

Quiz

- Go to **Quizzes > Quiz 10-20** on Canvas.
 - Session 13 Neural networks part 1: J+M 6-6.1, 6.3-6.4
 - Session 14 Neural networks part 2: J+M 6.5-6.6
 - Session 16 Introduction to LLMs: J+M 1-1.2, 1.6-1.7, 1.10-11, 7.6-7.7, 9.1
- You have until **1:10pm** to complete it
- Allowed resources
 - Textbook
 - Your notes (on a computer or on paper)
 - Course slides and website
- Resources not allowed
 - Generative AI
 - Internet searches

04. Transformers

Transformers may not fix all your NLP problems.

-
-
-

But they are worth some attention.



CS 1671

Human Language Technologies

Session 17: Transformers part 1

Michael Miller Yoder

October 20, 2026

Assessments: Homework 2

- Congrats to Ava, Logan, Jiakai, and Yiping!
- Grades for HW2 should be released next week

#	△	Team	Score
1	▲ 32	Ava L	0.784482
2	▲ 9	Logan Keebler4	0.750000
3	▲ 27	JKD	0.750000
4	▲ 27	Yiping Xu	0.750000
5	▲ 30	Aiden McCoy	0.715517
6	▲ 31	Aidan Krieger	0.715517

Assessments: project

- [Project deliverable 2](#) is due **next Thu Oct 29**
- Part 1: Basic data analysis (if any updates are required from the proposal)
- Part 2: Result from baseline approach
 - Ideally performance metric result from the baseline system you proposed
- Part 3: Advanced system (LLM) proposal
 - How might you use an LLM programmatically to attempt your task?
 - Zero-shot and more advanced approaches

Overview: Transformers part 1

- Discussion: LLM harms
- Coding activity: prompting SCI open-source LLMs
- Contextual word embeddings
- Self-attention
 - Activity: work through self-attention



Harms from LLMs

What Can You Do When A.I. Lies About You?

People have little protection or recourse when the technology creates and spreads falsehoods about them.

Hallucination

Air Canada loses court case after its chatbot hallucinated fake policies to a customer

The airline argued that the chatbot itself was liable. The court disagreed.

Copyright

Authors Sue OpenAI Claiming Mass Copyright Infringement of Hundreds of Thousands of Novels

Privacy

How Strangers Got My Email Address From ChatGPT's Model

Harms from LLMs

Toxicity and abuse

The New AI-Powered Bing Is Threatening Users.

Cleaning Up ChatGPT Takes Heavy Toll on Human Workers

Contractors in Kenya say they were traumatized by effort to screen out descriptions of violence and sexual abuse during run-up to OpenAI's hit chatbot

Misinformation

Chatbots are generating false and misleading information about U.S. elections

Coding activity: prompting SCl open-source LLMs

Notebook: prompting SCl open-source LLMs

1. Go to this [nbgitpuller link](#) (also available on course website)
2. Start a server with **TEACH – 6 CPUs – 45 GB**
3. Load custom environment at
`/ix1/cs1671-2026f/class-env/.pixi/envs/default`
4. Open **session16_prompting.ipynb**

Discussion: harms from LLMs

- What potential harms are you concerned about from LLMs?
- What training data could be linked to those harms?

Contextual word embeddings

Problem with static embeddings (word2vec)

They are static! The embedding for a word doesn't reflect how its meaning changes in context.

The chicken didn't cross the road because **it** was too tired

What is the meaning represented in the static embedding for "it"?

Contextual Embeddings

- Intuition: a representation of meaning of a word should be different in different contexts!
- **Contextual embedding**: each word has a different vector that expresses different meanings depending on the surrounding words
- How to compute contextual embeddings? **Attention**

Contextual Embeddings

The chicken didn't cross the road because it

What should be the properties of "it"?

The chicken didn't cross the road because it was too **tired**

The chicken didn't cross the road because it was too **wide**

At this point in the sentence, it's probably referring to either the chicken or the street

