Deep Learning

Transformers and Graph Neural Networks

Abuzar Khan

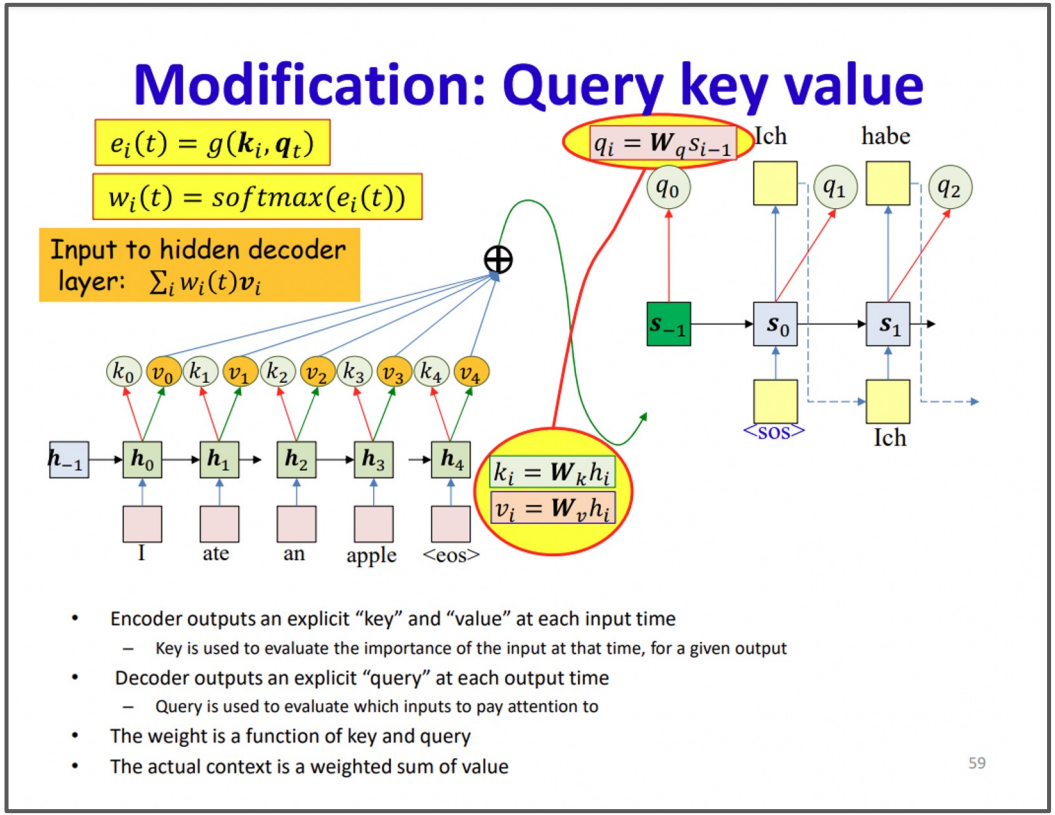
11-785, Spring 2023

Part 1

Transformers

Recall

1. Queries, Keys, and Values



Recall

1. Queries, Keys, and Values
2. Self Attention
   a. Energy Function

      – scoring function

   b. Attention Function

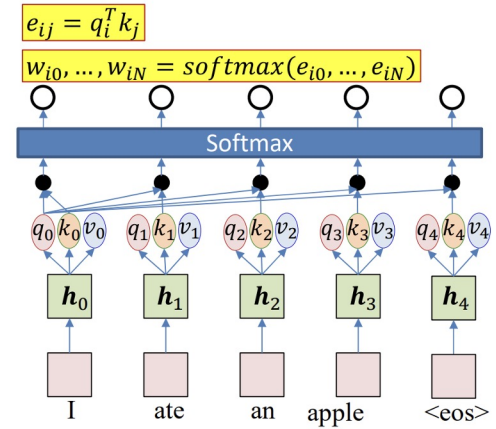
Self Attention

$q_i = \boldsymbol{W}_q h_i$

$k_i = \boldsymbol{W}_k h_i$

$v_i = \boldsymbol{W}_v h_i$

$w_{ij} = attn(q_i, k_{0:N})$

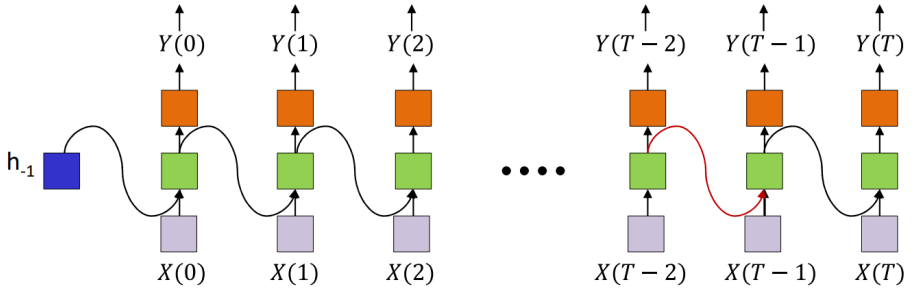


- For each word, we compute an attention weight between that word and all other words
  - The raw attention of the ith word to the jth word is a function of query q_i and key k_j
  - The raw attention values are put through a softmax to get the final attention weights

Recall

1. Queries, Keys, and Values
2. Self Attention
   a. Energy Function
   b. Attention Function
3. RNNs are slow and sequential
   a. Attention-based models can be **parallelized**!

Processing order



- Computing $Y(T)$ requires $Y(T-1)$…
- Which requires $Y(T-2)$, etc…
- RNN inputs must be processed *in order* → slow implementation

Why Transformers

- We want representations that are “dynamic” to context

  “I **like** this movie” vs. “I do not **like** this movie”

  like should have different representations in both cases
- Vanilla RNNs are **Slow** and have **terrible memory**
- LSTMs and GRUs fix the **memory** problem, but are still **slow** and **sequential**
- CNNs can be **parallelized** but the kernels are static.
- We want **parallelizability,** good **memory,** and **dynamic** computation

Q,K,V in Attention

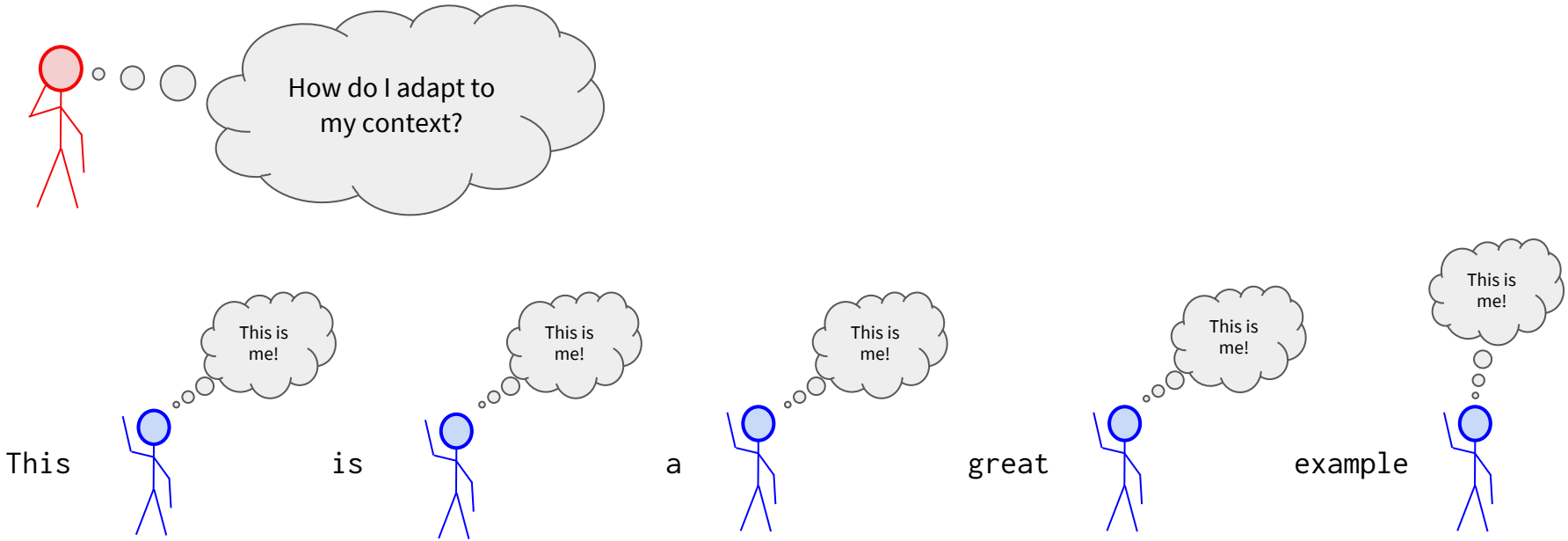
Query: This is what **pays the attention**

Values: These are **paid attention to**

Keys: These help queries figure out **how much attention to pay** to each of the values

Attention Weights: How much attention to pay.

Q,K,V in Attention



Q,K,V in Attention

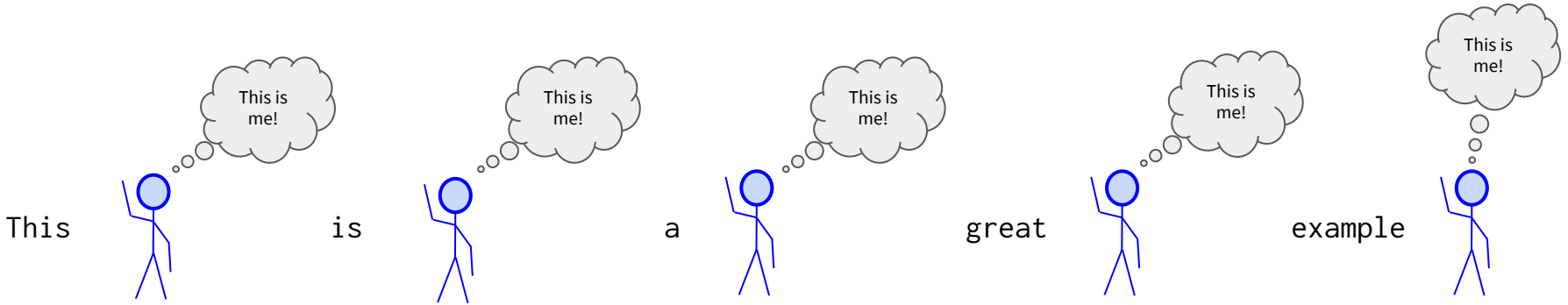
o_{this}

Contextualized Representation

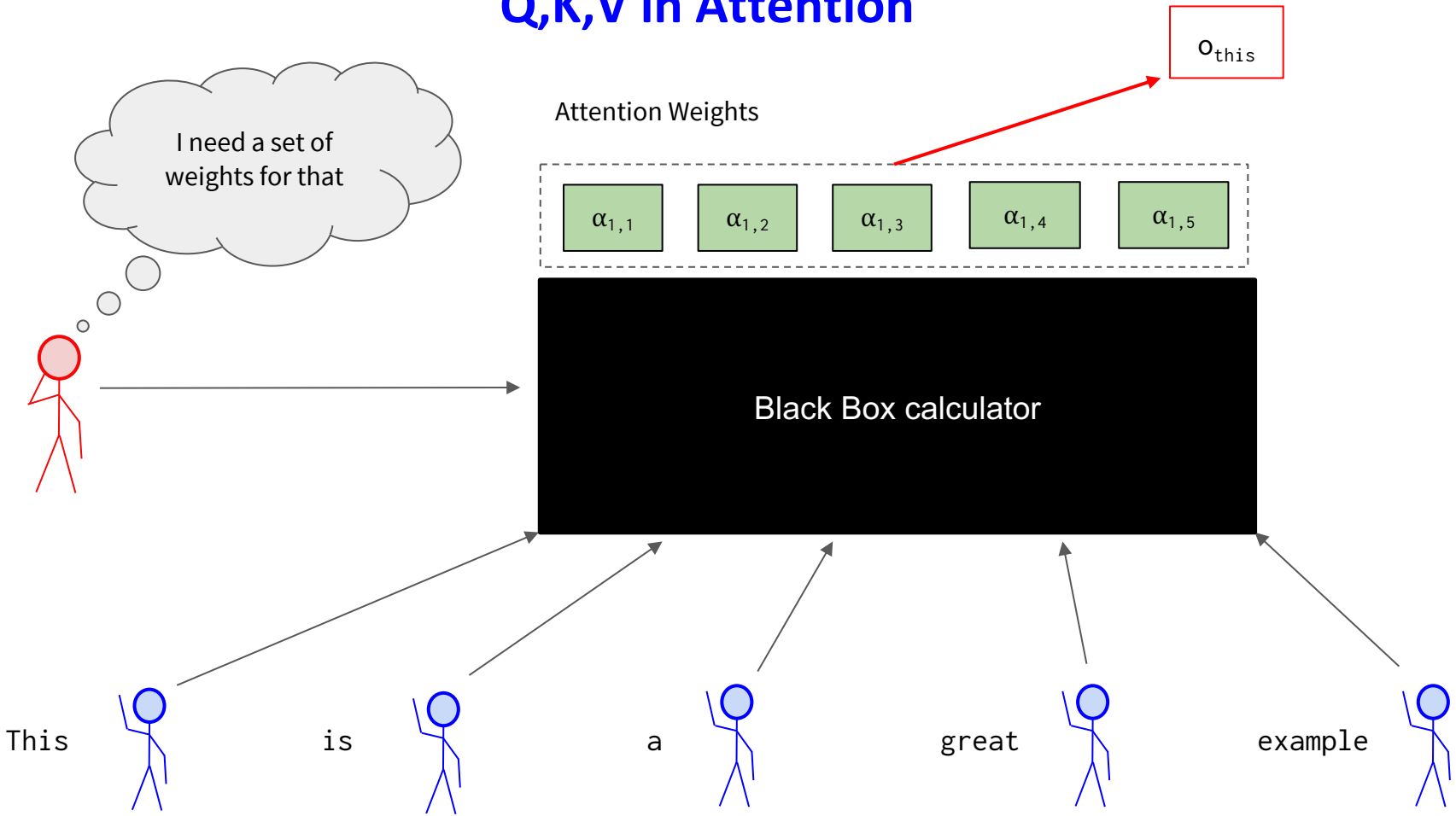
Goal

How do I adapt to my context?

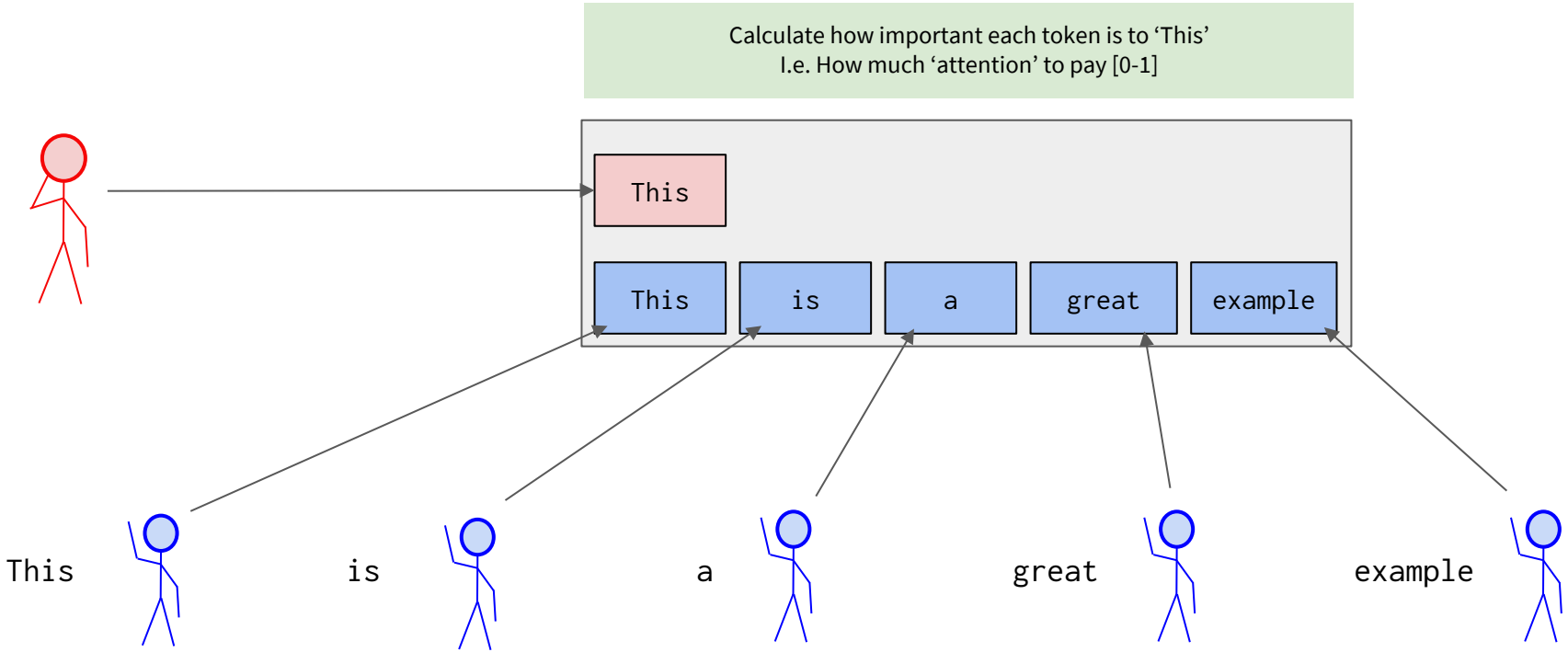
Maybe a weighted sum?



