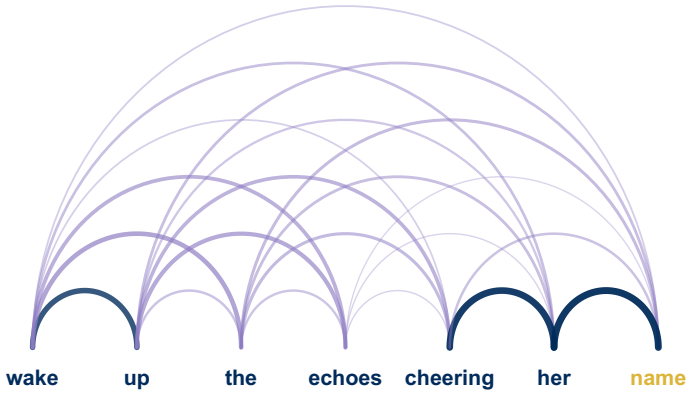
UNIT 03 · CHAPTER 5

Self-Attention

CSE 30124 · Introduction to Artificial Intelligence

William Theisen · University of Notre Dame



Flash Survey Results

I've recently started to try showing more code in lecture. What do you think about that?

Copy chart

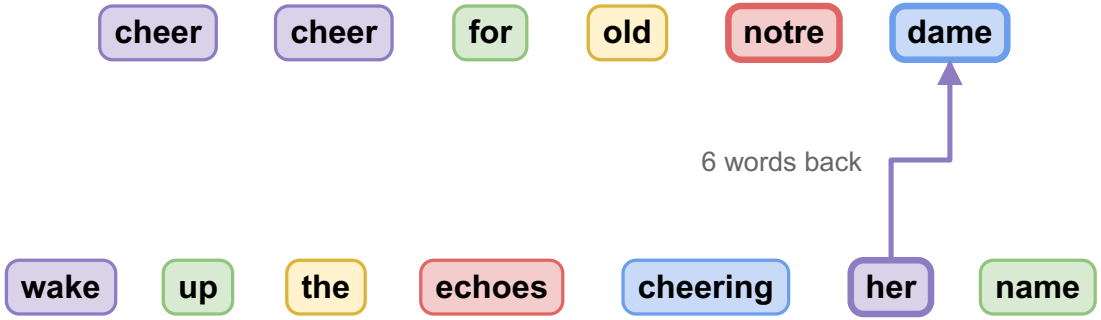
27 responses



I thought **"Good idea but do it better"** was the best answer personally

Context

Often times when working with **sequence data**, we need to recall things that happened a long time ago.



her is referring to **notre** **dame**, 6 words back. We call this a _____. However due to the sequential nature of an **RNN**, **notre** **dame** can only effect **her** indirectly, through the tokens in between.

