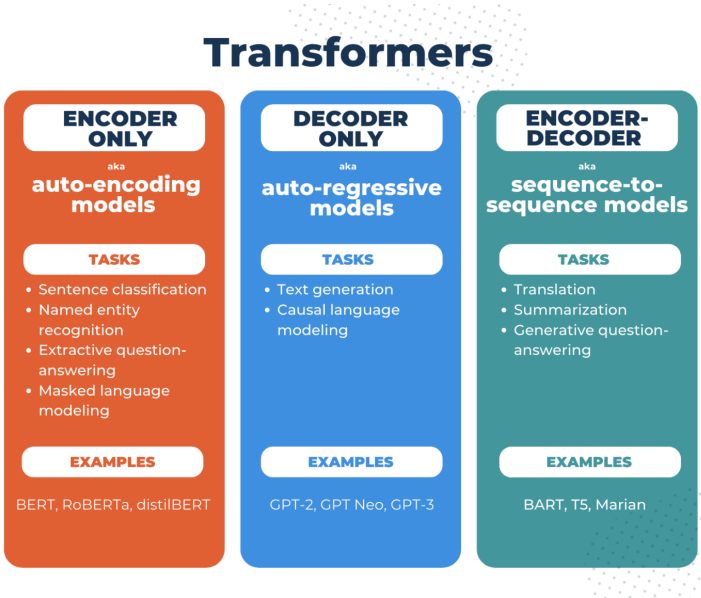
Transformers – Part 2

Excerpts from Chapter 10 of
Speech and Language Processing,
Jurafsky and Martin, Aug. 20, 2024 draft as well as some assistance from
Claude Sonnet 4.5
Michael Wollowski

1

Transformers



Source: https://www.comet.com/site/blog/explainable-ai-for-transformers/

2

Reminder: Dot Product

- Definition:

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i$$

- Outcome: Can be used to calculate the similarity between two vectors.

4

Bird's Eye View of Attention in Transformers

- Starting at the bottom, we calculate the dot product between:
  - X_3 and X_3
  - X_3 and X_2
  - X_3 and X_1
- This should give us a similarity of X_3 to the prior tokens, X_1, X_2 and X_3.

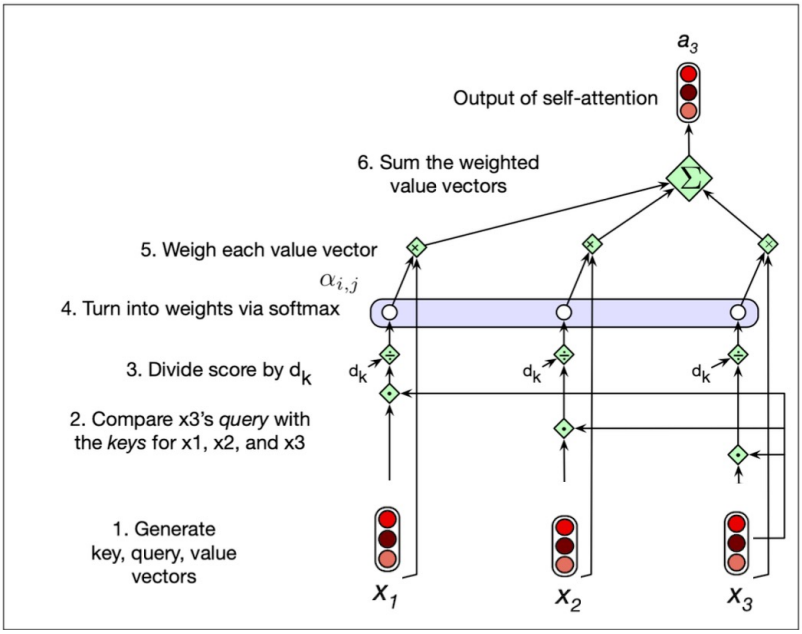


Image source: Modified from Speech and Language Processing, Jurafsky and Martin, Feb. 3, 2024 draft

5

Bird's Eye View of Attention in Transformers

- Next, we normalize the values of the dot product.
- Otherwise, the values may be quite large and impede training, due to loss of gradients.
- We divide by the square root of the dimensionality of the key vector, d_k