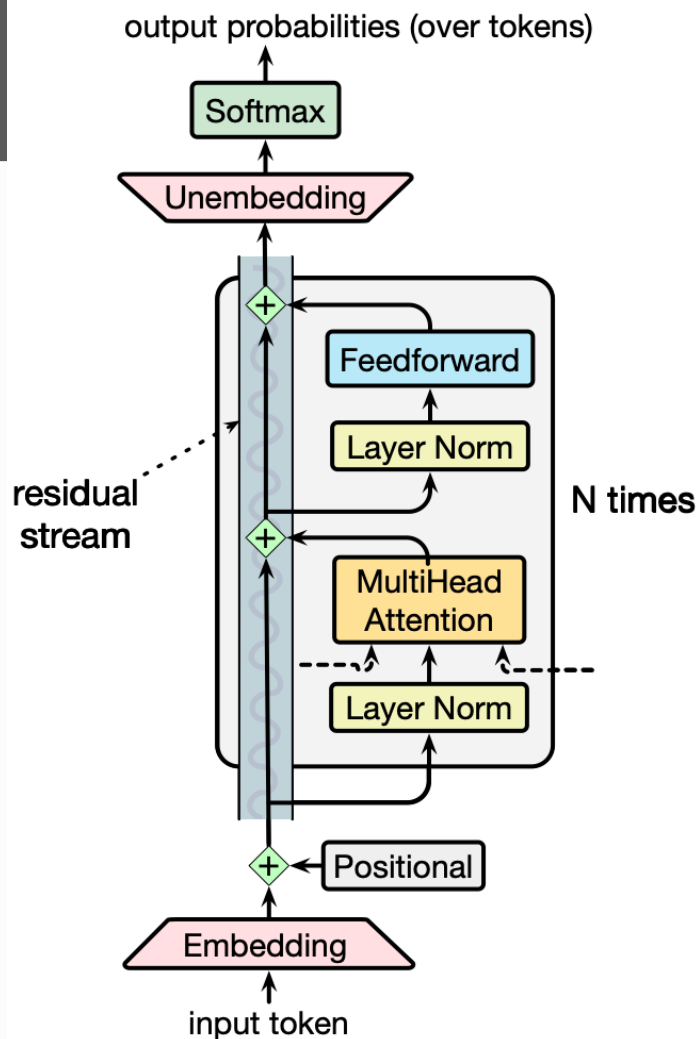
Self-attention

Vaswani et al. 2017,
“Attention is all you need”

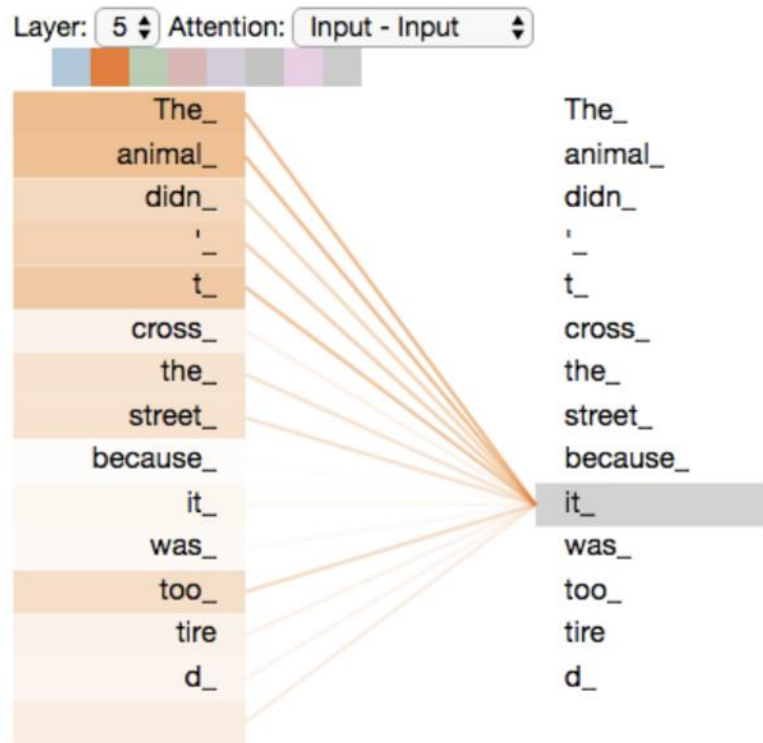


Intuition of self-attention

- Transformers give one output vector (embedding) for every input token
- Build up the contextual embedding for each token by selectively integrating information from all the neighboring words
- We say that a word "attends to" some neighboring words more than others



Self-attention illustrated



Attention definition

A mechanism for helping compute the embedding for a token by selectively attending to and integrating information from surrounding tokens (at the previous layer).

More formally: a method for doing a weighted sum of vectors.

