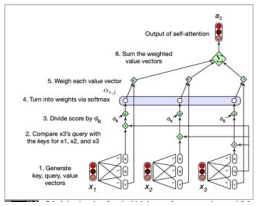
Back to Cats



- Embedding tokens.

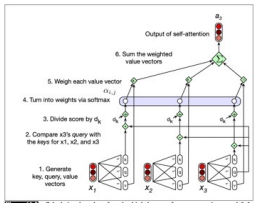
  More realistically, tokens from prior processing.

  X = [[1.0, 0.5, 0.2, 0.1], ← Token 1: "The"
  [0.5, 1.0, 0.3, 0.2], ← Token 2: "cat"
  [0.3, 0.2, 1.0, 0.5]] **← Token 3: "sat"**

  We follow "sat" (token 3) as the query token and ask how much it attends to "cat" (token 2).

1

Back to Cats



- Create Q, K, V with Different Weight Matrices

W_Q (transforms X to "what am I looking for?"):
W_Q = [[1.0, 0.0, 0.5, 0.0],
[0.0, 1.0, 0.0, 0.5],
[0.5, 0.0, 1.0, 0.0],
[0.0, 0.5, 0.0, 1.0]]

W_K (transforms X to "what do I offer?"):
W_K = [[1.0, 0.2, 0.0, 0.0],
[0.2, 1.0, 0.0, 0.0],
[0.0, 0.0, 1.0, 0.3],
[0.0, 0.0, 0.3, 1.0]]

W_V (transforms X to "what information will I contribute?"):

W_V = [[0.8, 0.0, 0.0, 0.1],
[0.0, 0.9, 0.1, 0.0],
[0.0, 0.1, 0.8, 0.0],
[0.1, 0.0, 0.0, 0.9]]

2

Back to Cats

- Computing q = X @ W_Q

For token 3: “sat” =

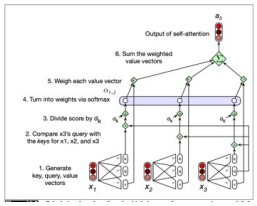
[0.3, 0.2, 1.0, 0.5] @ [[1.0, 0.0, 0.5, 0.0],
[0.0, 1.0, 0.0, 0.5],
[0.5, 0.0, 1.0, 0.0],
[0.0, 0.5, 0.0, 1.0]]

[0.3×1.0 + 0.2×0.0 + 1.0×0.5 + 0.5×0.0,
0.3×0.0 + 0.2×1.0 + 1.0×0.0 + 0.5×0.5,
0.3×0.5 + 0.2×0.0 + 1.0×1.0 + 0.5×0.0,
0.3×0.0 + 0.2×0.5 + 1.0×0.0 + 0.5×1.0]

= [0.8, 0.45, 1.15, 0.6]

“The”: [1.1, 0.55, 0.7, 0.35]

“cat”: [0.65, 1.1, 0.55, 0.7]



3

Back to Cats

- Computing k = X @ W_K

For token 2: “cat” =

[0.5, 1.0, 0.3, 0.2] @ [[1.0, 0.2, 0.0, 0.0],
[0.2, 1.0, 0.0, 0.0],
[0.0, 0.0, 1.0, 0.3],
[0.0, 0.0, 0.3, 1.0]]

[0.5×1.0 + 1.0×0.2 + 0.3×0.0 + 0.2×0.0,
0.5×0.2 + 1.0×1.0 + 0.3×0.0 + 0.2×0.0,
0.5×0.0 + 1.0×0.0 + 0.3×1.0 + 0.2×0.3,
0.5×0.0 + 1.0×0.0 + 0.3×0.3 + 0.2×1.0]

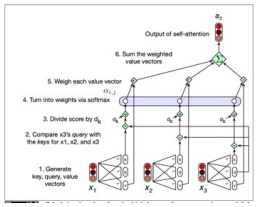
= [0.7, 1.1, 0.36, 0.29]

“The”: [1.1, 0.7, 0.23, 0.16]

“cat”: [0.7, 1.1, 0.36, 0.29]

“sat”: [0.34, 0.26, 1.15, 0.8]

K = [[1.1, 0.7, 0.23, 0.16],
[0.7, 1.1, 0.36, 0.29],
[0.34, 0.26, 1.15, 0.8]]



4

Back to Cats

- Computing v = X @ W_V

For token 2: “cat” =

[0.5, 1.0, 0.3, 0.2] @ [[0.8, 0.0, 0.0, 0.1],
[0.0, 0.9, 0.1, 0.0],
[0.0, 0.1, 0.8, 0.0],
[0.1, 0.0, 0.0, 0.9]]

