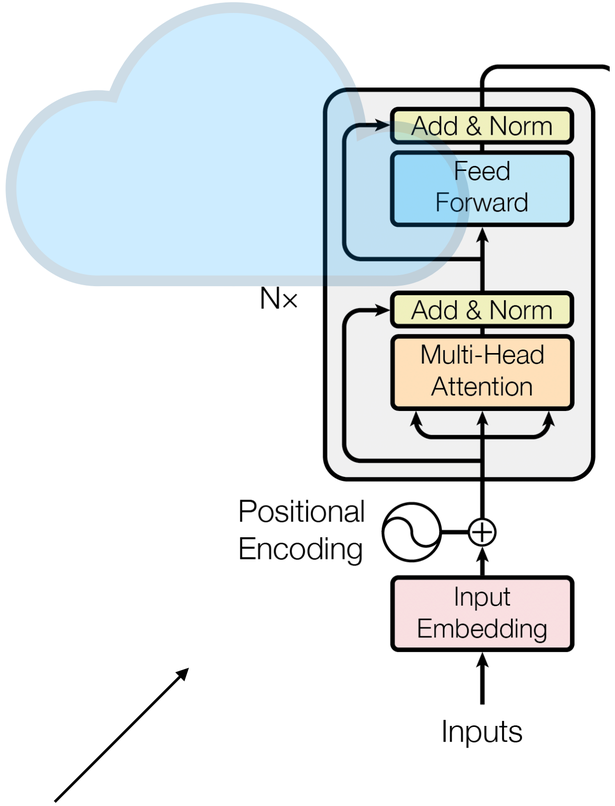
Thinking Like Transformers

Practical Session



13945 + 2903482

2917427

Follow along:

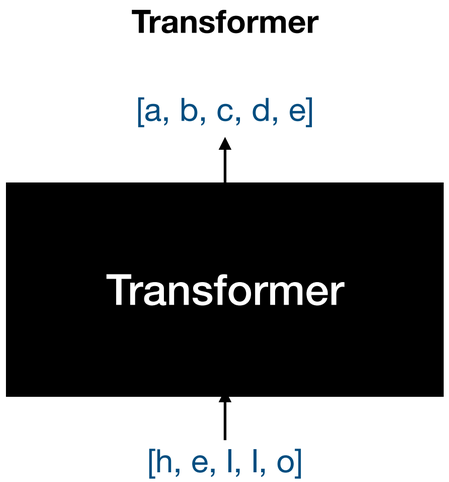
github.com/tech-srl/RASP

1. clone
2. set up:
   1. macOS, linux: ./install.sh
   2. windows: follow “windows instructions.txt”
3. run:
   1. macOS, linux: ./rasp.sh
   2. windows: python3 RASP_support/REPL.py

RASP introduced: Thinking like transformers (paper, ICML 2021) **Gail Weiss**, Yoav Goldberg, Eran Yahav

Long addition: Thinking like transformers (blog, ICLR 2023) Alexander Rush, **Gail Weiss**

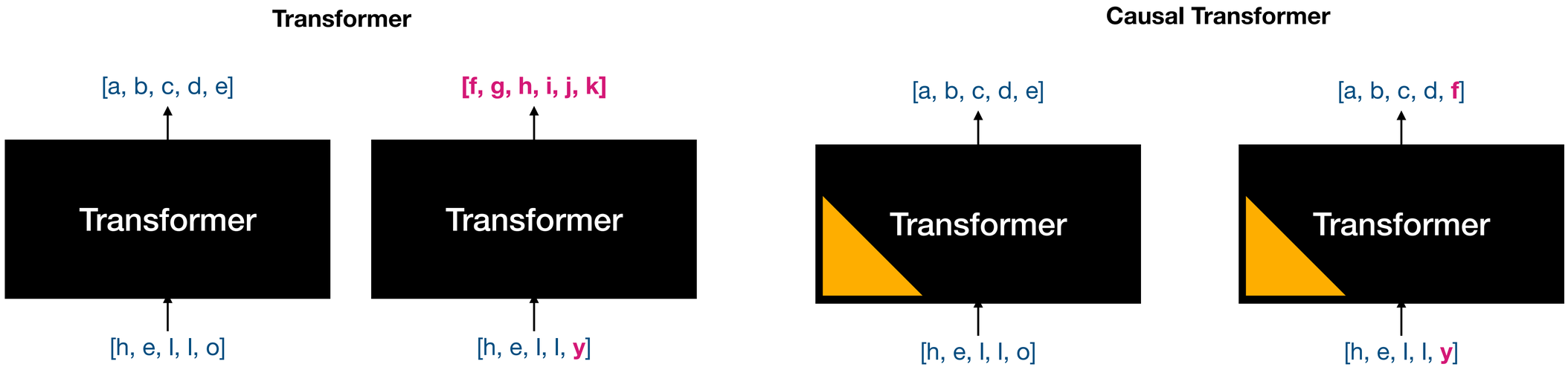
Note - model vs use



A **transformer** by itself is a length preserving function!

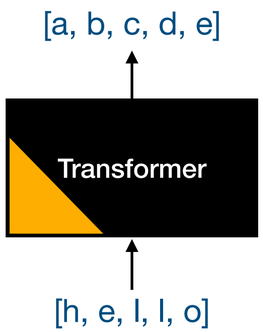
To avoid confusion, we'll briefly describe the non-length-preserving behaviour that you're used to before we start

Note - model vs use



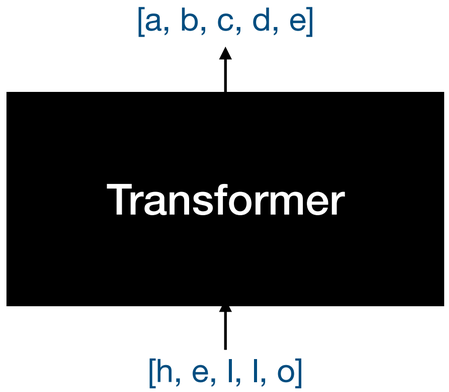
Note - model vs use

Causal/autoregressive transformer in classic generative loop

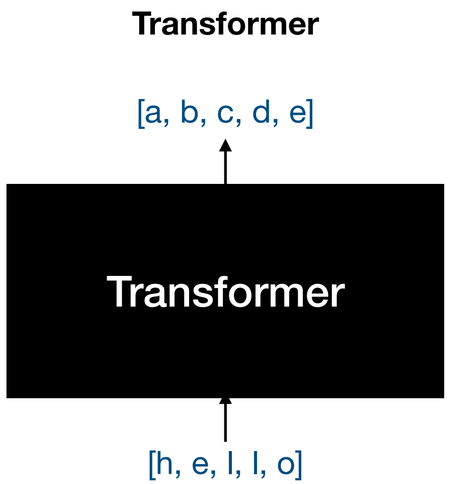


Prompt

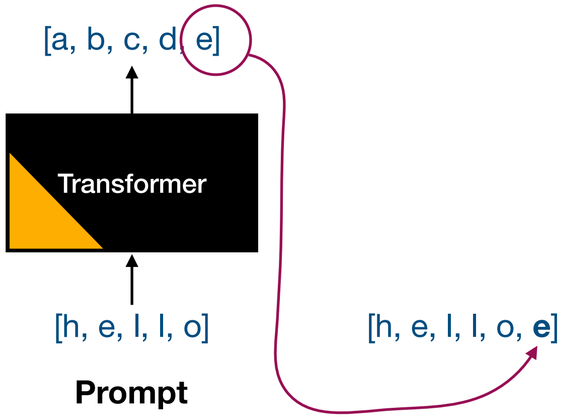
Transformer



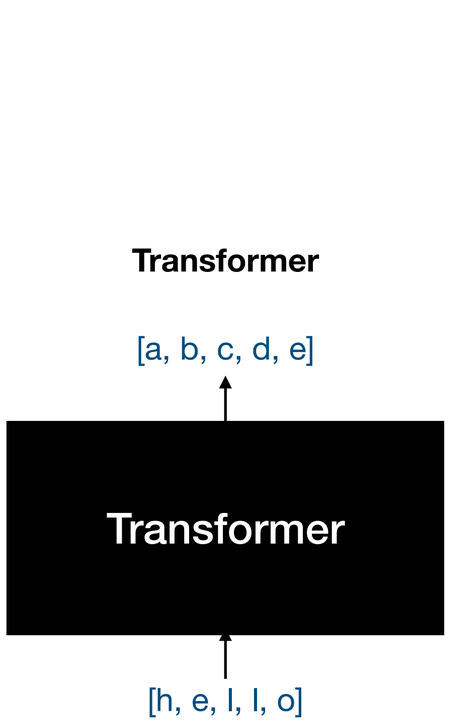
Note - model vs use



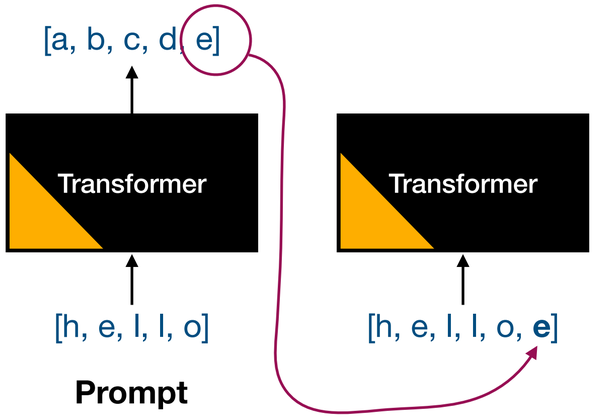
Causal/autoregressive transformer in classic generative loop



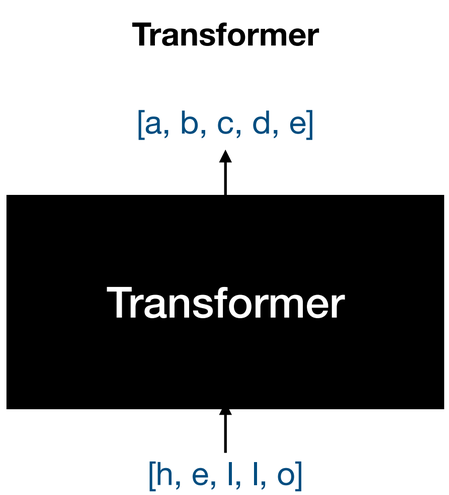
Note - model vs use



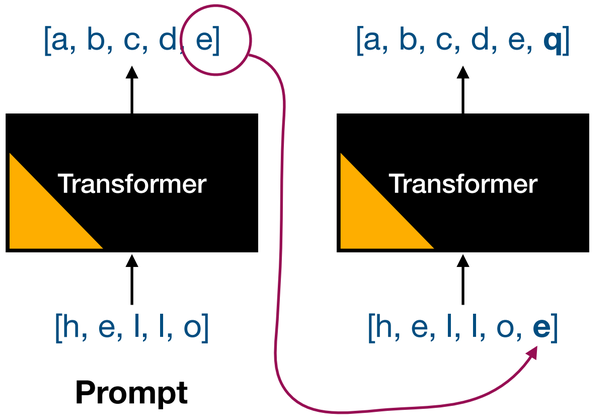
Causal/autoregressive transformer in classic generative loop



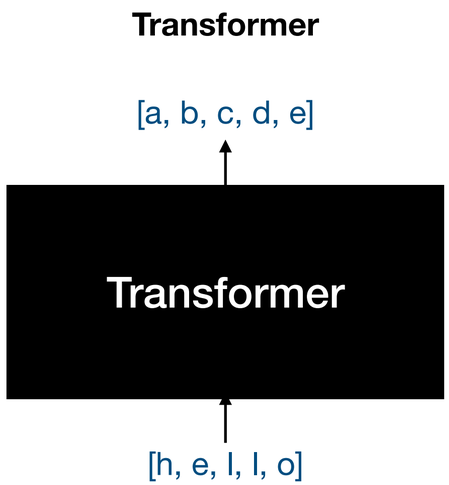
Note - model vs use



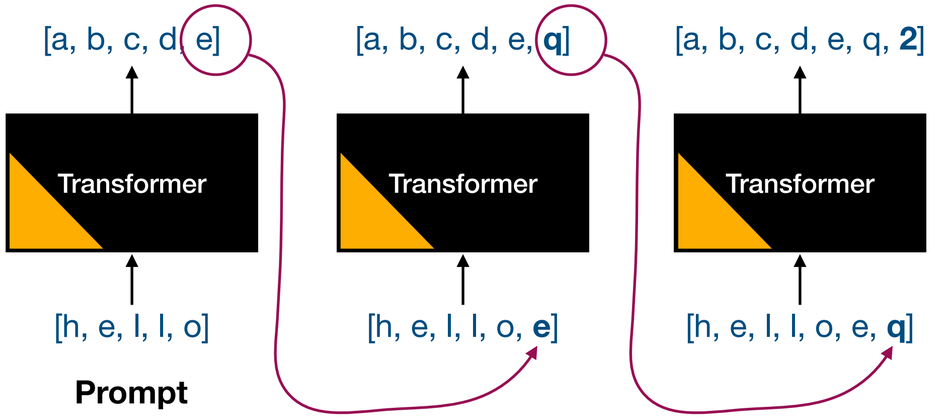
Causal/autoregressive transformer in classic generative loop



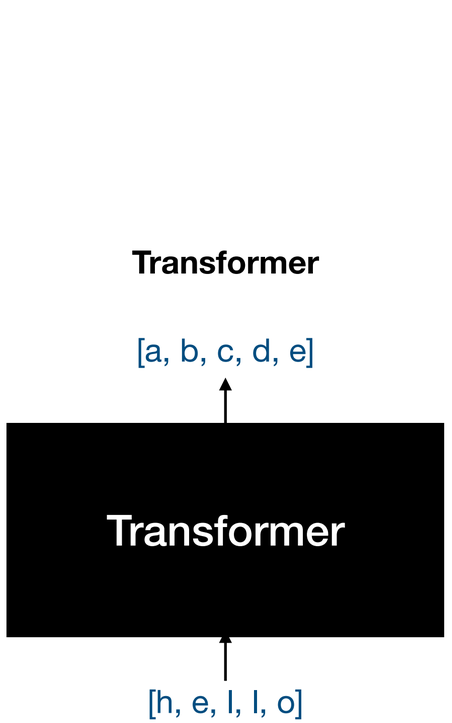
Note - model vs use

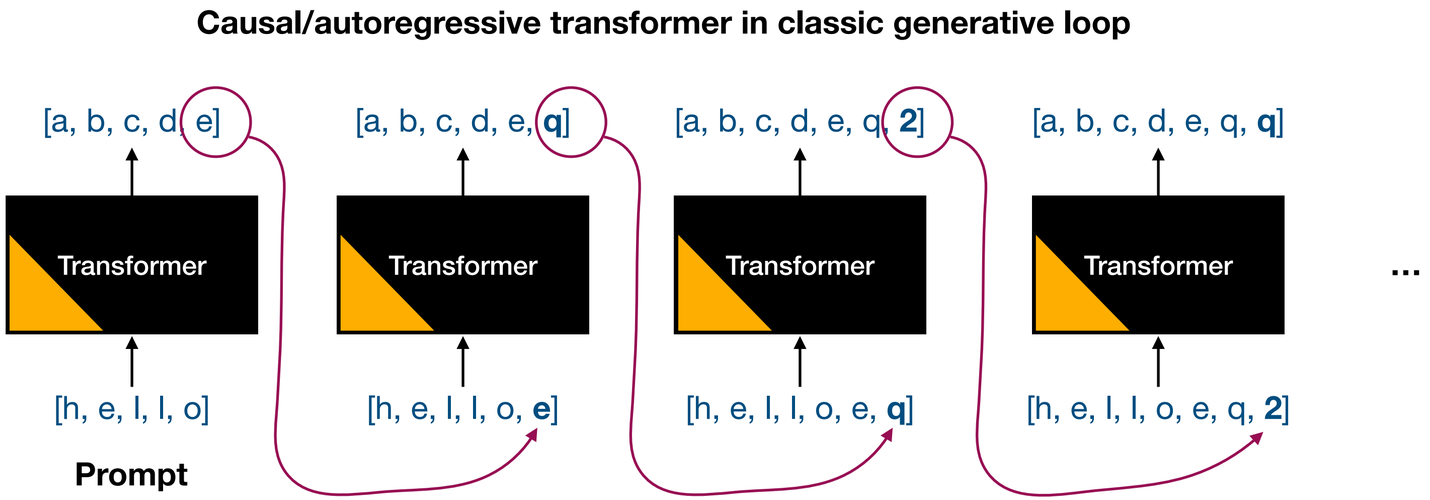


Causal/autoregressive transformer in classic generative loop

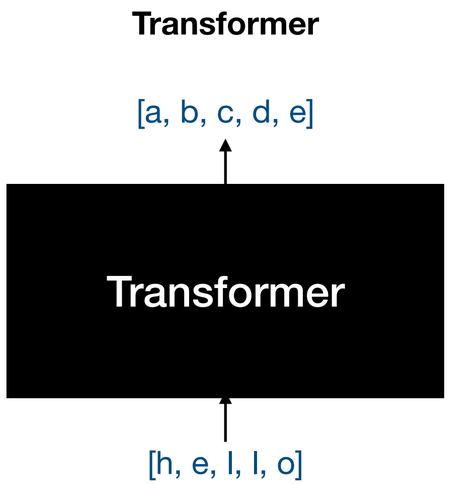


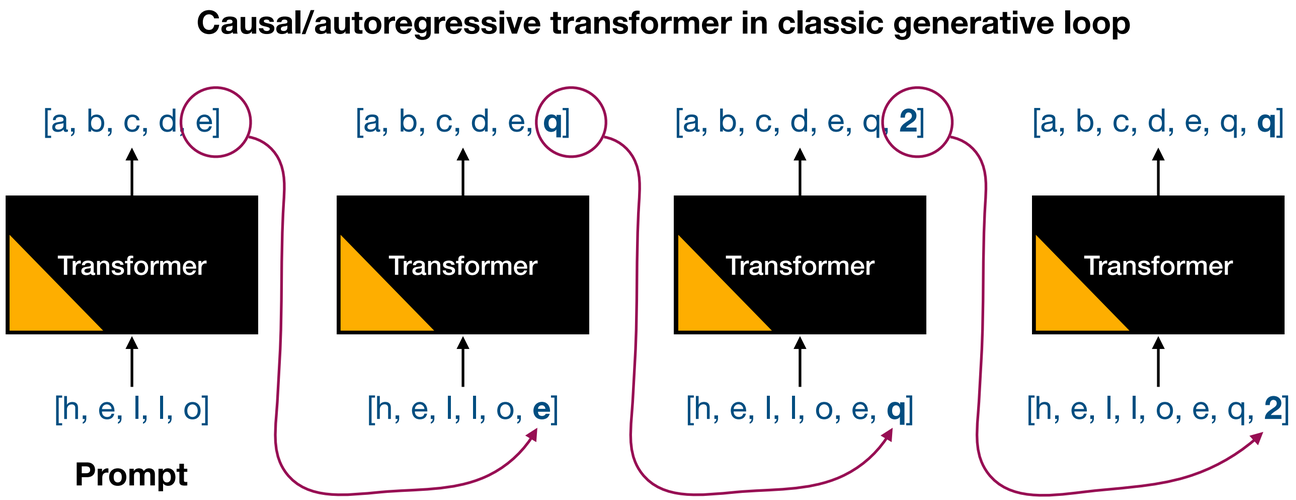
Note - model vs use





Note - model vs use

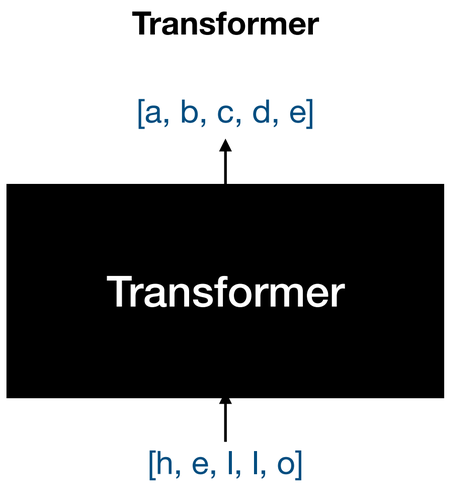




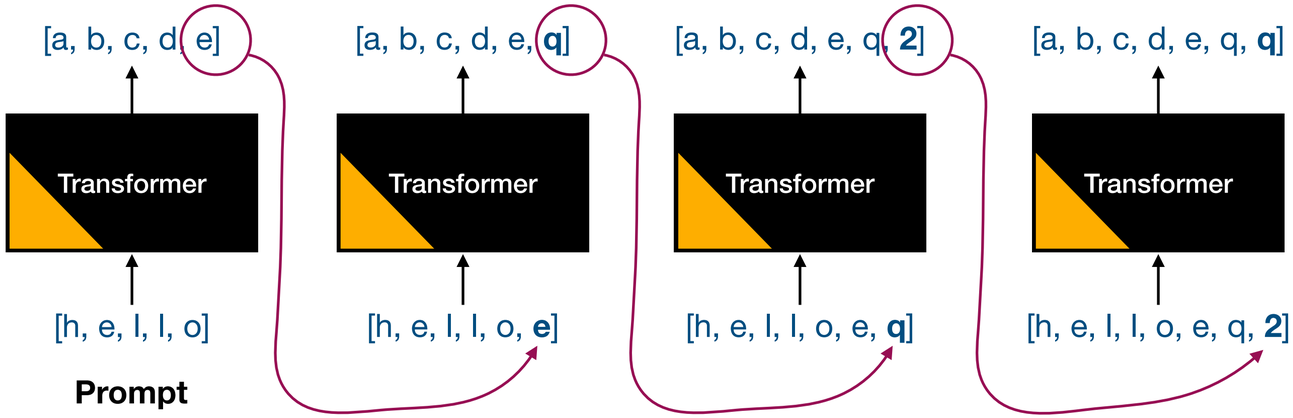
...

The generative loop moves to non-length preserving behaviour

Note - model vs use



Causal/autoregressive transformer in classic generative loop

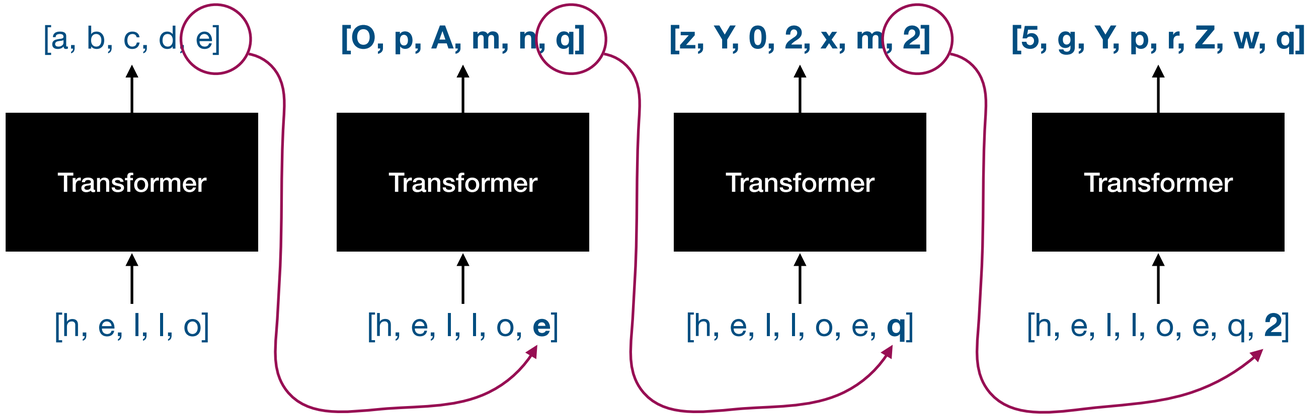


...

The generative loop moves to non-length preserving behaviour

works well

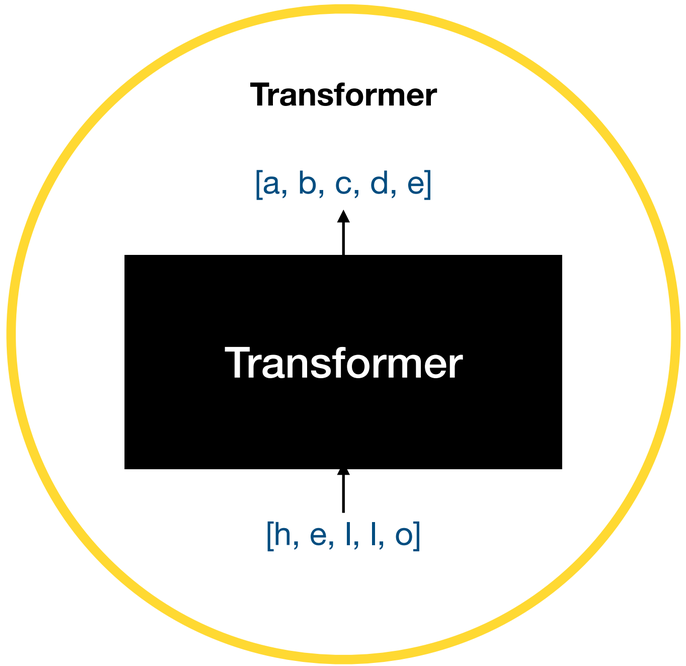
Plain transformer in classic generative loop

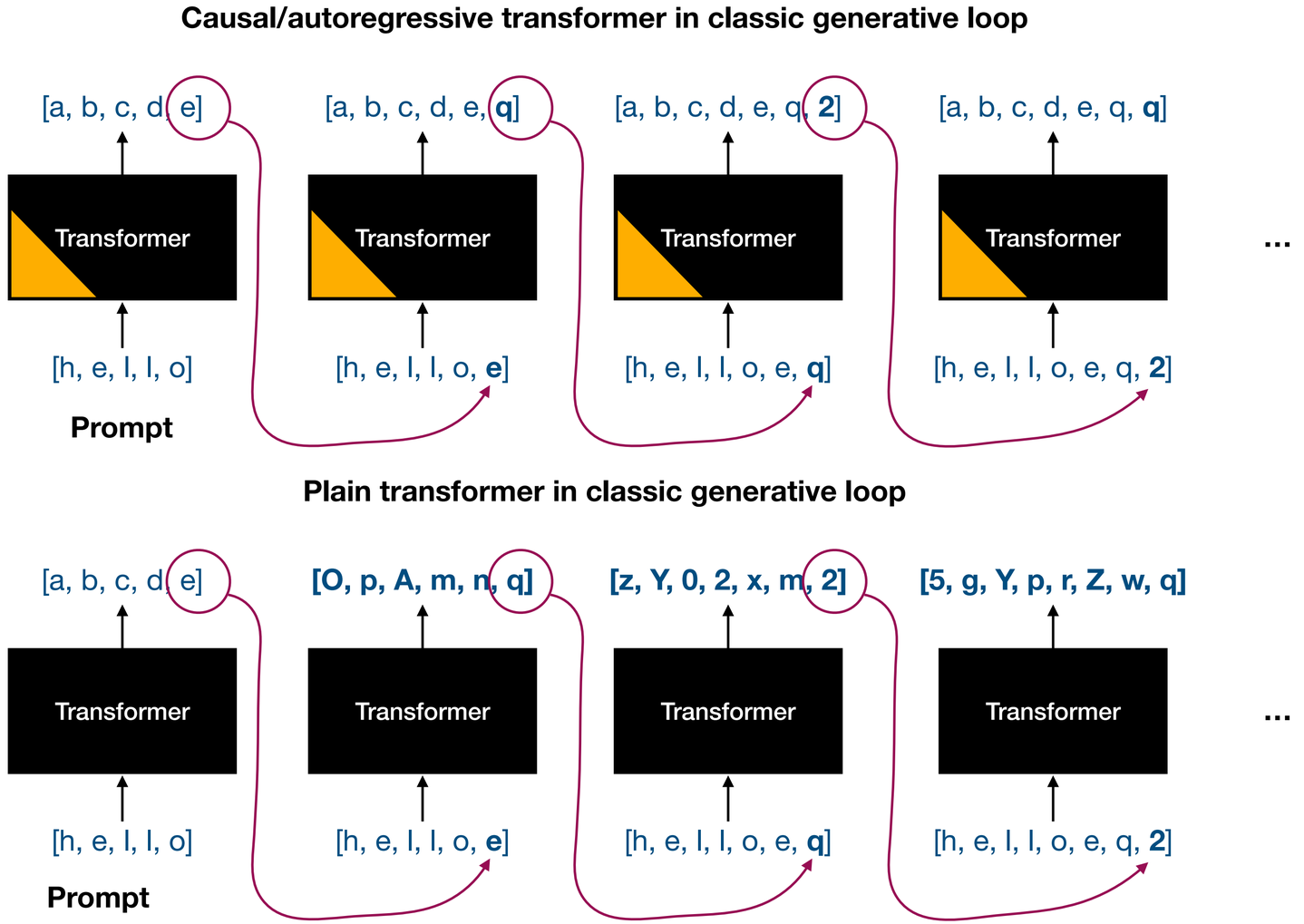


...