Embeddings

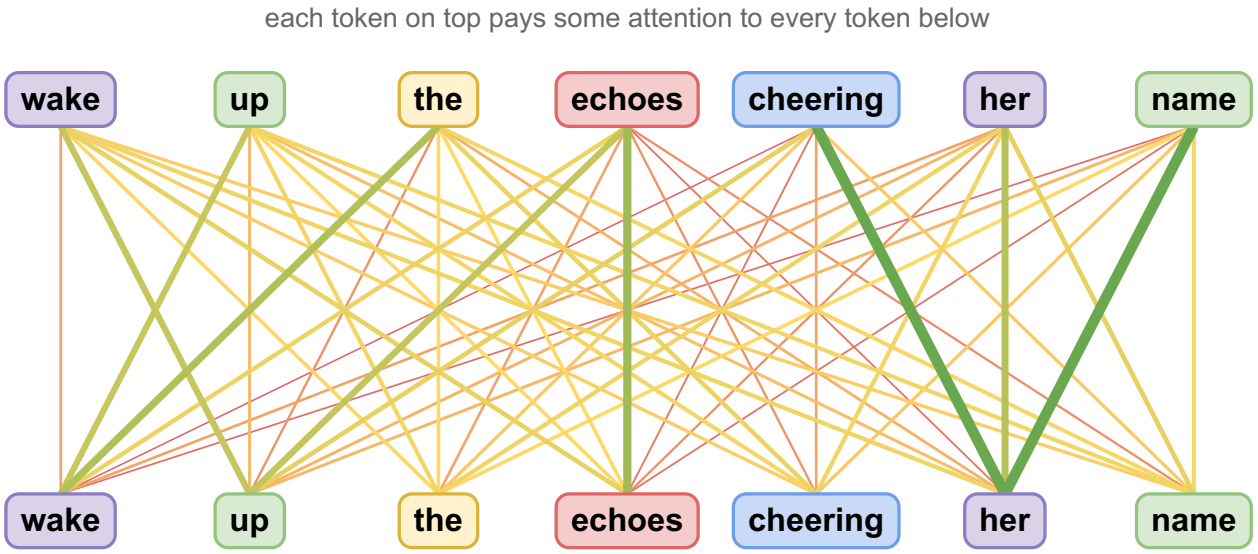
Additionally, when we have **tokens** with multiple definitions like **mole**, they currently all share the same **embedding**. We call these _____.

The	**mole**	dug a tunnel under the lawn	**the animal**
One	**mole**	of carbon weighs about 12 grams	**the unit**
The doctor checked the	**mole**	on her arm	**the spot on skin**
The	**mole**	passed secrets to the other side	**the spy**

This doesn't really make sense, when you see the word **mole**, you use **context clues** to determine which definition of **mole** the sentence is using.

Attention

In an ideal world, we could allow each **token** to decide how much **attention** they should pay to each **token** and which **context** from each **token** they should use.



We call this **self-attention**

The Self-Attention Equation

- $Q = X W_Q$, the _____: what this **token** is **looking for**
- $K = X W_K$, the ____: what it **offers to be matched against**
- $V = X W_V$, the _____: what it **passes along**

Self-Attention is a way to update our **embeddings** using the context of the surrounding **token's embeddings**

