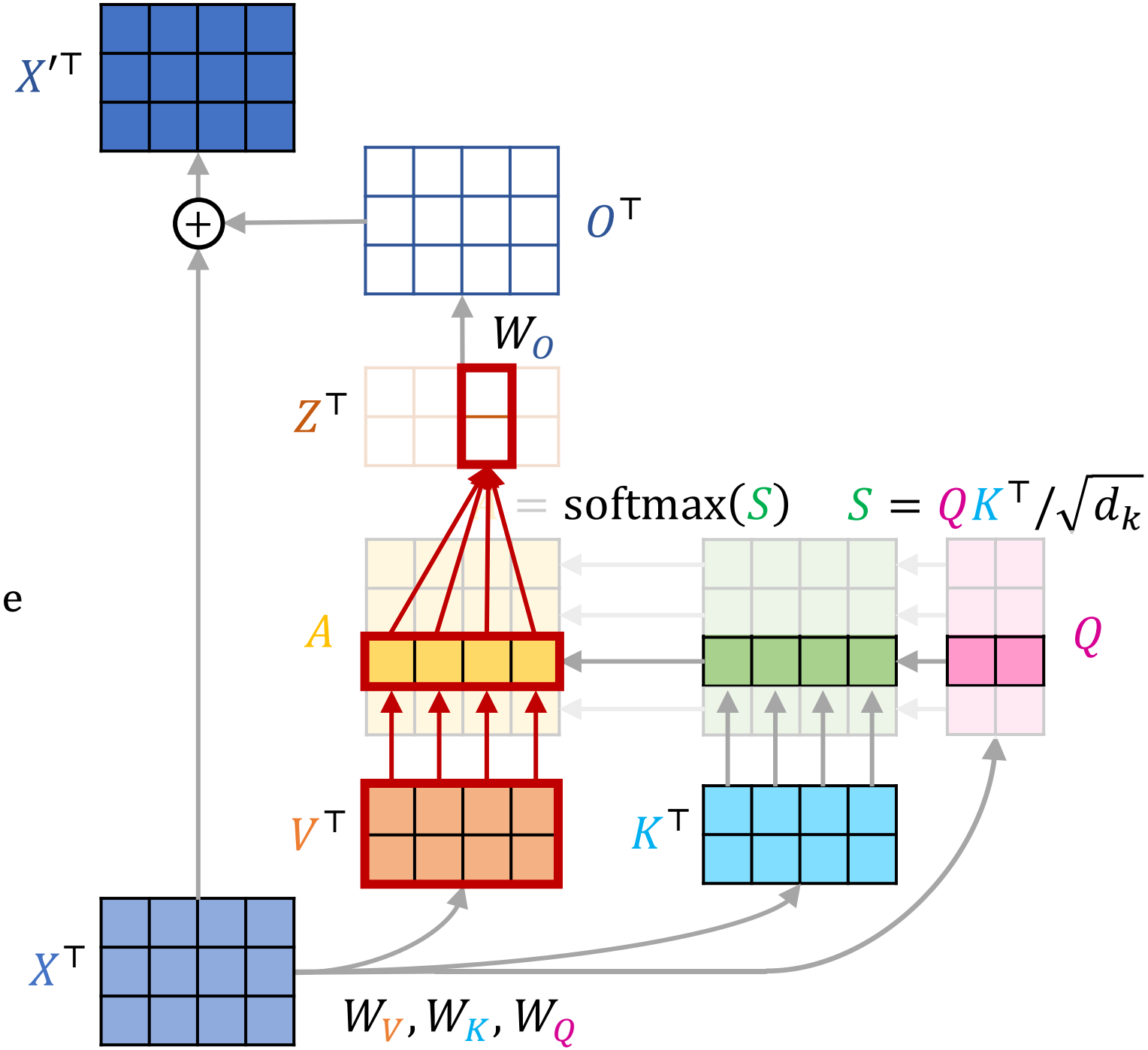
Matrix representation

$$X' = X \oplus O$$
$$O = ZW_O$$

$$Z = AV$$

$$A = \text{softmax}(S)_{\text{row-wise}}$$
$$S = QK^\top/\sqrt{d_k}$$

$$Q = XW_Q$$
$$K = XW_K$$
$$V = XW_V$$



Attention

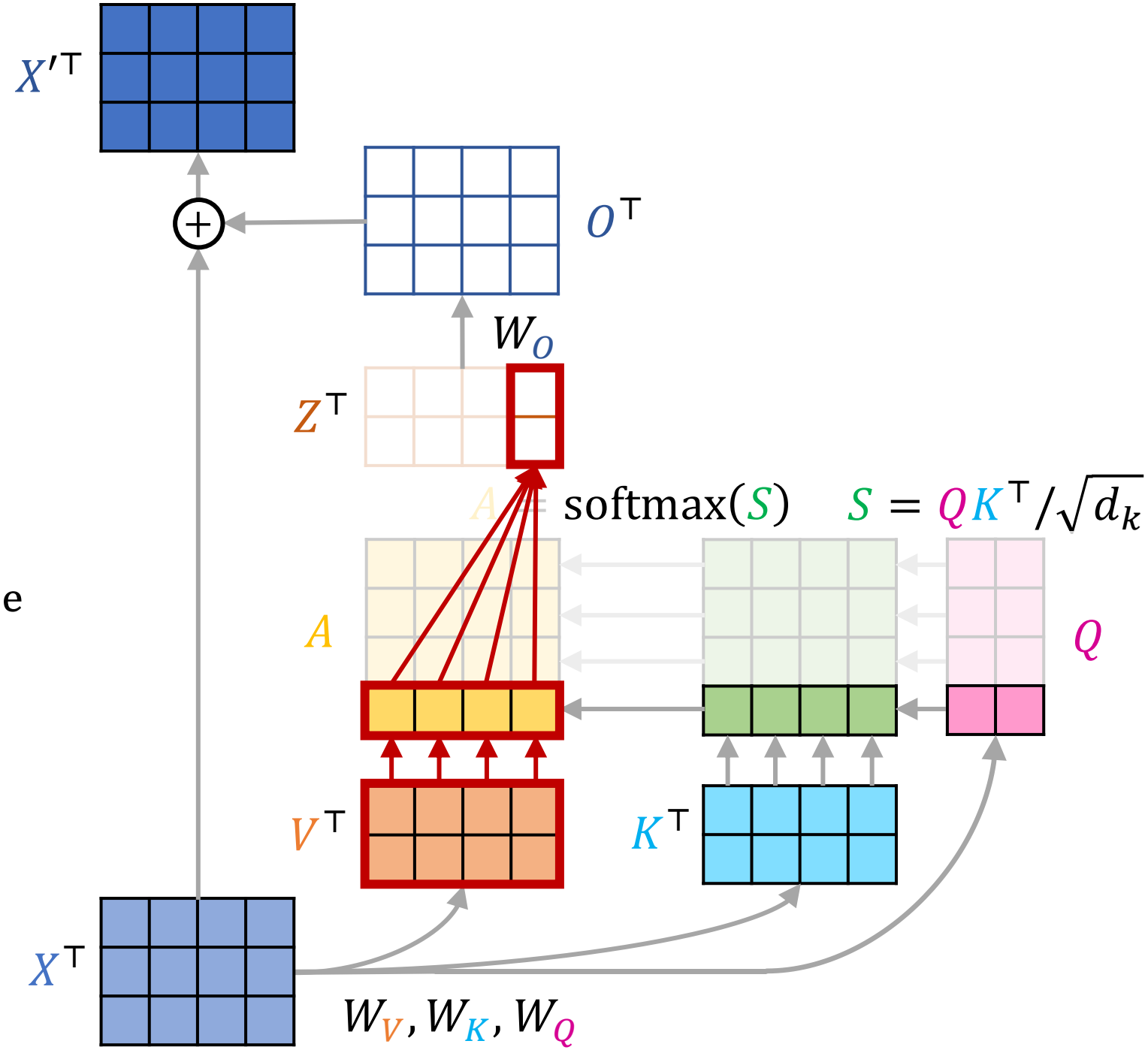
Matrix representation

$$X' = X \oplus O$$
$$O = ZW_O$$

$$Z = AV$$

$$A = \text{softmax}(S)_{\text{row-wise}}$$