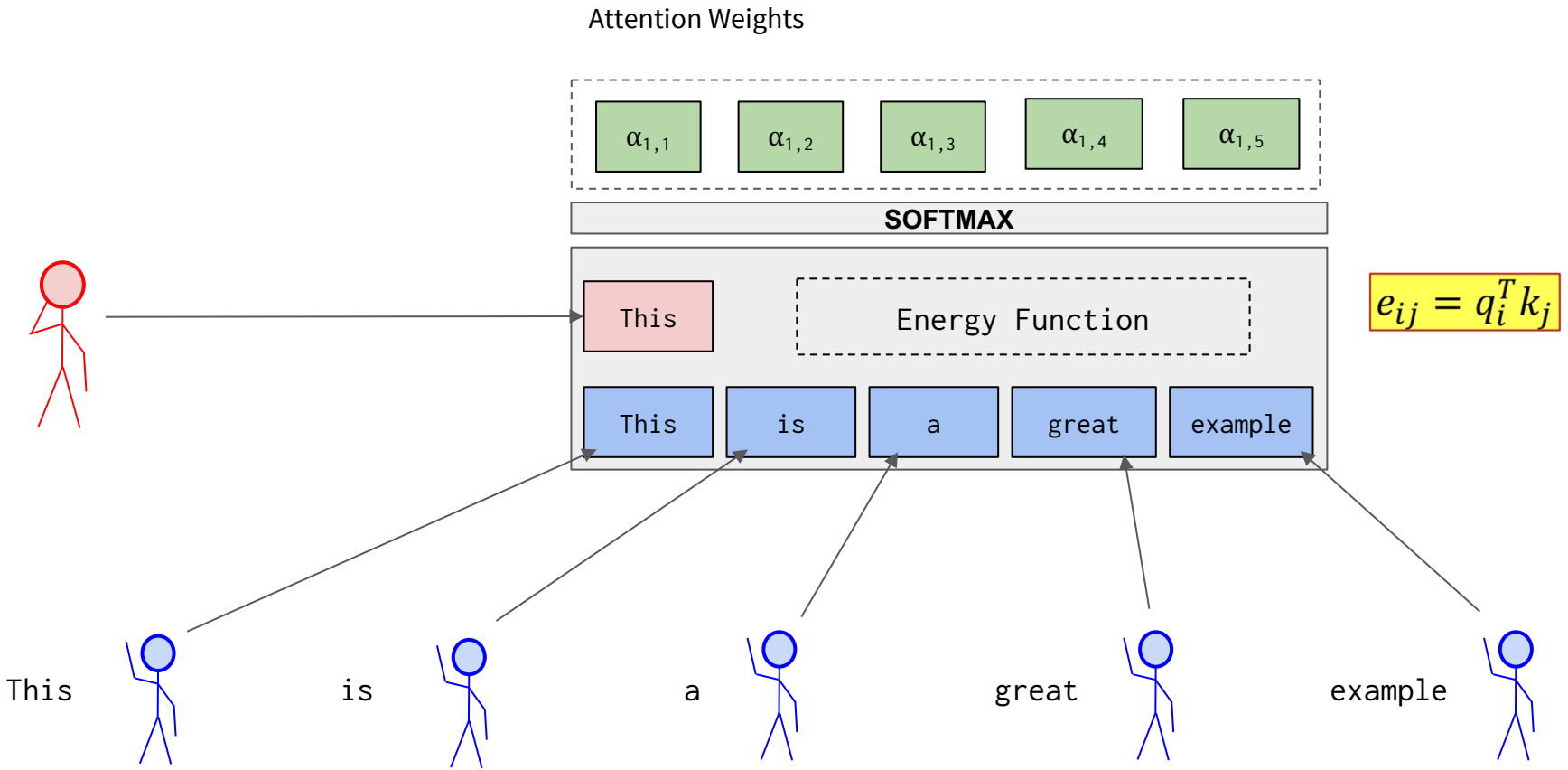
Q,K,V in Attention



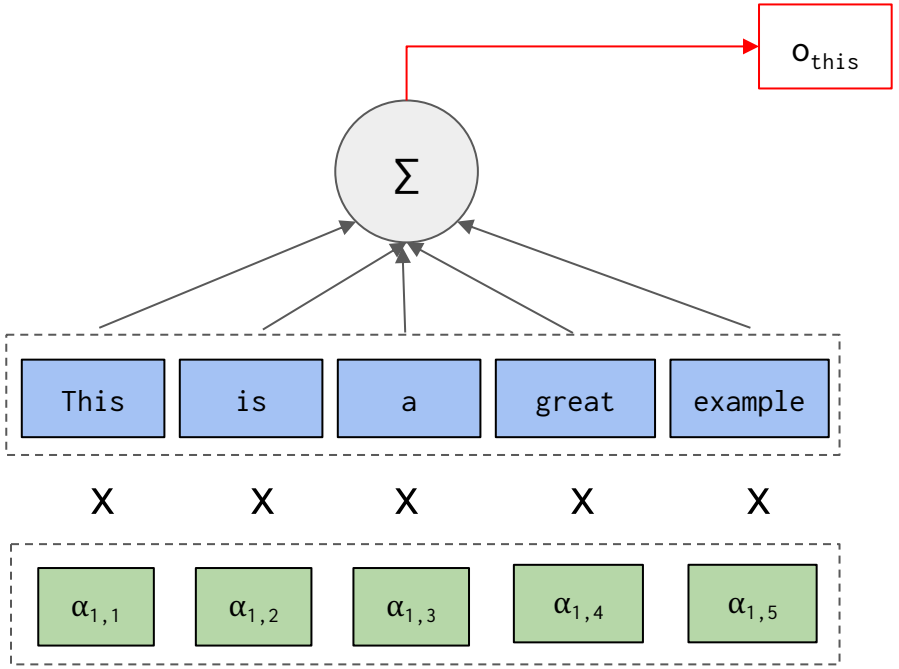
Q,K,V in Attention

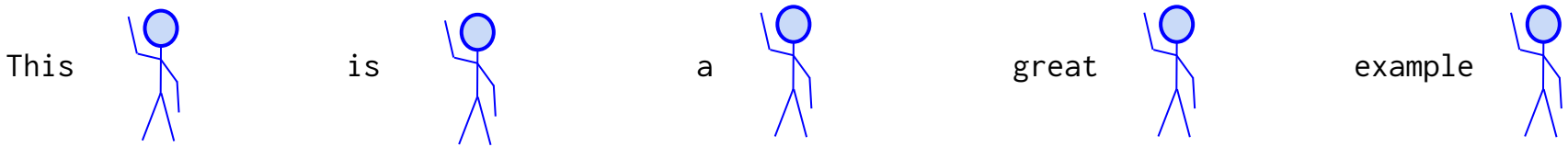


Q,K,V in Attention

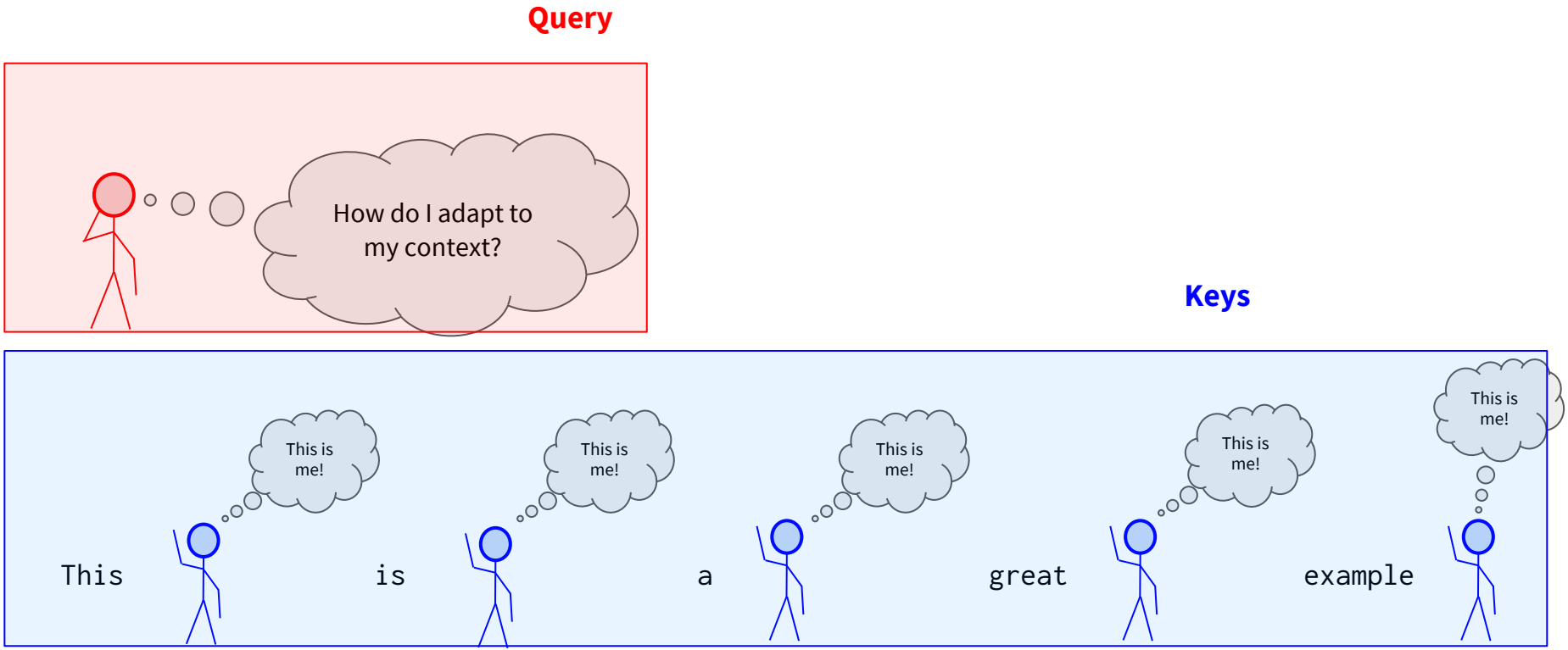


Q,K,V in Attention

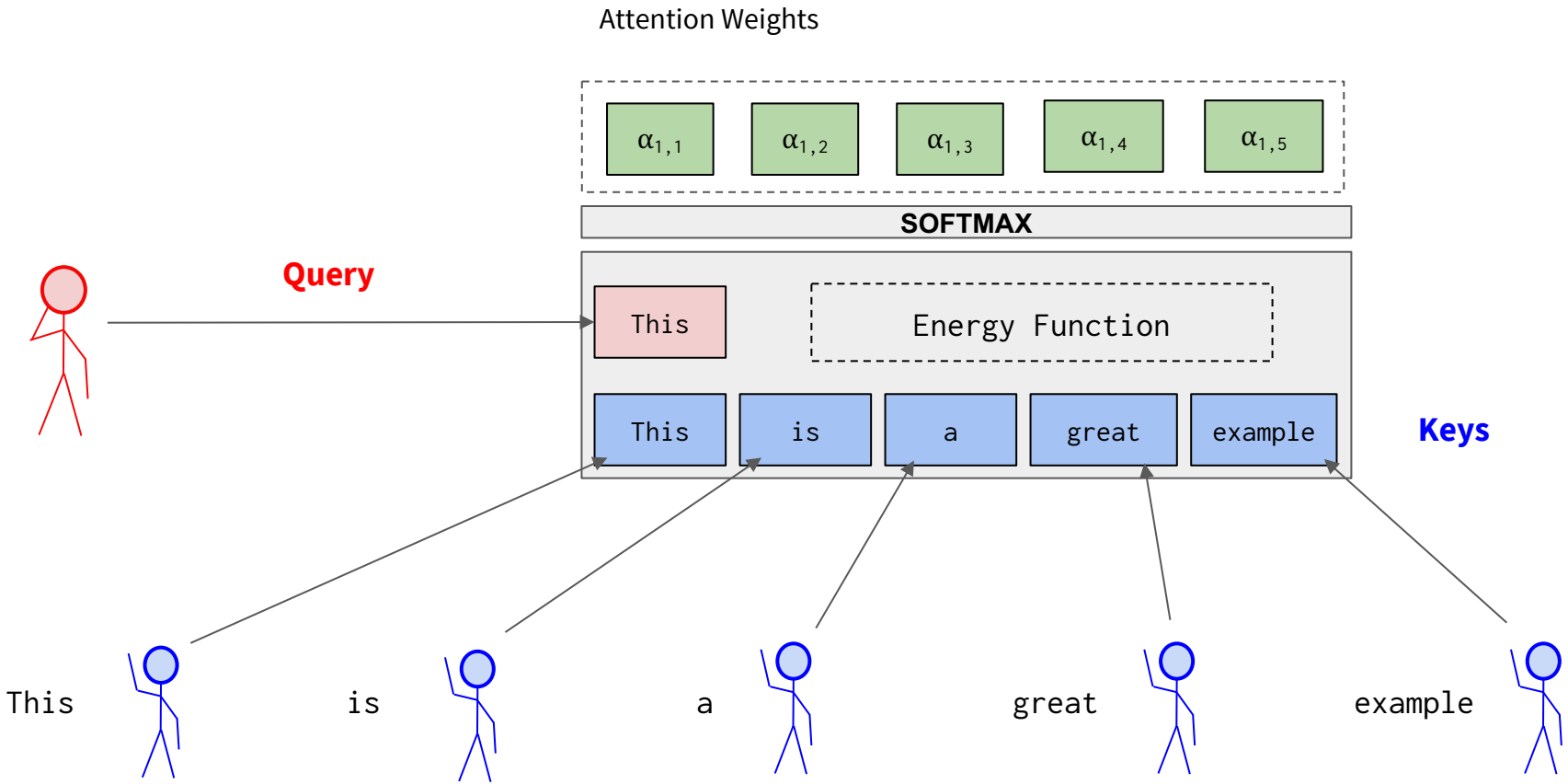




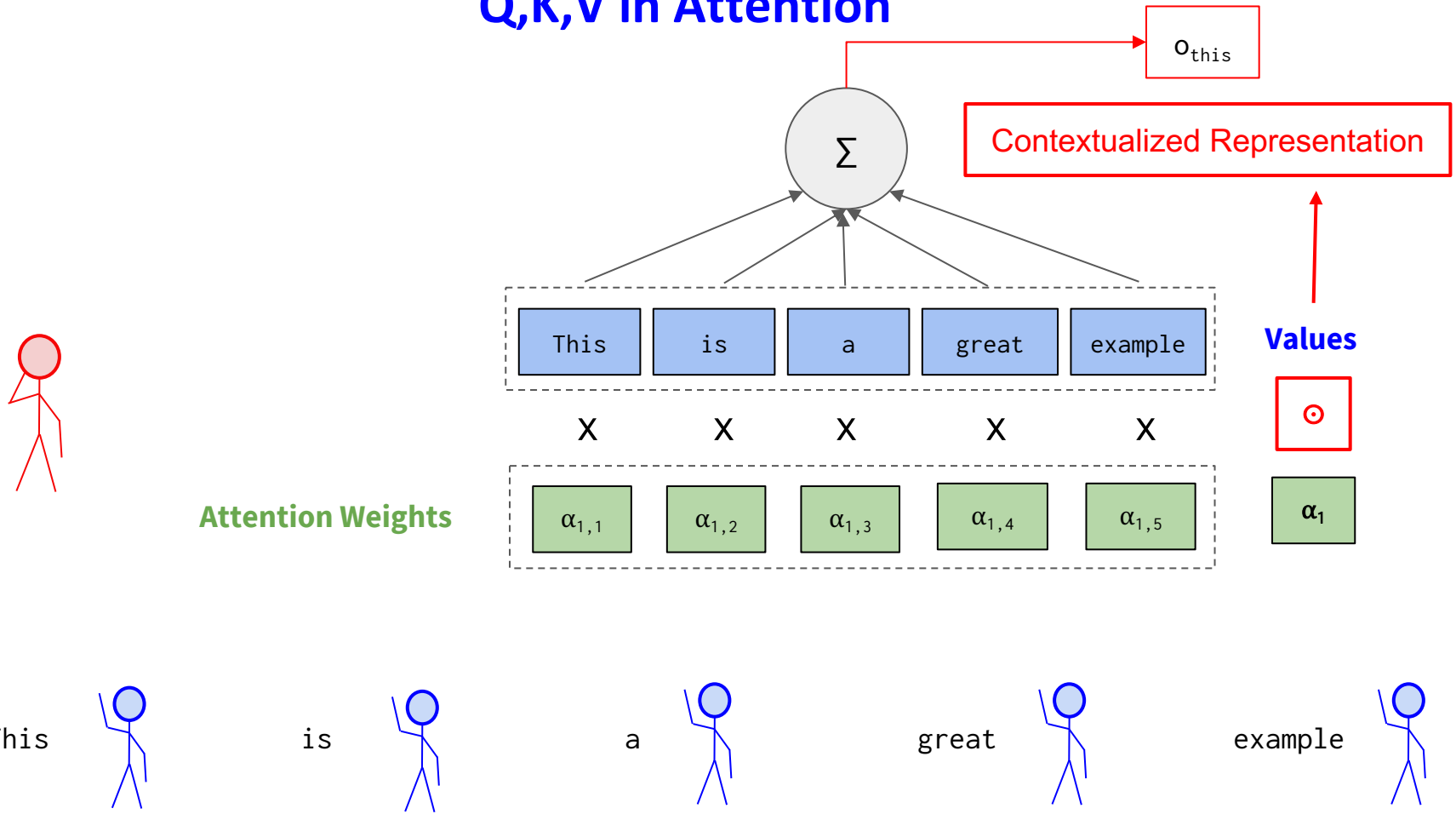
Q,K,V in Attention



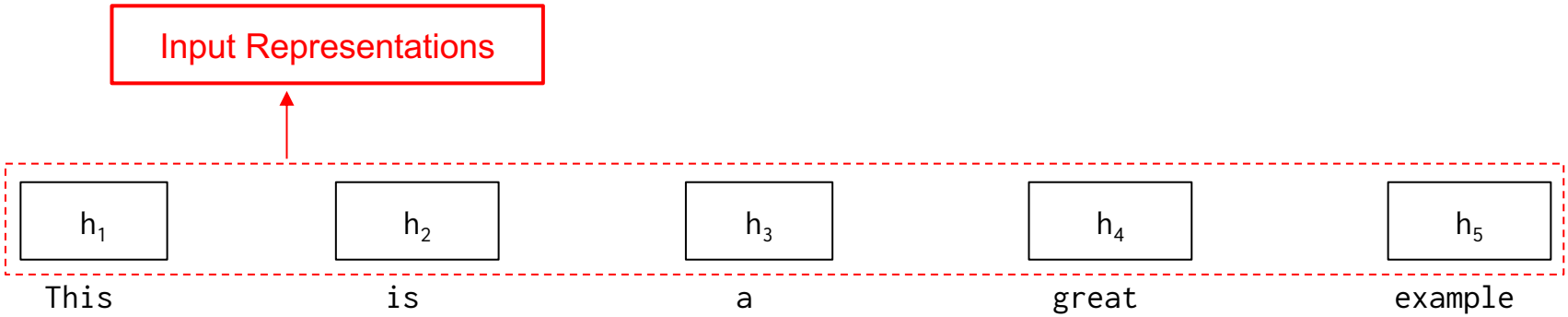
Q,K,V in Attention



Q,K,V in Attention

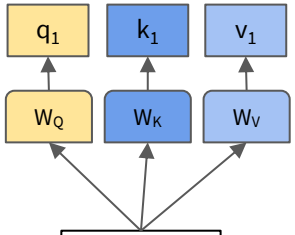


Self Attention



Self Attention

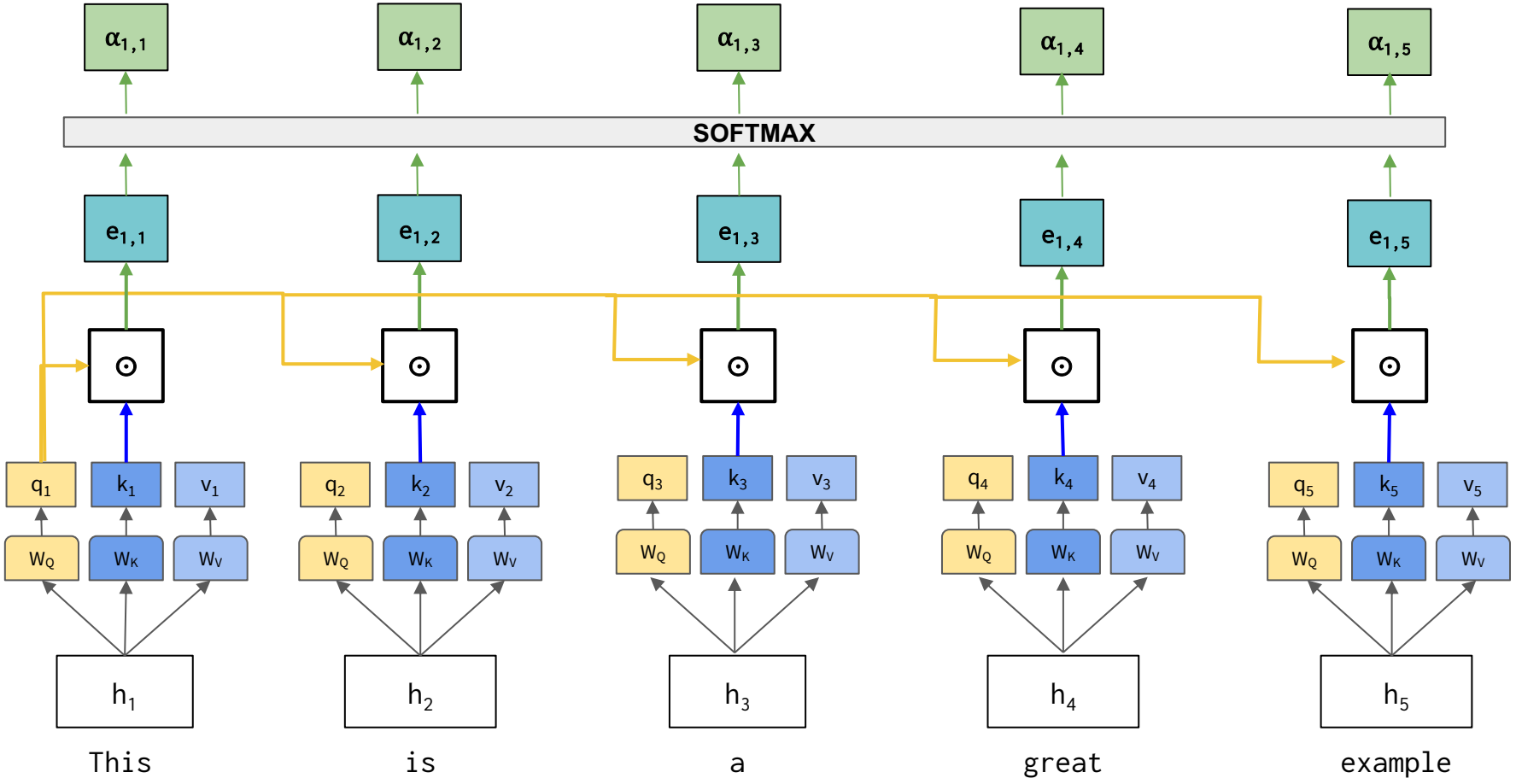
$$q_i = W_Q^T x_i$$
$$k_i = W_K^T x_i$$
$$v_i = W_V^T x_i$$

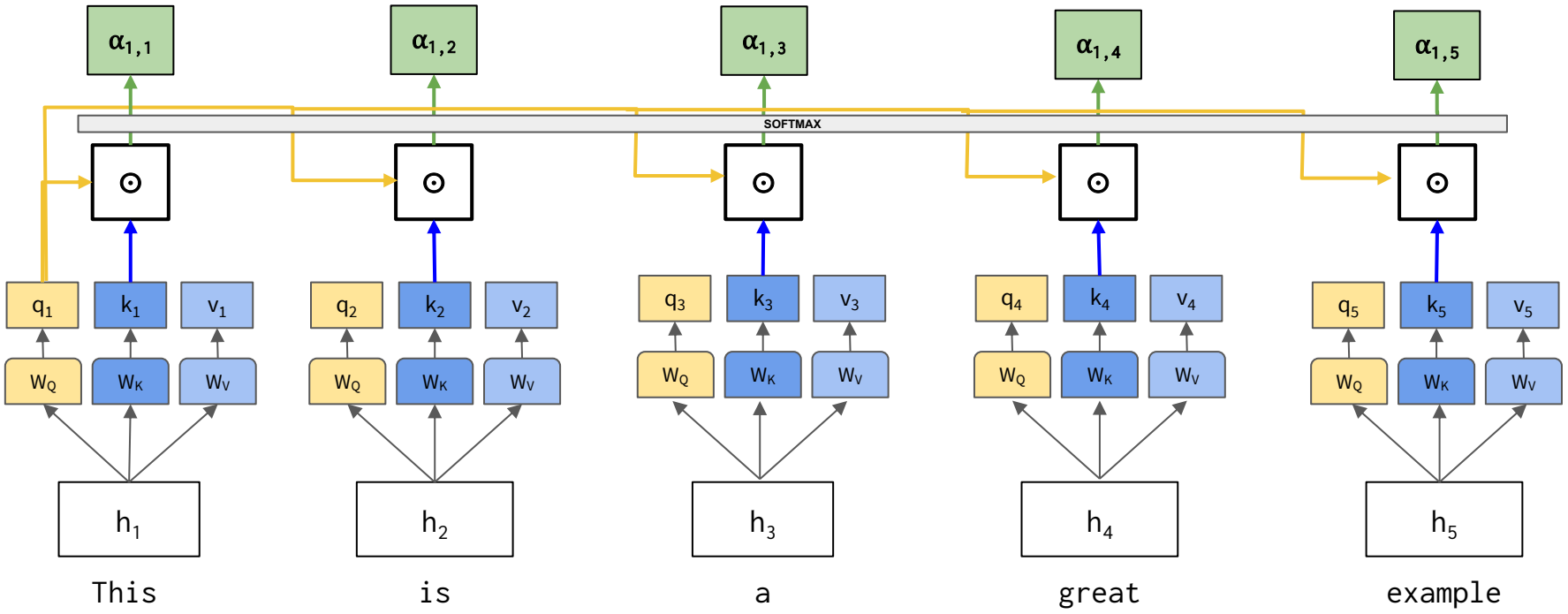


Projection: gives **q,k,v** of dimensions $\mathbf{d_q}$, $\mathbf{d_k}$, $\mathbf{d_v}$ (often the same)

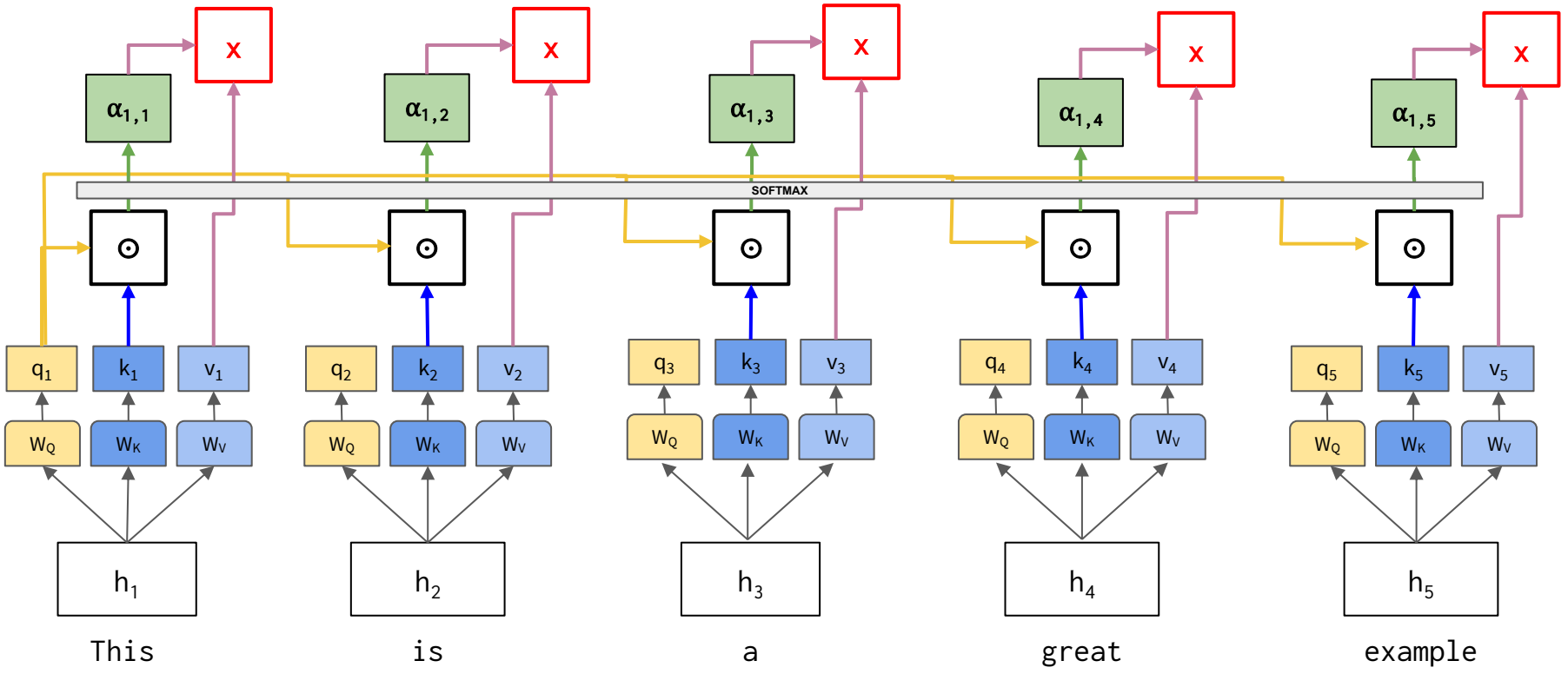
h_1	h_2	h_3	h_4	h_5
This	is	a	great	example

$\alpha_{m,n}$ = How important is token **n** to token **m**'s contextual meaning?

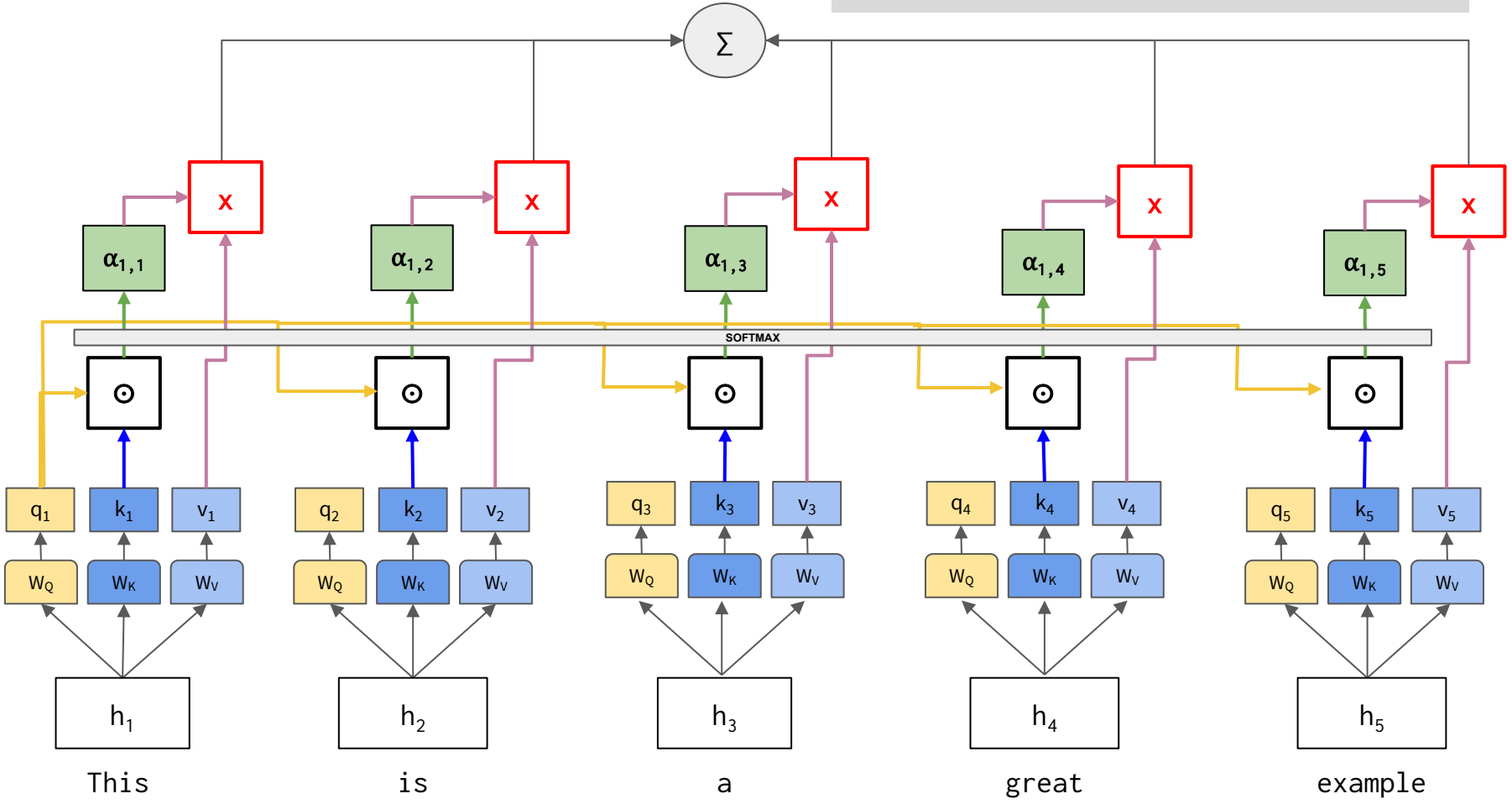


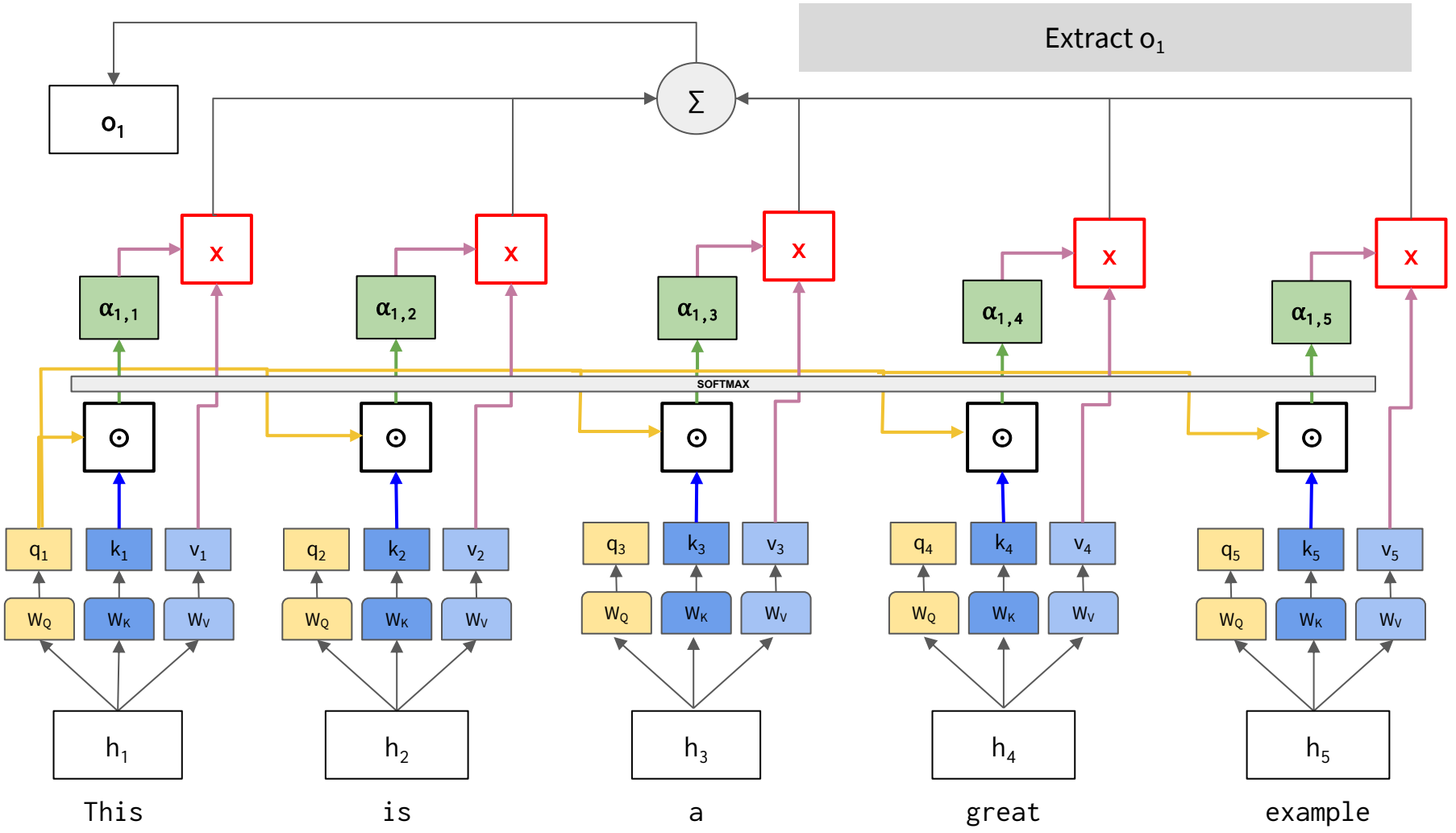

α1,1
α1,2
α1,3
α1,4
α1,5
SOFTMAX
q1
k1
v1
q2
k2
v2
q3
k3
v3
q4
k4
v4
q5
k5
v5
WQ
WK
WV
h1
h2
h3
h4
h5
This
is
a
great
example