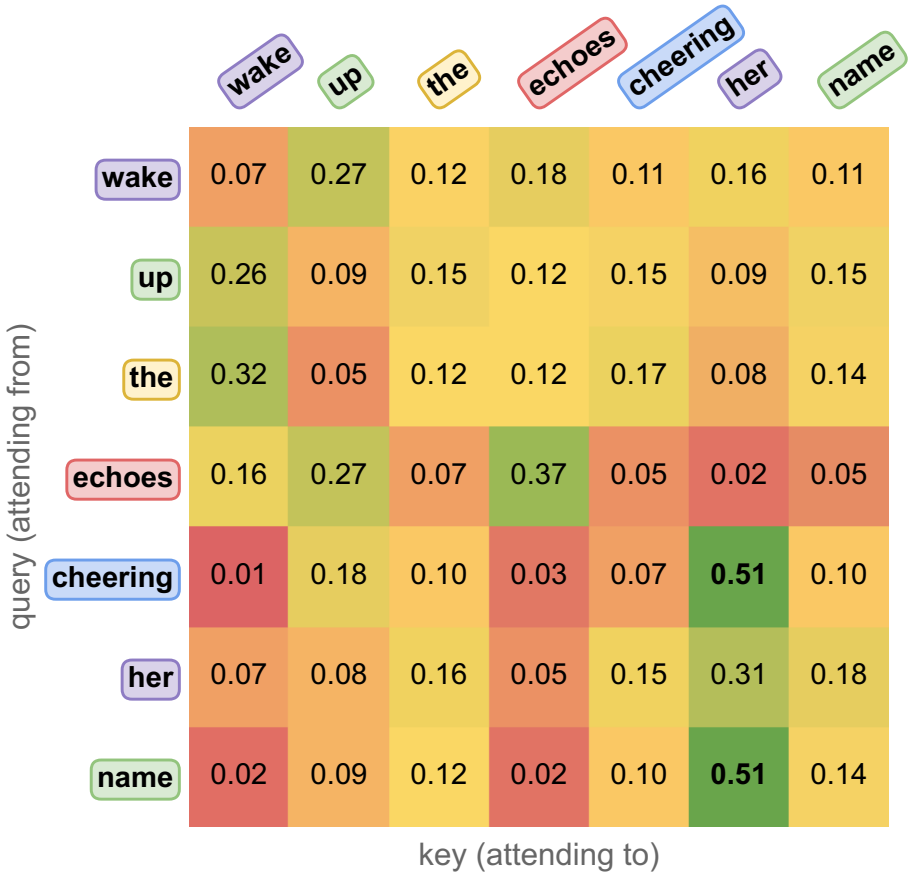
Note: X is our **embeddings**

An Analogy for Self-Attention

Another mental model for **self-attention** might be **"using tinder"**, where everyone on the app is simultaneously looking for someone and being looked at.

- **Q – Query**: What each person is looking for in a person
  - "I'm looking for a 10/10 terrachad ASU frat leader"
- **K – Key**: the first photo each person has on their profile
  - "I'm a rocket"
- **V – Value**: The rest of the profile

$\mathbf{softmax}(Q\,K^T\,/\,\sqrt{d_k})$



Self-Attention in PyTorch

Our seven words with their real **GloVe** **embeddings**, `300` numbers each

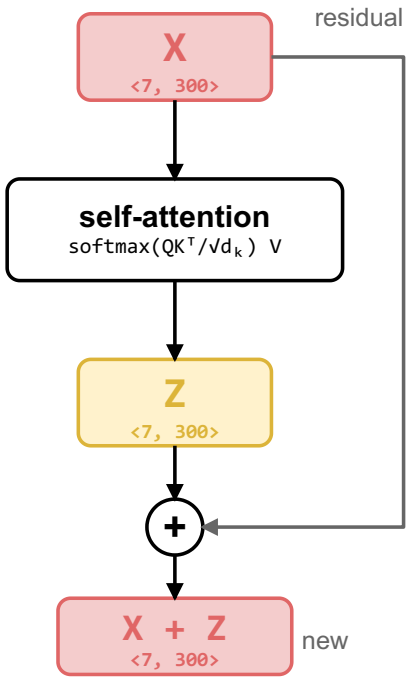
```python
import torch
words = "wake up the echoes cheering her name".split()
glove = {}
for line in open("glove-victory-march.txt"):
    w, *nums = line.split()  # a word, 300 numbers
    glove[w] = [float(n) for n in nums]
X = torch.tensor([glove[w] for w in words])  # 7 by 300
WQ = WK = WV = torch.eye(300)  # untrained: q = k = v = x

def attention(Q, K, V):
    S = Q @ K.T / K.shape[1] ** 0.5
    A = torch.softmax(S, dim=1)
    return A @ V, A

Z, A = attention(X @ WQ, X @ WK, X @ WV)
print(X.shape, Z.shape)
print(A[-1].round(decimals=2))  # name's weights
```

OUTPUT

```
torch.Size([7, 300]) torch.Size([7, 300])
tensor([0.0600, 0.1000, 0.1200, 0.0700, 0.0600, 0.1300, 0.4600])
```



How the Embeddings Change

Add Z back onto X and compare how similar **name** is to each word, before and after

```
new = X + Z  # add the context back onto X
def cos(M):  # cosine similarity of every pair
    M = M / M.norm(dim=1, keepdim=True)
    return M @ M.T
before, after = cos(X), cos(new)
i = words.index("name")
for j, w in enumerate(words[:-1]):
    b, a = before[i, j], after[i, j]
    print(f"'name' and {w!r:10} {b:.2f} -> {a:.2f}")
same = cos(torch.stack([X[i], new[i]]))[0, 1]
print(f"'name' before and after: {same:.2f}")
```