An actual attention head: slightly more complicated

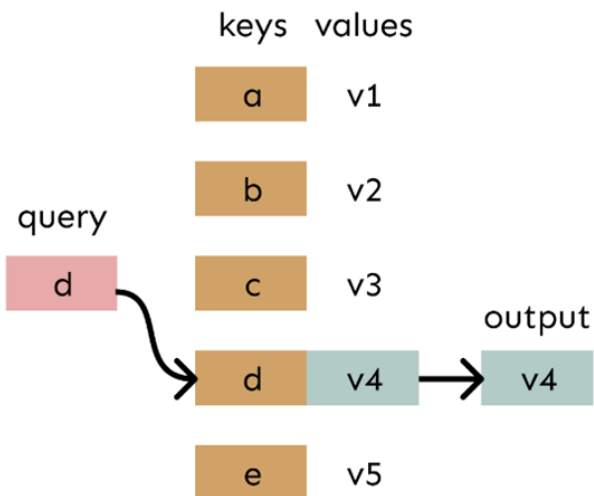
High-level idea: we'll represent 3 separate roles the vector for each word, $\mathbf{x}_i$ plays:

- **query:** *As the current element* being compared to the other inputs.
- **key:** *as an input* that is being compared to the current element to determine a similarity
- **value:** a value of a preceding element that gets weighted and summed

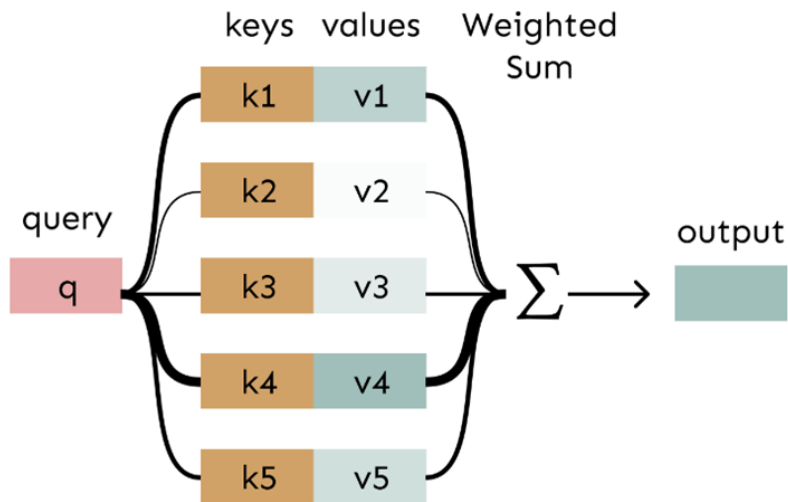
Attention as a soft, averaging lookup table

We can think of **attention** as performing fuzzy lookup in a key-value store.

In a **lookup table**, we have a table of **keys** that map to **values**. The **query** matches one of the keys, returning its value.



In **attention**, the **query** matches all **keys** *softly*, to a weight between 0 and 1. The keys' **values** are multiplied by the weights and summed.



Parameters: weight matrices for queries, keys and values

- We'll use matrices to project each vector $\mathbf{x}_i$ into a representation of its role as query, key, value:
- query: W^Q
- key: W^K
- value: W^V

$$\mathbf{q}_i = \mathbf{x}_i W^Q; \quad \mathbf{k}_i = \mathbf{x}_i W^K; \quad \mathbf{v}_i = \mathbf{x}_i W^V$$

An actual attention head: slightly more complicated

- Given these 3 representation of $\mathbf{x}_i$

$$\mathbf{q}_i = \mathbf{x}_i \mathbf{W}^Q; \quad \mathbf{k}_i = \mathbf{x}_i \mathbf{W}^K; \quad \mathbf{v}_i = \mathbf{x}_i \mathbf{W}^V$$

- To compute the similarity of current element $\mathbf{x}_i$ with some element (for self-attention) $\mathbf{x}_j$
- We'll use dot product between $\mathbf{q}_i$ and $\mathbf{k}_j$.
- And instead of summing up $\mathbf{x}_j$, we'll sum up $\mathbf{v}_j$

Final equations for one attention head

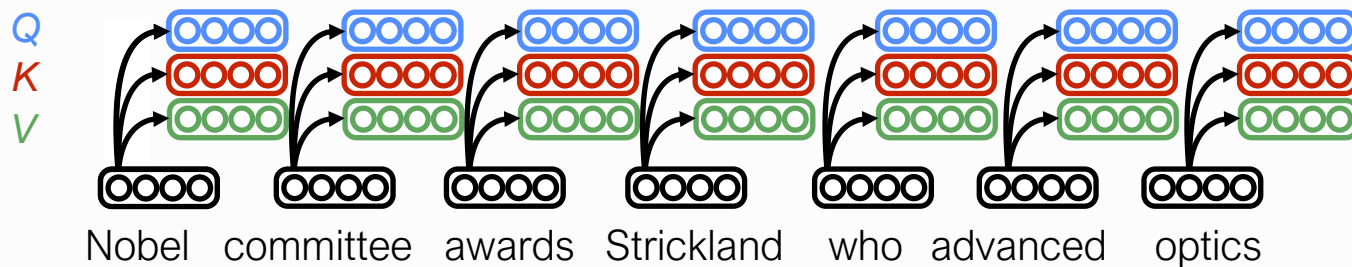
$$\mathbf{q}_i = \mathbf{x}_i \mathbf{W}^Q; \quad \mathbf{k}_j = \mathbf{x}_j \mathbf{W}^K; \quad \mathbf{v}_j = \mathbf{x}_j \mathbf{W}^V$$

