Deep Learning

Transformers and Graph Neural Networks

Abuzar Khan, Yue Jian

11-785, Fall 2022

Part 1

Transformers

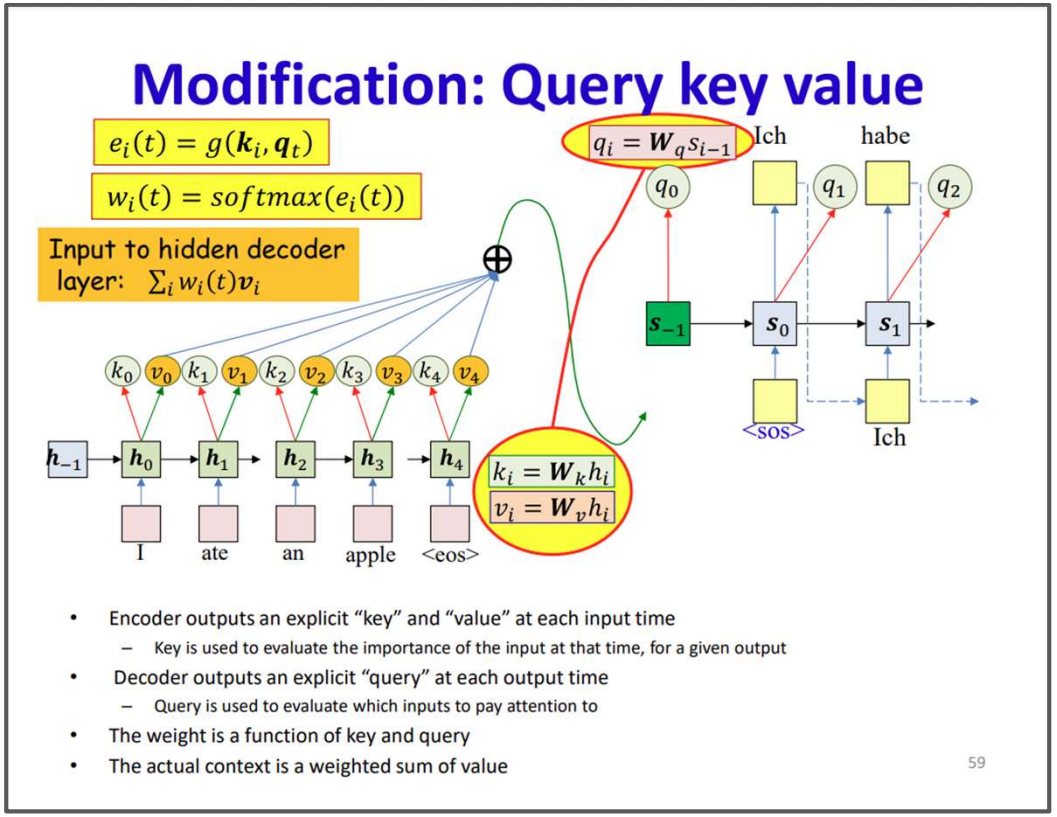
Credit where due

Probably influenced by:

- Louis-Philippe Morency, **11777**
- Graham Neubig, **11711**
- (obviously) Bhiksha Raj, **11-you-know-your-course-number**
- Bunch of youtube videos
- Medium Articles

Recall

1. Queries, Keys, and Values



Recall

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

Self Attention

$q_i = \boldsymbol{W}_q h_i$

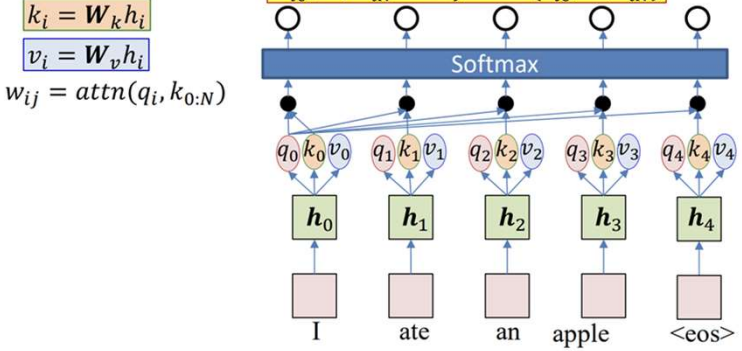
$k_i = \boldsymbol{W}_k h_i$

$v_i = \boldsymbol{W}_v h_i$

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

$e_{ij} = q_i^T k_j$

$w_{i0}, \ldots, w_{iN} = softmax(e_{i0}, \ldots, e_{iN})$

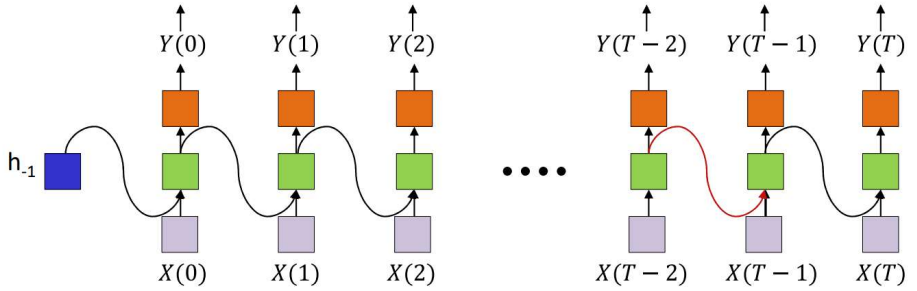


- For each word, we compute an attention weight between that word and all other words
  - The raw attention of the ith word to the jth word is a function of query q_i and key k_j
  - The raw attention values are put through a softmax to get the final attention weights

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

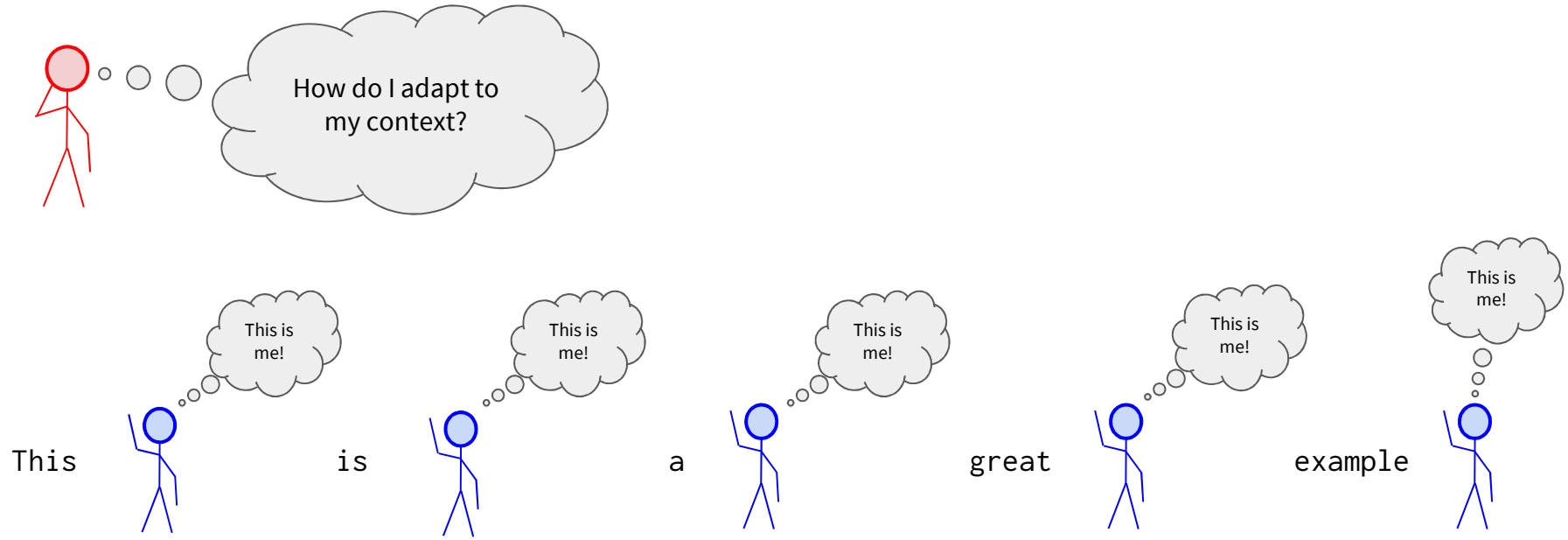
Query: This is what **pays the attention**

Values: These are **paid attention to**

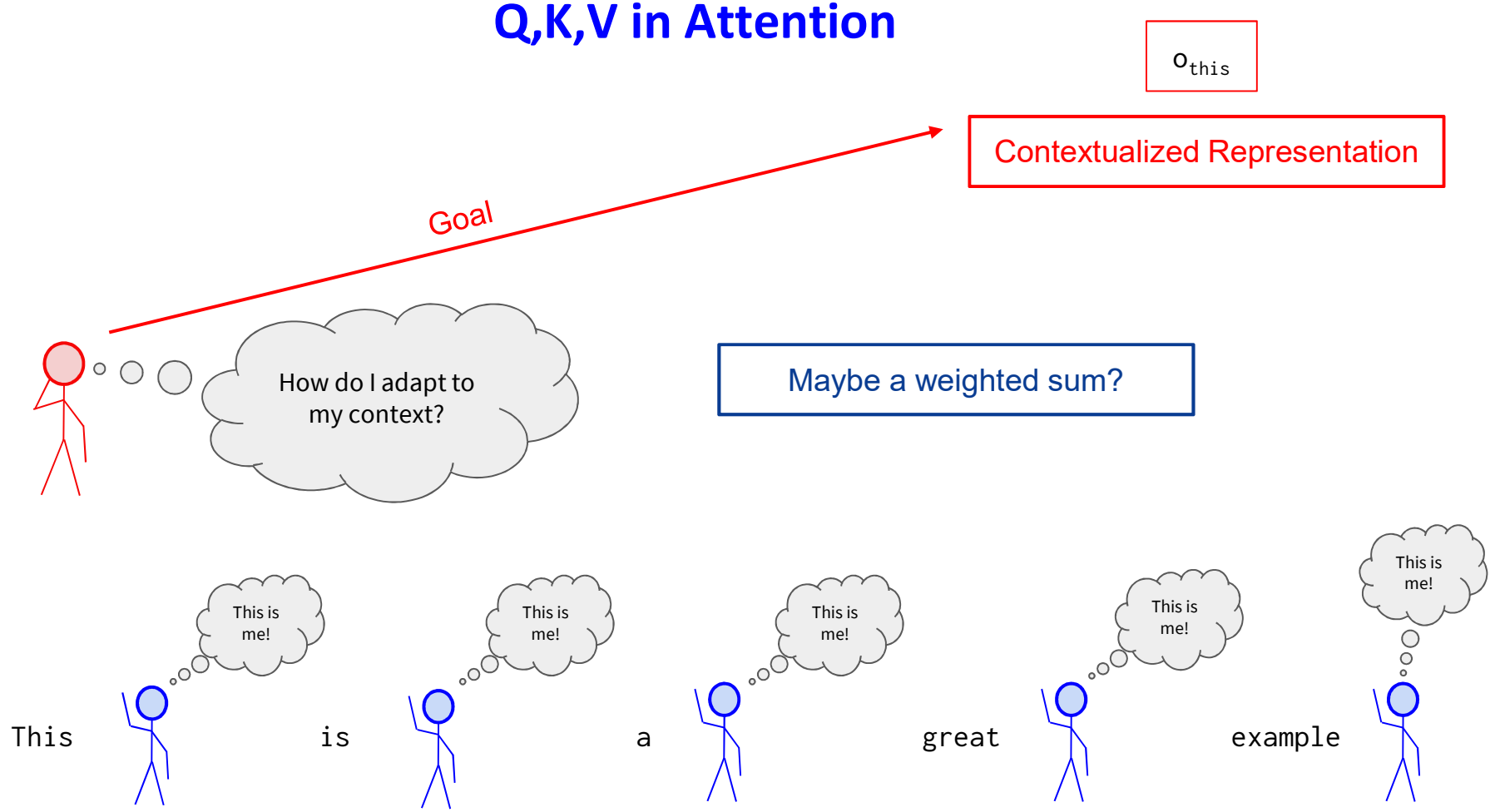
Keys: These help queries figure out **how much attention to pay** to each of the values

Attention Weights: How much attention to pay.