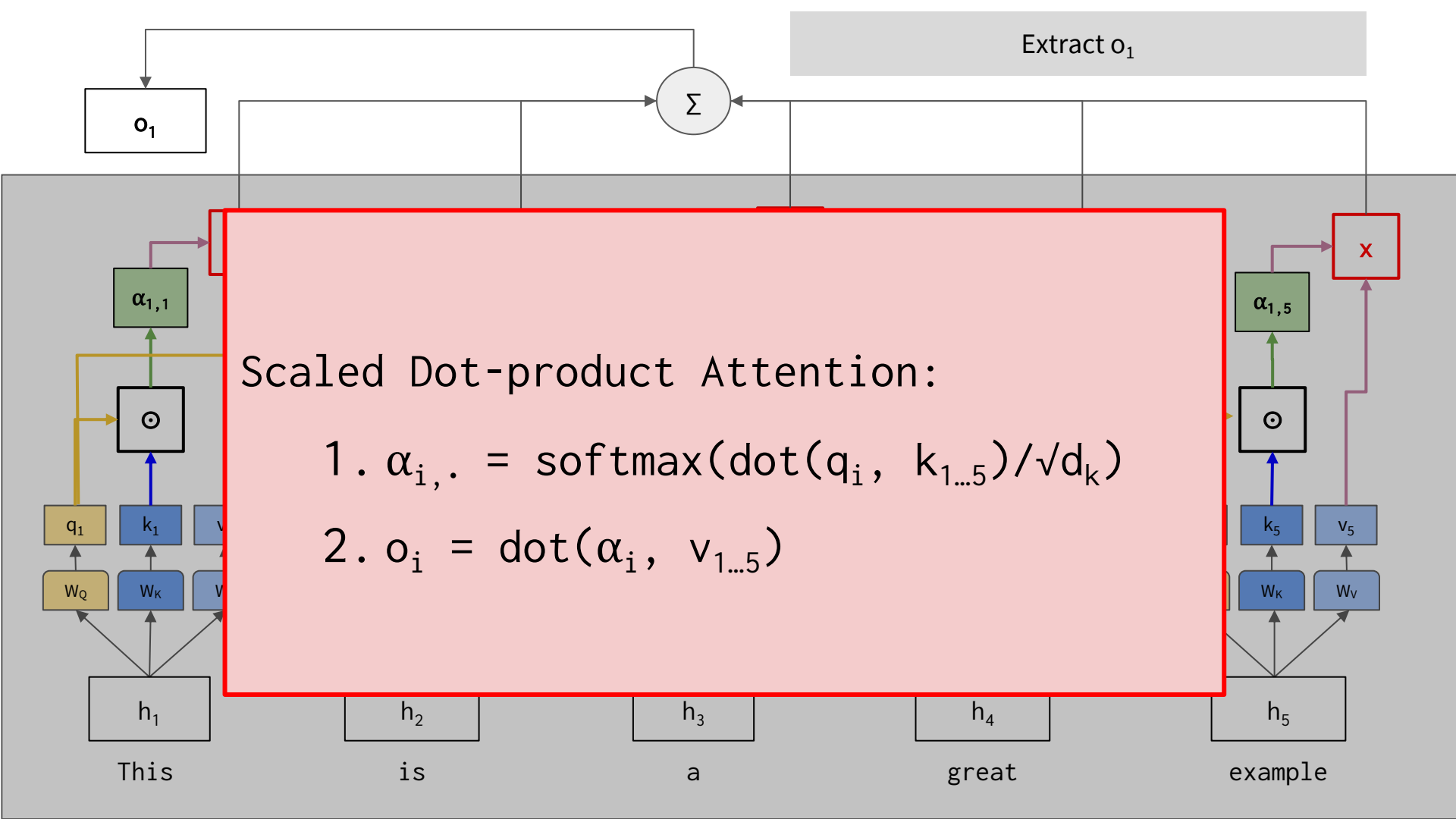
Multiply each $\alpha_{1,i}$ with v_i

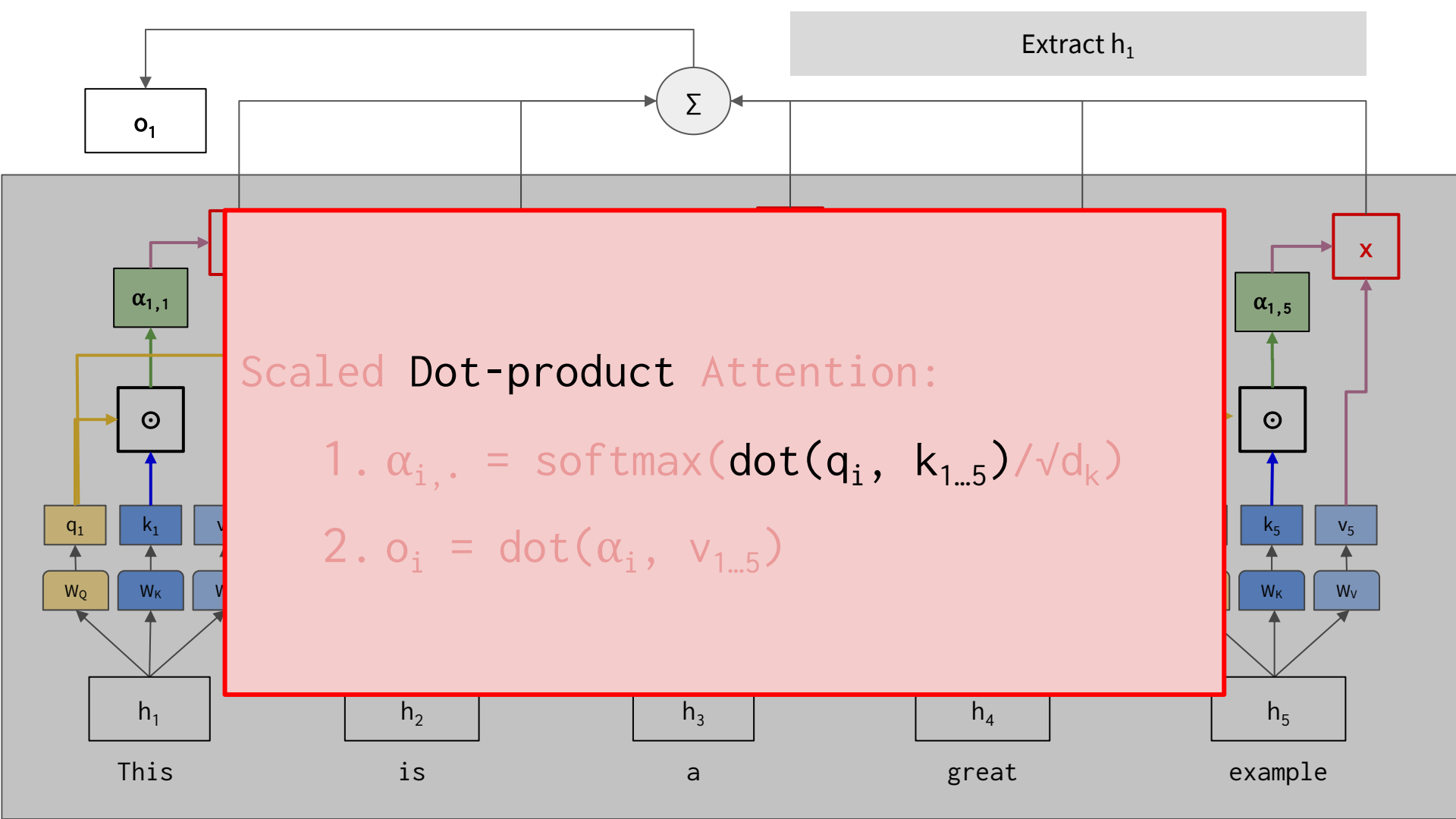


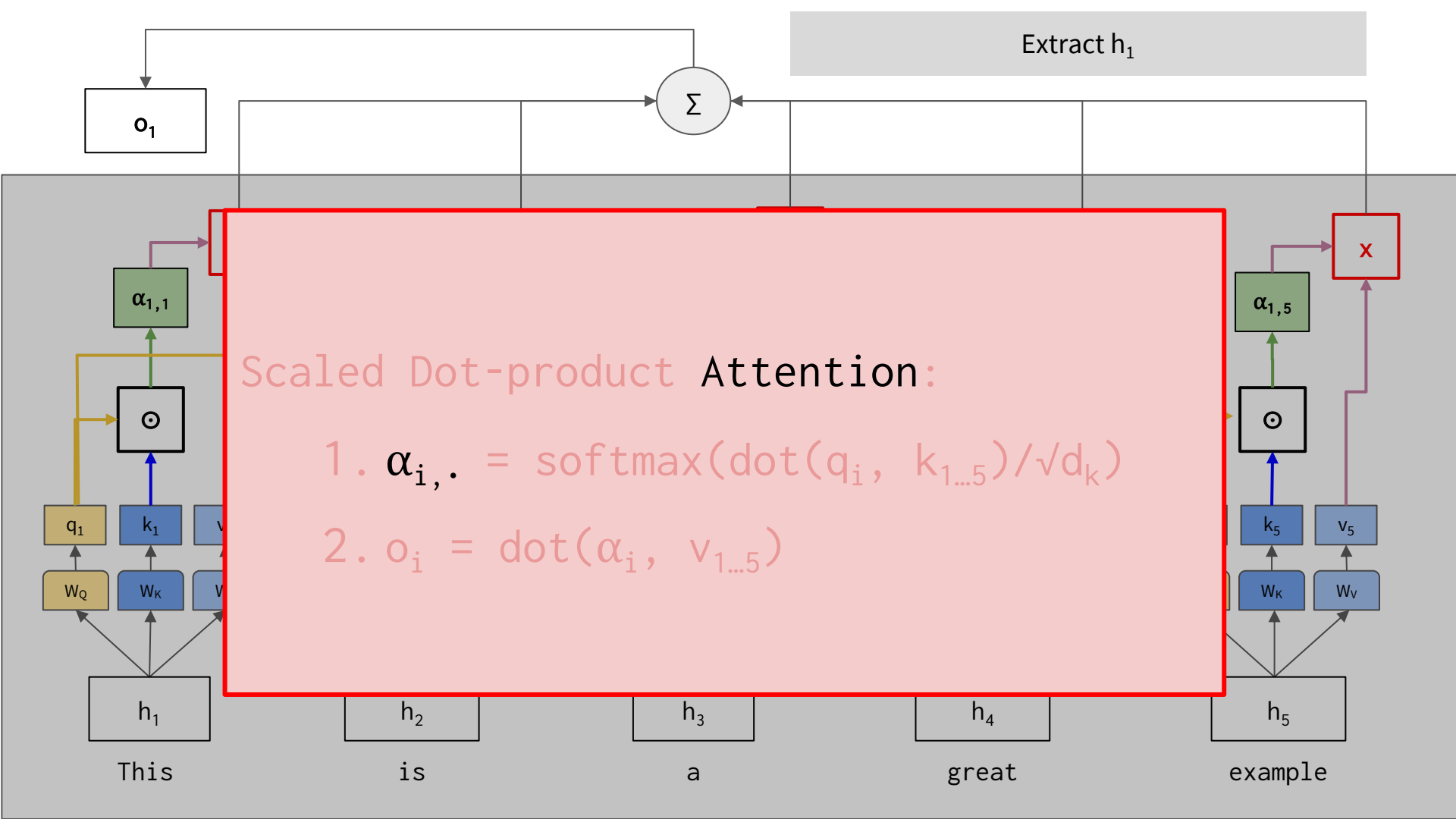
Sum all those multiples up

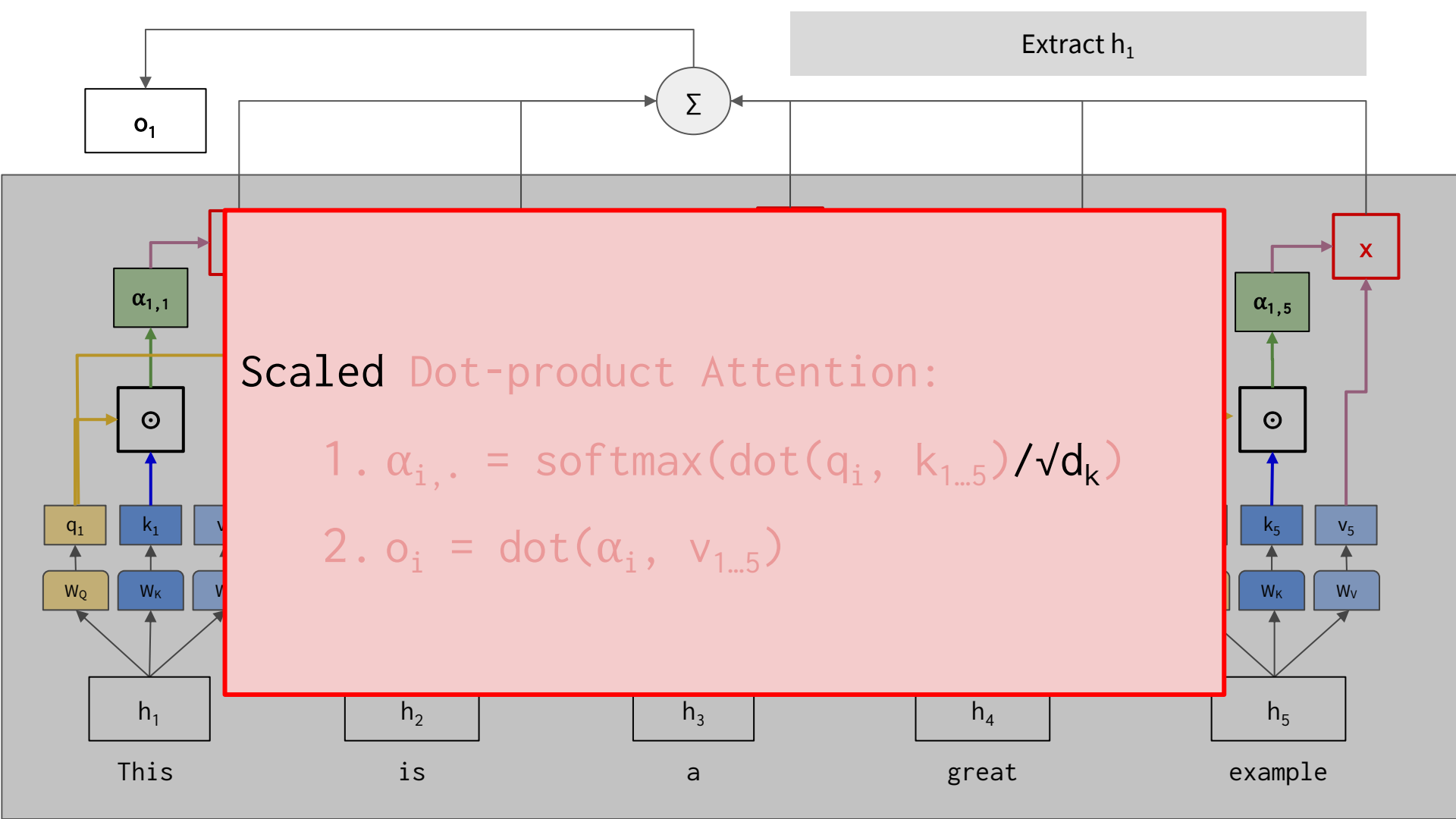


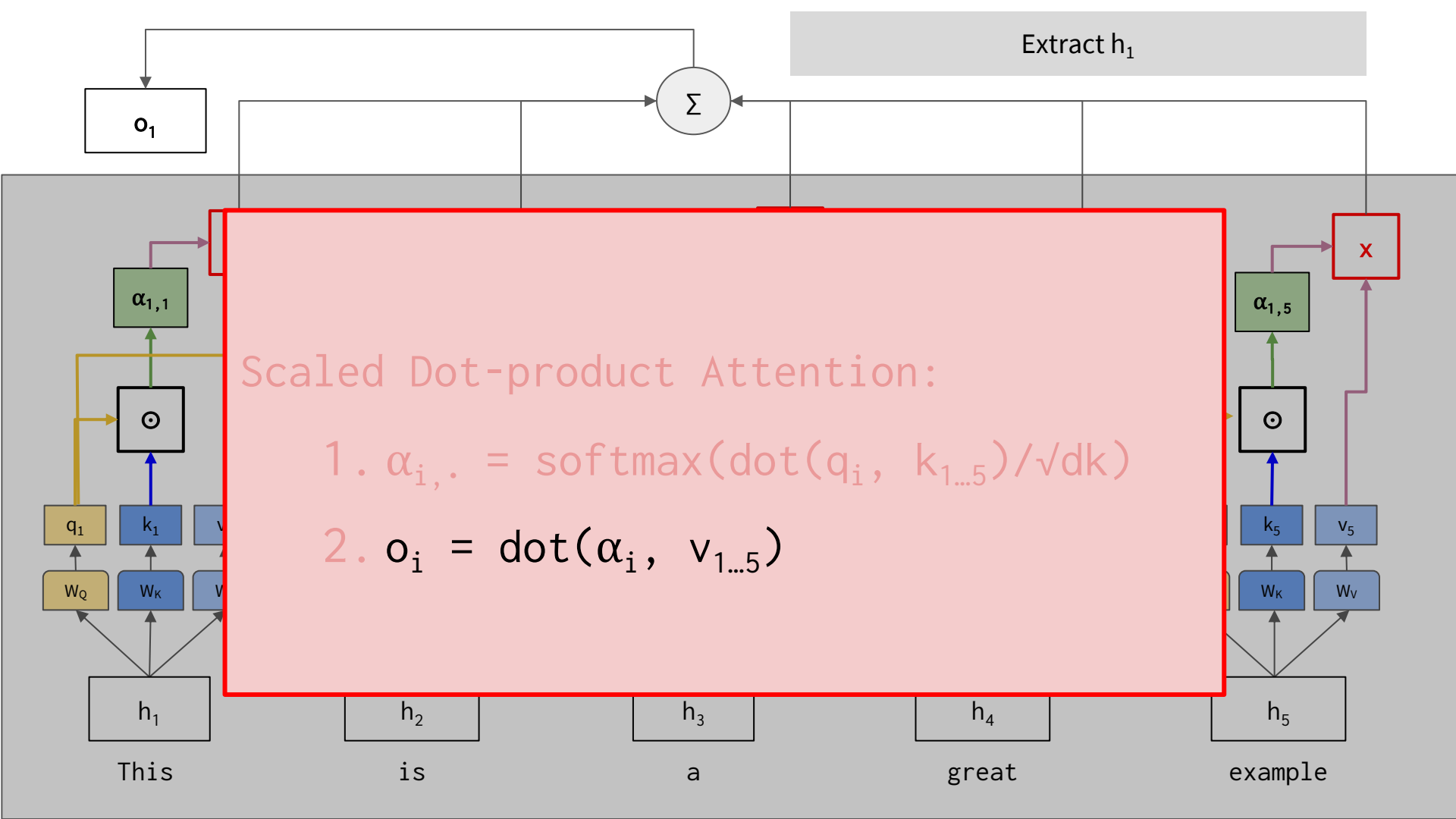

Extract o_1
o_1
Σ
x
x
x
x
x
$\alpha_{1,1}$
$\alpha_{1,2}$
$\alpha_{1,3}$
$\alpha_{1,4}$
$\alpha_{1,5}$
SOFTMAX
⊙
⊙
⊙
⊙
⊙
q_1 k_1 v_1
q_2 k_2 v_2
q_3 k_3 v_3
q_4 k_4 v_4
q_5 k_5 v_5
W_Q W_K W_V
h_1
h_2
h_3
h_4
h_5
This
is
a
great
example

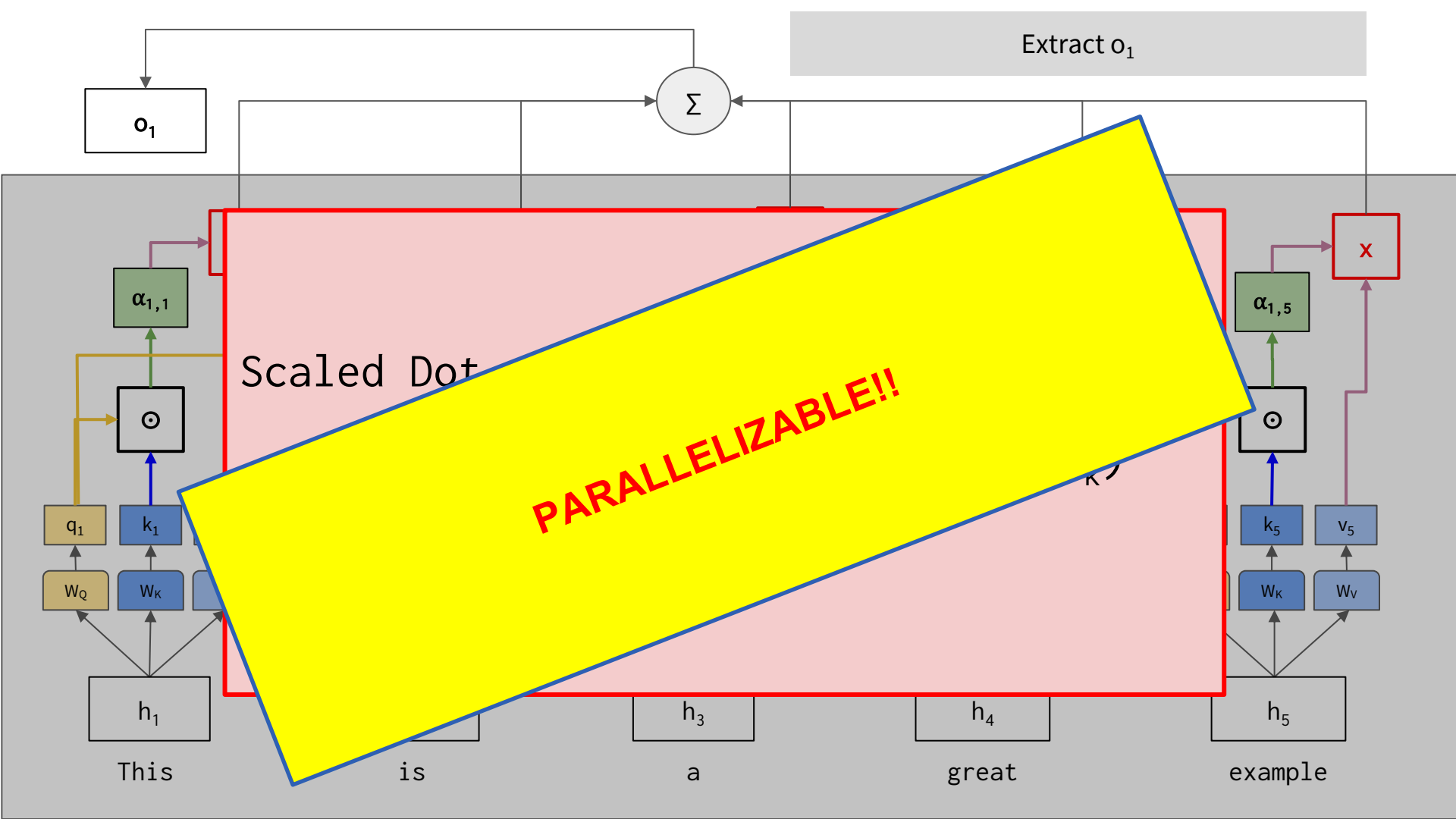

Extract o_1
o_1
Σ
Scaled Dot-product Attention:
1. $\alpha_{i,.} = \text{softmax}(\text{dot}(q_i, k_{1\ldots5})/\sqrt{d_k})$
2. $o_i = \text{dot}(\alpha_i, v_{1\ldots5})$
$\alpha_{1,1}$
$\alpha_{1,5}$
X
q_1
k_1
k_5
v_5
W_Q
W_K
W_K
W_V
h_1
h_2
h_3
h_4
h_5
This
is
a
great
example


Extract h_1
o_1
Σ
$\alpha_{1,1}$
$\alpha_{1,5}$
x
Scaled Dot-product Attention:
1. $\alpha_{i,.}$ = softmax(dot(q_i, $k_{1\ldots5}$)/$\sqrt{d_k}$)
2. o_i = dot(α_i, $v_{1\ldots5}$)
q_1
k_1
k_5
v_5
W_Q
W_K
W_K
W_V
h_1
h_2
h_3
h_4
h_5
This
is
a
great
example


Extract h_1
o_1
Σ
$\alpha_{1,1}$
$\alpha_{1,5}$
x
⊙
⊙
q_1
k_1
k_5
v_5
W_Q
W_K
W_K
W_V
h_1
h_2
h_3
h_4
h_5
This
is
a
great
example
Scaled Dot-product Attention:
1. $\alpha_{i,.} = \text{softmax}(\text{dot}(q_i, k_{1\ldots5})/\sqrt{d_k})$
2. $o_i = \text{dot}(\alpha_i, v_{1\ldots5})$


Extract h_1
o_1
Σ
Scaled Dot-product Attention:
1. $\alpha_{i,.} = \text{softmax}(\text{dot}(q_i, k_{1\ldots5})/\sqrt{d_k})$
2. $o_i = \text{dot}(\alpha_i, v_{1\ldots5})$
$\alpha_{1,1}$
$\alpha_{1,5}$
x
⊙
⊙
q_1
k_1
k_5
v_5
W_Q
W_K
W_K
W_V
h_1
h_2
h_3
h_4
h_5
This
is
a
great
example


Extract h_1
o_1
Σ
$\alpha_{1,1}$
$\alpha_{1,5}$
x
Scaled Dot-product Attention:
1. $\alpha_{i,.}$ = softmax(dot(q_i, $k_{1\ldots5}$)/√dk)
2. o_i = dot(α_i, $v_{1\ldots5}$)
q_1
k_1
k_5
v_5
W_Q
W_K
W_K
W_V
h_1
h_2
h_3
h_4
h_5
This
is
a
great
example


Extract o_1
o_1
Σ
Scaled Dot
PARALLELIZABLE!!
X
$\alpha_{1,1}$
$\alpha_{1,5}$
⊙
⊙
q_1
k_1
k_5
v_5
W_Q
W_K
W_K
W_V
h_1
h_3
h_4
h_5
This
is
a
great
example