$$S = QK^\top / \sqrt{d_k}$$

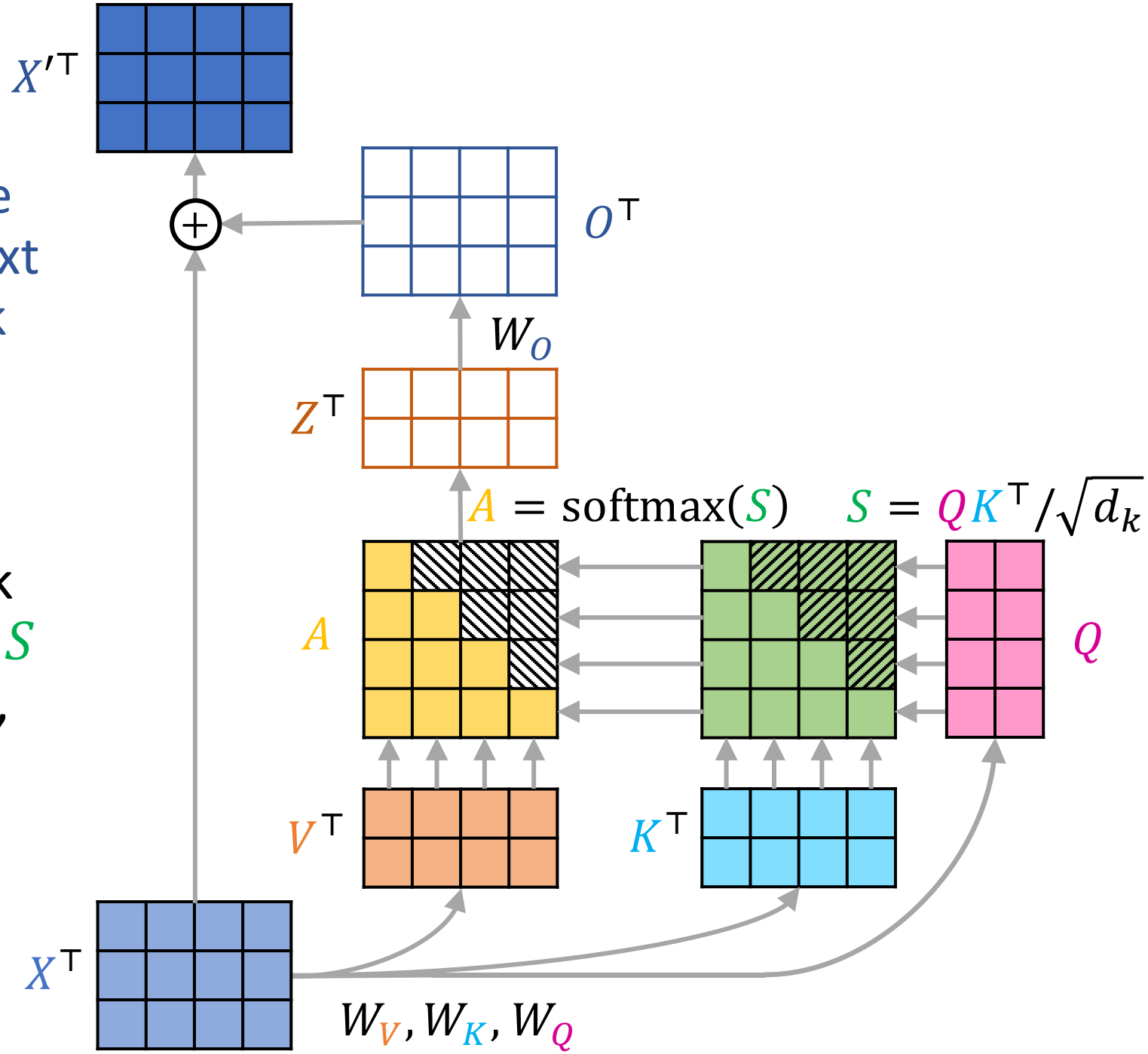
$$Q = XW_Q$$
$$K = XW_K$$
$$V = XW_V$$



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

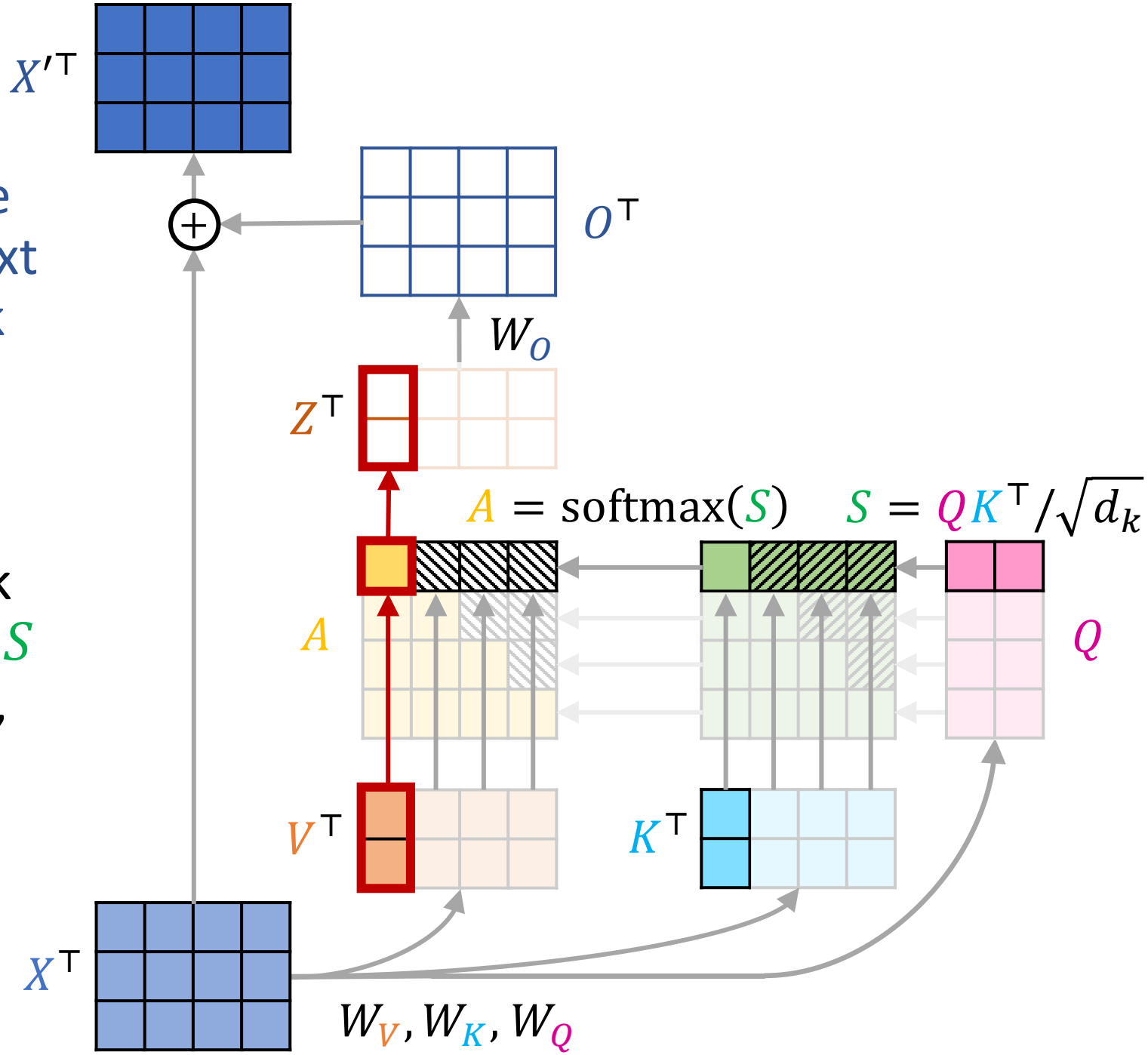
- We apply a causal mask to the attention scores S (filled with $-\infty$ values), which zeros out the appropriate attention weights in A