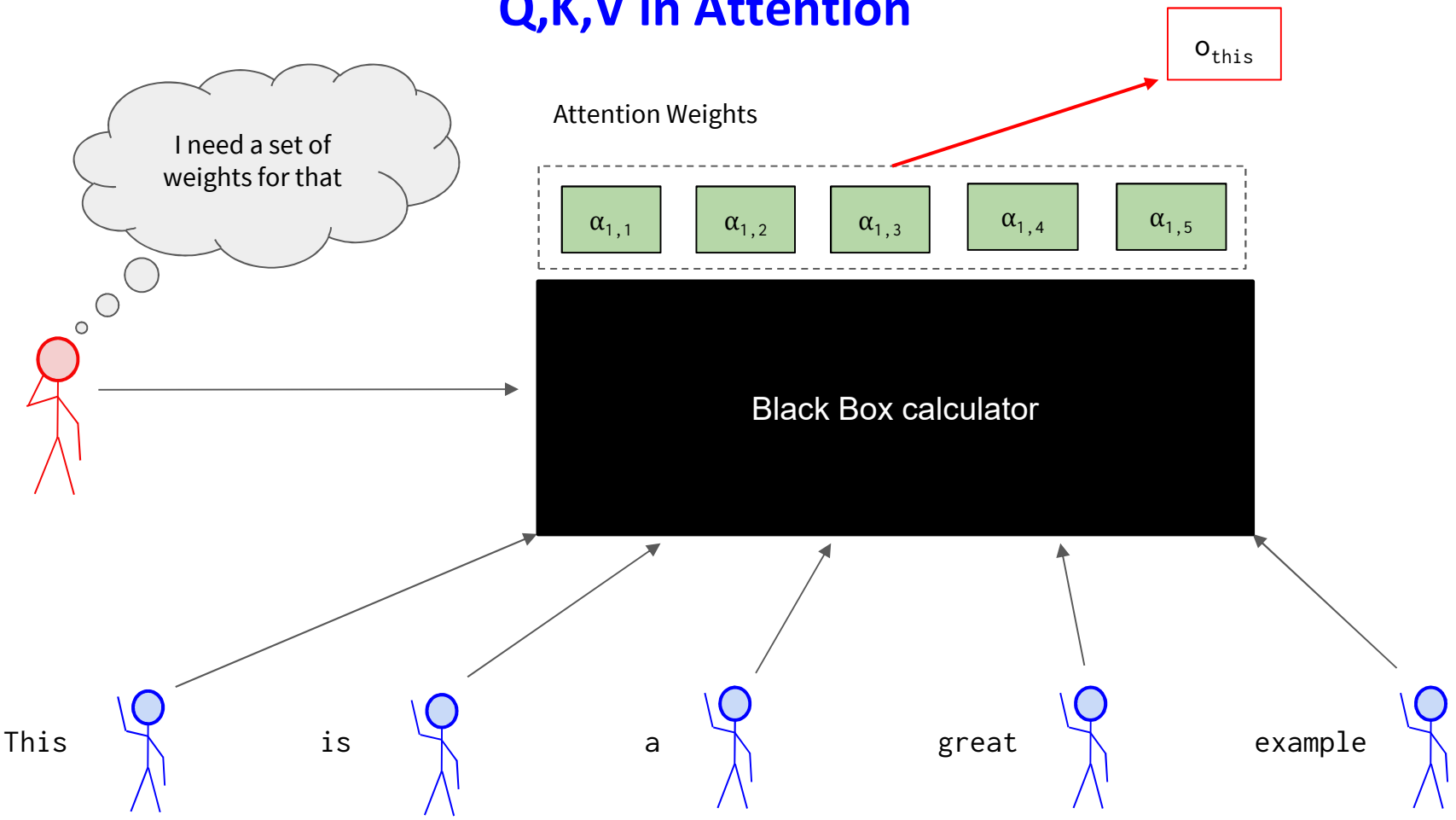
Q,K,V in Attention



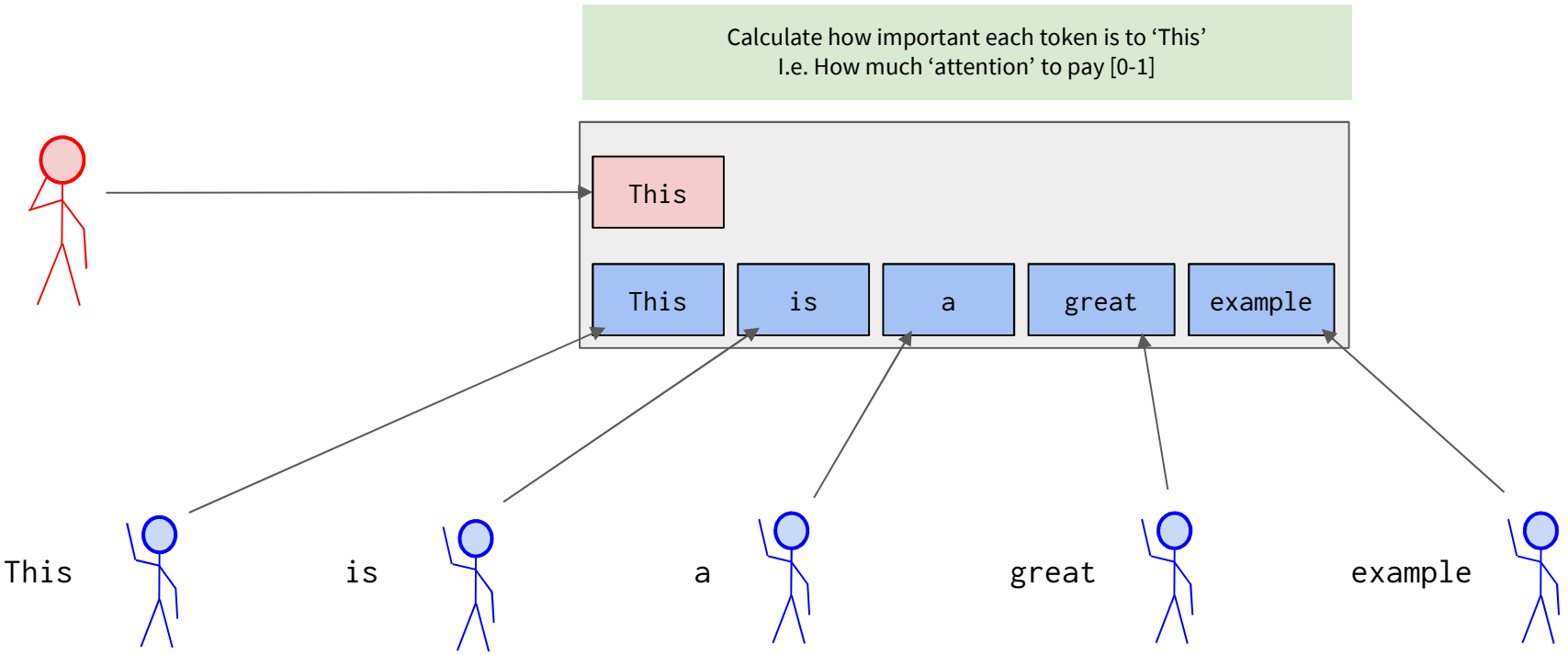
Q,K,V in Attention



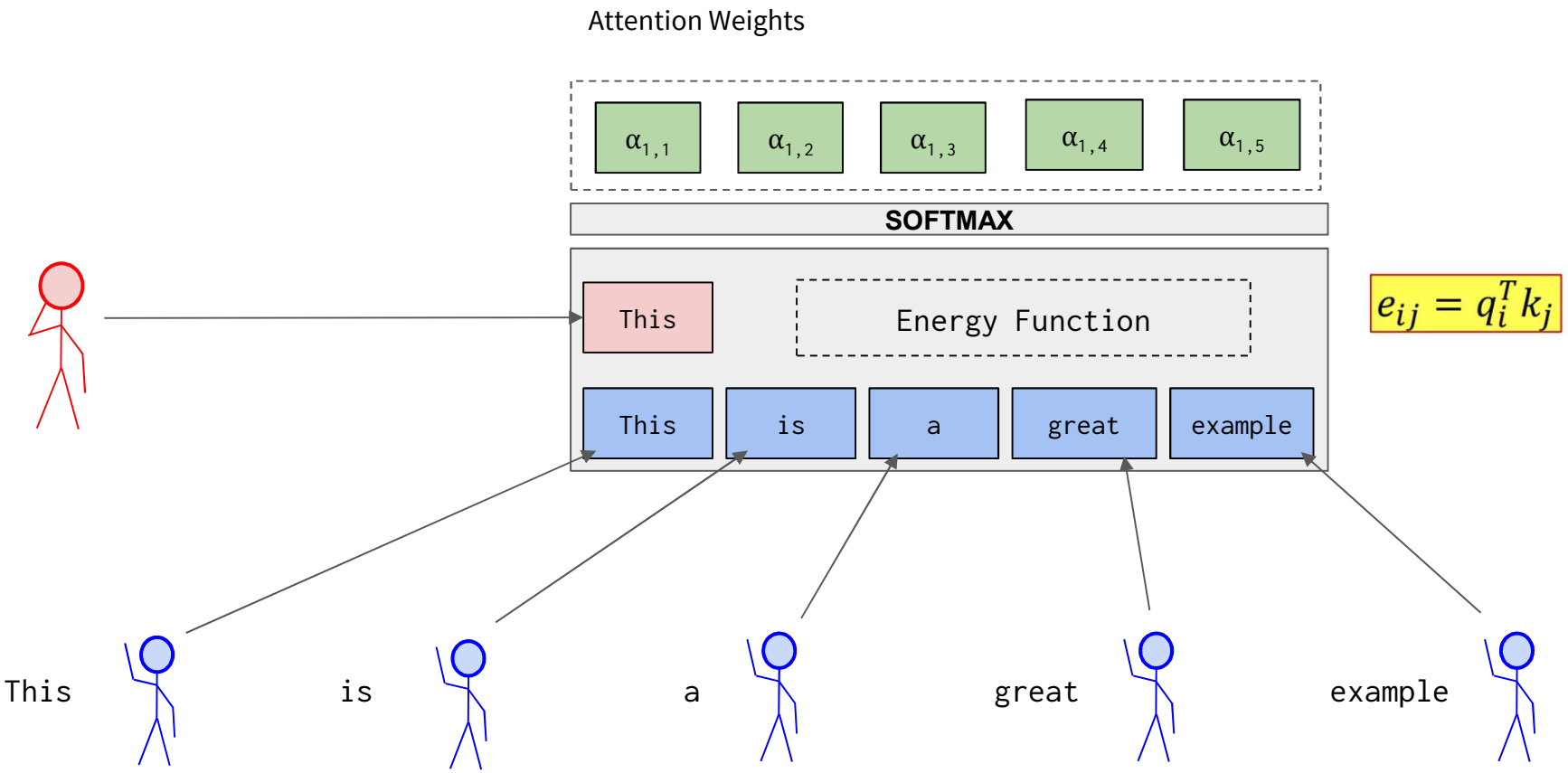
Q,K,V in Attention



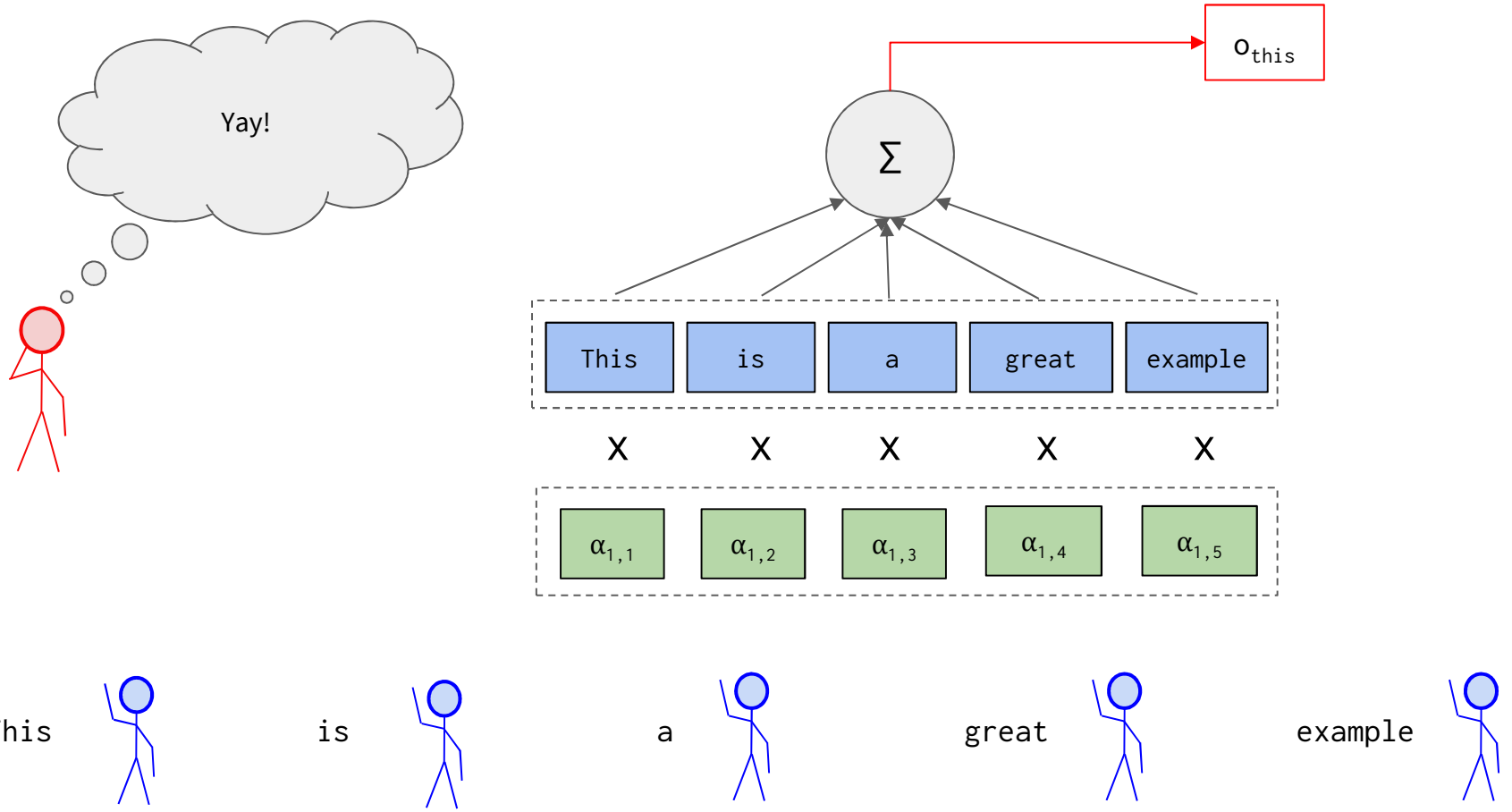
Q,K,V in Attention



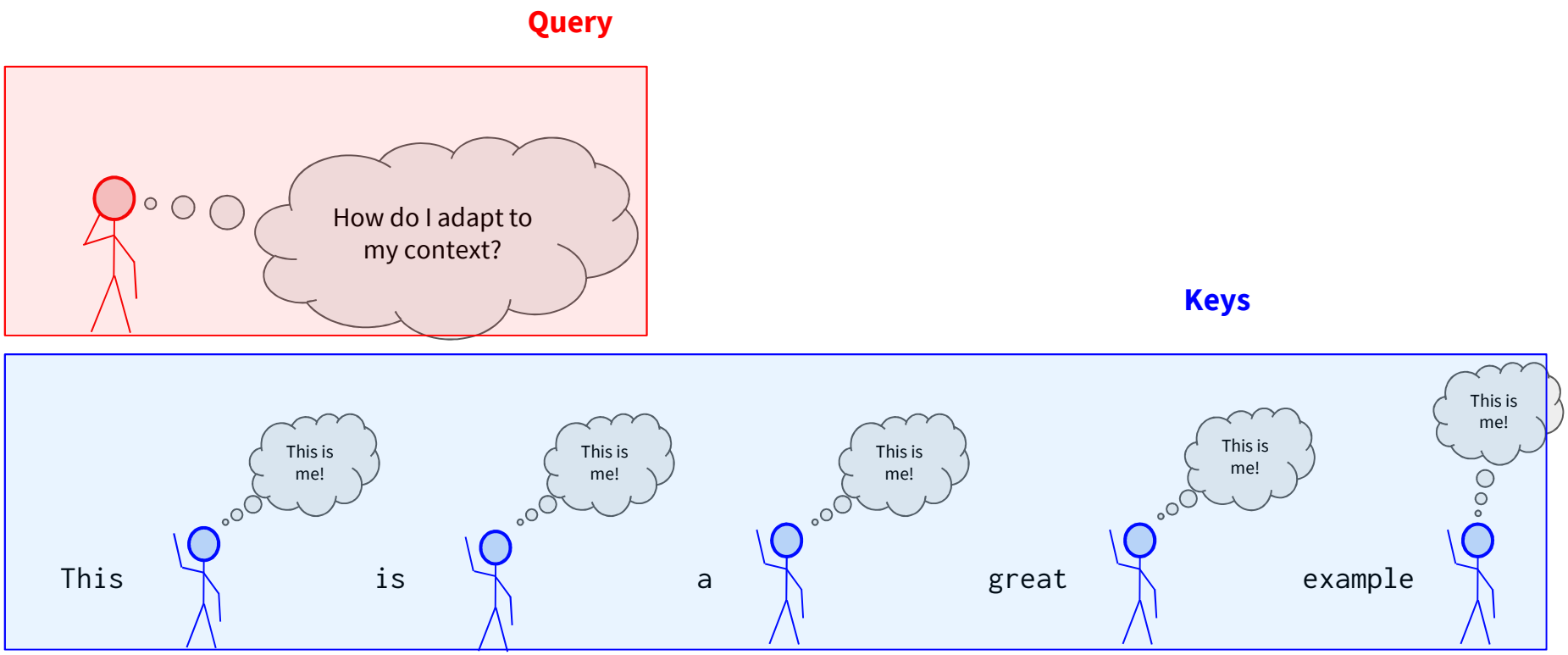
Q,K,V in Attention



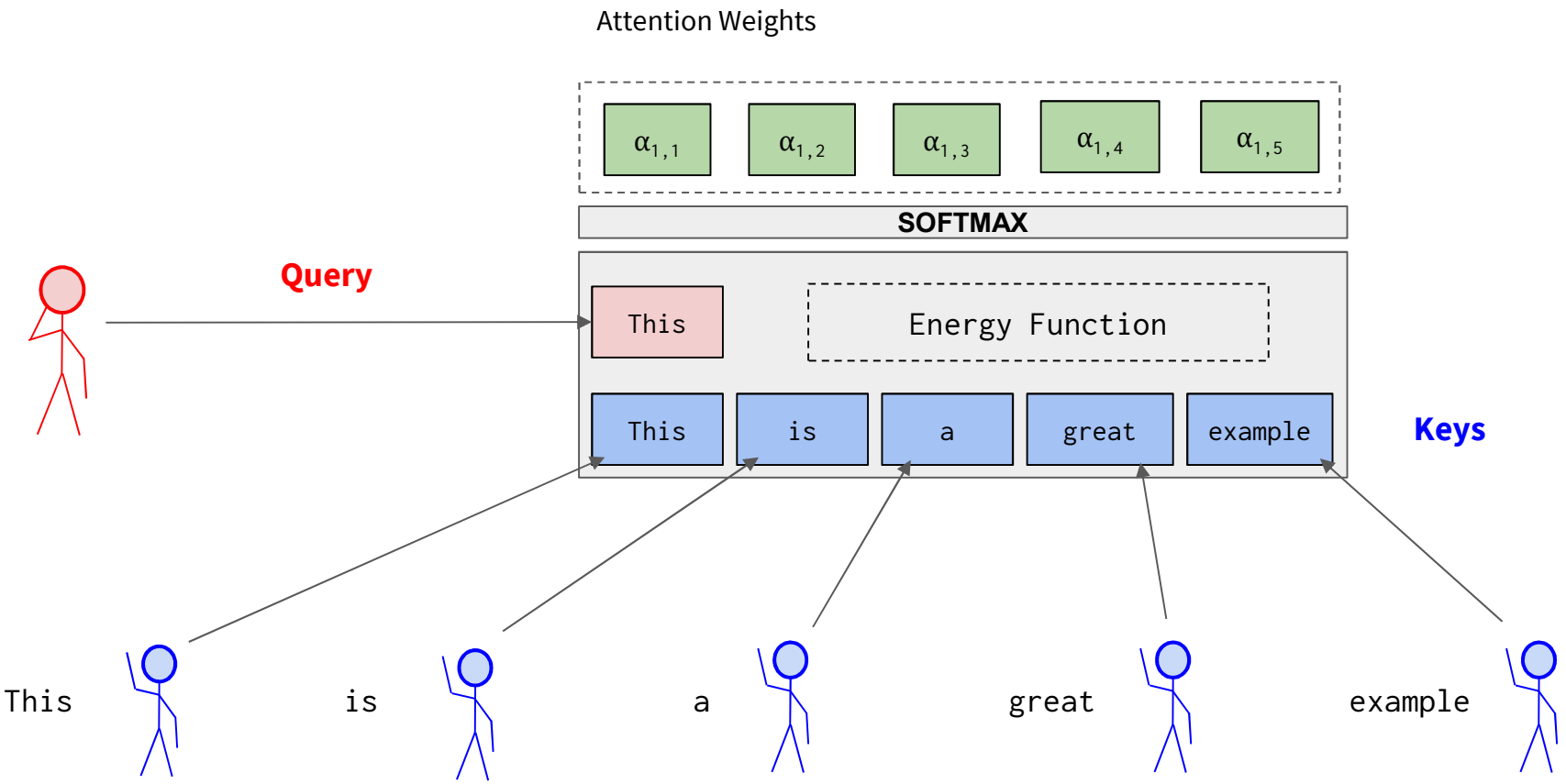
Q,K,V in Attention



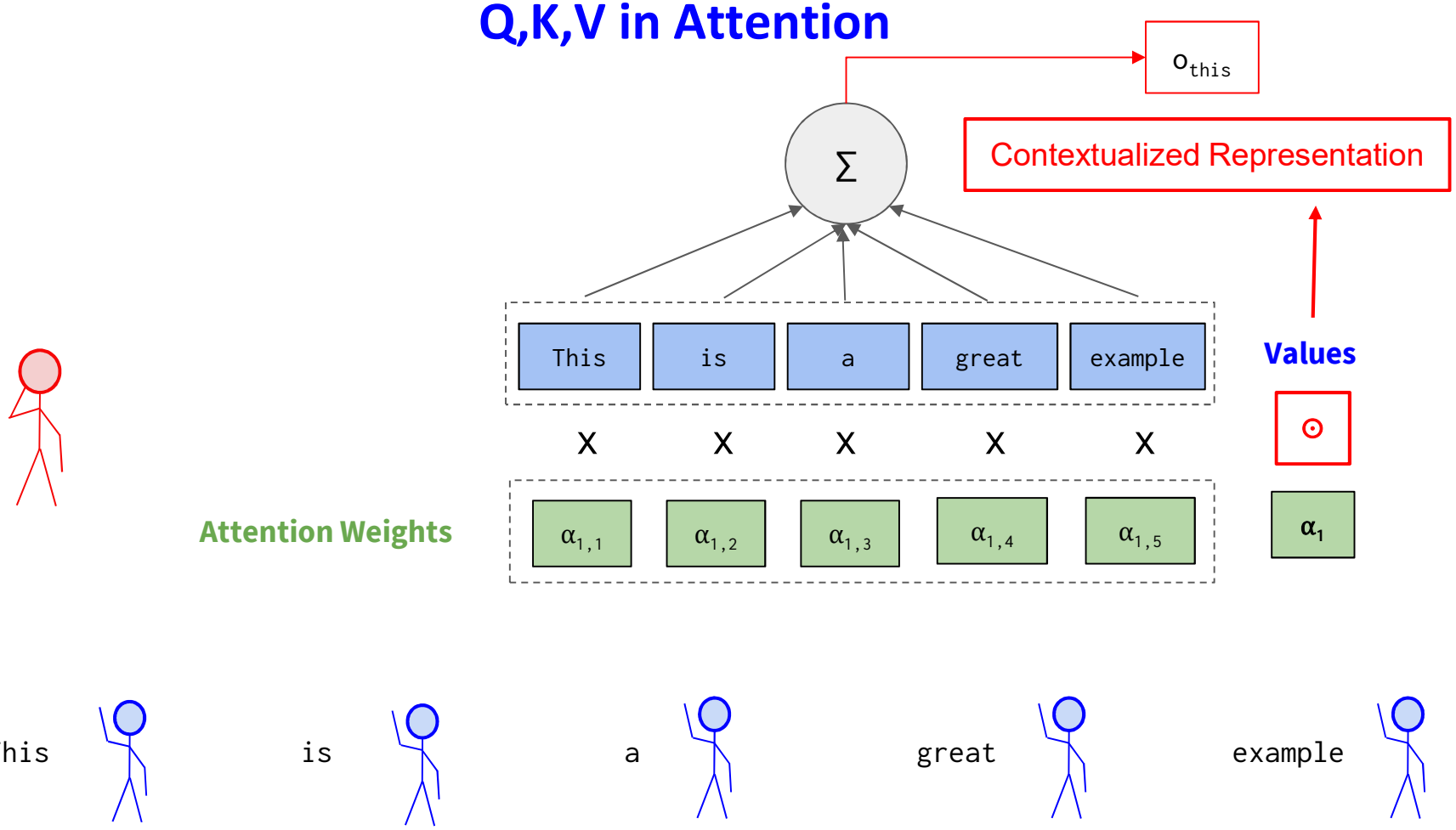
Q,K,V in Attention



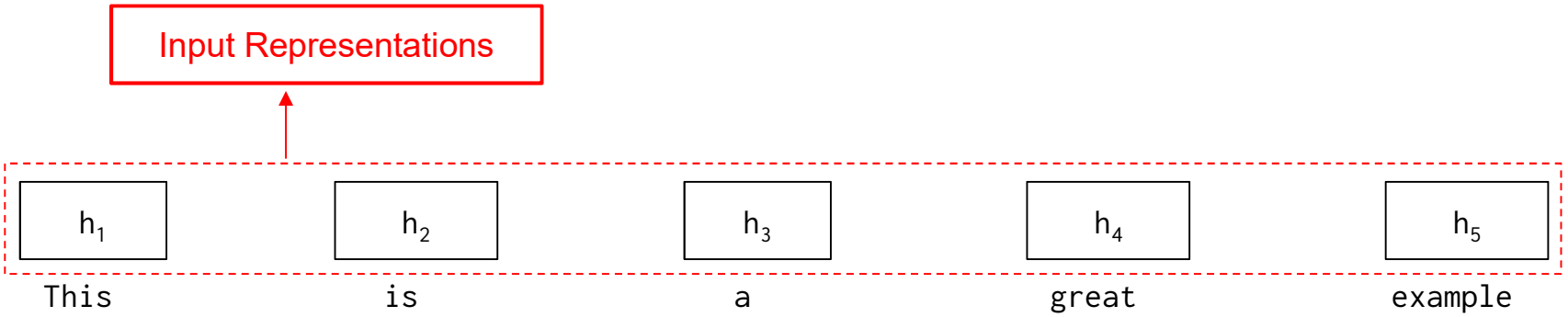
Q,K,V in Attention



Q,K,V in Attention

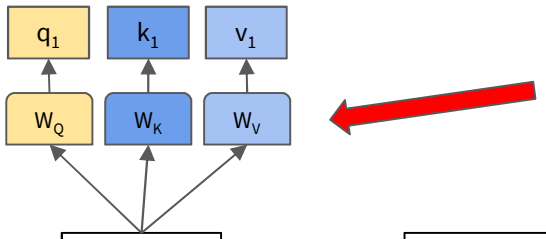


Self Attention



Self Attention

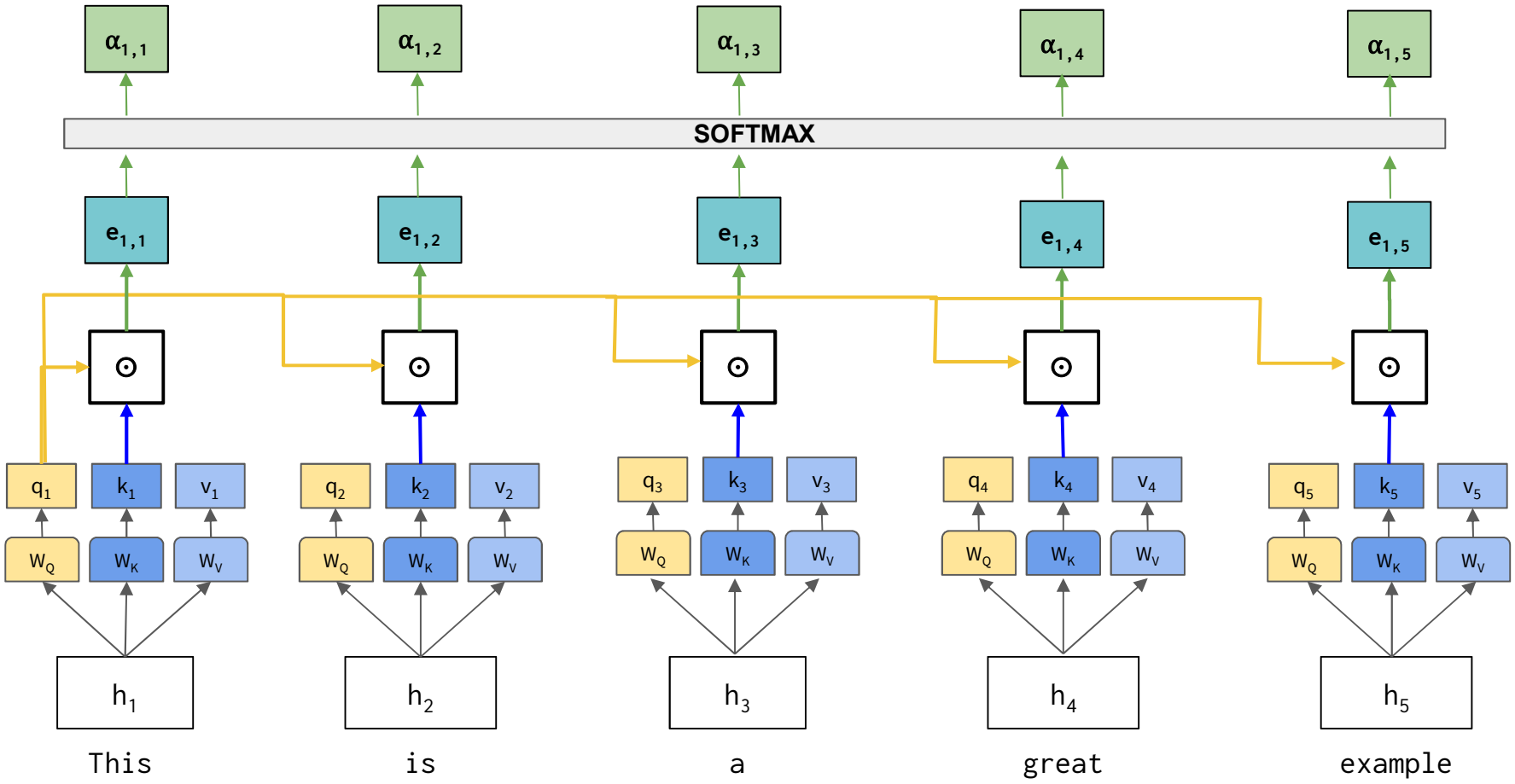
$$q_i = W_Q^T x_i$$
$$k_i = W_K^T x_i$$
$$v_i = W_V^T x_i$$

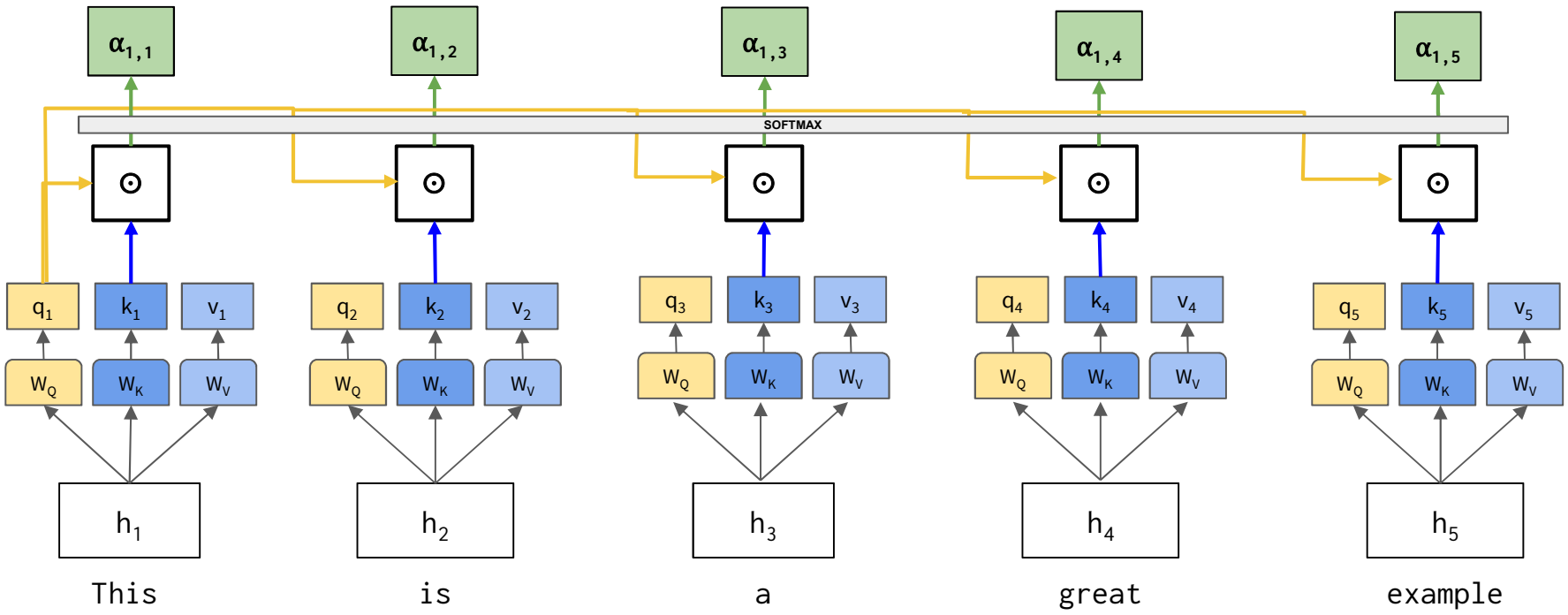


Projection: gives **q,k,v** of dimensions $\mathbf{d_q}$, $\mathbf{d_k}$, $\mathbf{d_v}$ (often the same)

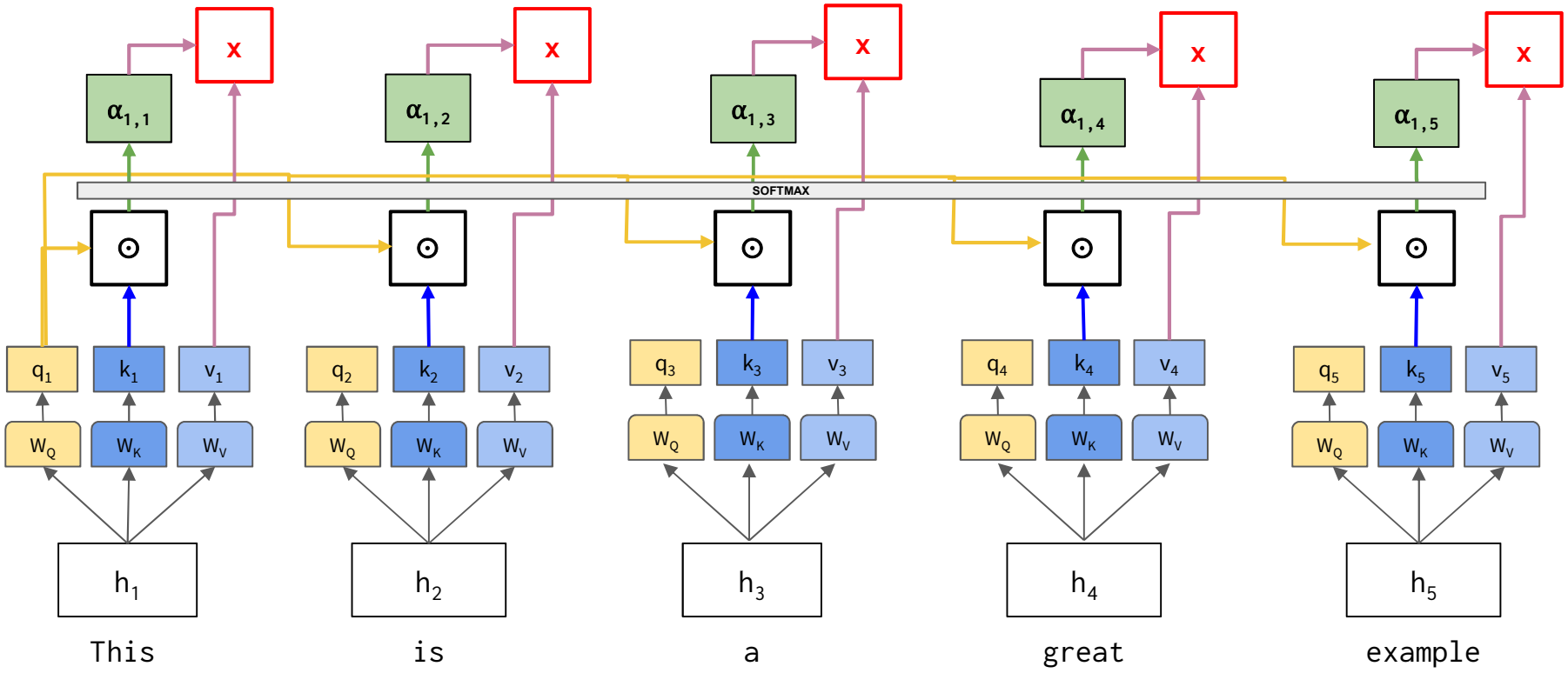
h_1	h_2	h_3	h_4	h_5
This	is	a	great	example

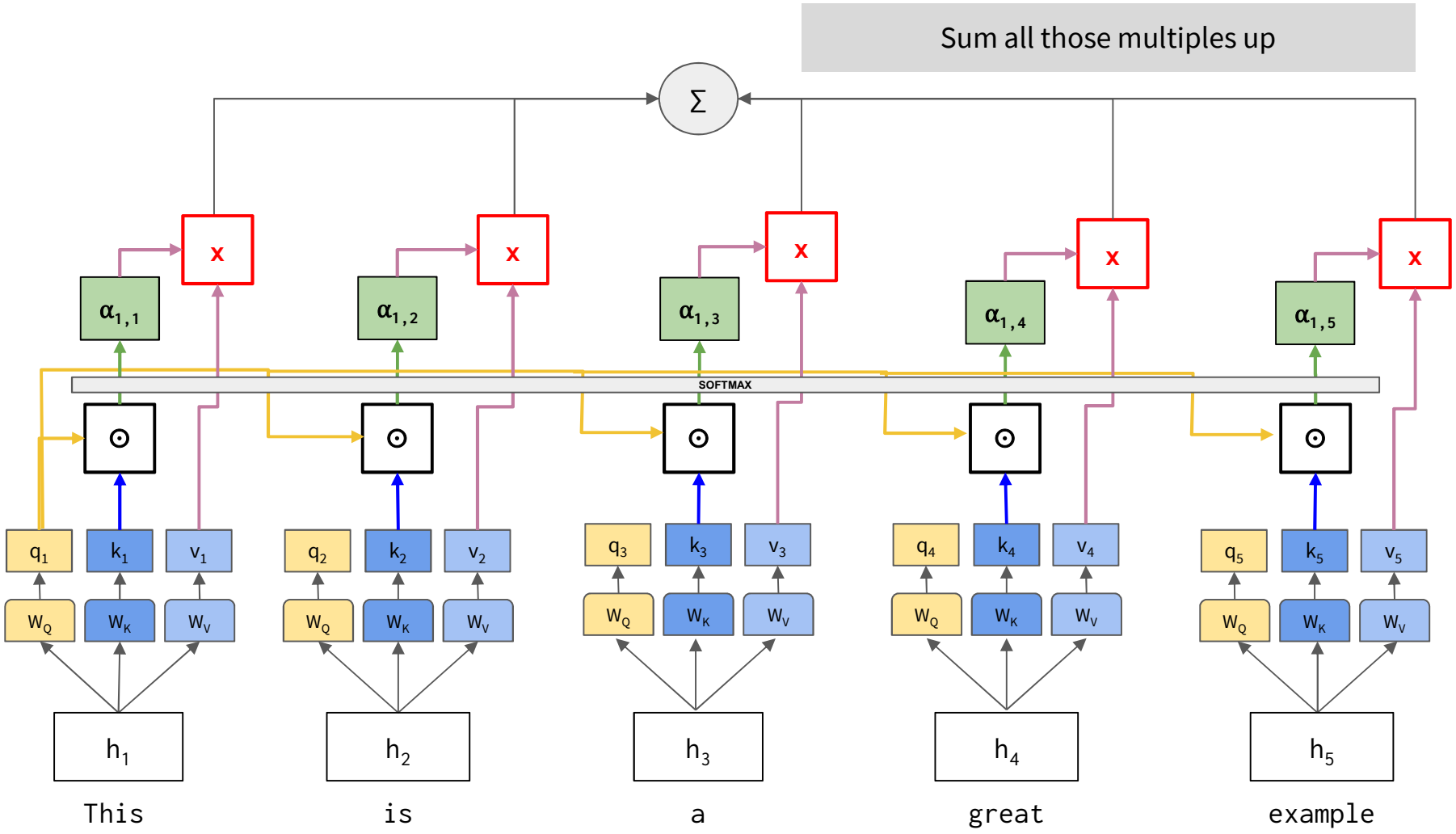
$\alpha_{m,n}$ = How important is token **n** to token **m**'s contextual meaning?

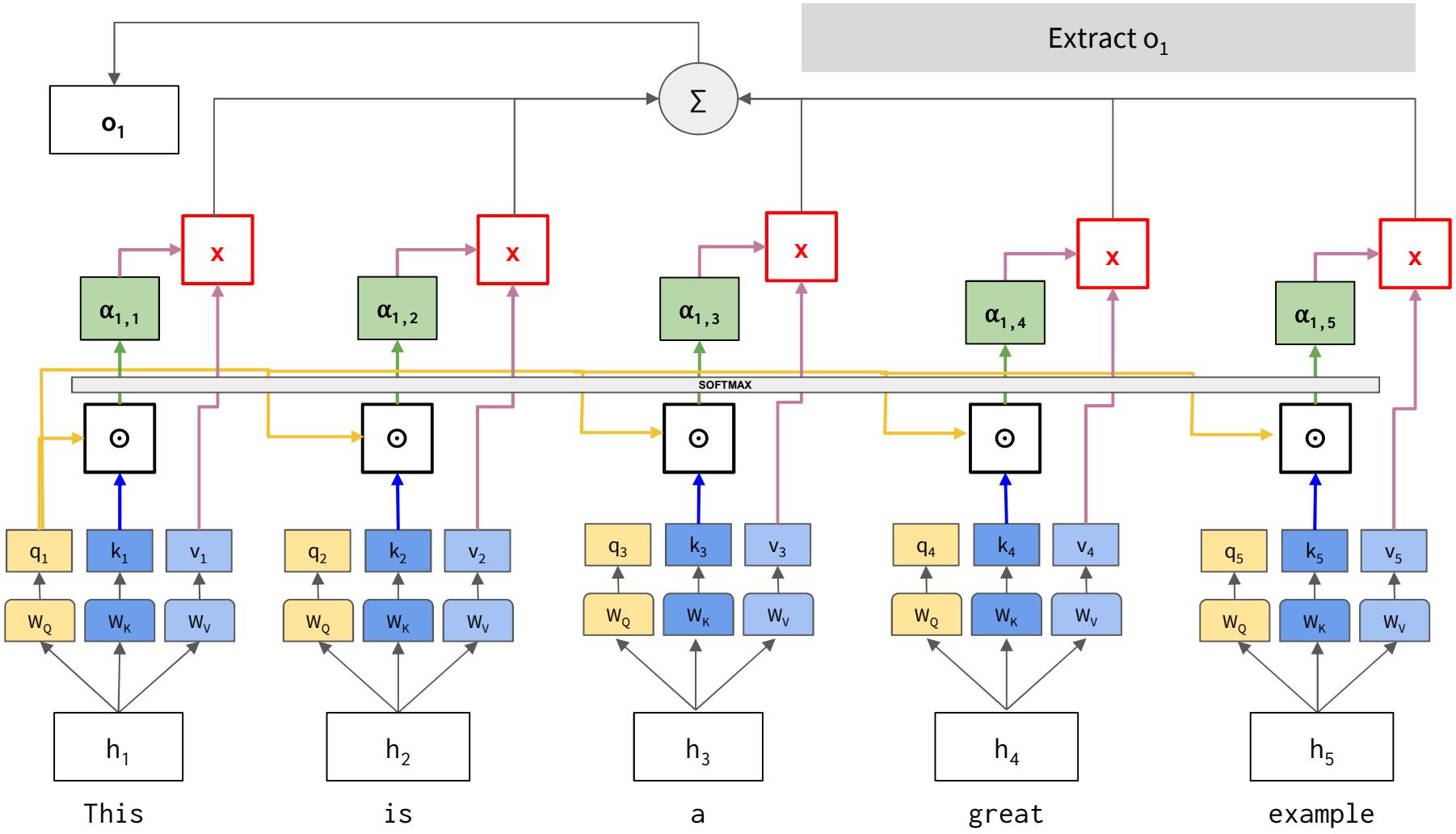


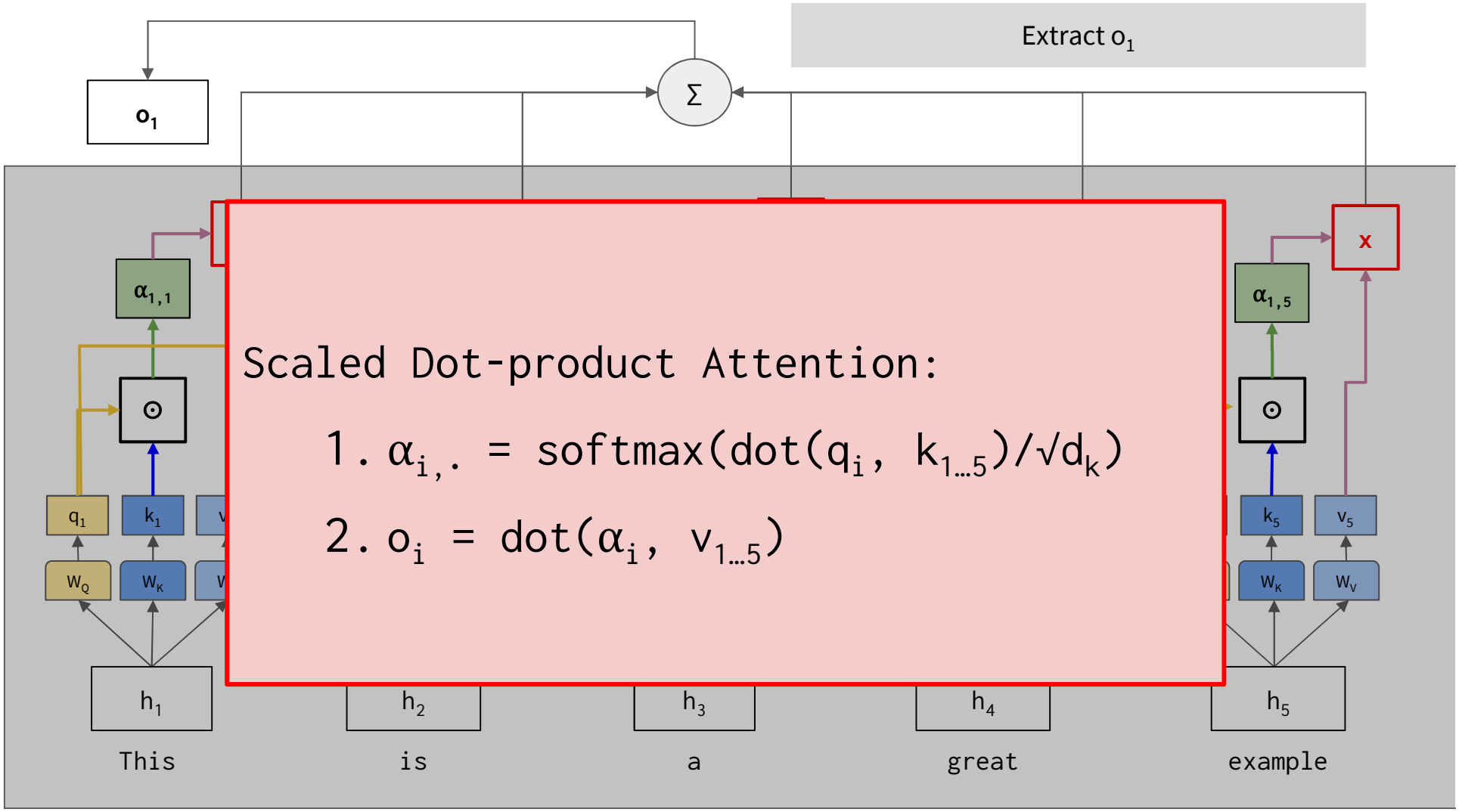

α1,1
α1,2
α1,3
α1,4
α1,5
SOFTMAX
q1
k1
v1
q2
k2
v2
q3
k3
v3
q4
k4
v4
q5
k5
v5
WQ
WK
WV
h1
h2
h3
h4
h5
This
is
a
great
example

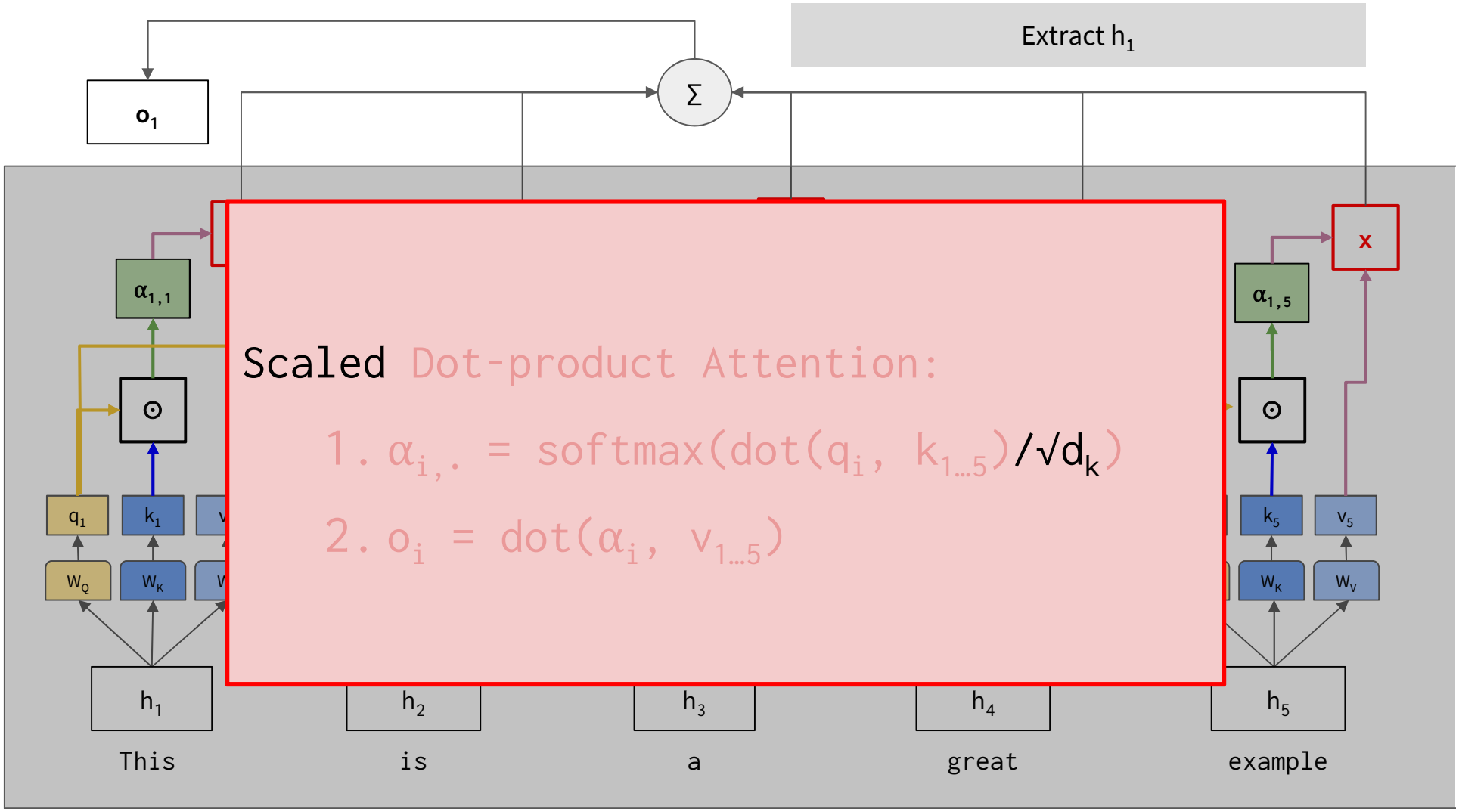
Multiply each $\alpha_{1,i}$ with $\mathbf{v}_i$