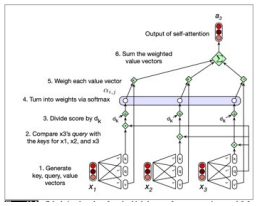
[0.5×0.8 + 1.0×0.0 + 0.3×0.0 + 0.2×0.1,
0.5×0.0 + 1.0×0.9 + 0.3×0.1 + 0.2×0.0,
0.5×0.0 + 1.0×0.1 + 0.3×0.8 + 0.2×0.0,
0.5×0.1 + 1.0×0.0 + 0.3×0.0 + 0.2×0.9]

= [0.42, 0.93, 0.34, 0.23]

“The”: [0.81, 0.47, 0.21, 0.19]

“cat”: [0.42, 0.93, 0.34, 0.23]

“sat”: [0.29, 0.28, 0.82, 0.48]



5

Back to Cats

- Calculating attention scores for “sat”: q · K

(dot product of q with each row of K)

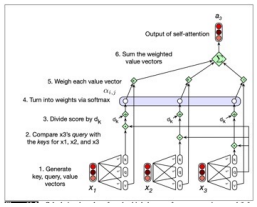
q for “sat” = [0.8, 0.45, 1.15, 0.6]

“sat” & “The”: 0.8×1.1 + 0.45×0.7 + 1.15×0.23 + 0.6×0.16 =
0.88 + 0.315 + 0.265 + 0.096 = 1.556

**“sat” & “cat”: 0.8×0.7 + 0.45×1.1 + 1.15×0.36 + 0.6×0.29 =
0.56 + 0.495 + 0.414 + 0.174 = 1.643**

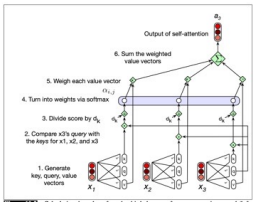
“sat” & “sat”: 0.8×0.34 + 0.45×0.26 + 1.15×1.15 + 0.6×0.8 =
0.272 + 0.117 + 1.323 + 0.48 = 2.192

K = [[1.1, 0.7, 0.23, 0.16],
[0.7, 1.1, 0.36, 0.29],
[0.34, 0.26, 1.15, 0.8]]



6

Back to Cats



- Scaling, i.e. divide by square root of dimensionality of key.

  Dimensionality = 4, so divide by 2.

  "sat" scores: [1.556/2, 1.643/2, 2.192/2] =

  [0.778, 0.822, 1.096]

- Softmax: exponentiate each scaled score

  "The": exp(0.778) = 2.177

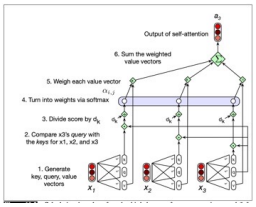
  "cat": exp(0.822) = 2.275

  "sat": exp(1.096) = 2.992

  Sum = 2.177 + 2.275 + 2.992 = 7.444

7

Back to Cats



- Softmax: divide each exponent by the sum

  "The": 2.177 / 7.444 = 0.292

  "cat": 2.275 / 7.444 = 0.306

  "sat": 2.992 / 7.444 = 0.402

  Attention weights for "sat": [0.292, 0.306, 0.402]

- Interpretation: "sat" pays

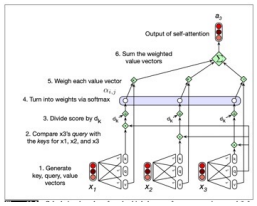
  29.2% attention to "The"

  30.6% attention to "cat"

  40.2% attention to itself

8

Back to Cats



- Output for “sat”

$0.292 \times V^{\text{“The”}} + 0.306 \times V^{\text{“cat”}} + 0.402 \times V^{\text{“sat”}} =$

$0.292 \times [0.81, 0.47, 0.21, 0.19] +$

$0.306 \times [0.42, 0.93, 0.34, 0.23] +$

$0.402 \times [0.29, 0.28, 0.82, 0.48]$

$= [0.237, 0.137, 0.061, 0.055] +$

$[0.129, 0.285, 0.104, 0.07] +$

$[0.117, 0.113, 0.33, 0.193]$

$= [0.483, 0.535, 0.495, 0.318]$

- This is the context-aware output vector for “sat”
- It has been enriched with weighted information from “The”, “cat” and “sat”.