OUTPUT

```
'name' and 'wake'     0.09 -> 0.38
'name' and 'up'       0.30 -> 0.54
'name' and 'the'      0.45 -> 0.66
'name' and 'echoes'   0.13 -> 0.38
'name' and 'cheering' 0.02 -> 0.20
'name' and 'her'      0.35 -> 0.54
'name' before and after: 0.98
```

name moves toward its **context**: every similarity goes up (**her** `0.35` → `0.54`), yet it is still `0.98` like its old self

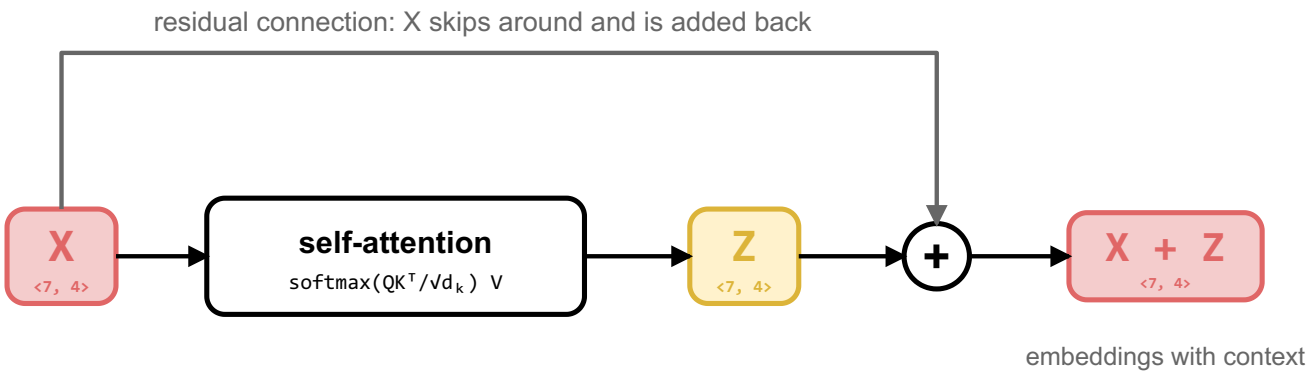
Self-Attention (old exam question)

Old exam question: What are **Q**, **K**, and **V** in relation to **self-attention** (not just what they're short for, provide a one line description of their intuitive function)? (3 pts.)

Values

All **softmax**(QK/d) does is compute how much **attention** each token should pay to the others.

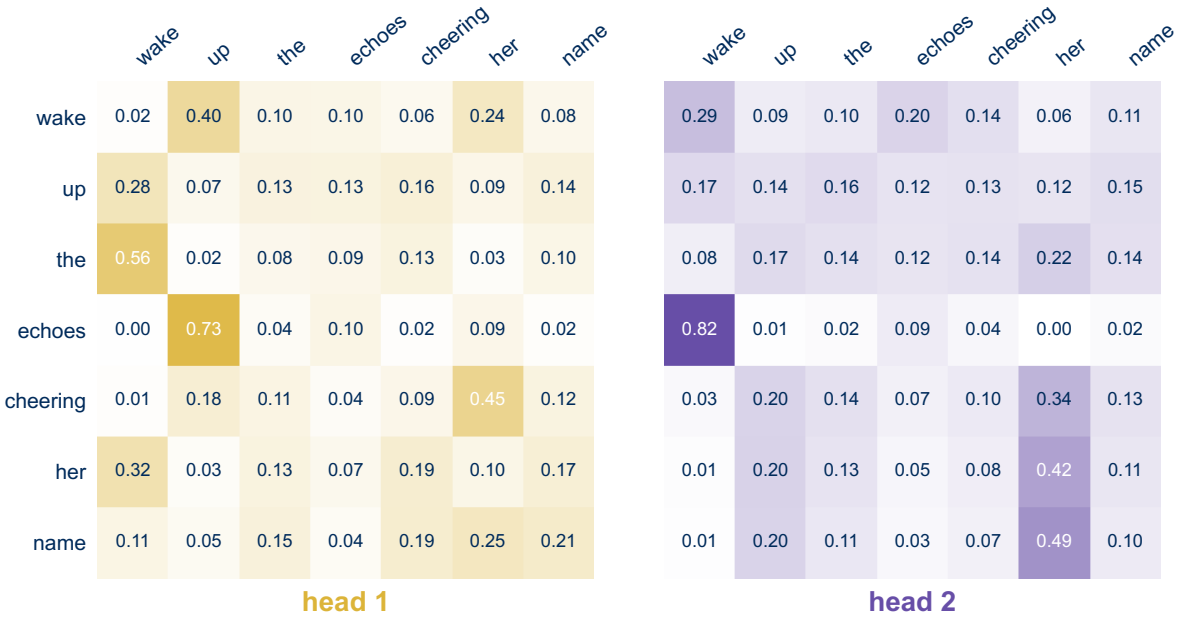
To actually update our **embeddings** we then multiply these **attention** scores by the **values** matrix V.