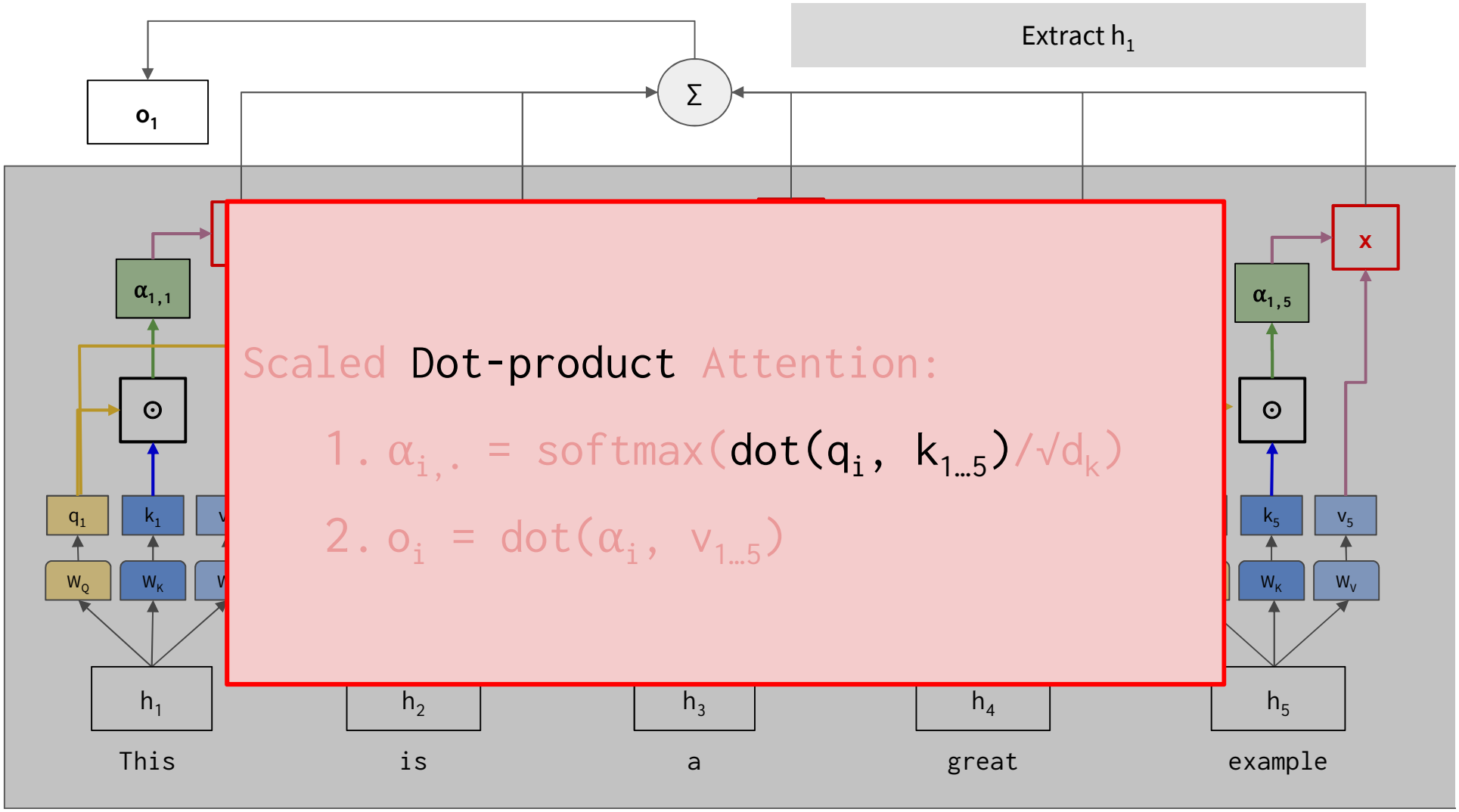


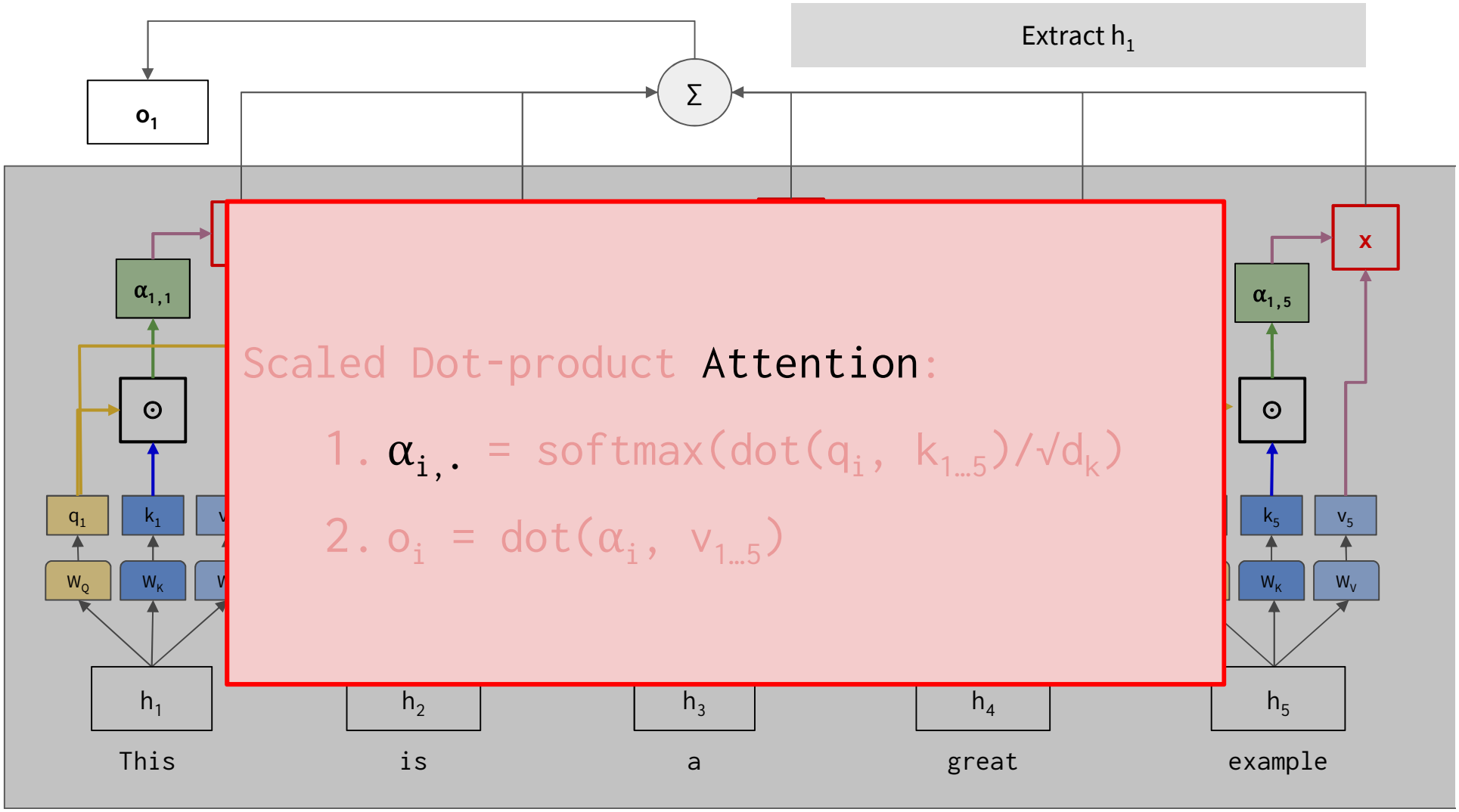

Sum all those multiples up
Σ
x
x
x
x
x
α1,1
α1,2
α1,3
α1,4
α1,5
SOFTMAX
⊙
⊙
⊙
⊙
⊙
q1
k1
v1
q2
k2
v2
q3
k3
v3
q4
k4
v4
q5
k5
v5
WQ
WK
WV
h1
h2
h3
h4
h5
This
is
a
great
example

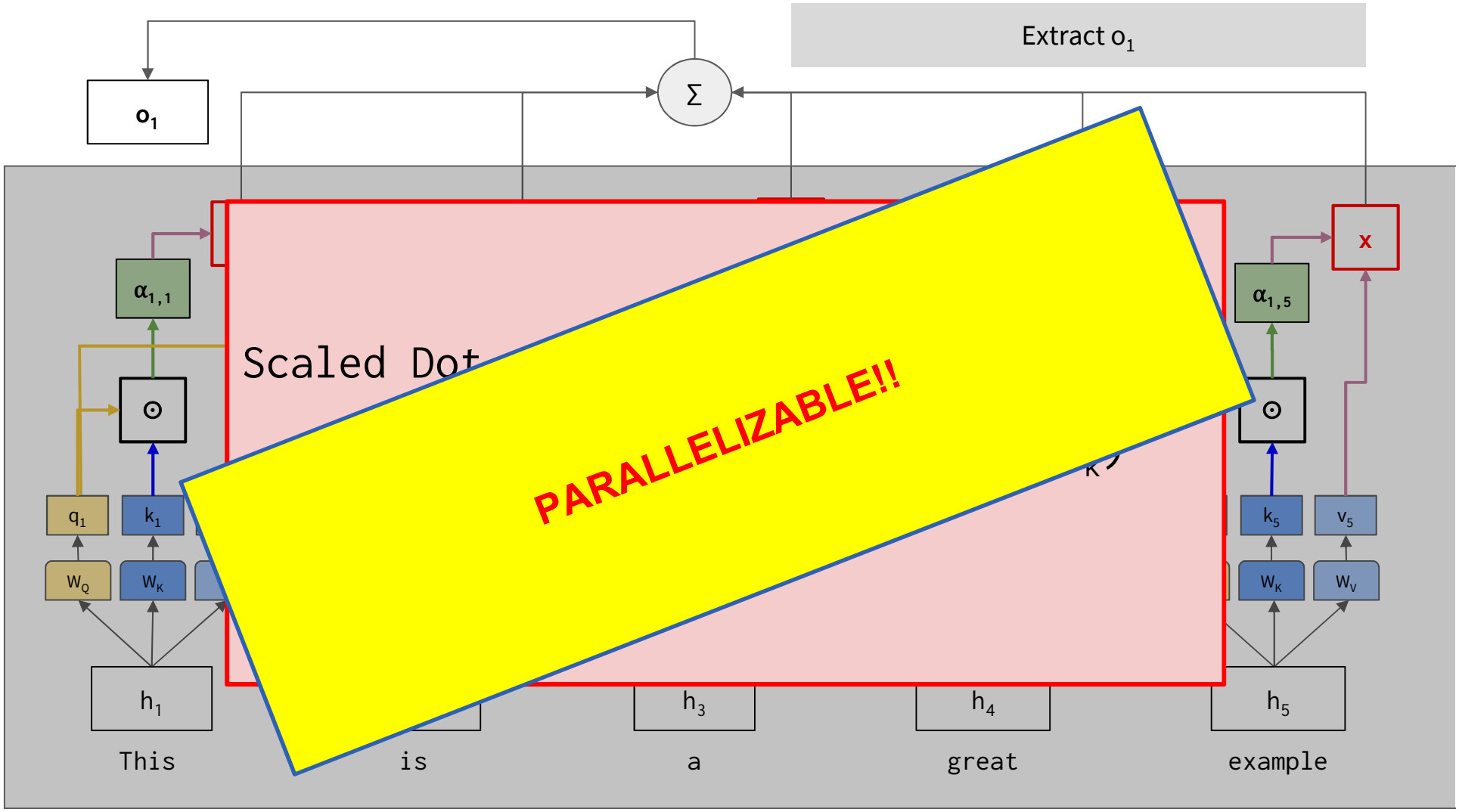

Extract o_1
o_1
Σ
x
x
x
x
x
$\alpha_{1,1}$
$\alpha_{1,2}$
$\alpha_{1,3}$
$\alpha_{1,4}$
$\alpha_{1,5}$
SOFTMAX
⊙
q_1 k_1 v_1
q_2 k_2 v_2
q_3 k_3 v_3
q_4 k_4 v_4
q_5 k_5 v_5
W_Q W_K W_V
h_1 h_2 h_3 h_4 h_5
This is a great example


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

Example

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

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

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

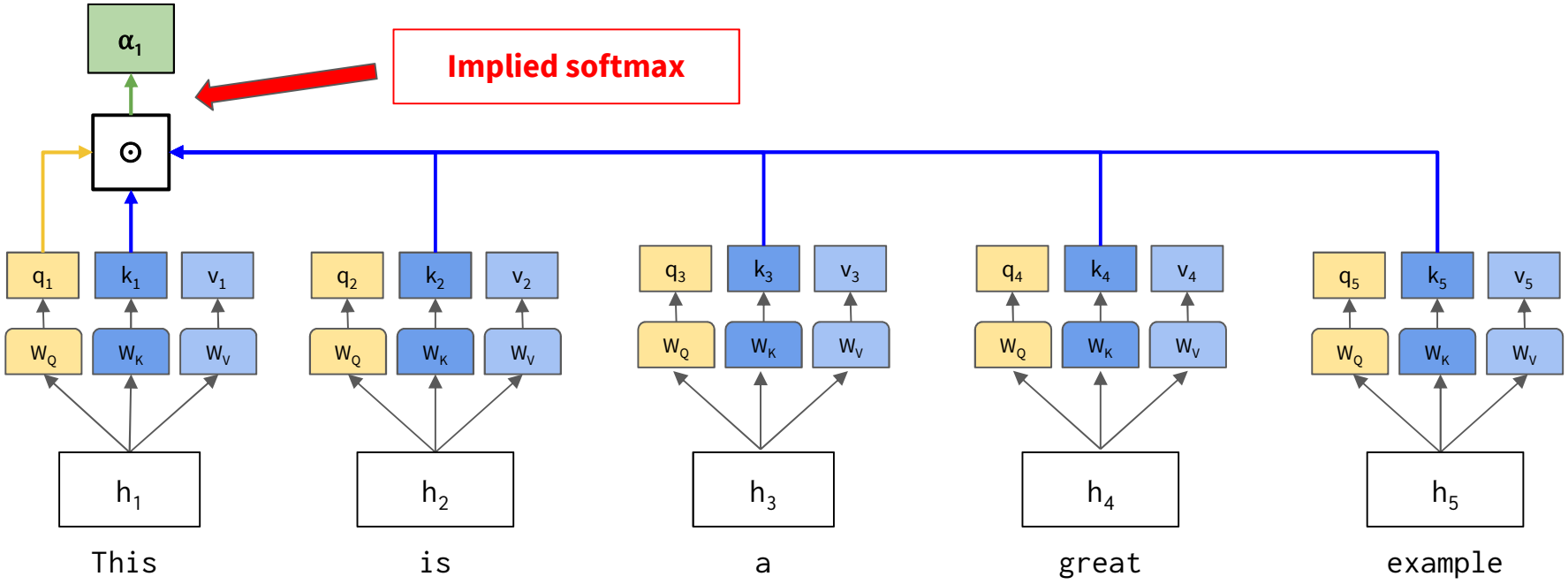
  $e_2 = q_1 k_2^T / \sqrt{4} = 22.5$

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

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

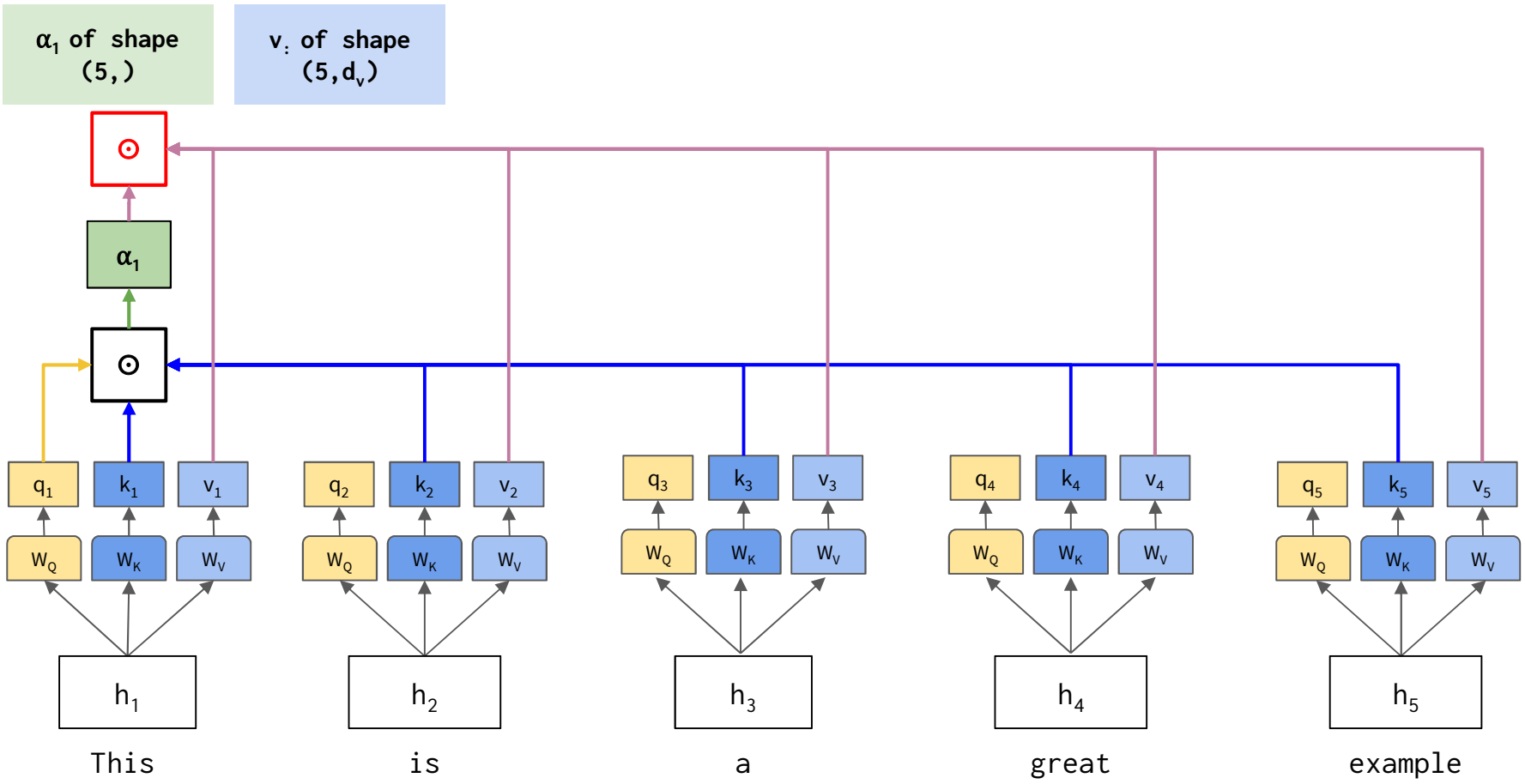
Self Attention

α_1 of shape (5,)