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

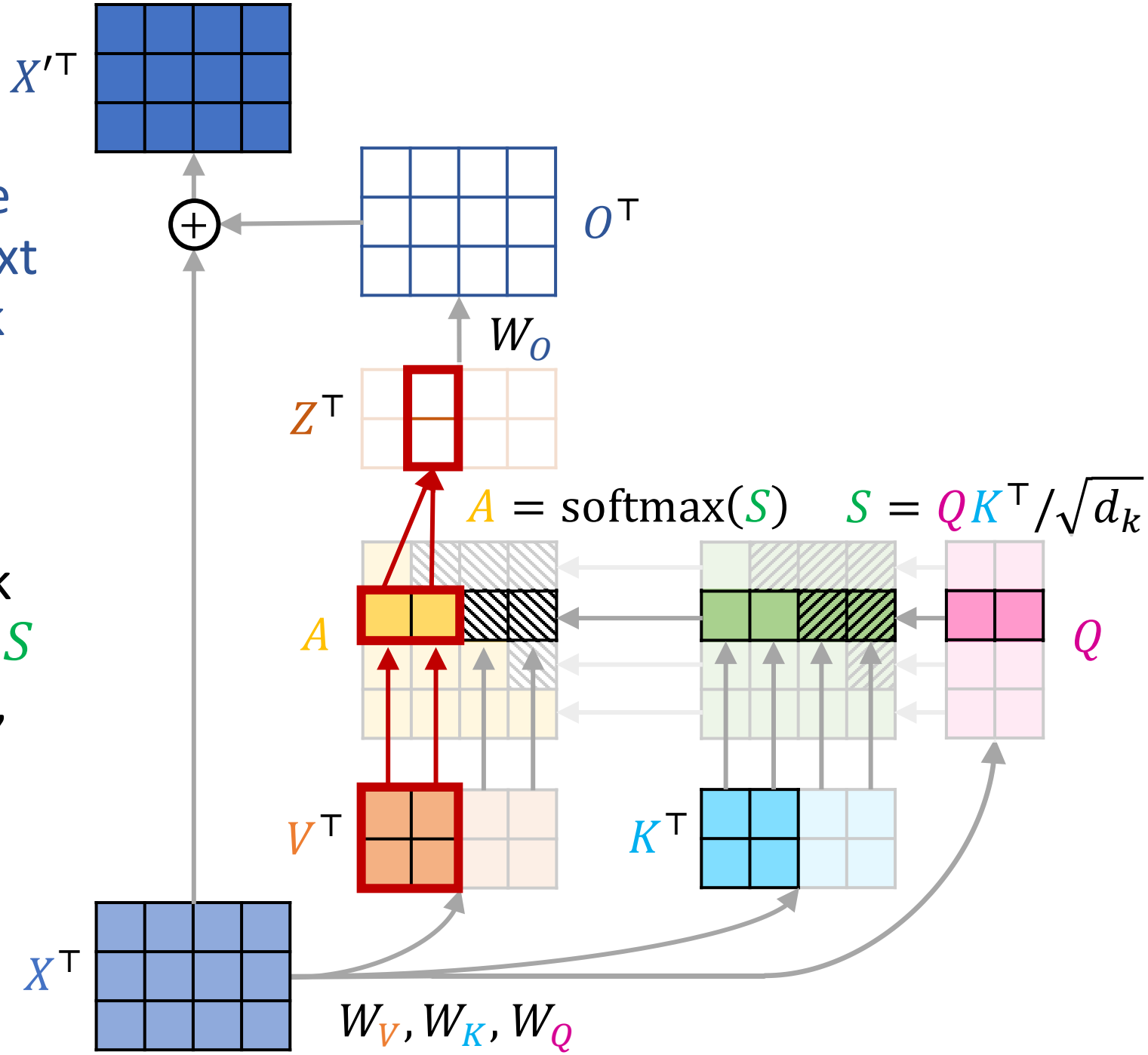
- We apply a causal mask to the attention scores S (filled with $-\infty$ values), which zeros out the appropriate attention weights in A



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

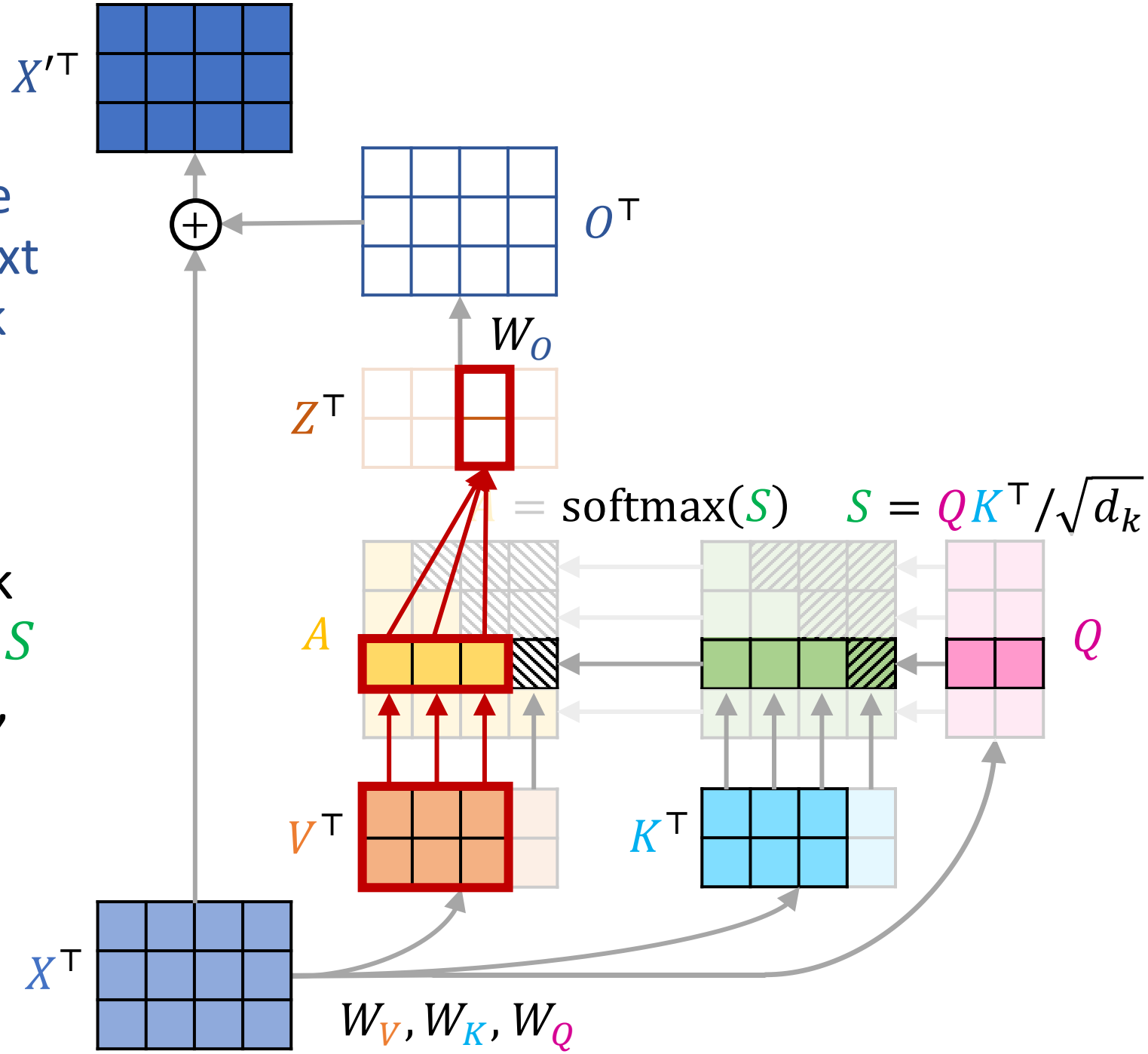
- We apply a causal mask to the attention scores S (filled with $-\infty$ values), which zeros out the appropriate attention weights in A