Example

- $q,k,v_1 = [1, 2, 3, 4]$

  $q,k,v_2 = [4, 5, 9, 1]$

  $q,k,v_3 = [6, 2, 1, 4]$
- $e_1 = q_1 k_1^T / \sqrt{4} = 15.0$

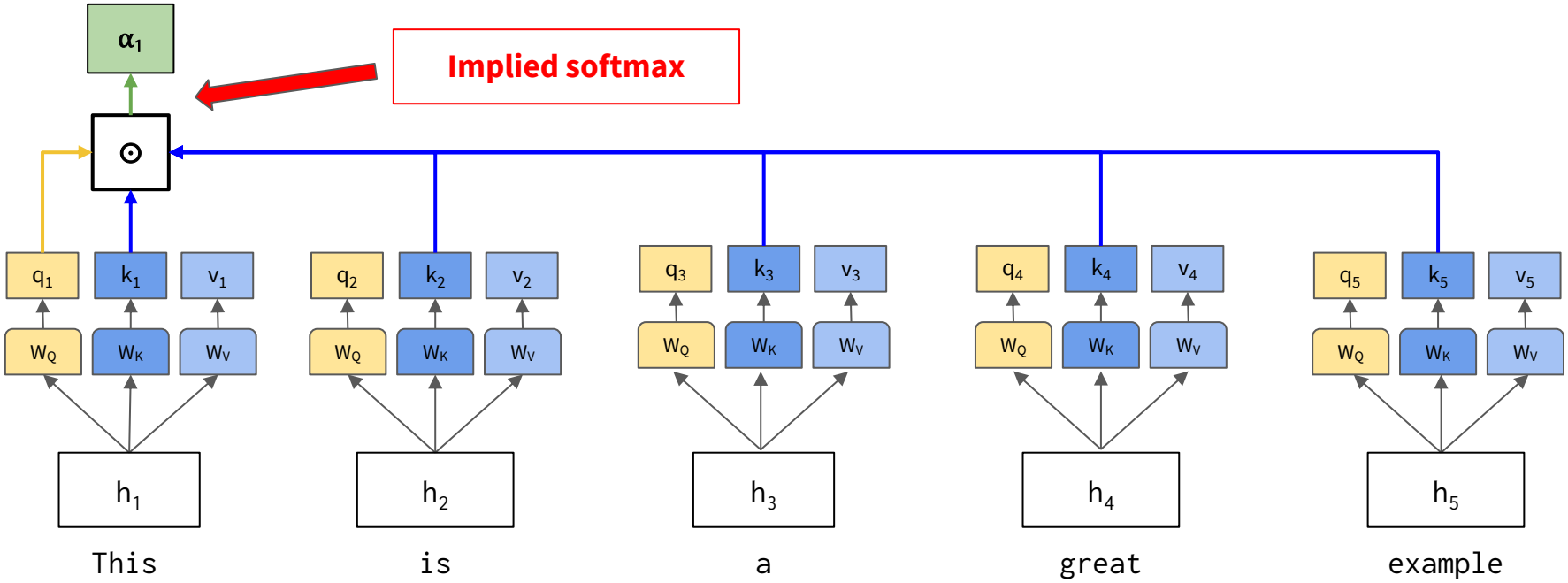
  $e_2 = q_1 k_2^T / \sqrt{4} = 22.5$

  $e_3 = q_1 k_3^T / \sqrt{4} = 14.5$
- $\alpha_1 = \text{softmax}(\mathbf{e}) = [0.00055, 0.99911, 0.00033]$
- $\mathbf{o_1} = \alpha_1^T \mathbf{V} = [3.99, 4.99, 8.99, 1.00]$

  $\mathbf{V}$ is the 3x4 matrix of all values

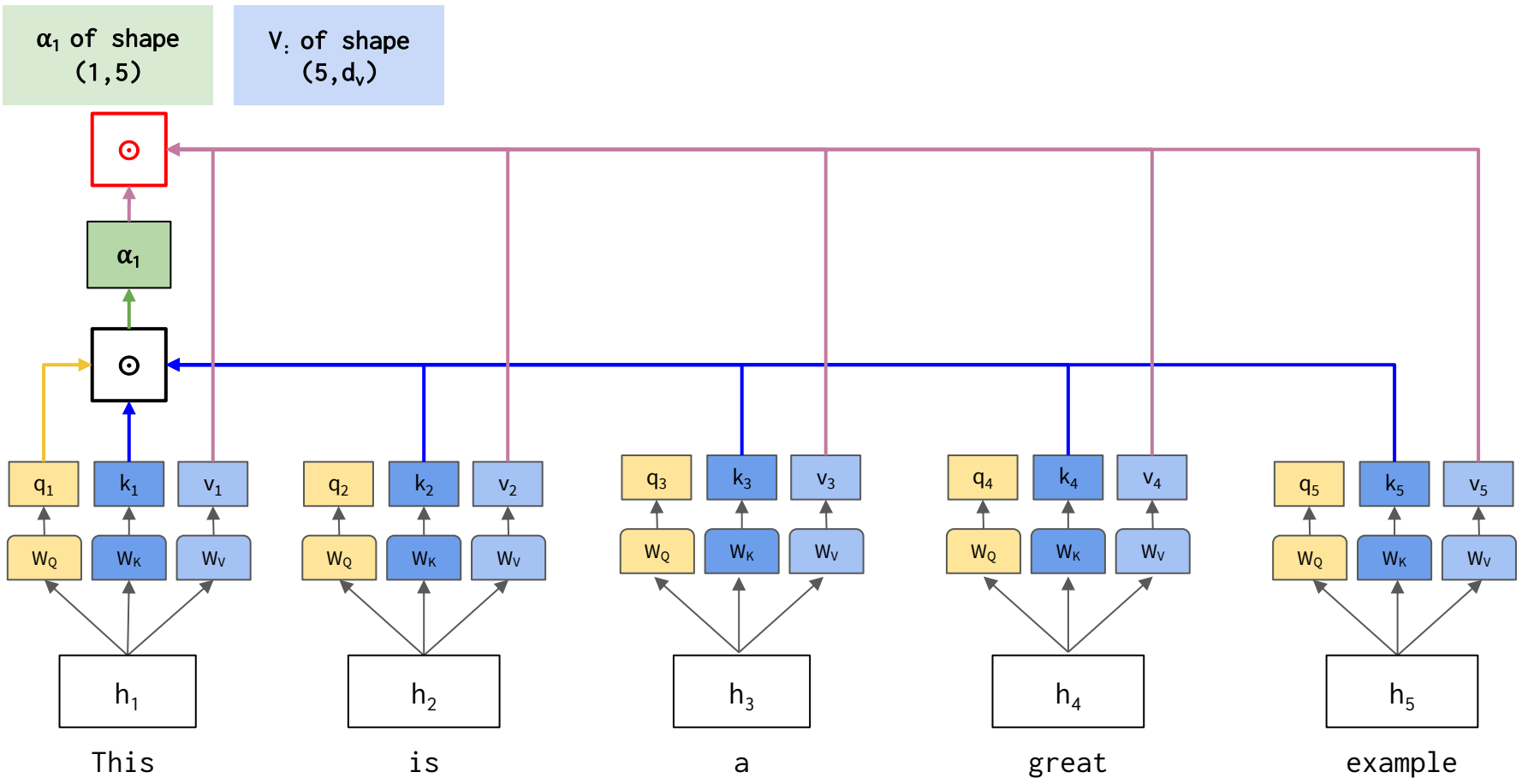
Self Attention

α_1 of shape (1,5)