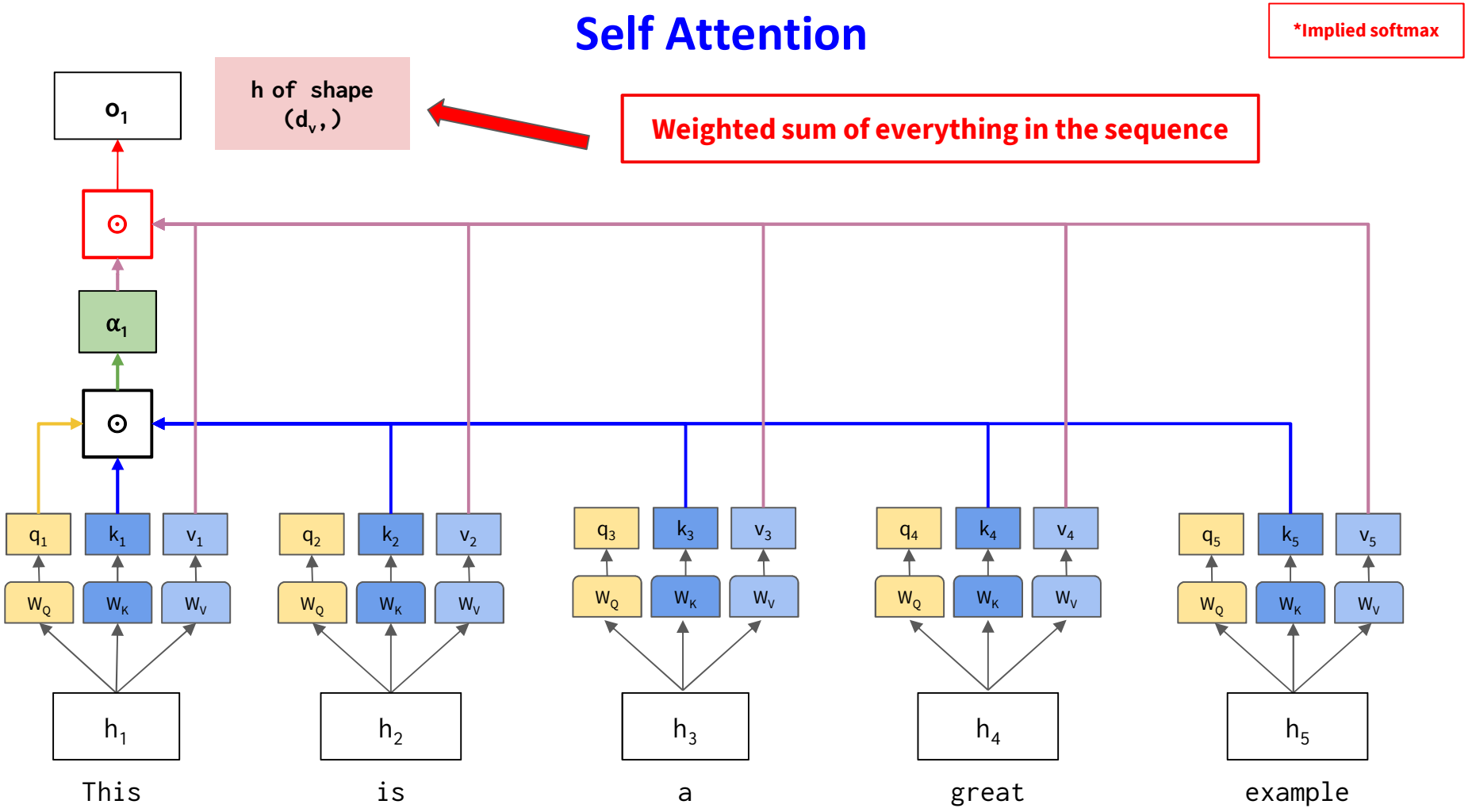
Self Attention

*Implied softmax



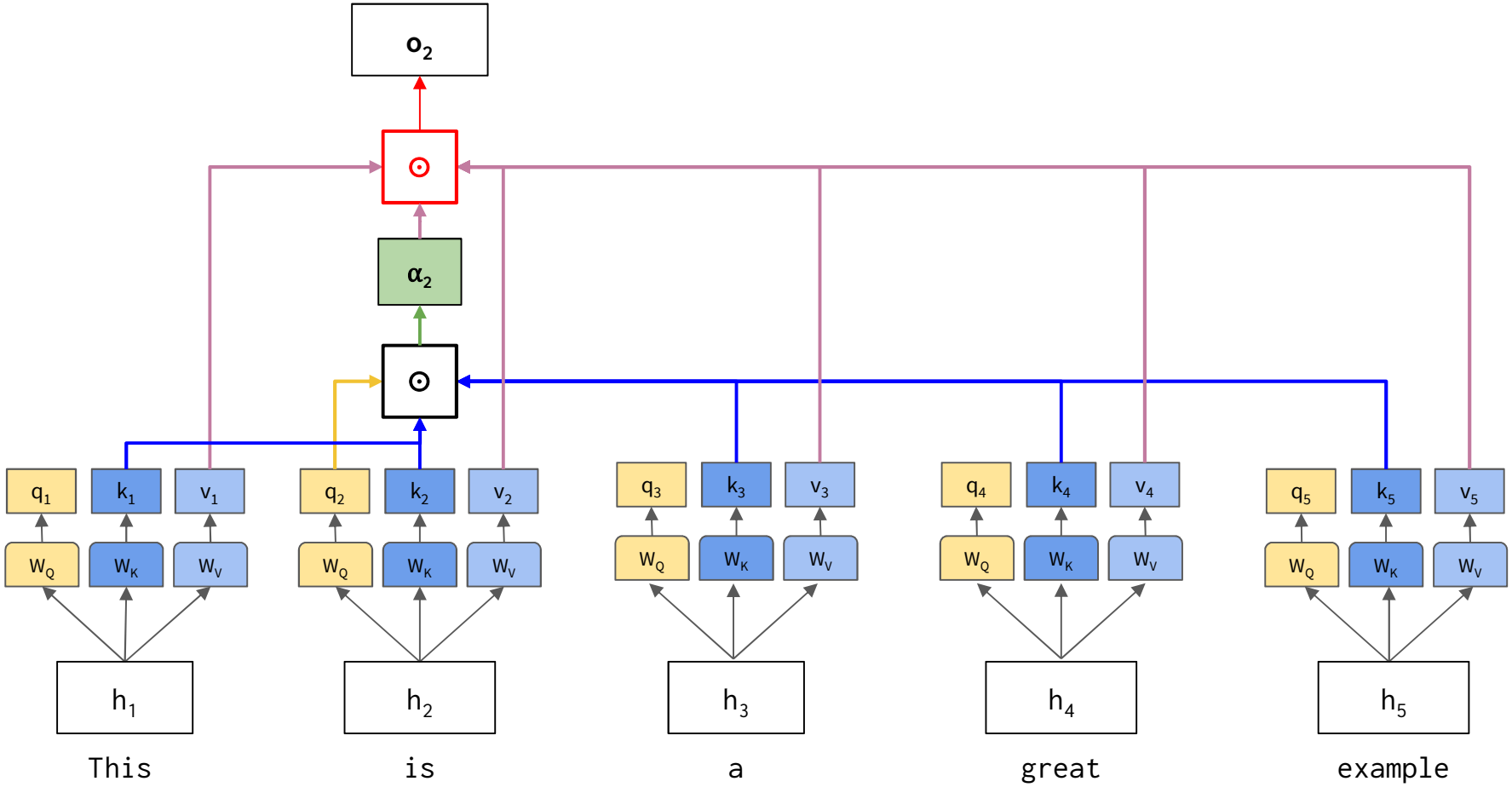
Self Attention

*Implied softmax



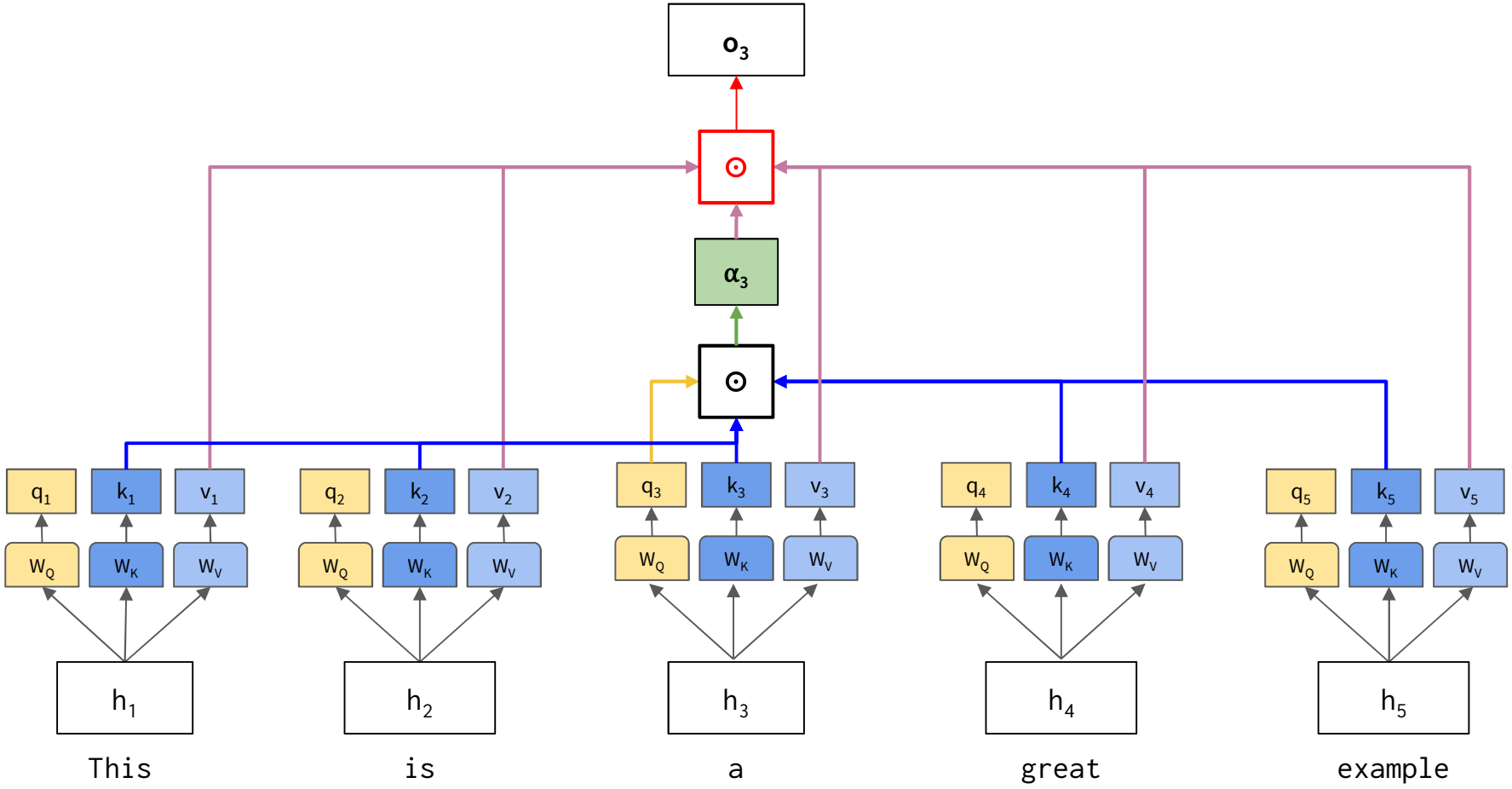
Self Attention

*Implied softmax



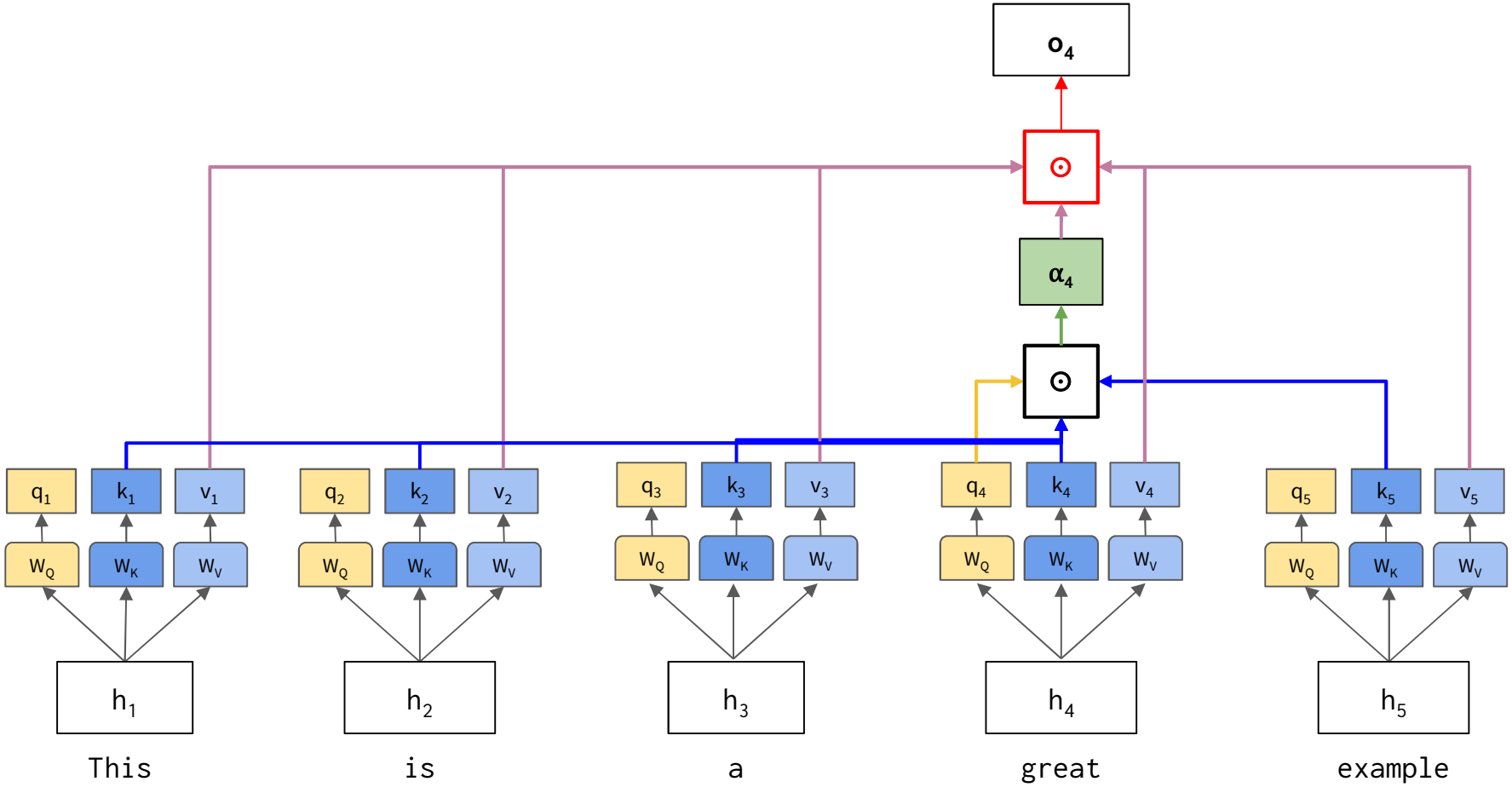
Self Attention

*Implied softmax

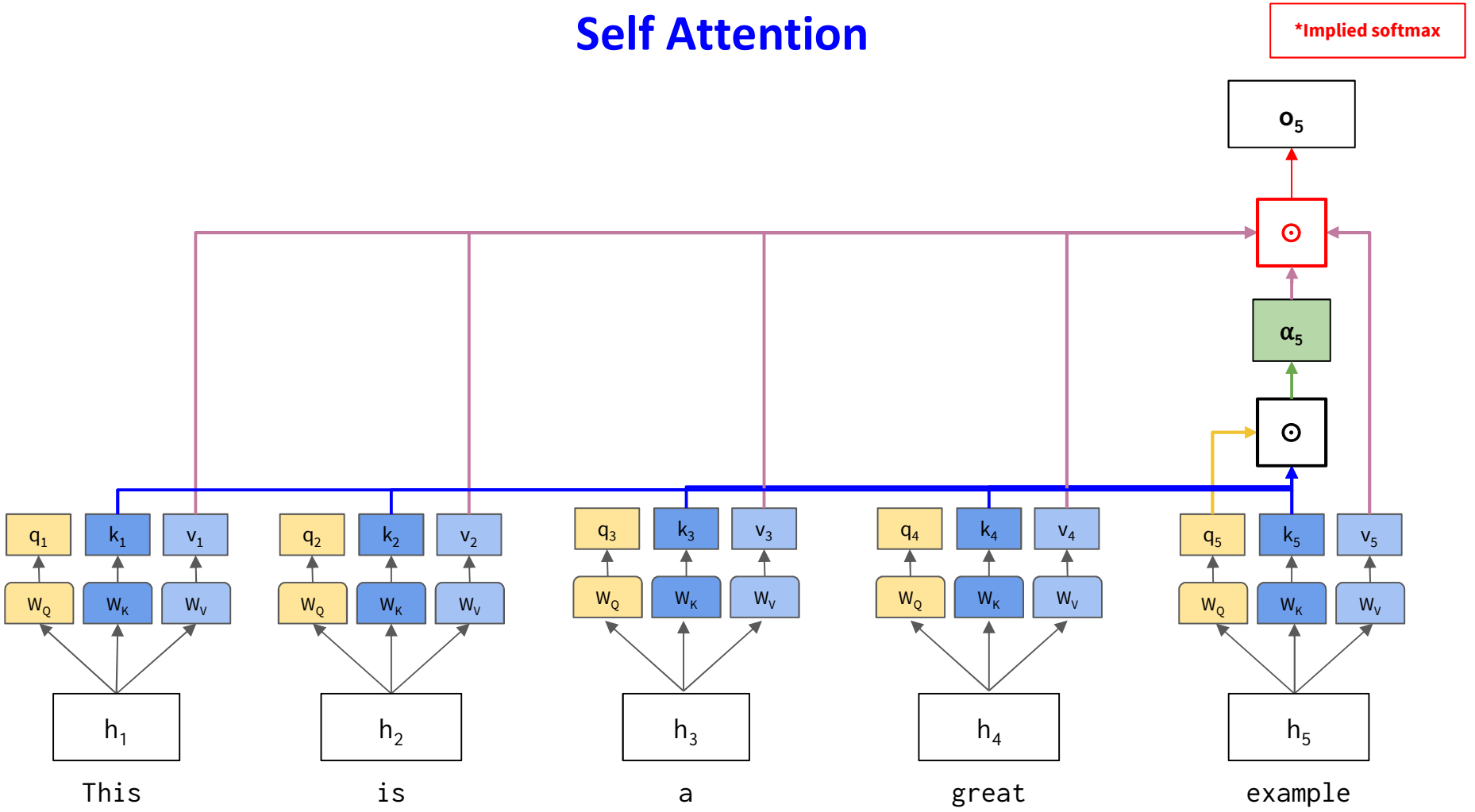


Self Attention

*Implied softmax

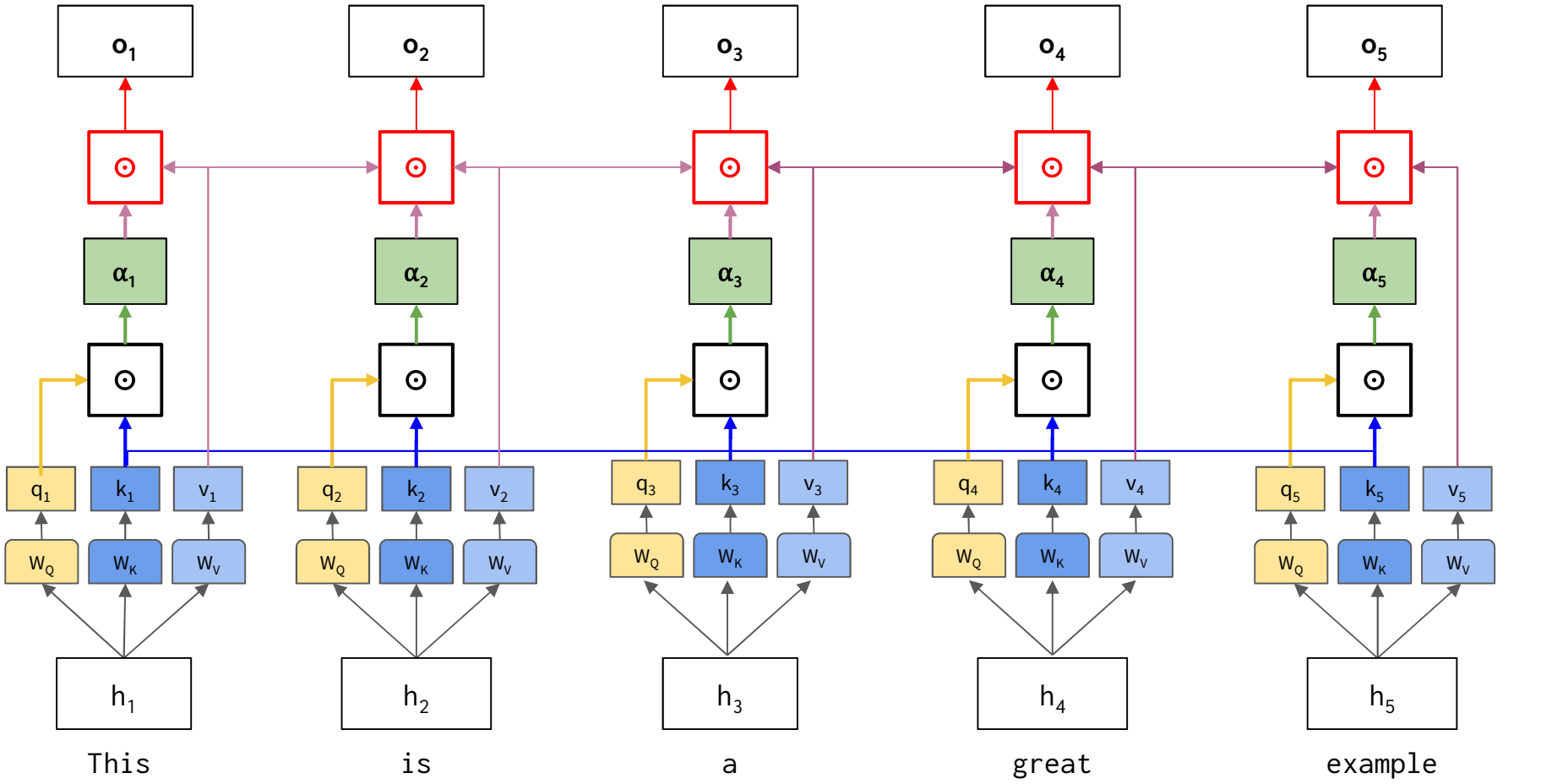


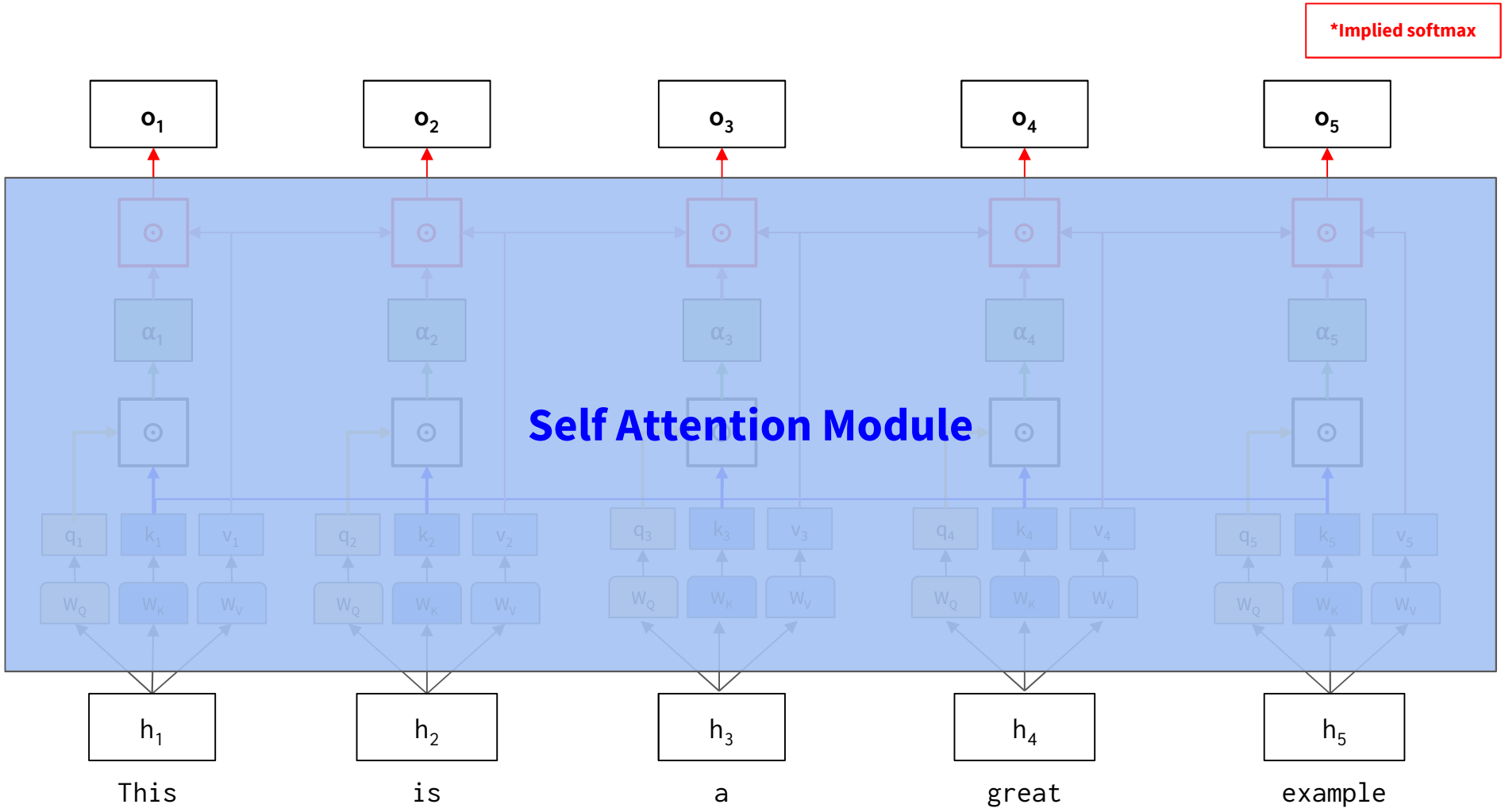
Self Attention

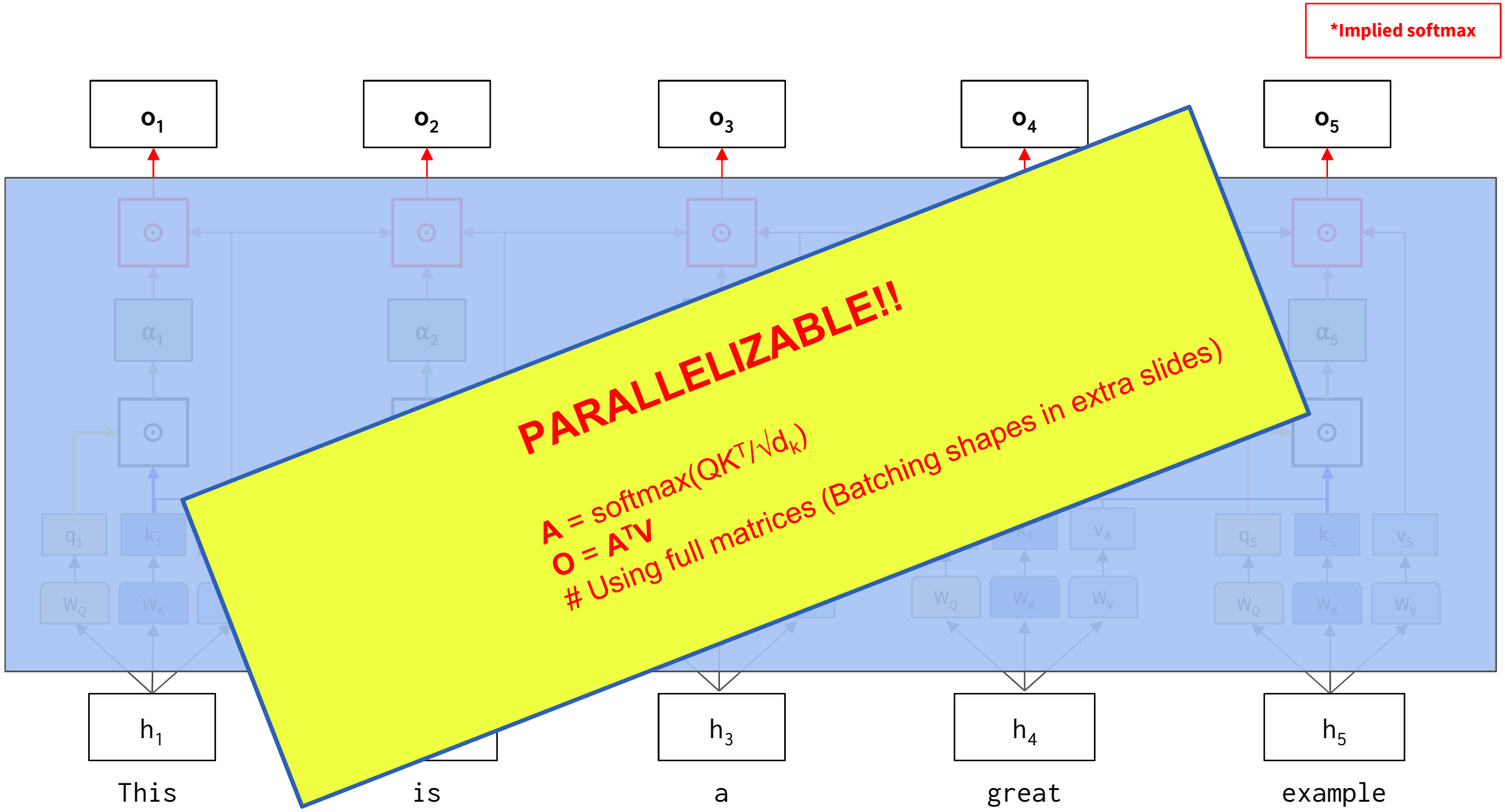


Self Attention

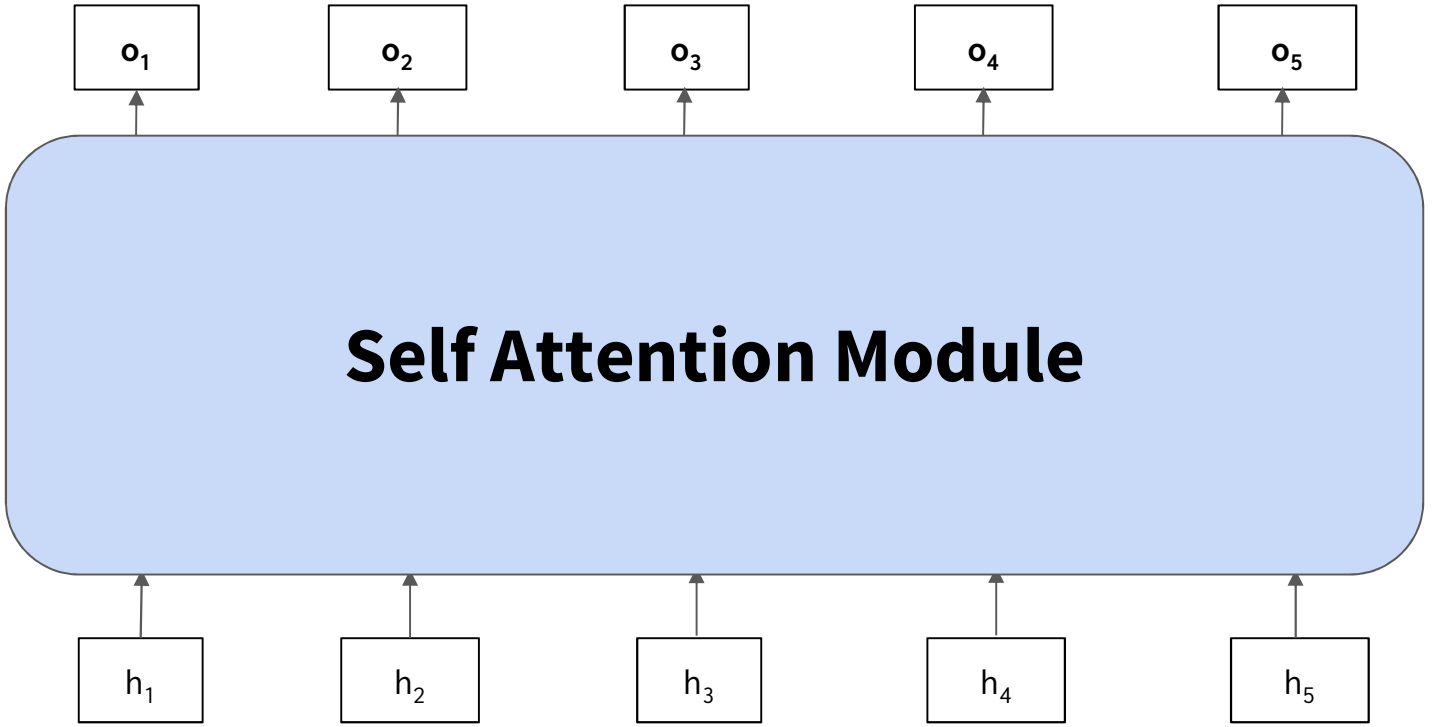
*Implied softmax




*Implied softmax
o1
o2
o3
o4
o5
Self Attention Module
h1
h2
h3
h4
h5
This
is
a
great
example


*Implied softmax
o_1
o_2
o_3
o_4
o_5
PARALLELIZABLE!!
$A = softmax(QK^T/\sqrt{d_k})$
$O = A^TV$
Using full matrices (Batching shapes in extra slides)
h_1
h_3
h_4
h_5
This
is
a
great
example

Single Headed Self Attention



Poll 1 (@1442)

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 (@1442)

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

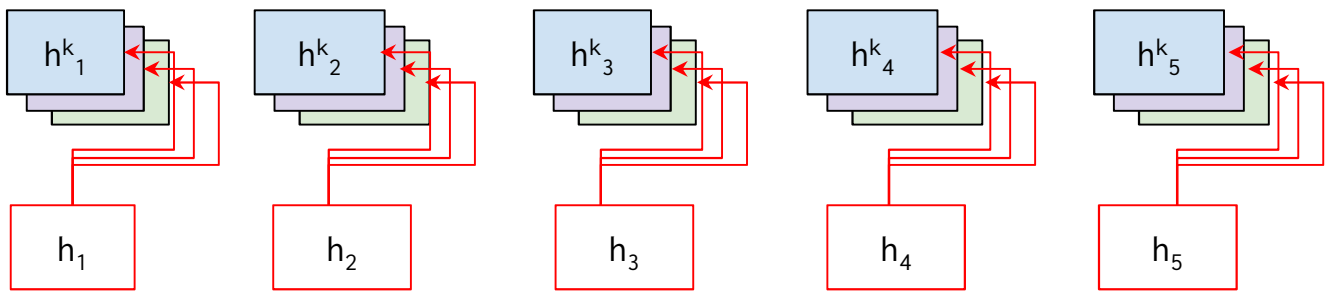
a. To calculate attention weights for input **h_i**, you would use key **k_i,** and all queries

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

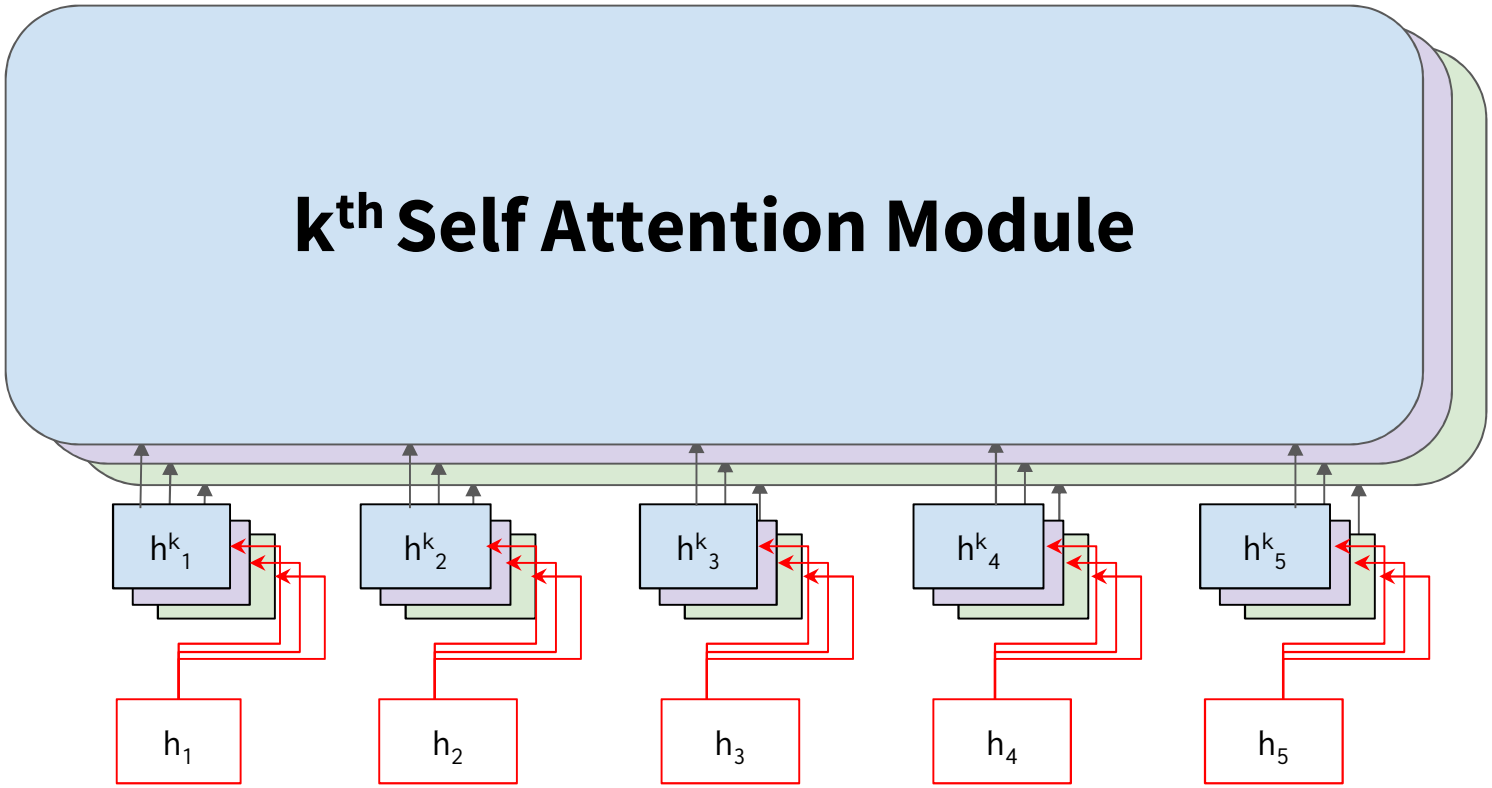
c. The energy function is scaled to bring attention weights in the range of [0,1]

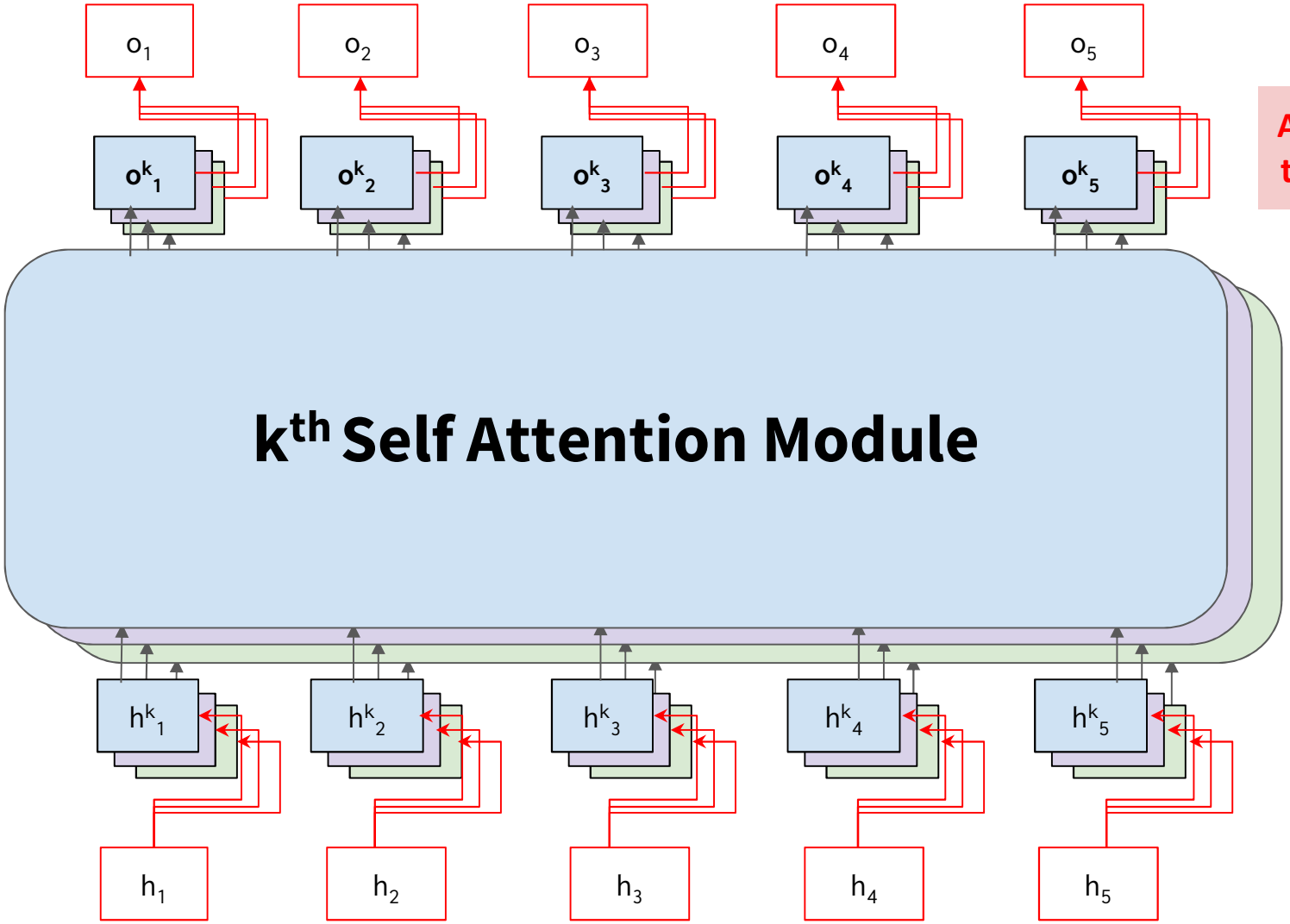
d. The energy function is scaled to allow for numerical stability

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

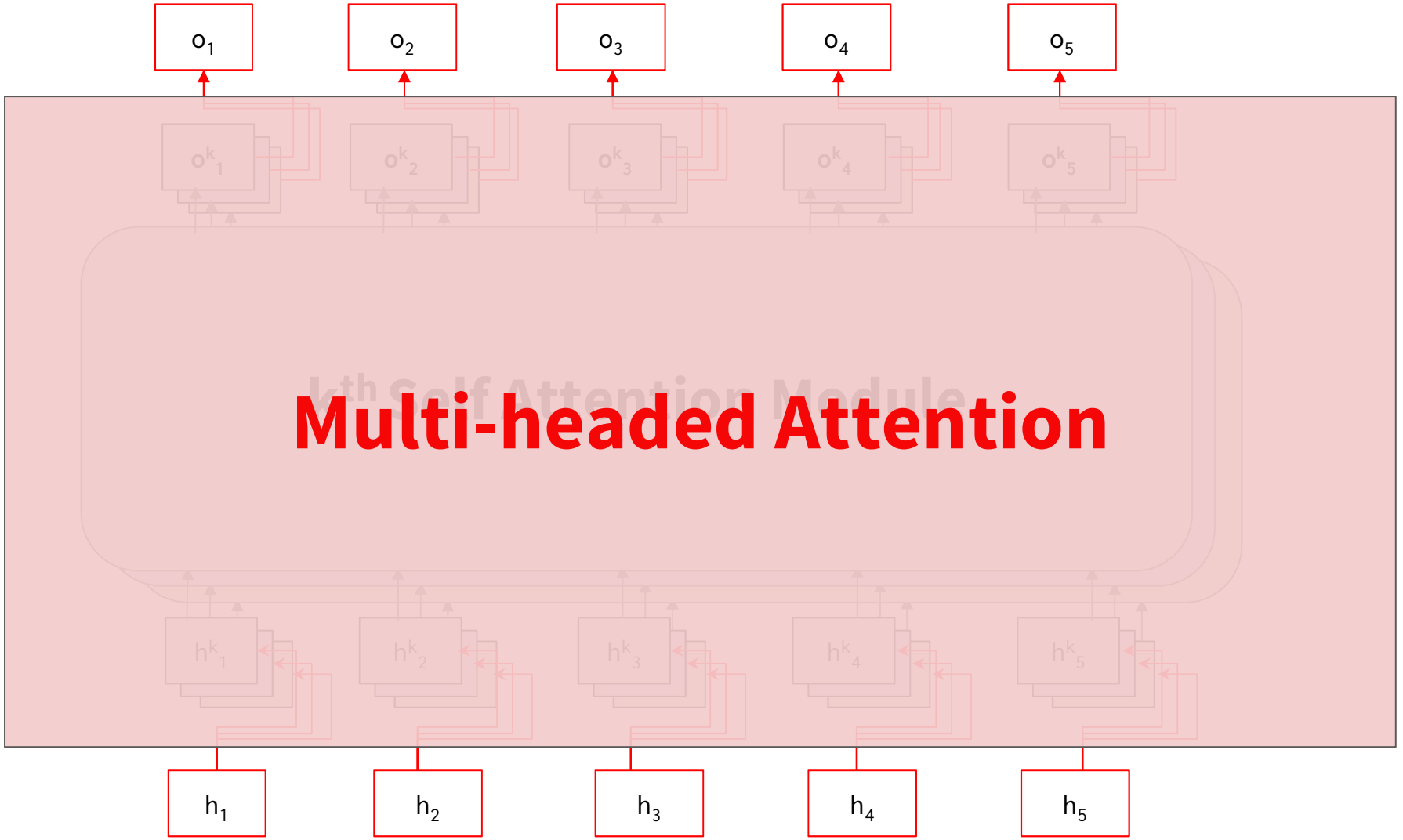


And pass each sub-input into a Self-Attention Module?





And combine the outputs?



Multi-headed Attention

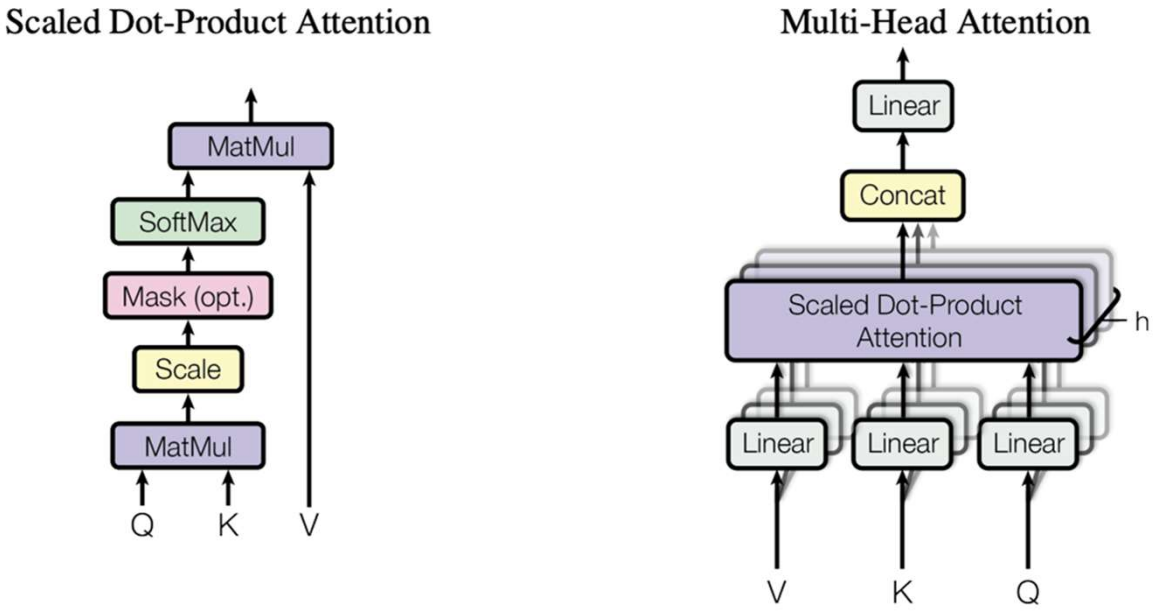
Multi Headed Self Attention

- 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



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

Breaking down the transformer

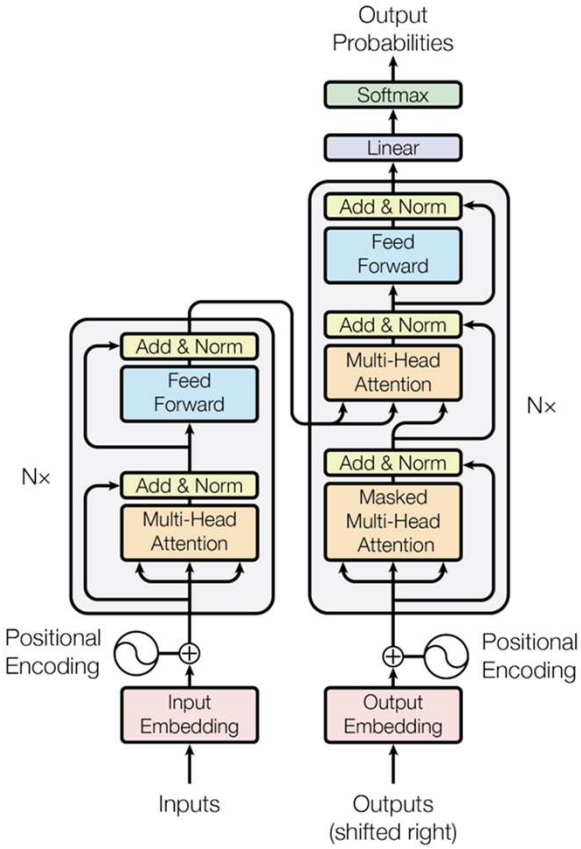


Figure 1: The Transformer - model architecture.

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

Breaking down the transformer

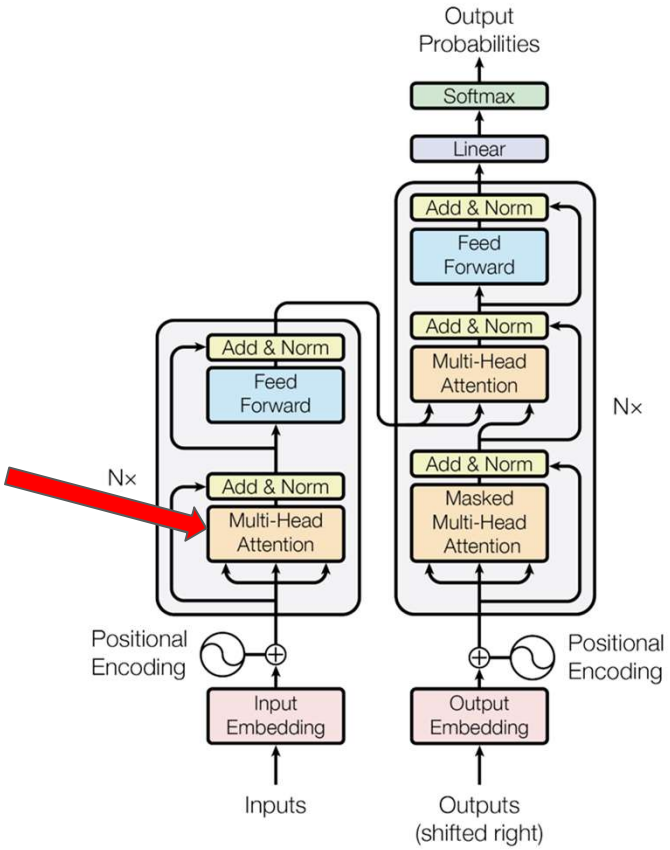


Figure 1: The Transformer - model architecture.

Encoder Self Attention

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

Breaking down the transformer

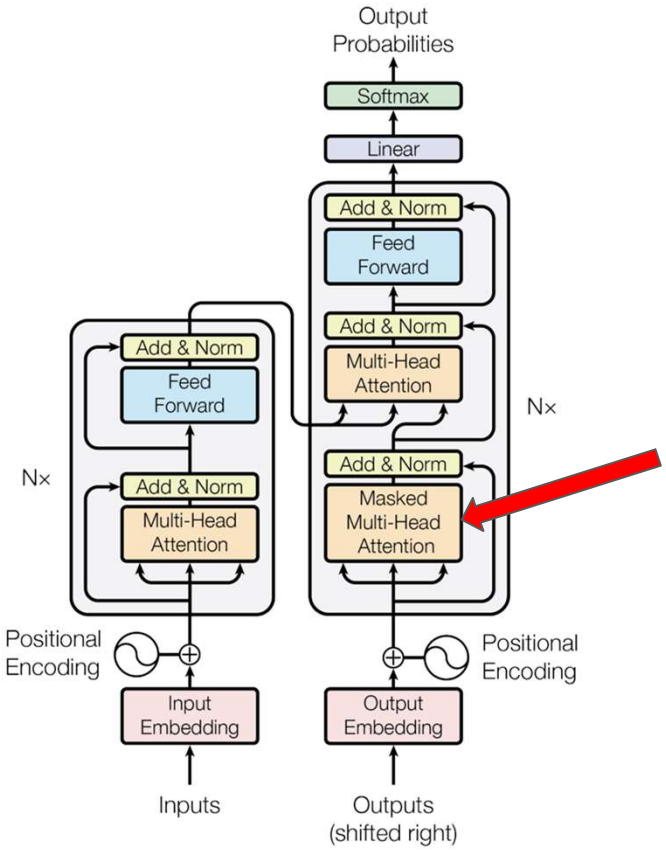


Figure 1: The Transformer - model architecture.