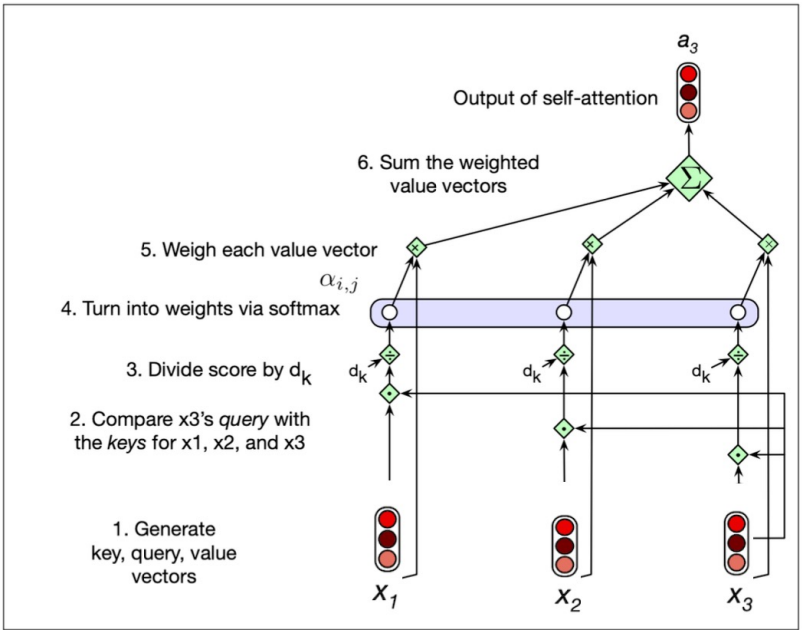


Image source: Modified from Speech and Language Processing, Jurafsky and Martin, Feb. 3, 2024 draft

6

Bird's Eye View of Attention in Transformers

- Now, we run the values through softmax to obtain weights.
- These weights are now used to determine the relevance of each of X_1 to X_3
- Finally, we produce the weighted sum with the X_1, X_2, and X_3.

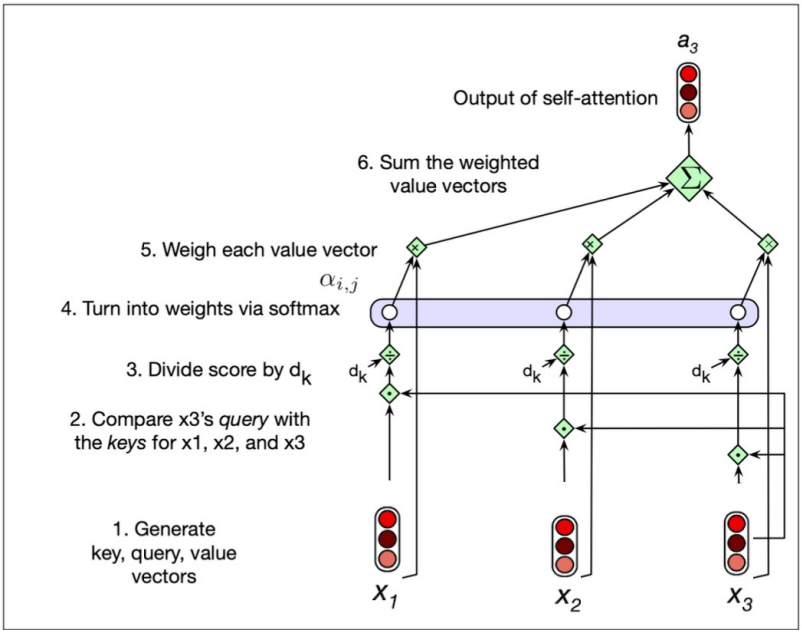


Image source: Modified from Speech and Language Processing, Jurafsky and Martin, Feb. 3, 2024 draft

7

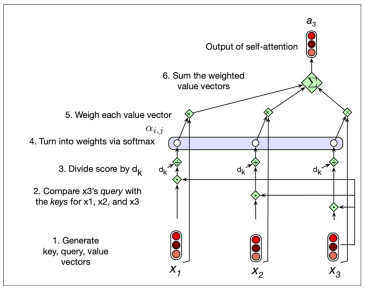
Reminder: Softmax

- Definition: $$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

- The softmax function takes as input a vector and:
  - turns each component into an interval (0,1)
  - the components will add up to 1,
  - they can be interpreted as probabilities