Probably works well, maybe even better - but very inefficient (redo entire computation for every prediction)

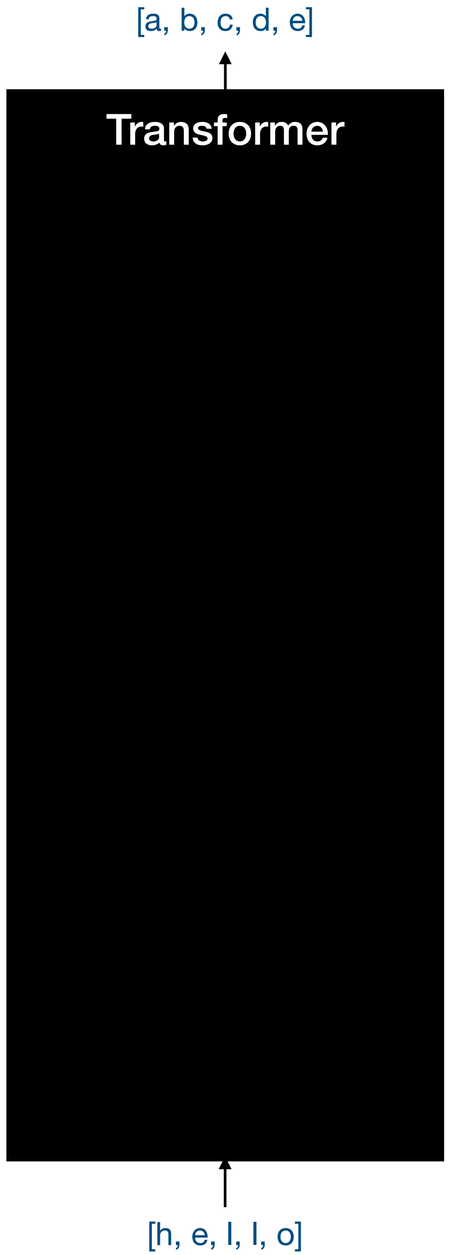
Note - model vs use



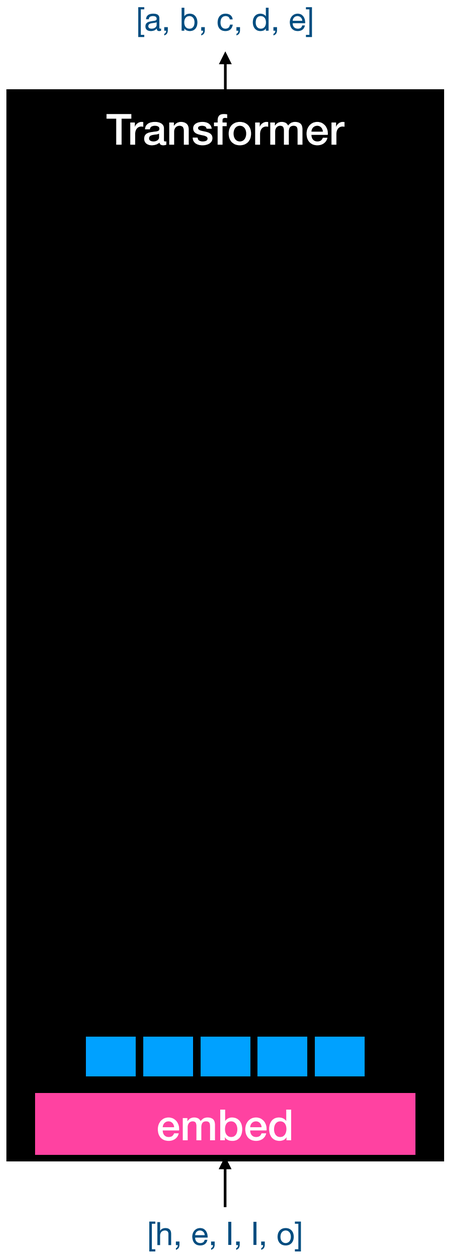


We will not focus on the “generative” case, or the causal restriction - just the transformer model itself

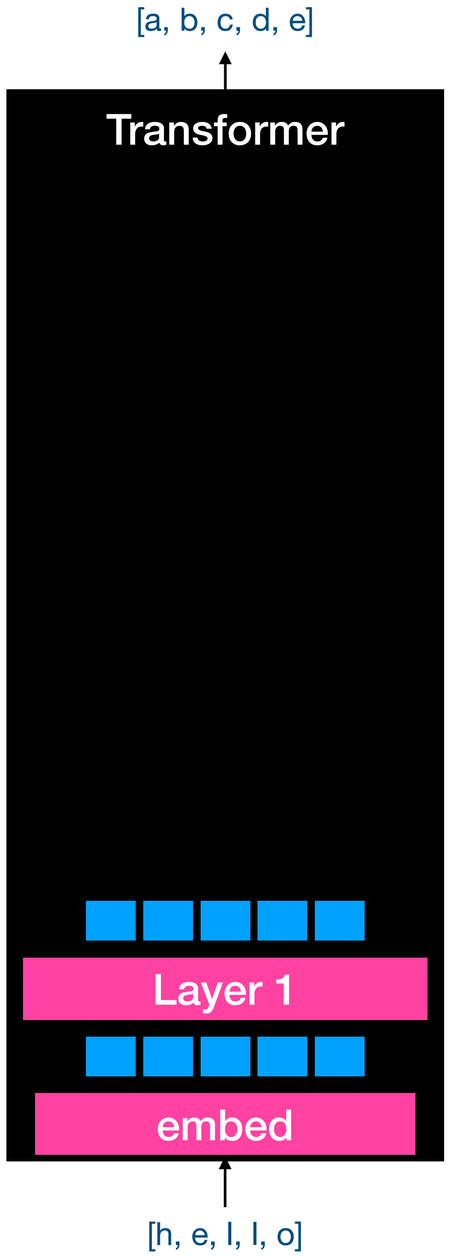
Transformer model



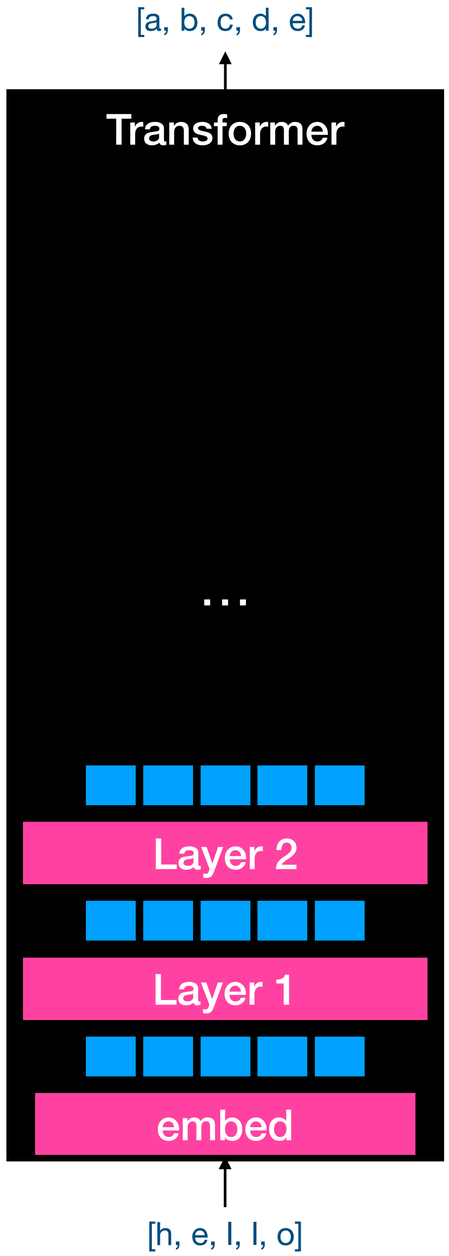
Transformer model



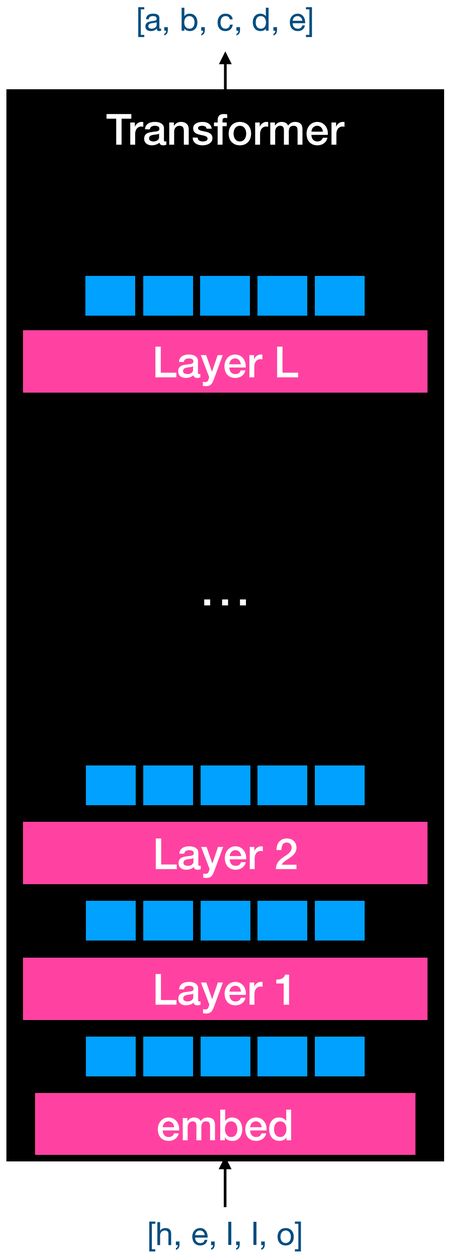
Transformer model



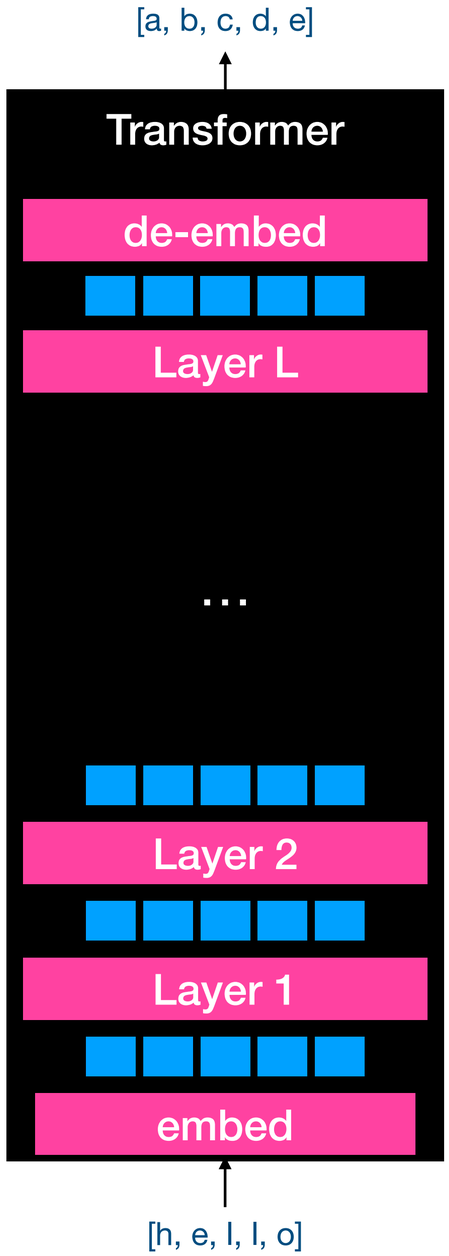
Transformer model



Transformer model

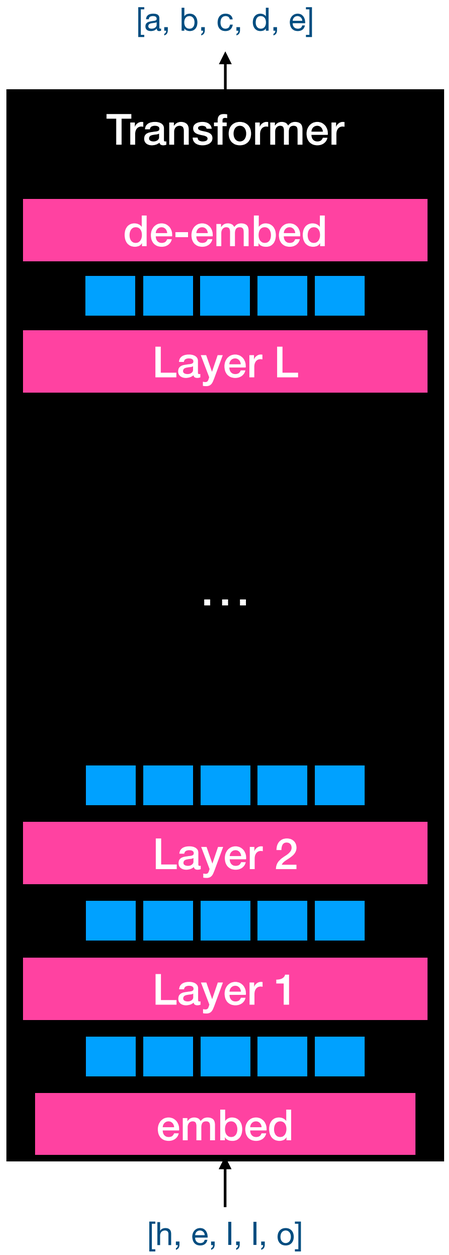


Transformer model



Transformer model

Goal: understand how transformers process input



Transformer model

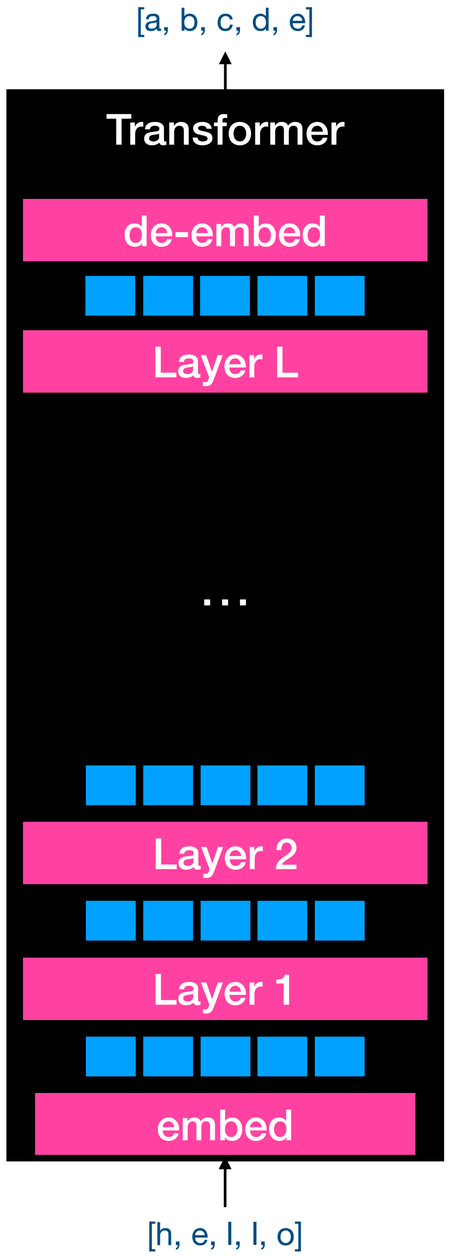
Goal: understand how transformers process input

Approach:

Internal embeddings are not meaningless vectors, they carry meaning. Describe with symbolic values

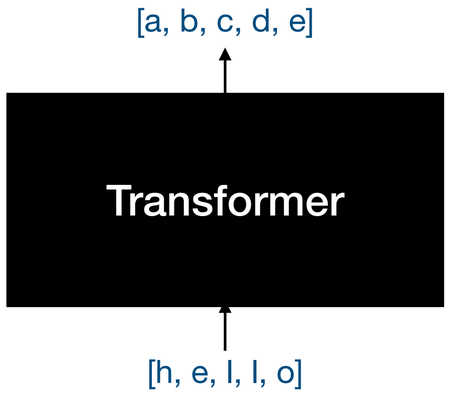
Layers are not meaningless manipulations, they perform meaningful operations on these values. Describe these operations

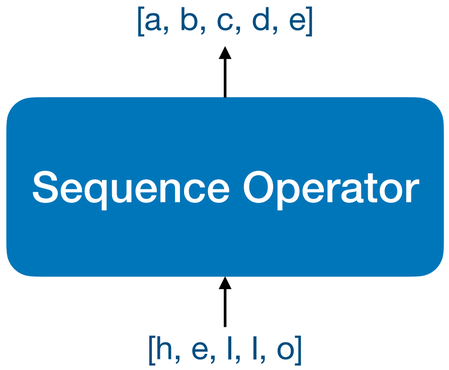
→ view transformer as executing a short program



RASP (Restricted Access Sequence Processing)

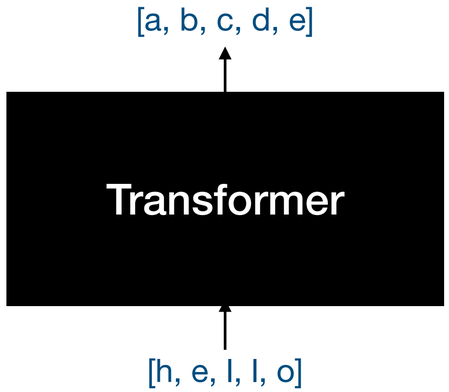
- State: Transformer architecture processes input sequences, with specific operations
- Goal: describe/understand these operations
- Result:
  - Abstract transformers as class of sequence-to-sequence functions, the “sequence operators” (**s-ops**)

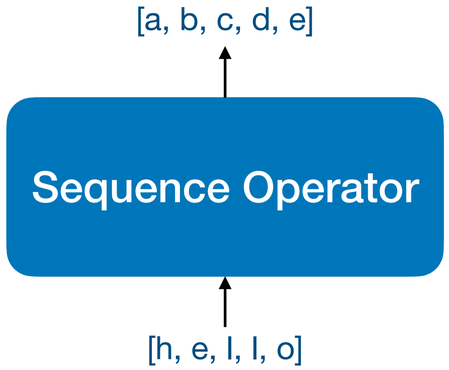




RASP (Restricted Access Sequence Processing)

- State: Transformer architecture processes input sequences, with specific operations
- Goal: describe/understand these operations
- Result:
  - Abstract transformers as class of sequence-to-sequence functions, the “sequence operators” (**s-ops**)
  - Define function space inductively with RASP:
    - base s-ops (program inputs), and
    - operations to create new s-ops (program lines)



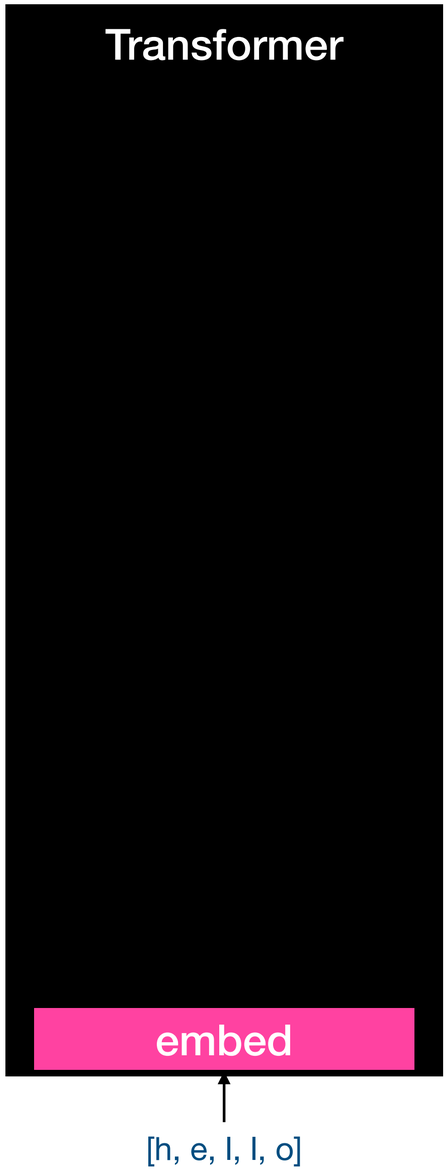


RASP s-ops and operations

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
   1. macOS, linux: ./install.sh
   2. windows: follow “windows instructions.txt”
3. run:
   1. macOS, linux: ./rasp.sh
   2. windows: python3 RASP_support/REPL.py

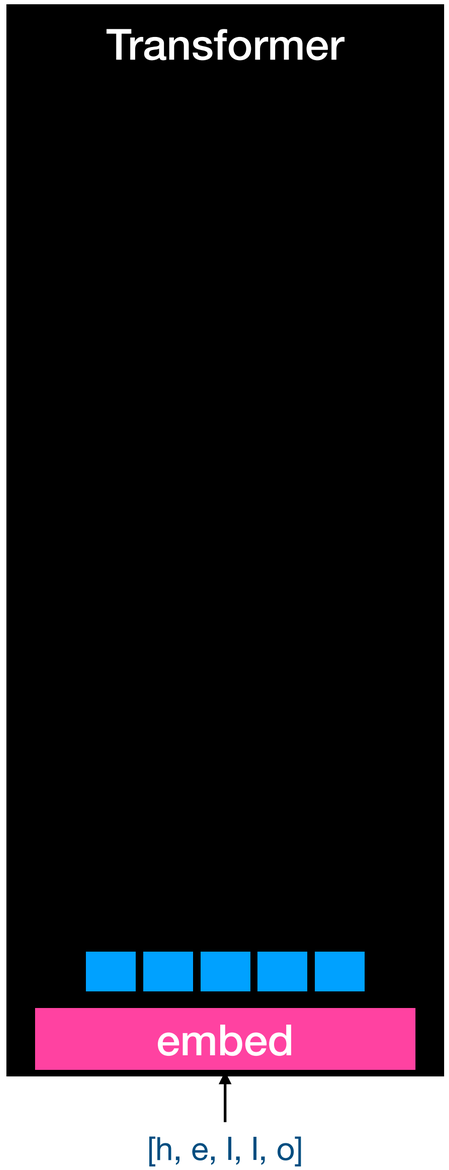


RASP s-ops and operations

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
    1. macOS, linux: ./install.sh
    2. windows: follow “windows instructions.txt”
3. run:
    1. macOS, linux: ./rasp.sh
    2. windows: python3 RASP_support/REPL.py

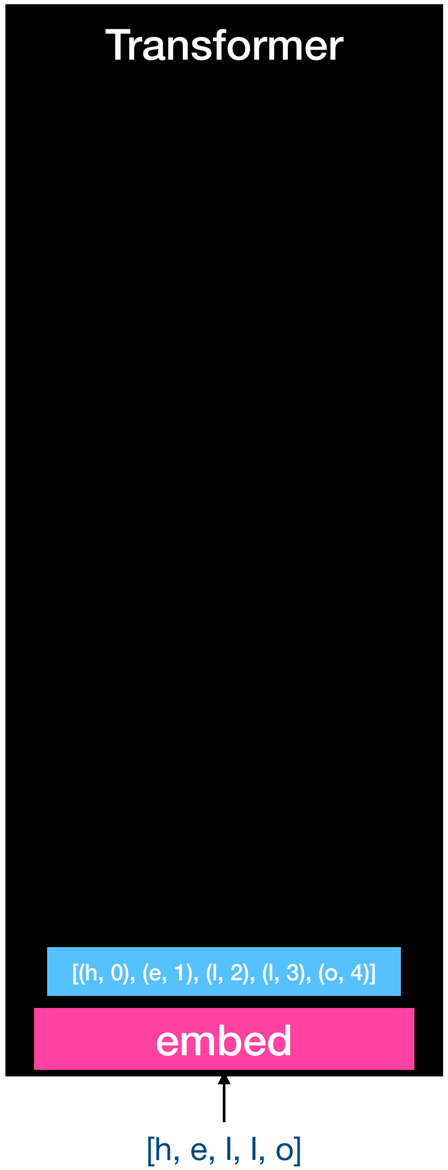


RASP s-ops and operations

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
   1. macOS, linux: ./install.sh
   2. windows: follow “windows instructions.txt”
3. run:
   1. macOS, linux: ./rasp.sh
   2. windows: python3 RASP_support/REPL.py

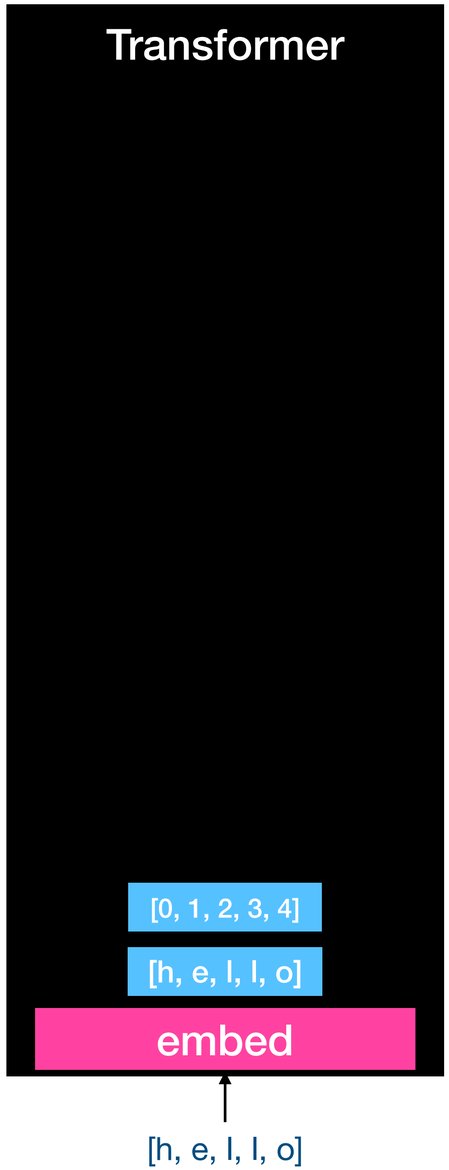


RASP s-ops and operations

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
   1. macOS, linux: ./install.sh
   2. windows: follow “windows instructions.txt”
3. run:
   1. macOS, linux: ./rasp.sh
   2. windows: python3 RASP_support/REPL.py



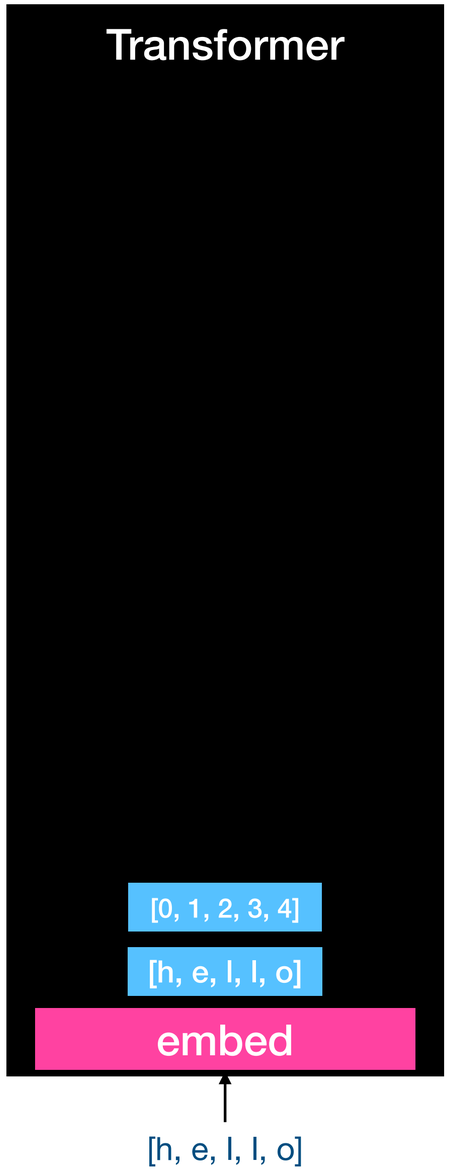
RASP s-ops and operations

- Base s-ops:
  - **tokens**([a, b, c]) = [a, b, c]
  - **indices**([a, b, c]) = [0, 1, 2]

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
   1. macOS, linux: ./install.sh
   2. windows: follow “windows instructions.txt”
3. run:
   1. macOS, linux: ./rasp.sh
   2. windows: python3 RASP_support/REPL.py



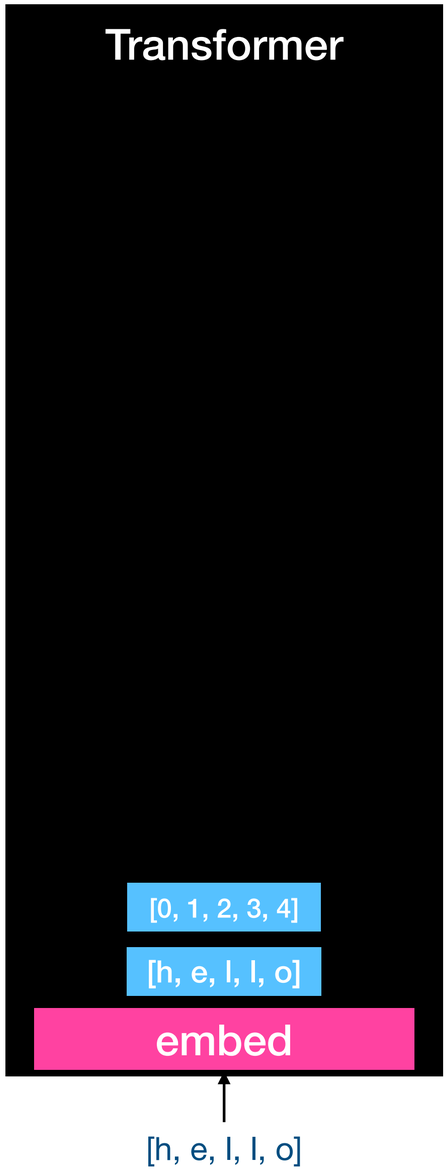
RASP s-ops and operations

- Base s-ops:
  - **tokens**([a, b, c]) = [a, b, c]
  - **indices**([a, b, c]) = [0, 1, 2]
  - **0**([a, b, c]) = [0, 0, 0]
  - (all other constants)

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
   1. macOS, linux: ./install.sh
   2. windows: follow “windows instructions.txt”
3. run:
   1. macOS, linux: ./rasp.sh
   2. windows: python3 RASP_support/REPL.py



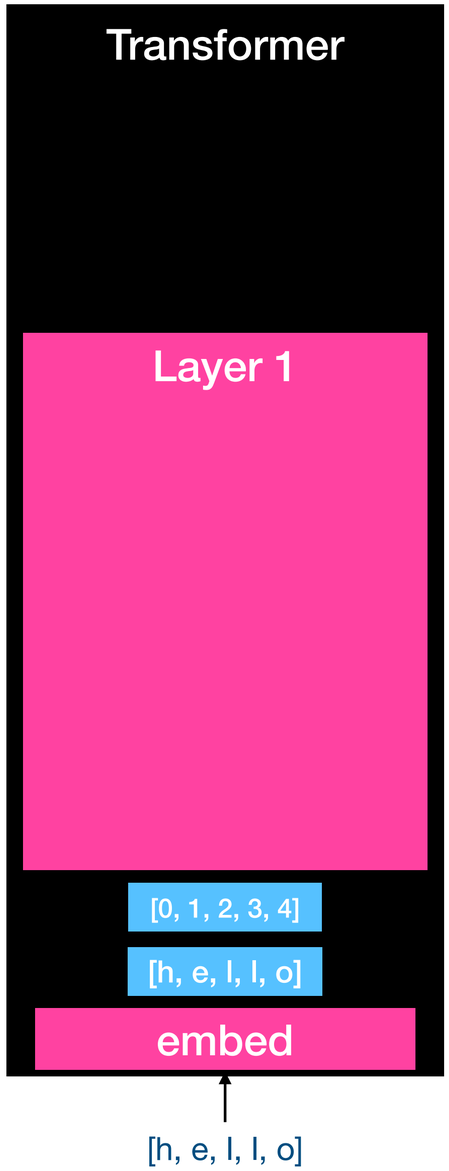
RASP s-ops and operations

- Base s-ops:
  - **tokens**([a, b, c]) = [a, b, c]
  - **indices**([a, b, c]) = [0, 1, 2]
  - **0**([a, b, c]) = [0, 0, 0]
  - (all other constants)

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
   1. macOS, linux: ./install.sh
   2. windows: follow “windows instructions.txt”
3. run:
   1. macOS, linux: ./rasp.sh
   2. windows: python3 RASP_support/REPL.py



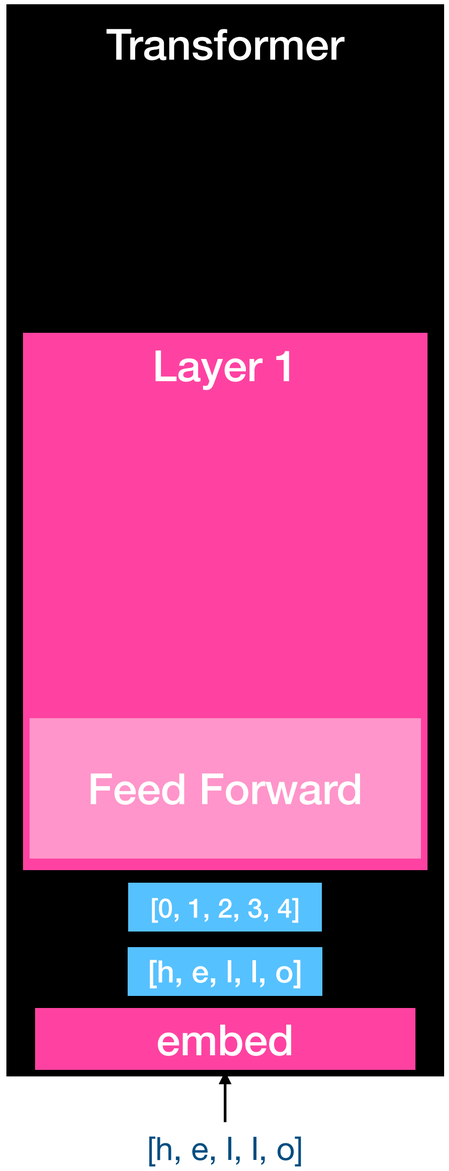
RASP s-ops and operations

- Base s-ops:
  - **tokens**([a, b, c]) = [a, b, c]
  - **indices**([a, b, c]) = [0, 1, 2]
  - **0**([a, b, c]) = [0, 0, 0]
  - (all other constants)

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
   1. macOS, linux: ./install.sh
   2. windows: follow “windows instructions.txt”
3. run:
   1. macOS, linux: ./rasp.sh
   2. windows: python3 RASP_support/REPL.py



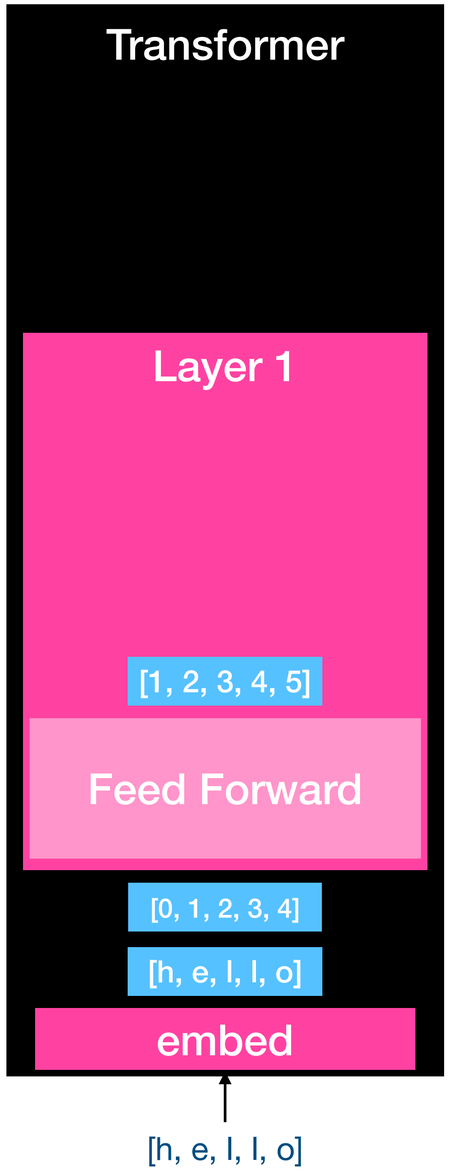
RASP s-ops and operations

- Base s-ops:
  - **tokens**([a, b, c]) = [a, b, c]
  - **indices**([a, b, c]) = [0, 1, 2]
  - **0**([a, b, c]) = [0, 0, 0]
  - (all other constants)

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
   1. macOS, linux: ./install.sh
   2. windows: follow “windows instructions.txt”
3. run:
   1. macOS, linux: ./rasp.sh
   2. windows: python3 RASP_support/REPL.py



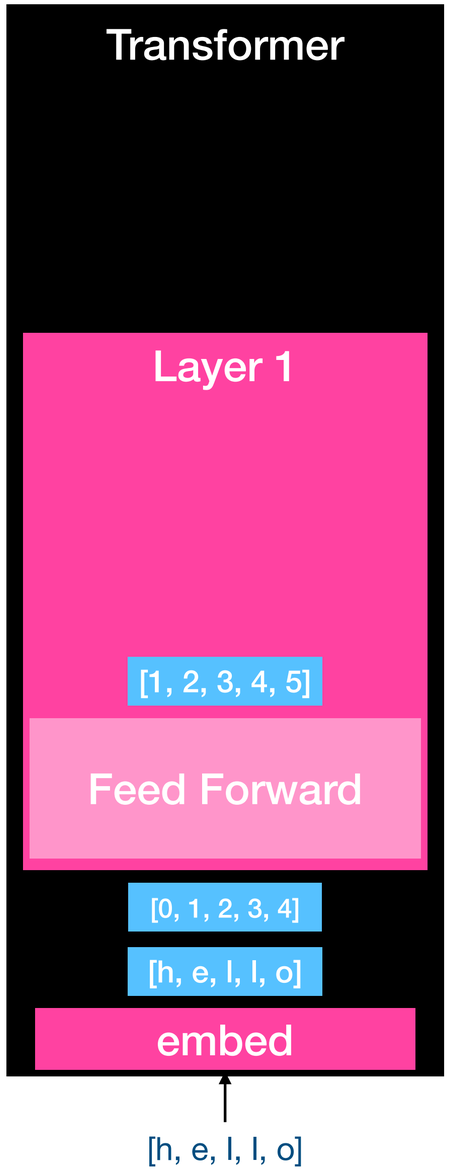
RASP s-ops and operations

- Base s-ops:
  - **tokens**([a, b, c]) = [a, b, c]
  - **indices**([a, b, c]) = [0, 1, 2]
  - **0**([a, b, c]) = [0, 0, 0]
  - (all other constants)
- Elementwise operations:
  - **(indices+1)**([a, b, c]) = [1, 2, 3]