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

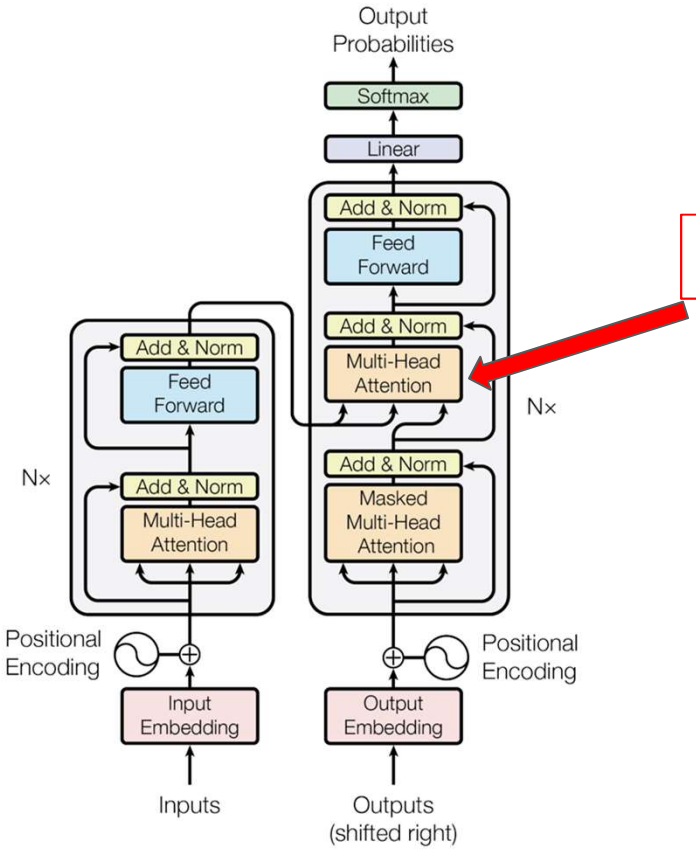


Figure 1: The Transformer - model architecture.

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

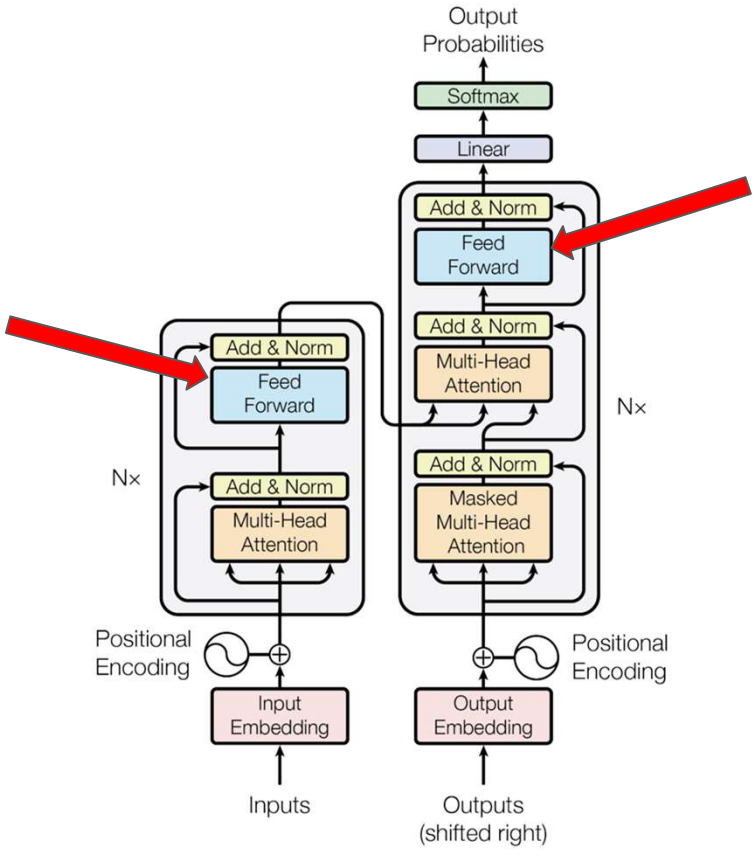
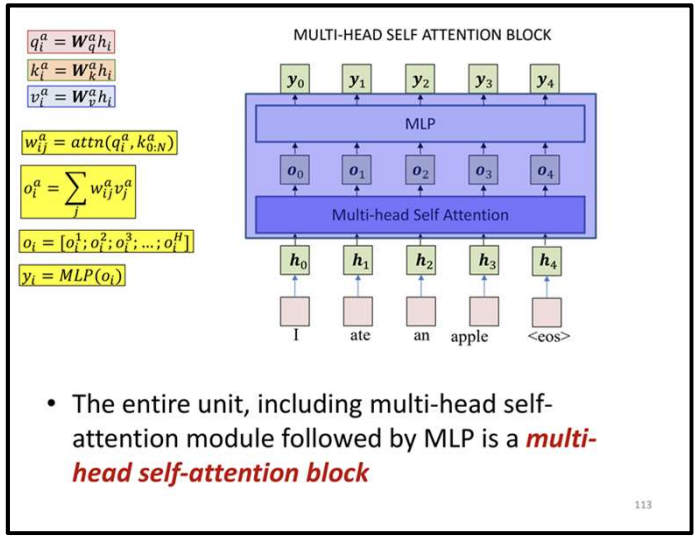


Figure 1: The Transformer - model architecture.

Feed Forward Layers

Feed Forward layers allow for high dimensional computations

Simply there to allow the model to capture more information



Breaking down the transformer

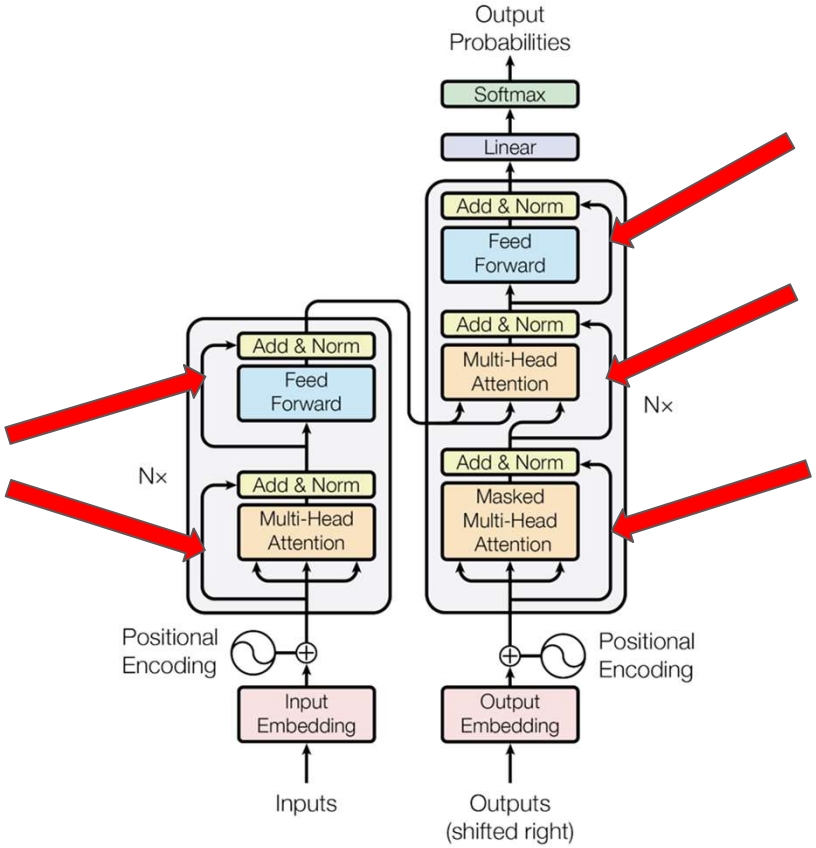


Figure 1: The Transformer - model architecture.

Residual Connections

`Add & Norm`:

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

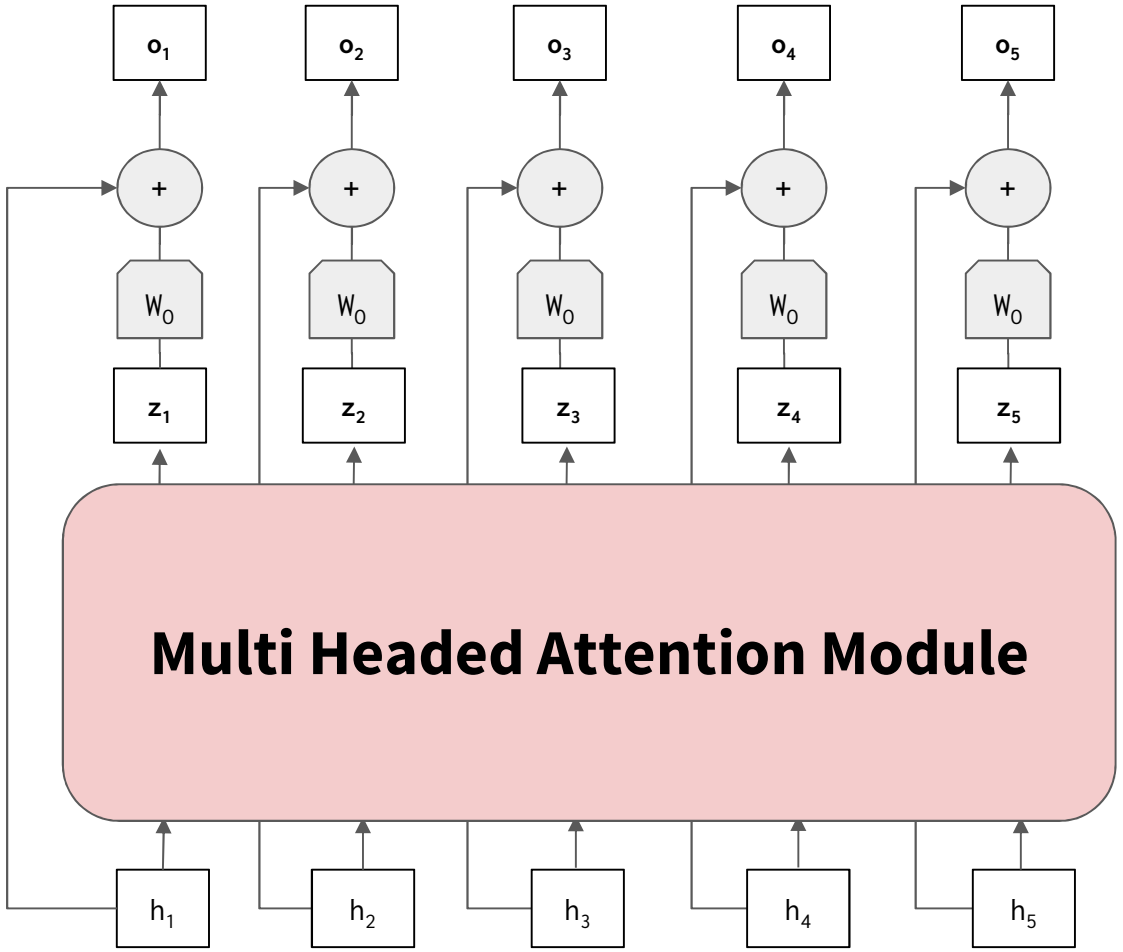
`Sublayer(x)` is whatever layer is below the **Add & Norm**

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

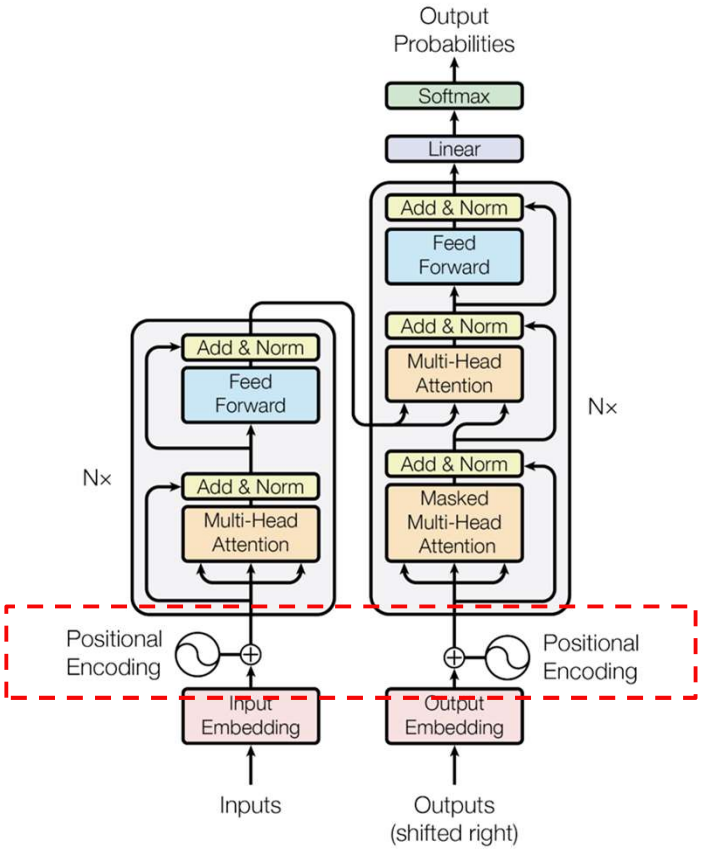


Figure 1: The Transformer - model architecture.

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

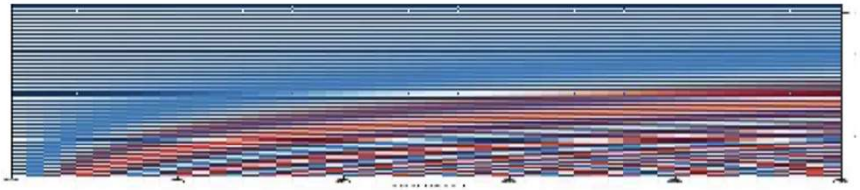
Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

Recall

Positional encodings as discussed in the last lecture.

Positional Encoding

regenerate



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

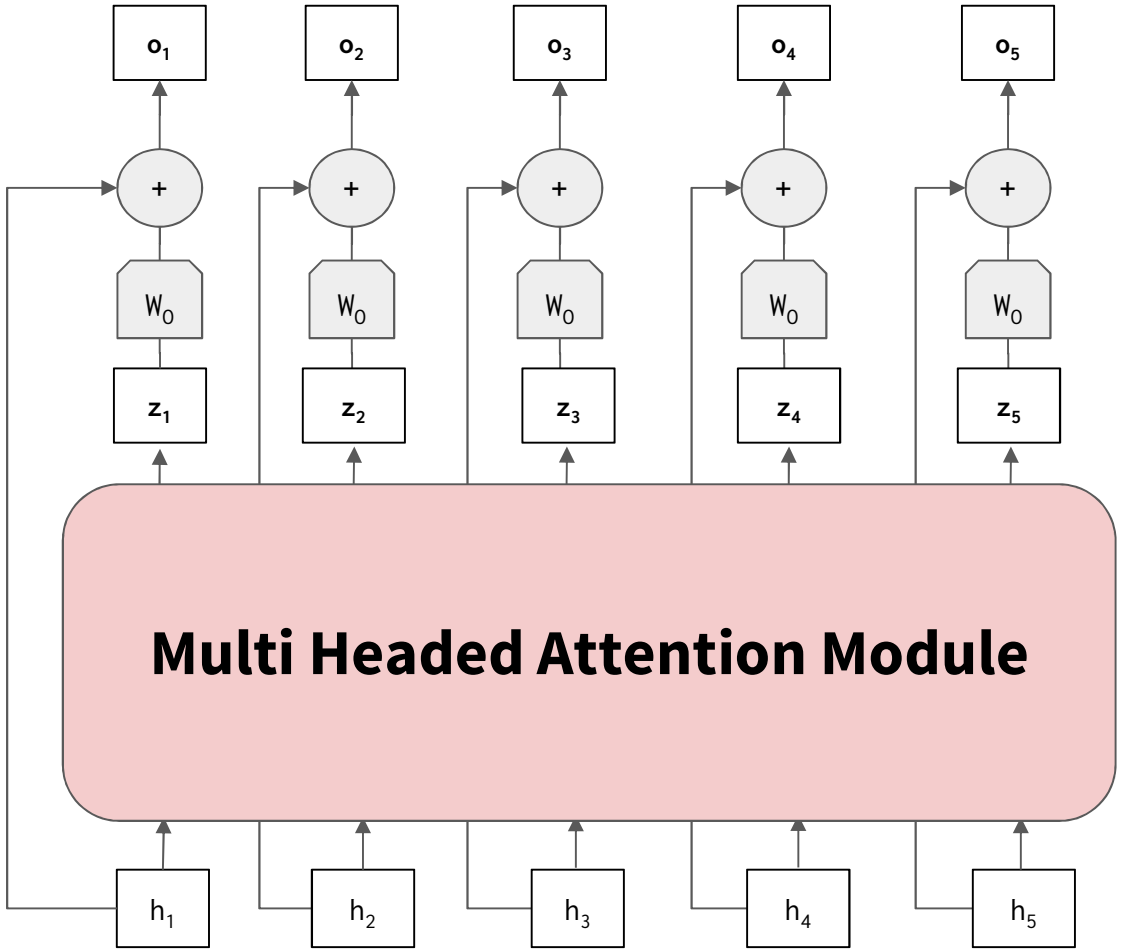
$$P_{t+\tau} = M_\tau P_t$$

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

- 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

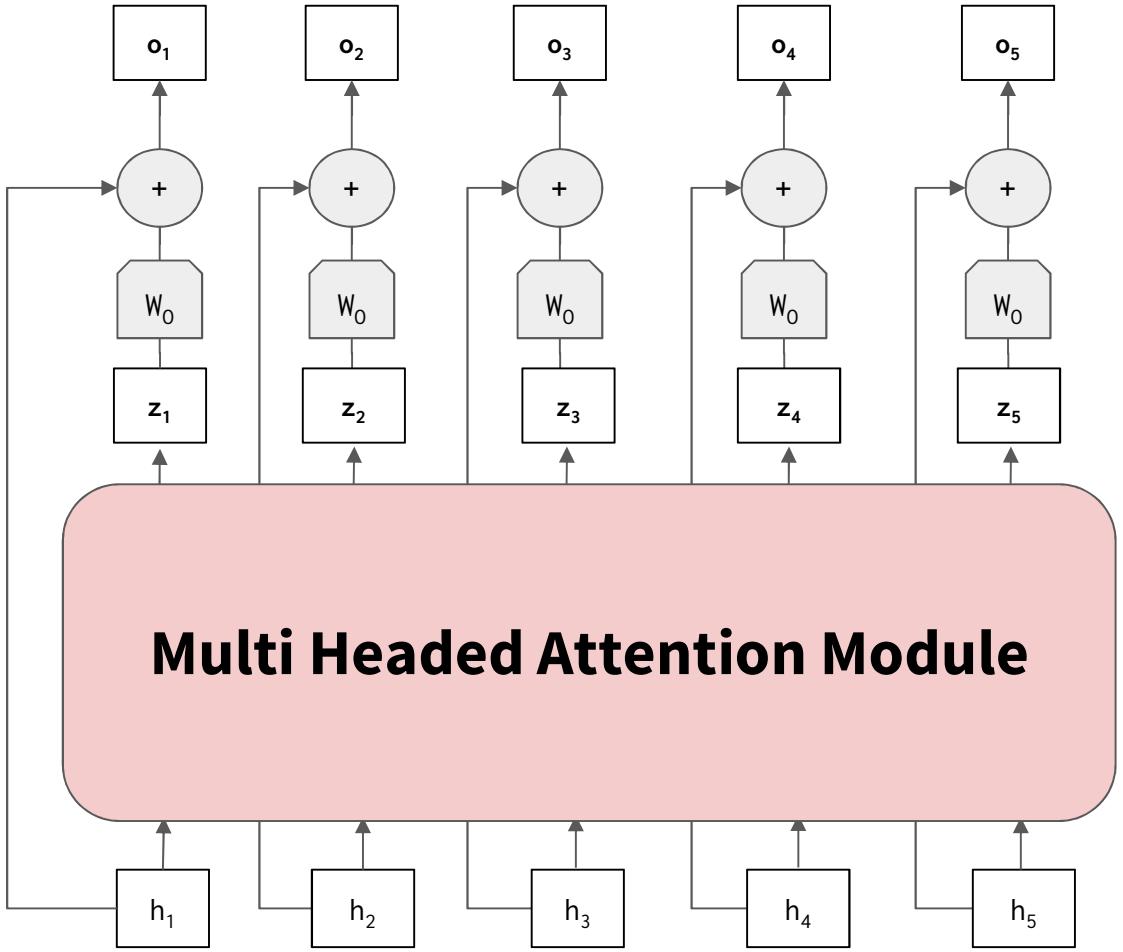
121

Positional Encoding



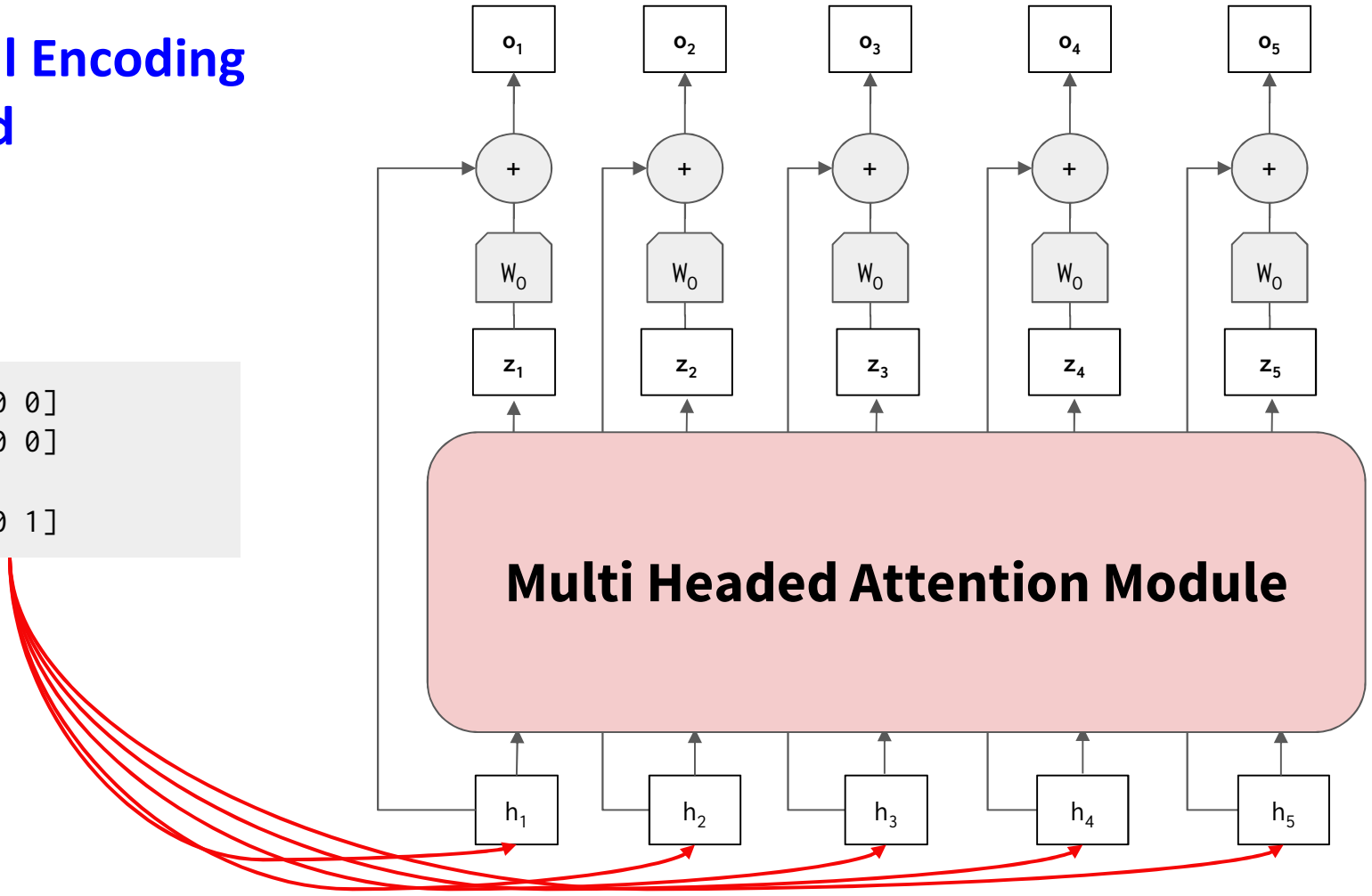
Positional Encoding simplified

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



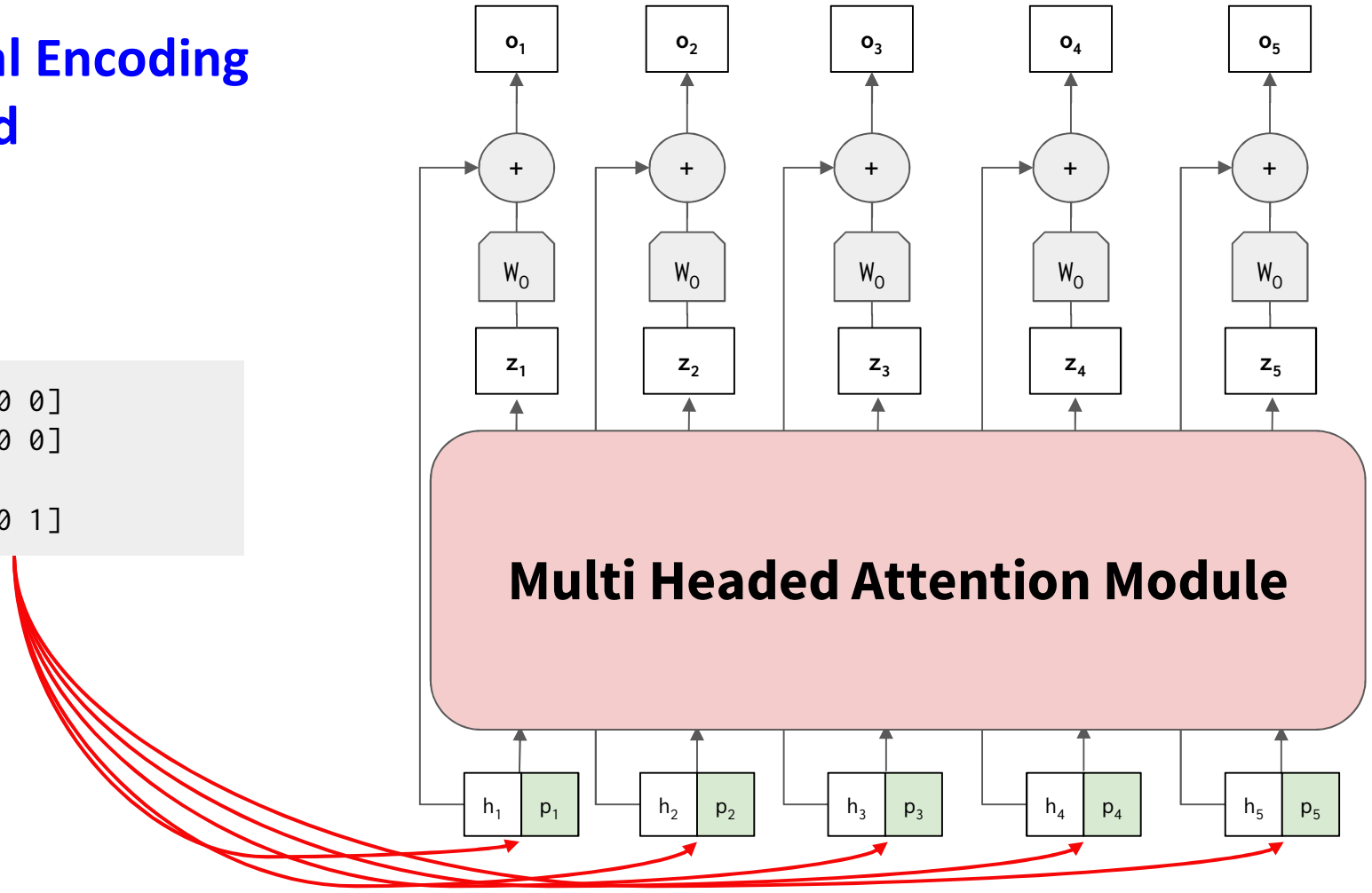
Positional Encoding simplified

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 (@1436)

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 (@1436)

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'
- On smaller sequence lengths, RNN-based models might perform better with fewer parameters.

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 **$\text{softmax}(e)^T v$**

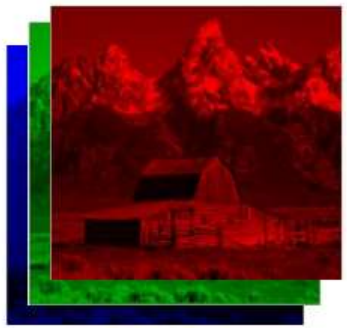
Part 2

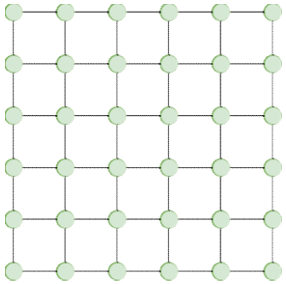
Permutation Invariance & Graph Neural Networks

Revisiting data structure we have met

Sequence data: text/speech

Grid data: image

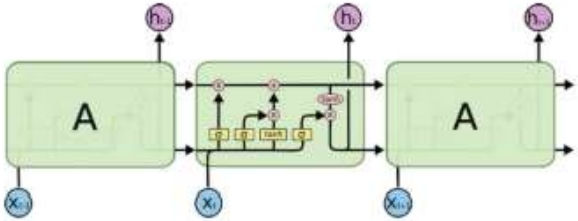




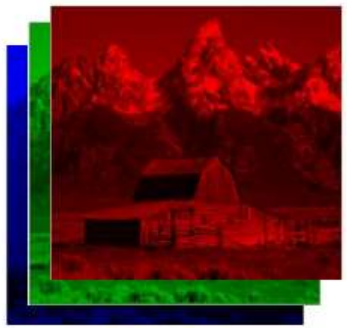
Revisiting data structure we have met

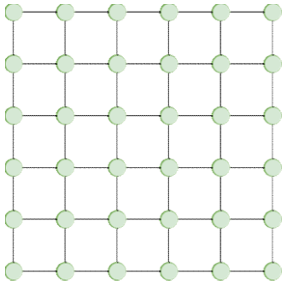
Sequence data: text/speech

Recurrent Neural Networks

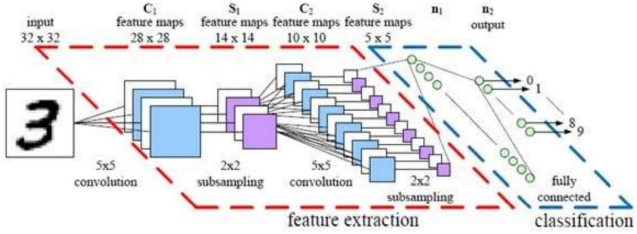


Grid data: image



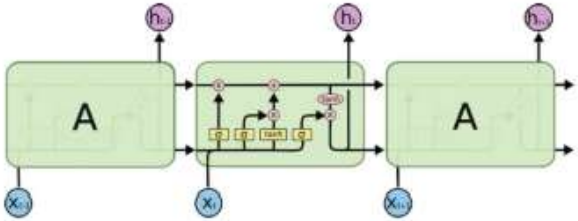


Convolution Neural Networks

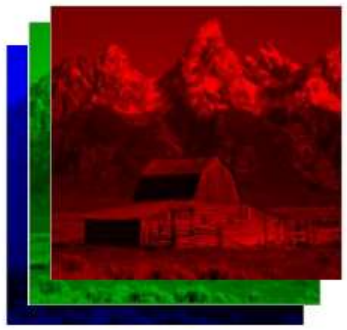


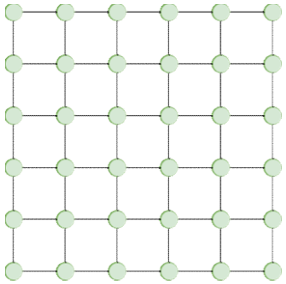
Revisiting data structure we have met

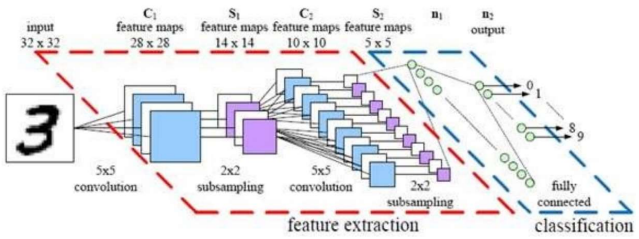
Sequence data: text/speech



Grid data: image

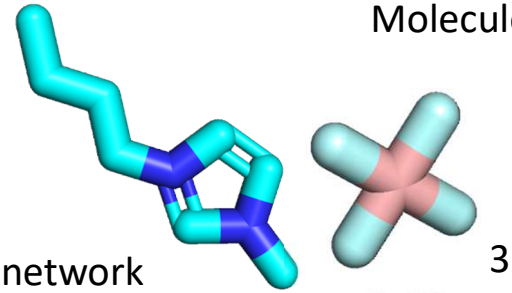






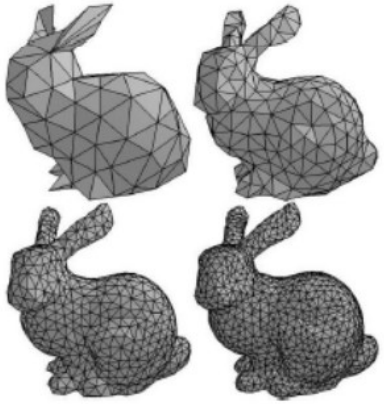
Unstructured data: Molecule, Social Networks ,3D Mesh

Molecule



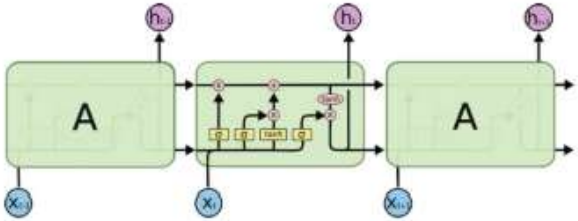
Social network

3D Mesh

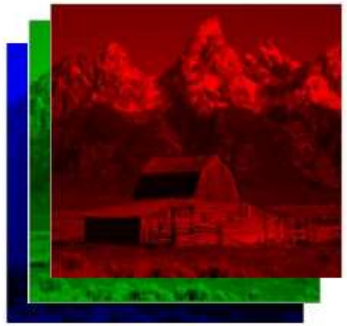


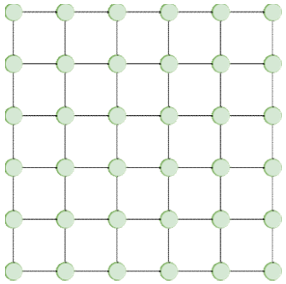
Revisiting data structure we have met

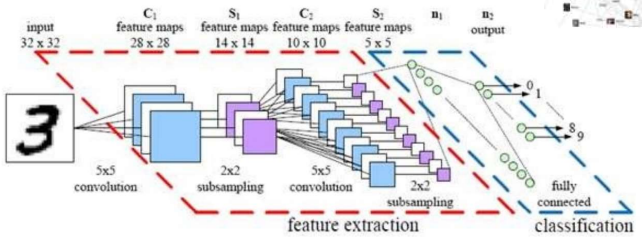
Sequence data: text/speech



Grid data: image

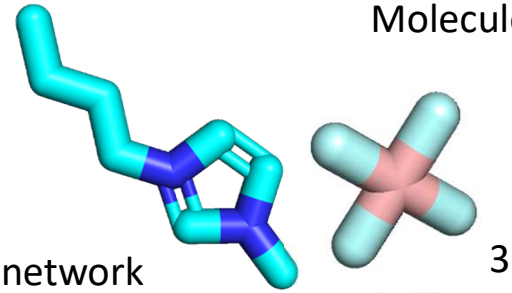






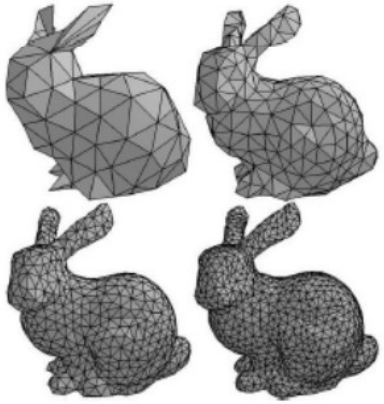
Unstructured data: Molecule, Social Networks ,3D Mesh

Molecule



Social network

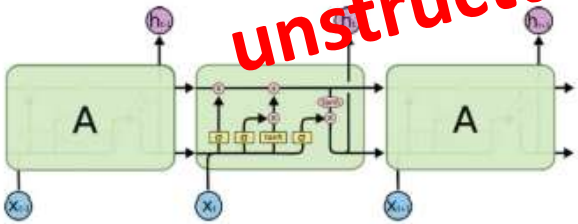
3D Mesh



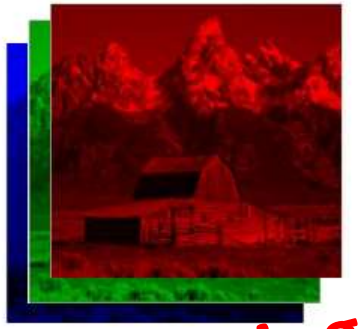
?