We then add these back onto our original **embeddings**, updating each with the context provided by the surrounding tokens

Attention Heads

One set of W_Q, W_K, W_V gives one grid of weights: **one kind of relevance**, context for a single "thing"



But a word relates to its sentence in many ways (**her** owns **name**, **echoes** get woken **up**). So we use several, each an _____

Multi-Head Attention

Several heads at once is called **multi-head attention**. Each head **focuses on a different facet of the context**, maybe one on the _____ and another on the _____

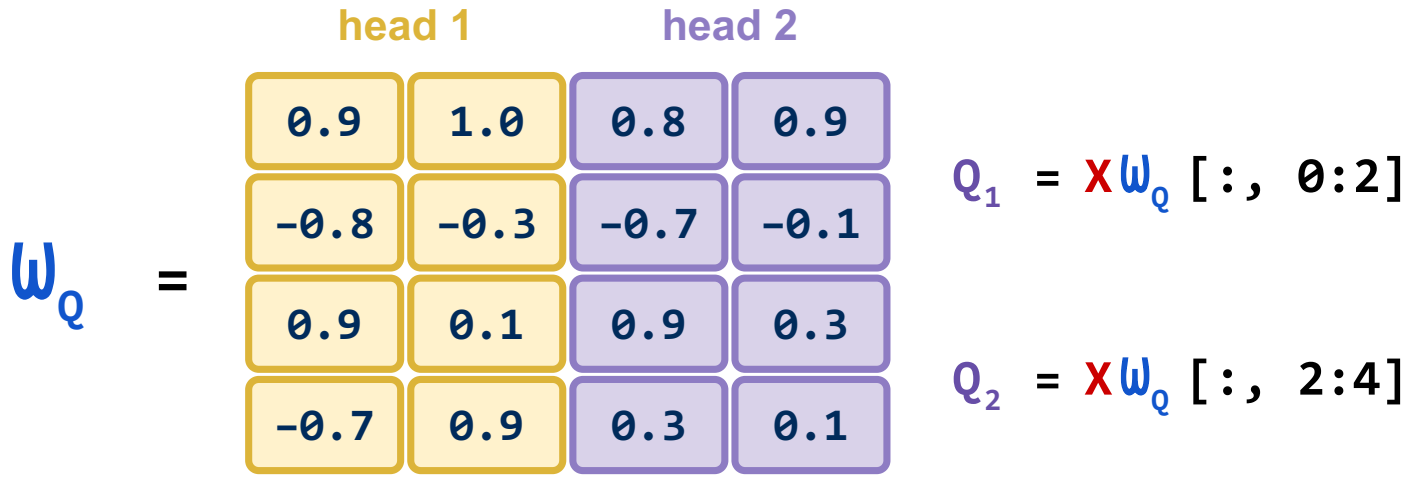
h heads, each on its own slice of the columns

$$\text{MultiHead}(X) = \left[\text{head}_1 \ \text{head}_2 \ \cdots \ \text{head}_h\right] W_O$$