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

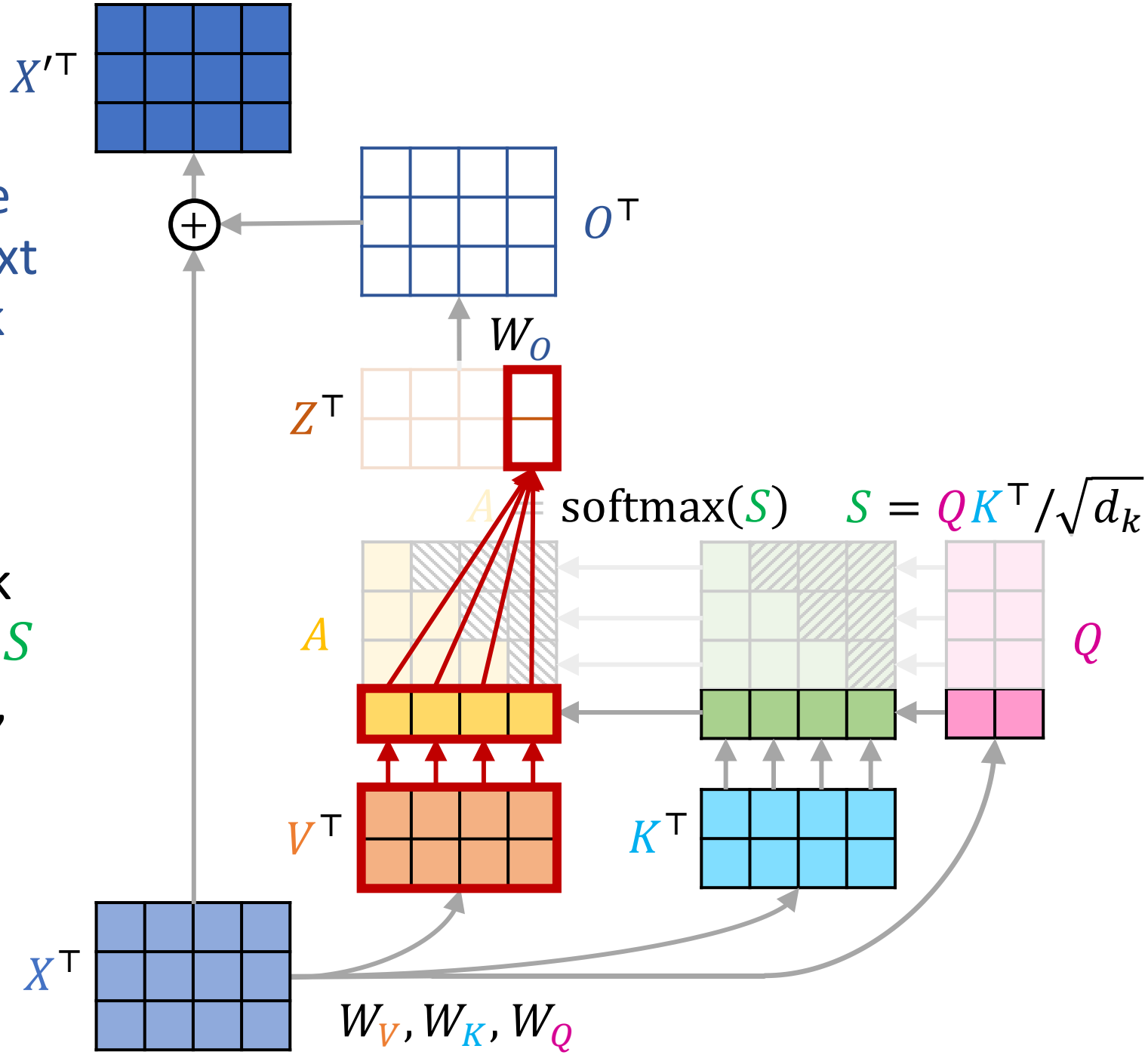
- We apply a causal mask to the attention scores S (filled with $-\infty$ values), which zeros out the appropriate attention weights in A



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

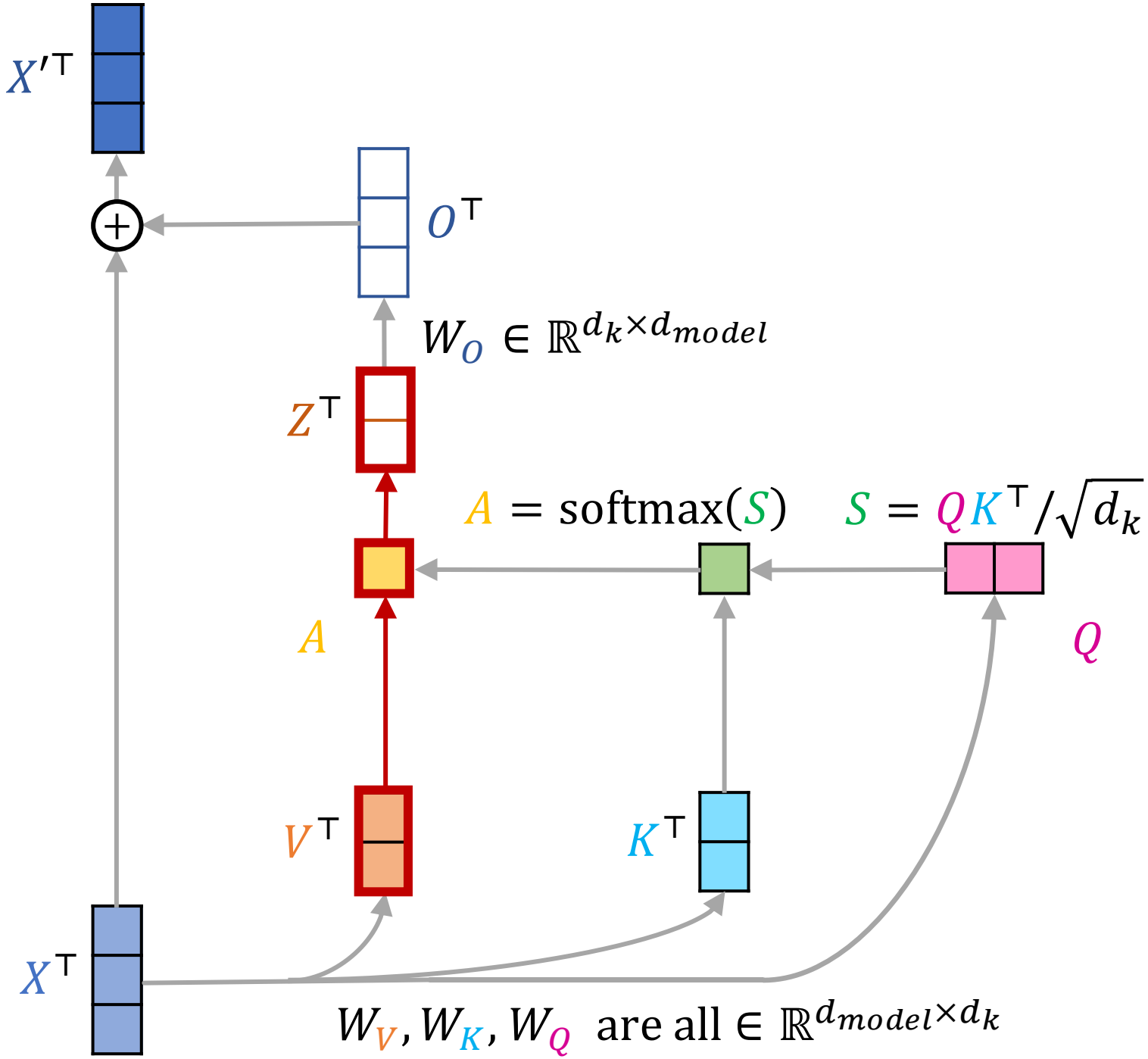
- We apply a causal mask to the attention scores S (filled with $-\infty$ values), which zeros out the appropriate attention weights in A