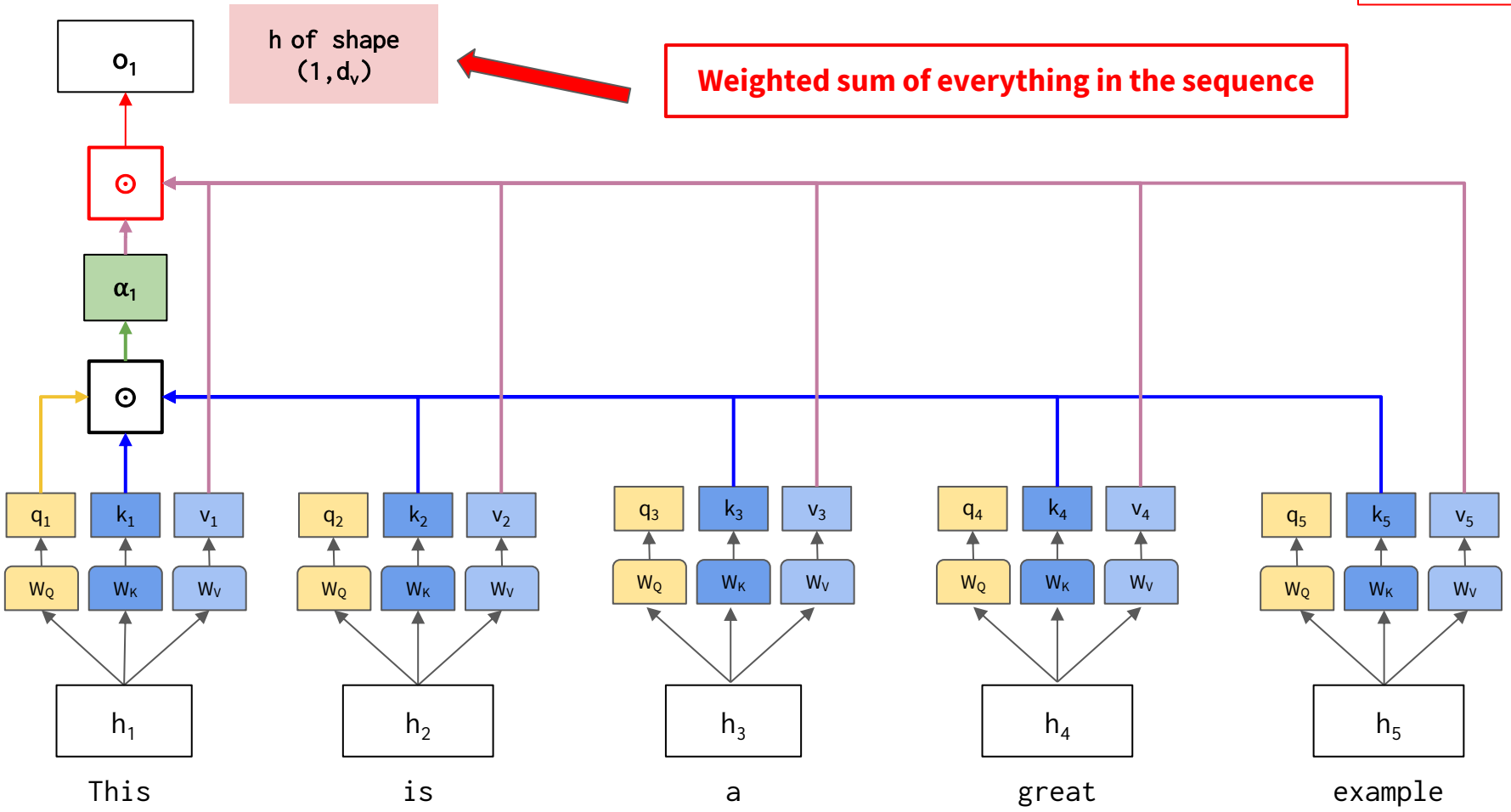
Self Attention

*Implied softmax



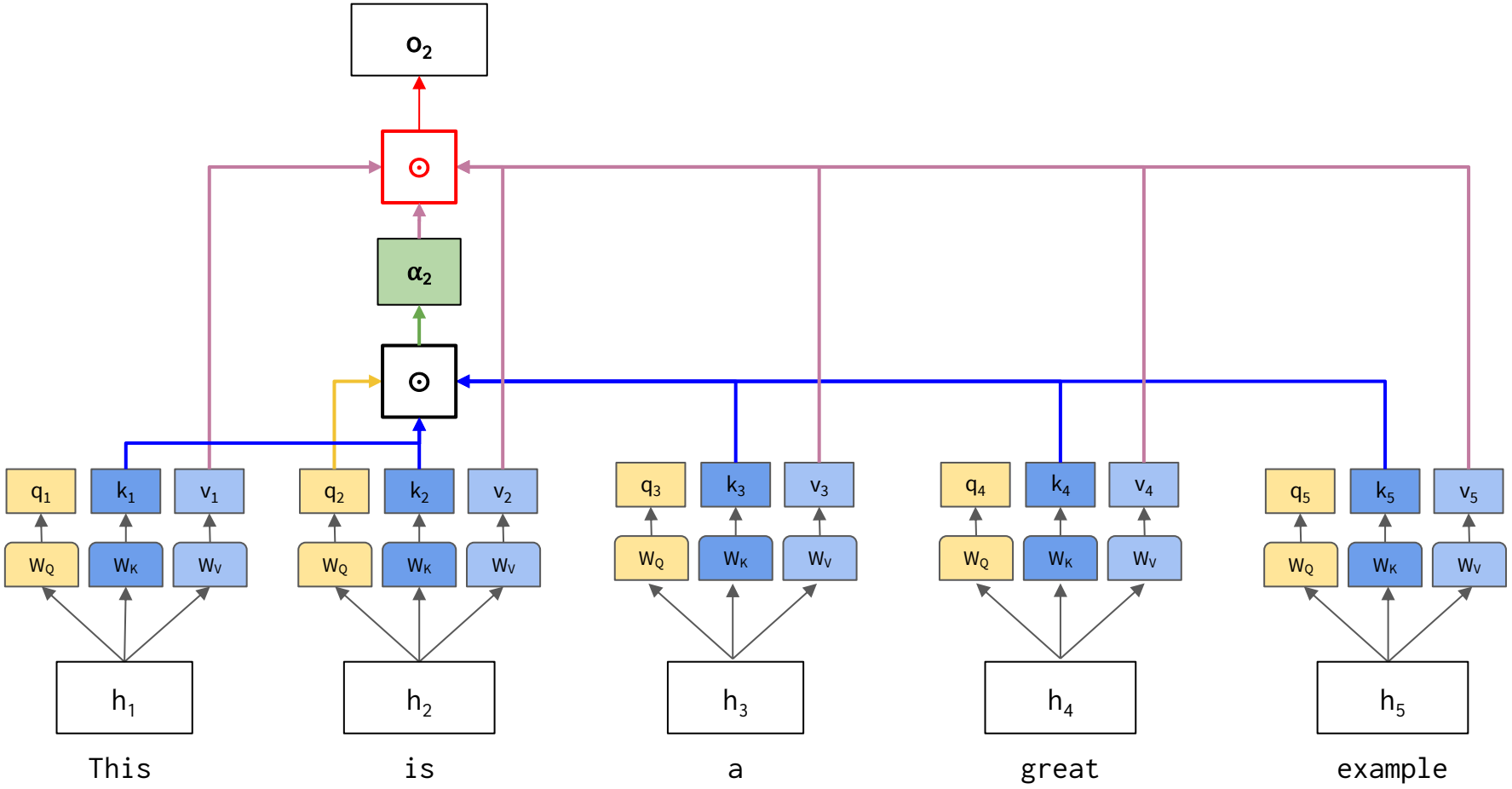
Self Attention

*Implied softmax



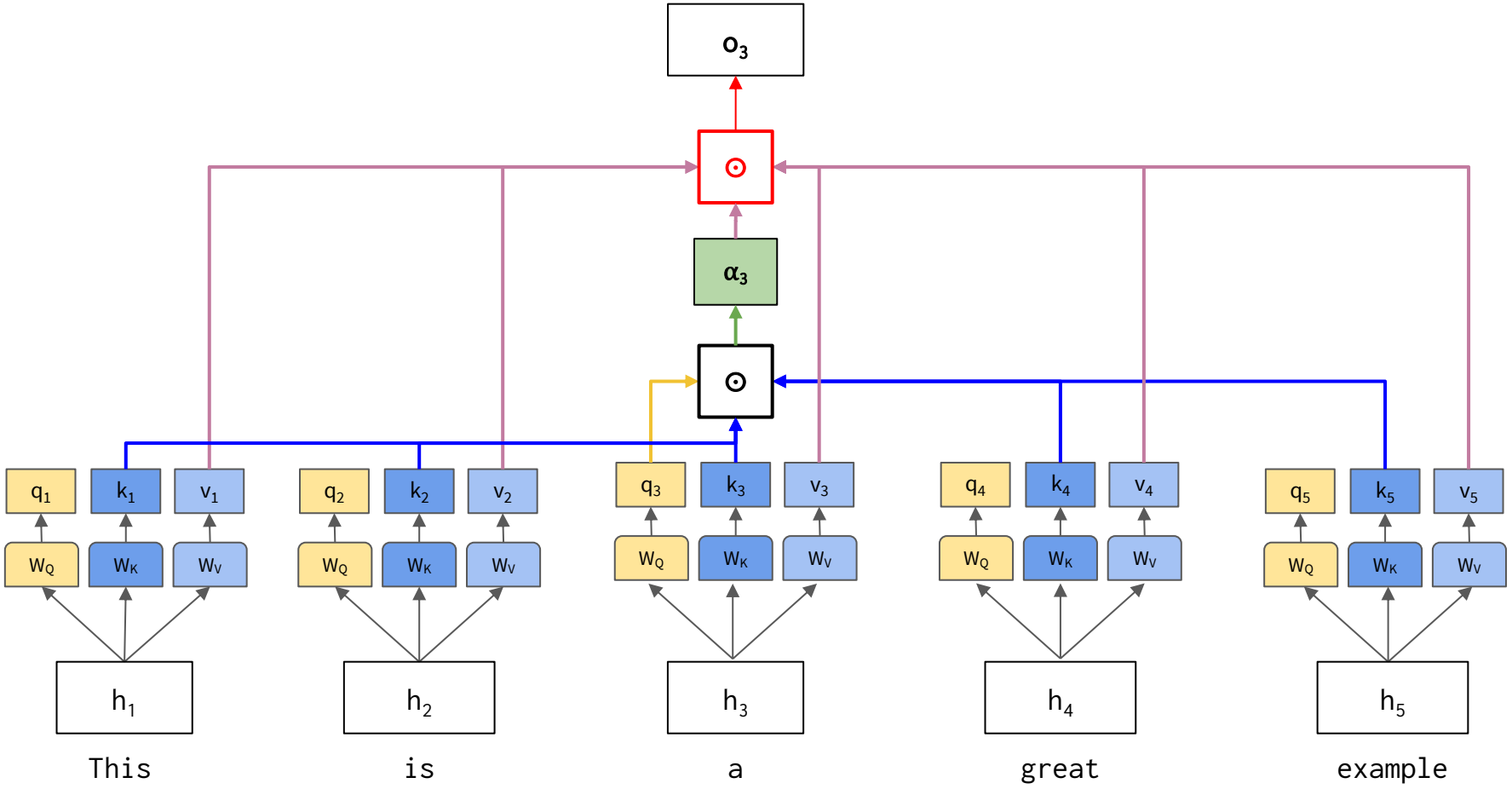
Self Attention

*Implied softmax



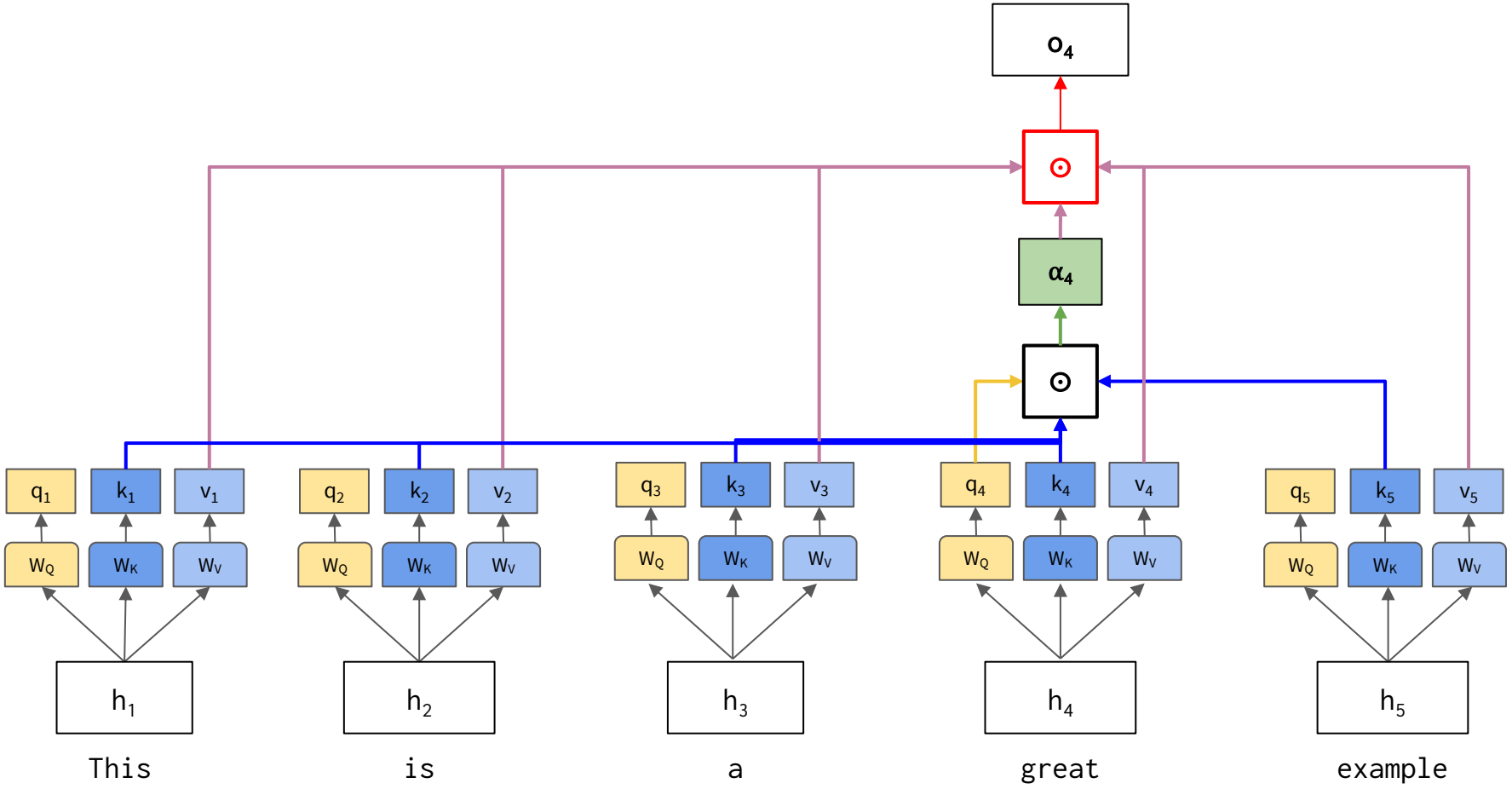
Self Attention

*Implied softmax



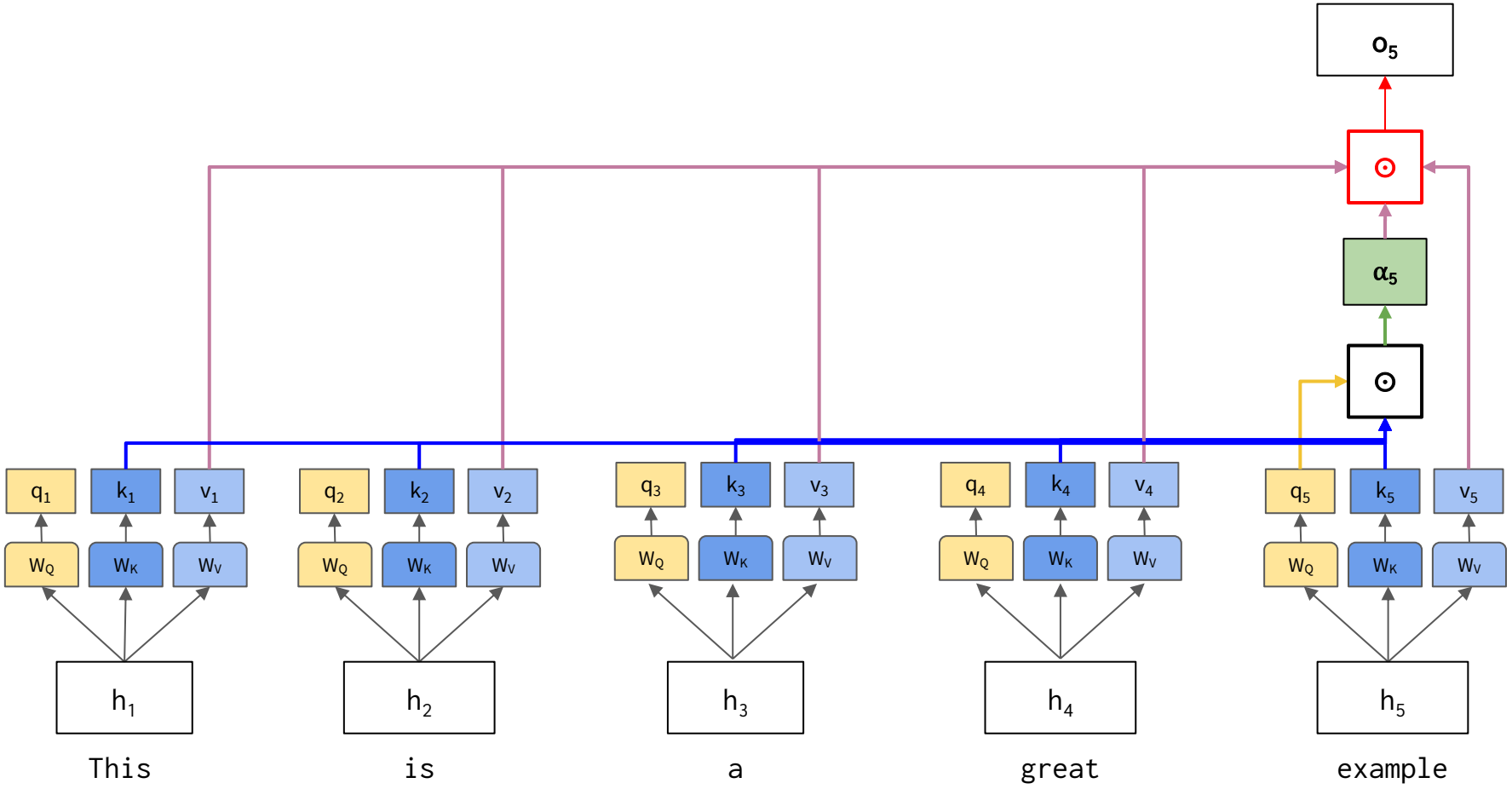
Self Attention

*Implied softmax



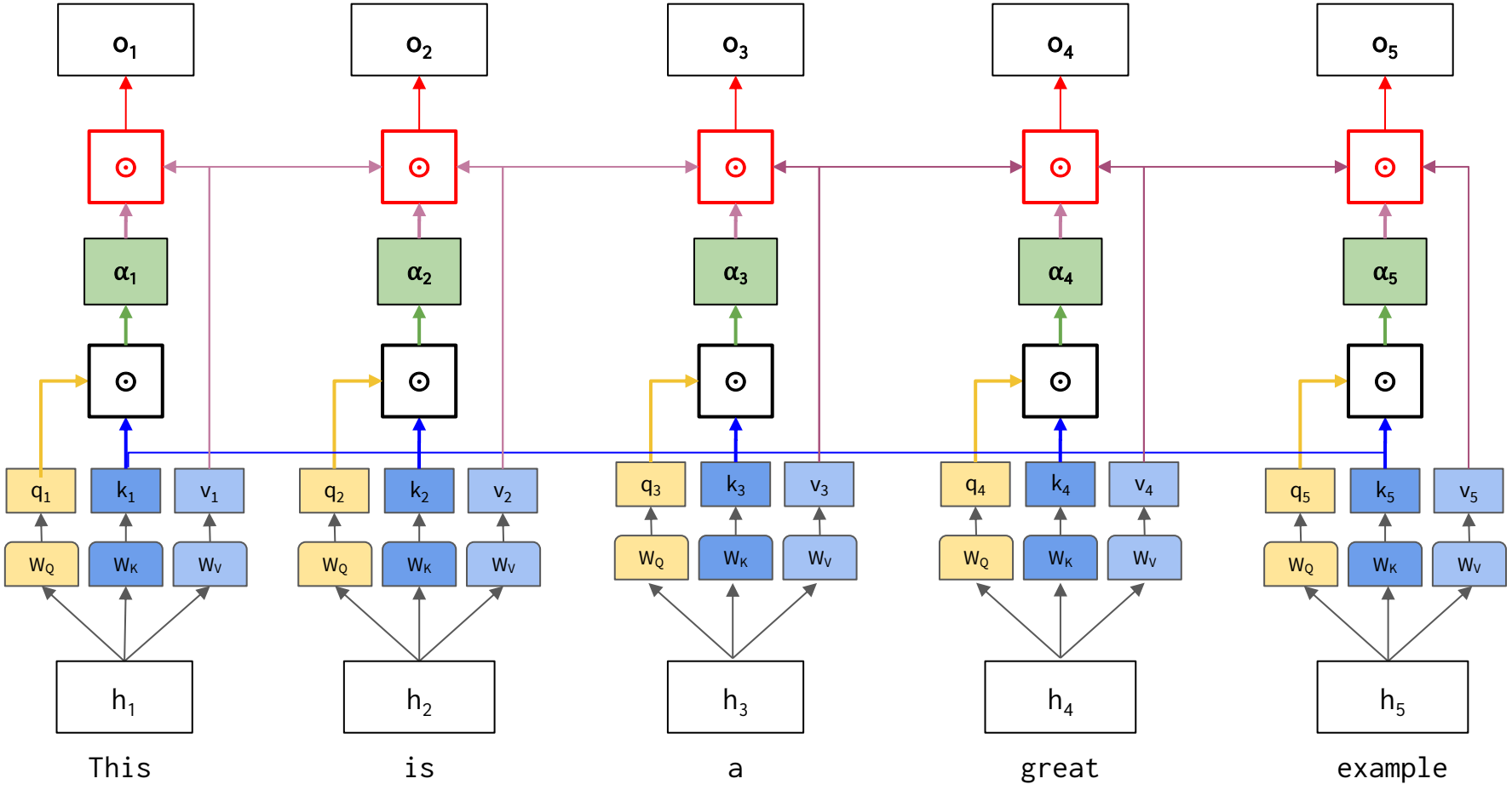
Self Attention

*Implied softmax

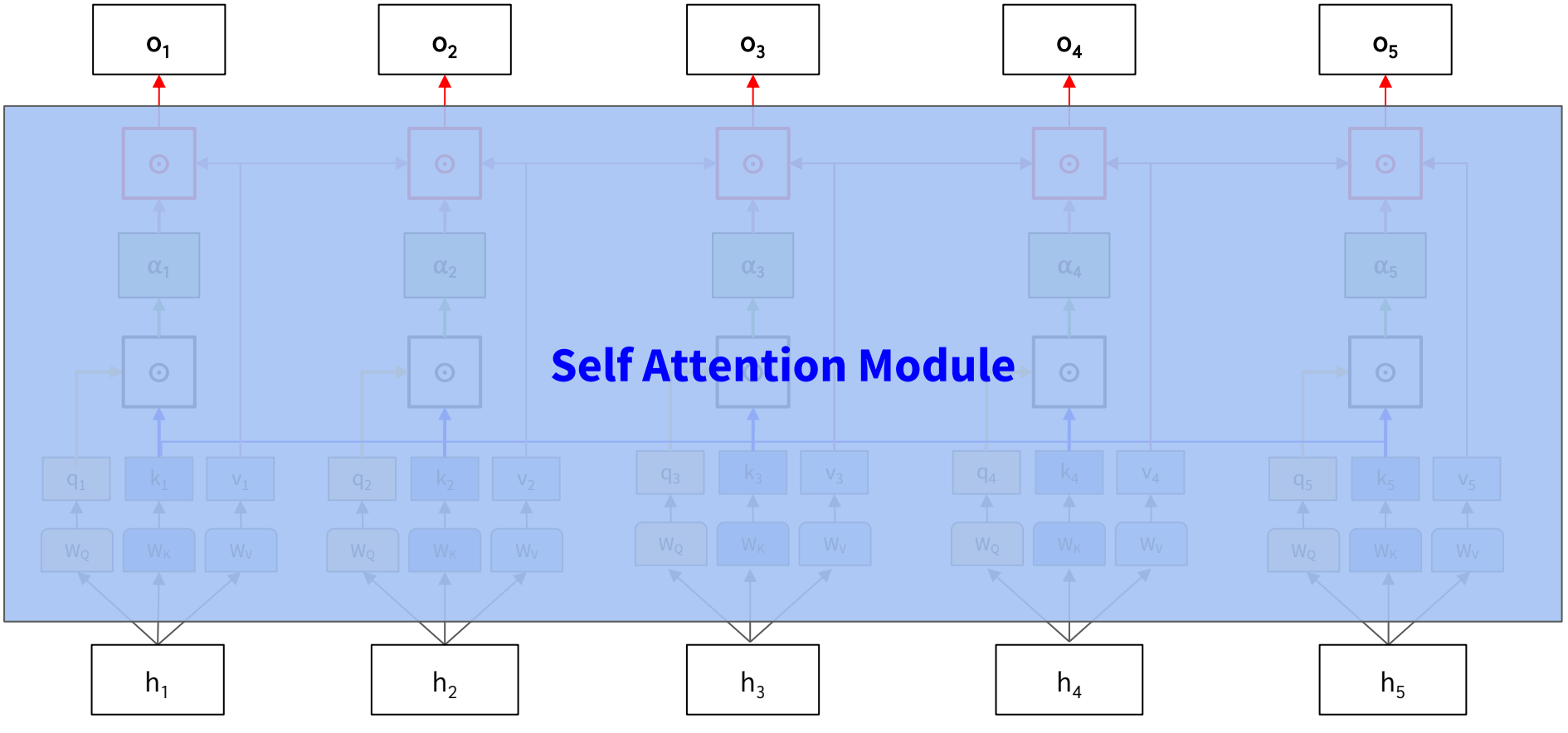


Self Attention

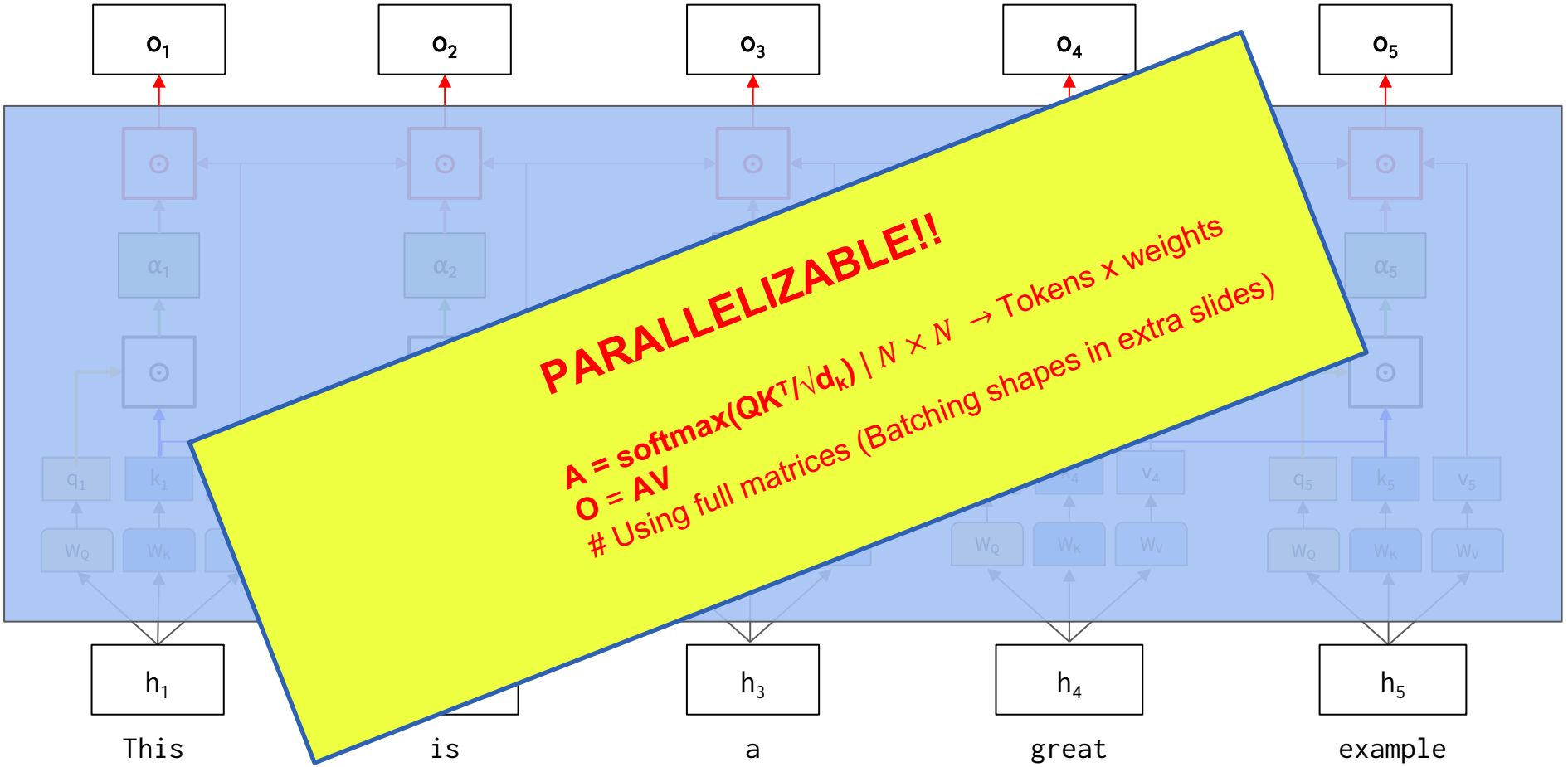
*Implied softmax



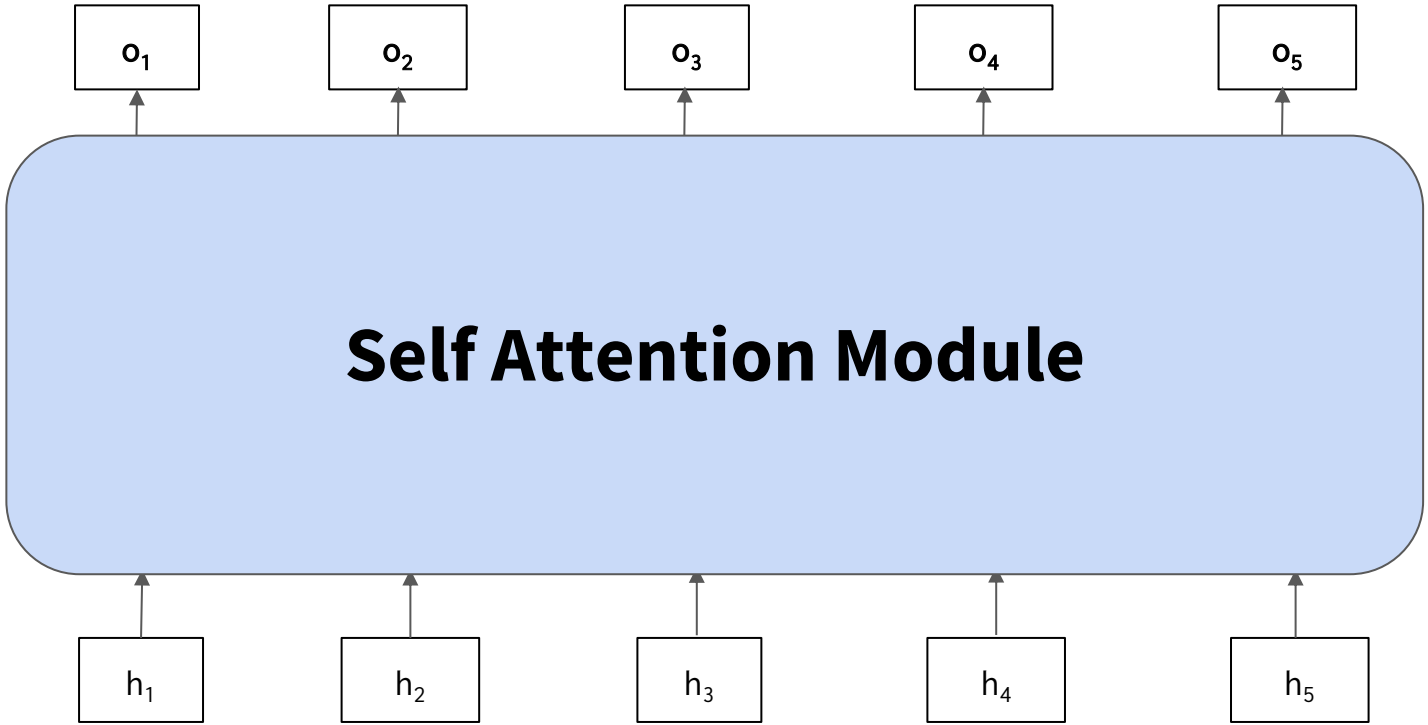
*Implied softmax



*Implied softmax



Single Head Self Attention



Poll 1 (@1125)

Which of the following are true about self attention? (Select all that apply)

a. To calculate attention weights for input **h_i**, you would use key **k_i,** and all queries

b. To calculate attention weights for input **h_i**, you would use query **q_i,** and all keys

c. The energy function is scaled to bring attention weights in the range of [0,1]

d. The energy function is scaled to allow for numerical stability

Poll 1 (@1125)

Which of the following are true about self attention? (Select all that apply)

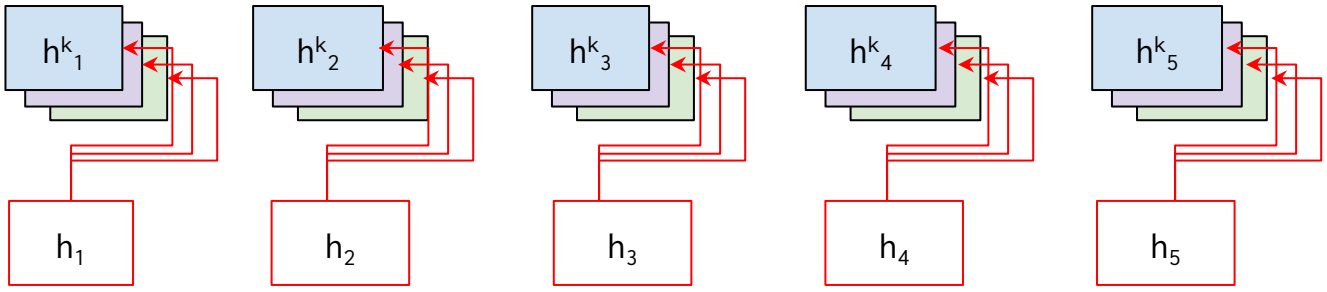
a. To calculate attention weights for input **h_i**, you would use key **k_i,** and all queries

b. To calculate attention weights for input h_i, you would use query q_i, and all keys

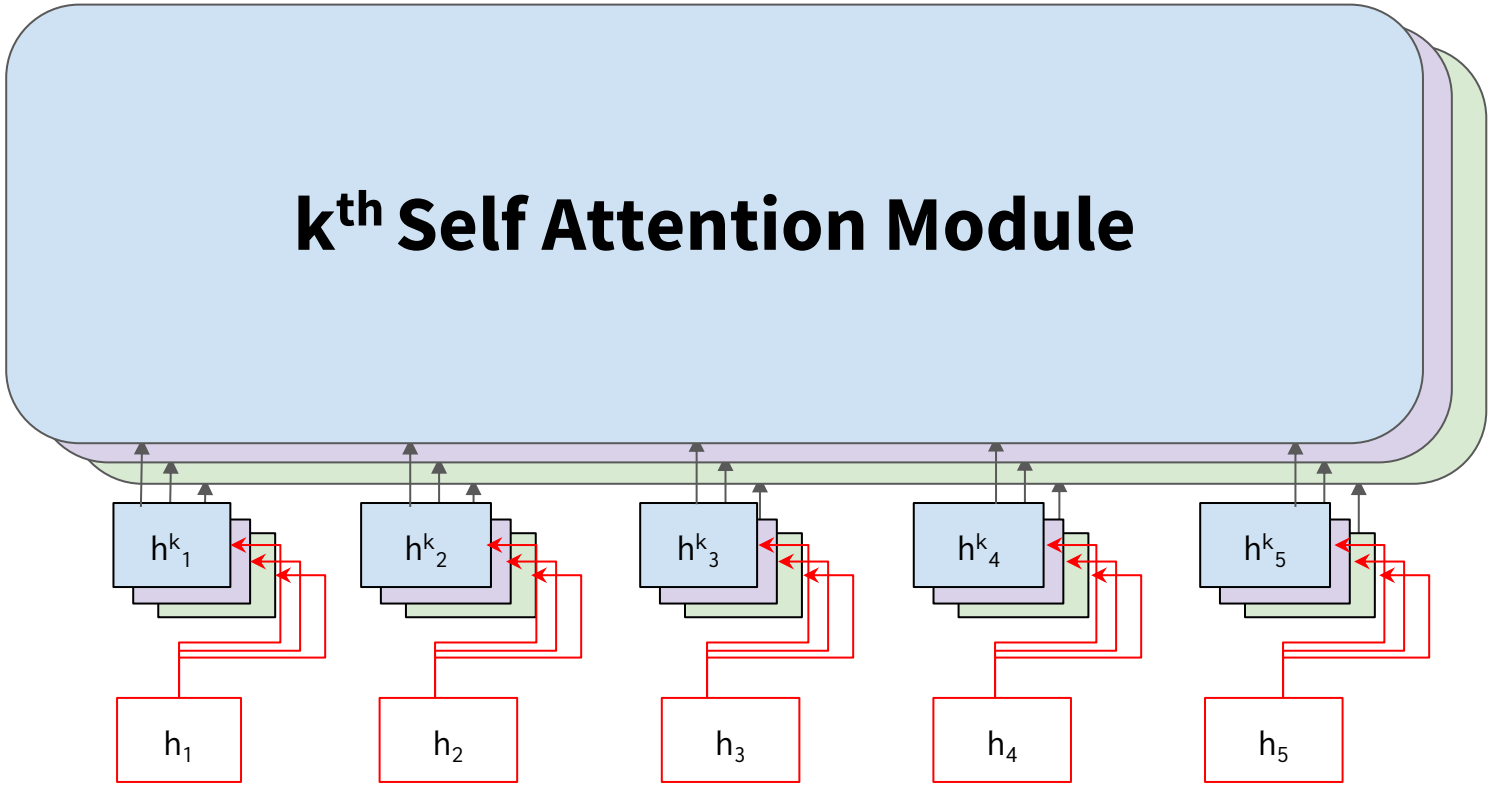
c. The energy function is scaled to bring attention weights in the range of [0,1]

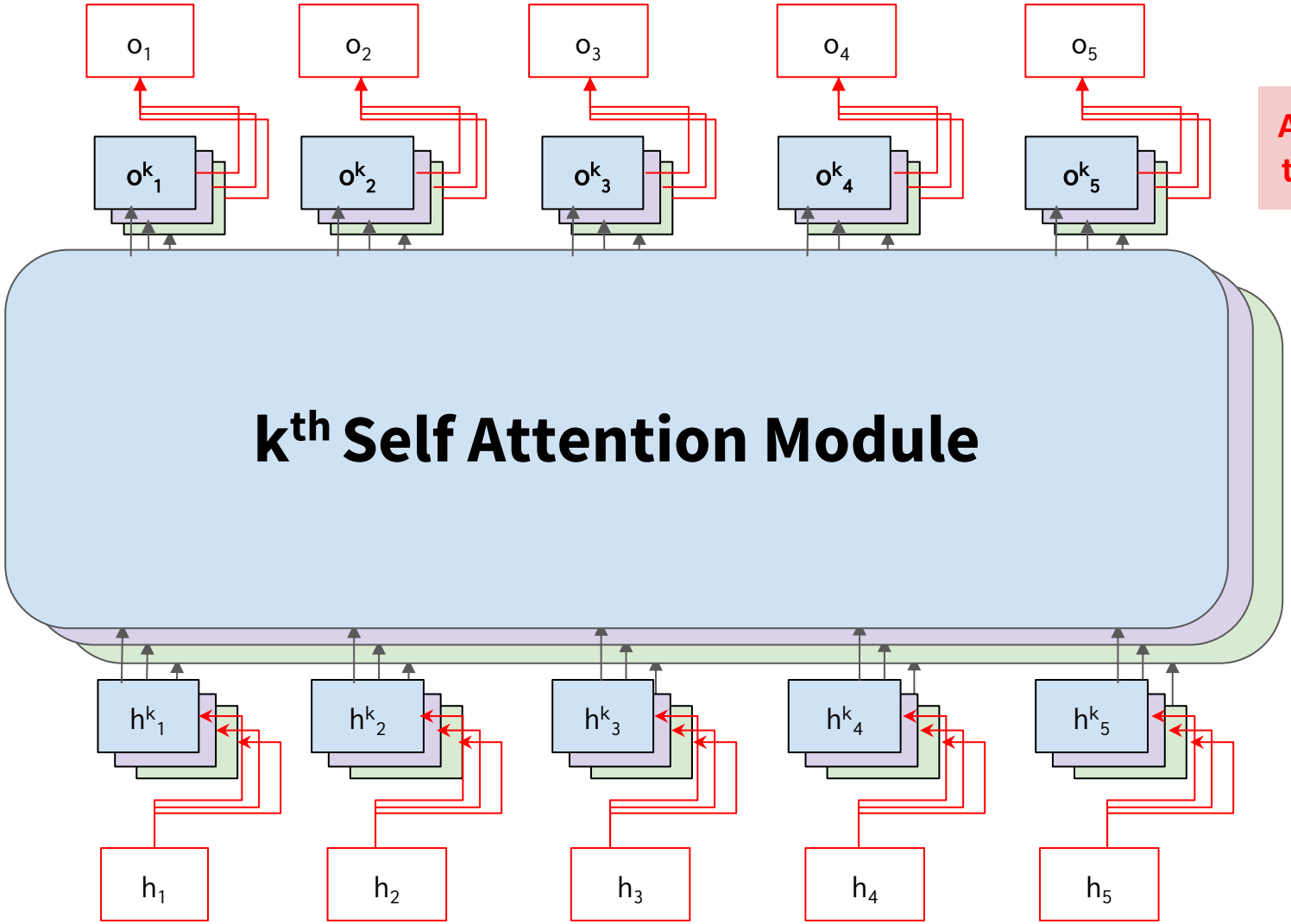
d. The energy function is scaled to allow for numerical stability

What if we split the input into ‘k’ sub-inputs?



Then pass each sub-input into a Self-Attention Module?





And combine the outputs?


o_1
o_2
o_3
o_4
o_5
o^k_1
o^k_2
o^k_3
o^k_4
o^k_5
k^{th} Self Attention Module
Multi-head Attention
h^k_1
h^k_2
h^k_3
h^k_4
h^k_5
h_1
h_2
h_3
h_4
h_5

Multi Head Self Attention

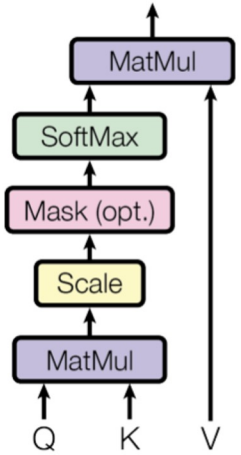
- Split input into **k** parts
- Pass the **j^{th}** part of **each input** into the **j^{th} attention head**
- Concatenate each of the **k** outputs

Why go through the trouble?

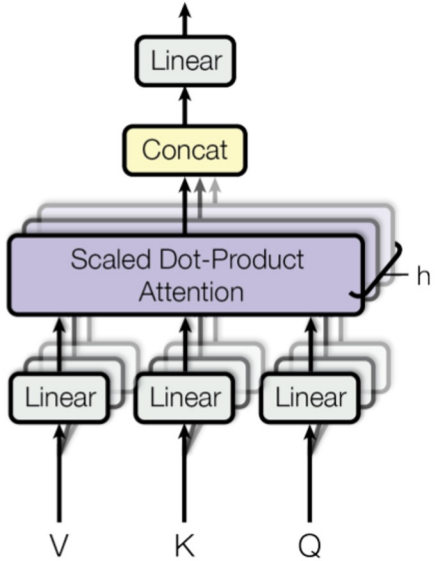
- Each head **could** find a different kind of relation between the tokens
    - Subject-verb, subject-object, verb-modifier, dependency, etc.

Attention is all you need

Scaled Dot-Product Attention



Multi-Head Attention



Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

Breaking down the transformer

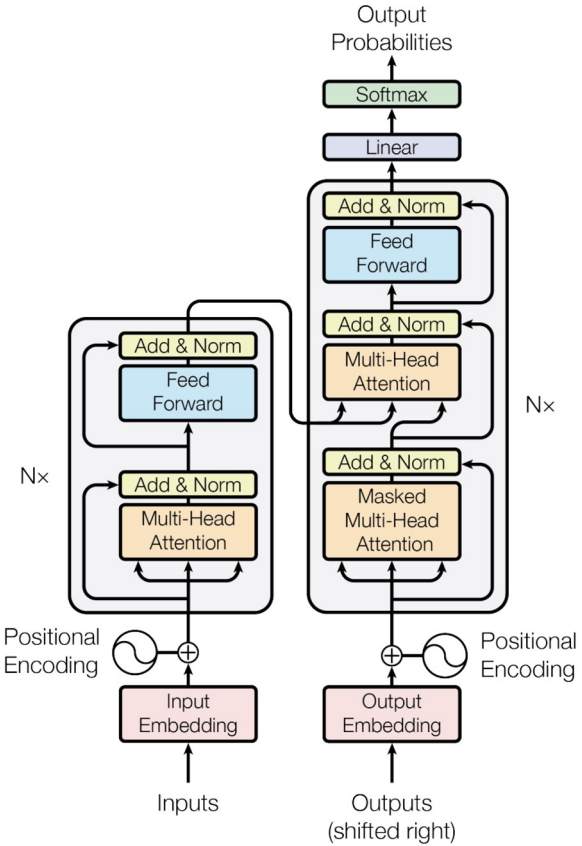


Figure 1: The Transformer - model architecture.

Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

Breaking down the transformer

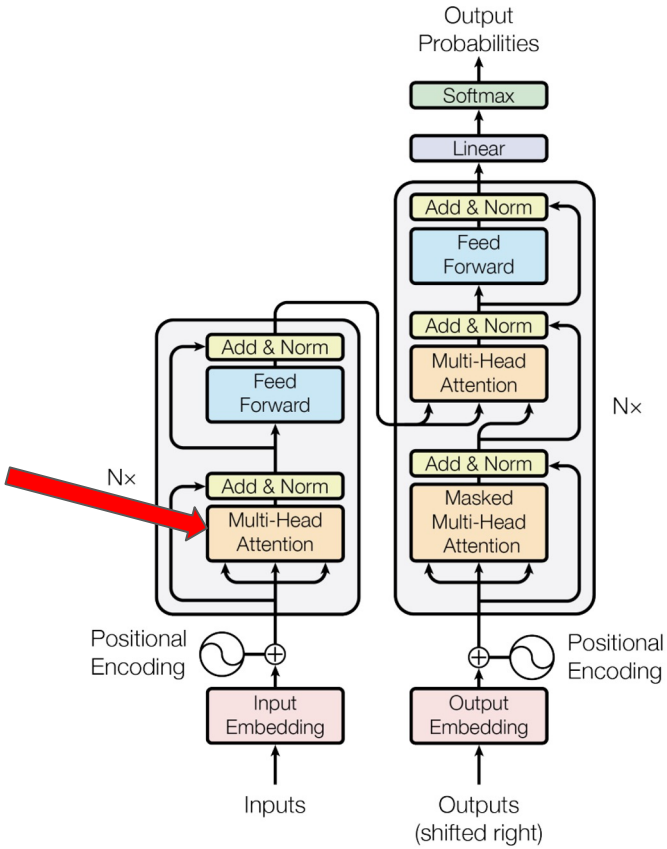


Figure 1: The Transformer - model architecture.