8

A Sitting Example

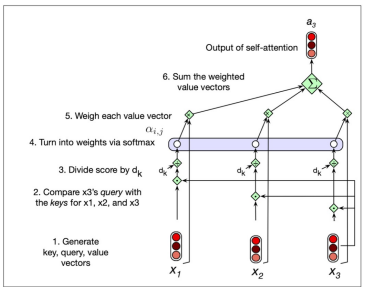


- Consider the token sequence: “The cat sat <end>”
- 4 tokens
- Embedding dimension: 4
- Step 1: Produce input embeddings:

  Token 1 "The": [1.0, 0.5, 0.2, 0.1]
  Token 2 "cat": [0.5, 1.0, 0.3, 0.2]
  Token 3 "sat": [0.3, 0.2, 1.0, 0.5]
  Token 4 "<end>": [0.1, 0.1, 0.1, 1.0]

9

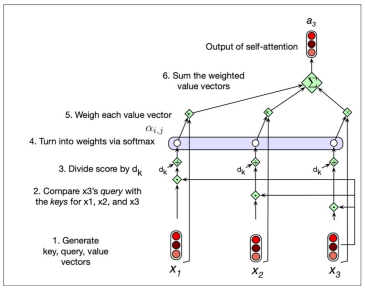
A Sitting Example



- Step 2: Determine Attention Score of "The":
  - Token 1 attending to Token 1:
    [1.0, 0.5, 0.2, 0.1] · [1.0, 0.5, 0.2, 0.1] = 1.0 + 0.25 + 0.04 + 0.01 = **1.30**
  - Token 1 attending to Token 2:
    [1.0, 0.5, 0.2, 0.1] · [0.5, 1.0, 0.3, 0.2] = 0.5 + 0.5 + 0.06 + 0.02 = 1.08
  - Token 1 attending to Token 3:
    [1.0, 0.5, 0.2, 0.1] · [0.3, 0.2, 1.0, 0.5] = 0.3 + 0.1 + 0.2 + 0.05 = 0.65
  - Token 1 attending to Token 4:
    [1.0, 0.5, 0.2, 0.1] · [0.1, 0.1, 0.1, 1.0] = 0.1 + 0.05 + 0.02 + 0.1 = 0.27

10

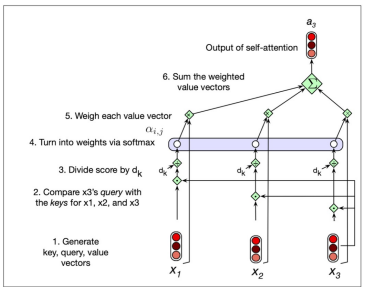
A Sitting Example



- Step 3: Normalize
  - Divide each score by dk, which is the square root of the dimensionality.
  - Dimensionality = 4.
  - Hence divide by 2.
  - Token 1 attending to Token 1: 1.30 / 2 = 0.65
  - Token 1 attending to Token 2: 1.08 / 2 = 0.54
  - Token 1 attending to Token 3: 0.65 / 2 = 0.325
  - Token 1 attending to Token 4: 0.27 / 2 = 0.135

11

A Sitting Example



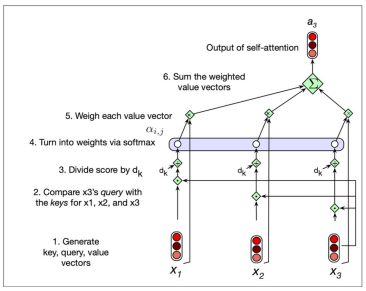
- Step 4: Apply Softmax
  - exp(0.65) = 1.916
  - exp(0.54) = 1.716
  - exp(0.325) = 1.384
  - exp(0.135) = 1.145
  - Sum = 6.161
- Attention weights

  [1.916/6.161, 1.716/6.161, 1.384/6.161, 1.145/6.161] = [0.311, 0.278, 0.225, 0.186]
- These weights tell us:

  Token 1 pays 31.1% attention to itself, 27.8% to "cat", 22.5% to "sat", and 18.6% to "<end>".