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
   1. macOS, linux: ./install.sh
   2. windows: follow “windows instructions.txt”
3. run:
   1. macOS, linux: ./rasp.sh
   2. windows: python3 RASP_support/REPL.py



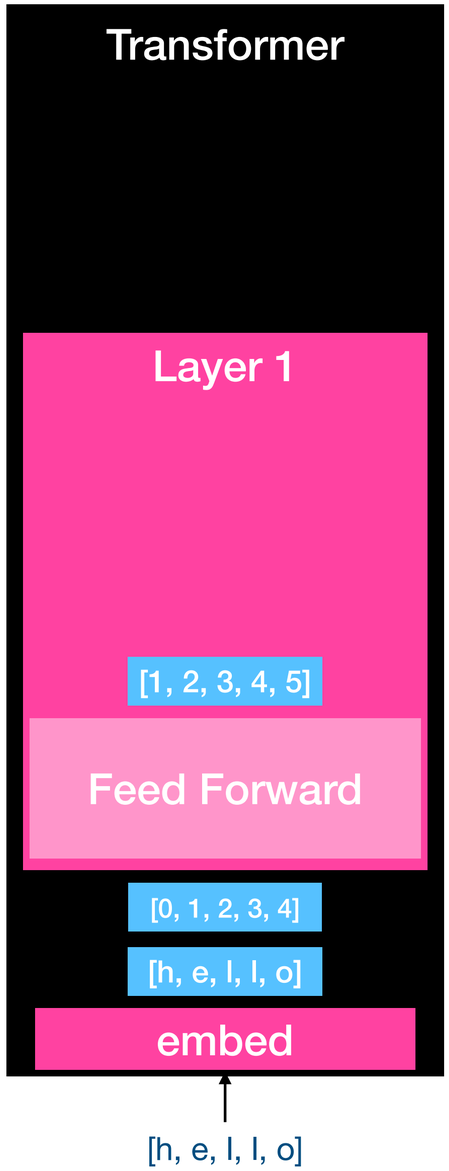
RASP s-ops and operations

- Base s-ops:
  - **tokens**([a, b, c]) = [a, b, c]
  - **indices**([a, b, c]) = [0, 1, 2]
  - **0**([a, b, c]) = [0, 0, 0]
  - (all other constants)
- Elementwise operations:
  - **(indices+1)**([a, b, c]) = [1, 2, 3]
  - **(tokens==“b”)**([a, b, c]) = [False, True, False]

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
   1. macOS, linux: ./install.sh
   2. windows: follow “windows instructions.txt”
3. run:
   1. macOS, linux: ./rasp.sh
   2. windows: python3 RASP_support/REPL.py



RASP s-ops and operations

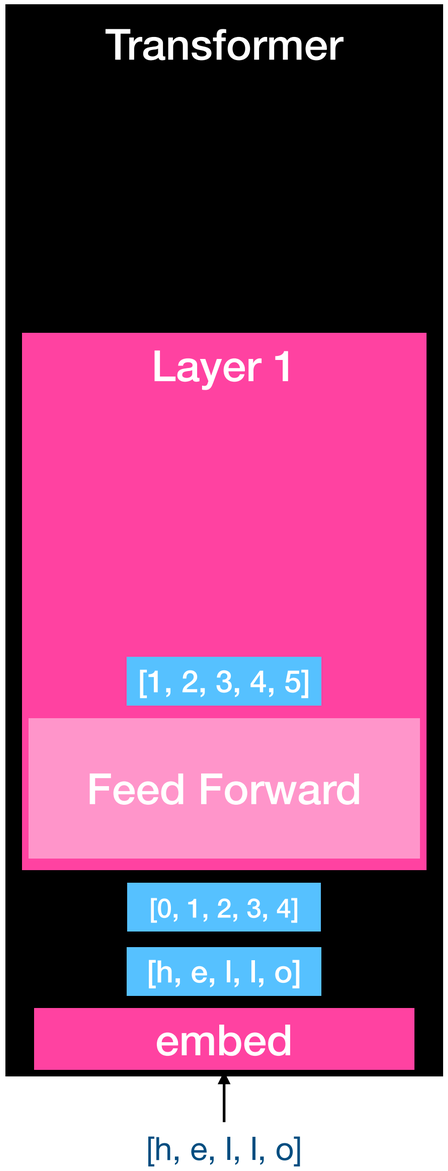
- Base s-ops:
  - **tokens**([a, b, c]) = [a, b, c]
  - **indices**([a, b, c]) = [0, 1, 2]
  - **0**([a, b, c]) = [0, 0, 0]
  - (all other constants)
- Elementwise operations:
  - **(indices+1)**([a, b, c]) = [1, 2, 3]
  - **(tokens=="b")**([a, b, c]) = [False, True, False]
  - **(3 if tokens=="b" else 0)**([a, b, c]) = [0, 3, 0]
  - (all other basic char/num/bool operations)

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
   1. macOS, linux: ./install.sh
   2. windows: follow "windows instructions.txt"
3. run:
   1. macOS, linux: ./rasp.sh
   2. windows: python3 RASP_support/REPL.py

Multilayer feedforward networks are universal approximators (Hornik et al, 1989)



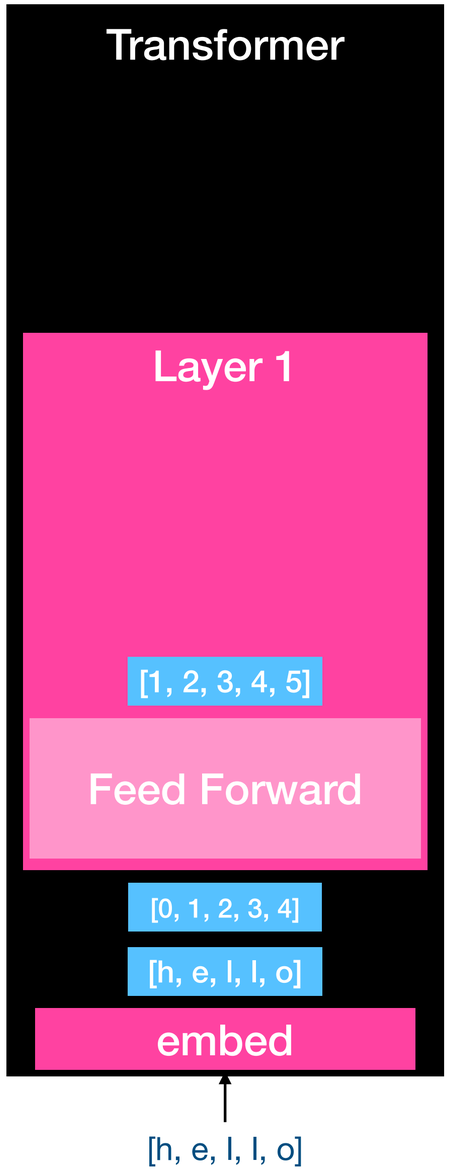
RASP s-ops and operations

- Base s-ops:
  - **tokens**([a, b, c]) = [a, b, c]
  - **indices**([a, b, c]) = [0, 1, 2]
  - **0**([a, b, c]) = [0, 0, 0]
  - (all other constants)
- Elementwise operations:
  - **(indices+1)**([a, b, c]) = [1, 2, 3]
  - **(tokens==“b”)**([a, b, c]) = [False, True, False]
  - **(3 if tokens==“b” else 0)**([a, b, c]) = [0, 3, 0]
  - (all other basic char/num/bool operations)

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
   1. macOS, linux: ./install.sh
   2. windows: follow “windows instructions.txt”
3. run:
   1. macOS, linux: ./rasp.sh
   2. windows: python3 RASP_support/REPL.py



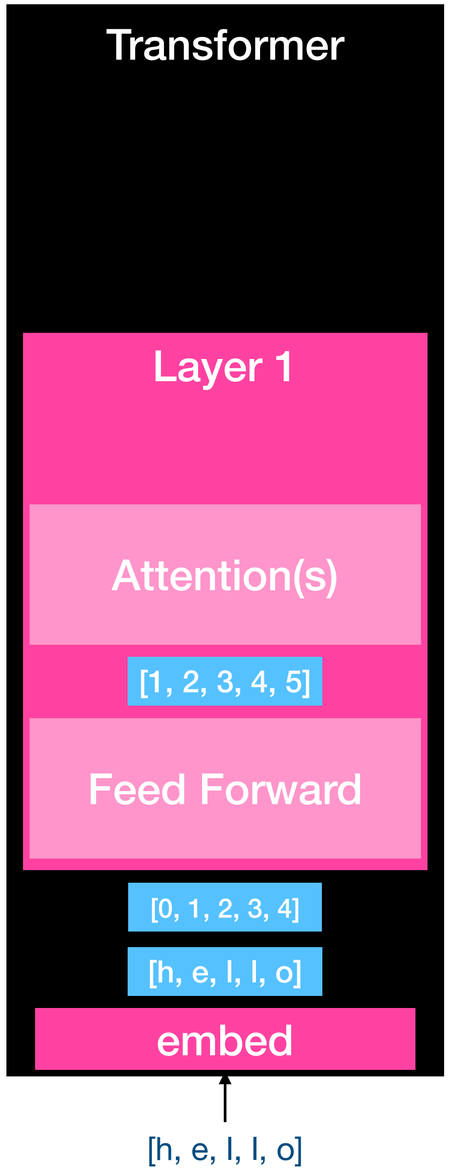
RASP s-ops and operations

- Base s-ops:
  - **tokens**([a, b, c]) = [a, b, c]
  - **indices**([a, b, c]) = [0, 1, 2]
  - **0**([a, b, c]) = [0, 0, 0]
  - (all other constants)
- Elementwise operations:
  - **(indices+1)**([a, b, c]) = [1, 2, 3]
  - **(tokens==“b”)**([a, b, c]) = [False, True, False]
  - **(3 if tokens==“b” else 0)**([a, b, c]) = [0, 3, 0]
  - (all other basic char/num/bool operations)

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
   1. macOS, linux: ./install.sh
   2. windows: follow “windows instructions.txt”
3. run:
   1. macOS, linux: ./rasp.sh
   2. windows: python3 RASP_support/REPL.py



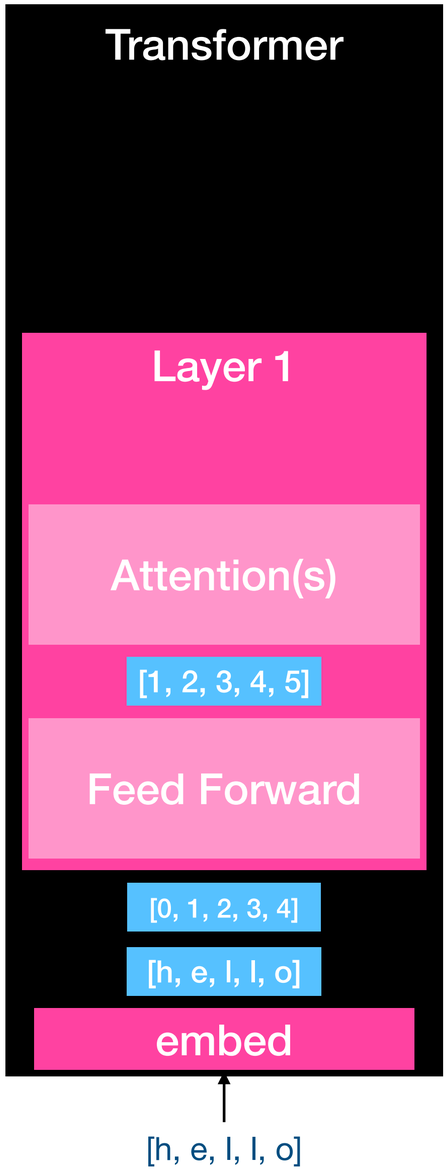
RASP s-ops and operations

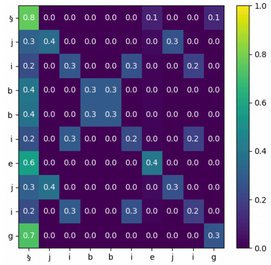
- Base s-ops:
  - **tokens**([a, b, c]) = [a, b, c]
  - **indices**([a, b, c]) = [0, 1, 2]
  - **0**([a, b, c]) = [0, 0, 0]
  - (all other constants)
- Elementwise operations:
  - **(indices+1)**([a, b, c]) = [1, 2, 3]
  - **(tokens=="b")**([a, b, c]) = [False, True, False]
  - **(3 if tokens=="b" else 0)**([a, b, c]) = [0, 3, 0]
  - (all other basic char/num/bool operations)

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
   1. macOS, linux: ./install.sh
   2. windows: follow "windows instructions.txt"
3. run:
   1. macOS, linux: ./rasp.sh
   2. windows: python3 RASP_support/REPL.py





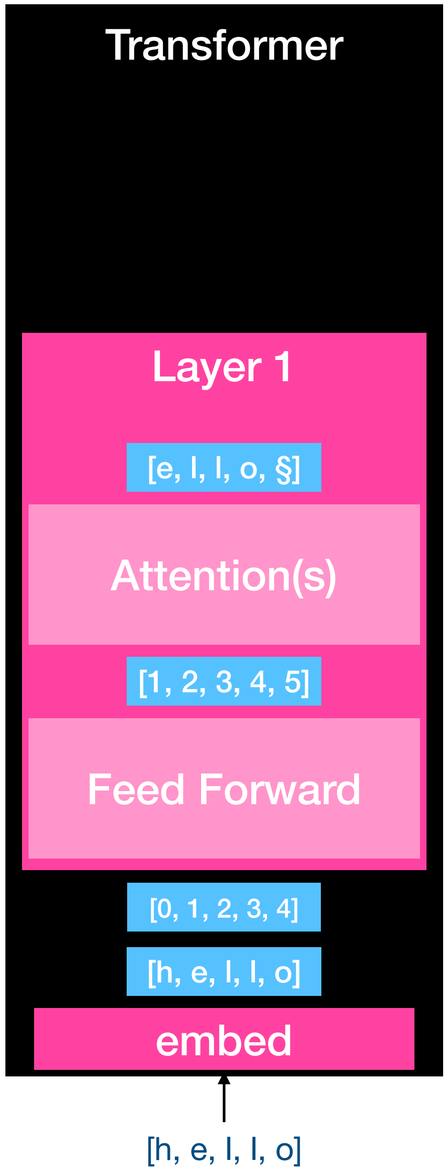
RASP s-ops and operations

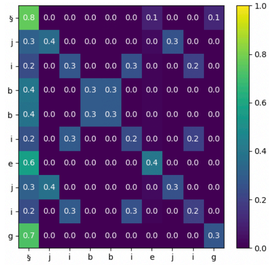
- Base s-ops:
  - **tokens**([a, b, c]) = [a, b, c]
  - **indices**([a, b, c]) = [0, 1, 2]
  - **0**([a, b, c]) = [0, 0, 0]
  - (all other constants)
- Elementwise operations:
  - **(indices+1)**([a, b, c]) = [1, 2, 3]
  - **(tokens=="b")**([a, b, c]) = [False, True, False]
  - **(3 if tokens=="b" else 0)**([a, b, c]) = [0, 3, 0]
  - (all other basic char/num/bool operations)

Follow along:

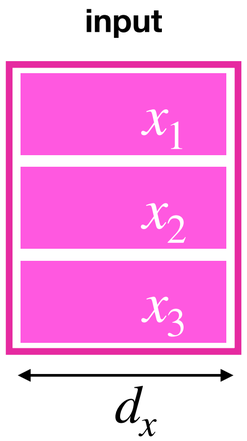
github.com/tech-srl/RASP

1. clone
2. set up:
   1. macOS, linux: ./install.sh
   2. windows: follow "windows instructions.txt"
3. run:
   1. macOS, linux: ./rasp.sh
   2. windows: python3 RASP_support/REPL.py

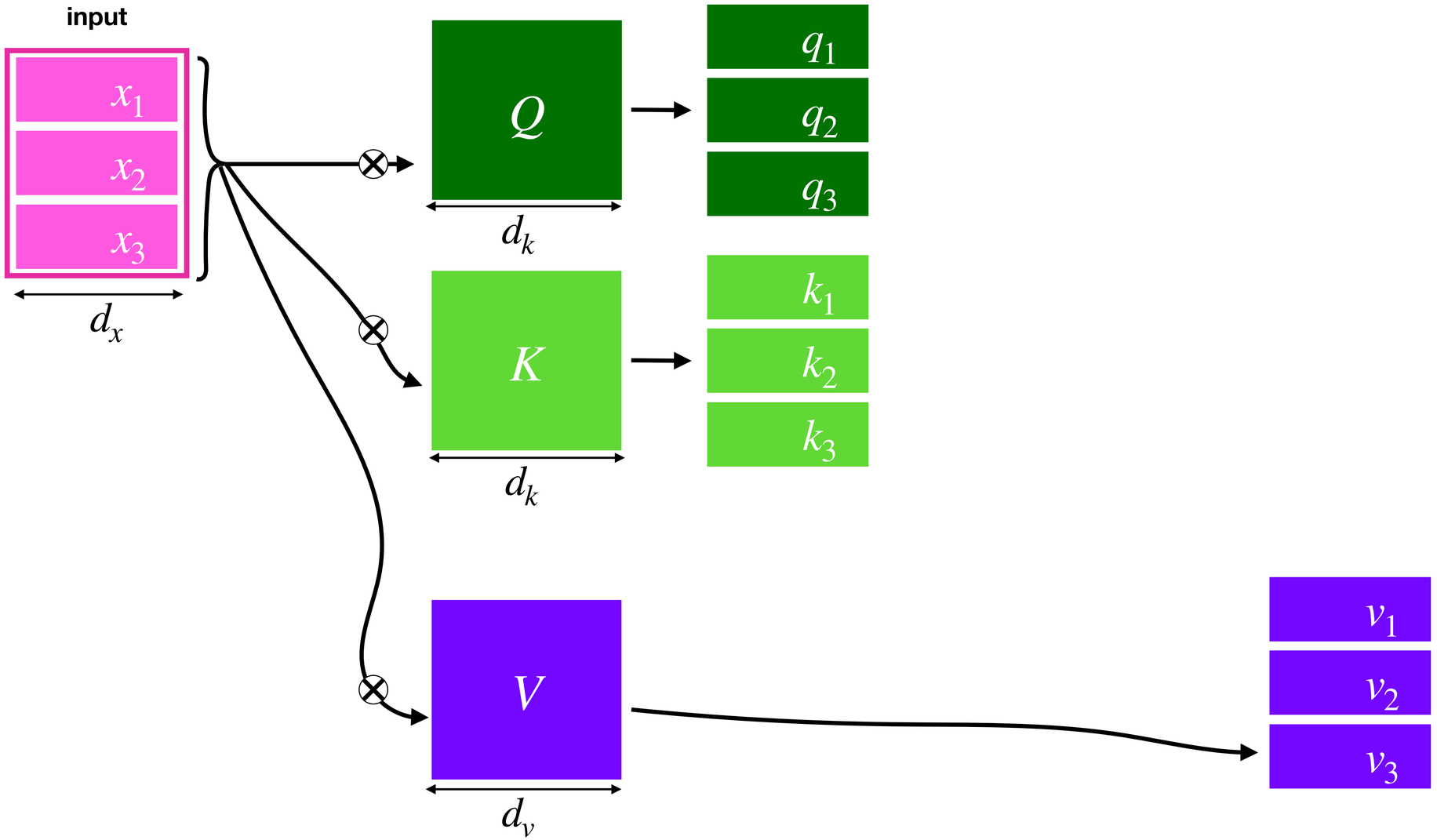




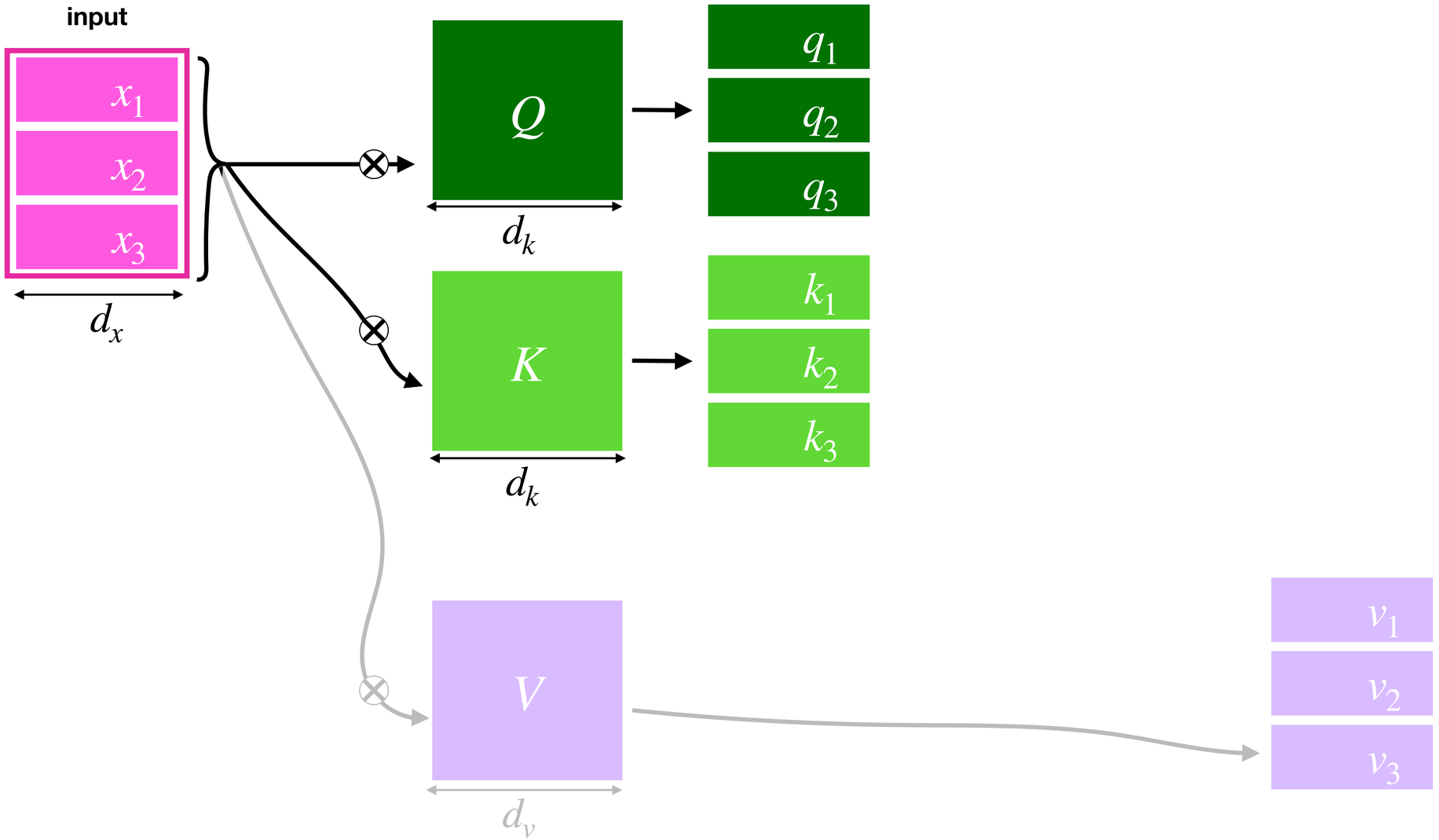
Background - Self Attention (Single Head)



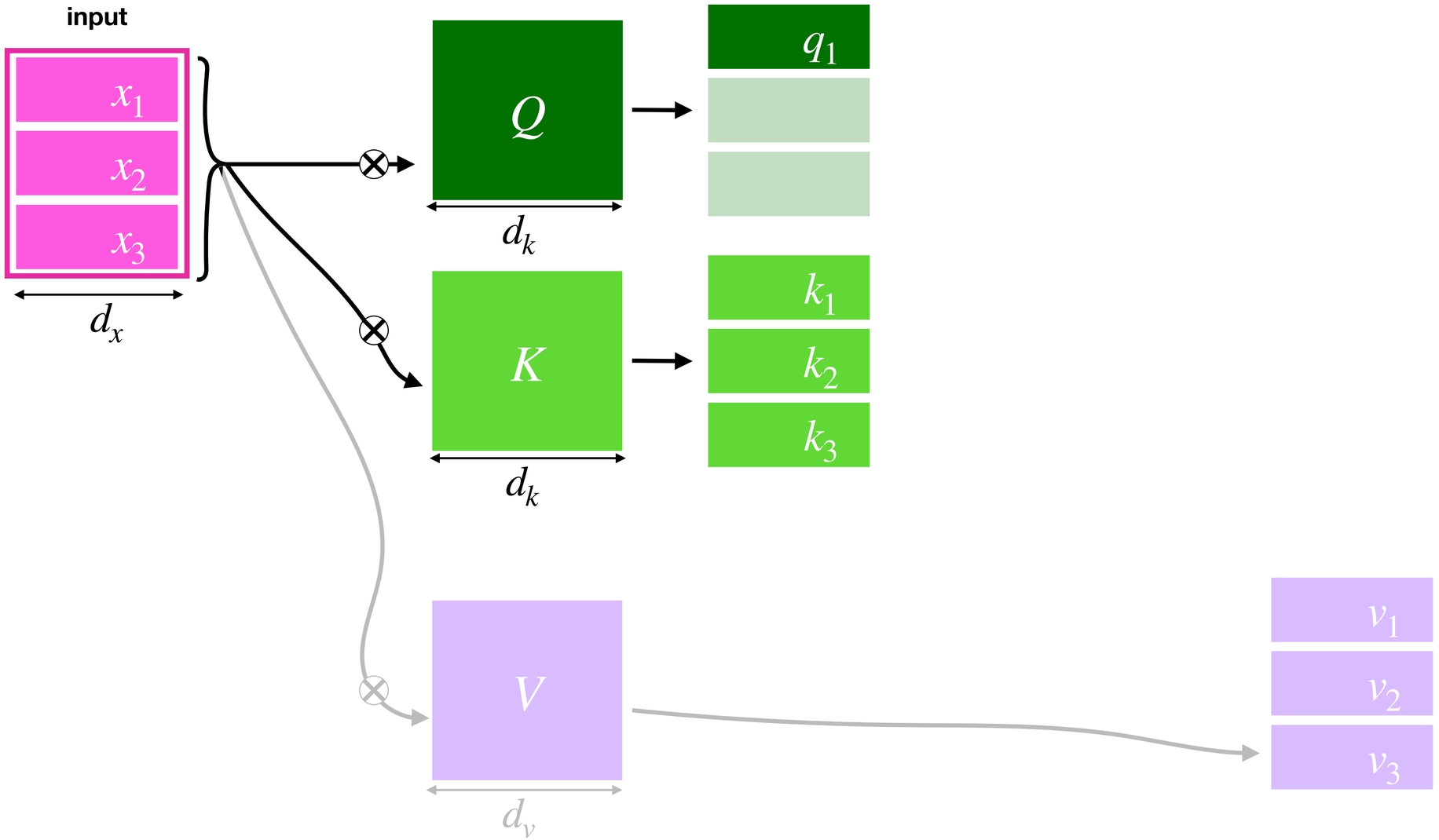
Background - Self Attention (Single Head)



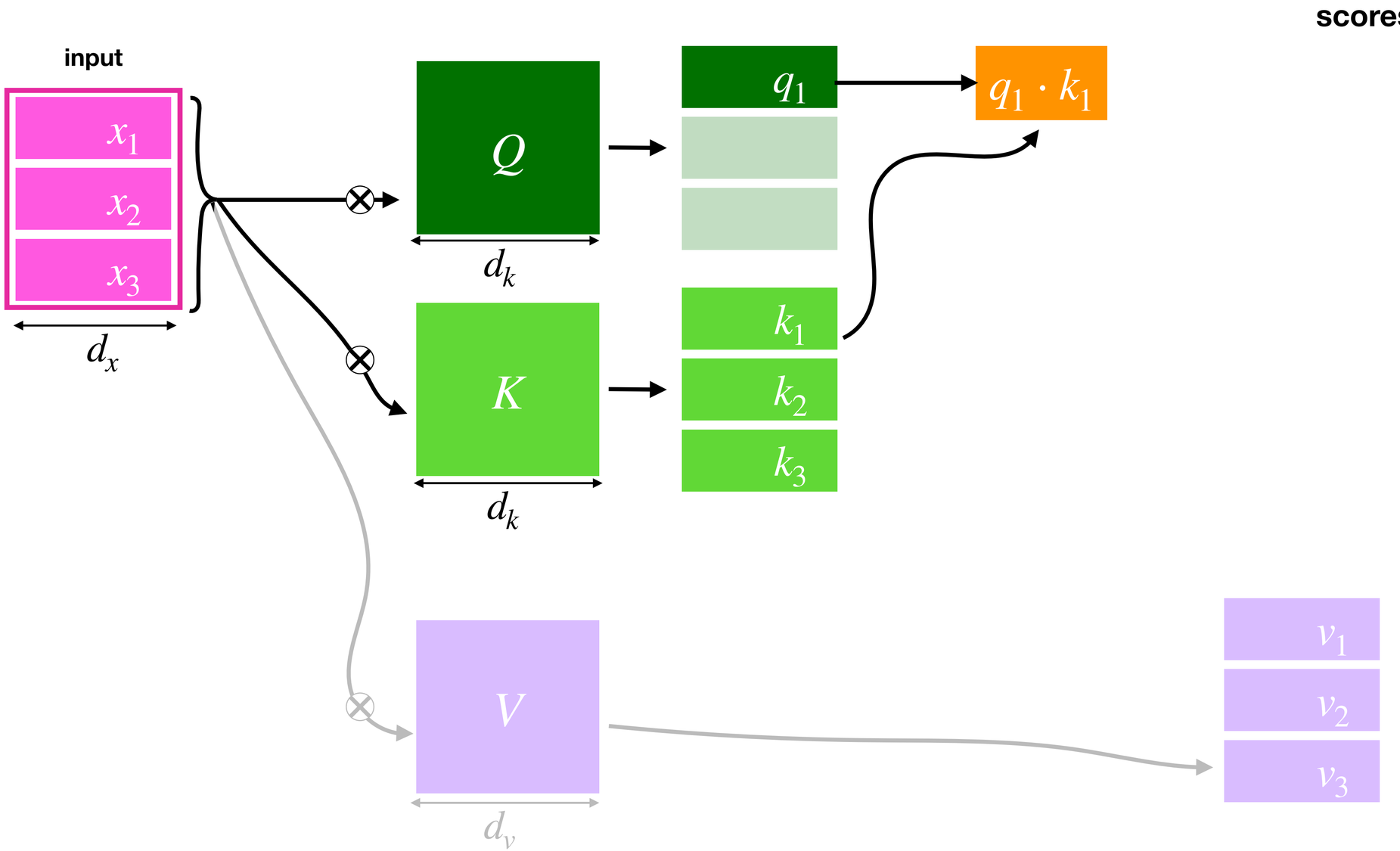
Background - Self Attention (Single Head)



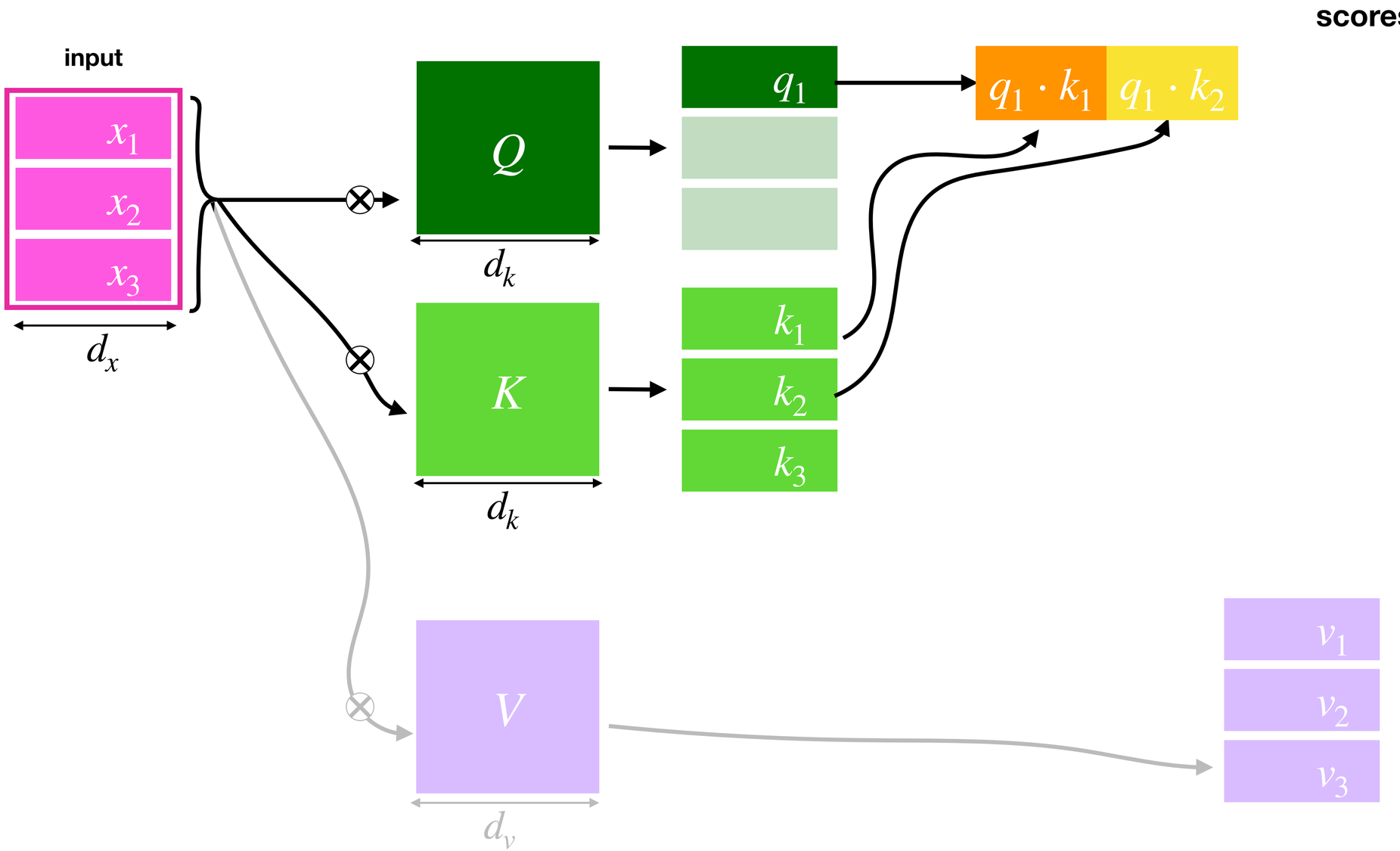
Background - Self Attention (Single Head)