Encoder Self Attention

Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

Breaking down the transformer

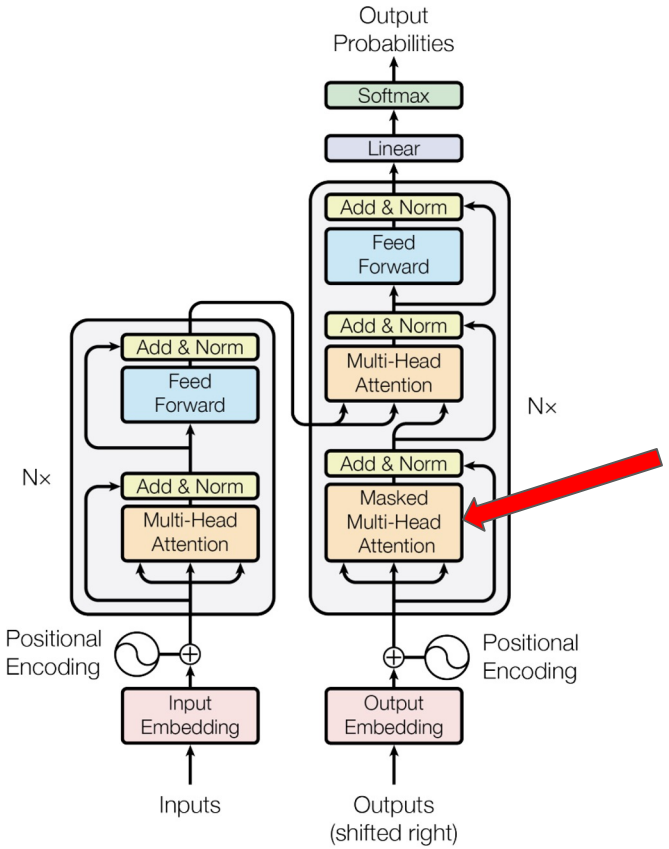


Figure 1: The Transformer - model architecture.

Decoder Self Attention

Masked Attention

Decoders can be parallelized during **training** (only)

Feed the whole output sequence (outputs) in at once

Need to ensure model doesn't cheat

Alter the attention weights to be **0** (set input to softmax to **-inf**) for all times `t' > t`

Ensure **autoregressive property**

Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

Breaking down the transformer

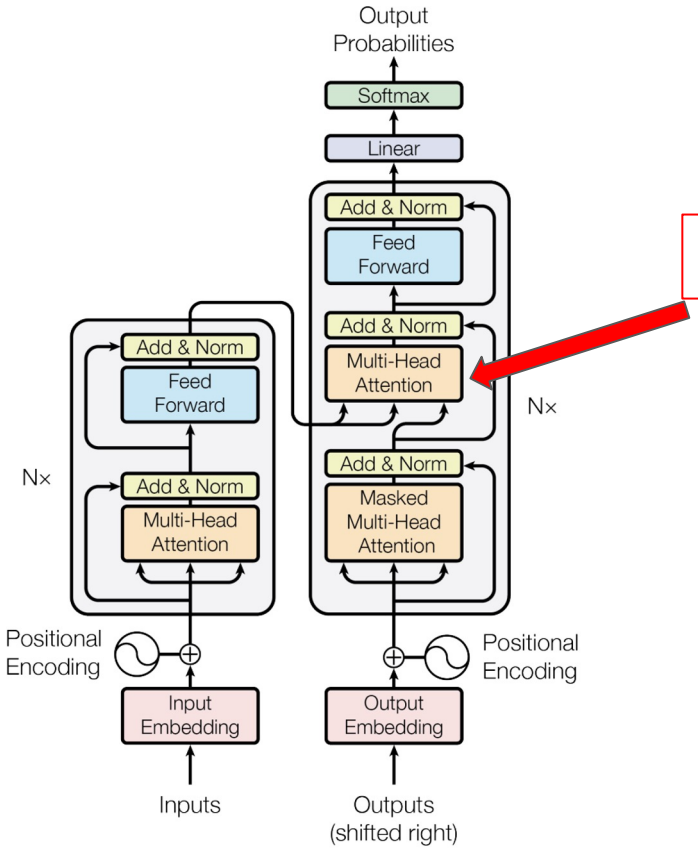


Figure 1: The Transformer - model architecture.

Cross Attention

During decoding, the **query** comes from the outputs, **keys** and **values** come from the encoder.

Decoder input **"pays" attention** to the encoder outputs.

Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

Breaking down the transformer

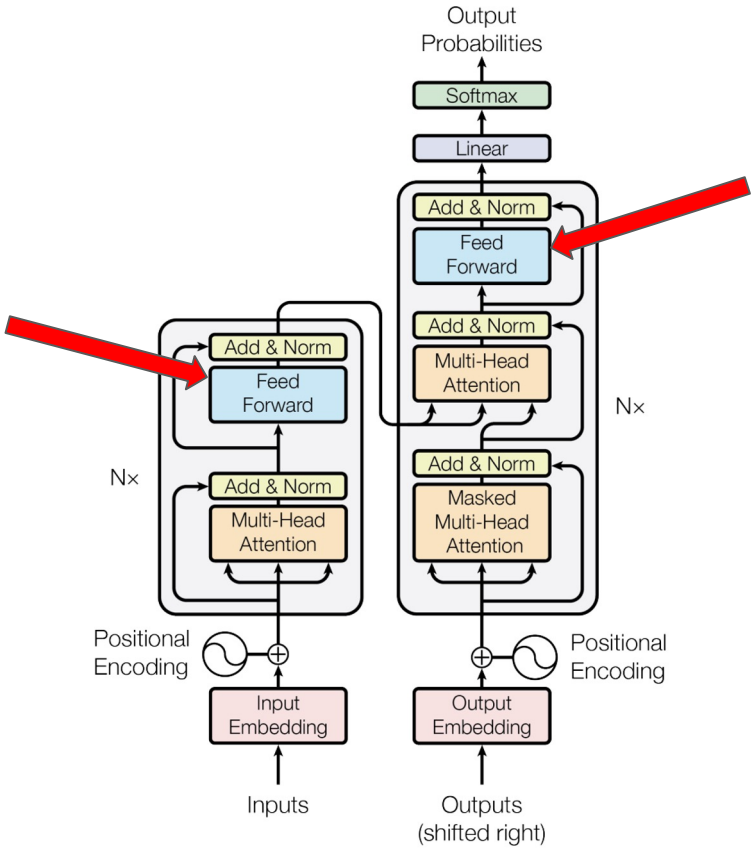
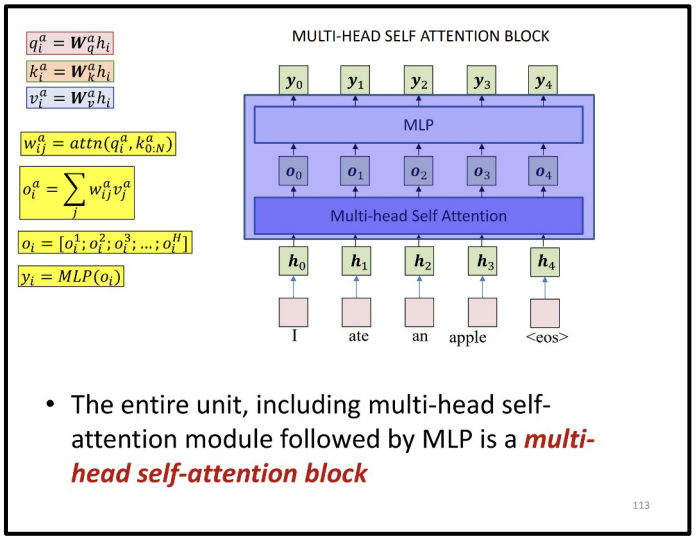


Figure 1: The Transformer - model architecture.

Feed Forward Layers

Feed Forward layers allow for high dimensional computations

Simply there to allow the model to capture more information



- The entire unit, including multi-head self-attention module followed by MLP is a ***multi-head self-attention block***

Breaking down the transformer

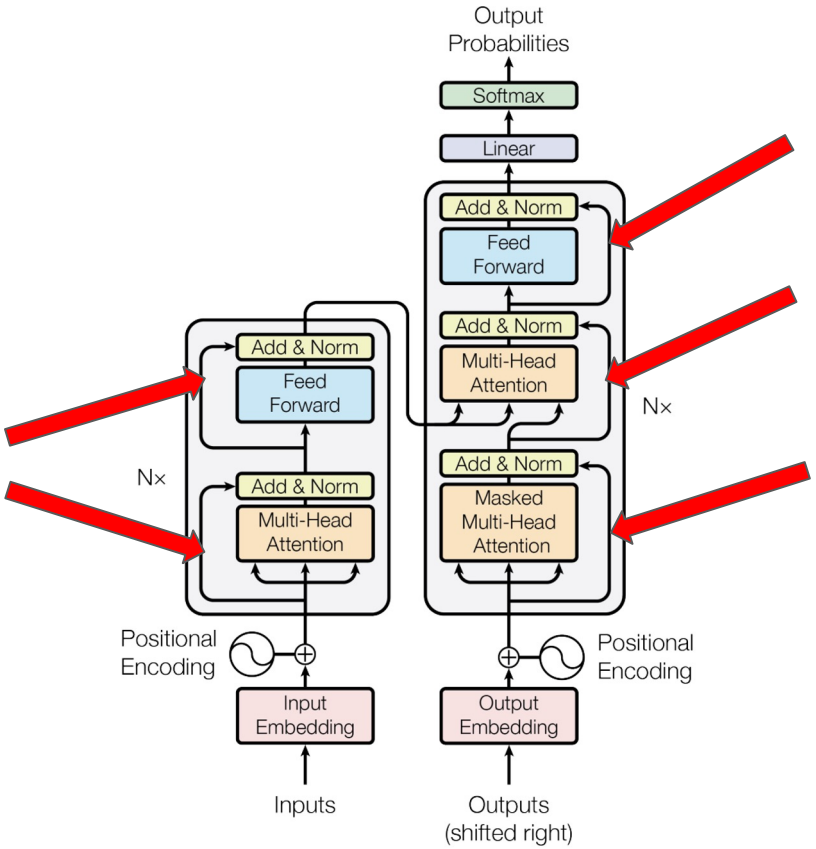


Figure 1: The Transformer - model architecture.

Residual Connections

`Add & Norm`:

```
out = LayerNorm(x + Sublayer(x)),
```

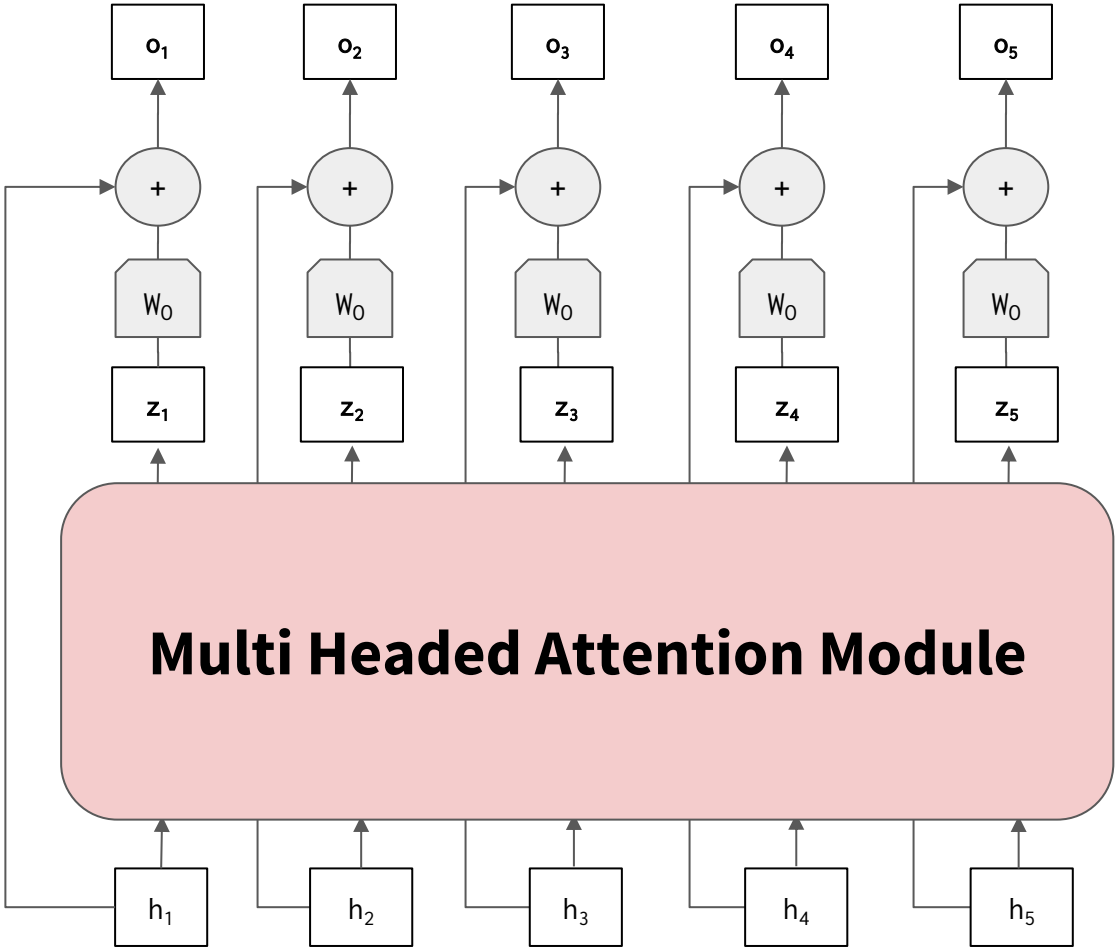
`Sublayer(x)` is whatever layer is below the **Add & Norm**

Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

Residual Connection

Transformers are residual machines

"How much do I **SHIFT** my meaning given my context?"



Breaking down the transformer

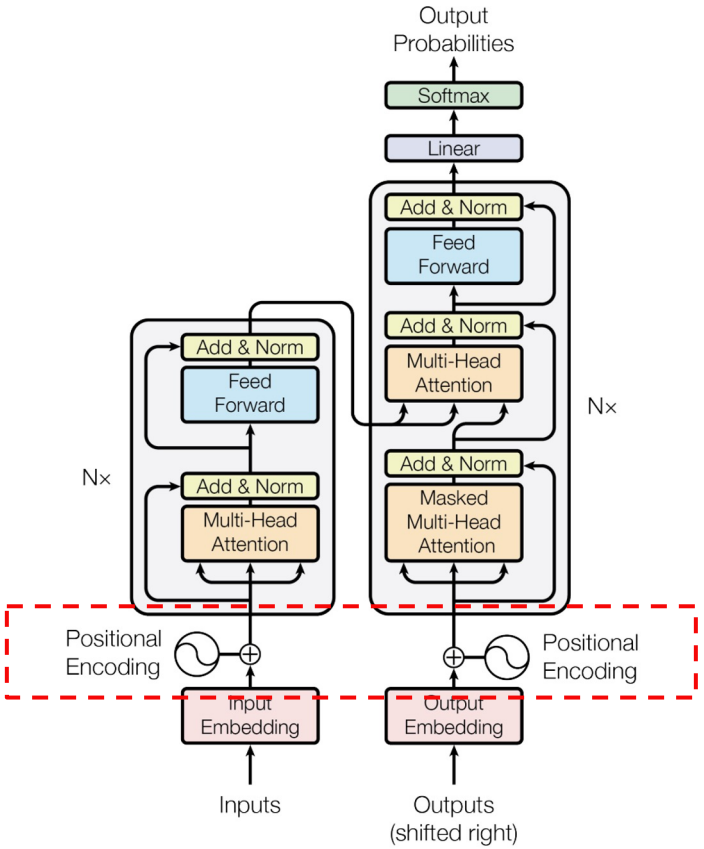


Figure 1: The Transformer - model architecture.

Transformers have no inherent notion of order in a sequence.

This notion has to be externally enforced.

Positional Encodings are added to transformer inputs to add information about order.

Positional Encoding

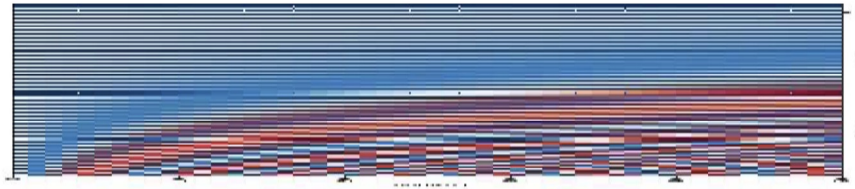
Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

Recall

Positional encodings as discussed in the last lecture.

Positional Encoding

regenerate



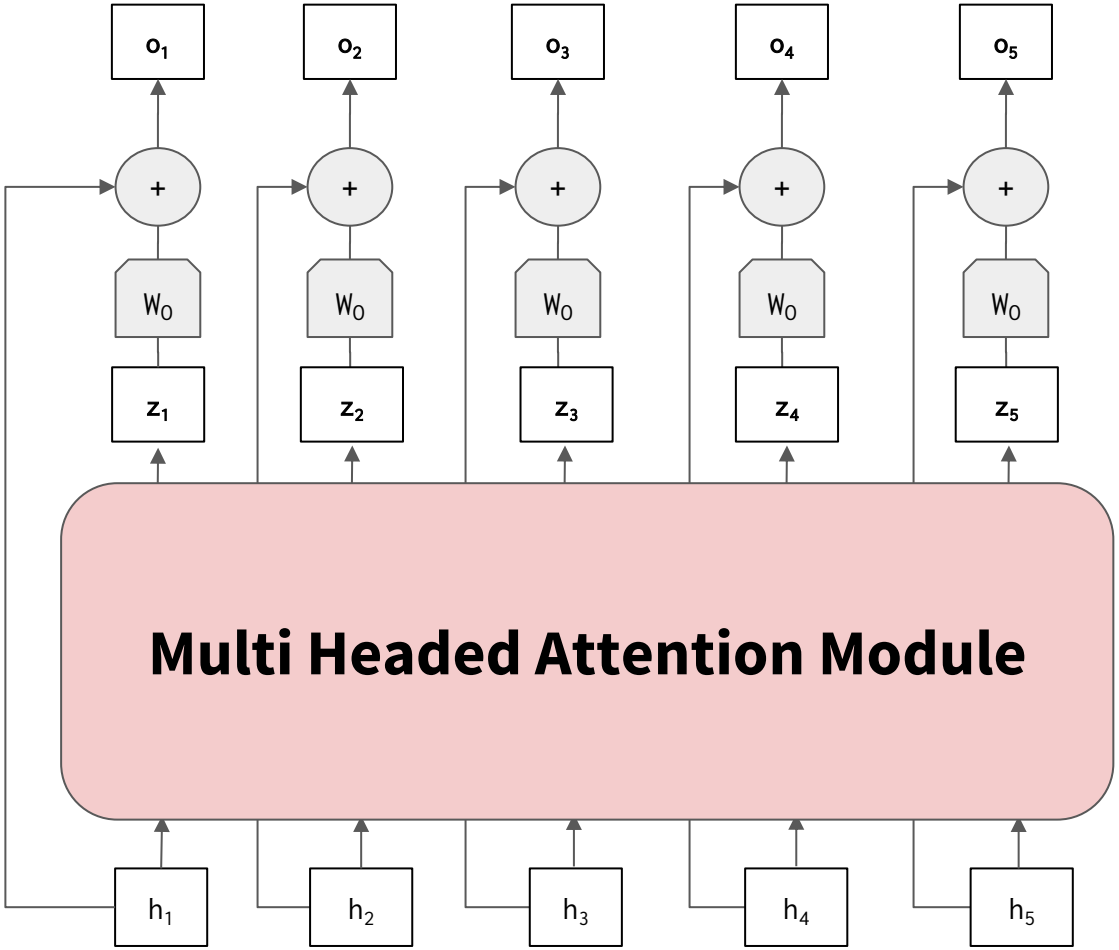
$$P_t = \begin{bmatrix} \sin \omega_1 t \\ \cos \omega_1 t \\ \sin \omega_2 t \\ \cos \omega_2 t \\ \vdots \\ \sin \omega_{d/2} t \\ \sin \omega_{d/2} t \end{bmatrix} \qquad \omega_l = \frac{1}{10000^{2l/d}}$$

$$P_{t+\tau} = M_\tau P_t$$

$$M_\tau = diag\left(\begin{bmatrix} \cos \omega_l \tau & \sin \omega_l \tau \\ -\sin \omega_l \tau & \cos \omega_l \tau \end{bmatrix}, l = 1 \ldots d/2\right)$$

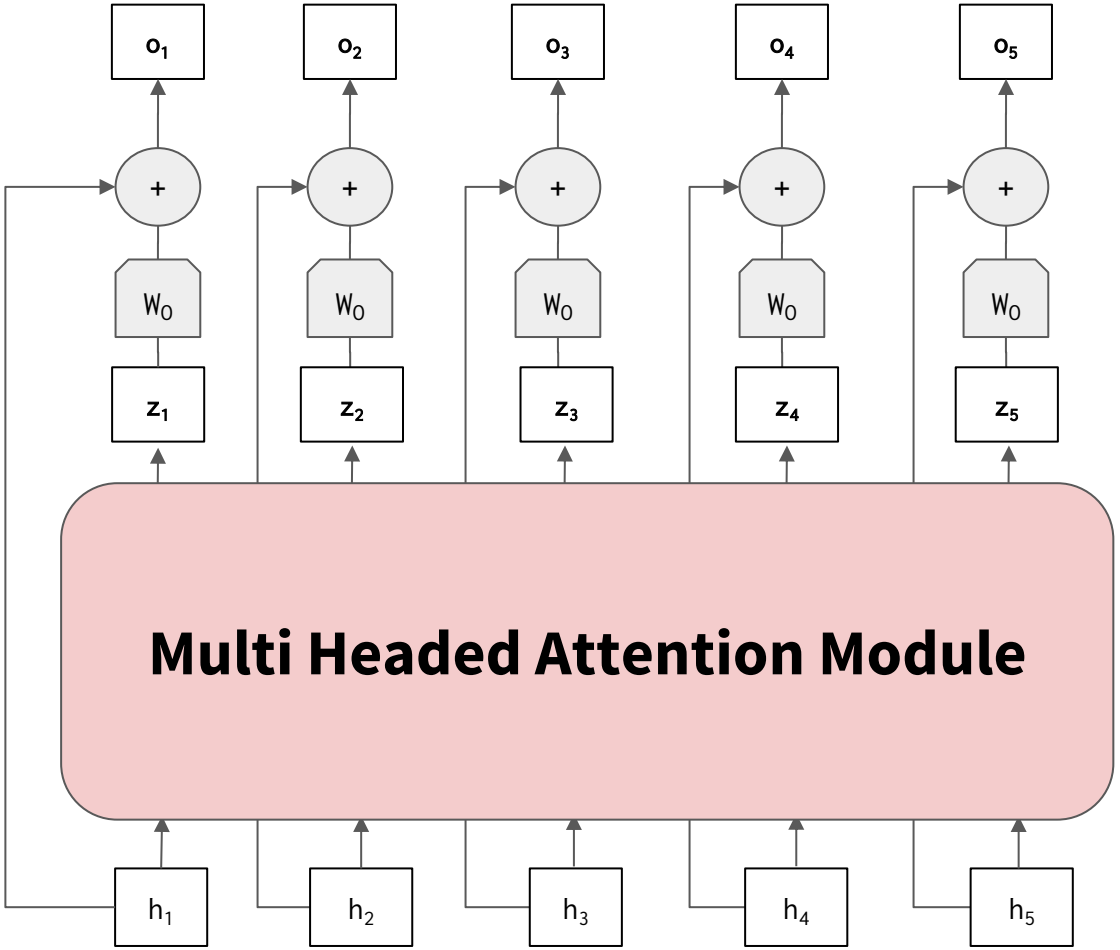
- A vector of sines and cosines of a harmonic series of frequencies
  - Every $2l$-th component of P_t is $\sin \omega_l t$
  - Every $2l + 1$-th component of P_t is $\cos \omega_l t$
- Never repeats
- Has the linearity property required

Positional Encoding



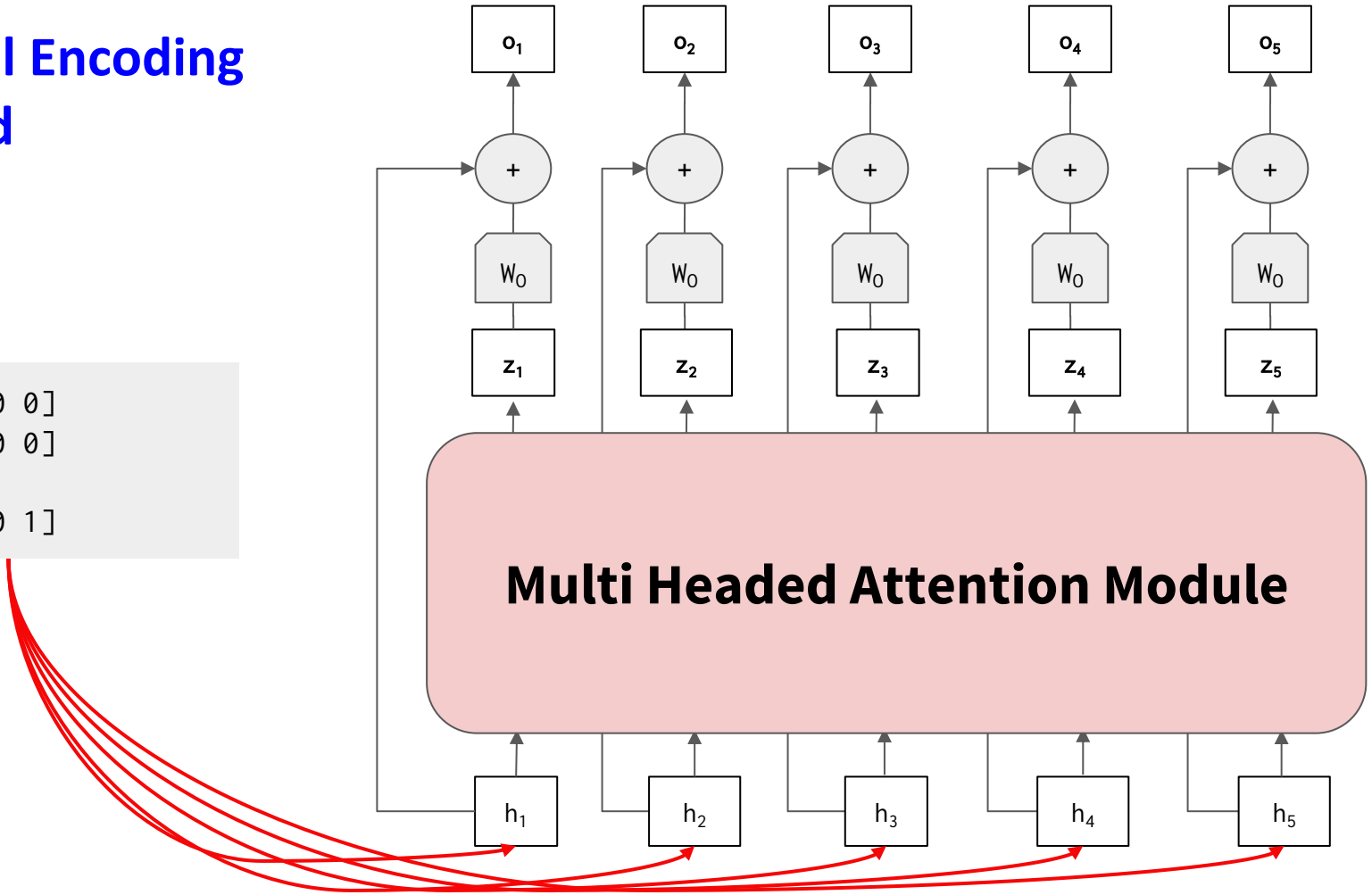
Positional Encoding simplified

p_1 = [1 0 0 0 0]
p_2 = [0 1 0 0 0]
…
p_5 = [0 0 0 0 1]