Causal Attention

Inference time

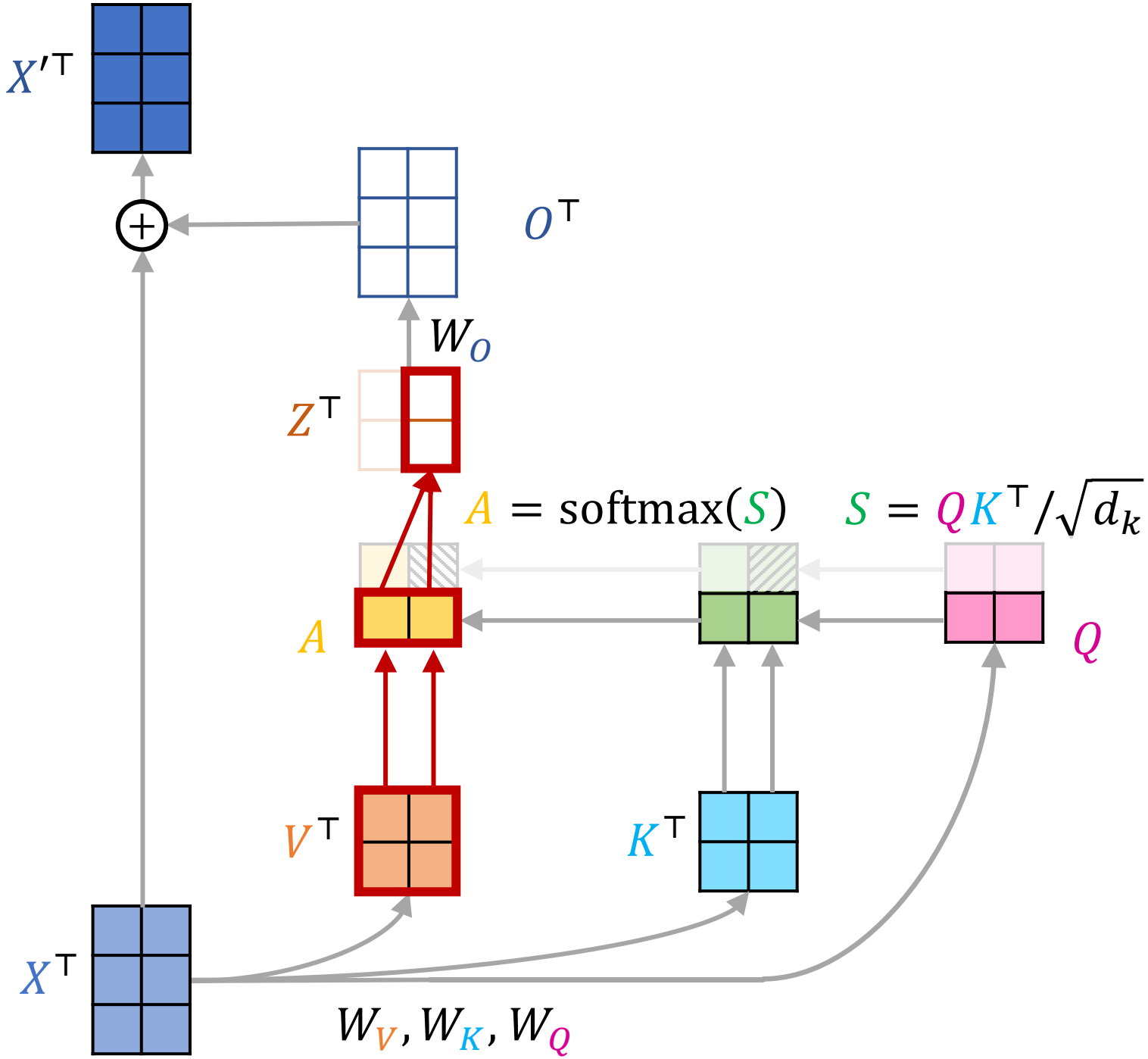
- Done training. Watch this attention block as the context size increases as we generate more and more tokens
- Note how different components build up as the context grows
- But the size of the parameters don't change!



Causal Attention

Inference time

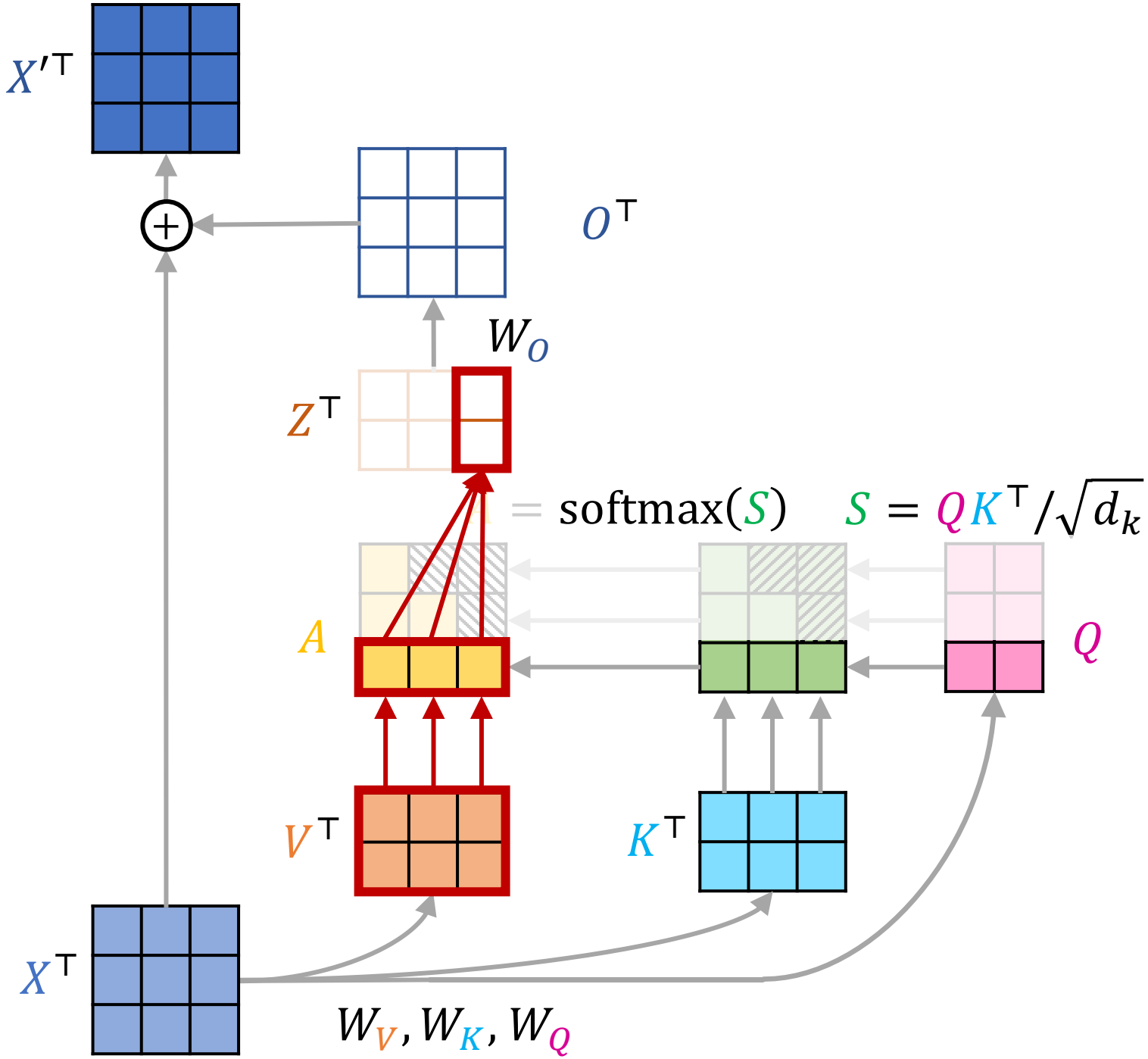
- Done training. Watch this attention block as the context size increases as we generate more and more tokens
- Note how different components build up as the context grows
- But the size of the parameters don't change!