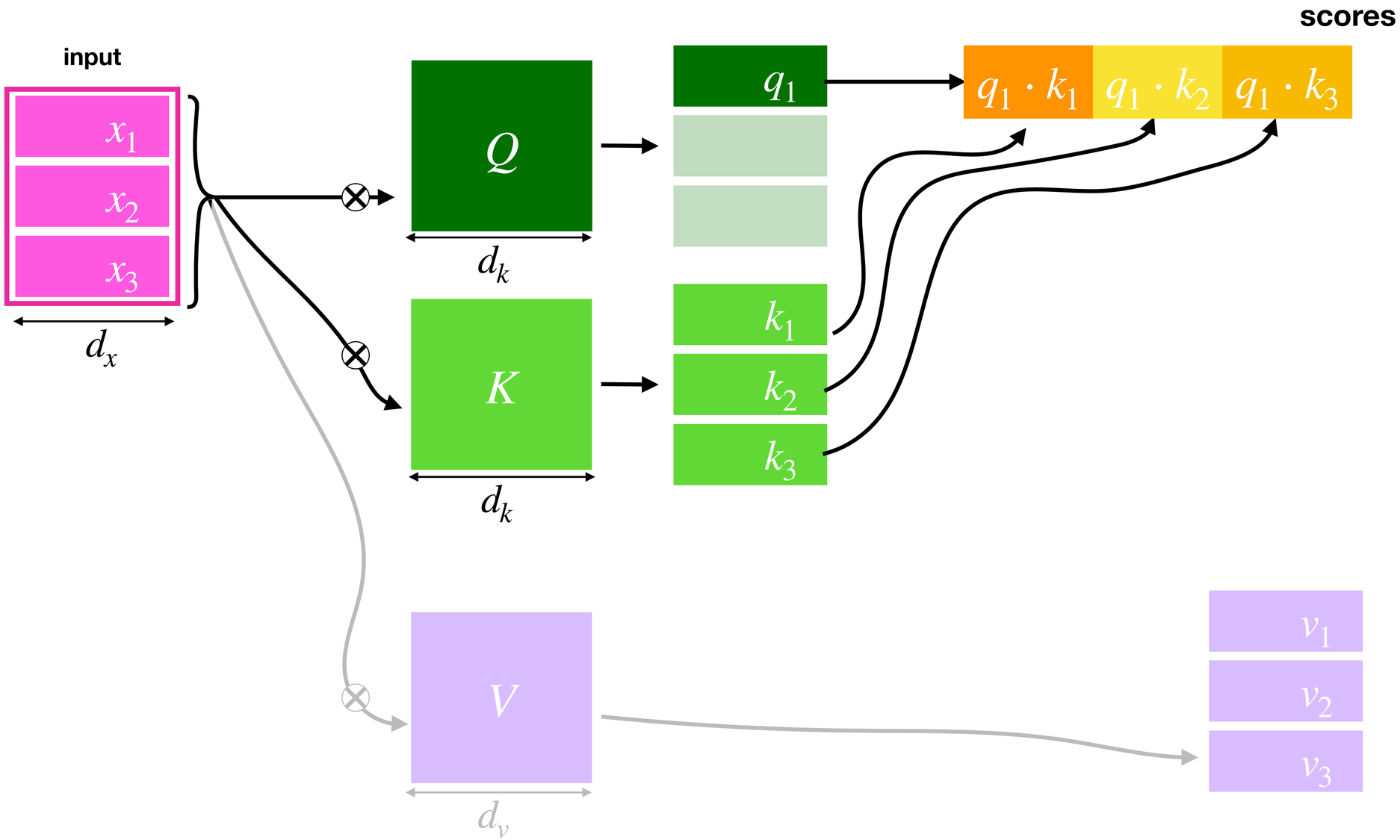
Background - Self Attention (Single Head)



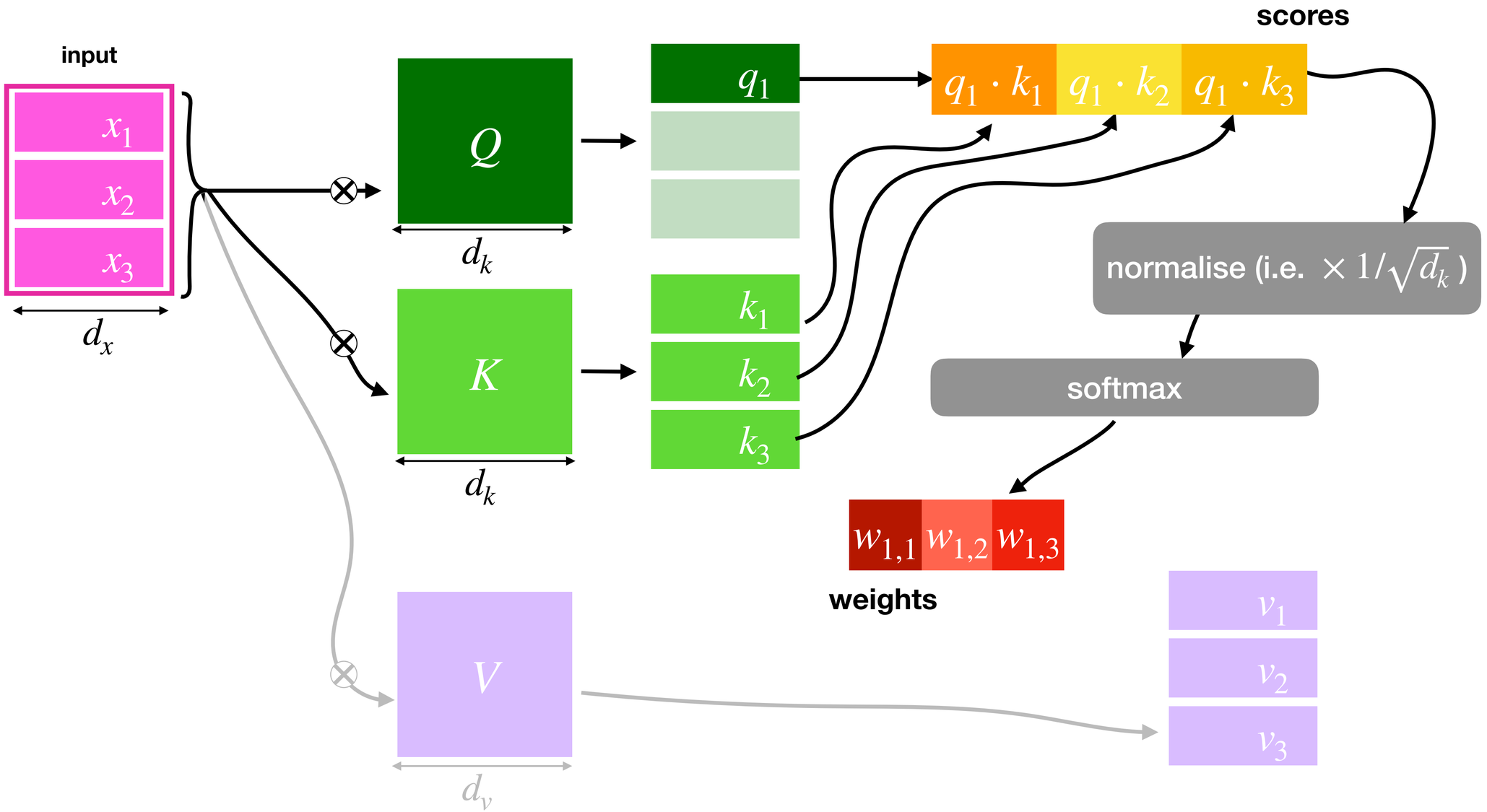
Background - Self Attention (Single Head)



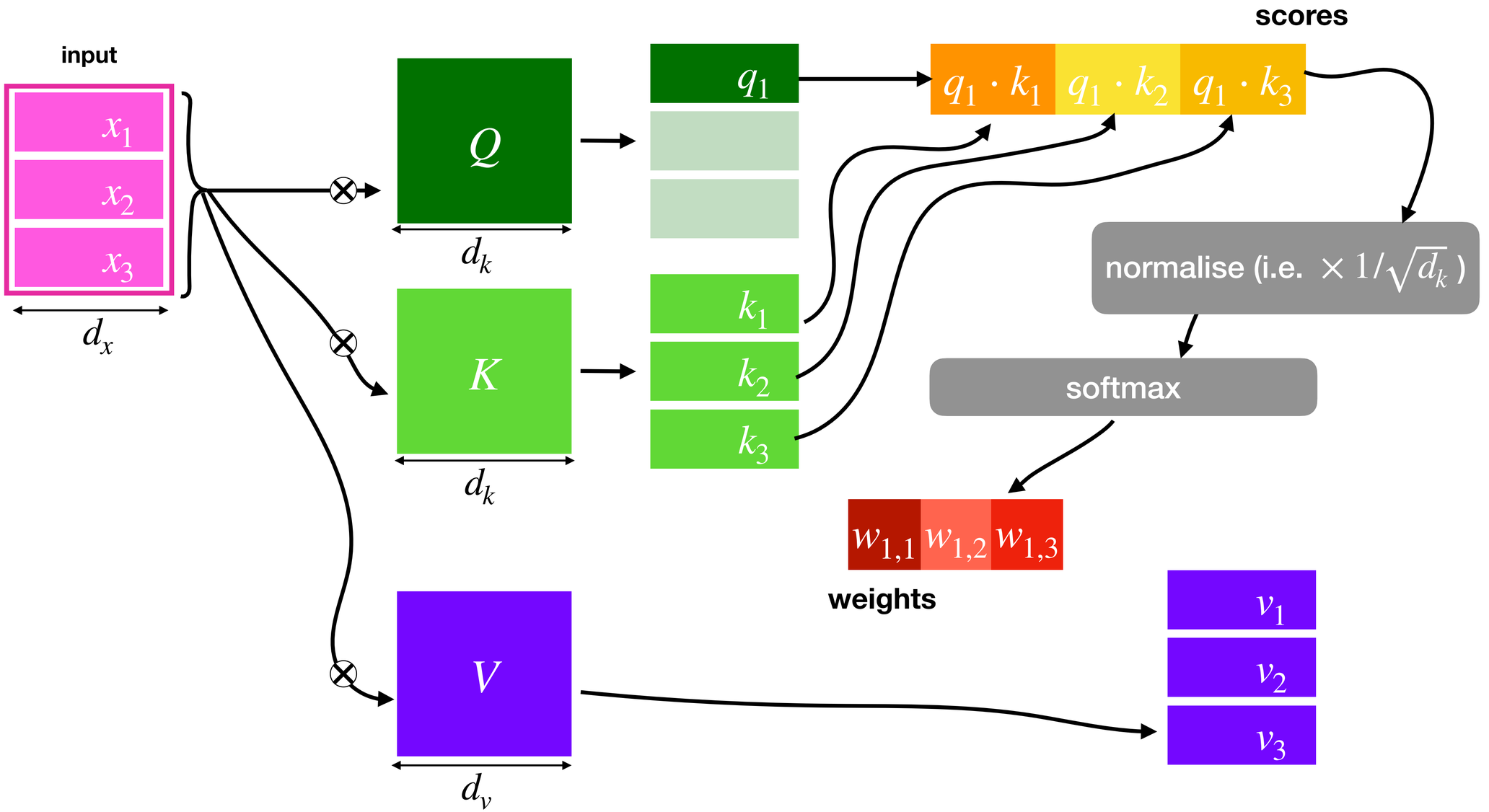
Background - Self Attention (Single Head)



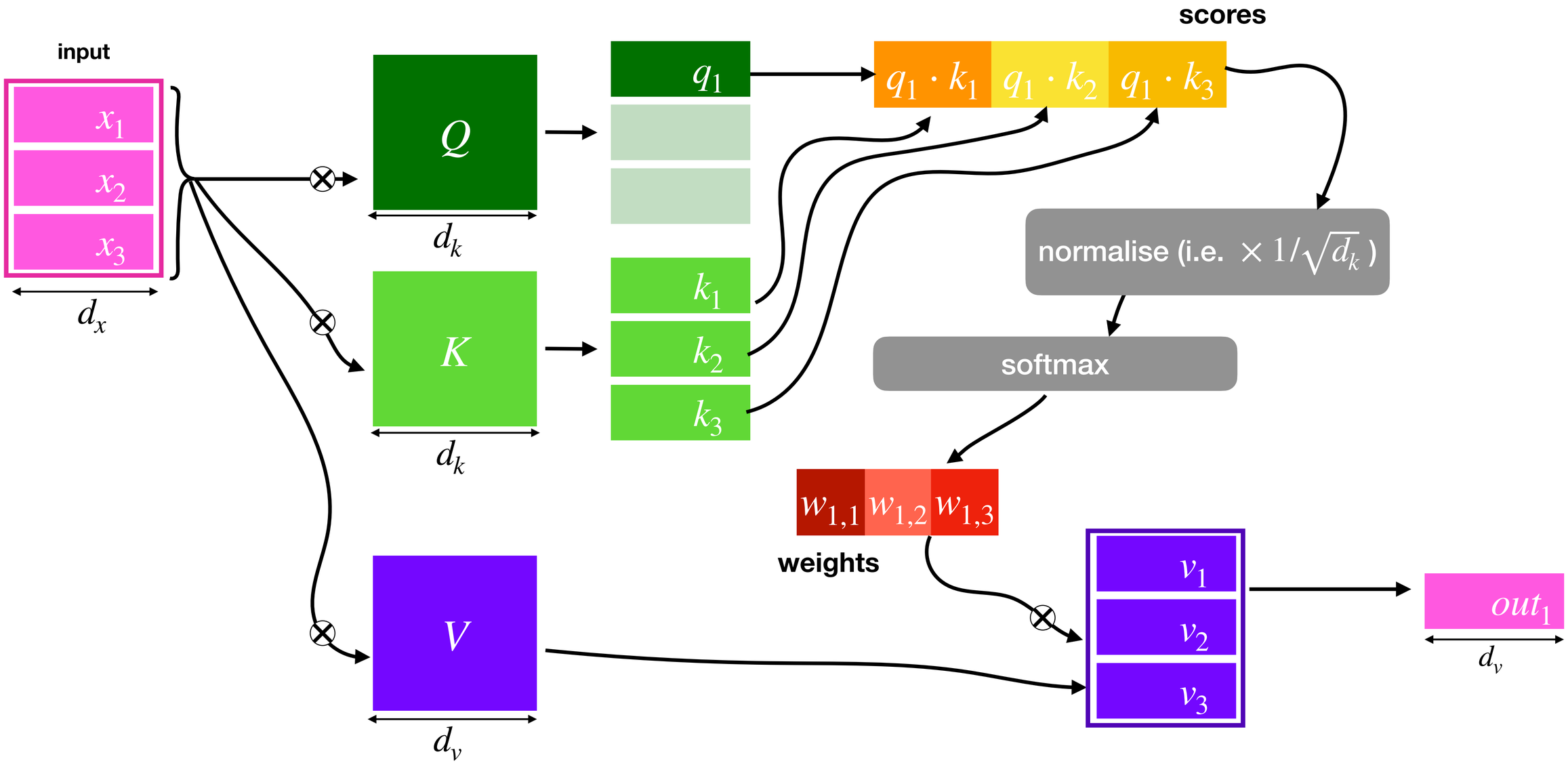
Background - Self Attention (Single Head)



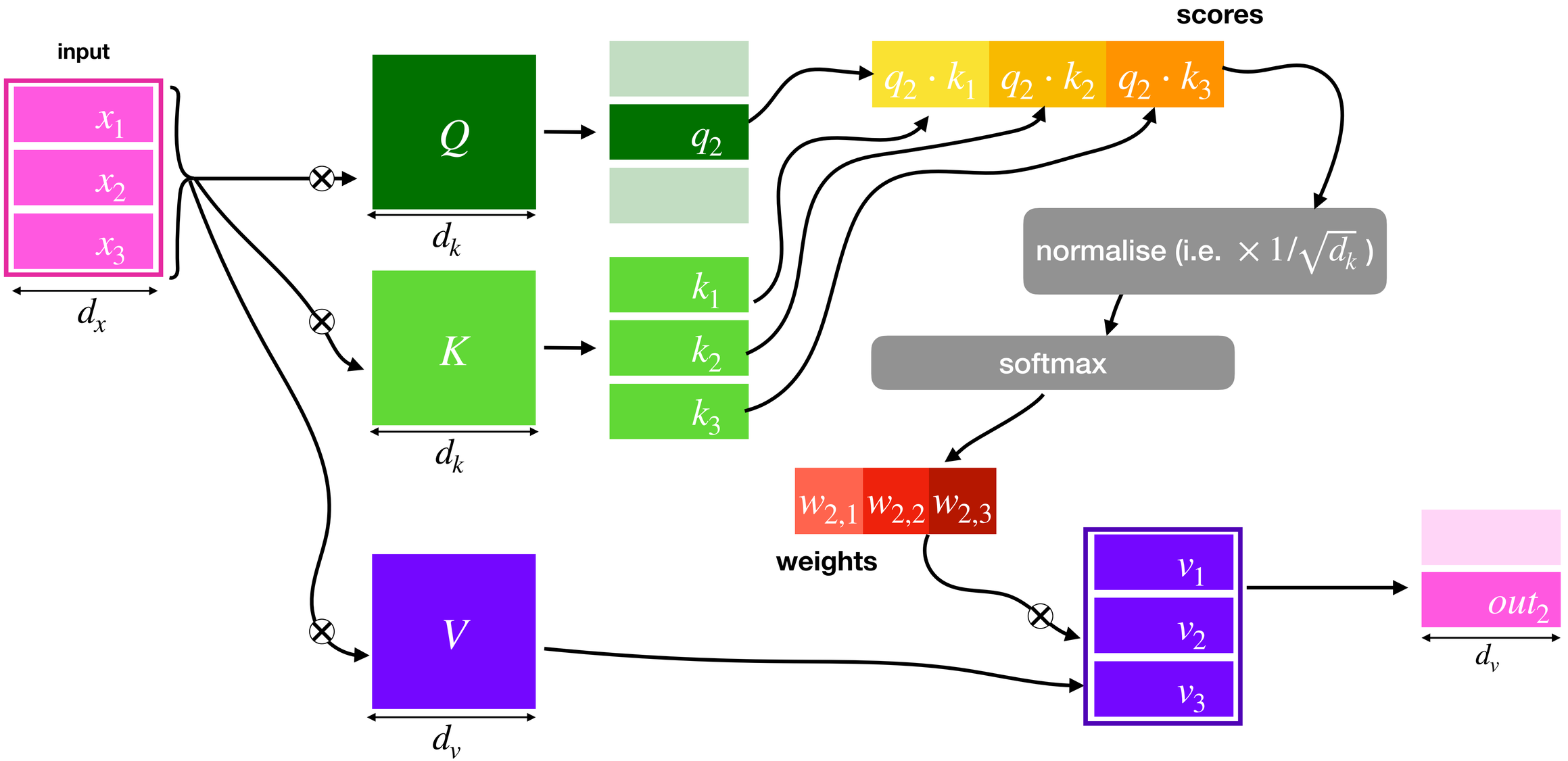
Background - Self Attention (Single Head)



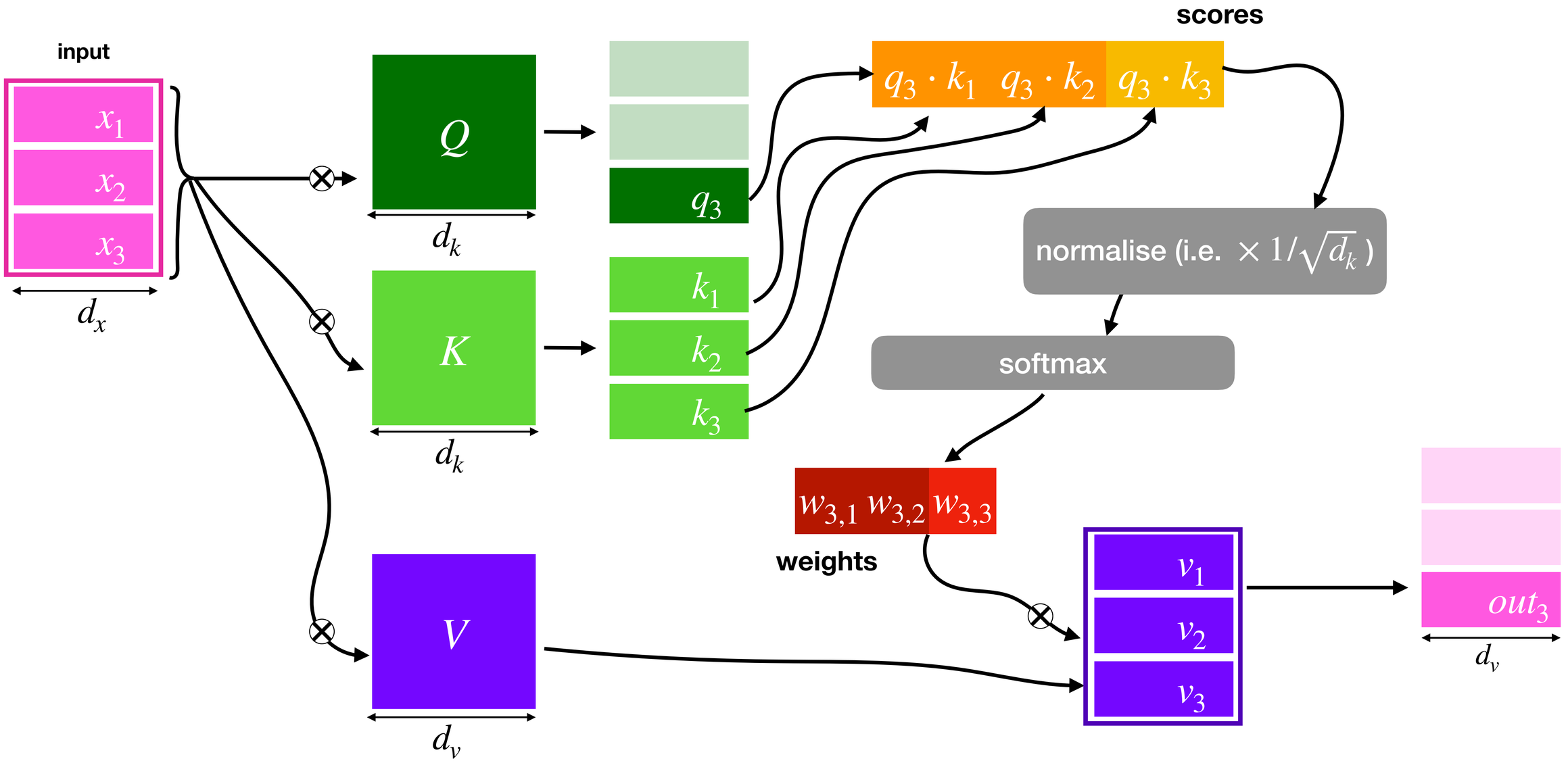
Background - Self Attention (Single Head)



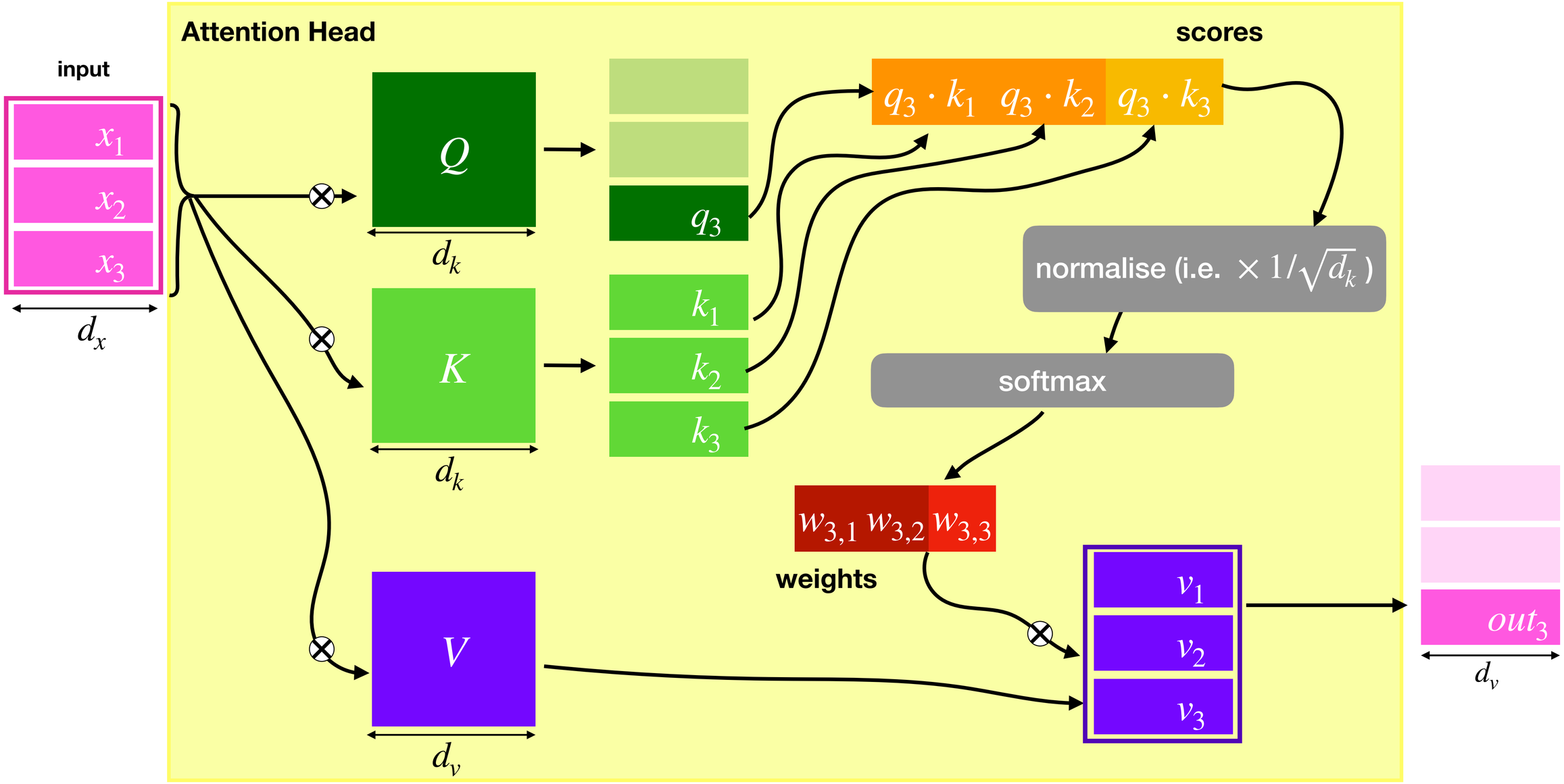
Background - Self Attention (Single Head)



Background - Self Attention (Single Head)

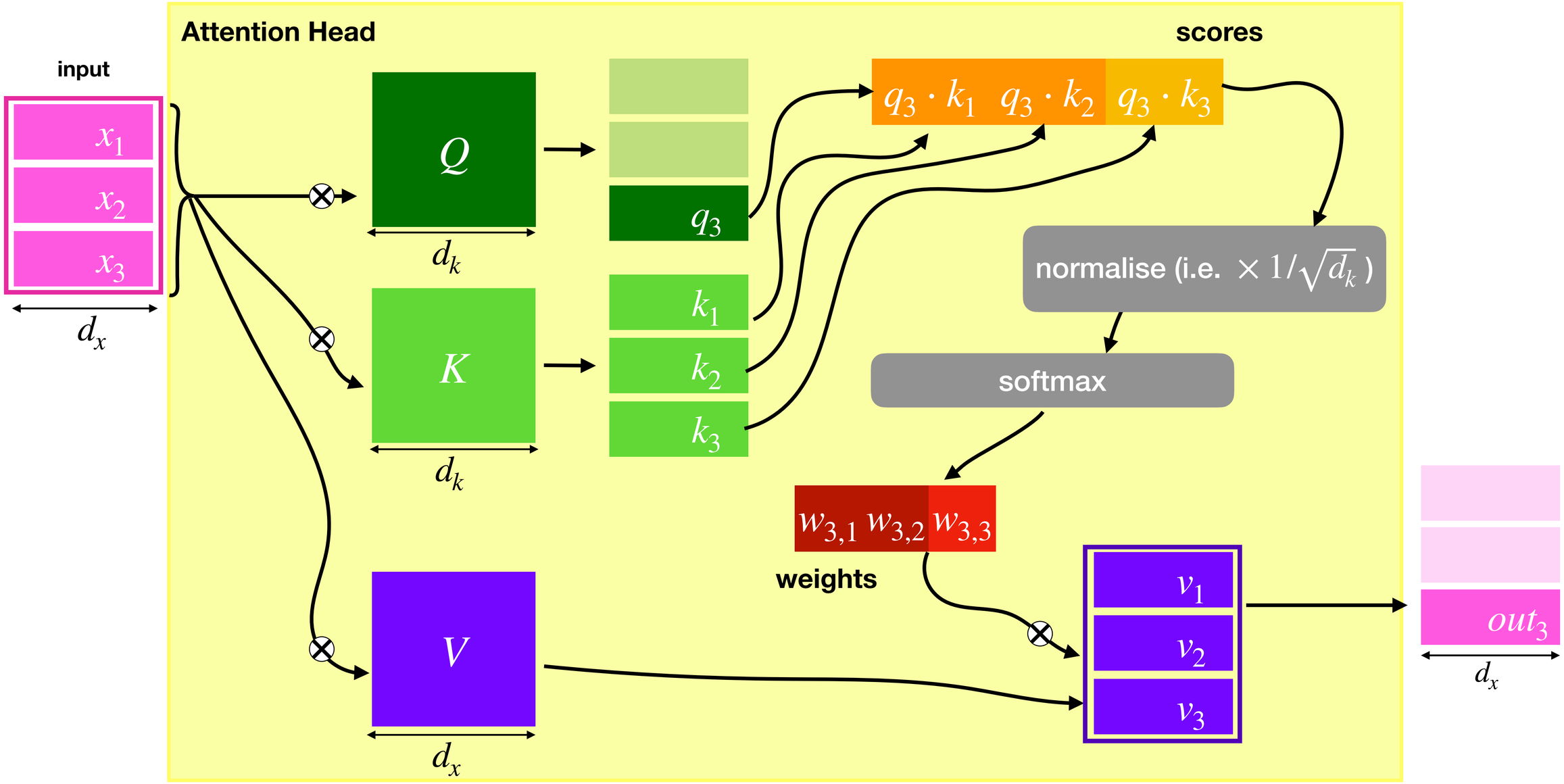


Background - Self Attention (Single Head)

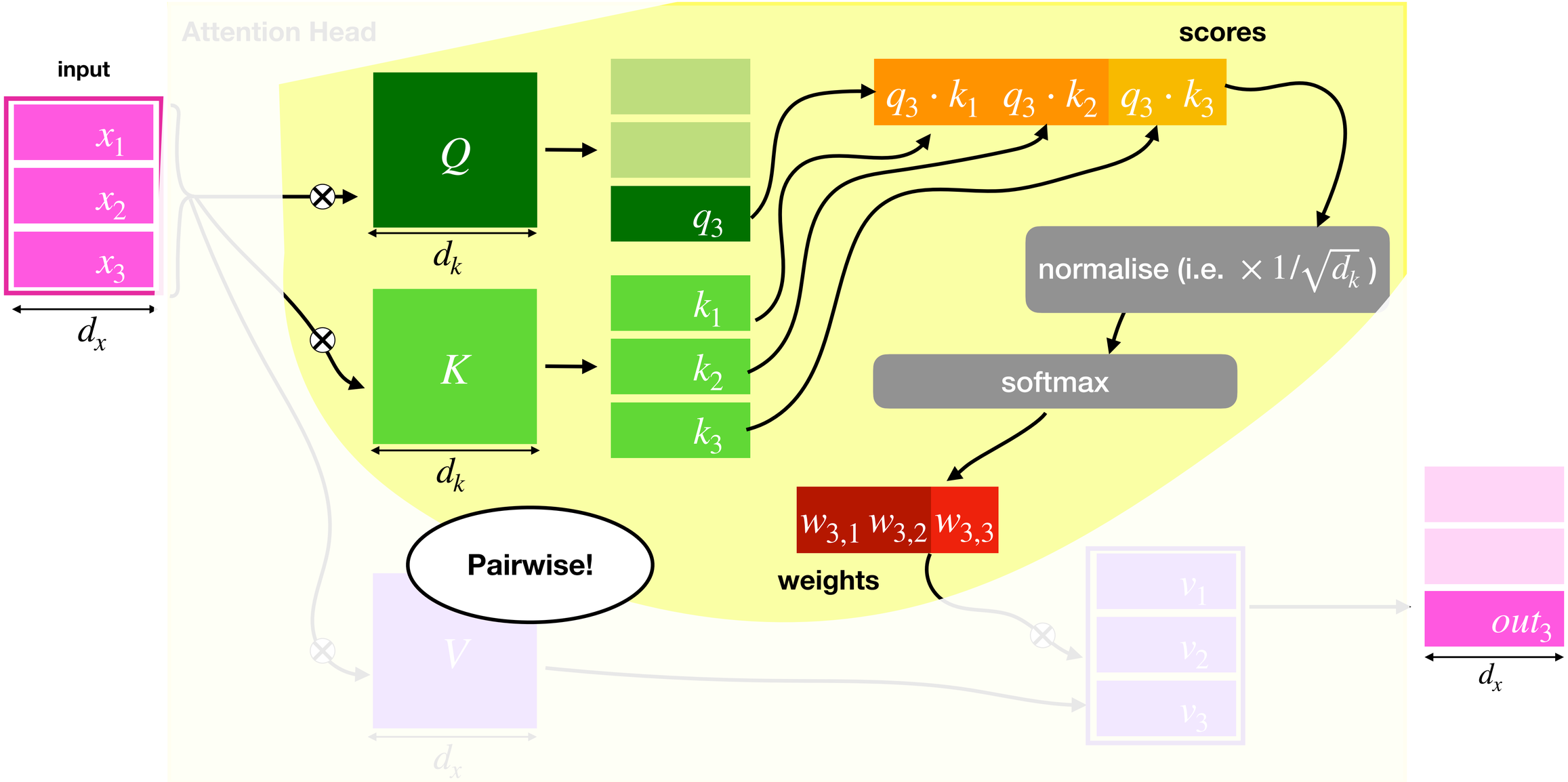


So, how do we present an attention head?