12

A Sitting Example



- Step 5: Output for token 1 (”The”)

  0.311 × [1.0, 0.5, 0.2, 0.1] ("The")
  + 0.278 × [0.5, 1.0, 0.3, 0.2] ("cat")
  + 0.225 × [0.3, 0.2, 1.0, 0.5] ("sat")
  + 0.186 × [0.1, 0.1, 0.1, 1.0] ("<end>")

  = [0.311, 0.156, 0.062, 0.031]
  + [0.139, 0.278, 0.083, 0.056]
  + [0.068, 0.045, 0.225, 0.113]
  + [0.019, 0.019, 0.019, 0.186]

  = [0.537, 0.498, 0.389, 0.386]

- This output vector for "The" is now context-aware.
- It contains information weighted from all tokens in the sequence.

13

Transformers – for reals

- We should add some weights.
- After all that is what NNs are all about.
- We want to have a trainable distance measure, rather than a static one.

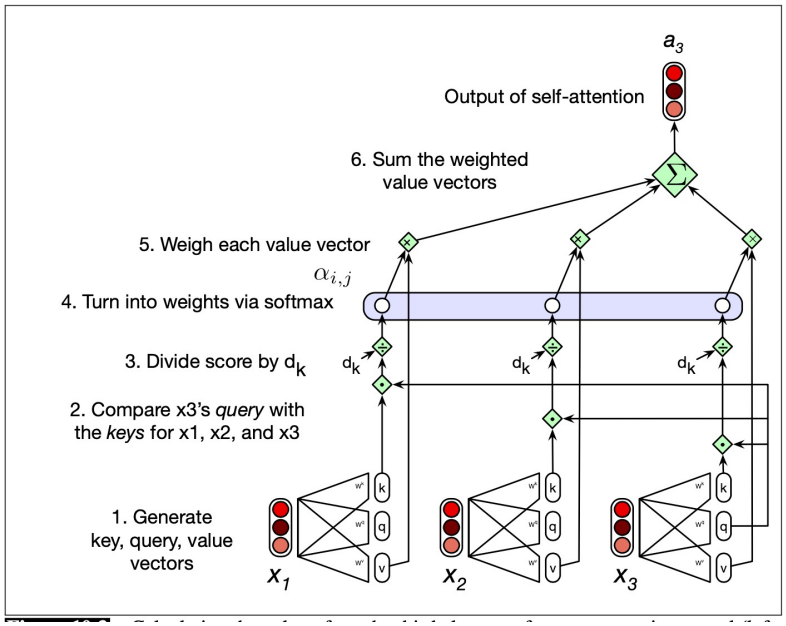


Image source: Speech and Language Processing, Jurafsky and Martin, Feb. 3, 2024 draft

14

Transformers – for reals

- Each X_i is run through a different weight matrix to produce:
  - A Key, **k**
  - A Query, **q** and
  - A Value, **v**

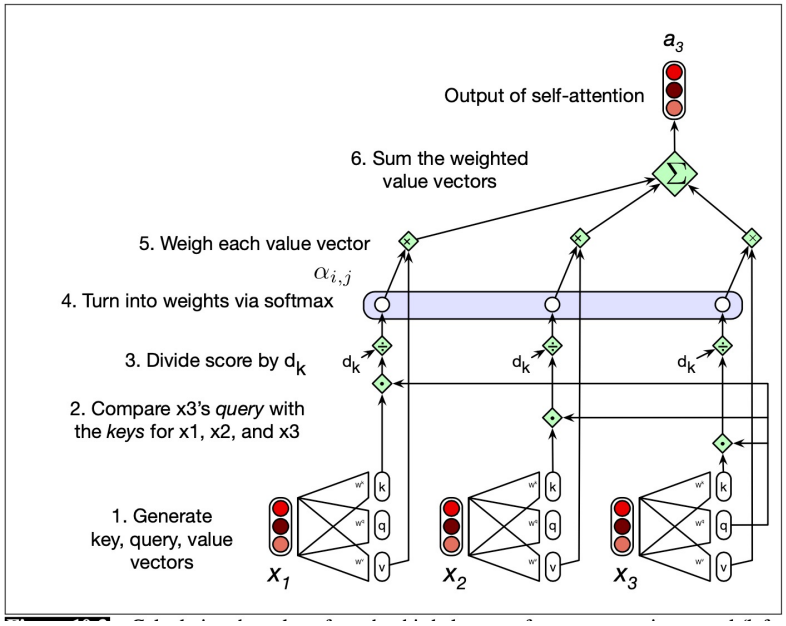


Image source: Speech and Language Processing, Jurafsky and Martin, Feb. 3, 2024 draft

15

Keys, Queries and Values

- Consider the phrase: “The cat sat <end>”
- Suppose we process the token “cat”
- What do we wish to learn about the cat?
- One thing: what is the cat doing?
- In this case, it is sitting.