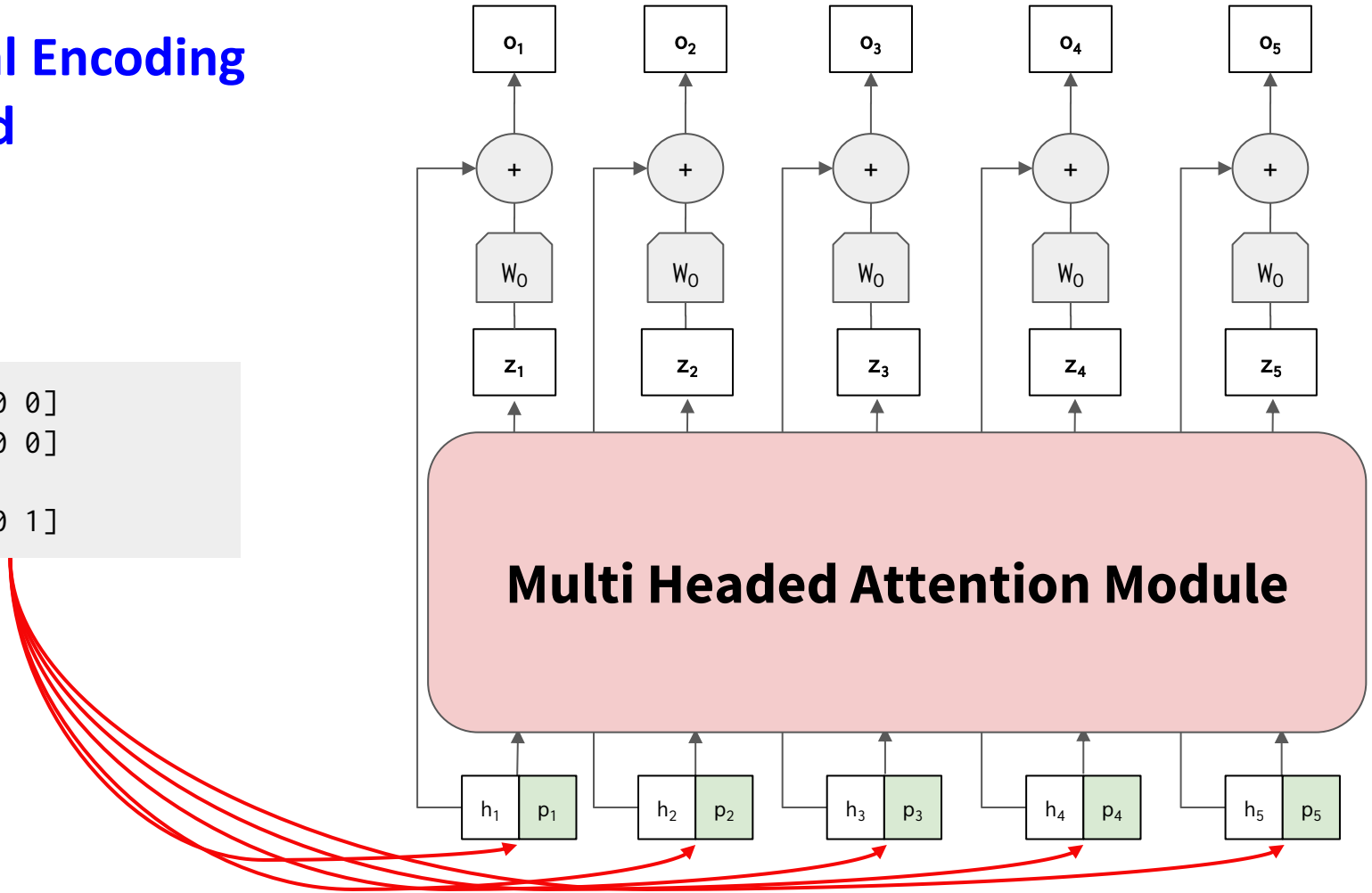
Positional Encoding simplified

p_1 = [1 0 0 0 0]
p_2 = [0 1 0 0 0]
…
p_5 = [0 0 0 0 1]



Positional Encoding simplified

p_1 = [1 0 0 0 0]
p_2 = [0 1 0 0 0]
…
p_5 = [0 0 0 0 1]



Poll 2 (@1126)

Which of the following are true about transformers?

a. The attention module tries to calculate the “shift” in meaning of a token given all other tokens in the batch

b. Transformers can always be run in parallel

c. Transformer decoders can only be parallelized during training

d. Positional encodings help parallelize the transformer encoder

e. Queries, keys, and values are obtained by splitting the input into 3 equal segments

f. Multiheaded attention helps transformers find different kinds of relations between the tokens

g. During decoding, decoder outputs function as queries and keys while the values come from the encoder

Poll 2 (@1126)

Which of the following are true about transformers?

a. **The attention module tries to calculate the "shift" in meaning of a token given all other tokens in the batch**

b. Transformers can always be run in parallel

c. **Transformer decoders can only be parallelized during training**

d. Positional encodings help parallelize the transformer encoder

e. Queries, keys, and values are obtained by splitting the input into 3 equal segments

f. **Multiheaded attention helps transformers find different kinds of relations between the tokens**

g. During decoding, decoder outputs function as queries and keys while the values come from the encoder

Summary (1)

- Roles of Queries, Keys, and Values

  Q pay attention to **V** according to computation with **K**

  “Computation” is the attention function.
- **Self** versus **Cross** attention
- Transformers are Residual Machines
- **Positional Encodings:** Transformers have no notion of order - this needs to be explicitly inserted.

Summary (2)

- Transformers' biggest advantage lies in parallelizability and 'omni-directionality'
- There are still cases where models from the RNN family might perform better than Transformers.

Extra Slides

Few types of energy functions

- MLP

  $e(q,k) = W_2^T(\tanh(W_1^T[q;k]))$

- Bilinear

  $e(q,k) = (q^T)(W)(k)$

- Scaled-Dot Product

  $e(q,k) = (q)(k^T) / (s)$ # s = scaling factor ($\sqrt{d_k}$)

Batching and shapes

The attention function takes in:

$q : (B, T, d_q)$

$k : (B, T, d_k)$

$v : (B, T, d_v)$

Energy / attention scores:

$e : (B, T, T)$ # Score between each pair of tokens **if $e = qk^T/s$**

Output vector:

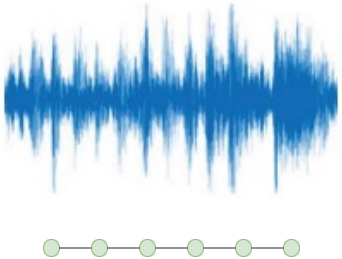
$o : (B, T, d_v)$ # calculated as **$\text{softmax}(e)^T v$**

Part 2

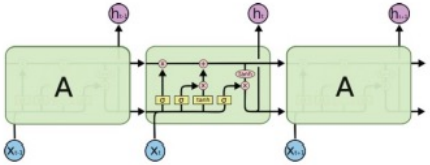
Graph Neural Networks

Revisiting some kinds of data

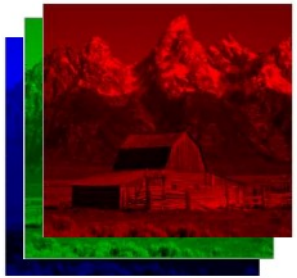
Sequence data: text/speech

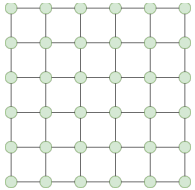


Recurrent Neural Networks

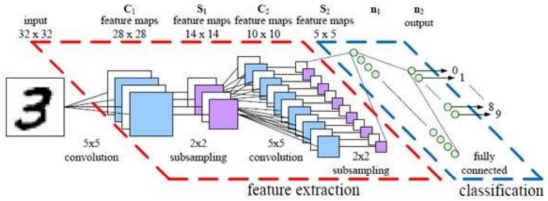


Grid data: image



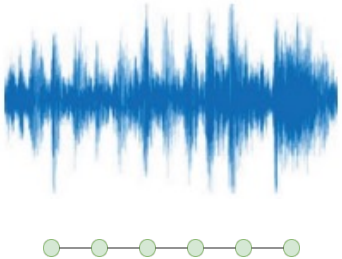


Convolution Neural Networks

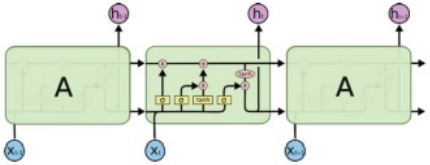


Revisiting some kinds of data

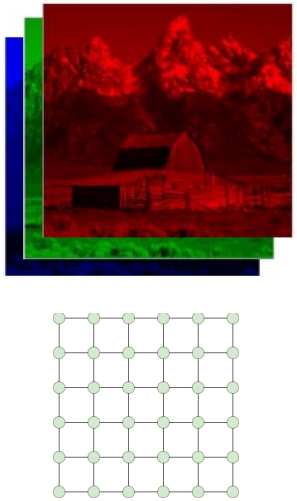
Sequence data: text/speech



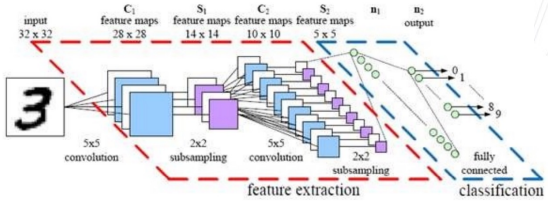
Recurrent Neural Networks



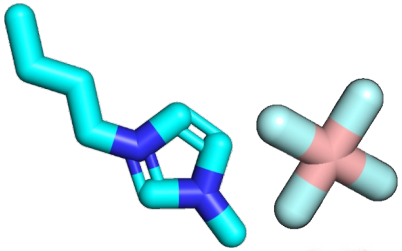
Grid data: image

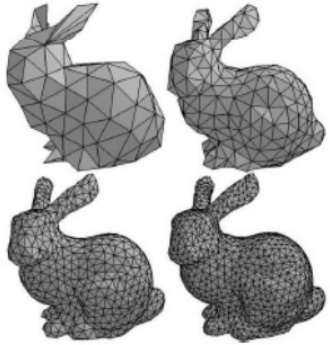


Convolution Neural Networks



Unstructured Data:
Molecules, Social Networks, 3D meshes





???

Graphs: Definition

A graph is defined as a tuple G = (V, E),

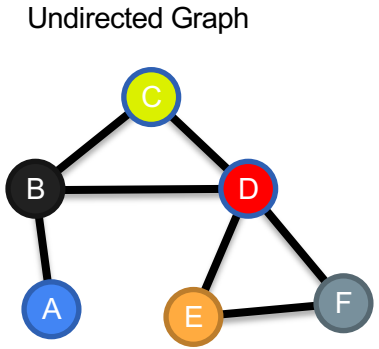
- where V is a set of nodes / vertices,
- and E is a set of edges connecting a pair or vertices.

Example:

G = (V,E)

V = {A,B,C,D,E,F}

E = {(A,B),(B,C),(C,D),(B,D),(C,D),(D,E),(D,F),(E,F)}



Invariance

- Say we have a mapping (our function / model) $f: X \to Y$
- And another mapping (a transformation) $g: X \to X$
- If (and only if) $f(x) = f\big(g(x)\big)\ \forall\ x \in X$, we can claim that **f is invariant to g.**
- **Poll 3**

Poll number	f	g	X	f invariant to g?
@1127	argmax	softmax	$\mathbb{R}^n$	
@1128	Euclidean distance between two points	Translation (of the origin)	$(\mathbb{R}^n, \mathbb{R}^n)$	
@1129	Angle between two vectors	Translation (of the origin)	$(\mathbb{R}^n, \mathbb{R}^n)$	

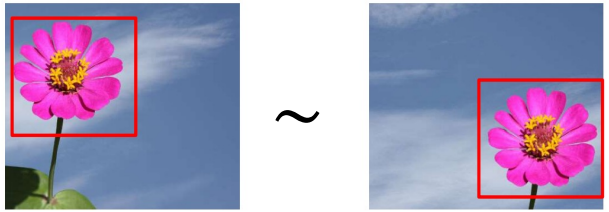
Invariance

- Say we have a mapping (our function / model) $f: X \to Y$
- And another mapping (a transformation) $g: X \to X$
- If (and only if) $f(x) = f\big(g(x)\big)\ \forall\ x \in X$, we can claim that ***f* is invariant to *g*.**
- **Poll 3**

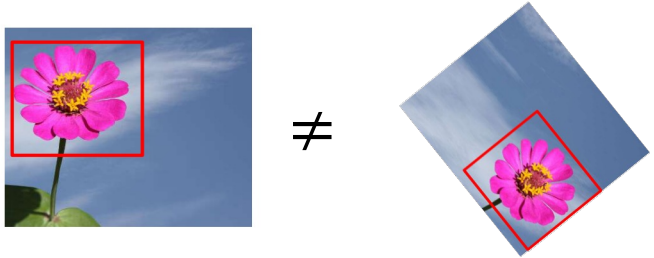
Poll number	f	g	X	f invariant to g?
@1127	argmax	softmax	$\mathbb{R}^n$	**YES**
@1128	Euclidean distance between two points	Translation (of the origin)	$(\mathbb{R}^n, \mathbb{R}^n)$	**YES**
@1129	Angle between two vectors	Translation (of the origin)	$(\mathbb{R}^n, \mathbb{R}^n)$	**NO**

Revisiting invariances we have discussed so far

- CNNs are (kind of) translation invariant.



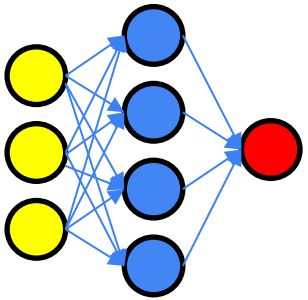
- CNNs are NOT rotation invariant (by default).



- Transformers are order invariant (without positional encodings).

Permutation Invariance

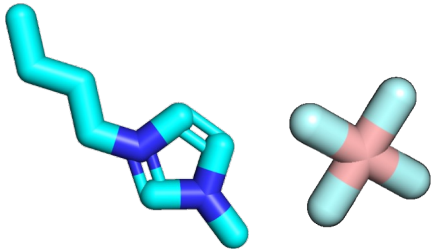
- Original input: [-0.5, 0.3, 0.8] $x^{(1)}$
- Possible permutations: [0.3, -0.5, 0.8] $x^{(2)}$, [0.8, 0.3, -0.5] $x^{(3)}$, …
- f: $\mathbb{R}^3 \rightarrow \mathbb{R}$ (an MLP)



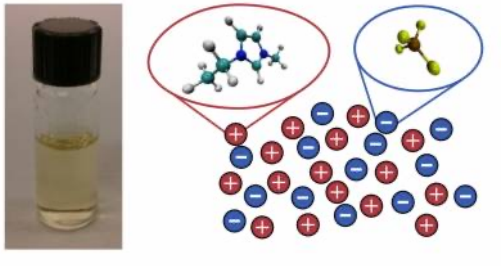
- Is $f(x^{(1)}) = f(x^{(2)}) = f(x^{(3)})$? **NO!** → f is not permutation invariant!
- Permutation invariance requires the output of **all 6! permutations** of the input to result in the **same answer**.

Problem Setup: ionic liquid for CO2 capturing

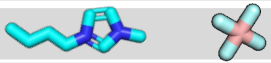
Ionic liquid molecules



Carbon dioxide (CO2)



Data

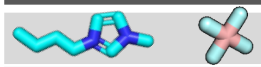
Ionic Liquid	Solubility (label)
	0.56
…	0.119

We want to use deep learning model to predict the solubility of ionic liquid based on these molecule data!

$$f: Molecule \rightarrow \mathbb{R}$$

Problem Setup: ionic liquid for CO2 capturing

Dataset

Ionic Liquid	Solubility (label)
	0.56
…	0.119

We want to use deep learning model to predict the solubility of ionic liquid based on these molecule data!

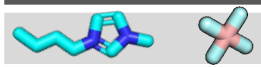
$$f: Molecule \rightarrow \mathbb{R}$$

Two questions:

1. Can we use MLP or CNN to solve this problem?
2. What are the desired properties of the model we would like to use?

Problem Setup: ionic liquid for CO2 capturing

Dataset

Ionic Liquid	Solubility (label)
	0.56
…	0.119

We want to use deep learning model to predict the solubility of ionic liquid based on these molecule data!

$$f: Molecule \rightarrow \mathbb{R}$$

Two questions:

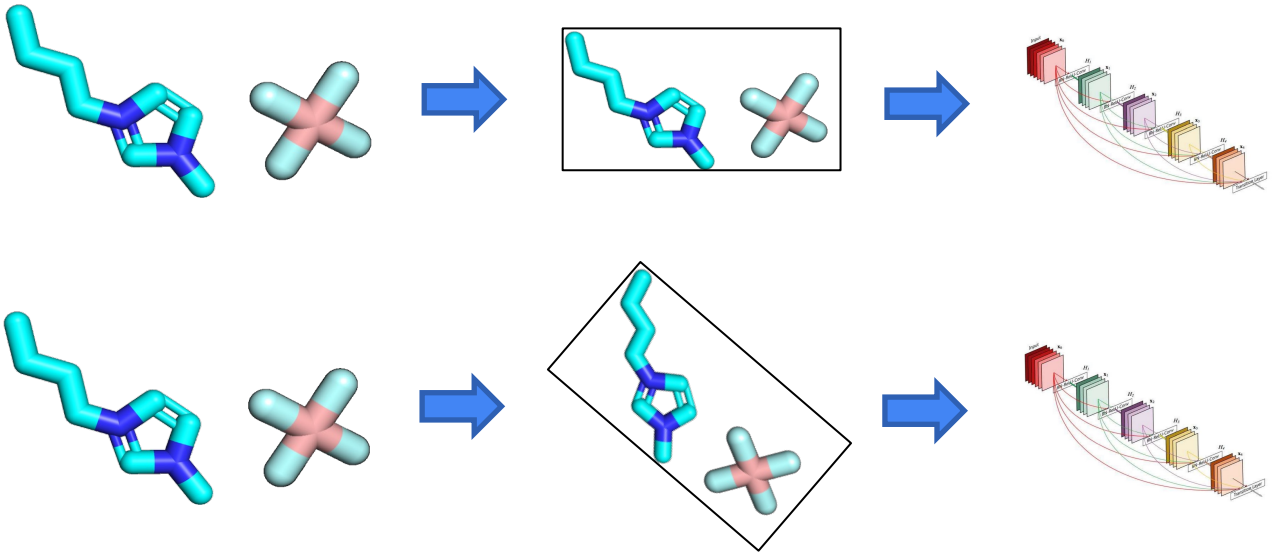
1. **Can we use MLP or CNN to solve this problem?**
2. What are the desired properties of the model we would like to use?

Short answer: Not really.

Possible Solution 1: Using CNNs for feature extraction

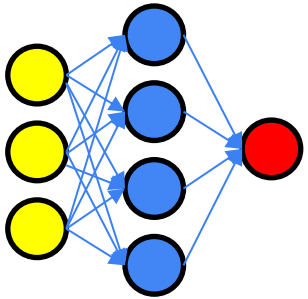
Ionic liquid molecules

Image data: image for molecule structure



- **The outputs don't match!**
- CNNs are not rotation invariant (we know this)

Possible Solution 2: Using MLPs for feature extraction

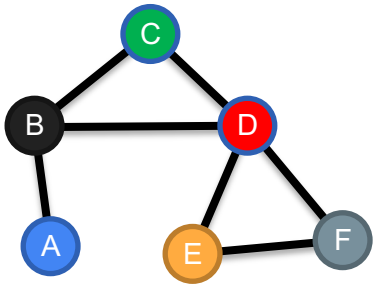


But… What do we pass as input?

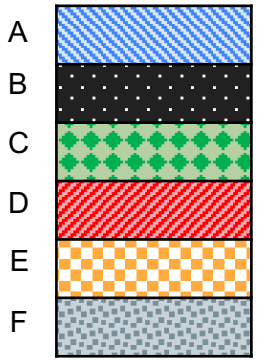
Feature engineering for graph-data

Matrix representation of a graph, G = (V, E)

Undirected Graph

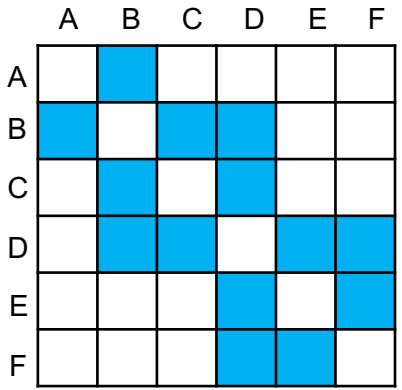


1. Node information



Node information Matrix (N×F)

2. Connectivity information



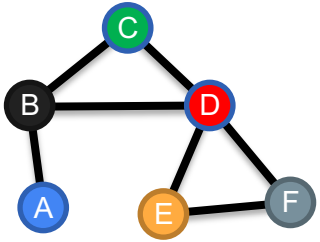
Adjacency Matrix (N×N)

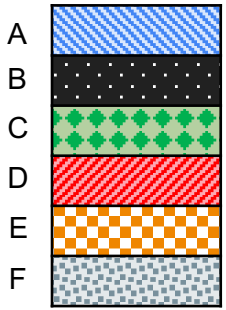
We can now define a model $f: (\mathbb{R}^F, \mathbb{R}^N) \rightarrow \mathbb{R}^d$, where **V** is captured by the node information matrix ($\mathbb{R}^F$), **E** is captured by the adjacency matrix ($\mathbb{R}^N$), and d is the desired output dimension.

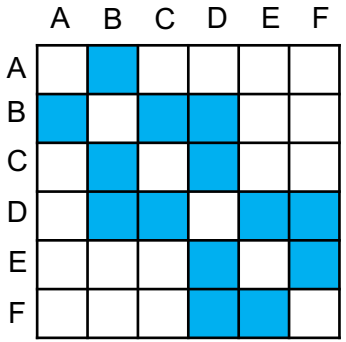
Feature engineering for graph-data

Graph do not have canonical order of the nodes!

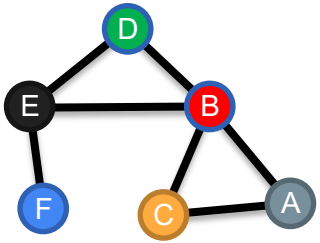
Order plan 1

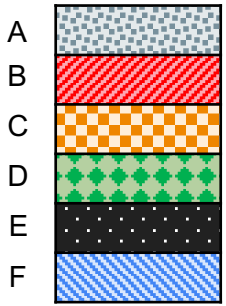


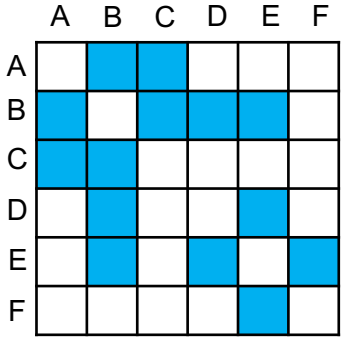




Order plan 2

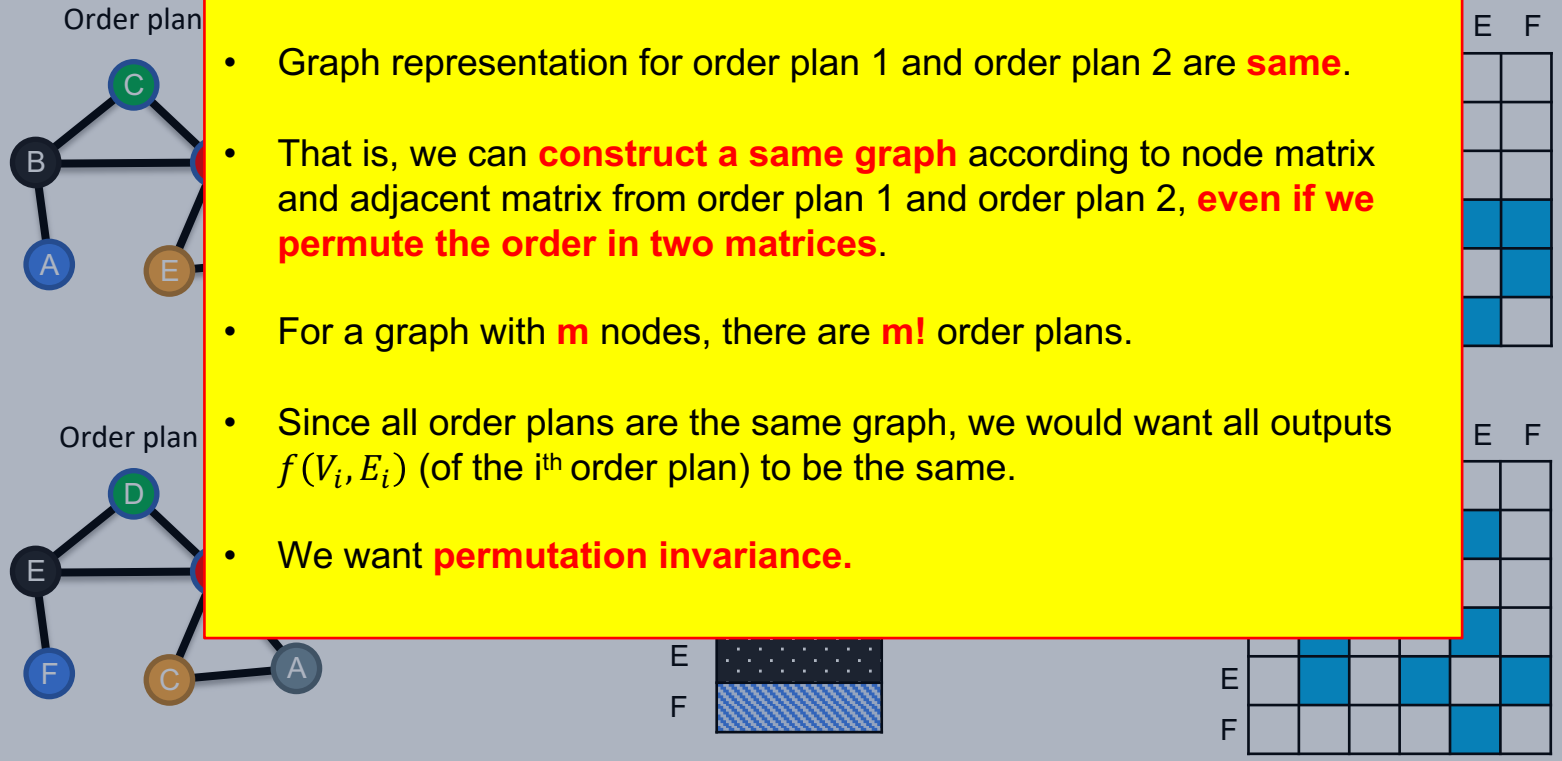






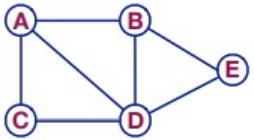
Feature engineering for graph-data

Graphs don't have a canonical order of the nodes!