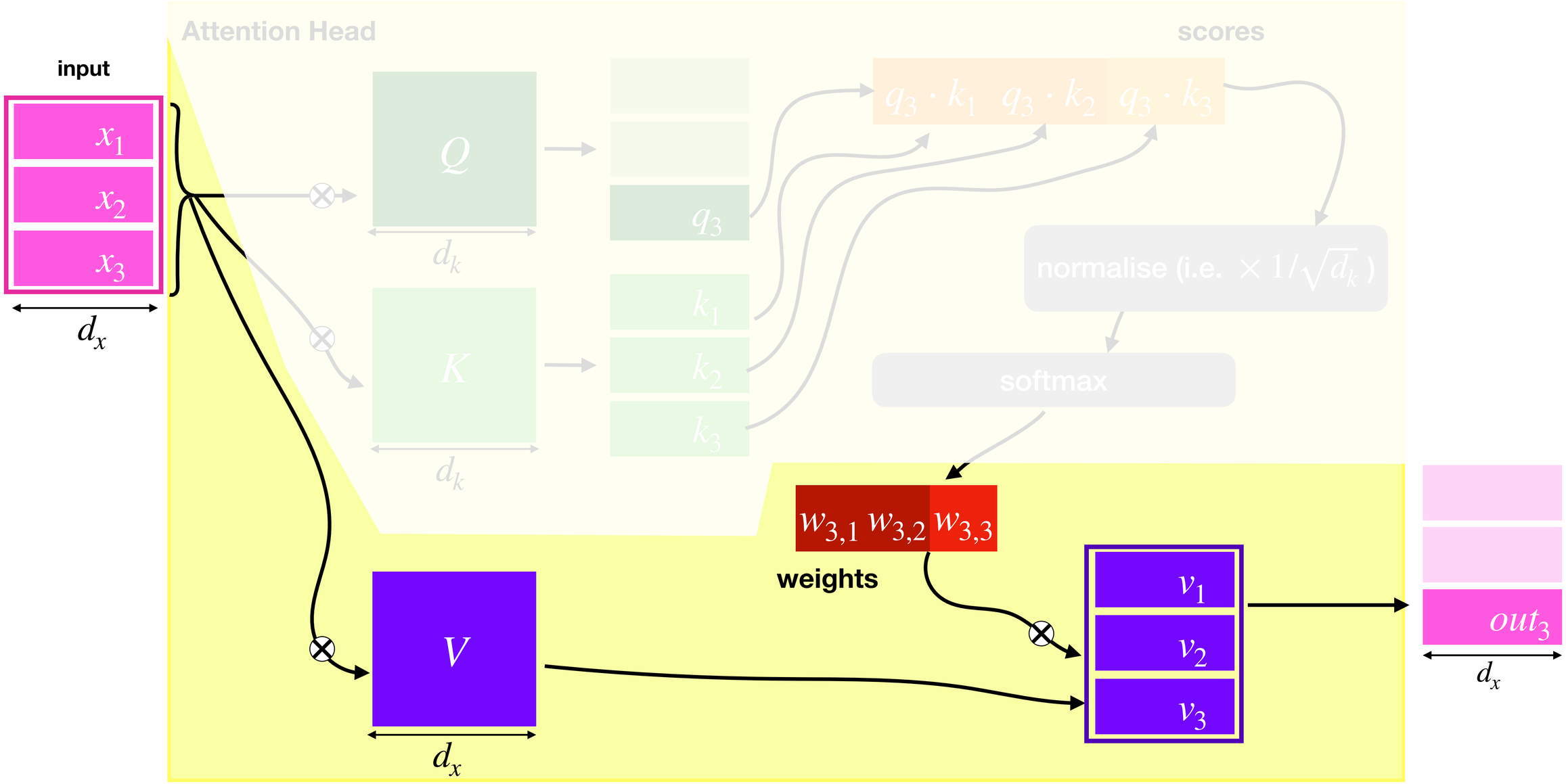
Self Attention (Single Head)



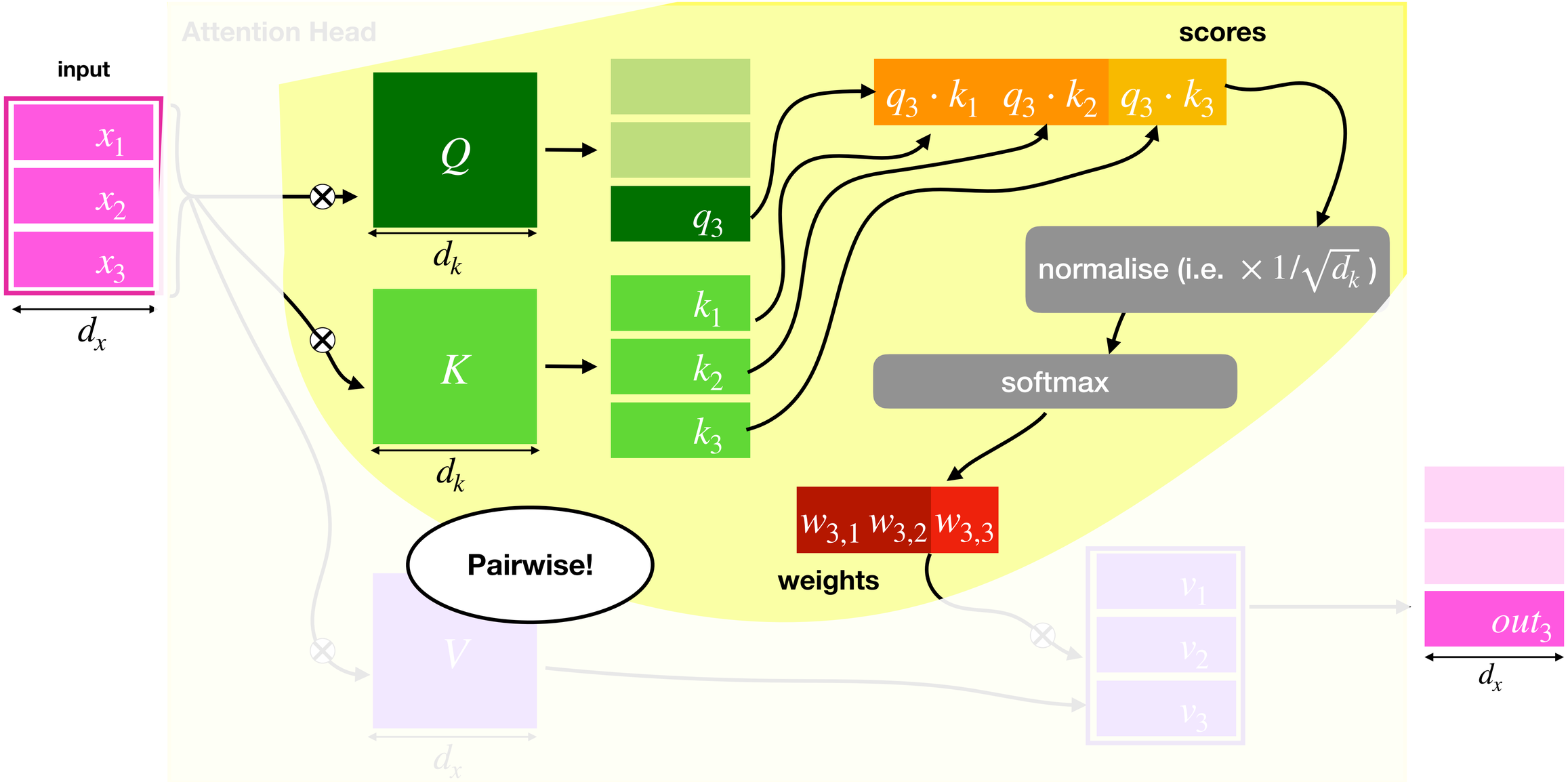
Self Attention (Single Head)



Self Attention (Single Head)



Single Head: Scoring ↔ Selecting



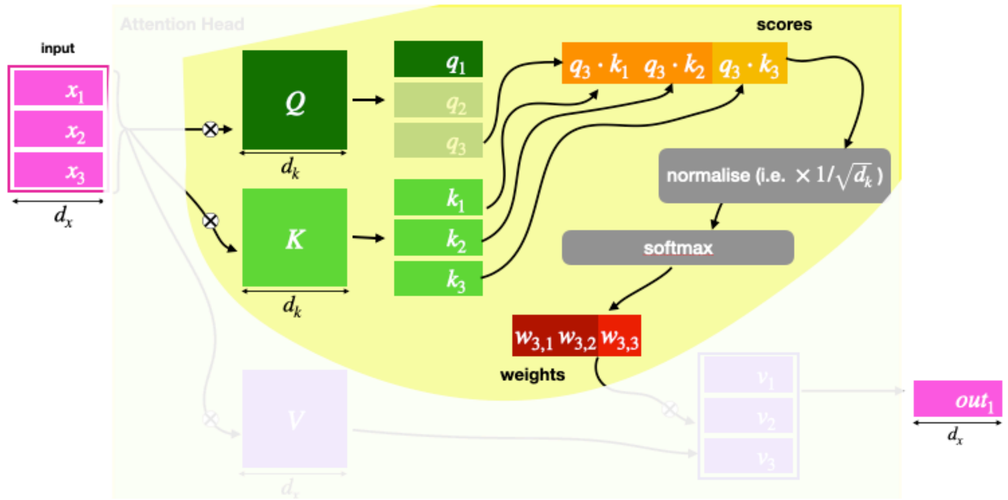
Single Head: Scoring ↔ Selecting

Decision: RASP abstracts to binary select/don't select decisions

sel = select([2,0,0],[0,1,2],==)

* This is not RASP syntax - RASP composes functions. Will see soon

	2	0	0
0	F	T	T
1	F	F	F
2	T	F	F



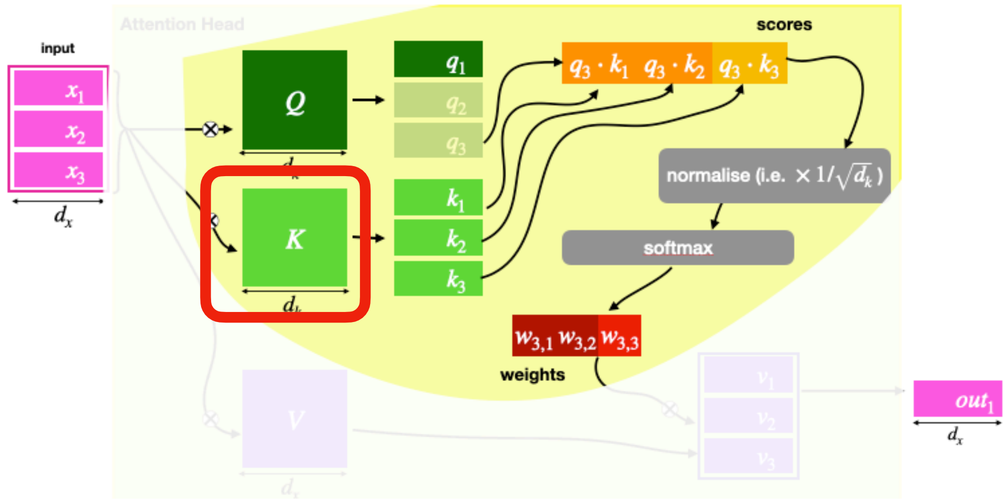
Single Head: Scoring ↔ Selecting

Decision: RASP abstracts to binary select/don't select decisions

sel = select([2,0,0],[0,1,2],==)

	2	0	0
0	F	T	T
1	F	F	F
2	T	F	F

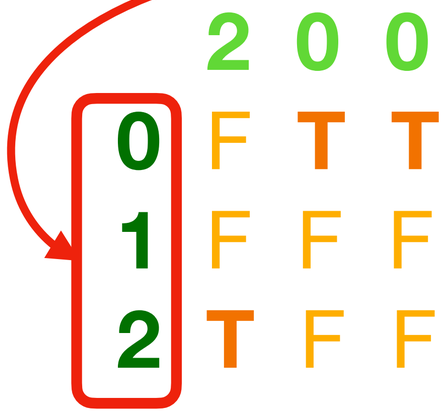
* This is not RASP syntax - RASP composes functions. Will see soon



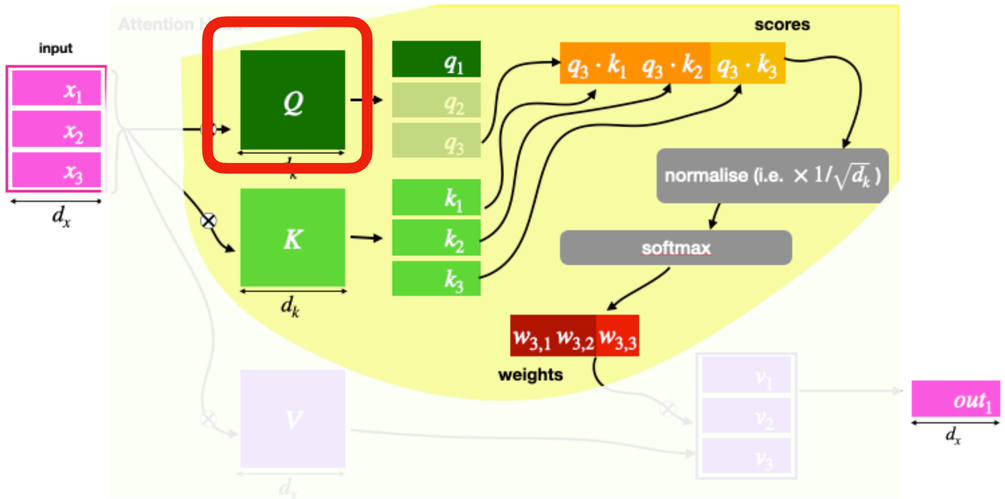
Single Head: Scoring ↔ Selecting

Decision: RASP abstracts to binary select/don't select decisions

sel = select([2,0,0], [0,1,2], ==)



* This is not RASP syntax - RASP composes functions. Will see soon



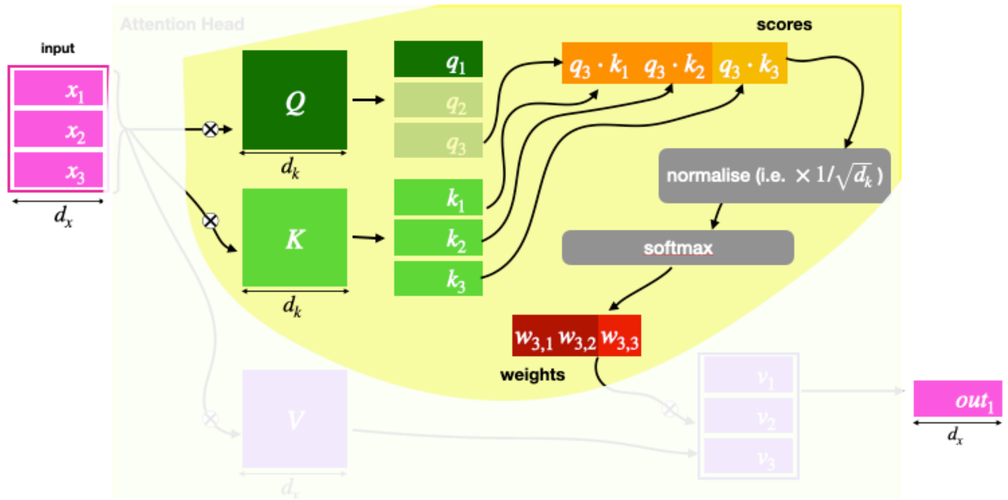
Single Head: Scoring ↔ Selecting

Decision: RASP abstracts to binary select/don't select decisions

sel = select([2,0,0],[0,1,2],==)

* This is not RASP syntax - RASP composes functions. Will see soon

	2	0	0
0	F	T	T
1	F	F	F
2	T	F	F



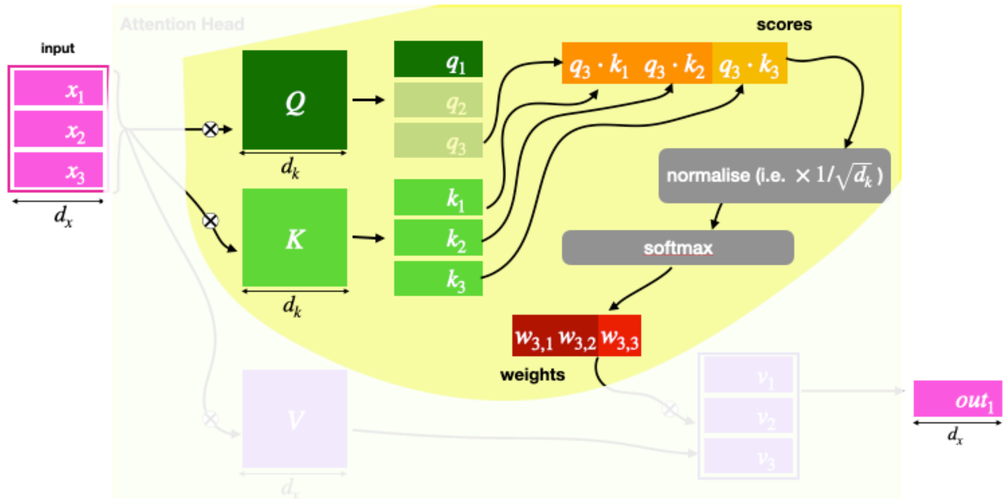
Single Head: Scoring ↔ Selecting

Decision: RASP abstracts to binary select/don't select decisions

sel = select([2,0,0],[0,1,2],==)

	2	0	0
0	F	T	T
1	F	F	F
2	T	F	F

* This is not RASP syntax - RASP composes functions. Will see soon



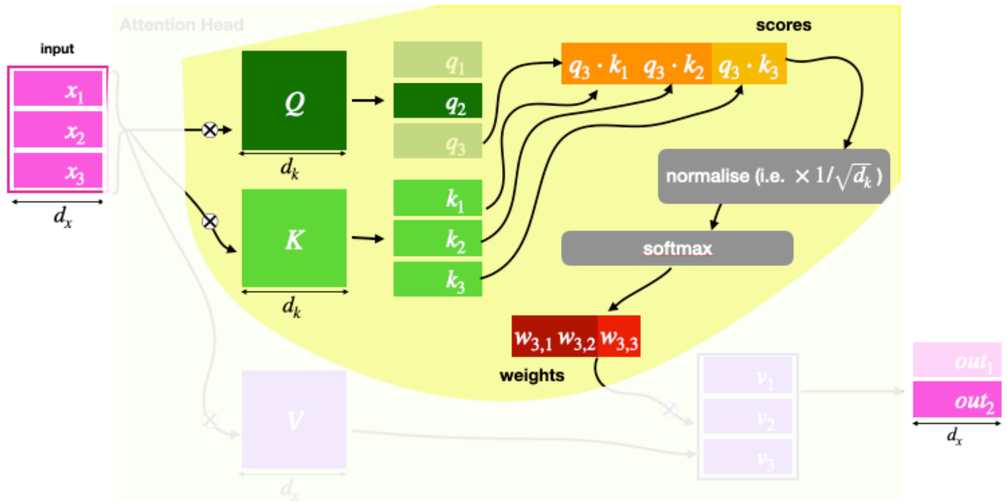
Single Head: Scoring ↔ Selecting

Decision: RASP abstracts to binary select/don't select decisions

sel = select([2,0,0],[0,1,2],==)

* This is not RASP syntax - RASP composes functions. Will see soon

	2	0	0
0	F	T	T
1	F	F	F
2	T	F	F



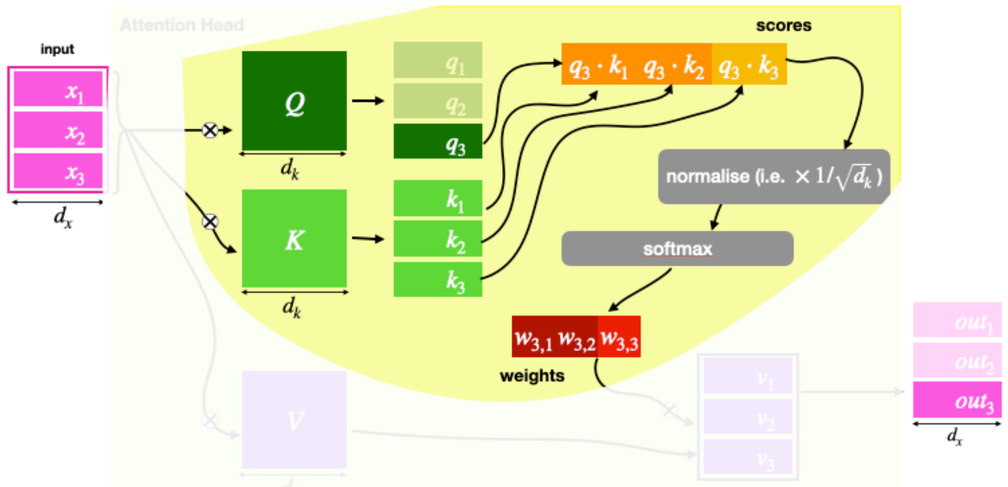
Single Head: Scoring ↔ Selecting

Decision: RASP abstracts to binary select/don't select decisions

sel = select([2,0,0],[0,1,2],==)

	2	0	0
0	F	T	T
1	F	F	F
2	T	F	F

* This is not RASP syntax - RASP composes functions. Will see soon



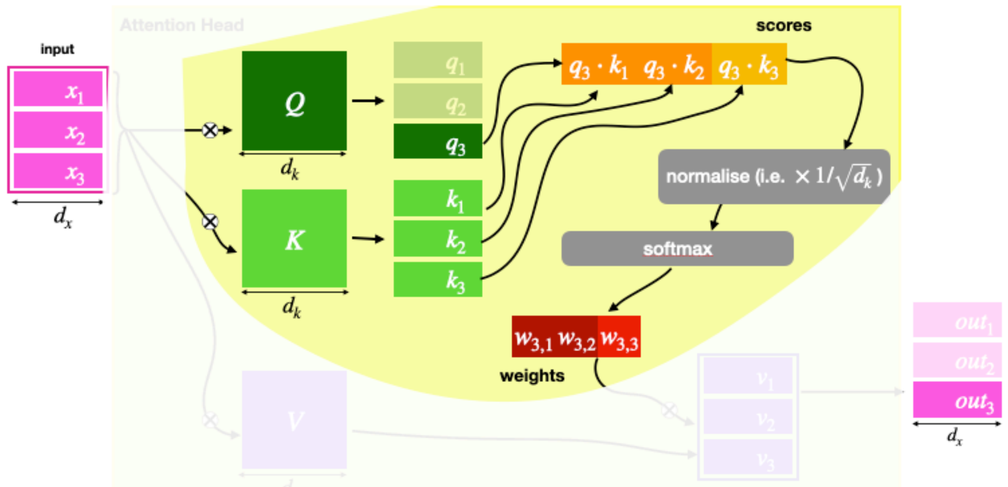
Single Head: Scoring ↔ Selecting

Decision: RASP abstracts to binary select/don't select decisions

sel = select([2,0,0],[0,1,2],==)

* This is not RASP syntax - RASP composes functions. Will see soon

	2	0	0
0	F	T	T
1	F	F	F
2	T	F	F

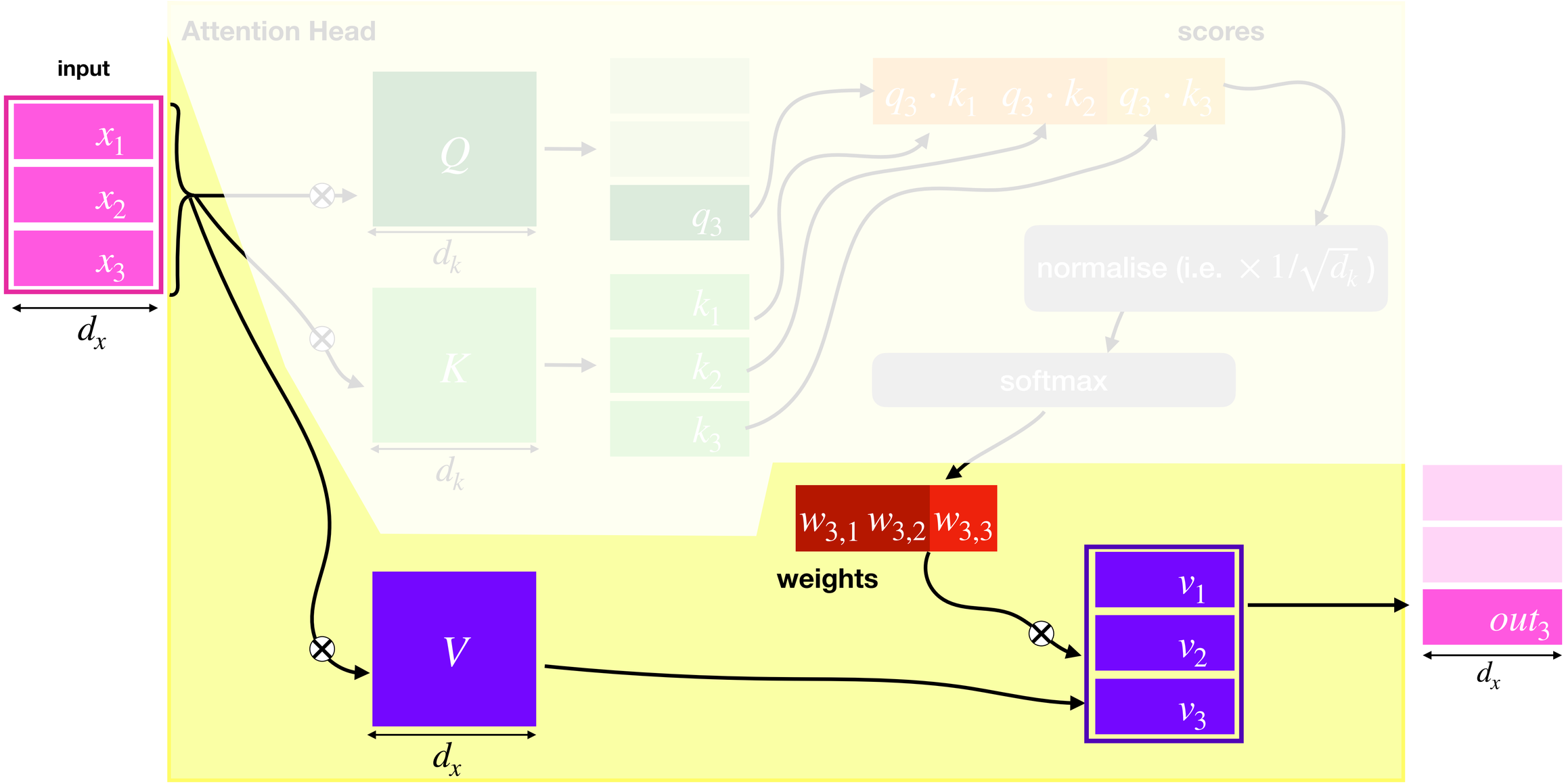


Another example:

sel2 = select([2,0,0],[0,1,2],>=)

	2	0	0
0	T	T	T
1	T	F	F
2	T	F	F

Single Head: Weighted Average ↔ Aggregation

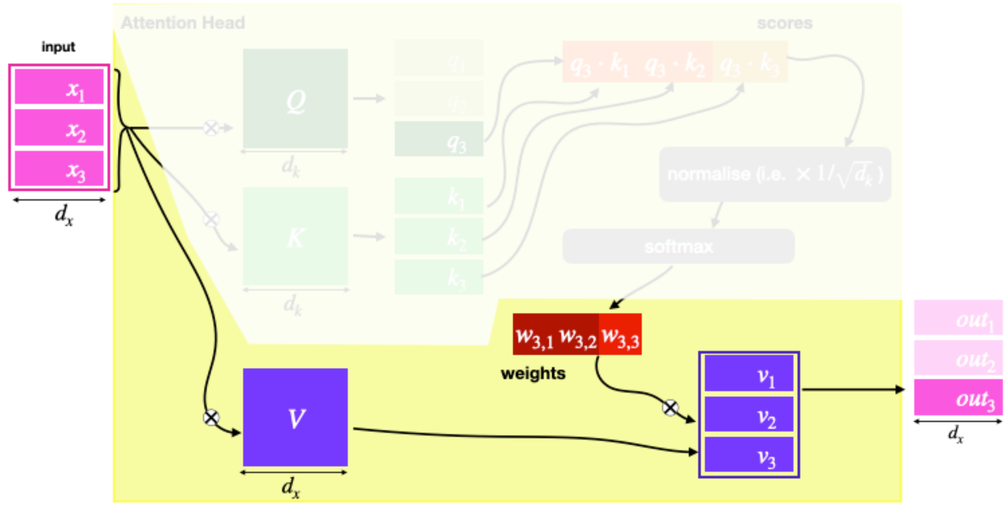


Single Head: Weighted Average ↔ Aggregation

new=aggregate(sel, [1,2,4])

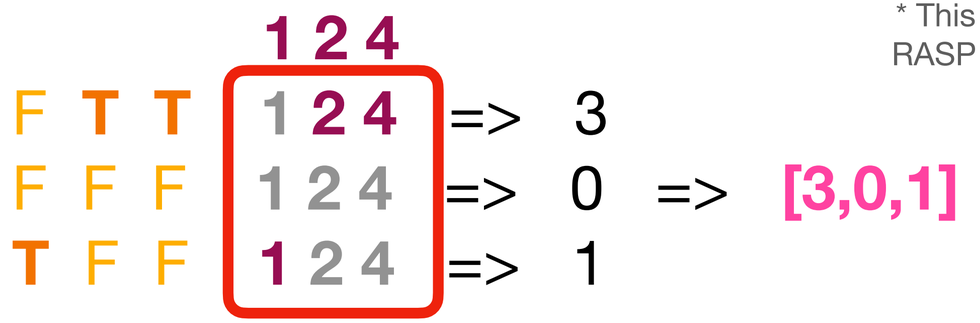
```
      1 2 4
F T T 1 2 4  =>  3
F F F 1 2 4  =>  0  =>  [3,0,1]
T F F 1 2 4  =>  1
```

* This is not RASP syntax - RASP composes functions. Will see soon

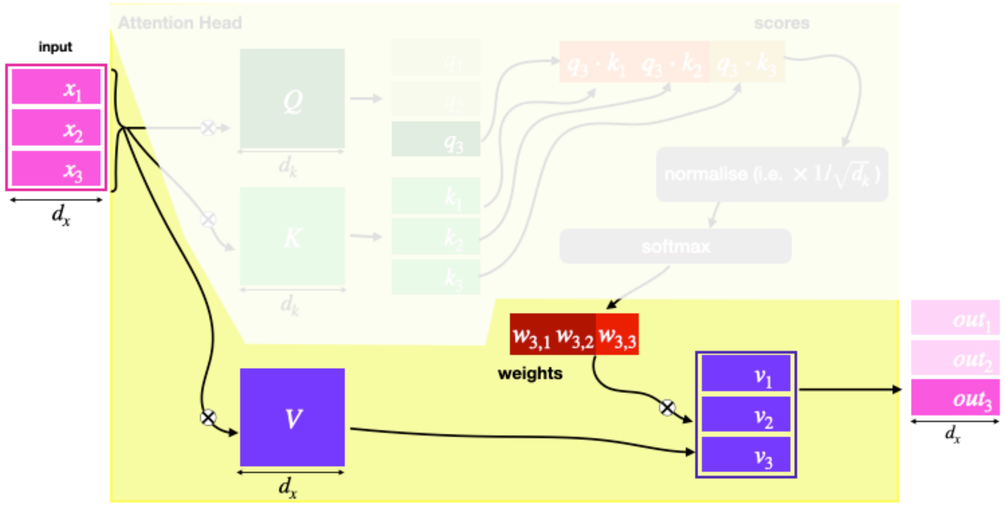


Single Head: Weighted Average ↔ Aggregation

new=aggregate(sel, [1,2,4])