Causal Attention

Inference time

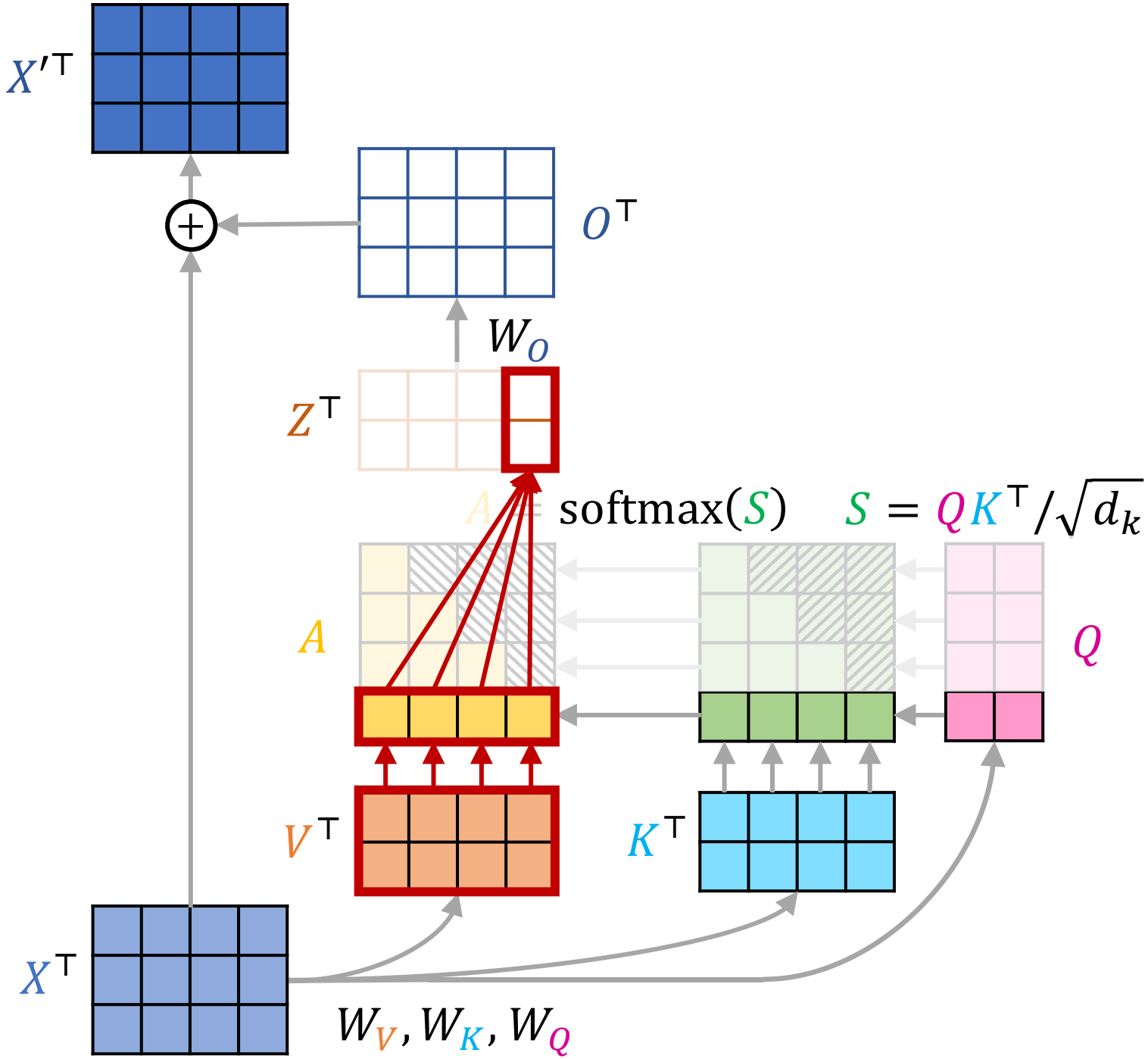
- Done training. Watch this attention block as the context size increases as we generate more and more tokens
- Note how different components build up as the context grows
- But the size of the parameters don't change!



Causal Attention

Inference time

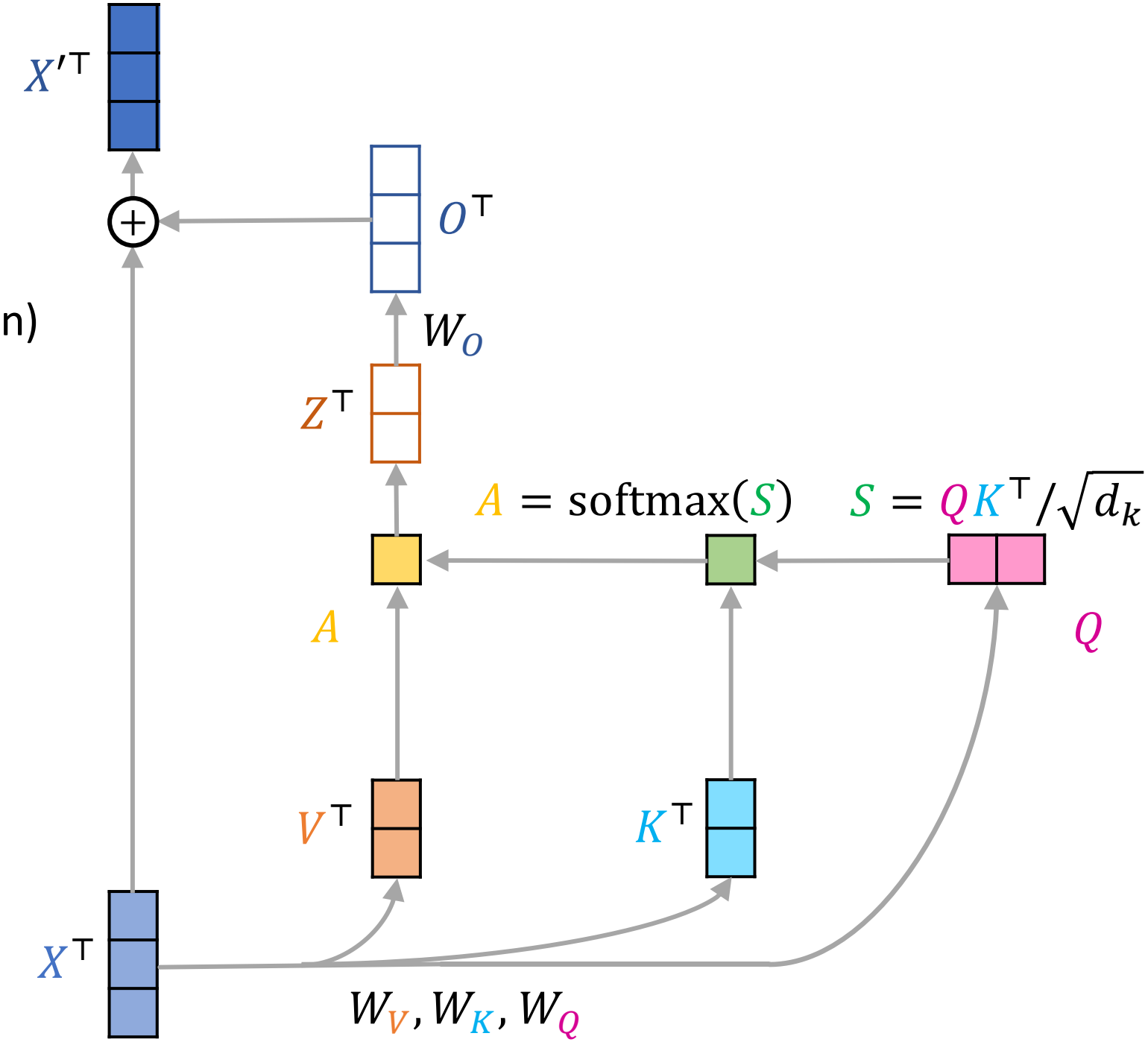
- Done training. Watch this attention block as the context size increases as we generate more and more tokens
- Note how different components build up as the context grows
- But the size of the parameters don't change!