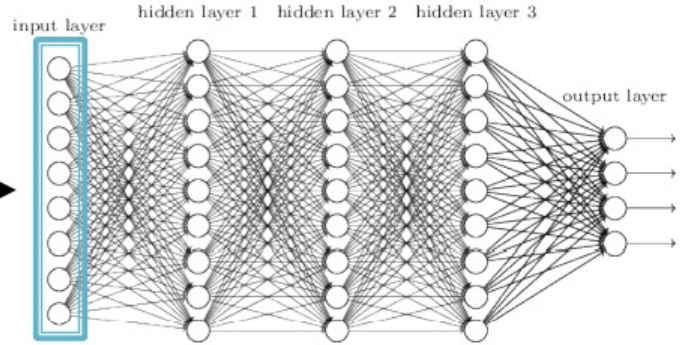


- Graph representation for order plan 1 and order plan 2 are **same**.
- That is, we can **construct a same graph** according to node matrix and adjacent matrix from order plan 1 and order plan 2, **even if we permute the order in two matrices**.
- For a graph with **m** nodes, there are **m!** order plans.
- Since all order plans are the same graph, we would want all outputs $f(V_i, E_i)$ (of the i[th] order plan) to be the same.
- We want **permutation invariance.**

Possible Solution 2: Using MLPs for feature extraction



	A	B	C	D	E	Feat	
A	0	1	1	1	0	1	0
B	1	0	0	1	1	0	0
C	1	0	0	1	0	0	1
D	1	1	1	0	1	1	1
E	0	1	0	1	0	1	0



What happens if we use a different order plan?

Changing the order plan will change the sequence order and thus produce a different result!

So, an MLP with graph-features also fails here.
We need a different way to process these inputs to work with the graph-properties that exist in the data.

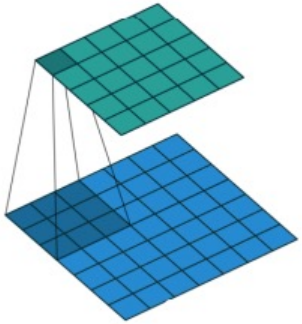
Story so far

- Graph can be represented by using a feature matrix and an adjacency matrix.
- Graph representations don’t have a canonical order of nodes.
- Permutation invariance is a desired property of the model we use for graph processing.

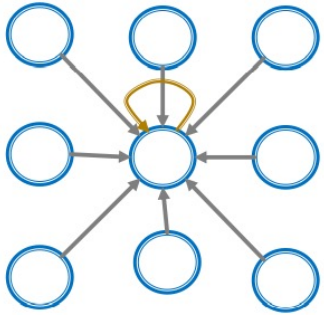
A single layer of GNN: Graph Convolution

Key idea: Node's neighborhood defines its features

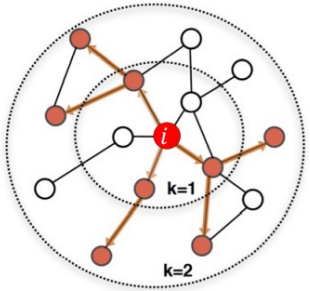
"Birds of a feather" assumption

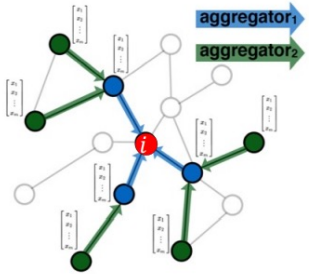


CNN: Pixel convolution



CNN: Pixel convolution (as a graph)



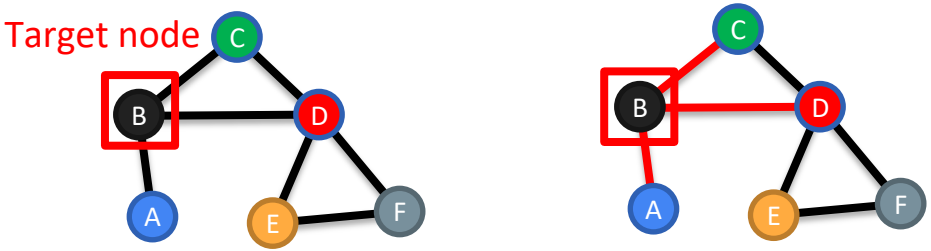


GNN: Graph convolution

- Node embedding can be defined by local network neighborhoods.
- Learn a node feature by propagating and aggregating neighbor information.

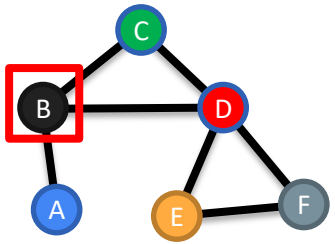
A single layer of GNN: Graph Convolution

Generate node embedding based on local network neighborhoods

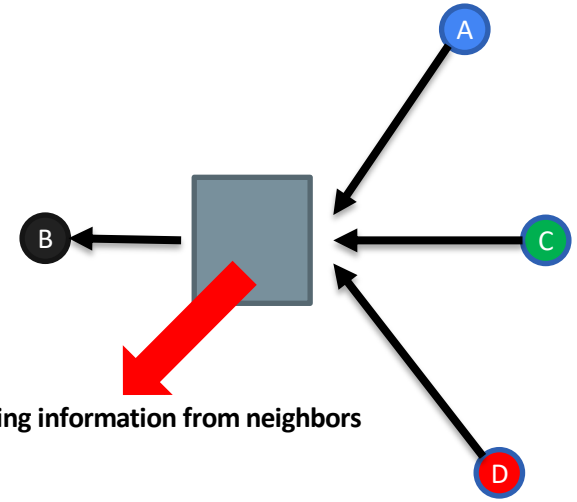


A single layer of GNN: Graph Convolution

Generate node embedding based on local network neighborhoods



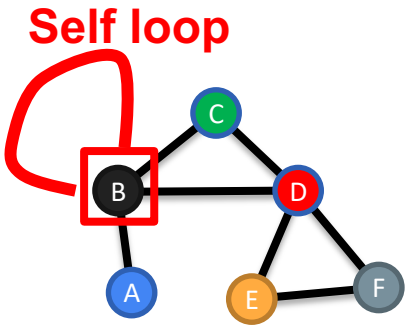
Generating features from nodes 1 **hop** away
A,C, and **D**



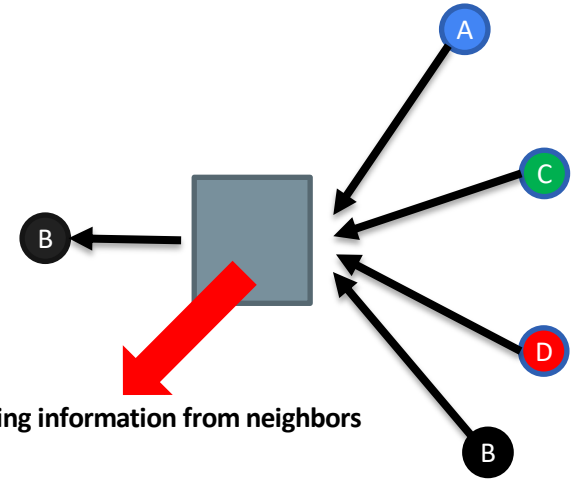
Now embedding for node **B** has information from A,C, and D.

A single layer of GNN: Graph Convolution

Generate node embedding based on local network neighborhoods



Generating features from nodes **1 hop** away
A,C, and **D**

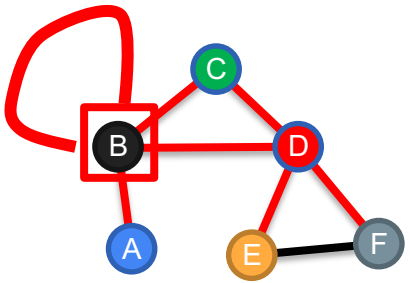


But we don't want to forget information about B either.

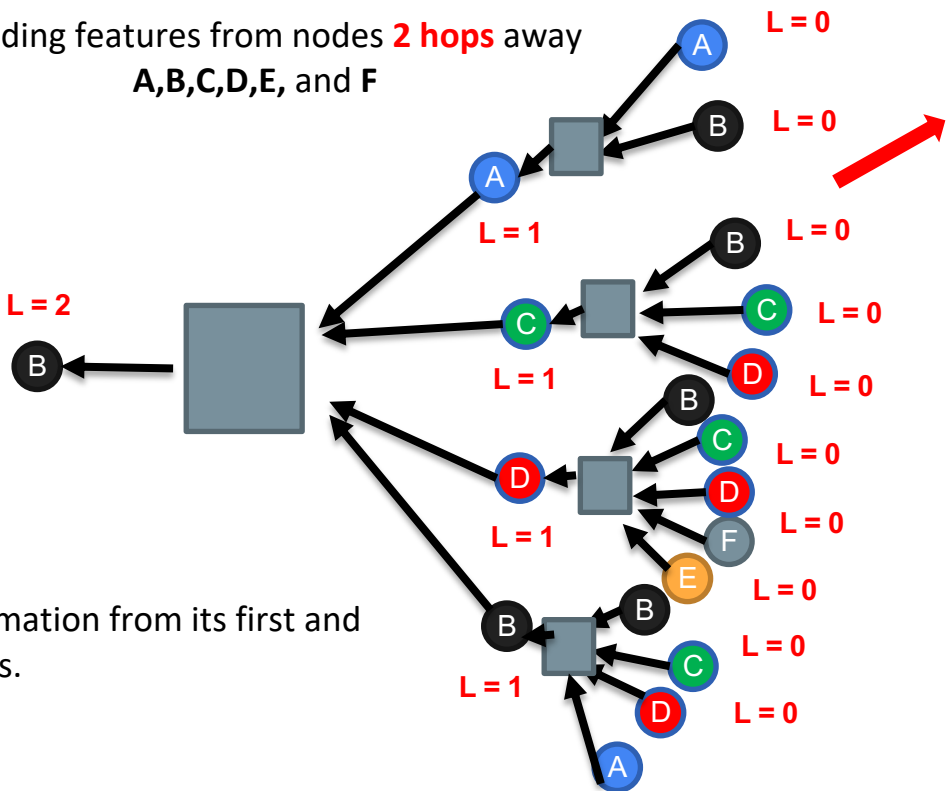
Now embedding for node **B** has information from A,C,D, **and B itself**.

Two layers of GNN: Graph Convolution

Generate node embedding based on local network neighborhoods



Including features from nodes **2 hops** away
A,B,C,D,E, and **F**

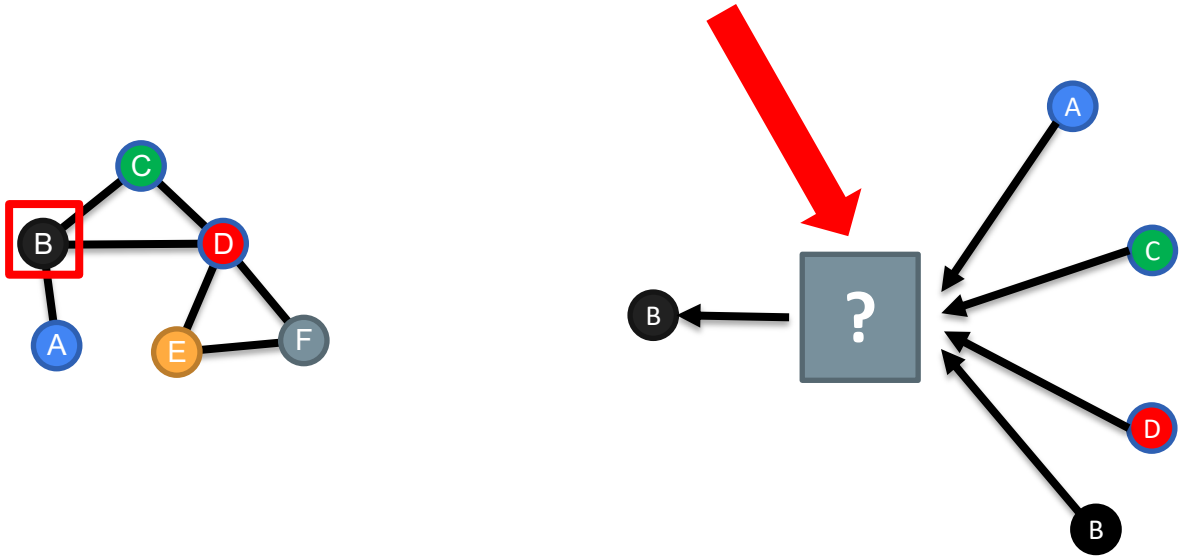


Embedding for node **B** (at layer 2) has information from its first and second hop neighbors.

A single layer of GNN: Graph Convolution

Generate node embedding based on local network neighborhoods

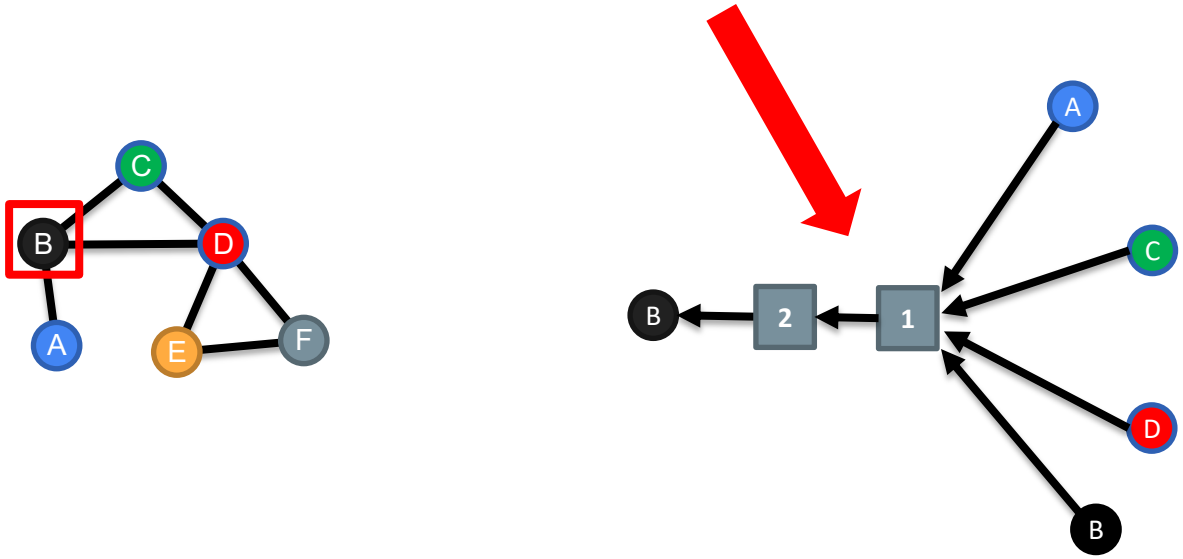
What happens during the processing (the grey box)?



A single layer of GNN: Graph Convolution

Generate node embedding based on local network neighborhoods

What happens during the processing (the grey box)?



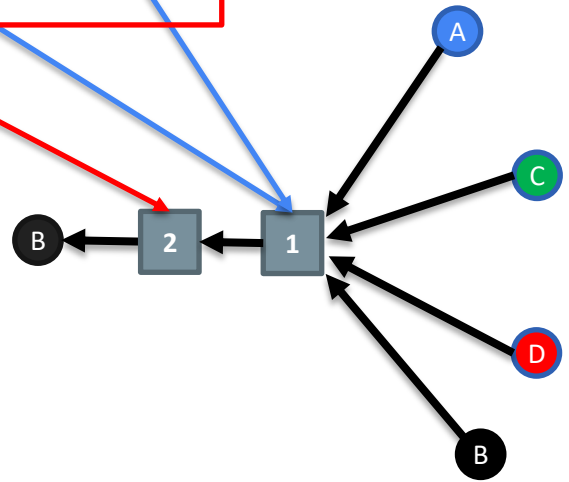
Two step process:

1. Aggregate information (sum, mean, etc.)
2. Apply activated linear transformation, Neural Networks. $\sigma(Wx + B)$

A single layer of GNN: Graph Convolution – Forward pass

The Math for the l^{th} layer

$$h_v^{(l+1)} = \sigma\left(W^{(l)}\sum_{u\in N(v)}\frac{h_u^{(l)}}{|N(v)|} + B^{(l)}h_v^{(l)}\right), \forall l \in [0, 1 \ldots L-1]$$



A single layer of GNN: Graph Convolution – Forward pass

The Math for the l^{th} layer

$$h_v^{(l+1)} = \sigma\left(W^{(l)} \sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|} + B^{(l)} h_v^{(l)}\right), \forall l \in [0, 1 \ldots L-1]$$

Non-linear activation

Learnable parameter

Accumulation of features from neighbors at previous layer

Learnable parameter

Self loop

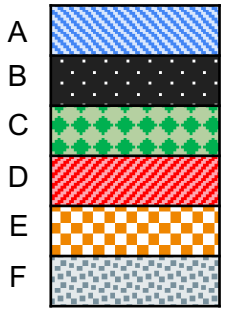
A single layer of GNN: Graph Convolution – Forward pass

The Math for the l^{th} layer

$$h_v^{(l+1)} = \sigma\left(W^{(l)} \sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|} + B^{(l)} h_v^{(l)}\right), \forall l \in [0, 1 \ldots L-1]$$

$$h_v^{(0)} = x_v$$

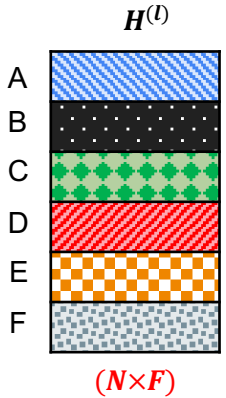
Initial node embeddings



A single layer of GNN: Graph Convolution – Forward pass

The Matrix form for the l^{th} layer

$$\underset{(1\times F)}{h_v^{(l+1)}} = \sigma\left(W^{(l)} \sum_{u\in N(v)} \underset{(1\times F)}{\frac{h_u^{(l)}}{|N(v)|}} + B^{(l)} \underset{(1\times F)}{h_v^{(l)}} \right), \forall l \in [0, 1 \dots L-1]$$



We stack multiple $h_v^{(l)}(1\times F)$ vectors together into one $H^{(l)}(N\times F)$ matrix.

A single layer of GNN: Graph Convolution – Forward pass

The Matrix form for the l^{th} layer

$$h_v^{(l+1)} = \sigma\left(W^{(l)} \sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|} + B^{(l)} h_v^{(l)} \right), \forall l \in [0, 1 \dots L-1]$$

$h_v^{(l+1)}$: $(1 \times F)$; $\sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|}$: $(1 \times F)$; $h_v^{(l)}$: $(1 \times F)$

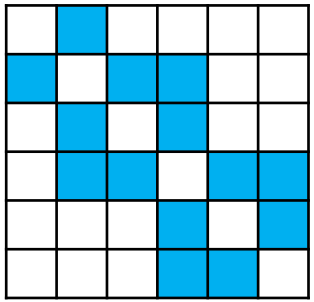
D^{-1}

$$\begin{bmatrix} \frac{1}{|N(v)|} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{1}{|N(v)|} \end{bmatrix}$$

$(N \times N)$

$\times$

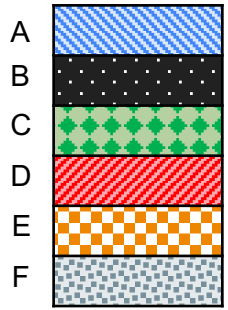
A



$(N \times N)$

$\times$

$H^{(l)}$



$(N \times F)$

A single layer of GNN: Graph Convolution – Forward pass

The Matrix form for the l^{th} layer

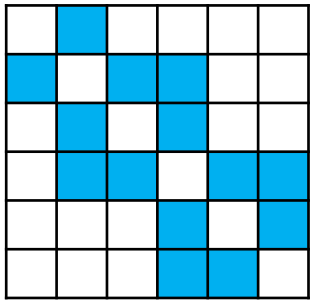
$$h_v^{(l+1)} = \sigma\left(W^{(l)} \sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|} + B^{(l)} h_v^{(l)}\right), \forall l \in [0, 1 \dots L-1]$$

$h_v^{(l+1)}$: $(1 \times F)$; $\sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|}$: $(1 \times F)$; $B^{(l)} h_v^{(l)}$: $(1 \times F)$

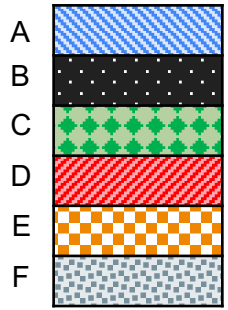
$$D^{-1} \times A \times H^{(l)} \times W^{(l)T}$$

$$\begin{bmatrix} \frac{1}{|N(v)|} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{1}{|N(v)|} \end{bmatrix}$$

$(N \times N)$



$(N \times N)$



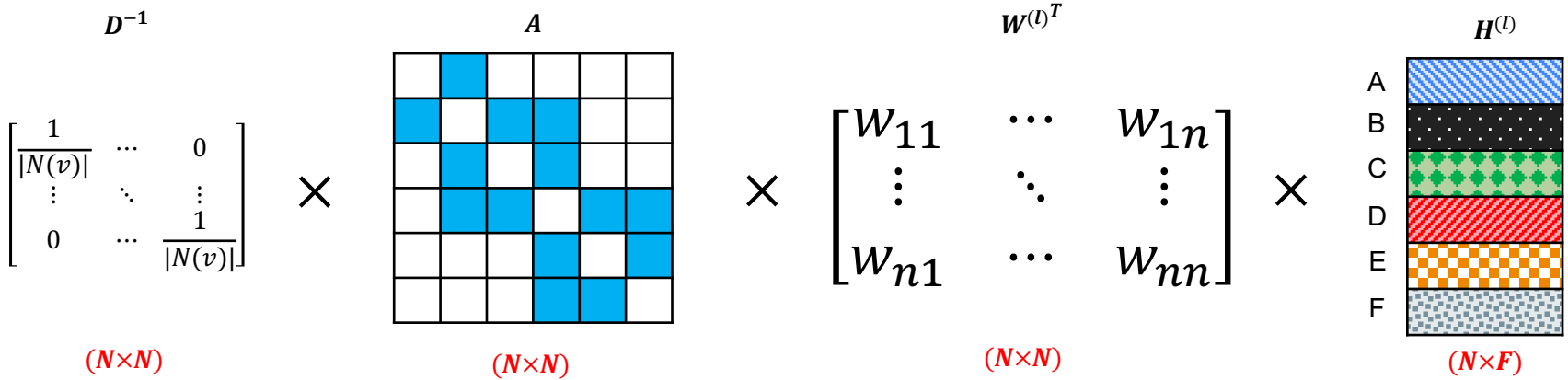
$(N \times F)$

$$\begin{bmatrix} w_{11} & \cdots & w_{1f} \\ \vdots & \ddots & \vdots \\ w_{f1} & \cdots & w_{ff} \end{bmatrix}$$

$(F \times F)$

A single layer of GNN: Graph Convolution – Forward pass

Food for thought



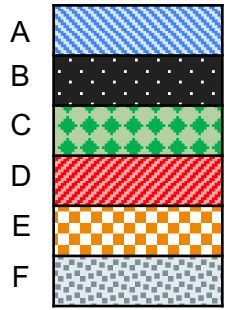
Why not multiply like this, with an $N\times N$ weight matrix?

A single layer of GNN: Graph Convolution – Forward pass

The Matrix form for the l^{th} layer

$$\underset{(1\times F)}{h_v^{(l+1)}} = \sigma\left(W^{(l)} \sum_{u\in N(v)} \underset{(1\times F)}{\frac{h_u^{(l)}}{|N(v)|}} + \underset{(1\times F)}{B^{(l)} h_v^{(l)}} \right), \forall l \in [0, 1 \dots L-1]$$

$$\underset{(N\times N)}{?} \times \underset{(N\times F)}{H^{(l)}} \times \underset{(F\times F)}{B^{(l)T}}$$

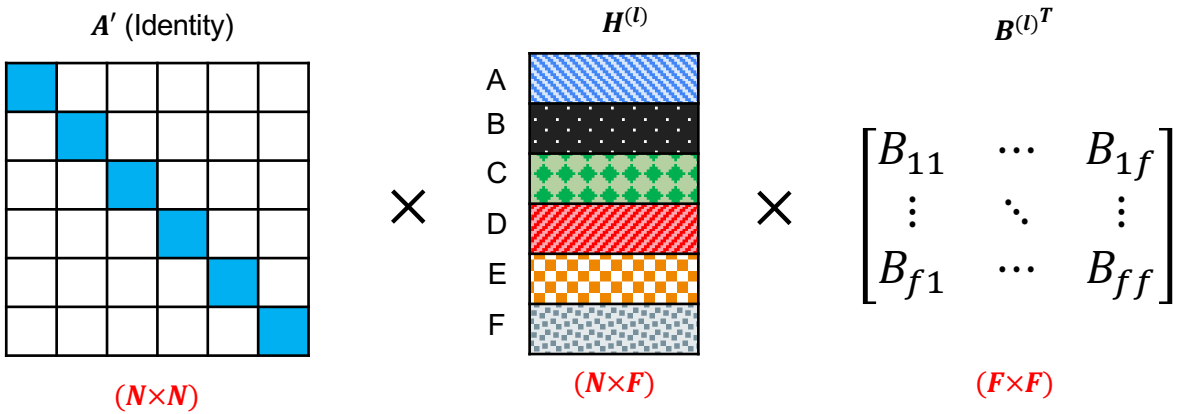


$$\begin{bmatrix} B_{11} & \cdots & B_{1f} \\ \vdots & \ddots & \vdots \\ B_{f1} & \cdots & B_{ff} \end{bmatrix}$$

A single layer of GNN: Graph Convolution – Forward pass

The Matrix form for the l^{th} layer

$$\underset{(1\times F)}{h_v^{(l+1)}} = \sigma\left(W^{(l)} \sum_{u\in N(v)} \underset{(1\times F)}{\frac{h_u^{(l)}}{|N(v)|}} + \underset{(1\times F)}{B^{(l)} h_v^{(l)}} \right), \forall l \in [0, 1 \dots L-1]$$



A single layer of GNN: Graph Convolution – Forward pass

The Matrix form for the l^{th} layer

$$h_v^{(l+1)} = \sigma\left(W^{(l)} \sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|} + B^{(l)} h_v^{(l)} \right), \forall l \in [0, 1 \dots L-1]$$

$$\boldsymbol{H^{(l+1)} = \sigma\left(D^{-1} A H^{(l)} W^{T} + A' H^{(l)} B^{T} \right)}, \forall l \in [0, 1 \dots L-1]$$

$$\boldsymbol{H^{(l+1)} = \sigma\left(\widehat{D}^{(-0.5)} \widehat{A} \widehat{D}^{(-0.5)} W'^{T} \right)}, \forall l \in [0, 1 \dots L-1]$$

Forward equation for GCN

Poll 4 (Not on piazza)

Which of the following are true statements? (Select all that apply)

a. LSTMs and GRUs are permutation invariant since they will eventually process every element of the sequence, and hence reach the same output for any permutation.

b. In GNNs to incorporate information from nodes that are k-hops away, we would need a model that has at most k-layers.

c. In GNNs to incorporate information from nodes that are k-hops away, we would need a model that has at least k-layers.

d. Since transformers are not permutation invariant, you cannot use the self-attention mechanism in GNNs.

Poll 4

Which of the following are true statements? (Select all that apply)

a. LSTMs and GRUs are permutation invariant since they will eventually process every element of the sequence, and hence reach the same output for any permutation.

b. In GNNs to incorporate information from nodes that are k-hops away, we would need a model that has at most k-layers.

c. In GNNs to incorporate information from nodes that are k-hops away, we would need a model that has at least k-layers.

d. Since transformers are not permutation invariant, you cannot use the self-attention mechanism in GNNs.

Reference

- Kipf, T.N. and Welling, M., 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.
- Stanford CS 224 W