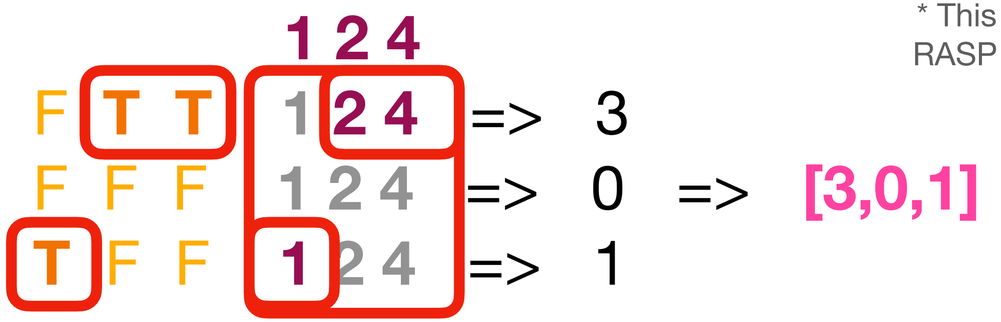


* This is not RASP syntax - RASP composes functions. Will see soon

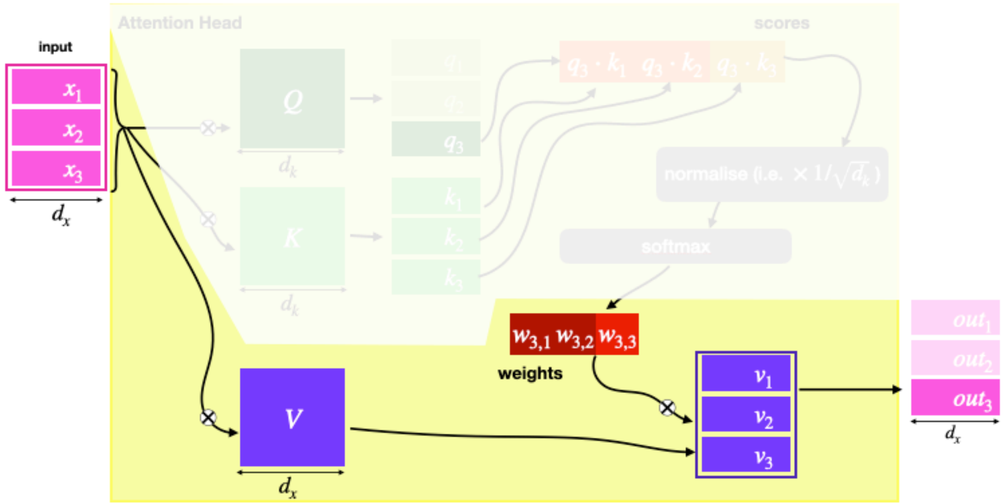


Single Head: Weighted Average ↔ Aggregation

new=aggregate(sel, [1,2,4])

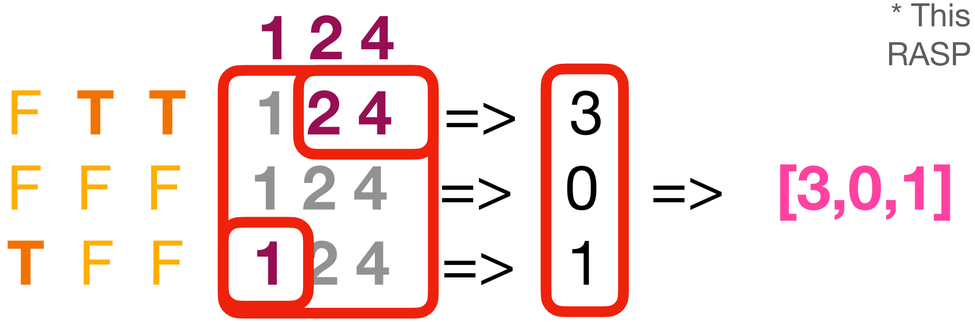


* This is not RASP syntax - RASP composes functions. Will see soon

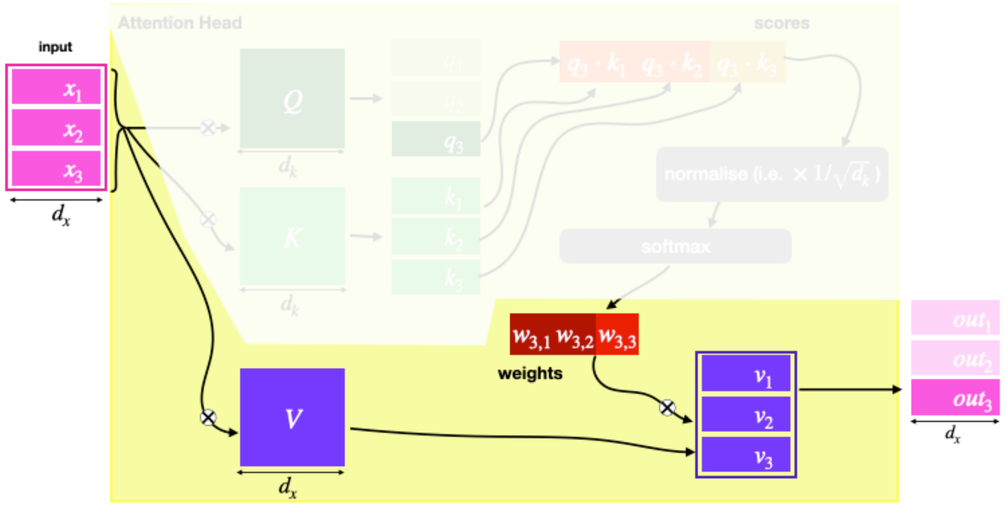


Single Head: Weighted Average ↔ Aggregation

new=aggregate(sel, [1,2,4])



* This is not RASP syntax - RASP composes functions. Will see soon

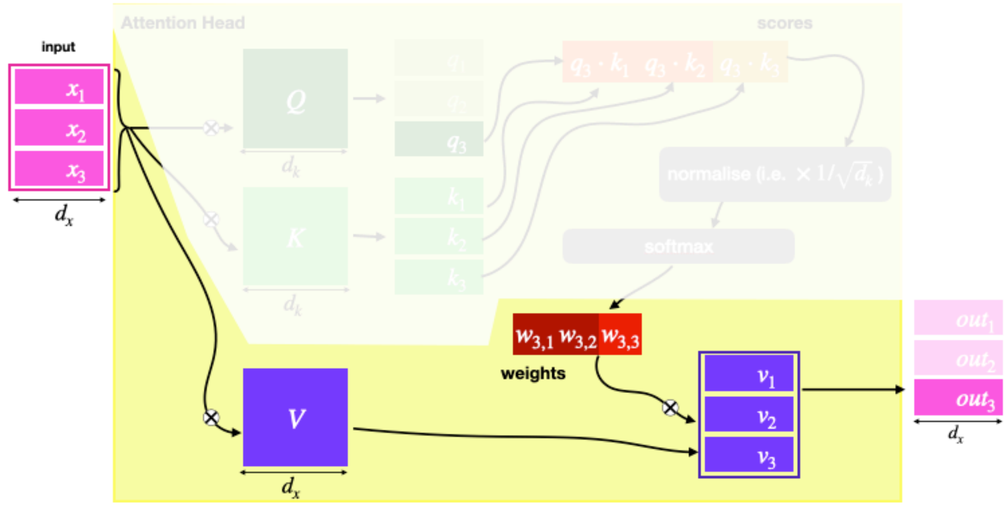


Single Head: Weighted Average ↔ Aggregation

new=aggregate(sel, [1,2,4])

```
          1 2 4
F T T     1 2 4  =>  3
F F F     1 2 4  =>  0  =>  [3,0,1]
T F F     1 2 4  =>  1
```

* This is not RASP syntax - RASP composes functions. Will see soon



Symbolic language + no averaging when only one position selected allows (for example):

reverse=aggregate(flip, [A,B,C])

```
          A B C
F F T     A B C  =>  C
F T F     A B C  =>  B  =>  [C,B,A]
T F F     A B C  =>  A
```

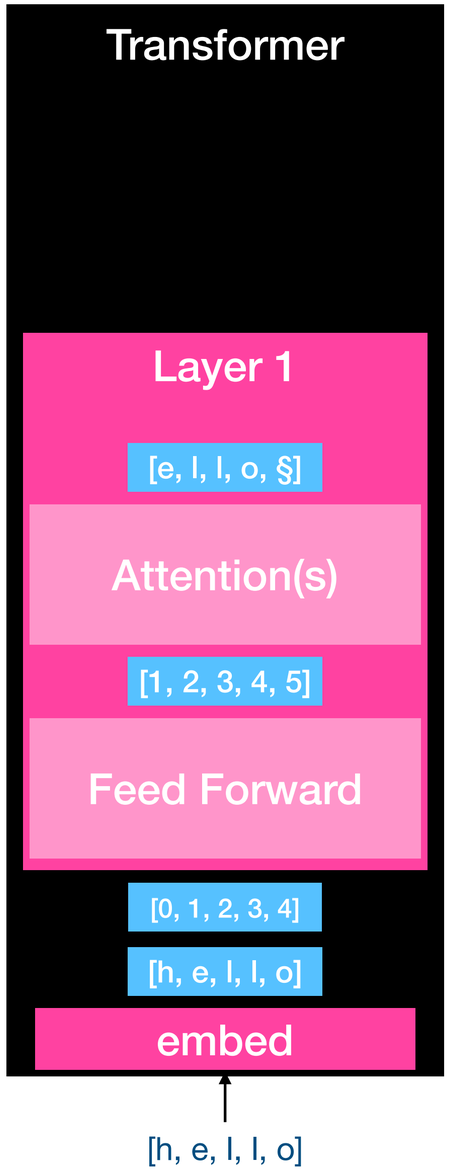
RASP s-ops and operations

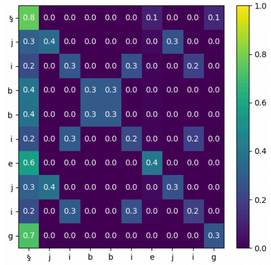
- Base s-ops:
  - **tokens**([a, b, c]) = [a, b, c]
  - **indices**([a, b, c]) = [0, 1, 2]
  - **0**([a, b, c]) = [0, 0, 0]
  - (all other constants)
- Elementwise operations:
  - **(indices+1)**([a, b, c]) = [1, 2, 3]
  - **(tokens=="b")**([a, b, c]) = [False, True, False]
  - **(3 if tokens=="b" else 0)**([a, b, c]) = [0, 3, 0]
  - (all other basic char/num/bool operations)

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
   1. macOS, linux: ./install.sh
   2. windows: follow "windows instructions.txt"
3. run:
   1. macOS, linux: ./rasp.sh
   2. windows: python3 RASP_support/REPL.py





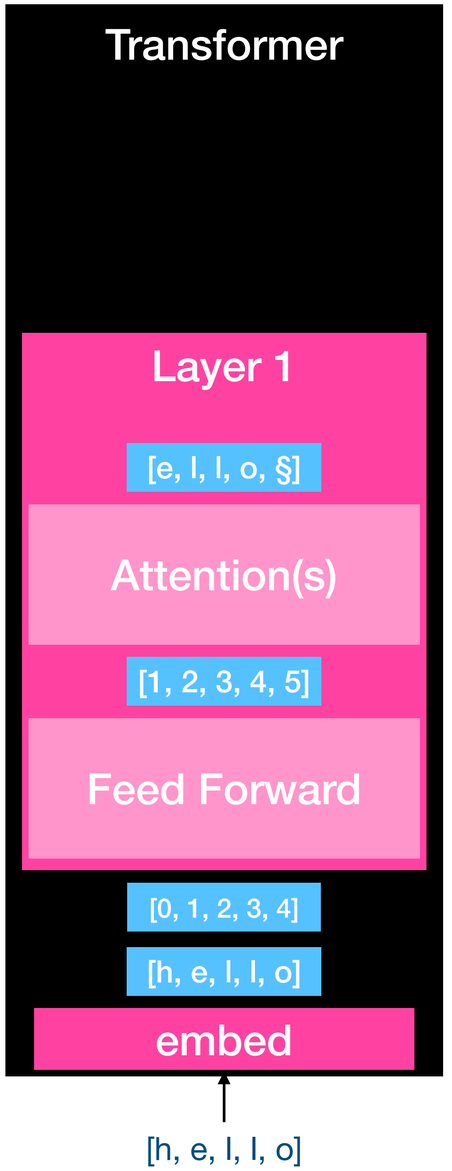
RASP s-ops and operations

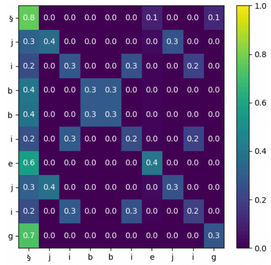
- Base s-ops:
  - **tokens**([a, b, c]) = [a, b, c]
  - **indices**([a, b, c]) = [0, 1, 2]
  - **0**([a, b, c]) = [0, 0, 0]
  - (all other constants)
- Elementwise operations:
  - **(indices+1)**([a, b, c]) = [1, 2, 3]
  - **(tokens=="b")**([a, b, c]) = [False, True, False]
  - **(3 if tokens=="b" else 0)**([a, b, c]) = [0, 3, 0]
  - (all other basic char/num/bool operations)
- Select-Aggregate operations:
  - **select(indices, indices+1, ==)**([a, b, c])= $\begin{pmatrix} 0,1,0 \\ 0,0,1 \\ 0,0,0 \end{pmatrix}$ (mark as **s**)
  - **aggregate(s, tokens, "!")**([a, b, c]) = [b, c, !]

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
   1. macOS, linux: ./install.sh
   2. windows: follow "windows instructions.txt"
3. run:
   1. macOS, linux: ./rasp.sh
   2. windows: python3 RASP_support/REPL.py





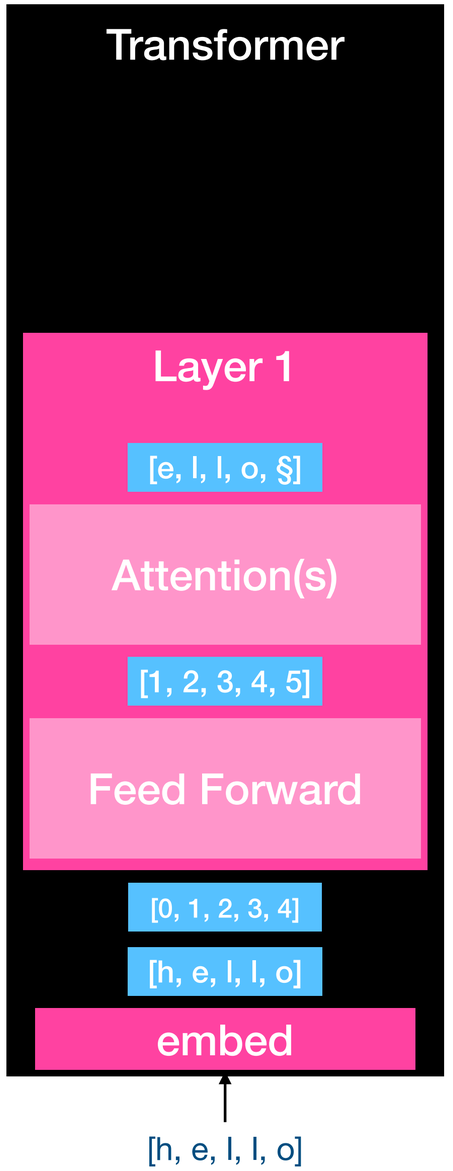
RASP s-ops and operations

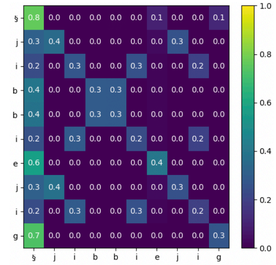
- Base s-ops:
  - **tokens**([a, b, c]) = [a, b, c]
  - **indices**([a, b, c]) = [0, 1, 2]
  - **0**([a, b, c]) = [0, 0, 0]
  - (all other constants)
- Elementwise operations:
  - **(indices+1)**([a, b, c]) = [1, 2, 3]
  - **(tokens=="b")**([a, b, c]) = [False, True, False]
  - **(3 if tokens=="b" else 0)**([a, b, c]) = [0, 3, 0]
  - (all other basic char/num/bool operations)
- Select-Aggregate operations:
  - **select(indices, indices+1, ==)**([a, b, c])= $\begin{pmatrix}0,1,0\\0,0,1\\0,0,0\end{pmatrix}$ (mark as **s**)
  - **aggregate(s, tokens, "!")**([a, b, c]) = [b, c, !]
- Extra: Selector combinations:
  - **(select(indices, indices+1, ==) or select(indices, 1, <))**([a, b, c]) = $\begin{pmatrix}1,1,0\\1,0,1\\1,0,0\end{pmatrix}$ $\begin{pmatrix}0,1,0\\0,0,1\\0,0,0\end{pmatrix}$ $\begin{pmatrix}1,0,0\\1,0,0\\1,0,0\end{pmatrix}$

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
   1. macOS, linux: ./install.sh
   2. windows: follow "windows instructions.txt"
3. run:
   1. macOS, linux: ./rasp.sh
   2. windows: python3 RASP_support/REPL.py





RASP s-ops and operations

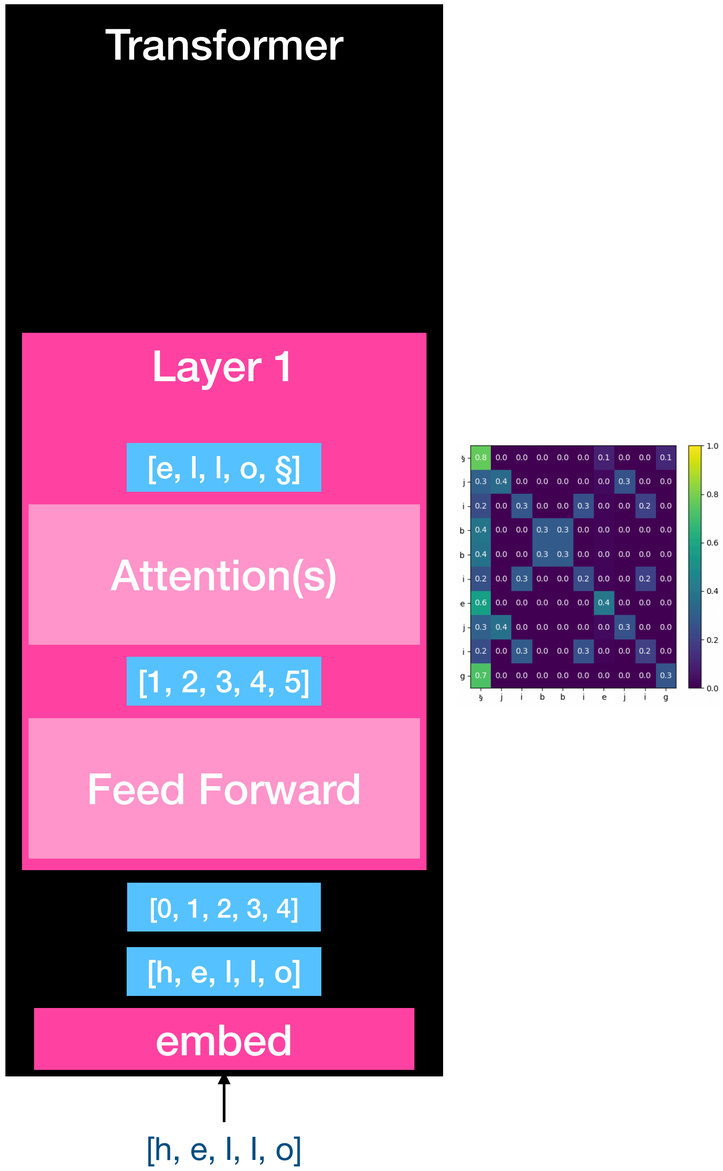
- Base s-ops:
  - **tokens**([a, b, c]) = [a, b, c]
  - **indices**([a, b, c]) = [0, 1, 2]
  - **0**([a, b, c]) = [0, 0, 0]
  - (all other constants)
- Elementwise operations:
  - **(indices+1)**([a, b, c]) = [1, 2, 3]
  - **(tokens=="b")**([a, b, c]) = [False, True, False]
  - **(3 if tokens=="b" else 0)**([a, b, c]) = [0, 3, 0]
  - (all other basic char/num/bool operations)
- Select-Aggregate operations:
  - **select(indices, indices+1, ==)**([a, b, c])= $\begin{pmatrix}0,1,0\\0,0,1\\0,0,0\end{pmatrix}$ (mark as **s**)
  - **aggregate(s, tokens, "!")**([a, b, c]) = [b, c, !]
- Extra: Selector combinations:
  - **(select(indices, indices+1, ==) or select(indices, 1, <))**([a, b, c]) = $\begin{pmatrix}1,1,0\\1,0,1\\1,0,0\end{pmatrix}$

$\begin{pmatrix}0,1,0\\0,0,1\\0,0,0\end{pmatrix}$ $\begin{pmatrix}1,0,0\\1,0,0\\1,0,0\end{pmatrix}$ can also be **and** or **not**

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
   1. macOS, linux: ./install.sh
   2. windows: follow "windows instructions.txt"
3. run:
   1. macOS, linux: ./rasp.sh
   2. windows: python3 RASP_support/REPL.py



RASP s-ops and operations

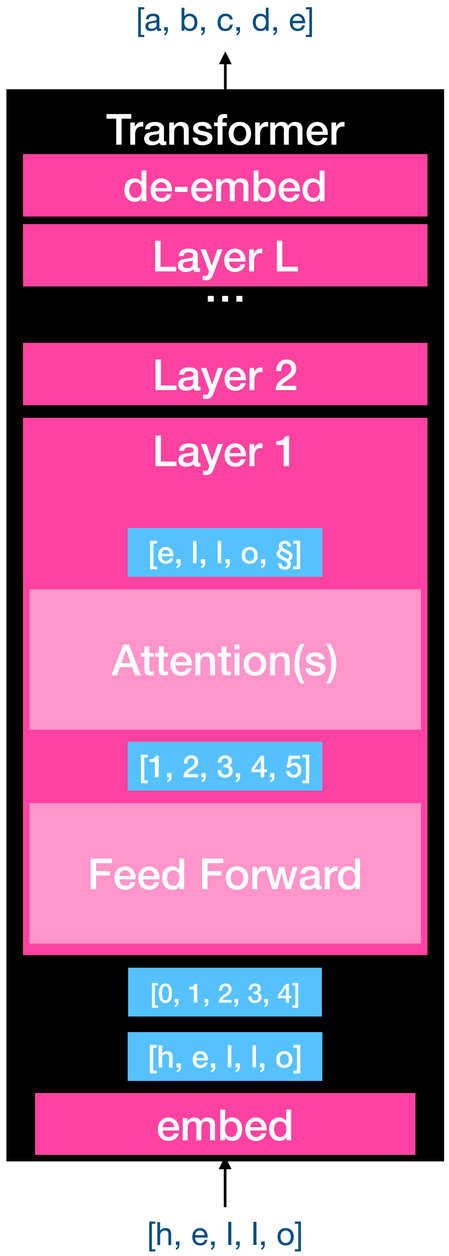
- Base s-ops:
  - **tokens**([a, b, c]) = [a, b, c]
  - **indices**([a, b, c]) = [0, 1, 2]
  - **0**([a, b, c]) = [0, 0, 0]
  - (all other constants)
- Elementwise operations:
  - **(indices+1)**([a, b, c]) = [1, 2, 3]
  - **(tokens=="b")**([a, b, c]) = [False, True, False]
  - **(3 if tokens=="b" else 0)**([a, b, c]) = [0, 3, 0]
  - (all other basic char/num/bool operations)
- Select-Aggregate operations:
  - **select(indices, indices+1, ==)**([a, b, c])= $\begin{pmatrix}0,1,0\\0,0,1\\0,0,0\end{pmatrix}$ (mark as **s**)
  - **aggregate(s, tokens, "!")**([a, b, c]) = [b, c, !]
- Extra: Selector combinations:
  - **(select(indices, indices+1, ==) or select(indices, 1, <))**([a, b, c]) = $\begin{pmatrix}1,1,0\\1,0,1\\1,0,0\end{pmatrix}$

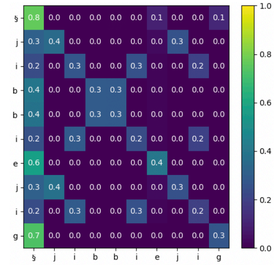
$\begin{pmatrix}0,1,0\\0,0,1\\0,0,0\end{pmatrix}$ $\begin{pmatrix}1,0,0\\1,0,0\\1,0,0\end{pmatrix}$ can also be **and** or **not**

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
   1. macOS, linux: ./install.sh
   2. windows: follow "windows instructions.txt"
3. run:
   1. macOS, linux: ./rasp.sh
   2. windows: python3 RASP_support/REPL.py





Small RASP exercises

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
    1. macOS, linux: ./install.sh
    2. windows: follow “windows instructions.txt”
3. run:
    1. macOS, linux: ./rasp.sh
    2. windows: python3 RASP_support/REPL.py

Small RASP exercises

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
    1. macOS, linux: ./install.sh
    2. windows: follow “windows instructions.txt”
3. run:
    1. macOS, linux: ./rasp.sh
    2. windows: python3 RASP_support/REPL.py

Quick prep

```
>> set example "banana"
>> tokens;
    s-op: tokens
        Example: tokens ("banana") = [b, a, n, a, n, a] (strings)
>> indices;
    s-op: indices
        Example: indices ("banana") = [0, 1, 2, 3, 4, 5] (ints)
```

Small RASP exercises

1. Mark “a”’s

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
    1. macOS, linux: ./install.sh
    2. windows: follow “windows instructions.txt”
3. run:
    1. macOS, linux: ./rasp.sh
    2. windows: python3 RASP_support/REPL.py

Target: set “a” tokens as “!”, leave others unchanged

Small RASP exercises

1. Mark “a”’s

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
    1. macOS, linux: ./install.sh
    2. windows: follow “windows instructions.txt”
3. run:
    1. macOS, linux: ./rasp.sh
    2. windows: python3 RASP_support/REPL.py

Target: set “a” tokens as “!”, leave others unchanged

Solution: comparison + ternary operator

Small RASP exercises

1. Mark “a”’s

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
    1. macOS, linux: ./install.sh
    2. windows: follow “windows instructions.txt”
3. run:
    1. macOS, linux: ./rasp.sh
    2. windows: python3 RASP_support/REPL.py

Target: set “a” tokens as “!”, leave others unchanged

Solution: comparison + ternary operator

```
tokens=="a"
```

Small RASP exercises

1. Mark “a”’s

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
    1. macOS, linux: ./install.sh
    2. windows: follow “windows instructions.txt”
3. run:
    1. macOS, linux: ./rasp.sh
    2. windows: python3 RASP_support/REPL.py

Target: set “a” tokens as “!”, leave others unchanged

Solution: comparison + ternary operator

```
>> sol1 = "!" if tokens=="a" else tokens;
```

Small RASP exercises

1. Mark “a”’s

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
   1. macOS, linux: ./install.sh
   2. windows: follow “windows instructions.txt”
3. run:
   1. macOS, linux: ./rasp.sh
   2. windows: python3 RASP_support/REPL.py

Target: set “a” tokens as “!”, leave others unchanged

Solution: comparison + ternary operator

```
>> sol1 = "!" if tokens=="a" else tokens;
      s-op: sol1
          Example: sol1 ("banana") = [b, !, n, !, n, !] (strings)
```

Small RASP exercises

1. Mark “a”’s

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
   1. macOS, linux: ./install.sh
   2. windows: follow “windows instructions.txt”
3. run:
   1. macOS, linux: ./rasp.sh
   2. windows: python3 RASP_support/REPL.py

Target: set “a” tokens as “!”, leave others unchanged

Solution: comparison + ternary operator

```
>> sol1 = "!" if tokens=="a" else tokens;
     s-op: sol1
         Example: sol1 ("banana") = [b, !, n, !, n, !] (strings)
>> sol1("abc");
         =  [!, b, c] (strings)
```

Small RASP exercises

2. Repeat first token

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
    1. macOS, linux: ./install.sh
    2. windows: follow “windows instructions.txt”
3. run:
    1. macOS, linux: ./rasp.sh
    2. windows: python3 RASP_support/REPL.py

Target: output first token at each position, e.g. “abc” ↦ “aaa”

Small RASP exercises

2. Repeat first token

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
    1. macOS, linux: ./install.sh
    2. windows: follow “windows instructions.txt”
3. run:
    1. macOS, linux: ./rasp.sh
    2. windows: python3 RASP_support/REPL.py

Target: output first token at each position, e.g. “abc” ↦ “aaa”

Solution: (1) have all tokens focus on first position, (2) copy it

Small RASP exercises

2. Repeat first token

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
   1. macOS, linux: ./install.sh
   2. windows: follow “windows instructions.txt”
3. run:
   1. macOS, linux: ./rasp.sh
   2. windows: python3 RASP_support/REPL.py

Target: output first token at each position, e.g. “abc” ↦ “aaa”

Solution: (1) have all tokens focus on first position, (2) copy it

```
>> sel2 = select(indices,0,==);
     selector: sel2
         Example:
```

	b	a	n	a	n	a
b	1					
a	1					
n	1					
a	1					
n	1					
a	1					

Small RASP exercises

2. Repeat first token

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
   1. macOS, linux: ./install.sh
   2. windows: follow “windows instructions.txt”
3. run:
   1. macOS, linux: ./rasp.sh
   2. windows: python3 RASP_support/REPL.py

Target: output first token at each position, e.g. “abc” ↦ “aaa”

Solution: (1) have all tokens focus on first position, (2) copy it

```
>> sel2 = select(indices,0,==);
     selector: sel2
         Example:
                       b a n a n a
                   b | 1
                   a | 1
                   n | 1
                   a | 1
                   n | 1
                   a | 1
>> sol2 = aggregate(sel2, tokens);
     s-op: sol2
         Example: sol2 ("banana") = [b]*6 (strings)
```

Small RASP exercises

3. Mark first instances

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
    1. macOS, linux: ./install.sh
    2. windows: follow “windows instructions.txt”
3. run:
    1. macOS, linux: ./rasp.sh
    2. windows: python3 RASP_support/REPL.py

Target: mark for each position whether it shows new token, e.g. “aba” ↦ [T, T, F]

Small RASP exercises

3. Mark first instances

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
    1. macOS, linux: ./install.sh
    2. windows: follow “windows instructions.txt”
3. run:
    1. macOS, linux: ./rasp.sh
    2. windows: python3 RASP_support/REPL.py

Target: mark for each position whether it shows new token, e.g. “aba” ↦ [T, T, F]

Solution: (1) seek previous positions with same token, (2) aggregate their positions, (3) report whether legal

Small RASP exercises

3. Mark first instances

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
   1. macOS, linux: ./install.sh
   2. windows: follow “windows instructions.txt”
3. run:
   1. macOS, linux: ./rasp.sh
   2. windows: python3 RASP_support/REPL.py

Target: mark for each position whether it shows new token, e.g. “aba” ↦ [T, T, F]

Solution: (1) seek previous positions with same token, (2) aggregate their positions, (3) report whether legal

```
>> prev_instances = select(tokens, tokens, ==) and select(indices, indices, <);
    selector: prev_instances
        Example:
                    b a n a n a
                b |
                a |
                n |
                a |   1
                n |     1
                a |   1   1
```

Small RASP exercises

3. Mark first instances

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
   1. macOS, linux: ./install.sh
   2. windows: follow “windows instructions.txt”
3. run:
   1. macOS, linux: ./rasp.sh
   2. windows: python3 RASP_support/REPL.py

Target: mark for each position whether it shows new token, e.g. “aba” ↦ [T, T, F]

Solution: (1) seek previous positions with same token, (2) aggregate their positions, (3) report whether legal

```
>> prev_instances = select(tokens, tokens, ==) and select(indices, indices, <);
     selector: prev_instances
         Example:
                          b a n a n a
                      b |
                      a |
                      n |
                      a |   1
                      n |     1
                      a |   1   1
>> prev_locs = aggregate(prev_instances, indices, -1);
     s-op: prev_locs
         Example: prev_locs ("banana") = [-1, -1, -1, 1, 2, 2.0] (floats)
```

Small RASP exercises

3. Mark first instances

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
   1. macOS, linux: ./install.sh
   2. windows: follow “windows instructions.txt”
3. run:
   1. macOS, linux: ./rasp.sh
   2. windows: python3 RASP_support/REPL.py

Target: mark for each position whether it shows new token, e.g. “aba” ↦ [T, T, F]

Solution: (1) seek previous positions with same token, (2) aggregate their positions, (3) report whether legal

```
>> prev_instances = select(tokens, tokens, ==) and select(indices, indices, <);
     selector: prev_instances
         Example:
                          b a n a n a
                      b |
                      a |
                      n |
                      a |   1
                      n |     1
                      a |   1   1
>> prev_locs = aggregate(prev_instances, indices, -1);
     s-op: prev_locs
         Example: prev_locs ("banana") = [-1, -1, -1, 1, 2, 2.0] (floats)
>> sol3 = prev_locs==-1;
     s-op: sol3
         Example: sol3 ("banana") = [T, T, T, F, F, F] (bools)
```

Small RASP exercises

4. Get position of first “a”

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
    1. macOS, linux: ./install.sh
    2. windows: follow “windows instructions.txt”
3. run:
    1. macOS, linux: ./rasp.sh
    2. windows: python3 RASP_support/REPL.py

Target: output first “a” position at each position, e.g. “abc” ↦ [0,0,0]