Causal Attention

Inference time

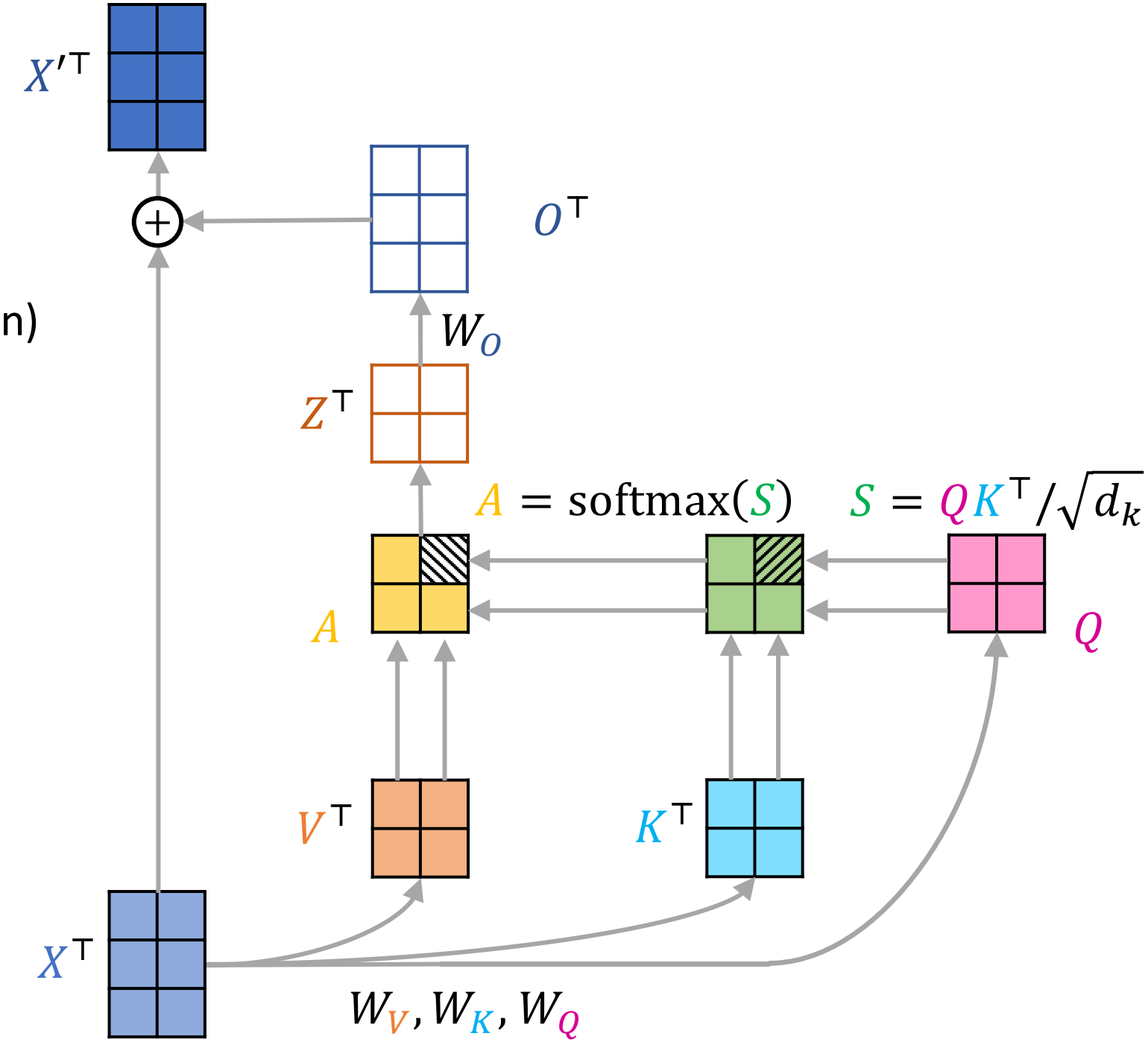
(Repeated without red arrows showing attention combination)



Causal Attention

Inference time

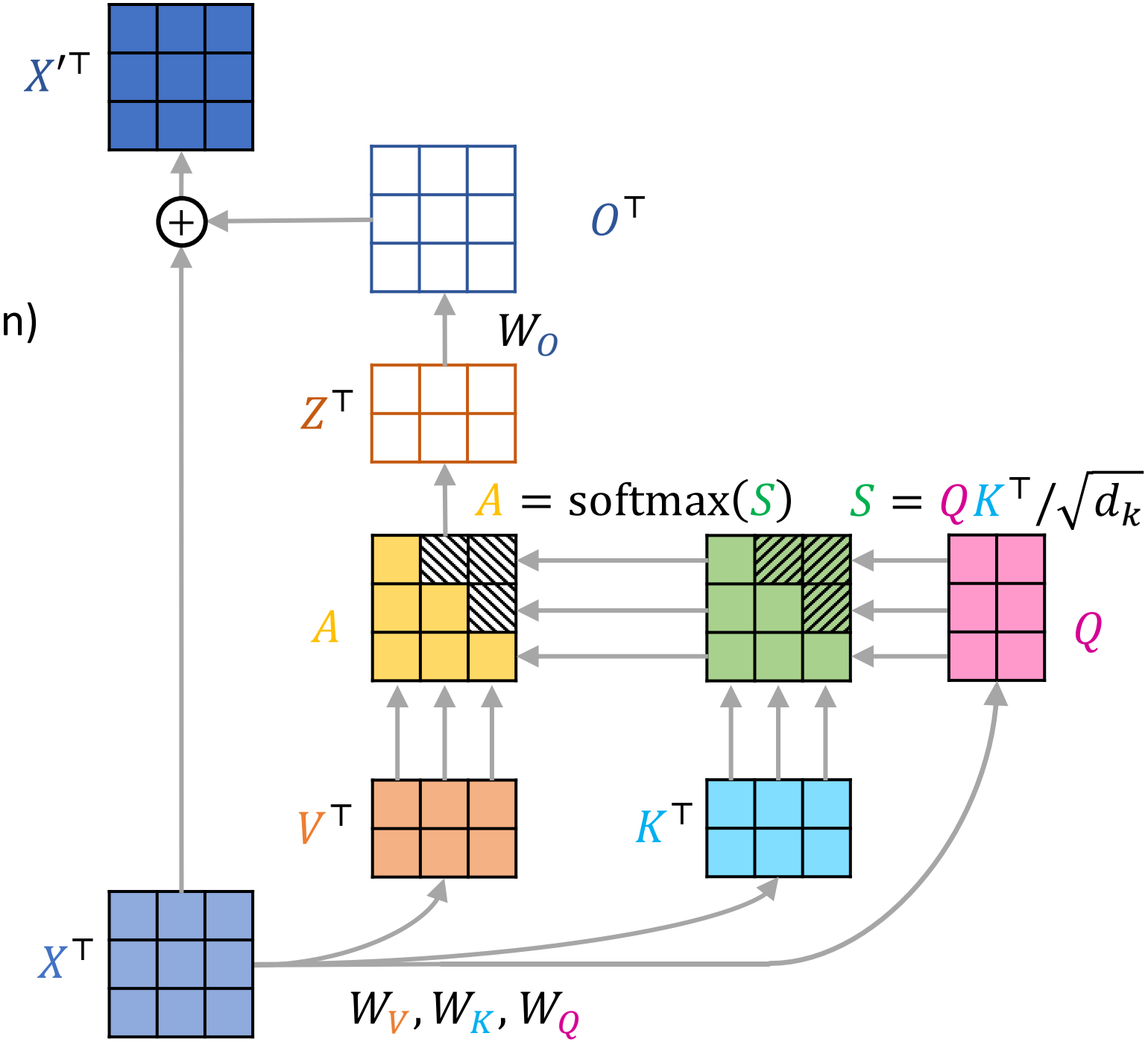
(Repeated without red arrows showing attention combination)