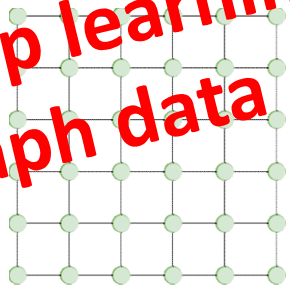
Revisiting data structure we have met

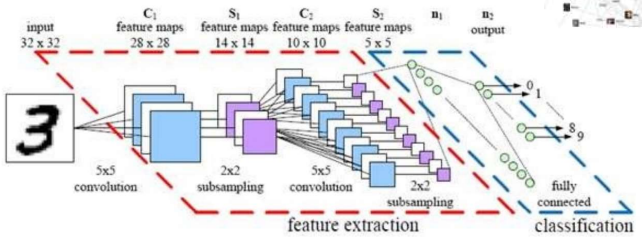
Sequence data: text/speech



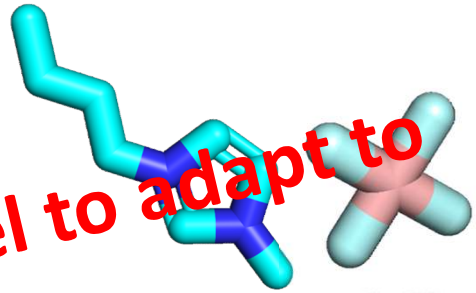
Grid data: image

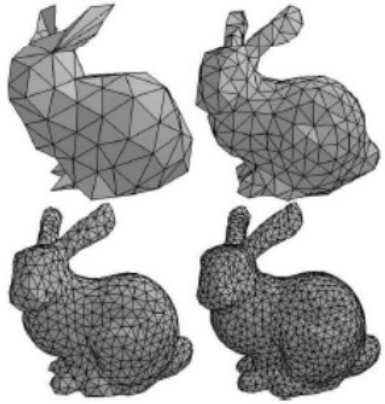






Unstructured data: Molecule, Social Networks ,3D Mesh



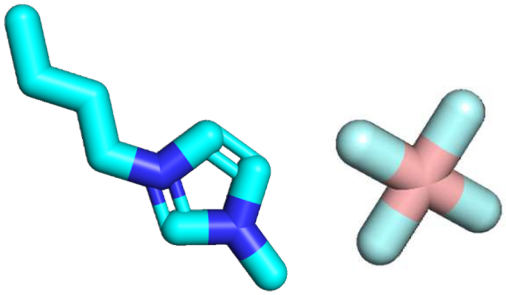


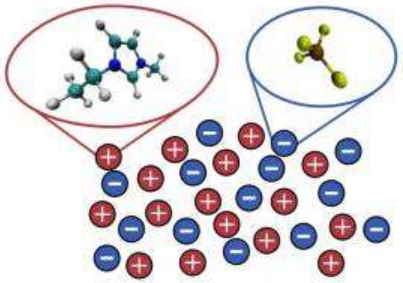
We want the deep learning model to adapt to unstructured graph data

?

Problem Setup: ionic liquid for CO2 capturing

Ionic liquid molecules

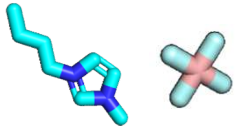




Carbon dioxide (CO2)

dataset

data	label
IL molecule structure	Solubility(volume percentage)



0.56,
0.119……

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

Can we use MLP, CNN or RNN to do this job?

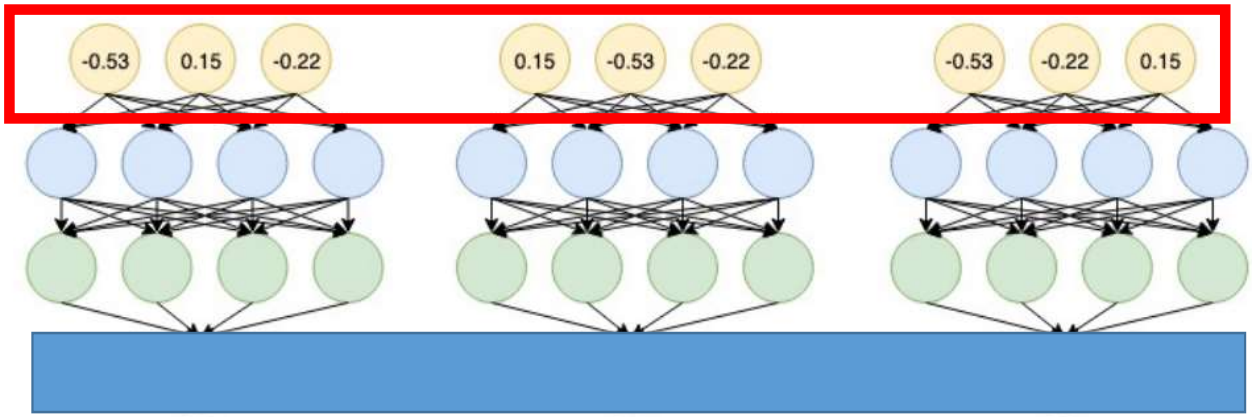
Invariance

Let's define a mapping from X to Y : Y = f(X)

- If there exist a mapping function of g(), such that f(g(X)) = f(X) = Y
- we say f() is invariant to g()

Example 1 (Multi-layer Perceptron + Sequence Data):

permutation



f(): MLP

g():permutation of input order

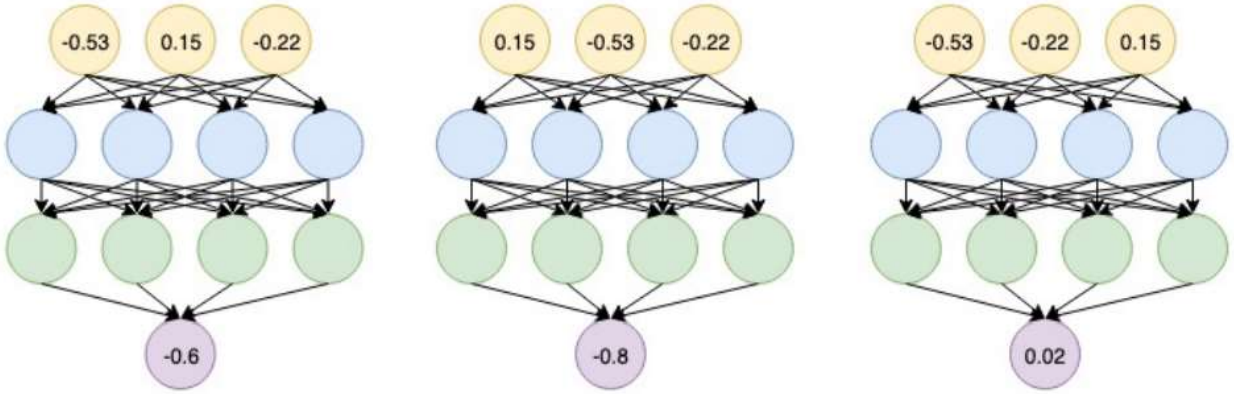
Is f() invariant to g()?

Invariance

Let's define a mapping from X to Y : Y = f(X)

- **If there exist a mapping function of g(), such that f(g(X)) = f(X) = Y**
- **we say f() is invariant to g()**

Example 1 (Multi-layer Perceptron + Sequence Data):



f(): MLP
g():permutation of input order

Is f() invariant to g()?

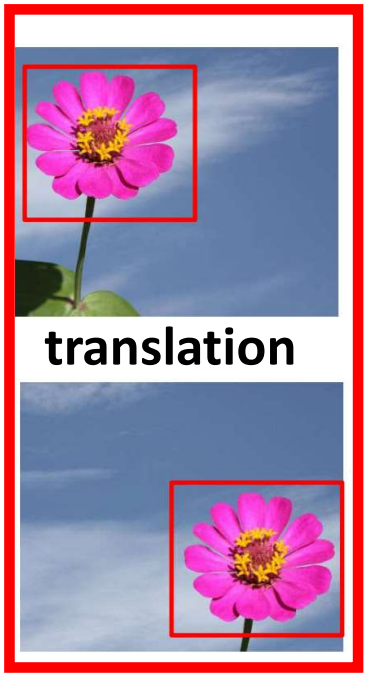
No!

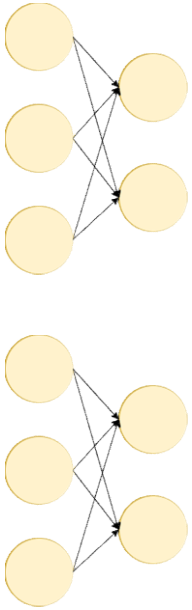
Invariance

Let's define a mapping from X to Y : Y = f(X)

- **If there exist a mapping function of g(), such that f(g(X)) = f(X) = Y**
- **we say f() is invariant to g()**

Example 2 (MLP + image data):





result

|| ?

result

f(): MLP
g():translation

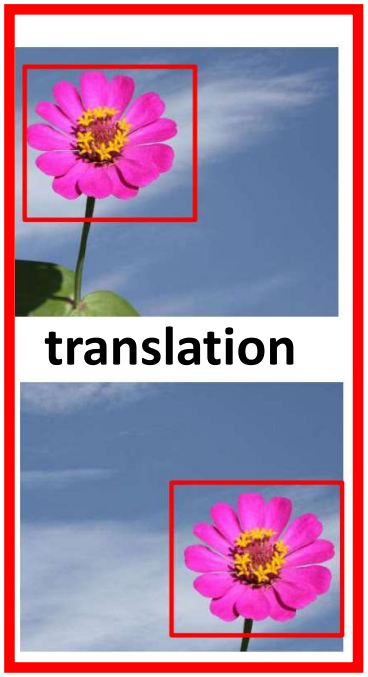
Is f() invariant to g()?

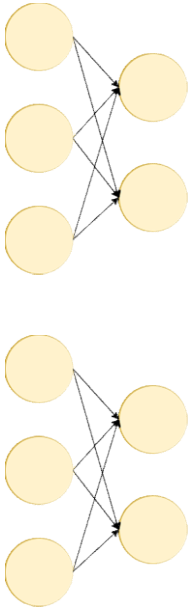
Invariance

Let's define a mapping from X to Y : Y = f(X)

- **If there exist a mapping function of g(), such that f(g(X)) = f(X) = Y**
- **we say f() is invariant to g()**

Example 2 (MLP + image data):





result

?

result

f(): MLP
g():translation

Is f() invariant to g()?

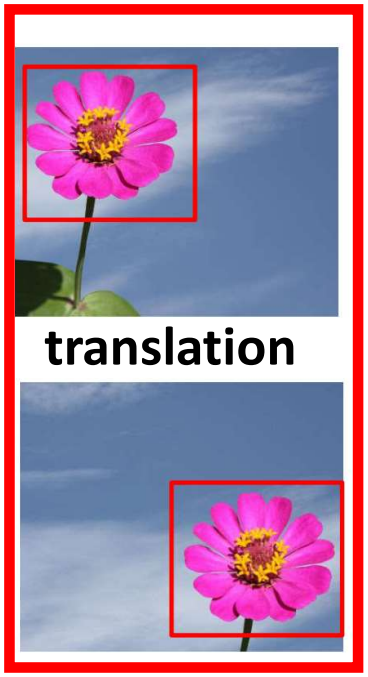
No!

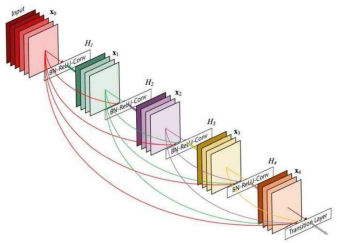
Invariance

Let's define a mapping from X to Y : Y = f(X)

- **If there exist a mapping function of g(), such that f(g(X)) = f(X) = Y**
- **we say f() is invariant to g()**

Example 3 (CNN + image data):

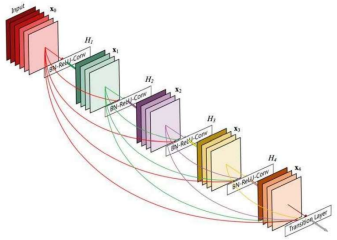




Output_1

f(): CNN
g():translation

|| ? **Is f() invariant to g()?**



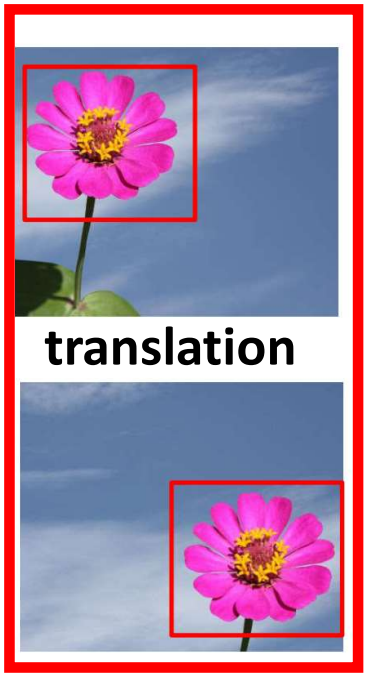
Output_2

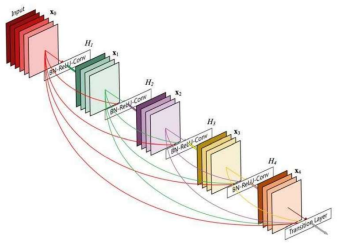
Invariance

Let's define a mapping from X to Y : Y = f(X)

- **If there exist a mapping function of g(), such that f(g(X)) = f(X) = Y**
- **we say f() is invariant to g()**

Example 3 (CNN + image data):



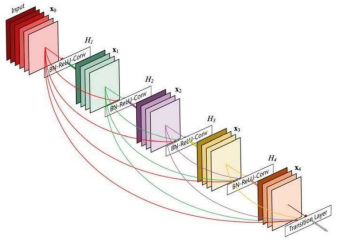


Output_1

f(): CNN
g():translation

|| ?

Is f() invariant to g()?



Output_2

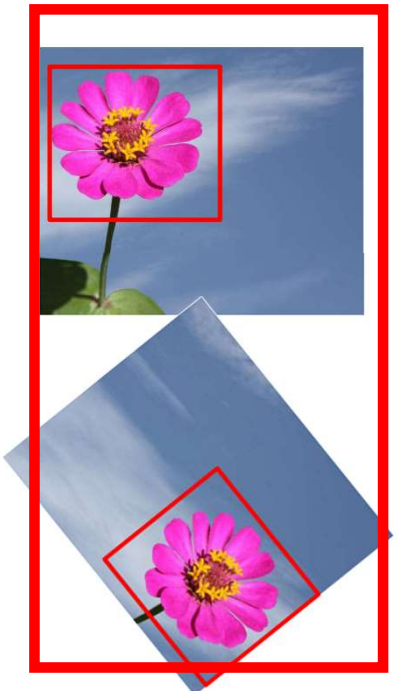
Yes !

Invariance

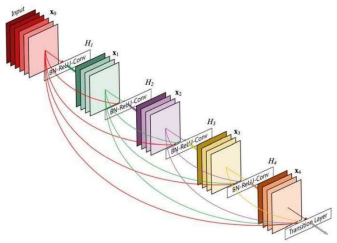
Let's define a mapping from X to Y : Y = f(X)

- **If there exist a mapping function of g(), such that f(g(X)) = f(X) = Y**
- **we say f() is invariant to g()**

Example 4 (CNN + image data):



rotation

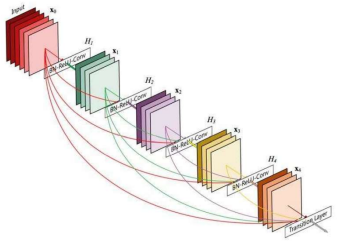


Output_1

f(): CNN
g():rotation

|| ?

Is f() invariant to g()?



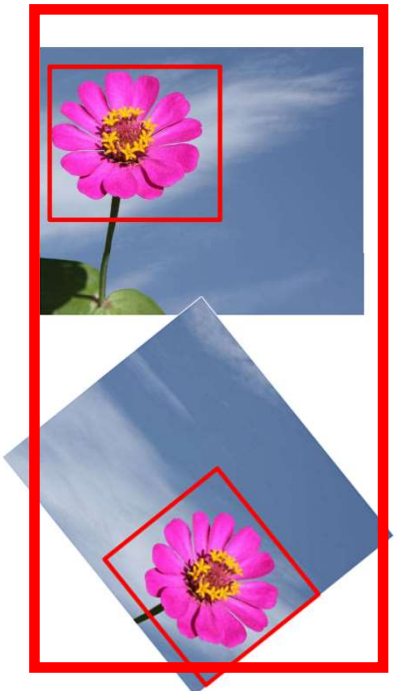
Output_2

Invariance

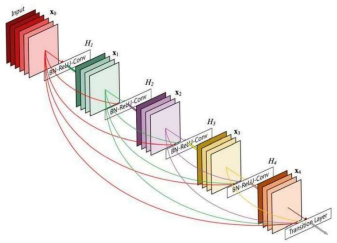
Let's define a mapping from X to Y : Y = f(X)

- **If there exist a mapping function of g(), such that f(g(X)) = f(X) = Y**
- **we say f() is invariant to g()**

Example 4 (CNN + image data):



rotation

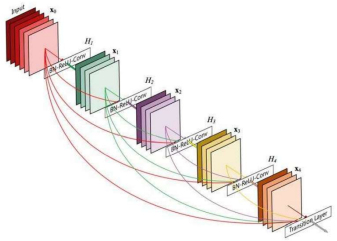


Output_1

f(): CNN
g():rotation

|| ?

Is f() invariant to g()?

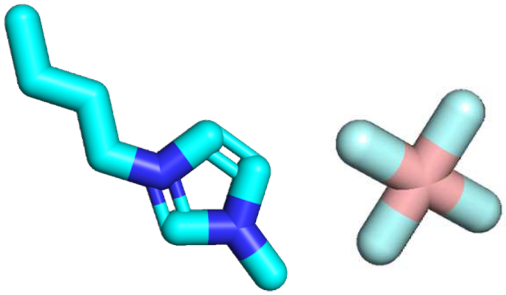


Output_2

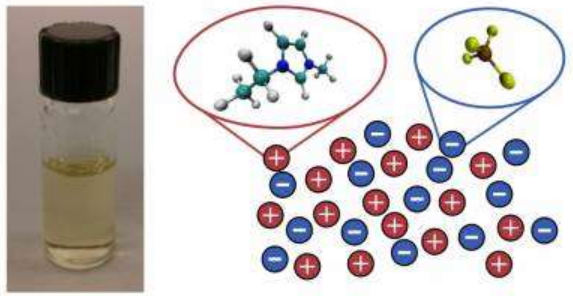
No !

Problem Setup: ionic liquid for CO2 capturing

Ionic liquid molecules

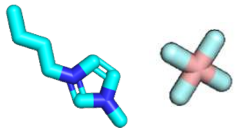


Carbon dioxide (CO2)



dataset

data	label
IL molecule structure	Solubility(volume percentage)



0.56,
0.119……

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

(1) How many ways can you come up with?

(2) Can we use MLP, CNN or RNN to do this job?

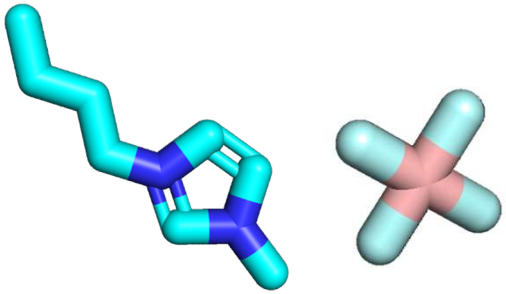
(3) What are the desired properties of the model we used for molecule?

- **permutation invariant?**
- **translation invariant?**
- **rotation invariant?**

Possible solution for molecule properties prediction

Use MLP to solve the problem

Ionic liquid molecules



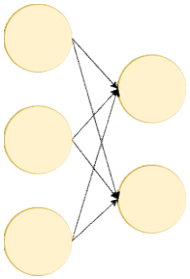
Sequence data: Empirical Formula

$C_8H_{15}F_6N_2P+BF_4$

C8H15F6N2P+BF4

C	8
H	15
N	2
P	1
F	6 4
B	1

Feed sequence to MLP

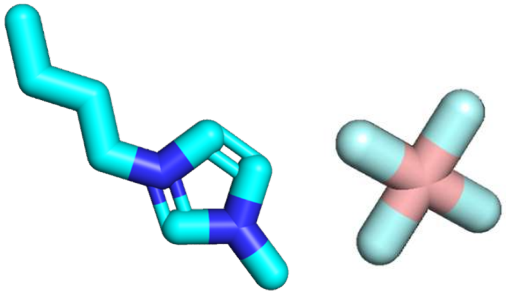


Result

Possible solution for molecule properties prediction

Use MLP to solve the problem

Ionic liquid molecules



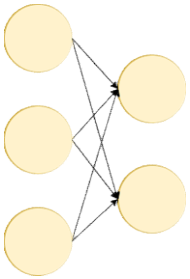
Sequence data: Empirical Formula

$C_8H_{15}F_6N_2P+BF_4$

C8H15F6N2P+BF4

C	8
H	15
N	2
P	1
F	6 4
B	1

Feed sequence to MLP

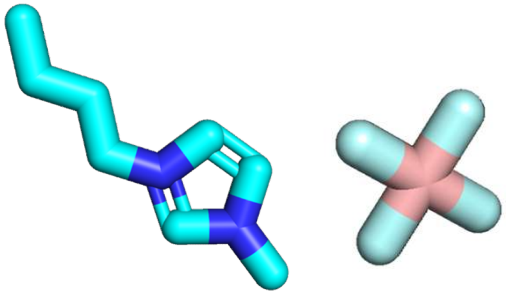


Result

Possible solution for molecule properties prediction

Use MLP to solve the problem

Ionic liquid molecules



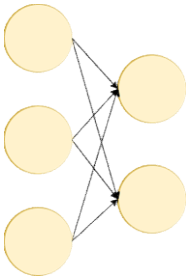
Sequence data: Empirical Formula

$C_8H_{15}N_2F_6P+BF_4$

C8H15N2F6P+BF4

C	8
H	15
N	2
P	1
F	6 4
B	1

Feed sequence to MLP



Result?

MLP is not permutation invariant!

Not good for molecule with unstructured data

Possible solution for molecule properties prediction

Another possible solution

Ionic liquid molecules

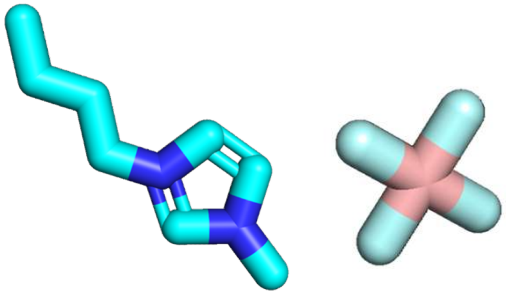
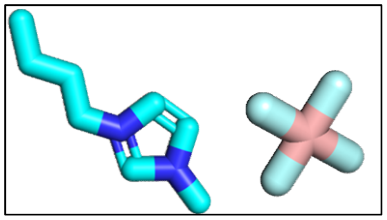
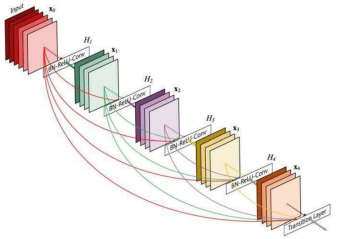


Image data: image for molecule structure



Feed image to CNN



Result?

Possible solution for molecule properties prediction

Another possible solution

Ionic liquid molecules

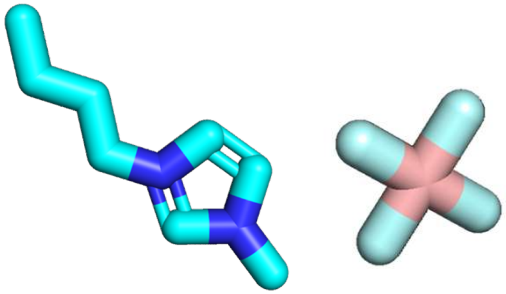
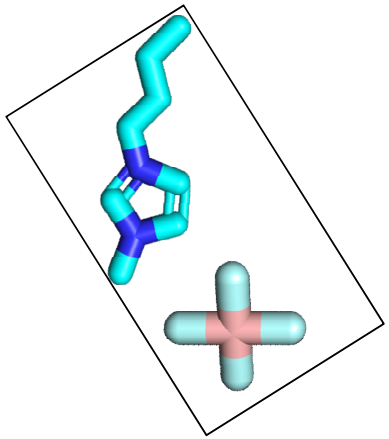
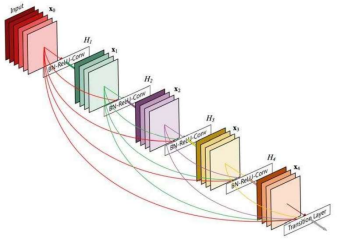


Image data: image for molecule structure



Rotate image

Feed image to CNN



Result?

Possible solution for molecule properties prediction

Another possible solution

Ionic liquid molecules

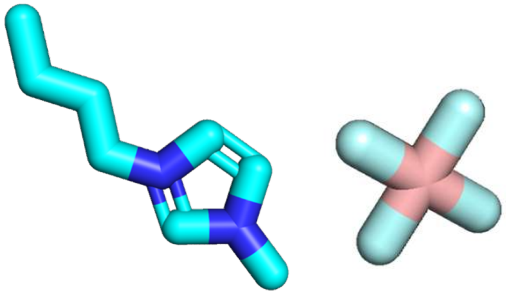
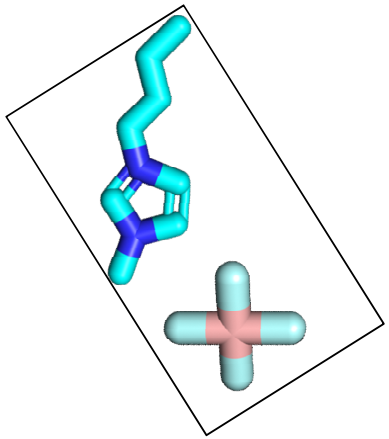
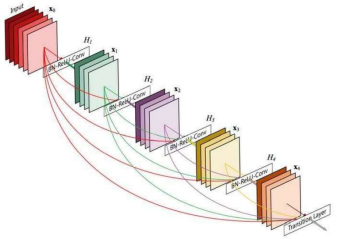


Image data: image for molecule structure



Rotate image

Feed image to CNN



Result?

CNN is not rotational invariant!

Not good for molecule with unstructured data

Story so far

- We want to use deep learning to do prediction for unstructured graph like data such as molecule
  - MLP is not permutation nor translation invariant
  - CNN is not permutation invariant nor rotation invariant

- Those draw backs make MLP and CNN not a good candidates for processing molecule data
- We want a model with a strong invariant property

Graph: Definition

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

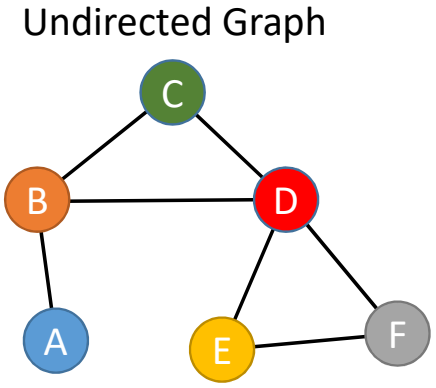
- **where V is a set whose elements are called nodes(vertices),**
- **and E is a set of paired vertices, whose elements are called edges.**

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)}

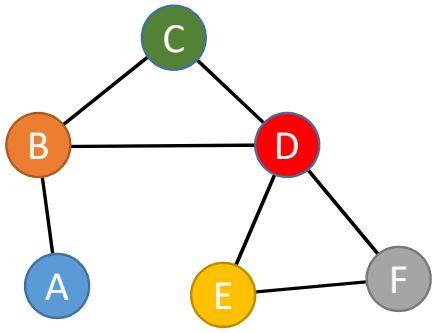


Graph: Matrix representation for graph

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

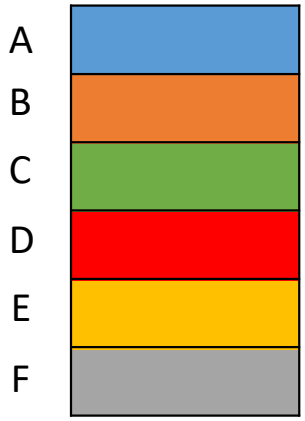
- **where V is a set whose elements are called nodes(vertices),**
- **and E is a set of paired vertices, whose elements are called edges.**

Undirected Graph



Node information Matrix (N × F)

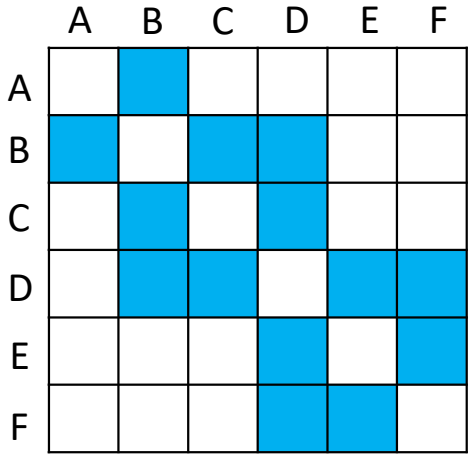
6



Node information

Adjacency Matrix (N × N)

6 6

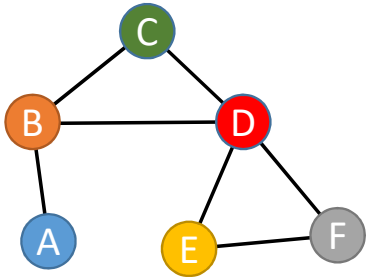


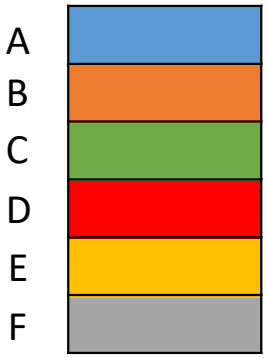
Connectivity information

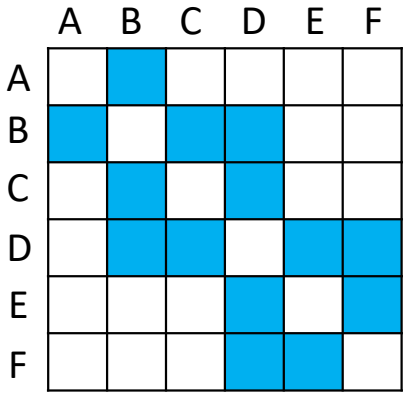
Graph: No canonical order of nodes

Graph do not have canonical order of the nodes!