Each head runs the **attention** equation on its own slice, then we _____ their outputs and pass them through **one linear layer**, W_O:

Multi-Head Attention (note)

Done **not with multiple weight matrices**, but a **logical partition of a single matrix**



More heads don't add a single weight: **Attention Is All You Need** split 512 columns into 8 heads of 64, **GPT-3** splits 12,288 into 96 heads of 128

Multi-Head Attention in PyTorch

```python
import numpy as np
import torch
from torch import nn

rng = np.random.default_rng(28)
X = rng.uniform(-1, 1, (7, 4)).round(1) + 0.0
Ws = rng.uniform(-1, 1, (4, 4, 4)).round(1)
X = torch.tensor(X, dtype=torch.float32)
WQ, WK, WV, WO = torch.tensor(Ws, dtype=torch.float32)

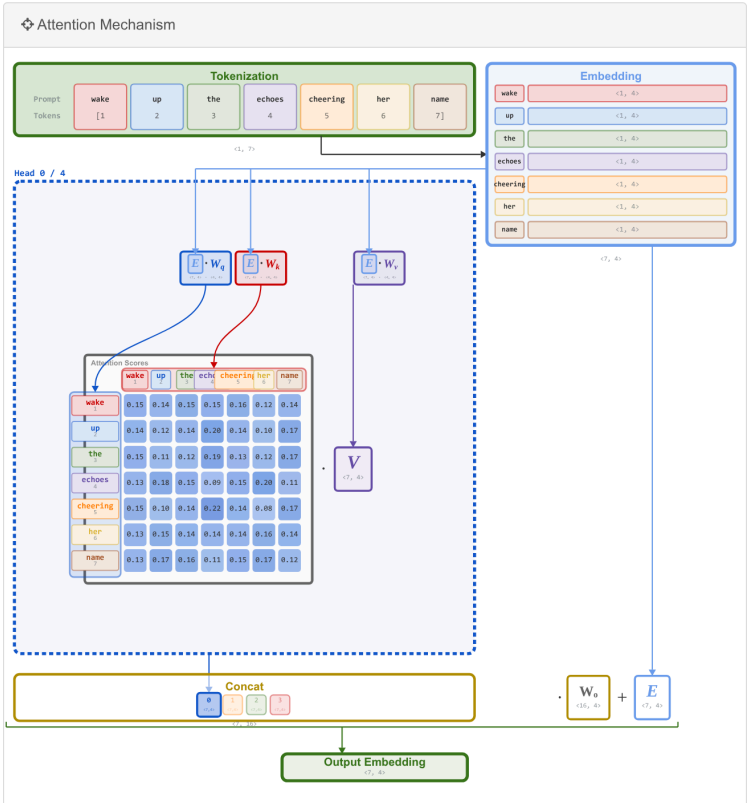
mha = nn.MultiheadAttention(embed_dim=4, num_heads=2,
                            batch_first=True)
W = torch.cat([WQ.T, WK.T, WV.T])  # 12 by 4
with torch.no_grad():  # our weights, no biases
    mha.in_proj_weight.copy_(W)
    mha.out_proj.weight.copy_(WO.T)
    mha.in_proj_bias.zero_()
    mha.out_proj.bias.zero_()
Z, A = mha(X[None], X[None], X[None])  # a batch of 1
print(Z.shape, Z[0, -1].detach().round(decimals=2))
```

OUTPUT

```
torch.Size([1, 7, 4]) tensor([ 0.2600,
-0.3000,  0.2300,  0.2900])
```

Both heads and W_0 in one call (nn.MultiheadAttention, our weights loaded)

Visualization



Word Order

A **GPT** is mostly **multi-head self-attention** stacked many times (**GPT-3**: **96** layers), with **no RNN anywhere**

Shuffle our line: **the** **cheering** **echoes** **name** **her** **wake** **up**. Every token gets **the same output, just in a different row**

Self-attention treats a sentence as a _____: **wake up the echoes** = **the echoes wake up**

Next lecture: _____