Causal Attention

Inference time

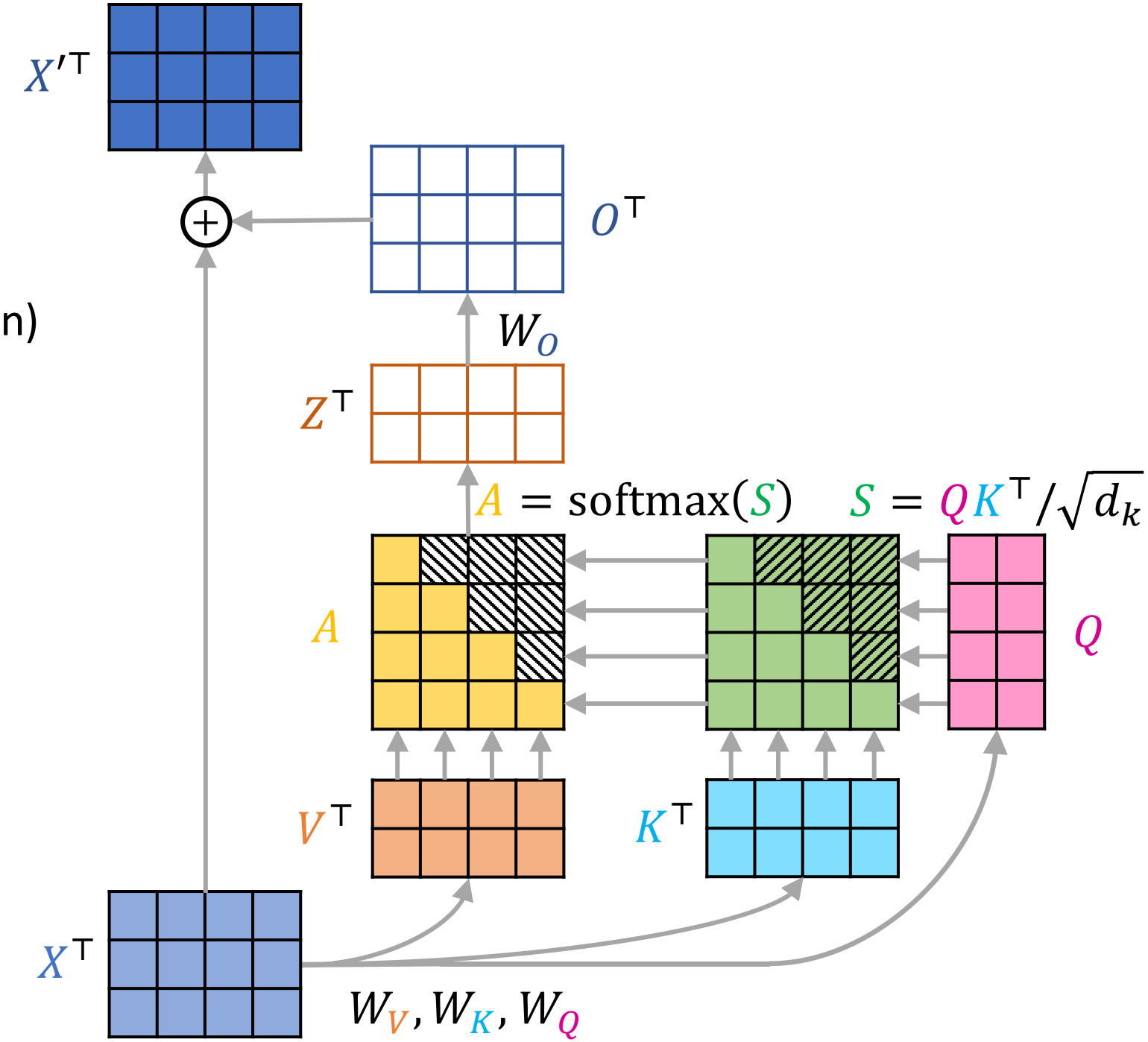
(Repeated without red arrows showing attention combination)



Causal Attention

Inference time

(Repeated without red arrows showing attention combination)

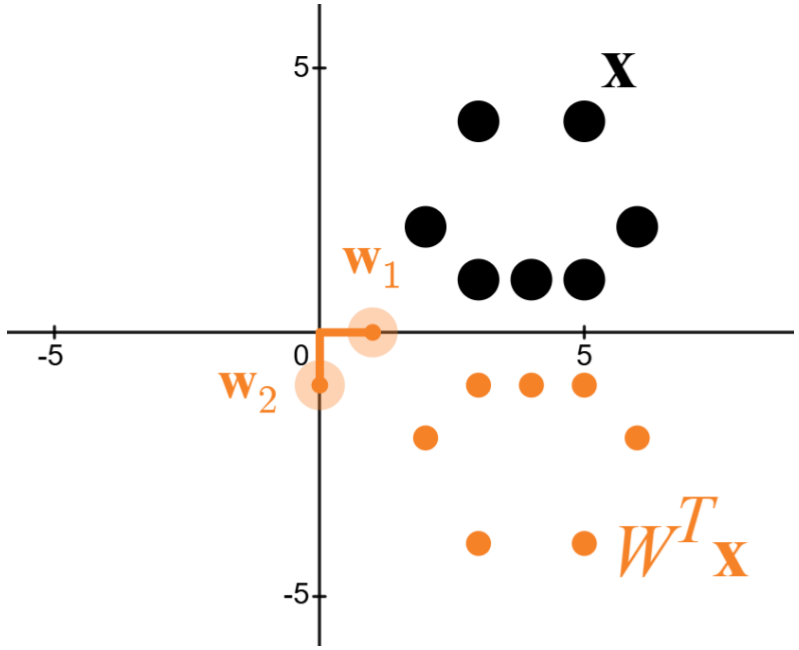


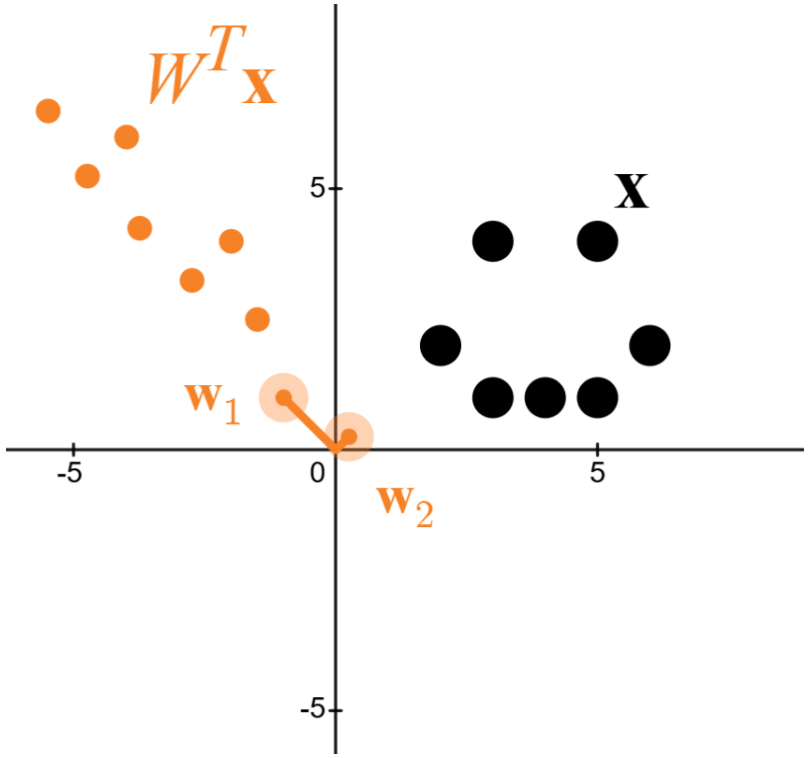
Linear Transforms: Graphical Intuition

In Transformer models, we see quite a few linear transforms

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

Desmos example for $\boldsymbol{x}$ and $\mathbf{z}$ in $\mathbb{R}^2$
https://www.desmos.com/calculator/gl5ljvorcy





Linear Transforms: Graphical Intuition

Two different transforms $W_{tech}^{\top}\mathbf{z}$ and $W_{fruit}^{\top}\mathbf{z}$ 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/tbeclbo83h