Small RASP exercises

4. Get position of first “a”

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
    1. macOS, linux: ./install.sh
    2. windows: follow “windows instructions.txt”
3. run:
    1. macOS, linux: ./rasp.sh
    2. windows: python3 RASP_support/REPL.py

Target: output first “a” position at each position, e.g. “abc” ↦ [0,0,0]

Solution: (1) mark first token positions, (2) focus on first “a” position, (3) aggregate its position

Small RASP exercises

4. Get position of first “a”

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
   1. macOS, linux: ./install.sh
   2. windows: follow “windows instructions.txt”
3. run:
   1. macOS, linux: ./rasp.sh
   2. windows: python3 RASP_support/REPL.py

Target: output first “a” position at each position, e.g. “abc” ↦ [0,0,0]

Solution: (1) mark first token positions, (2) focus on first “a” position, (3) aggregate its position

```
>> is_first = sol3;
     s-op: is_first
         Example: is_first ("banana") = [T, T, T, F, F, F] (bools)
```

Small RASP exercises

4. Get position of first “a”

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
   1. macOS, linux: ./install.sh
   2. windows: follow “windows instructions.txt”
3. run:
   1. macOS, linux: ./rasp.sh
   2. windows: python3 RASP_support/REPL.py

Target: output first “a” position at each position, e.g. “abc” ↦ [0,0,0]

Solution: (1) mark first token positions, (2) focus on first “a” position, (3) aggregate its position

```
>> is_first = sol3;
     s-op: is_first
         Example: is_first ("banana") = [T, T, T, F, F, F] (bools)
>> sel_first_a = select(is_first,True,==) and select(tokens,"a",==);
     selector: sel_first_a
         Example:
                        b a n a n a
                    b |   1
                    a |   1
                    n |   1
                    a |   1
                    n |   1
                    a |   1
```

Small RASP exercises

4. Get position of first “a”

Follow along:

github.com/tech-srl/RASP

1. clone
2. set up:
   1. macOS, linux: ./install.sh
   2. windows: follow “windows instructions.txt”
3. run:
   1. macOS, linux: ./rasp.sh
   2. windows: python3 RASP_support/REPL.py

Target: output first “a” position at each position, e.g. “abc” ↦ [0,0,0]

Solution: (1) mark first token positions, (2) focus on first “a” position, (3) aggregate its position

```
>> is_first = sol3;
     s-op: is_first
         Example: is_first ("banana") = [T, T, T, F, F, F] (bools)
>> sel_first_a = select(is_first,True,==) and select(tokens,"a",==);
     selector: sel_first_a
         Example:
                          b a n a n a
                      b |   1
                      a |   1
                      n |   1
                      a |   1
                      n |   1
                      a |   1
>> sol4 = aggregate(sel_first_a, indices, -1);
     s-op: sol4
         Example: sol4 ("banana") = [1]*6 (ints)
```

Small RASP exercises

5. Compute length

Target: output input length at each position,

e.g. “abc” ↦ [3, 3, 3]

Small RASP exercises

5. Compute length

Target: output input length at each position,

e.g. “abc” ↦ [3, 3, 3]

Solution (attempt):

(1) select all positions

(2) send 1 from all positions

(3) aggregate the 1

Small RASP exercises

5. Compute length

Target: output input length at each position,

e.g. “abc” ↦ [3, 3, 3]

Solution (attempt):

(1) select all positions

(2) send 1 from all positions

(3) aggregate the 1

```
>> sel_all = select(1,1,==);
    selector: sel_all
        Example:
```

	b	a	n	a	n	a
b	1	1	1	1	1	1
a	1	1	1	1	1	1
n	1	1	1	1	1	1
a	1	1	1	1	1	1
n	1	1	1	1	1	1
a	1	1	1	1	1	1

Small RASP exercises

5. Compute length

Target: output input length at each position,

e.g. “abc” ↦ [3, 3, 3]

Solution (attempt):

(1) select all positions

(2) send 1 from all positions

(3) aggregate the 1

```
>> sel_all = select(1,1,==);
    selector: sel_all
        Example:
                                  b a n a n a
                              b | 1 1 1 1 1 1
                              a | 1 1 1 1 1 1
                              n | 1 1 1 1 1 1
                              a | 1 1 1 1 1 1
                              n | 1 1 1 1 1 1
                              a | 1 1 1 1 1 1
>> v = aggregate(sel_all, 1);
```

Small RASP exercises

5. Compute length

Target: output input length at each position,

e.g. “abc” ↦ [3, 3, 3]

Solution (attempt):

(1) select all positions

(2) send 1 from all positions

(3) aggregate the 1

(4) … get 1

```
>> sel_all = select(1,1,==);
     selector: sel_all
         Example:
                                 b a n a n a
                               b | 1 1 1 1 1 1
                               a | 1 1 1 1 1 1
                               n | 1 1 1 1 1 1
                               a | 1 1 1 1 1 1
                               n | 1 1 1 1 1 1
                               a | 1 1 1 1 1 1
>> v = aggregate(sel_all, 1);
     s-op: v
         Example: v("banana") = [1.0]*6 (floats)
```

Small RASP exercises

5. Compute length

Target: output input length at each position,

e.g. “abc” ↦ [3, 3, 3]

Solution (attempt):

(1) select all positions

(2) send 1 from all positions

(3) aggregate the 1

(4) … get 1 regardless of length

```
>> sel_all = select(1,1,==);
     selector: sel_all
         Example:
                                  b a n a n a
                              b | 1 1 1 1 1 1
                              a | 1 1 1 1 1 1
                              n | 1 1 1 1 1 1
                              a | 1 1 1 1 1 1
                              n | 1 1 1 1 1 1
                              a | 1 1 1 1 1 1
>> v = aggregate(sel_all, 1);
     s-op: v
         Example: v("banana") = [1.0]*6 (floats)
>> v("hi");
         =  [1.0]*2 (floats)
```

Small RASP exercises

5. Compute length

Target: output input length at each position,

e.g. “abc” ↦ [3, 3, 3]

Solution (attempt):

(1) select all positions

(2) send 1 from all positions

(3) aggregate the 1

(4) … get 1 regardless of length

attention *averages*, doesn’t sum!

```
>> sel_all = select(1,1,==);
     selector: sel_all
         Example:
                                  b a n a n a
                              b | 1 1 1 1 1 1
                              a | 1 1 1 1 1 1
                              n | 1 1 1 1 1 1
                              a | 1 1 1 1 1 1
                              n | 1 1 1 1 1 1
                              a | 1 1 1 1 1 1
>> v = aggregate(sel_all, 1);
     s-op: v
         Example: v("banana") = [1.0]*6 (floats)
>> v("hi");
         =  [1.0]*2 (floats)
```

Small RASP exercises

5. Compute length (v1)

Target: output input length at each position,

e.g. “abc” ↦ [3, 3, 3]

Solution (v1):

Small RASP exercises

5. Compute length (v1)

Target: output input length at each position,

e.g. “abc” ↦ [3, 3, 3]

Solution (v1):

(1) mark position without higher position

(2) get location of that position

Small RASP exercises

5. Compute length (v1)

Target: output input length at each position,

e.g. “abc” ↦ [3, 3, 3]

Solution (v1):

(1) mark position without higher position

(2) get location of that position

Small RASP exercises

5. Compute length (v1)

Target: output input length at each position,

e.g. “abc” ↦ [3, 3, 3]

Solution (v1):

(1) mark position without higher position:

(a) select next position

(b) aggregate its index

(c) mark if illegal result

(2) get location of that position

Small RASP exercises

5. Compute length (v1)

```
>> sel_next = select(indices, indices+1, ==);
    selector: sel_next
        Example:
                  b a n a n a
              b |   1
              a |     1
              n |       1
              a |         1
              n |           1
              a |
```

Target: output input length at each position,

e.g. “abc” ↦ [3, 3, 3]

Solution (v1):

(1) mark position without higher position:

(a) select next position

(b) aggregate its index

(c) mark if illegal result

(2) get location of that position

Small RASP exercises

5. Compute length (v1)

Target: output input length at each position,

e.g. “abc” ↦ [3, 3, 3]

Solution (v1):

(1) mark position without higher position:

(a) select next position

(b) aggregate its index

(c) mark if illegal result

(2) get location of that position

```
>> sel_next = select(indices, indices+1, ==);
     selector: sel_next
         Example:
                           b a n a n a
                       b |   1
                       a |     1
                       n |       1
                       a |         1
                       n |           1
                       a |
>> next_pos = aggregate(sel_next, indices, -1);
     s-op: next_pos
         Example: next_pos ("banana") = [1, 2, 3, 4, 5, -1] (ints)
```

Small RASP exercises

5. Compute length (v1)

Target: output input length at each position,

e.g. “abc” ↦ [3, 3, 3]

Solution (v1):

(1) mark position without higher position:

(a) select next position

(b) aggregate its index

(c) mark if illegal result

(2) get location of that position

```
>> sel_next = select(indices, indices+1, ==);
     selector: sel_next
         Example:
                           b a n a n a
                       b |   1
                       a |     1
                       n |       1
                       a |         1
                       n |           1
                       a |
>> next_pos = aggregate(sel_next, indices, -1);
     s-op: next_pos
         Example: next_pos ("banana") = [1, 2, 3, 4, 5, -1] (ints)
>> is_highest_pos = next_pos == -1;
     s-op: is_highest_pos
         Example: is_highest_pos ("banana") = [F, F, F, F, F, T] (bools)
```

Small RASP exercises

5. Compute length (v1)

Target: output input length at each position,

e.g. “abc” ↦ [3, 3, 3]

Solution (v1):

(1) mark position without higher position:

(a) select next position

(b) aggregate its index

(c) mark if illegal result

(2) get location of that position

```
>> sel_next = select(indices, indices+1, ==);
     selector: sel_next
         Example:
                           b a n a n a
                       b |   1
                       a |     1
                       n |       1
                       a |         1
                       n |           1
                       a |
>> next_pos = aggregate(sel_next, indices, -1);
     s-op: next_pos
         Example: next_pos ("banana") = [1, 2, 3, 4, 5, -1] (ints)
>> is_highest_pos = next_pos == -1;
     s-op: is_highest_pos
         Example: is_highest_pos ("banana") = [F, F, F, F, F, T] (bools)
```

Small RASP exercises

5. Compute length (v1)

Target: output input length at each position,

e.g. “abc” ↦ [3, 3, 3]

Solution (v1):

(1) mark position without higher position:

(a) select next position

(b) aggregate its index

(c) mark if illegal result

(2) get location of that position:

(a) select highest position

(b) aggregate its index

```
>> sel_next = select(indices, indices+1, ==);
     selector: sel_next
         Example:
                           b a n a n a
                       b |   1
                       a |     1
                       n |       1
                       a |         1
                       n |           1
                       a |
>> next_pos = aggregate(sel_next, indices, -1);
     s-op: next_pos
         Example: next_pos ("banana") = [1, 2, 3, 4, 5, -1] (ints)
>> is_highest_pos = next_pos == -1;
     s-op: is_highest_pos
         Example: is_highest_pos ("banana") = [F, F, F, F, F, T] (bools)
```

Small RASP exercises

5. Compute length (v1)

Target: output input length at each position,

e.g. “abc” ↦ [3, 3, 3]

Solution (v1):

(1) mark position without higher position:

(a) select next position

(b) aggregate its index

(c) mark if illegal result

(2) get location of that position

(a) select highest position

(b) aggregate its index

```
>> sel_next = select(indices, indices+1, ==);
     selector: sel_next
         Example:
                         b a n a n a
                     b |   1
                     a |     1
                     n |       1
                     a |         1
                     n |           1
                     a |
>> next_pos = aggregate(sel_next, indices, -1);
     s-op: next_pos
         Example: next_pos ("banana") = [1, 2, 3, 4, 5, -1] (ints)
>> is_highest_pos = next_pos == -1;
     s-op: is_highest_pos
         Example: is_highest_pos ("banana") = [F, F, F, F, F, T] (bools)
>> sel_highest = select(is_highest_pos, True, ==);
     selector: sel_highest
         Example:
                         b a n a n a
                     b |           1
                     a |           1
                     n |           1
                     a |           1
                     n |           1
                     a |           1
```

Small RASP exercises

5. Compute length (v1)

Target: output input length at each position,

e.g. “abc” ↦ [3, 3, 3]

Solution (v1):

(1) mark position without higher position:

(a) select next position

(b) aggregate its index

(c) mark if illegal result

(2) get location of that position

(a) select highest position

(b) aggregate its index

```
>> sel_next = select(indices, indices+1, ==);
     selector: sel_next
         Example:
                           b a n a n a
                       b |   1
                       a |     1
                       n |       1
                       a |         1
                       n |           1
                       a |
>> next_pos = aggregate(sel_next, indices, -1);
     s-op: next_pos
         Example: next_pos ("banana") = [1, 2, 3, 4, 5, -1] (ints)
>> is_highest_pos = next_pos == -1;
     s-op: is_highest_pos
         Example: is_highest_pos ("banana") = [F, F, F, F, F, T] (bools)
>> sel_highest = select(is_highest_pos, True, ==);
     selector: sel_highest
         Example:
                           b a n a n a
                       b |           1
                       a |           1
                       n |           1
                       a |           1
                       n |           1
                       a |           1
>> length_v1 = aggregate(sel_highest, indices) + 1;
     s-op: length_v1
         Example: length_v1 ("banana") = [6]*6 (ints)
```

Small RASP exercises

5. Compute length (v1)

Target: output input length at each position,

e.g. “abc” ↦ [3, 3, 3]

Solution (v1):

(1) mark position without higher position:

(a) select next position

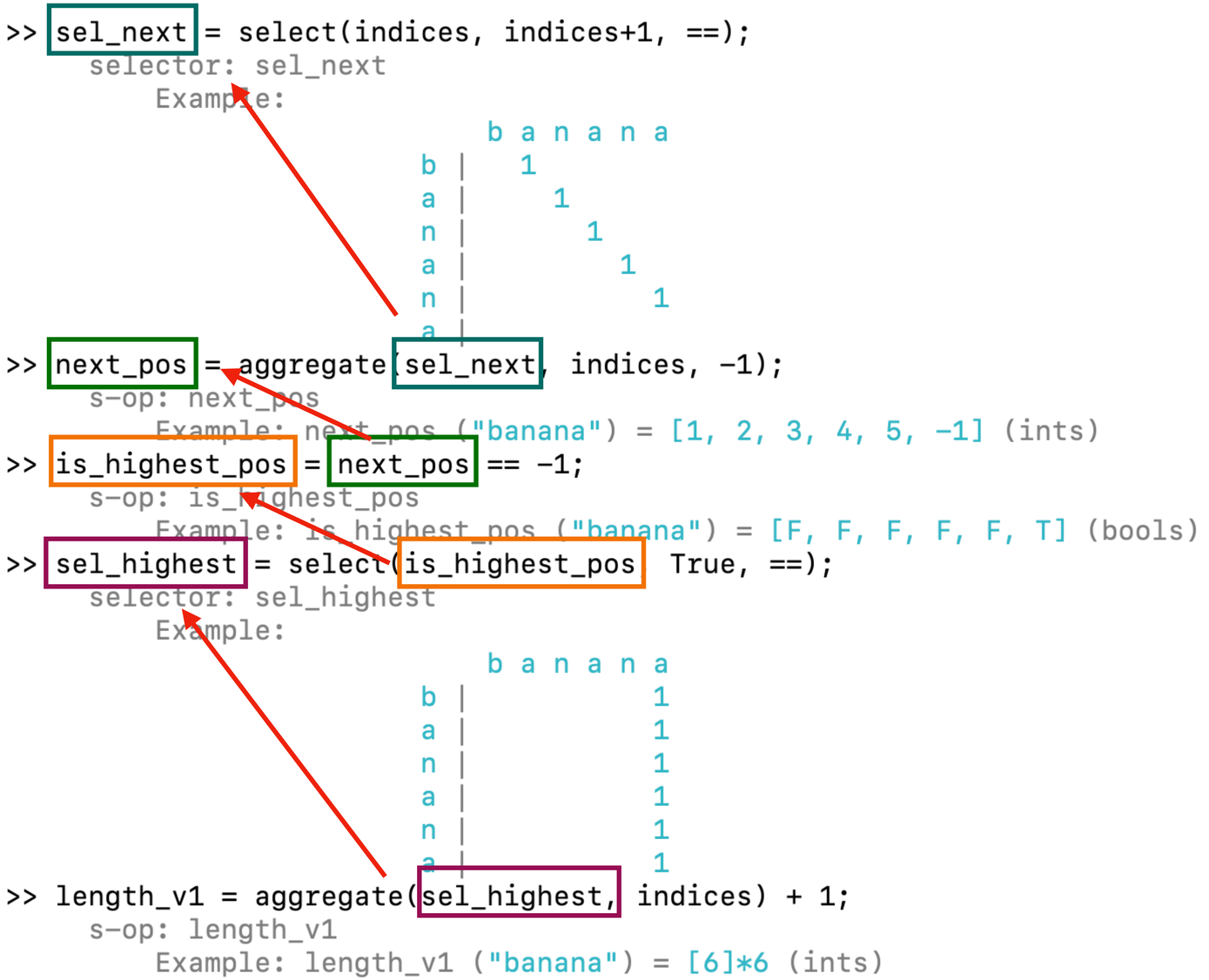
(b) aggregate its index

(c) mark if illegal result

(2) get location of that position

(a) select highest position

(b) aggregate its index



Small RASP exercises

5. Compute length (v1)

Target: output input length at each position,

e.g. “abc” ↦ [3, 3, 3]

Solution (v1):

(1) mark position without higher position:

(a) select next position

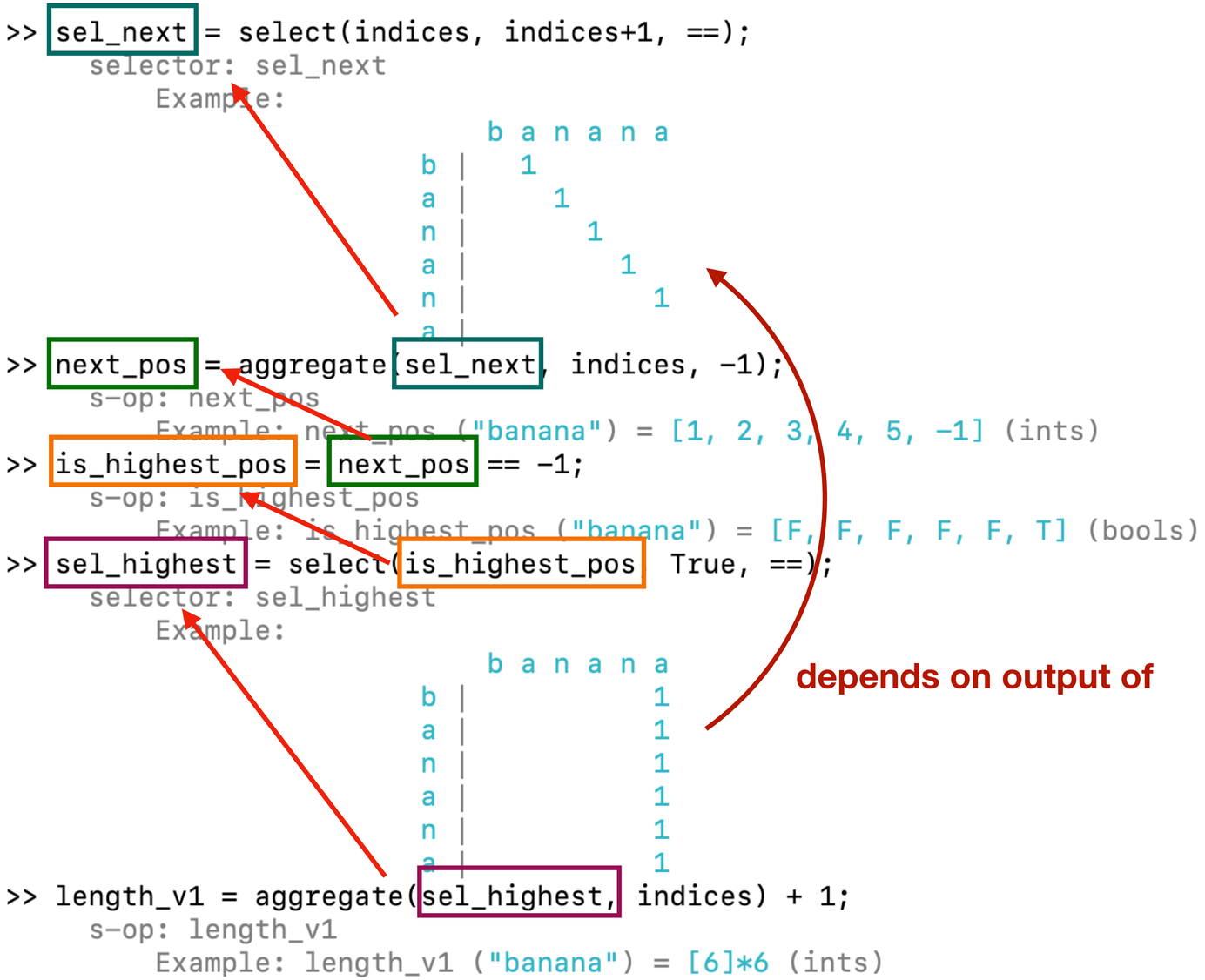
(b) aggregate its index

(c) mark if illegal result

(2) get location of that position

(a) select highest position

(b) aggregate its index



Small RASP exercises

5. Compute length (v1)

Target: output input length at each position,

e.g. “abc” ↦ [3, 3, 3]

Solution (v1):

(1) mark position without higher position:

(a) select next position

(b) aggregate its index

(c) mark if illegal result

(2) get location of that position

(a) select highest position

(b) aggregate its index

```
>> sel_next = select(indices, indices+1, ==);
     selector: sel_next
          Example:
                        b a n a n a
                  b |   1
                  a |     1
                  n |       1
                  a |         1
                  n |           1
                  a |
>> next_pos = aggregate(sel_next, indices, -1);
     s-op: next_pos
          Example: next_pos ("banana") = [1, 2, 3, 4, 5, -1] (ints)
>> is_highest_pos = next_pos == -1;
     s-op: is_highest_pos
          Example: is_highest_pos ("banana") = [F, F, F, F, F, T] (bools)
>> sel_highest = select(is_highest_pos, True, ==);
     selector: sel_highest
          Example:
                        b a n a n a
                  b |             1
                  a |             1
                  n |             1
                  a |             1
                  n |             1
                  a |             1
>> length_v1 = aggregate(sel_highest, indices) + 1;
     s-op: length_v1
          Example: length_v1 ("banana") = [6]*6 (ints)
```

depends on output of

2 layers!

Small RASP exercises

5. Compute length (v1)

Target: output input length at each position,

e.g. “abc” ↦ [3, 3, 3]

Solution (v1):

(1) mark position without higher position:

(a) select next position

(b) aggregate its index

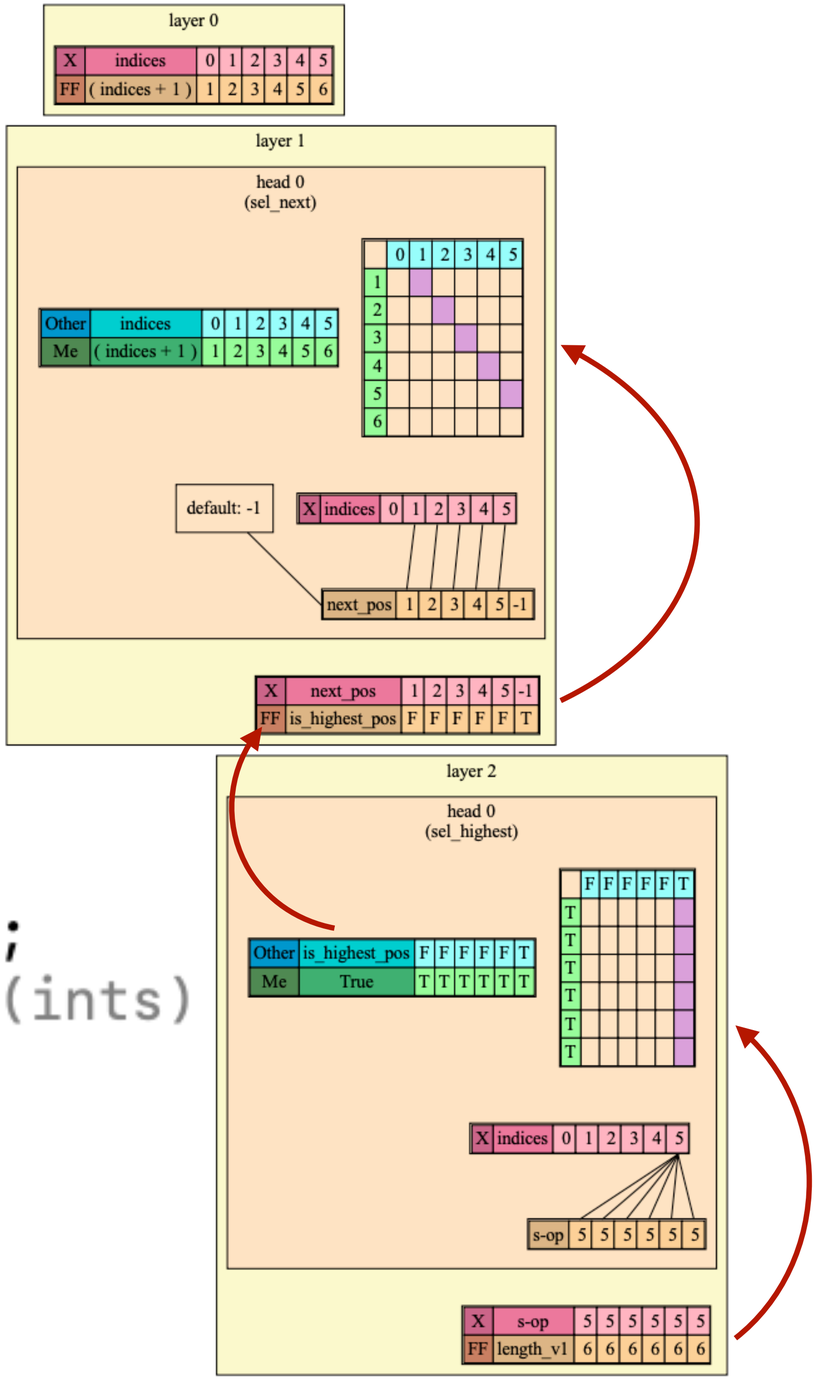
(c) mark if illegal result

(2) get location of that position

(a) select highest position

(b) aggregate its index

```
>> draw(length_v1);
      =  [6]*6 (ints)
```



2 layers

Small RASP exercises

6. Compute length (v2 - one layer)

Target: output input length at each position,

e.g. “abc” $\mapsto$ [3, 3, 3]

Solution (v2):

Small RASP exercises

6. Compute length (v2 - one layer)

Target: output input length at each position,

e.g. “abc” ↦ [3, 3, 3]

Solution (v2):

(1) mark *all* positions

(2) send 1 from only one location

(3) invert the average!

Small RASP exercises

6. Compute length (v2 - one layer)

Target: output input length at each position,

e.g. “abc” ↦ [3, 3, 3]

```
>> sel_all = select(1, 1, ==);
     selector: sel_all
         Example:
```

	b	a	n	a	n	a
b	1	1	1	1	1	1
a	1	1	1	1	1	1
n	1	1	1	1	1	1
a	1	1	1	1	1	1
n	1	1	1	1	1	1
a	1	1	1	1	1	1

Solution (v2):

(1) mark *all* positions

(2) send 1 from only one location

(3) invert the average!

Small RASP exercises

6. Compute length (v2 - one layer)

Target: output input length at each position,

e.g. “abc” ↦ [3, 3, 3]

Solution (v2):

(1) mark *all* positions

(2) send 1 from only one location

(3) invert the average!

```
>> sel_all = select(1, 1, ==);
     selector: sel_all
         Example:
                          b a n a n a
                        b | 1 1 1 1 1 1
                        a | 1 1 1 1 1 1
                        n | 1 1 1 1 1 1
                        a | 1 1 1 1 1 1
                        n | 1 1 1 1 1 1
                        a | 1 1 1 1 1 1
>> mark_zero = indicator(indices==0);
     s-op: mark_zero
         Example: mark_zero ("banana") = [1, 0, 0, 0, 0, 0] (ints)
```

Small RASP exercises

6. Compute length (v2 - one layer)

Target: output input length at each position,

e.g. “abc” ↦ [3, 3, 3]

Solution (v2):

(1) mark *all* positions

(2) send 1 from only one location

(3) invert the average!

```
>> sel_all = select(1, 1, ==);
     selector: sel_all
         Example:
                              b a n a n a
                          b | 1 1 1 1 1 1
                          a | 1 1 1 1 1 1
                          n | 1 1 1 1 1 1
                          a | 1 1 1 1 1 1
                          n | 1 1 1 1 1 1
                          a | 1 1 1 1 1 1
>> mark_zero = indicator(indices==0);
     s-op: mark_zero
         Example: mark_zero ("banana") = [1, 0, 0, 0, 0, 0] (ints)
>> inverted_length = aggregate(sel_all, mark_zero);
     s-op: inverted_length
         Example: inverted_length ("banana") = [0.167]*6 (floats)
```

Small RASP exercises

6. Compute length (v2 - one layer)

Target: output input length at each position,

e.g. “abc” ↦ [3, 3, 3]

Solution (v2):

(1) mark *all* positions

(2) send 1 from only one location

(3) invert the average!

```
>> sel_all = select(1, 1, ==);
     selector: sel_all
         Example:
                          b a n a n a
                        b | 1 1 1 1 1 1
                        a | 1 1 1 1 1 1
                        n | 1 1 1 1 1 1
                        a | 1 1 1 1 1 1
                        n | 1 1 1 1 1 1
                        a | 1 1 1 1 1 1
>> mark_zero = indicator(indices==0);
     s-op: mark_zero
         Example: mark_zero ("banana") = [1, 0, 0, 0, 0, 0] (ints)
>> inverted_length = aggregate(sel_all, mark_zero);
     s-op: inverted_length
         Example: inverted_length ("banana") = [0.167]*6 (floats)
>> length_v2 = round(1/inverted_length);
     s-op: length_v2
         Example: length_v2 ("banana") = [6]*6 (ints)
```

Small RASP exercises

6. Compute length (v2 - one layer)

Target: output input length at each position,

e.g. “abc” ↦ [3, 3, 3]

Solution (v2):

(1) mark *all* positions

(2) send 1 from only one location

(3) invert the average!

```
>> sel_all = select(1, 1, ==);
     selector: sel_all
         Example:
                          b a n a n a
                      b | 1 1 1 1 1 1
                      a | 1 1 1 1 1 1
                      n | 1 1 1 1 1 1
                      a | 1 1 1 1 1 1
                      n | 1 1 1 1 1 1
                      a | 1 1 1 1 1 1
>> mark_zero = indicator(indices==0);
     s-op: mark_zero
         Example: mark_zero ("banana") = [1, 0, 0, 0, 0, 0] (ints)
>> inverted_length = aggregate(sel_all, mark_zero);
     s-op: inverted_length
         Example: inverted_length ("banana") = [0.167]*6 (floats)
>> length_v2 = round(1/inverted_length);
     s-op: length_v2
         Example: length_v2 ("banana") = [6]*6 (ints)
```

Library s-op in RASP (“length”)

Small RASP exercises

7. Reverse

Target: flip input sequence, e.g. “abc” ↦ “cba”

Small RASP exercises

7. Reverse

Target: flip input sequence, e.g. “abc” ↦ “cba”

Solution: (1) compute opposite position, (2) seek it, (3) get its token

Small RASP exercises

7. Reverse

Target: flip input sequence, e.g. “abc” ↦ “cba”

Solution: (1) compute opposite position, (2) seek it, (3) get its token

```
>> opposite_pos = length_v2 - indices - 1;
     s-op: opposite_pos
         Example: opposite_pos ("banana") = [5, 4, 3, 2, 1, 0] (ints)
```

Small RASP exercises

7. Reverse

Target: flip input sequence, e.g. “abc” ↦ “cba”

Solution: (1) compute opposite position, (2) seek it, (3) get its token

```
>> opposite_pos = length_v2 - indices - 1;
     s-op: opposite_pos
         Example: opposite_pos ("banana") = [5, 4, 3, 2, 1, 0] (ints)
>> sel_flip = select(indices, opposite_pos, ==);
     selector: sel_flip
         Example:
                       b a n a n a
                   b |           1
                   a |         1
                   n |       1
                   a |     1
                   n |   1
                   a | 1
```

Small RASP exercises

7. Reverse

Target: flip input sequence, e.g. “abc” ↦ “cba”

Solution: (1) compute opposite position, (2) seek it, (3) get its token

```
>> opposite_pos = length_v2 - indices - 1;
     s-op: opposite_pos
         Example: opposite_pos ("banana") = [5, 4, 3, 2, 1, 0] (ints)
>> sel_flip = select(indices, opposite_pos, ==);
     selector: sel_flip
         Example:
                          b a n a n a
                      b |           1
                      a |         1
                      n |       1
                      a |     1
                      n |   1
                      a | 1
>> reverse = aggregate(sel_flip, tokens);
     s-op: reverse
         Example: reverse ("banana") = [a, n, a, n, a, b] (strings)
```