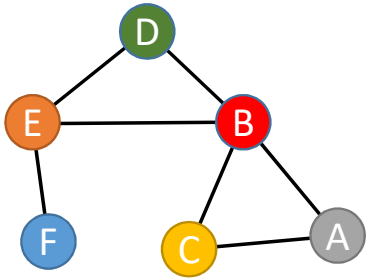
Order plan 1

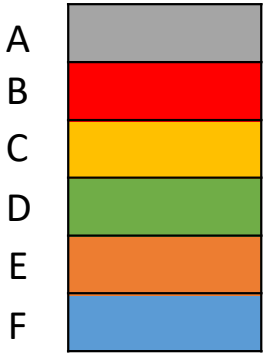


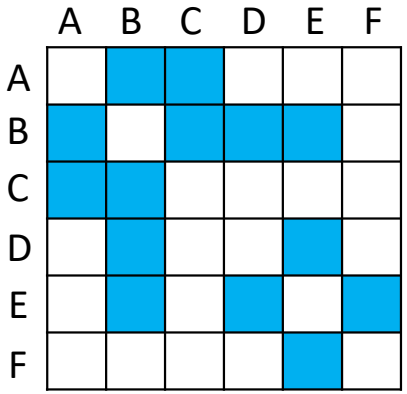




Order plan 2



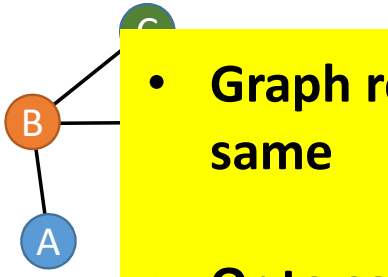


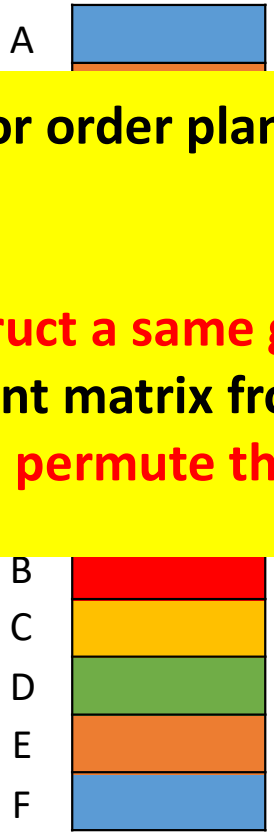


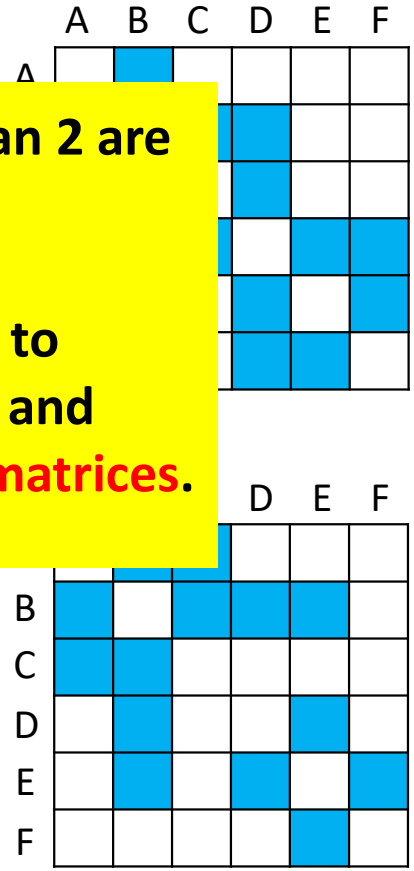
Graph: No canonical order of nodes

Graph do not have canonical order of the nodes!

Order plan 1

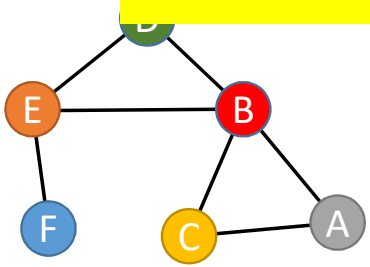






- **Graph representation for order plan 1 and order plan 2 are same**
- **Or to say, we can construct a same graph according to node matrix and adjacent matrix from order plan 1 and order plan 2, even if we permute the order in two matrices.**

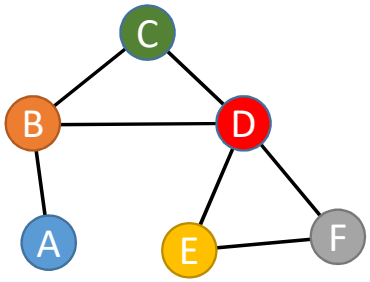
Order



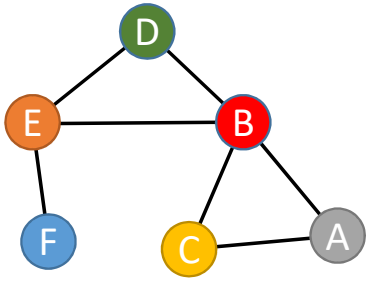
Graph: Permutation invariance

Desired properties for GNN

Order plan 1



Order plan 2

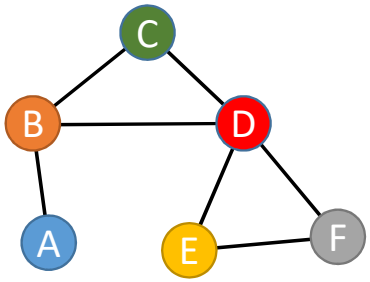


- **Consider we use deep learning model to learn a function $f()$ that map a graph G = (V,E) to a vector R^d**
- **We can write f: $f(V, E) \rightarrow R^d$ (V is node feature matrix, E is represent by adjacency matrix)**
- **What we want? $f(V_1, E_1) = f(V_2, E_2)$**

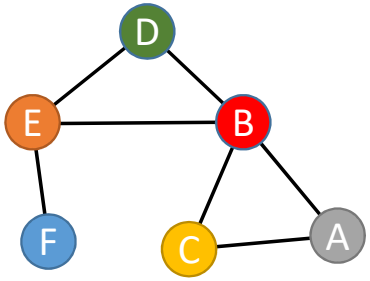
Graph: Permutation invariance

Desired properties for GNN

Order plan i

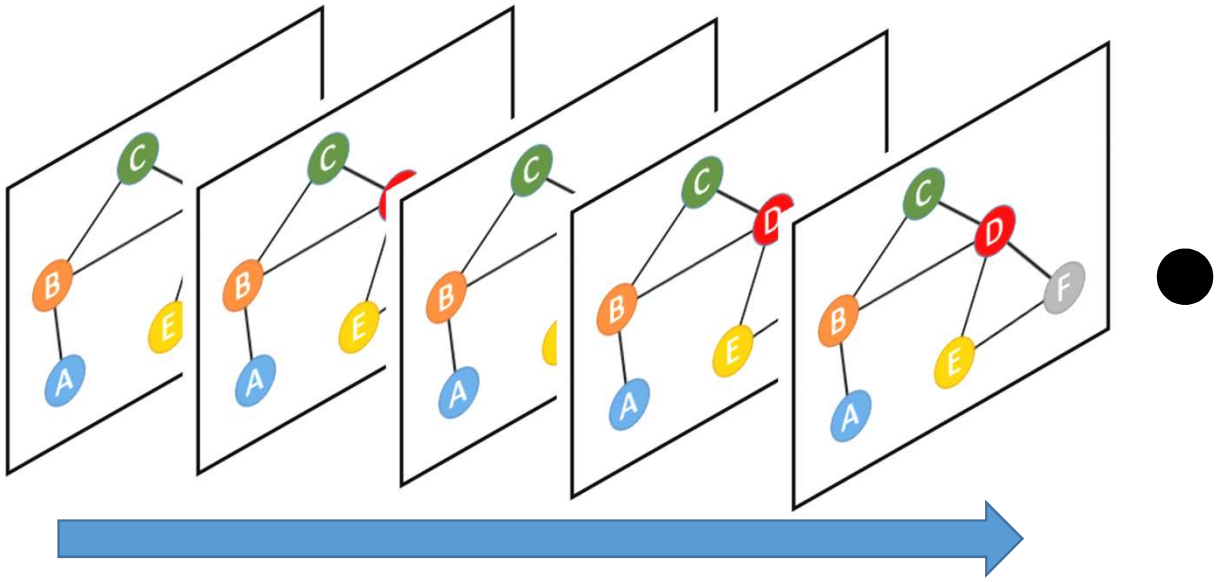


Order plan j



- Consider we use deep learning model to learn a function $f()$ that map a graph G = (V,E) to a vector R^d
- We can write f: $f(V, E) \rightarrow R^d$ (V is node feature matrix, E is represent by adjacency matrix)
- For a graph with m node , there are $m!$ order plans
- Then if $f(V_i, E_i) = f(V_j, E_j)$ works for every pair of order plans i,j
- We formally define that function f is permutation invariant for the graph represented with V , E matrix

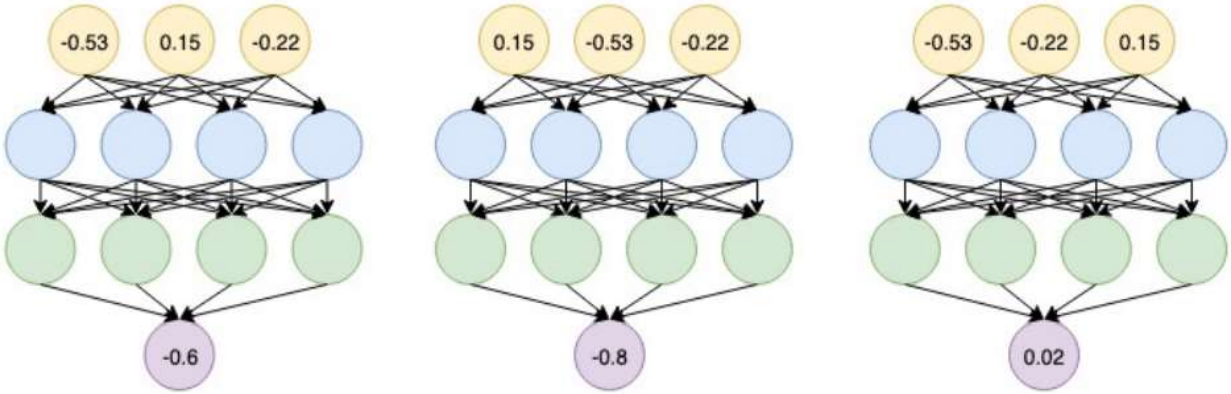
Graph: Permutation invariance



GNN consist of a series of permutation invariant layers!

Graph: Revisit MLP

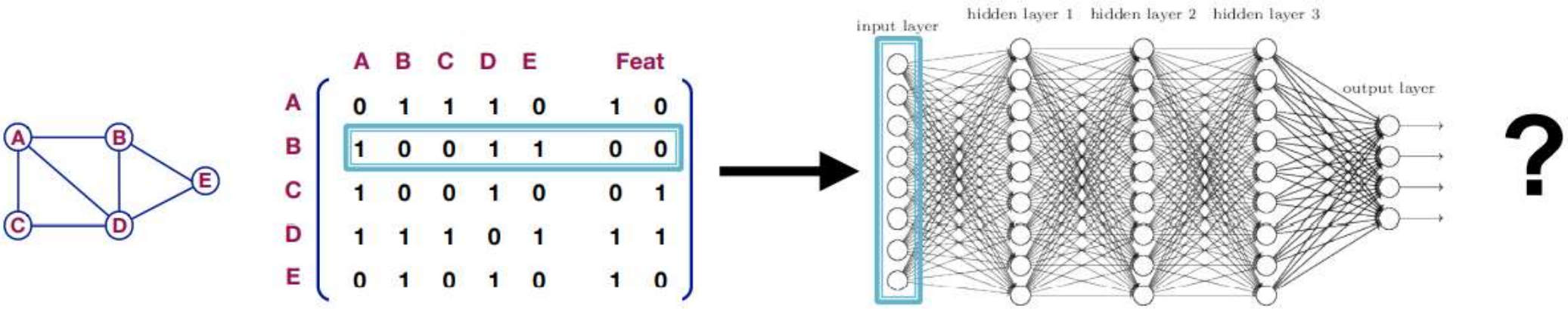
We have discussed that MLP is not permutation invariant



Graph: Revisit MLP

We have discussed that MLP is not permutation invariant

This also hold when we use graph data as input

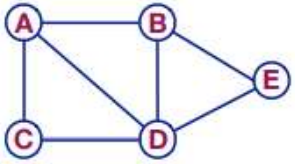


Change the order plan will change the sequence order and thus produce a different result!

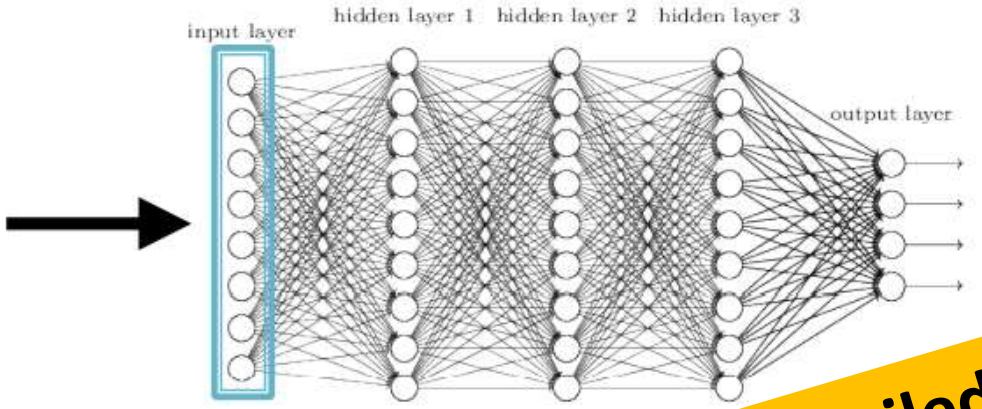
Graph: Revisit MLP

We have discussed that MLP is not permutation invariant

This also hold when we use graph data as input



	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



?

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

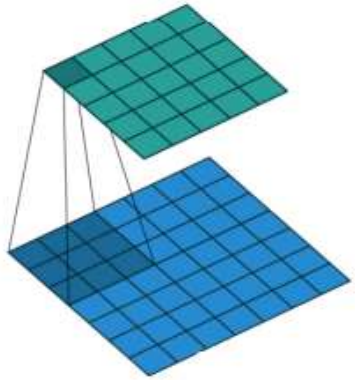
Naïve MLP failed on graph data!!

Story so far

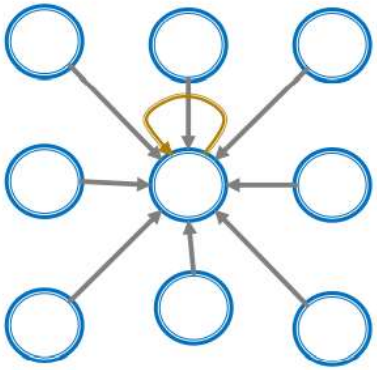
- **Graph can be represented by jointly use a feature matrix and a adjaceny matrix**
- **Graph representation does not have canonical order of node**
- **Permutation invariant is what we want to design a GNN**

A single layer of GNN: Graph Convolution

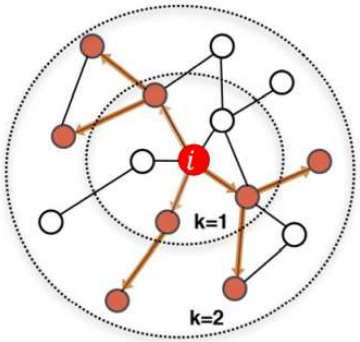
Key idea: Node's neighborhood defines a computation graph

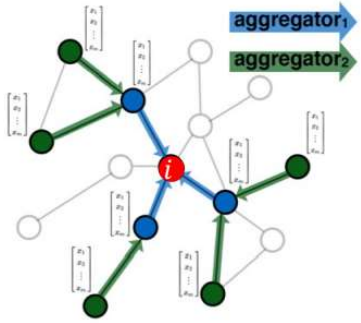


CNN: pixel convolution



CNN: pixel convolution



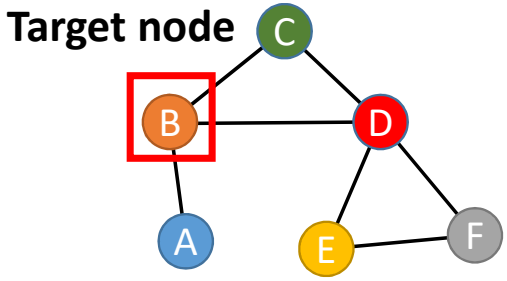


GNN: graph convolution

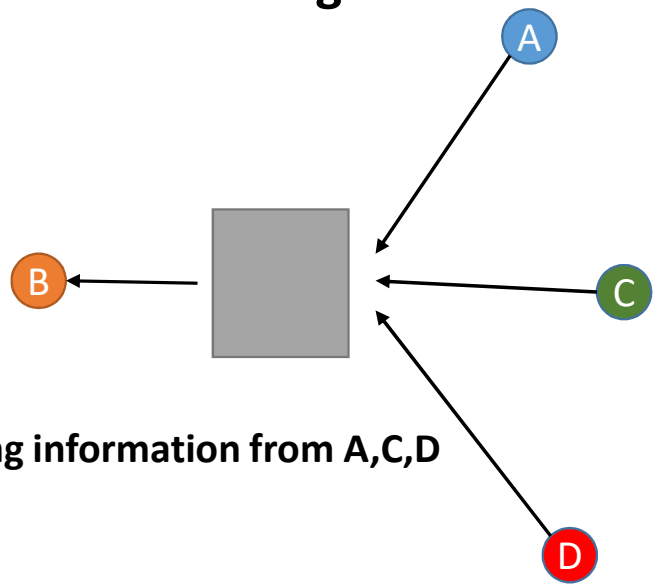
- Learning a node feature by propagating and aggregating neighbor information!
- Node embedding can be defined by local network neighborhoods!

A single layer of GNN: Graph Convolution

Key idea: Generate node embedding based on local network neighborhoods



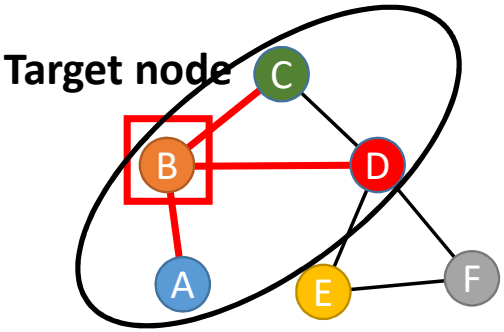
Considering 1 step of feature aggregation of the nearest neighbor



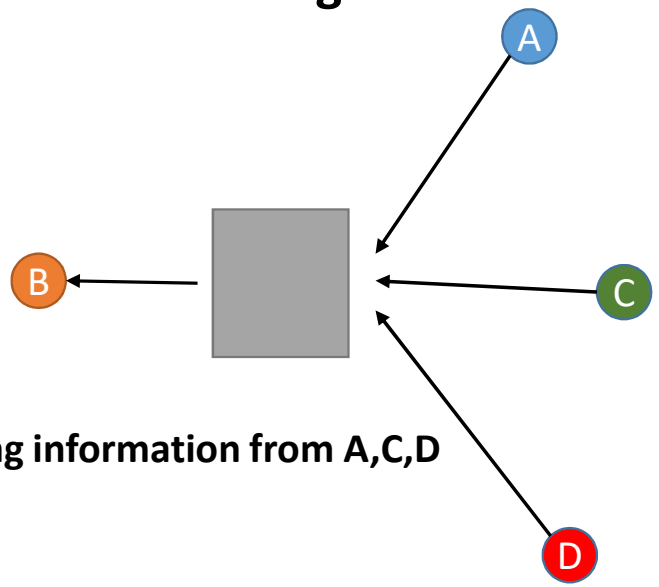
Processing information from A,C,D

A single layer of GNN: Graph Convolution

Key idea: Generate node embedding based on local network neighborhoods



Considering 1 step of feature aggregation of the nearest neighbor

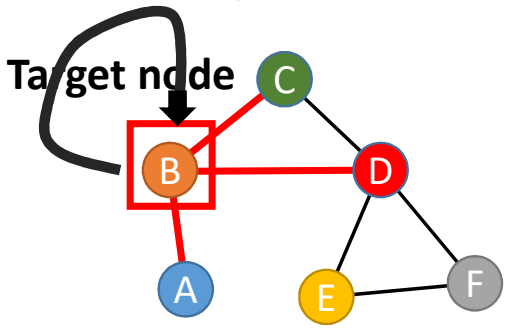


Processing information from A,C,D

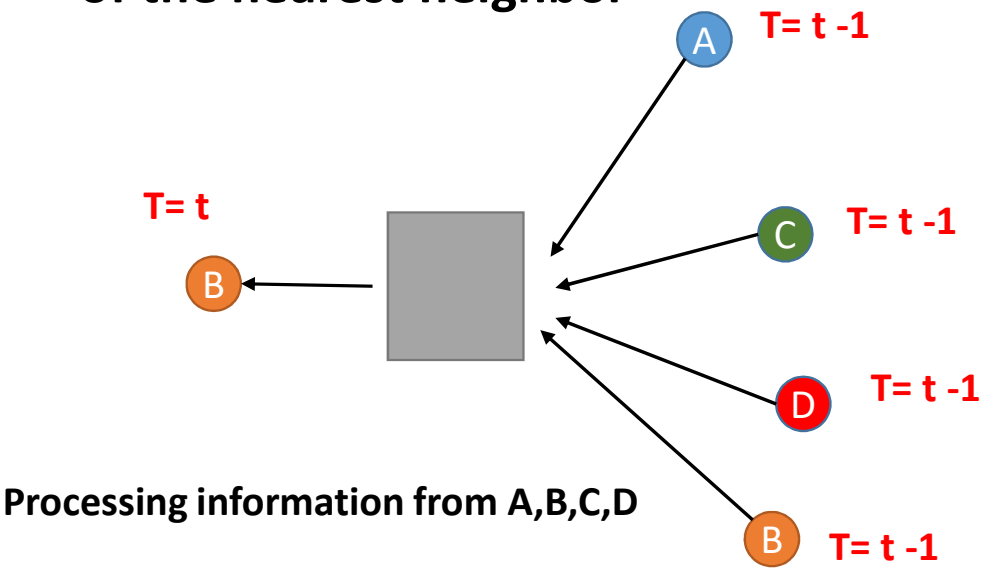
Now B have the information from it's first nearest neighbors

A single layer of GNN: Graph Convolution

Key idea: Generate node embedding based on local network neighborhoods



Considering 1 step of feature aggregation of the nearest neighbor

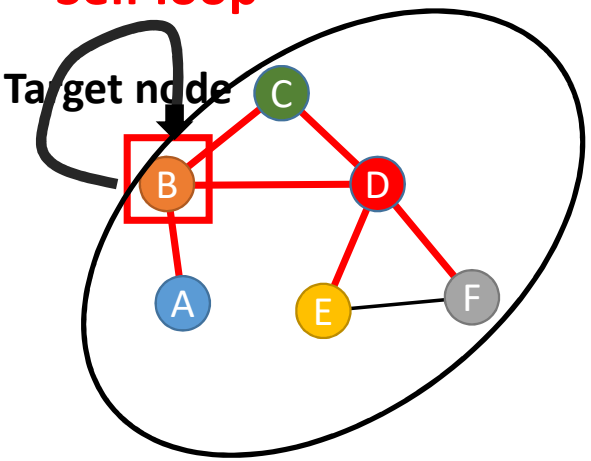


Also we don't want to lose information from B itself

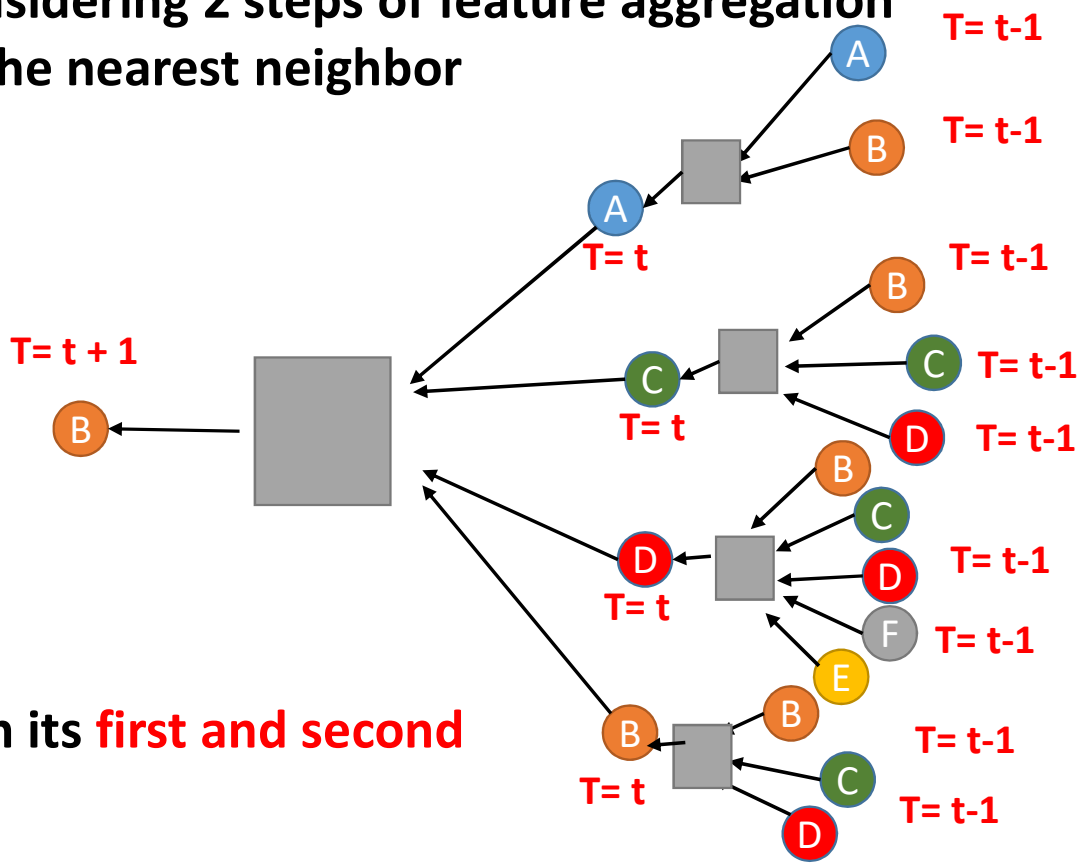
A single layer of GNN: Graph Convolution

Key idea: Generate node embedding based on local network neighborhoods

Self loop



Considering 2 steps of feature aggregation of the nearest neighbor

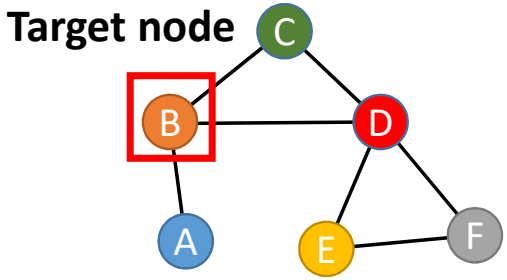


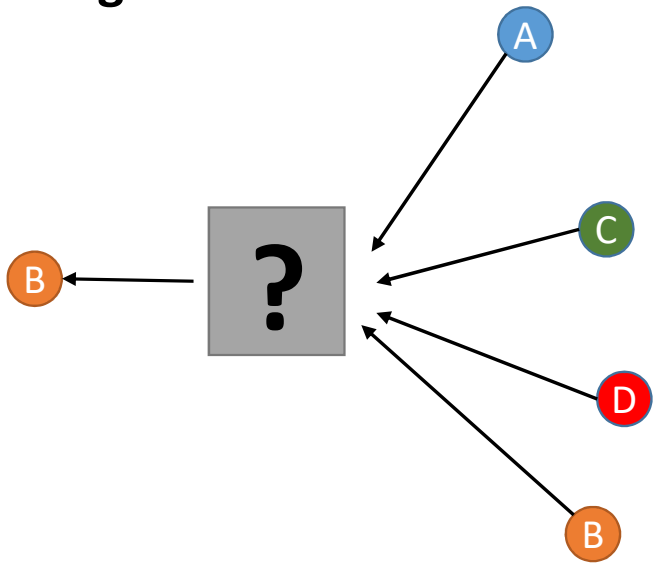
Now B have the information from its first and second nearest neighbors

A single layer of GNN: Graph Convolution

Key idea: Generate node embedding based on local network neighborhoods

How to process and mix the information from neighbor?

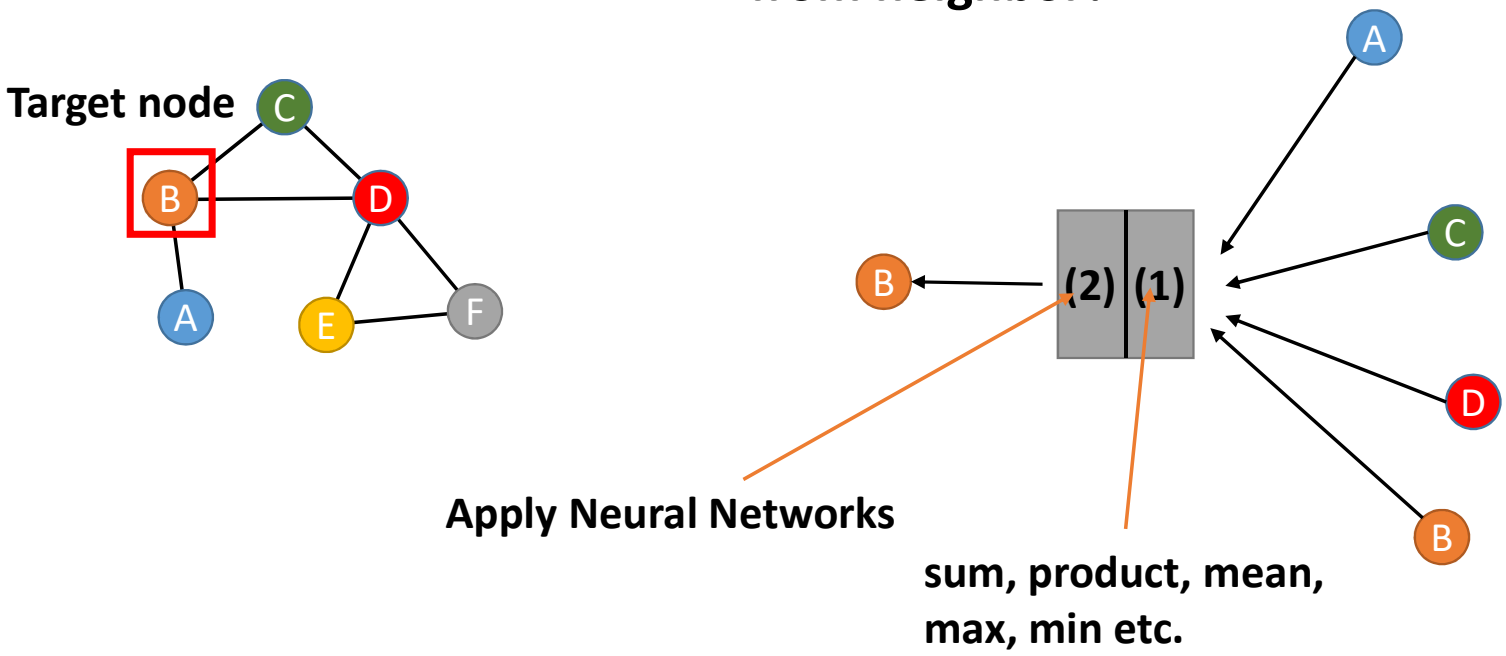




A single layer of GNN: Graph Convolution

Key idea: Generate node embedding based on local network neighborhoods

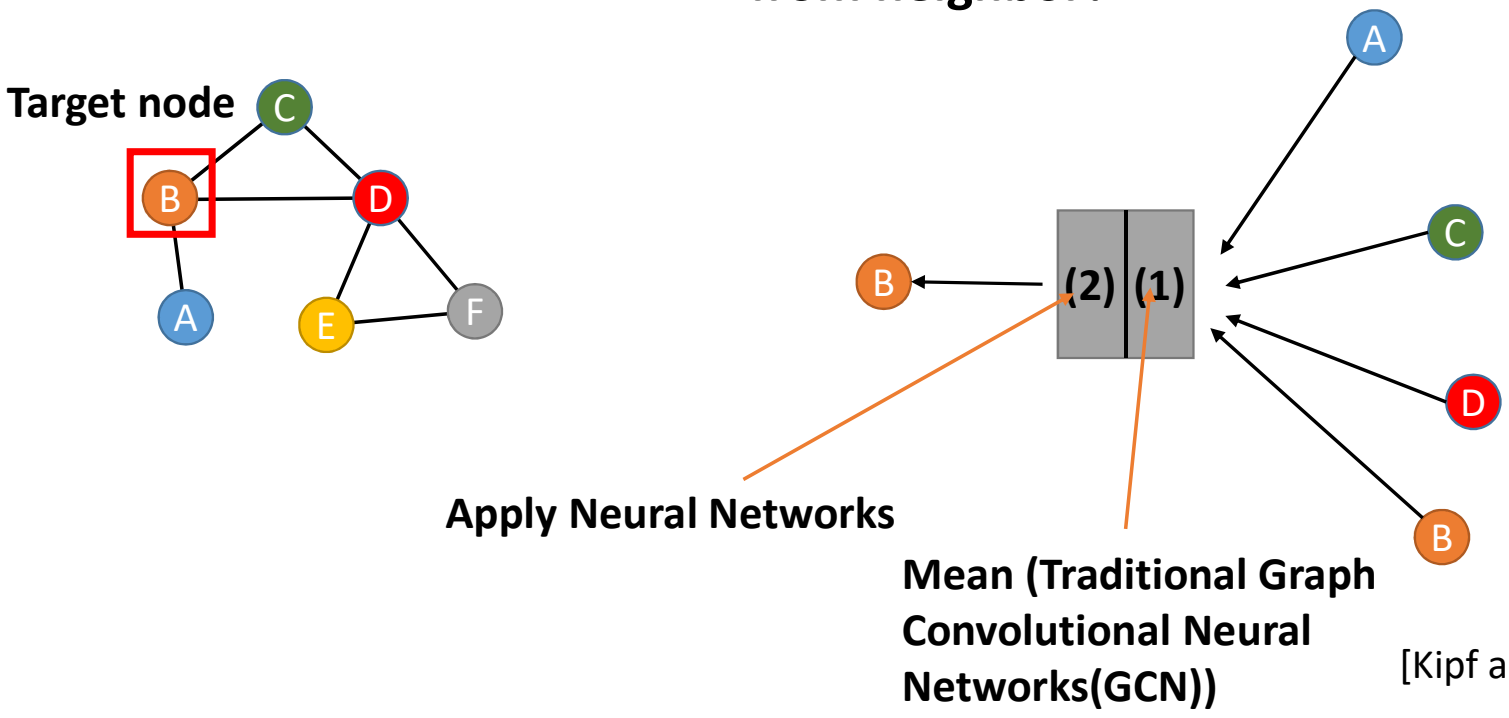
How to process and mix the information from neighbor?



A single layer of GNN: Graph Convolution

Key idea: Generate node embedding based on local network neighborhoods

How to process and mix the information from neighbor?