16

Keys, Queries and Values

- Consider the phrase: “The cat sat <end>”
- When processing the above phrase, we wish to learn more about the token ”cat”, we say that we issue a **query** about “cat”.
- Each word in the phrase provides some more or less relevant information.
- The token “sat” is a verb and would provide its verbness as a **key**.
- the attention mechanism provides information, called the **value**.
- In our example, the value is that “sat” provides semantic content for “cat”.

17

Transformers – for reals

- In the diagram, the attention is determined for X_3.
- The **q**uery of X_3 is multiplied with the **k**ey of X_1, X_2, and X_3.
- Output a_3, is the product of the **v**alue of X_3, with the softmax weights.

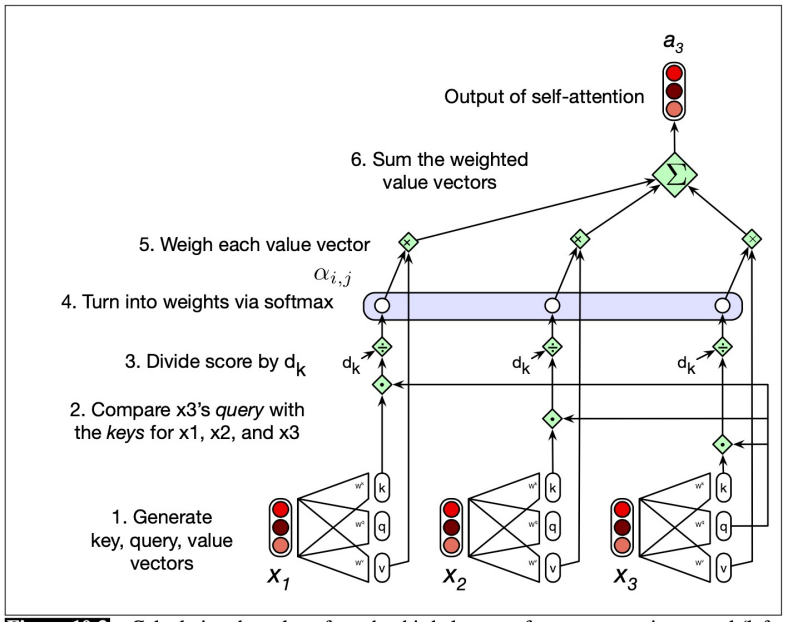
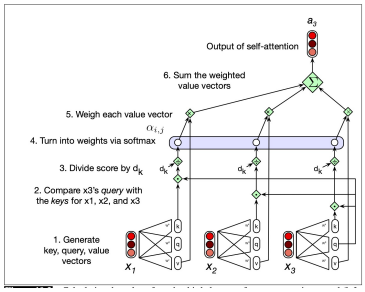


Image source: Speech and Language Processing, Jurafsky and Martin, Feb. 3, 2024 draft

18

Back to Cats



Embedding tokens.

More realistically, tokens from prior processing.

X = [[1.0, 0.5, 0.2, 0.1], ← Token 1: "The"

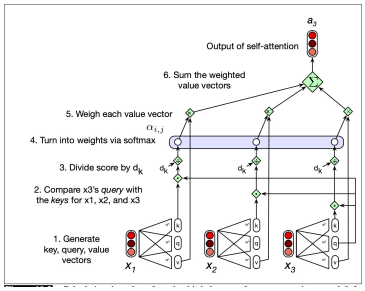
[0.5, 1.0, 0.3, 0.2], ← Token 2: "cat"

[0.3, 0.2, 1.0, 0.5], ← Token 3: "sat"

[0.1, 0.1, 0.1, 1.0]] ← Token 4: "<end>"