Small RASP exercises

Reverse - comparison with trained transformer

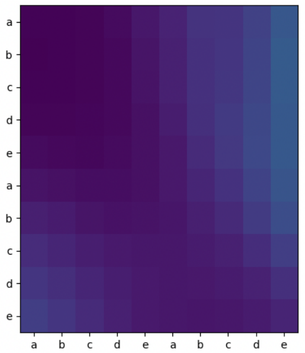
```
>> sel_all = select(1, 1, ==);
     selector: sel_all
         Example:
                          b a n a n a
                      b | 1 1 1 1 1 1
                      a | 1 1 1 1 1 1
                      n | 1 1 1 1 1 1
                      a | 1 1 1 1 1 1
                      n | 1 1 1 1 1 1
                      a | 1 1 1 1 1 1
>> mark_zero = indicator(indices==0);
     s-op: mark_zero
         Example: mark_zero ("banana") = [1, 0, 0, 0, 0, 0] (ints)
>> inverted_length = aggregate(sel_all, mark_zero);
     s-op: inverted_length
         Example: inverted_length ("banana") = [0.167]*6 (floats)
>> length_v2 = round(1/inverted_length);
     s-op: length_v2
         Example: length_v2 ("banana") = [6]*6 (ints)
>> opposite_pos = length_v2 - indices - 1;
     s-op: opposite_pos
         Example: opposite_pos ("banana") = [5, 4, 3, 2, 1, 0] (ints)
>> sel_flip = select(indices, opposite_pos, ==);
     selector: sel_flip
         Example:
                          b a n a n a
                      b |           1
                      a |         1
                      n |       1
                      a |     1
                      n |   1
                      a | 1
>> reverse = aggregate(sel_flip, tokens);
     s-op: reverse
         Example: reverse ("banana") = [a, n, a, n, a, b] (strings)
```

Test:
Training small transformers on lengths 0-100:

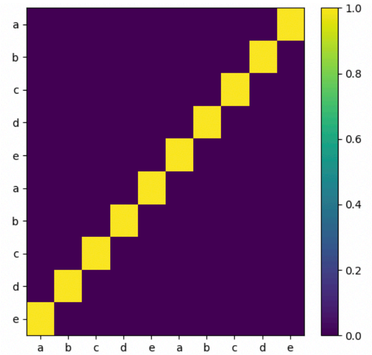
2 layers: **99.6**% accuracy after 20 epochs
1 layer: **39.6**% accuracy after 50 epochs

Bonus: the 2 layer transformer's attention patterns:

Layer 1 (*sel_all*)



Layer 2 (*sel_flip*)



Small RASP exercises

8. Count “a”’s

Target: count “a”’s, e.g. “abc” $\mapsto$ [1, 1, 1]

Small RASP exercises

8. Count “a”’s

Target: count “a”’s, e.g. “abc” ↦ [1, 1, 1]

Solution: (1) focus on all (2) send 1 from “a”’s and 0 from others, (3) average, and multiply by length

Small RASP exercises

8. Count “a”’s

```
>> sel_all = select(1, 1, ==);
     selector: sel_all
         Example:
```

	b	a	n	a	n	a
b	1	1	1	1	1	1
a	1	1	1	1	1	1
n	1	1	1	1	1	1
a	1	1	1	1	1	1
n	1	1	1	1	1	1
a	1	1	1	1	1	1

Target: count “a”’s, e.g. “abc” ↦ [1, 1, 1]

Solution: (1) focus on all (2) send 1 from “a”’s and 0 from others, (3) average, and multiply by length

Small RASP exercises

8. Count “a”’s

```
>> sel_all = select(1, 1, ==);
     selector: sel_all
         Example:
                      b a n a n a
                  b | 1 1 1 1 1 1
                  a | 1 1 1 1 1 1
                  n | 1 1 1 1 1 1
                  a | 1 1 1 1 1 1
                  n | 1 1 1 1 1 1
                  a | 1 1 1 1 1 1
```

Target: count “a”’s, e.g. “abc” ↦ [1, 1, 1]

Solution: (1) focus on all (2) send 1 from “a”’s and 0 from others, (3) average, and multiply by length

```
>> a_indicator = indicator(tokens=="a");
     s-op: a_indicator
         Example: a_indicator ("banana") = [0, 1, 0, 1, 0, 1] (ints)
```

Small RASP exercises

8. Count “a”’s

```
>> sel_all = select(1, 1, ==);
     selector: sel_all
         Example:
```

	b	a	n	a	n	a
b	1	1	1	1	1	1
a	1	1	1	1	1	1
n	1	1	1	1	1	1
a	1	1	1	1	1	1
n	1	1	1	1	1	1
a	1	1	1	1	1	1

Target: count “a”’s, e.g. “abc” ↦ [1, 1, 1]

Solution: (1) focus on all (2) send 1 from “a”’s and 0 from others, (3) average, and multiply by length

```
>> a_indicator = indicator(tokens=="a");
     s-op: a_indicator
         Example: a_indicator ("banana") = [0, 1, 0, 1, 0, 1] (ints)
>> frac_as = aggregate(sel_all, a_indicator);
     s-op: frac_as
         Example: frac_as ("banana") = [0.5]*6 (floats)
```

Small RASP exercises

8. Count “a”’s

```
>> sel_all = select(1, 1, ==);
     selector: sel_all
         Example:
```

	b	a	n	a	n	a
b	1	1	1	1	1	1
a	1	1	1	1	1	1
n	1	1	1	1	1	1
a	1	1	1	1	1	1
n	1	1	1	1	1	1
a	1	1	1	1	1	1

Target: count “a”’s, e.g. “abc” ↦ [1, 1, 1]

Solution: (1) focus on all (2) send 1 from “a”’s and 0 from others, (3) average, and multiply by length

```
>> a_indicator = indicator(tokens=="a");
     s-op: a_indicator
         Example: a_indicator ("banana") = [0, 1, 0, 1, 0, 1] (ints)
>> frac_as = aggregate(sel_all, a_indicator);
     s-op: frac_as
         Example: frac_as ("banana") = [0.5]*6 (floats)
>> num_as = round(frac_as * length_v2);
     s-op: num_as
         Example: num_as ("banana") = [3]*6 (ints)
```

Medium RASP exercises

9. In place histogram

Target: mark each token with its frequency, e.g: [a, b, a] ↦ [2, 1, 2]

Medium RASP exercises

9. In place histogram

Target: mark each token with its frequency, e.g: [a, b, a] ↦ [2, 1, 2]

Solution (v1, like counting “a”’s): (1) look at all tokens, (2) send 1 from token being counted, 0 from others, … ??? wait… different positions are counting different tokens. Who sends 1 and who sends 0?

Medium RASP exercises

9. In place histogram

Target: mark each token with its frequency, e.g: [a, b, a] ↦ [2, 1, 2]

Solution (v1, like counting “a”’s): (1) look at all tokens, (2) send 1 from token being counted, 0 from others, … ??? wait… different positions are counting different tokens. Who sends 1 and who sends 0?

Solution **(v2, like length_v2)**: (1) look at tokens being counted, (2) send 1 from only one position, … ??? wait… which is the one position?

Medium RASP exercises

9. In place histogram

Target: mark each token with its frequency, e.g: [a, b, a] ↦ [2, 1, 2]

Solution (v1, like counting “a”’s): (1) look at all tokens, (2) send 1 from token being counted, 0 from others, … ??? wait… different positions are counting different tokens. Who sends 1 and who sends 0?

Solution **(v2, like length_v2)**: (1) look at tokens being counted, (2) send 1 from only one position, … ??? wait… which is the one position?

can mark first instance of each token as the one position ✅

Medium RASP exercises

9. In place histogram

Target: mark each token with its frequency, e.g: [a, b, a] ↦ [2, 1, 2]

Solution (v1, like counting “a”’s): (1) look at all tokens, (2) send 1 from token being counted, 0 from others, … ??? wait… different positions are counting different tokens. Who sends 1 and who sends 0?

Solution **(v2, like length_v2)**: (1) look at tokens being counted, (2) send 1 from only one position, … ??? wait… which is the one position?

can mark first instance of each token as the one position ✅

but there’s something more generalisable…!

Medium RASP exercises

9. In place histogram

Target: mark each token with its frequency, e.g: [a, b, a] ↦ [2, 1, 2]

Solution (v1, like counting "a"'s): (1) look at all tokens, (2) send 1 from token being counted, 0 from others, … ??? wait… different positions are counting different tokens. Who sends 1 and who sends 0?

Solution **(v2, like length_v2)**: (1) look at tokens being counted, (2) send 1 from only one position, … ??? wait… which is the one position?

can mark first instance of each token as the one position ✅

but there's something more generalisable…!

Force 0 as the one position, and correct for it after (singular check at pos. 0)

Medium RASP exercises

9. In place histogram

Target: mark each token with its frequency, e.g: [a, b, a] ↦ [2, 1, 2]

Solution:

(1a) look at same tokens,

(1b) AND at pos. 0,

(2) send 1 only from 0,

(3) average and invert,

(4) get token at 0,

(5) correct for pos. 0

Medium RASP exercises

9. In place histogram

Target: mark each token with its frequency, e.g: [a, b, a] ↦ [2, 1, 2]

Solution:

(1a) look at same tokens,

(1b) AND at pos. 0,

(2) send 1 only from 0,

(3) average and invert,

(4) get token at 0,

(5) correct for pos. 0

```
>> same      = select(tokens, tokens, ==)
     selector: same
         Example:
                          b a n a n a
                      b | 1
                      a |   1   1   1
                      n |     1   1
                      a |   1   1   1
                      n |     1   1
                      a |   1   1   1
```

Medium RASP exercises

9. In place histogram

Target: mark each token with its frequency, e.g: [a, b, a] ↦ [2, 1, 2]

Solution:

(1a) look at same tokens,

(1b) AND at pos. 0,

(2) send 1 only from 0,

(3) average and invert,

(4) get token at 0,

(5) correct for pos. 0

```
>> same_or_0 = select(tokens, tokens, ==) or select(indices, 0, ==);
     selector: same_or_0
         Example:
                          b a n a n a
                      b | 1
                      a | 1 1   1   1
                      n | 1   1   1
                      a | 1 1   1   1
                      n | 1   1   1
                      a | 1 1   1   1
```

Medium RASP exercises

9. In place histogram

Target: mark each token with its frequency, e.g: [a, b, a] ↦ [2, 1, 2]

Solution:

(1a) look at same tokens,

(1b) AND at pos. 0,

(2) send 1 only from 0,

(3) average and invert,

(4) get token at 0,

(5) correct for pos. 0

```
>> same_or_0 = select(tokens, tokens, ==) or select(indices, 0, ==);
      selector: same_or_0
          Example:
```

	b	a	n	a	n	a
b	1					
a	1	1		1		1
n	1		1		1	
a	1	1		1		1
n	1		1		1	
a	1	1		1		1

```
indicator(indices==0)
```

Medium RASP exercises

9. In place histogram

Target: mark each token with its frequency, e.g: [a, b, a] ↦ [2, 1, 2]

Solution:

(1a) look at same tokens,

(1b) AND at pos. 0,

(2) send 1 only from 0,

(3) average and invert,

(4) get token at 0,

(5) correct for pos. 0

```
>> same_or_0 = select(tokens, tokens, ==) or select(indices, 0, ==);
    selector: same_or_0
        Example:
                        b a n a n a
                    b | 1
                    a | 1 1   1   1
                    n | 1   1   1
                    a | 1 1   1   1
                    n | 1   1   1
                    a | 1 1   1   1
>> inverse_with_0 = aggregate(same_or_0, indicator(indices==0));
    s-op: inverse_with_0
        Example: inverse_with_0 ("banana") = [1, 0.25, 0.333, 0.25, 0.333, 0.25] (floats)
```

Medium RASP exercises

9. In place histogram

Target: mark each token with its frequency, e.g: [a, b, a] ↦ [2, 1, 2]

Solution:

(1a) look at same tokens,

(1b) AND at pos. 0,

(2) send 1 only from 0,

(3) average and invert,

(4) get token at 0,

(5) correct for pos. 0

```
>> same_or_0 = select(tokens, tokens, ==) or select(indices, 0, ==);
    selector: same_or_0
        Example:
                          b a n a n a
                      b | 1
                      a | 1 1   1   1
                      n | 1   1   1
                      a | 1 1   1   1
                      n | 1   1   1
                      a | 1 1   1   1
>> inverse_with_0 = aggregate(same_or_0, indicator(indices==0));
    s-op: inverse_with_0
        Example: inverse_with_0 ("banana") = [1, 0.25, 0.333, 0.25, 0.333, 0.25] (floats)
>> hist_with_0 = round(1/inverse_with_0);
    s-op: hist_with_0
        Example: hist_with_0 ("banana") = [1, 4, 3, 4, 3, 4] (ints)
```

Medium RASP exercises

9. In place histogram

Target: mark each token with its frequency, e.g: [a, b, a] ↦ [2, 1, 2]

Solution:

(1a) look at same tokens,

(1b) AND at pos. 0,

(2) send 1 only from 0,

(3) average and invert,

(4) get token at 0,

(5) correct for pos. 0

```
>> same_or_0 = select(tokens, tokens, ==) or select(indices, 0, ==);
    selector: same_or_0
        Example:
                          b a n a n a
                      b | 1
                      a | 1 1   1   1
                      n | 1   1   1
                      a | 1 1   1   1
                      n | 1   1   1
                      a | 1 1   1   1
>> inverse_with_0 = aggregate(same_or_0, indicator(indices==0));
    s-op: inverse_with_0
        Example: inverse_with_0 ("banana") = [1, 0.25, 0.333, 0.25, 0.333, 0.25] (floats)
>> hist_with_0 = round(1/inverse_with_0);
    s-op: hist_with_0
        Example: hist_with_0 ("banana") = [1, 4, 3, 4, 3, 4] (ints)
>> val_at_0 = aggregate(select(indices, 0, ==), tokens);
    s-op: val_at_0
        Example: val_at_0 ("banana") = [b]*6 (strings)
```

Medium RASP exercises

9. In place histogram

Target: mark each token with its frequency, e.g: [a, b, a] ↦ [2, 1, 2]

Solution:

(1a) look at same tokens,

(1b) AND at pos. 0,

(2) send 1 only from 0,

(3) average and invert,

(4) get token at 0,

(5) correct for pos. 0

```
>> same_or_0 = select(tokens, tokens, ==) or select(indices, 0, ==);
    selector: same_or_0
        Example:
                          b a n a n a
                      b | 1
                      a | 1 1   1   1
                      n | 1   1   1
                      a | 1 1   1   1
                      n | 1   1   1
                      a | 1 1   1   1
>> inverse_with_0 = aggregate(same_or_0, indicator(indices==0));
    s-op: inverse_with_0
        Example: inverse_with_0 ("banana") = [1, 0.25, 0.333, 0.25, 0.333, 0.25] (floats)
>> hist_with_0 = round(1/inverse_with_0);
    s-op: hist_with_0
        Example: hist_with_0 ("banana") = [1, 4, 3, 4, 3, 4] (ints)
>> val_at_0 = aggregate(select(indices, 0, ==), tokens);
    s-op: val_at_0
        Example: val_at_0 ("banana") = [b]*6 (strings)
>> histogram = hist_with_0 - 1 + indicator(tokens==val_at_0);
    s-op: histogram
        Example: histogram ("banana") = [1, 3, 2, 3, 2, 3] (ints)
```

Medium RASP exercises

10. Selector width

Target: compute how many positions are chosen in each row of a selector

Medium RASP exercises

10. Selector width

Target: compute how many positions are chosen in each row of a selector

Solution: generalisation of histogram solution:

(1a) look at ~~same tokens~~ given selector,

(1b) AND at pos. 0,

(2) send 1 only from 0,

(3) average and invert,

(4) ~~get token at 0~~

get 1 iff selector hits 0,

(5) correct for pos. 0

Medium RASP exercises

10. Selector width

Target: compute how many positions are chosen in each row of a selector

Solution: generalisation of histogram solution:

(1a) look at ~~same tokens~~ given selector,

(1b) AND at pos. 0,

(2) send 1 only from 0,

(3) average and invert,

(4) ~~get token at 0~~

get 1 iff selector hits 0,

(5) correct for pos. 0

Library function in RASP

```
>> histogram = selector_width(select(tokens, tokens, ==));
     s-op: histogram
         Example: histogram("banana") = [1, 3, 2, 3, 2, 3] (ints)
>> count_as = selector_width(select(tokens,"a",==));
     s-op: count_as
         Example: count_as("banana") = [3]*6 (ints)
>> running_histogram = selector_width(select(tokens,tokens,==) and select(indices,indices,<=));
     s-op: running_histogram
         Example: running_histogram("banana") = [1, 1, 1, 2, 2, 3] (ints)
```

https://github.com/tech-srl/RASP/blob/main/RASP_support/rasplib.rasp

Medium RASP exercises

11. Sorting

Target: sort arbitrary sequence of values, e.g.

dkhs → dhks

Solution:

Medium RASP exercises

11. Sorting

Target: sort arbitrary sequence of values, e.g.

dkhs → dhks

Solution: each token finds all tokens smaller than itself, input position is used as a tie-breaker. Counting these gives us that token's order, i.e., its final position in the sorted sequence

Medium RASP exercises

11. Sorting

Target: sort arbitrary sequence of values, e.g.

dkhs → dhks

Solution: each token finds all tokens smaller than itself, input position is used as a tie-breaker. Counting these gives us that token's order, i.e., its final position in the sorted sequence

```
>> sel_smaller = select(tokens, tokens, <);
    selector: sel_smaller
        Example:
                        b a n a n a
                    b |   1   1   1
                    a |
                    n | 1 1   1   1
                    a |
                    n | 1 1   1   1
                    a |
```

Medium RASP exercises

11. Sorting

Target: sort arbitrary sequence of values, e.g.

dkhs → dhks

Solution: each token finds all tokens smaller than itself, input position is used as a tie-breaker. Counting these gives us that token's order, i.e., its final position in the sorted sequence

```
>> sel_smaller = select(tokens, tokens, <);
    selector: sel_smaller
        Example:
                        b a n a n a
                    b |   1   1   1
                    a |
                    n | 1 1   1   1
                    a |
                    n | 1 1   1   1
                    a |
>> sel_earlier_equal = select(tokens, tokens, ==) and select(indices, indices, <);
    selector: sel_earlier_equal
        Example:
                        b a n a n a
                    b |
                    a |
                    n |
                    a |   1
                    n |     1
                    a |   1   1
```

Medium RASP exercises

11. Sorting

Target: sort arbitrary sequence of values, e.g.

dkhs → dhks

Solution: each token finds all tokens smaller than itself, input position is used as a tie-breaker. Counting these gives us that token's order, i.e., its final position in the sorted sequence

```
>> sel_smaller = select(tokens, tokens, <);
    selector: sel_smaller
        Example:
                          b a n a n a
                      b |   1   1   1
                      a |
                      n | 1 1   1   1
                      a |
                      n | 1 1   1   1
                      a |
>> sel_earlier_equal = select(tokens, tokens, ==) and select(indices, indices, <);
    selector: sel_earlier_equal
        Example:
                          b a n a n a
                      b |
                      a |
                      n |
                      a |   1
                      n |     1
                      a |   1   1
>> order = selector_width(sel_smaller or sel_earlier_equal);
    s-op: order
        Example: order("banana") = [3, 0, 4, 1, 5, 2] (ints)
```

Medium RASP exercises

11. Sorting

Target: sort arbitrary sequence of values, e.g.

dkhs → dhks

Solution: each token finds all tokens smaller than itself, input position is used as a tie-breaker. Counting these gives us that token's order, i.e., its final position in the sorted sequence

```
>> sel_smaller = select(tokens, tokens, <);
     selector: sel_smaller
         Example:
                           b a n a n a
                       b |   1   1   1
                       a |
                       n | 1 1   1   1
                       a |
                       n | 1 1   1   1
                       a |
>> sel_earlier_equal = select(tokens, tokens, ==) and select(indices, indices, <);
     selector: sel_earlier_equal
         Example:
                           b a n a n a
                       b |
                       a |
                       n |
                       a |   1
                       n |     1
                       a |   1   1
>> order = selector_width(sel_smaller or sel_earlier_equal);
     s-op: order
         Example: order("banana") = [3, 0, 4, 1, 5, 2] (ints)
>> sorted = aggregate(select(order, indices, ==), tokens);
     s-op: sorted
         Example: sorted("banana") = [a, a, a, b, n, n] (strings)
```

Medium RASP exercises

11. Sorting

Target: sort arbitrary sequence of values, e.g.

dkhs → dhks

Solution: each token finds all tokens smaller than itself, input position is used as a tie-breaker. Counting these gives us that token's order, i.e., its final position in the sorted sequence

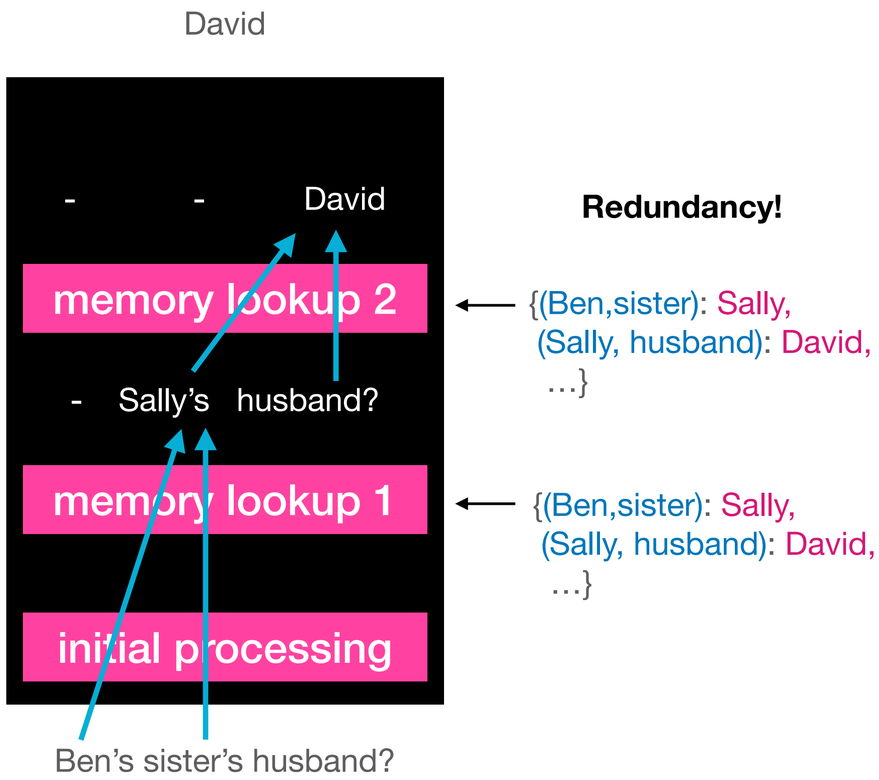
```
>> sel_smaller = select(tokens, tokens, <);
     selector: sel_smaller
         Example:
                          b a n a n a
                      b |   1   1   1
                      a |
                      n | 1 1   1   1
                      a |
                      n | 1 1   1   1
                      a |
>> sel_earlier_equal = select(tokens, tokens, ==) and select(indices, indices, <);
     selector: sel_earlier_equal
         Example:
                          b a n a n a
                      b |
                      a |
                      n |
                      a |   1
                      n |     1
                      a |   1   1
>> order = selector_width(sel_smaller or sel_earlier_equal);
     s-op: order
         Example: order("banana") = [3, 0, 4, 1, 5, 2] (ints)
>> sorted = aggregate(select(order, indices, ==), tokens);
     s-op: sorted
         Example: sorted("banana") = [a, a, a, b, n, n] (strings)
```

What does this mean for linear transformers?

Multi Hop Reasoning

High level discussion

In parameters



Redundancy!

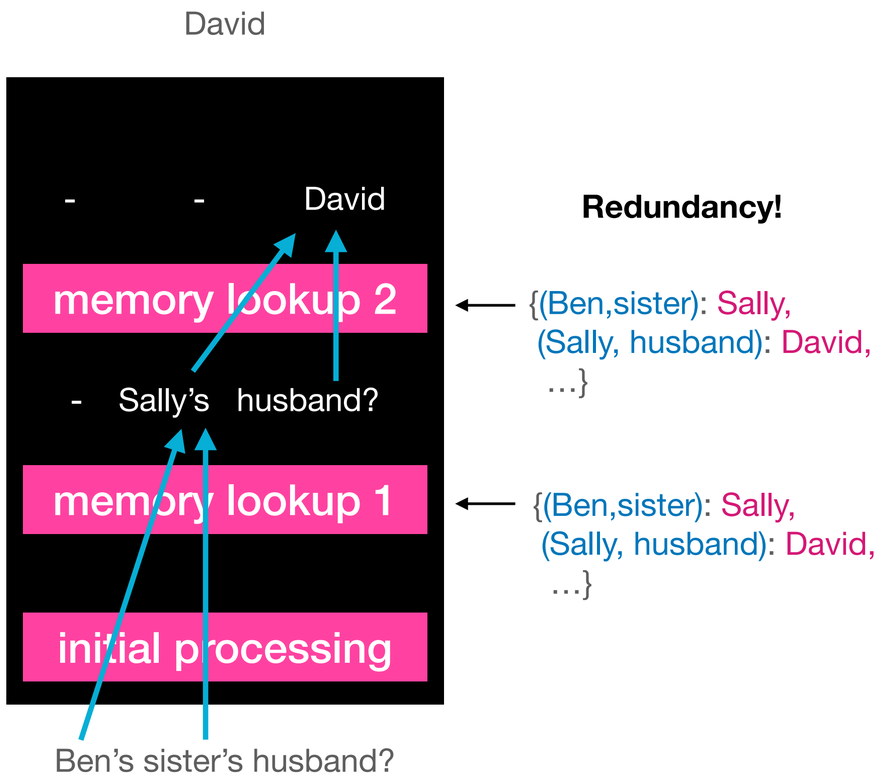
{(Ben,sister): Sally,
(Sally, husband): David,
…}

{(Ben,sister): Sally,
(Sally, husband): David,
…}

Multi Hop Reasoning

High level discussion

In parameters



Redundancy!

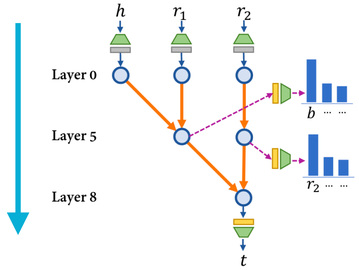
{(Ben,sister): Sally,
(Sally, husband): David,
…}

{(Ben,sister): Sally,
(Sally, husband): David,
…}

"Circuit" observed:

Grokked transformers are implicit reasoners: a mechanistic journey to the edge of generalization

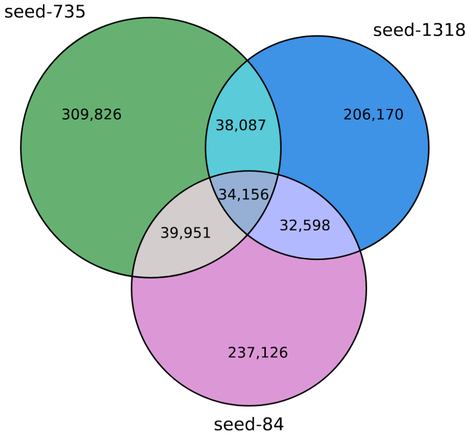
Wang et al, 2024



Redundancy observed:

Discovering knowledge-critical subnetworks in pretrained language models

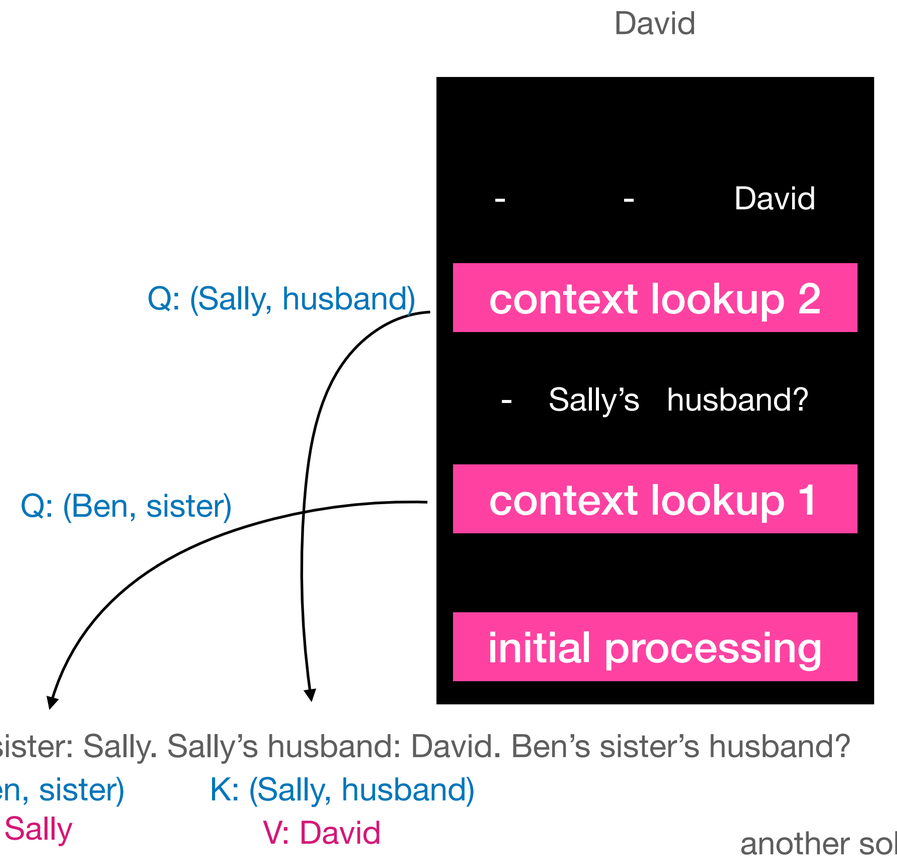
Bayazit et al, 2024



Multi Hop Reasoning

High level discussion

In context

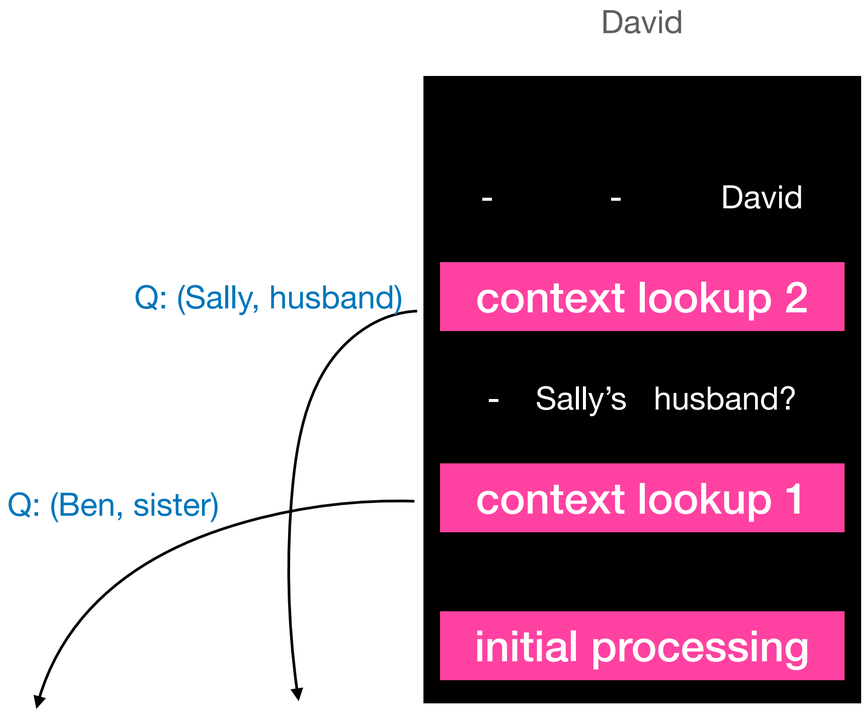


another solution (builds up descriptions in place): https://github.com/tech-srl/RASP/blob/main/rover.rasp

Multi Hop Reasoning

High level discussion

In context



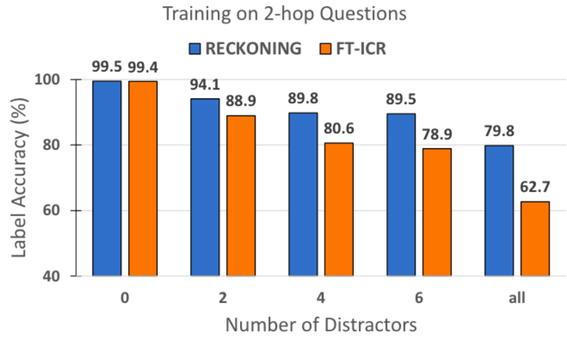
Lot of detail needed in the queries and keys…

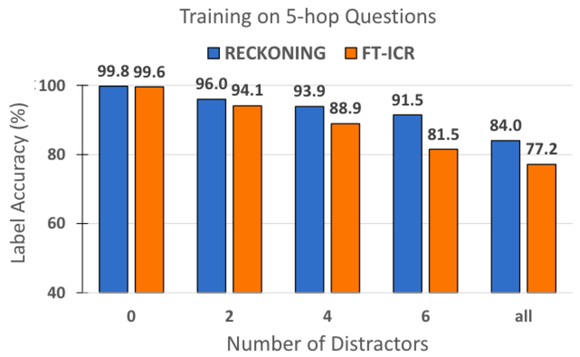
Makes finding relevant facts and rules harder?

Benefits of memory over context:

RECKONING: Reasoning through dynamic knowledge encoding

Chen et al, 2023





another solution (builds up descriptions in place): https://github.com/tech-srl/RASP/blob/main/rover.rasp

Long addition walkthrough:

Thinking Like Transformers
ICLR 2023 Blog Track
🌟 https://srush.github.io/raspy/ 🌟

End

RASP REPL
github.com/tech-srl/RASP

(or email me if you can't get on github)

RASP itself:
Thinking Like Transformers
ICML 2021

Tracr (Partial RASP compiler)
Lindner et al, NeurIPS 2023

Learning Transformer Programs
Friedman et al, NeurIPS 2023