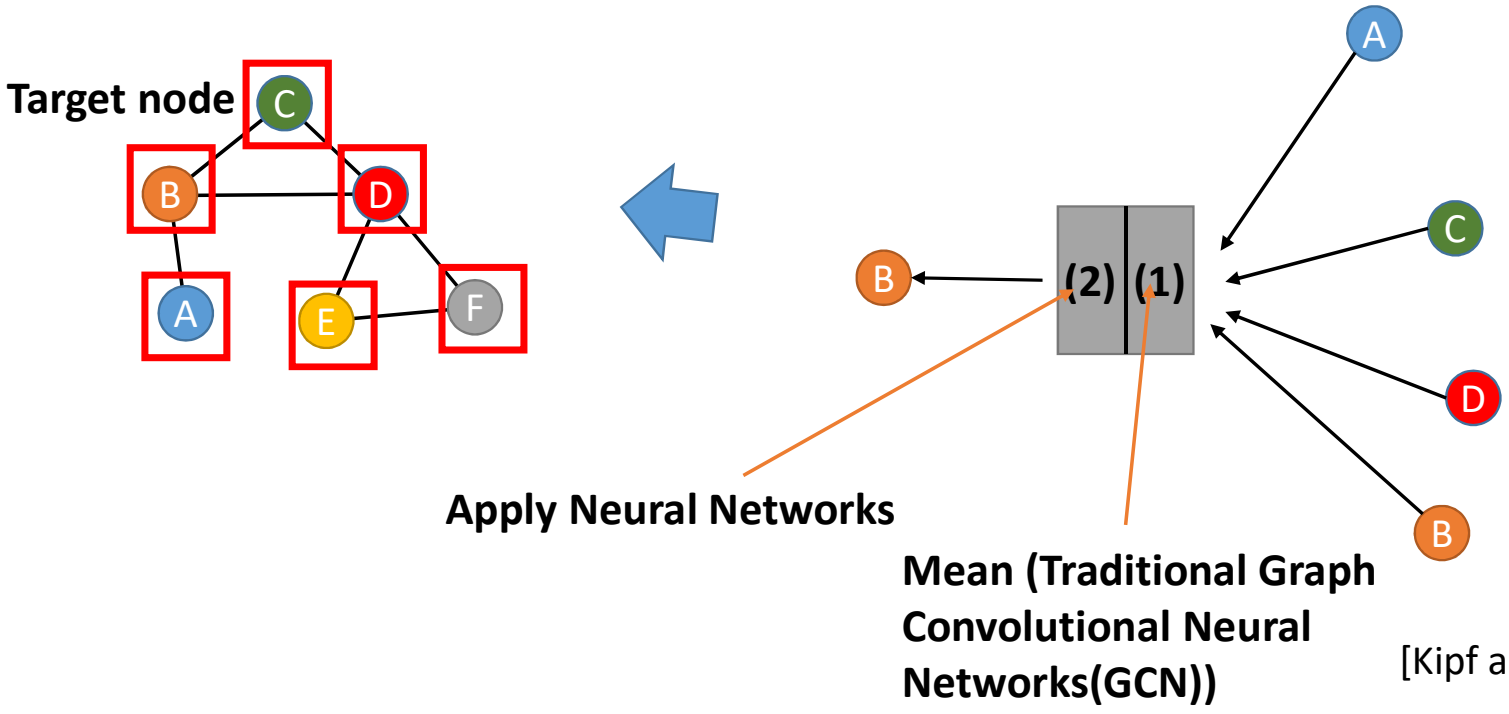


[Kipf and Welling, ICLR 2017]

A single layer of GNN: Graph Convolution

Key idea: Generate node embedding based on local network neighborhoods

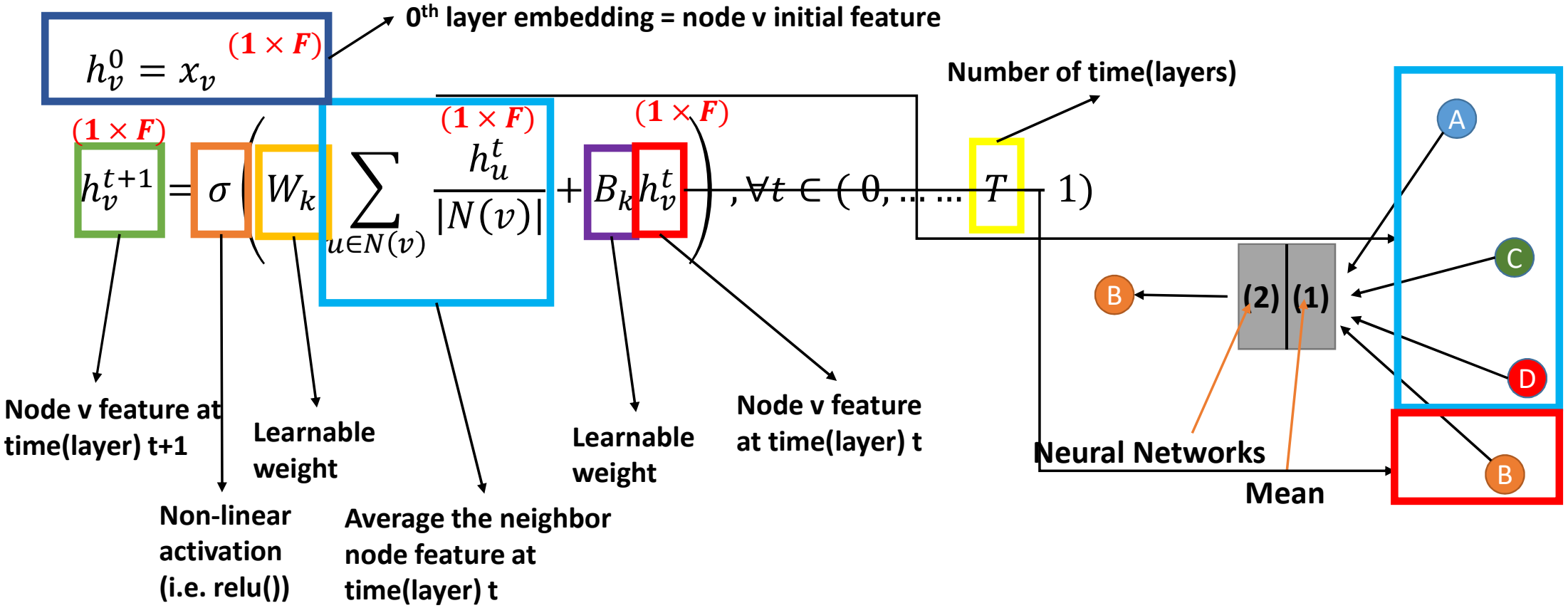
During a single Graph Convolution layer, we apply the feature aggregation to every node in the graph at the same time (T)



[Kipf and Welling, ICLR 2017]

A single layer of GNN: Graph Convolution-Forward

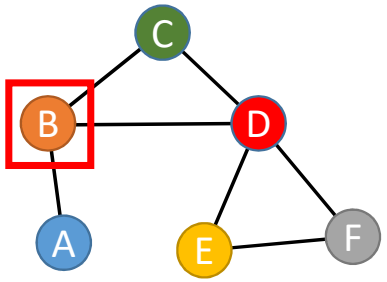
Math for a single layer of graph convolution

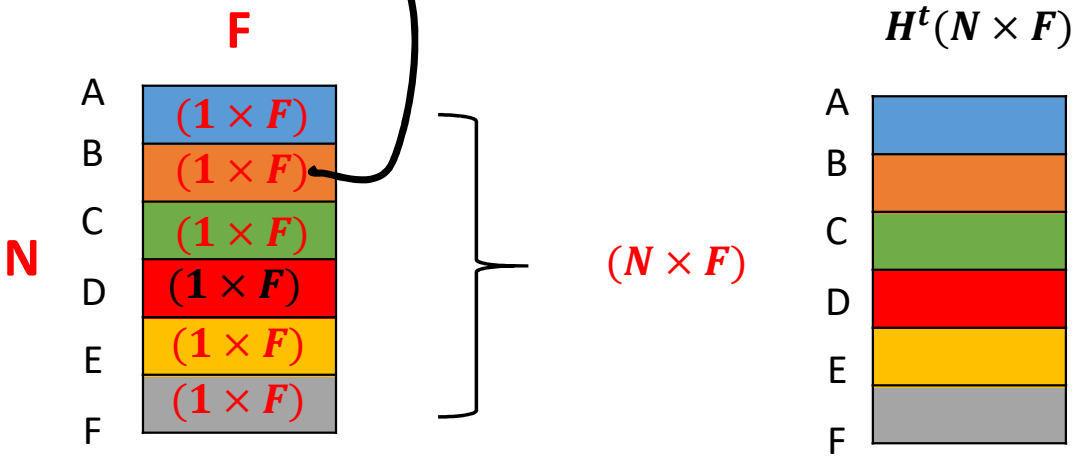


A single layer of GNN: Graph Convolution-Forward

Matrix form for a single layer of graph convolution

$$\underbrace{h_v^{t+1}}_{(1\times F)} = \sigma\left(W_k \sum_{u\in N(v)} \frac{\overbrace{h_u^t}^{(1\times F)}}{|N(v)|} + B_k \overbrace{h_v^t}^{(1\times F)} \right), \forall t \in (0, \ldots\ldots, T-1)$$





We stack multiple $h_v^t(1\times F)$ together into $H^t(N\times F)$

A single layer of GNN: Graph Convolution-Forward

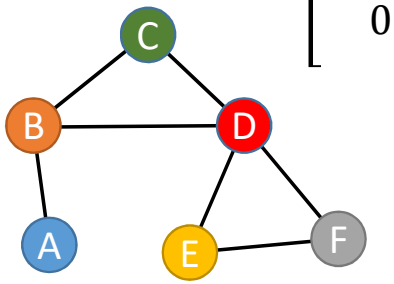
Matrix form for a single layer of graph convolution

$$h_v^{t+1} = \sigma\left(W_k \sum_{u \in N(v)} \frac{h_u^t}{|N(v)|} + B_k h_v^t\right), \forall t \in (0, \ldots\ldots, T-1)$$

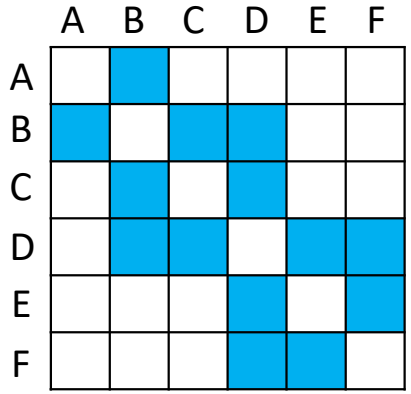
$(1 \times F)$ $(1 \times F)$ $(1 \times F)$

$$D^{-1}(N \times N) = \begin{bmatrix} \frac{1}{|N(v)|} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{1}{|N(v)|} \end{bmatrix}$$

N N

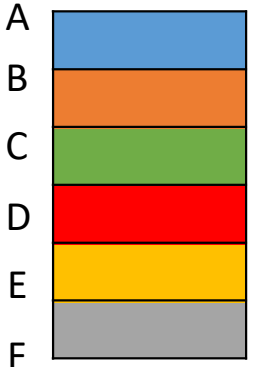


$A(N \times N)$



N N

$H^t(N \times F)$



F

A single layer of GNN: Graph Convolution-Forward

Matrix form for a single layer of graph convolution

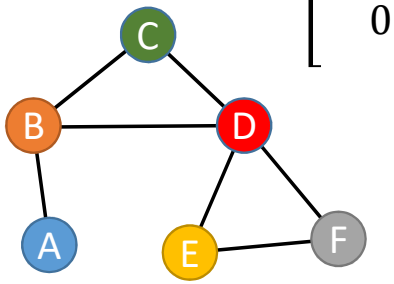
$$h_v^{t+1} = \sigma\left(W_k \sum_{u \in N(v)} \frac{h_u^t}{|N(v)|} + B_k h_v^t\right), \forall t \in (0, \ldots\ldots, T-1)$$

$(1 \times F)$ $(1 \times F)$ $(1 \times F)$

$\boldsymbol{D^{-1}(N \times N)}$

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

N N



$\times$

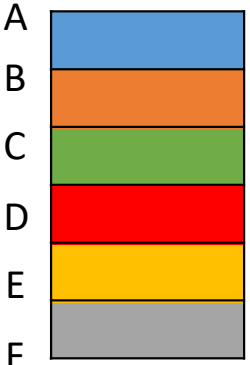
$\boldsymbol{A(N \times N)}$

	A	B	C	D	E	F
A		■				
B	■		■	■		
C		■		■		
D		■	■		■	■
E				■		■
F				■	■	

N N

$\times$ N

$\boldsymbol{H^t(N \times F)}$



F F

$\times$

Noted that W^T is a learnable weight matrix

$\boldsymbol{W^T(F \times F)}$

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

F

A single layer of GNN: Graph Convolution-Forward

Matrix form for a single layer of graph convolution

$$h_v^{t+1} = \sigma\left(W_k \sum_{u \in N(v)} \frac{h_u^t}{|N(v)|} + B_k h_v^t\right), \forall t \in (0, \ldots\ldots, T-1)$$

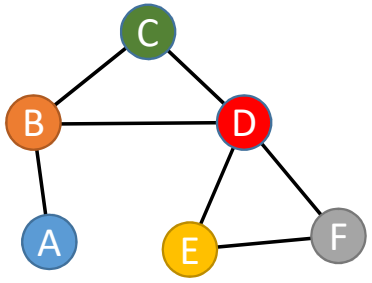
$(1 \times F)$ $(1 \times F)$ $(1 \times F)$

$H^t(N \times F)$ $W^T(F \times F)$

N F F F

Why put W^T on the right hand site of H^t?

Why not left? With a shape of $(N \times N)$?



A single layer of GNN: Graph Convolution-Forward

Matrix form for a single layer of graph convolution

$$h_v^{t+1} = \sigma\left(W_k \sum_{u \in N(v)} \frac{h_u^t}{|N(v)|} + B_k h_v^t\right), \forall t \in (0, \ldots\ldots, T-1)$$

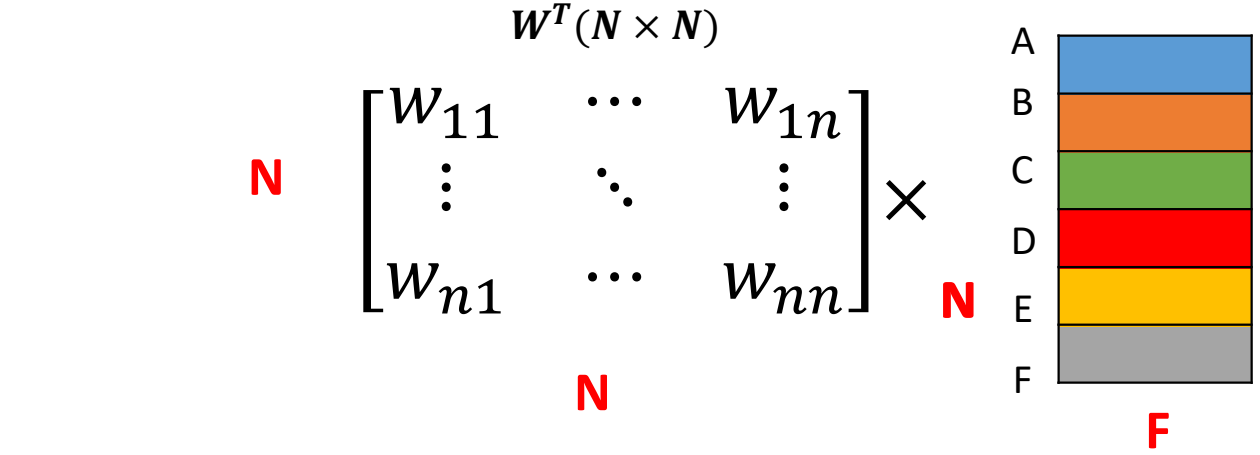
$(1 \times F)$ $(1 \times F)$ $(1 \times F)$

$W^T(N \times N)$ $H^t(N \times F)$

$$\begin{matrix} N \\ \end{matrix} \begin{bmatrix} W_{11} & \cdots & W_{1n} \\ \vdots & \ddots & \vdots \\ W_{n1} & \cdots & W_{nn} \end{bmatrix} \times \begin{matrix} N \\ \end{matrix}$$

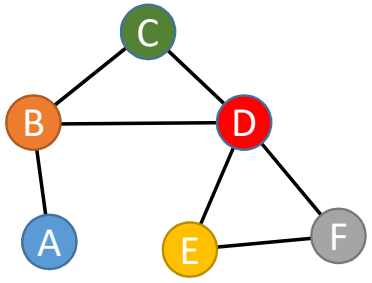
N

N



Like this?

Seems like nothing goes wrong, the result matrix shape is still $(N \times F)$?



What happen if we still put W on the left hand site?

A single layer of GNN: Graph Convolution-Forward

Matrix form for a single layer of graph convolution

$$h_v^{t+1} = \sigma\left(W_k \sum_{u \in N(v)} \frac{h_u^t}{|N(v)|} + B_k h_v^t\right), \forall t \in (0, \ldots\ldots, T-1)$$

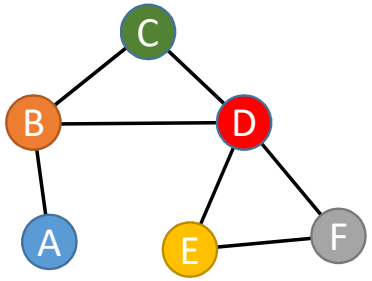
$(1 \times F)$ $(1 \times F)$ $(1 \times F)$

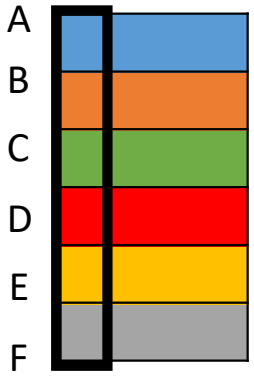
$$W^T(N \times N) \quad H^t(N \times F)$$

$$\begin{matrix} N \\ \end{matrix} \begin{bmatrix} W_{11} & \cdots & W_{1n} \\ \vdots & \ddots & \vdots \\ W_{n1} & \cdots & W_{nn} \end{bmatrix} \times \begin{matrix} N \\ \end{matrix}$$

N

F





Seems like nothing goes wrong, the result matrix shape is still $(N \times F)$?

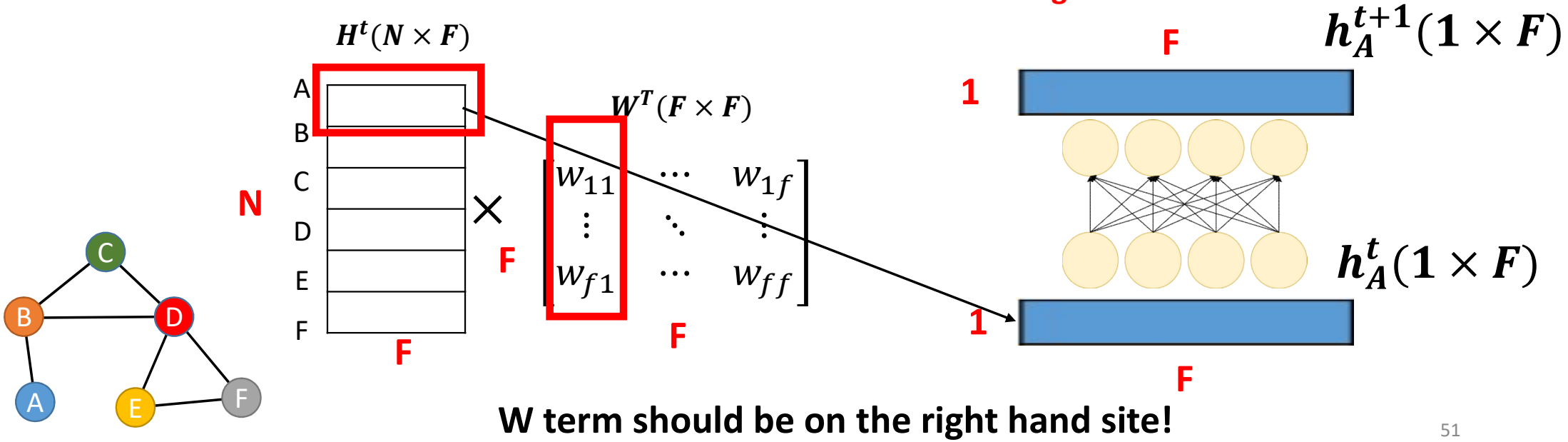
No, it's wrong, because we are still mixing information among different nodes, which has the same function with adjacent matrix, feature within node does not receive any mixing

A single layer of GNN: Graph Convolution-Forward

Matrix form for a single layer of graph convolution

$$h_v^{t+1} = \sigma\left(\underbrace{W_k}_{(1\times F)} \sum_{u\in N(v)} \frac{\overbrace{h_u^t}^{(1\times F)}}{|N(v)|} + \overbrace{B_k h_v^t}^{(1\times F)}\right), \forall t \in (0, \ldots\ldots, T-1)$$

Learnable weight is used to mix information along the feature within a single node

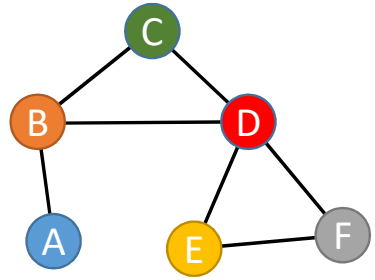


W term should be on the right hand site!

A single layer of GNN: Graph Convolution-Forward

Matrix form for a single layer of graph convolution

$$h_v^{t+1} = \sigma\left(\underbrace{W_k \sum_{u \in N(v)} \frac{h_u^t}{|N(v)|}}_{(1\times F)} + \underbrace{B_k h_v^t}_{(1\times F)}\right), \forall t \in (0, \ldots\ldots, T-1) \quad (1\times F)$$



$$\underset{N\times N}{\boldsymbol{D^{-1}}} \times \underset{N\times N}{\boldsymbol{A}} \times \underset{N\times F}{\boldsymbol{H^t}} \times \underset{F\times F}{\boldsymbol{W^T}} = \underset{N\times F}{\boldsymbol{H^{t+1}}}$$

$$\boldsymbol{D^{-1}}(N\times N) = \begin{bmatrix} \frac{1}{|N(v)|} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{1}{|N(v)|} \end{bmatrix}$$

$$\boldsymbol{A}(N\times N) = \begin{array}{c|cccccc} & A & B & C & D & E & F \\ \hline A & & \blacksquare & & & & \\ B & \blacksquare & & \blacksquare & \blacksquare & & \\ C & & \blacksquare & & \blacksquare & & \\ D & & \blacksquare & \blacksquare & & \blacksquare & \blacksquare \\ E & & & & \blacksquare & & \blacksquare \\ F & & & & \blacksquare & \blacksquare & \end{array}$$

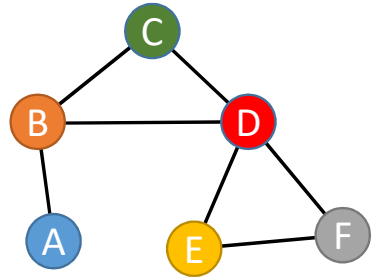
$$\boldsymbol{H^t}(N\times F) = \begin{array}{c|c} & \\ \hline A & \\ B & \\ C & \\ D & \\ E & \\ F & \end{array}$$

$$\boldsymbol{W^T}(F\times F) = \begin{bmatrix} w_{11} & \cdots & w_{1f} \\ \vdots & \ddots & \vdots \\ w_{f1} & \cdots & w_{ff} \end{bmatrix}$$

$$\boldsymbol{H^{t+1}}(N\times F) = \begin{array}{c|c} & \\ \hline A & \\ B & \\ C & \\ D & \\ E & \\ F & \end{array}$$

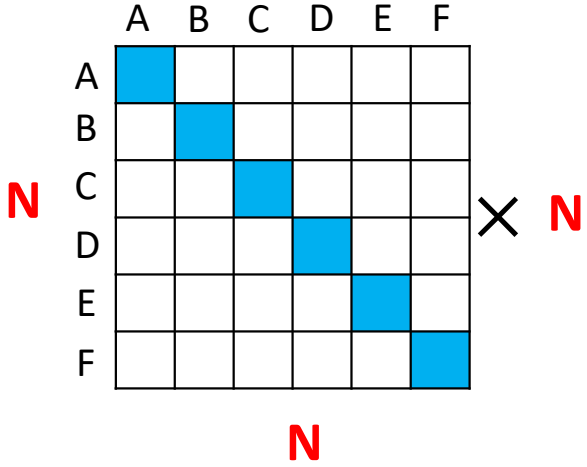
A single layer of GNN: Graph Convolution-Forward

Matrix form for a single layer of graph convolution

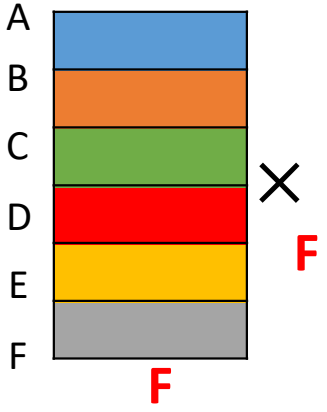


$$\underset{(1\times F)}{h_v^{t+1}} = \sigma\left(W_k \sum_{u\in N(v)} \frac{\overset{(1\times F)}{h_u^t}}{|N(v)|} + \overset{(1\times F)}{B_k h_v^t} \right), \forall t \in (0, \ldots\ldots, T-1)$$

$A'(N\times N)$ Indentity matrix



$H_v^t\ (N\times F)$



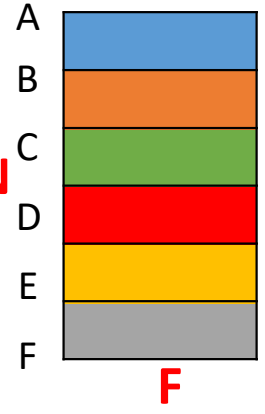
Noted that B^T is a learnable weight matrix

$B^T(F\times F)$

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

F

$H_v^{t+1}(N\times F)$



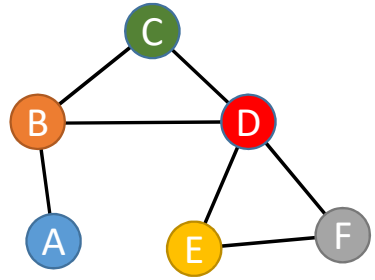
Self loop adjacent matrix is a diagonal matrix!

A single layer of GNN: Graph Convolution-Forward

Matrix form for a single layer of graph convolution

$$h_v^{t+1} = \sigma\left(W_k \sum_{u \in N(v)} \frac{h_u^t}{|N(v)|} + B_k h_v^t \right), \forall t \in (0, \ldots\ldots, T-1)$$

$(1 \times F)$ $(1 \times F)$ $(1 \times F)$



Now let's rewrite the scalar form above into matrix form

$$\boldsymbol{H^{t+1} = \sigma(D^{-1} A H^t W^T + A' H_v^t B^T)}$$

$(N \times F)$ $(F \times F)$

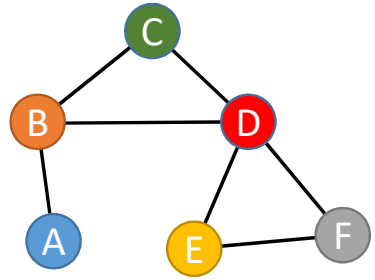
Non-Linear Activation

Aggregating neighbor node feature

Aggregating self node feature

A single layer of GNN: Graph Convolution-Forward

Matrix form for a single layer of graph convolution



$(1 \times F)$ $(1 \times F)$ $(1 \times F)$

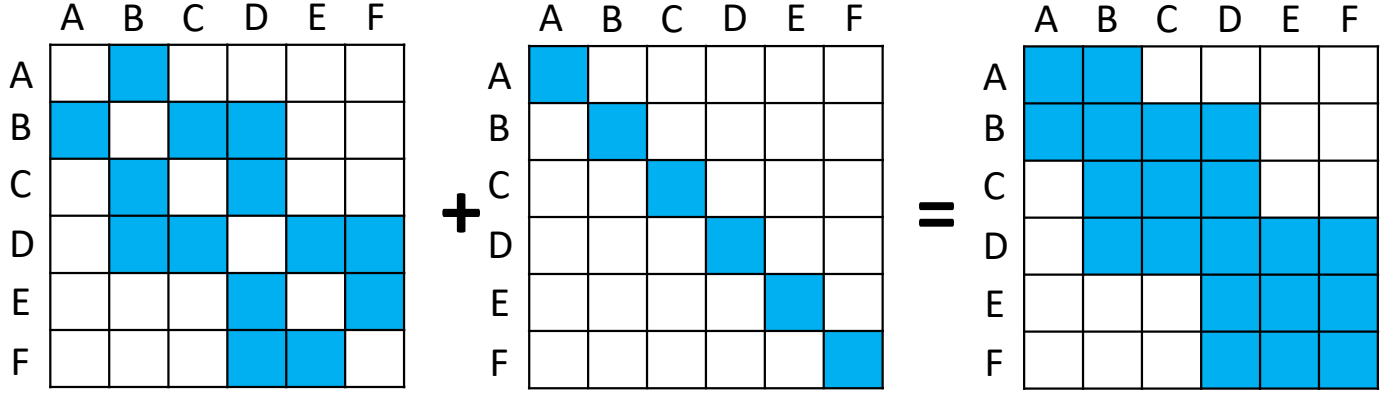
$$h_v^{t+1} = \sigma\left(W_k \sum_{u \in N(v)} \frac{h_u^t}{|N(v)|} + B_k h_v^t\right), \forall t \in (0, \ldots\ldots, T-1)$$

$(N \times N)$

$(N \times N)(N \times N)(N \times F)(F \times F)$

$$\boldsymbol{H^{t+1} = \sigma(D^{-1}\widehat{A}H^{t'}W'^{T})}$$

$\boldsymbol{A(N \times N)}$ $\boldsymbol{A'(N \times N)}$ $\boldsymbol{\widehat{A}(N \times N)}$

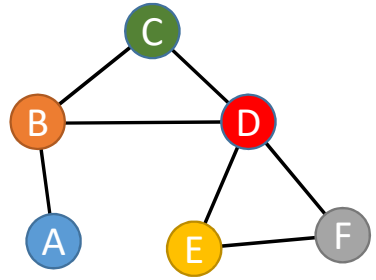


A single layer of GNN: Graph Convolution-Forward

Matrix form for a single layer of graph convolution

$$h_v^{t+1} = \sigma\left(W_k \sum_{u \in N(v)} \frac{h_u^t}{|N(v)|} + B_k h_v^t\right), \forall t \in (0, \ldots\ldots, T-1)$$

$(1 \times F)$ $(1 \times F)$ $(1 \times F)$

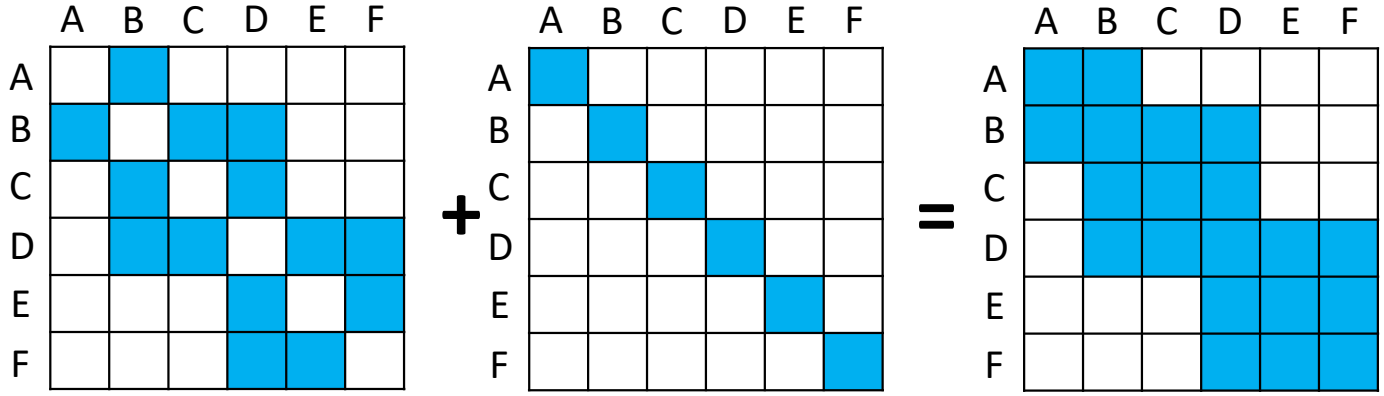


$$\boldsymbol{H^{t+1} = \sigma(D^{-1}\hat{A}H^{t'}W'^{T})}$$

$(N \times N)(N \times N)(N \times F)(F \times F)$ $(N \times N)$

Forward equation for GCN

$\boldsymbol{A(N \times N)}$ $\boldsymbol{A'(N \times N)}$ $\boldsymbol{\hat{A}(N \times N)}$



Story so far

- **Node's neighborhood defines a computation graph**
- **Graph Convolution layer forward**

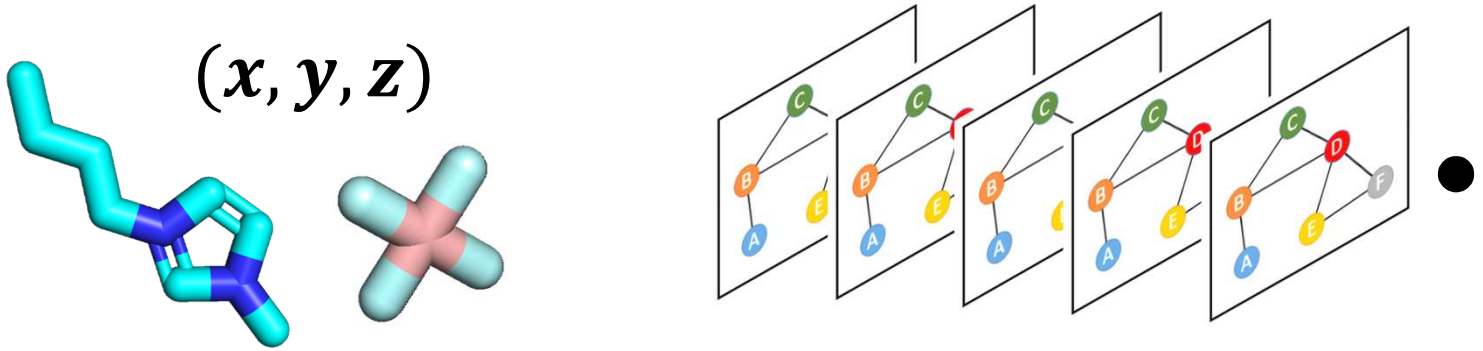
Reference

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

Graph: Revisit MLP

Actually, if we include spatial information into GNN node feature

GNN will become neither translation invariant nor rotation invariant



But the permutation invariance for GNN still hold

This problem leads to another topic on GNN which is Equivariant Graph Neural Network, but we don't have time to discuss on that in the lecture