19

Back to Cats



- Create Q, K, V with Different Weight Matrices

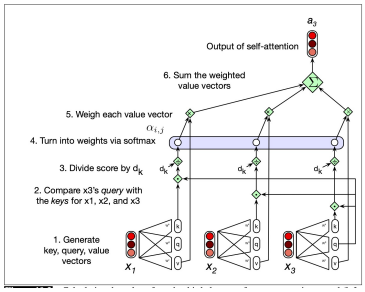
W_Q (transforms X to "what am I looking for?"):
W_Q = [[1.0, 0.0, 0.5, 0.0],
[0.0, 1.0, 0.0, 0.5],
[0.5, 0.0, 1.0, 0.0],
[0.0, 0.5, 0.0, 1.0]]

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]]

20

Back to Cats



- Computing q = X @ W_Q

For token 1: "The" =
[1.0, 0.5, 0.2, 0.1] @ [[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]]

[1.0×1.0 + 0.5×0.0 + 0.2×0.5 + 0.1×0.0,
1.0×0.0 + 0.5×1.0 + 0.2×0.0 + 0.1×0.5,
1.0×0.5 + 0.5×0.0 + 0.2×1.0 + 0.1×0.0,
1.0×0.0 + 0.5×0.5 + 0.2×0.0 + 0.1×1.0]
= [1.1, 0.55, 0.7, 0.35]

"cat": [0.65, 1.1, 0.55, 0.7]
"sat": [0.8, 0.45, 1.15, 0.6]
<end>: [0.15, 0.6, 0.15, 1.05]

21

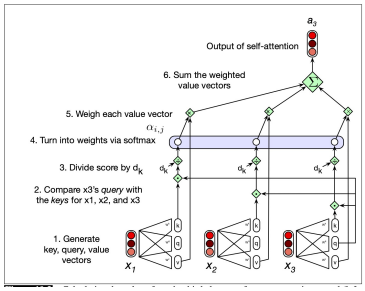
Back to Cats

- Computing k = X @ W_K

For token 1: "The" =

[1.0, 0.5, 0.2, 0.1] @ [[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]]

[1.0×1.0 + 0.5×0.2 + 0.2×0.0 + 0.1×0.0,
1.0×0.2 + 0.5×1.0 + 0.2×0.0 + 0.1×0.0,
1.0×0.0 + 0.5×0.0 + 0.2×1.0 + 0.1×0.3,
1.0×0.0 + 0.5×0.0 + 0.2×0.3 + 0.1×1.0]
= [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]
<end>: [0.12, 0.12, 0.33, 1.03]

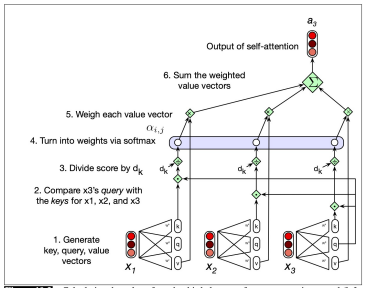
K = [[1.1, 0.7, 0.23, 0.16],
[0.7, 1.1, 0.36, 0.29],
[0.34, 0.26, 1.15, 0.8],
[0.12, 0.12, 0.33, 1.03]]

22

Back to Cats

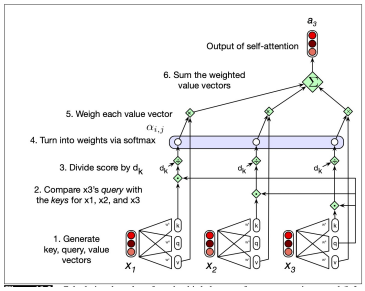
- Computing v = X @ W_V

"The": [0.82, 0.47, 0.17, 0.19]
"cat": [0.7, 1.1, 0.36, 0.29]
"sat": [0.34, 0.26, 1.15, 0.8]
<end>: [0.12, 0.12, 0.33, 1.03]



23

Back to Cats



- Calculating attention score for "The": q @ K

K = [[1.1, 0.7, 0.34, 0.12],
[0.7, 1.1, 0.26, 0.12],
[0.23, 0.36, 1.15, 0.33],
[0.16, 0.29, 0.8, 1.03]]

Q for "The" = [1.1, 0.55, 0.7, 0.35]

"The" & "The": 1.1×1.1 + 0.55×0.7 + 0.7×0.23 + 0.35×0.16 =
1.21 + 0.385 + 0.161 + 0.056 = 1.812

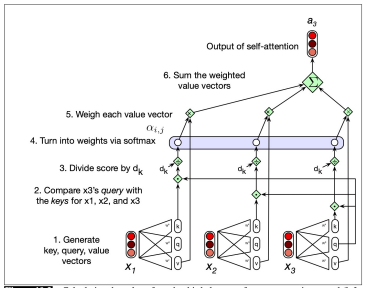
"The" & "cat": 1.1×0.7 + 0.55×1.1 + 0.7×0.36 + 0.35×0.29 =
0.77 + 0.605 + 0.252 + 0.102 = 1.729

"The" & "sat": 1.1×0.34 + 0.55×0.26 + 0.7×1.15 + 0.35×0.8 =
0.374 + 0.143 + 0.805 + 0.28 = 1.602

"The" & "<end>": 1.1×0.12 + 0.55×0.12 + 0.7×0.33 + 0.35×1.03 =
0.132 + 0.066 + 0.231 + 0.361 = 0.790

24

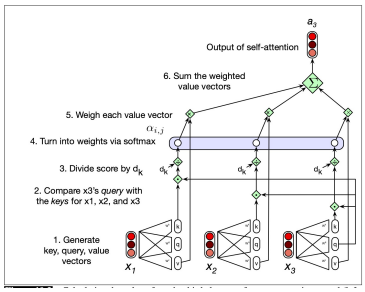
Back to Cats



- Scaling, i.e. divide by square root of dimensionality of key.
  "The" score: [1.812/2, 1.729/2, 1.602/2, 0.790/2] =
  [0.906, 0.865, 0.801, 0.395]
- Softmax:
  exp(0.906) = 2.475, exp(0.865) = 2.375, exp(0.801) = 2.228, exp(0.395) = 1.484
  Sum = 8.562
  Attention weights for "The":
  [2.475/8.562, 2.375/8.562, 2.228/8.562, 1.484/8.562] =
  [0.289, 0.277, 0.260, 0.173]
  Interpretation: "The" pays:
  - 28.9% attention to itself
  - 27.7% attention to "cat"
  - 26.0% attention to "sat"
  - 17.3% attention to "<end>"

25

Back to Cats



- Output for "The"

$0.289 \times V_{\text{"The"}} + 0.277 \times V_{\text{"cat"}} + 0.260 \times V_{\text{"sat"}} + 0.173 \times V_{\text{"<end>"}} =$

$0.289 \times [0.82, 0.47, 0.17, 0.19] +$
$0.277 \times [0.42, 0.93, 0.34, 0.23] +$
$0.260 \times [0.26, 0.19, 0.82, 0.48] +$
$0.173 \times [0.18, 0.10, 0.09, 0.91]$

$= [0.237, 0.136, 0.049, 0.055] +$
$[0.116, 0.258, 0.094, 0.064] +$
$[0.068, 0.049, 0.213, 0.125] +$
$[0.031, 0.017, 0.016, 0.157]$

$= [0.452, 0.460, 0.372, 0.401]$ ⟸

- This is the context-aware output vector for "The"
- It has been enriched with weighted information from all other tokens.

26

Multi-head Attention

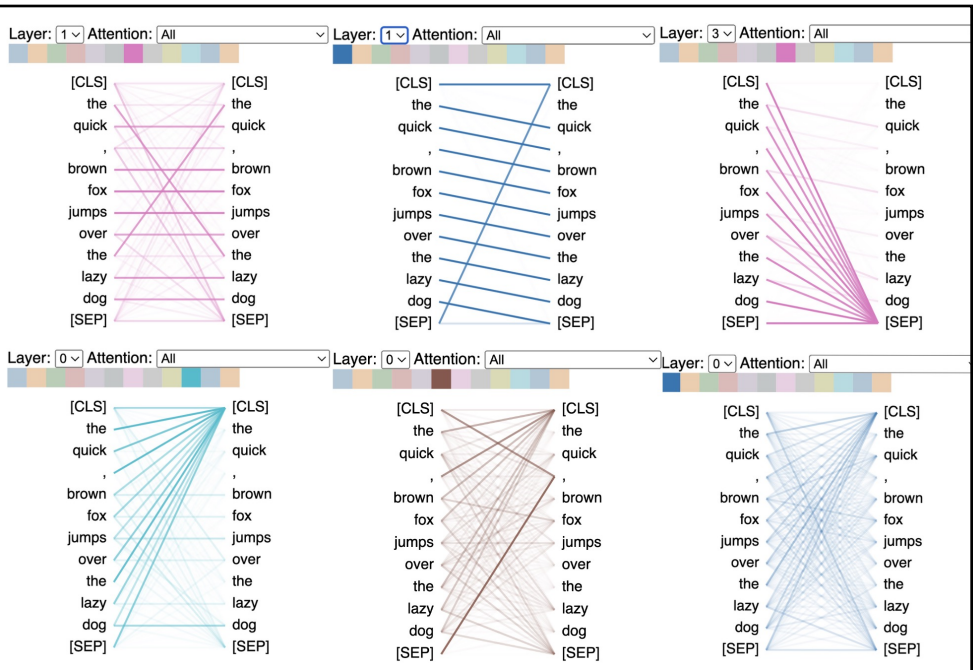
- In the context of:

- We will add more than one attention head!
- More attention heads, more weights, more things to pay attention to.
- Recall that for CNNs, we applied several filters to a matrix, to "look" at different aspects of an image.

Image source: https://muppet.fandom.com/wiki/Me_Want_Cookie?file=MeWantCookie.jpg

27

BertViz shows that attention captures various patterns in language, including positional patterns, delimiter patterns, and bag-of-words.

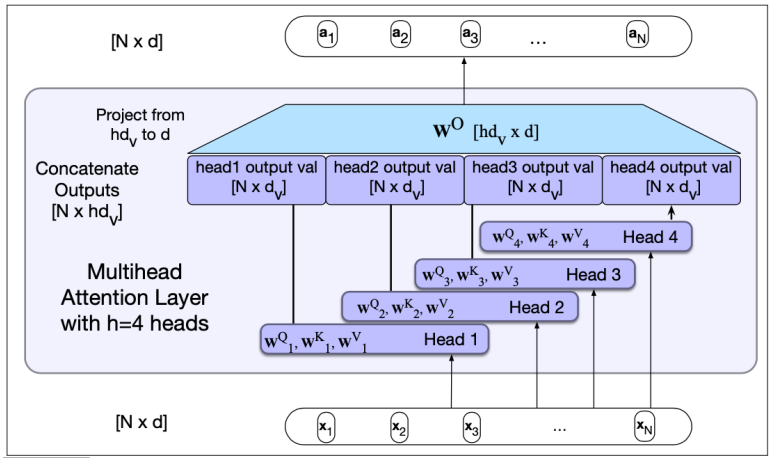


Source: https://www.comet.com/site/blog/explainable-ai-for-transformers/

28

Multi-head Attention

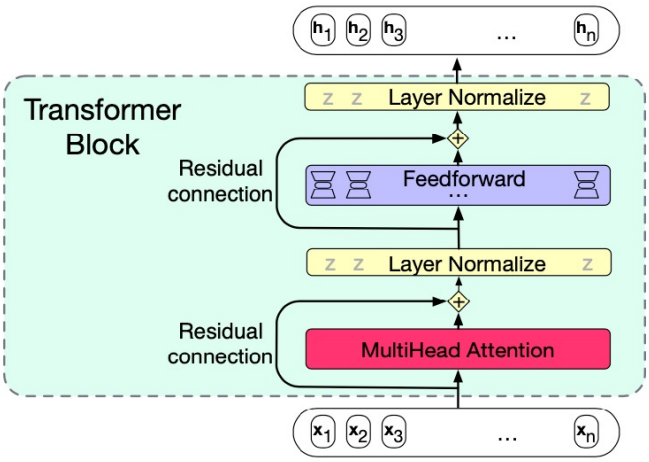
- Each of the multi-head self-attention layers is provided with its own set of key, query and value weight matrices.
- The outputs from each of the layers are concatenated.
- They are then projected to *d*.
- Thus producing an output of the same size as the input.



29

Transformer Blocks

- The self-attention calculation lies at the core of what is called a **transformer block**.
- In addition to the self-attention layer, it includes feedforward layers, residual connections, and normalizing layers.



30