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}$$

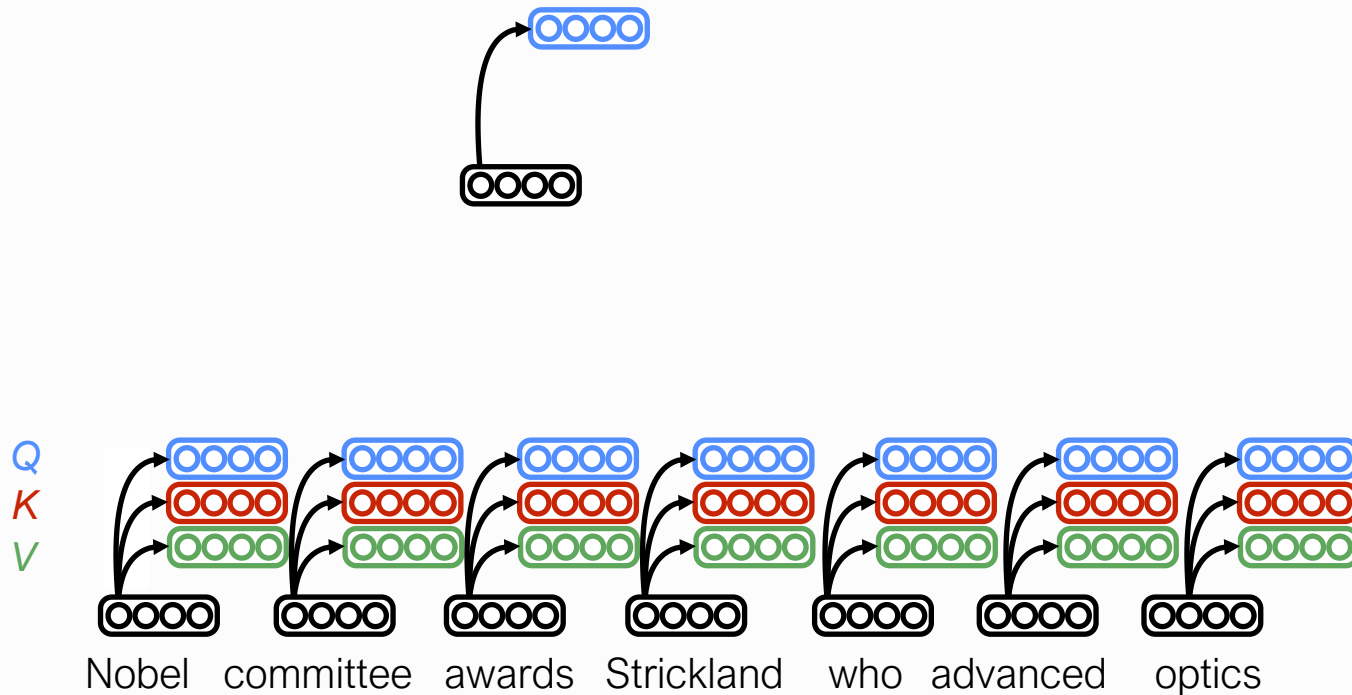
$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \quad \forall j \leq i$$

$$\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{v}_j$$

Transformer self-attention

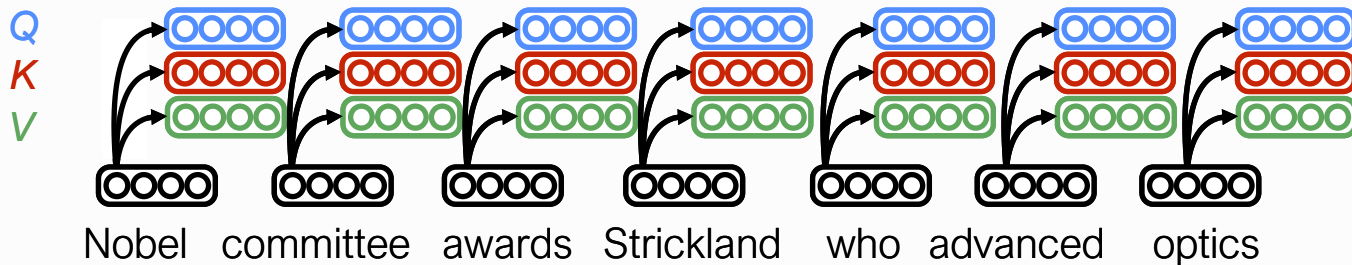


Transformer self-attention

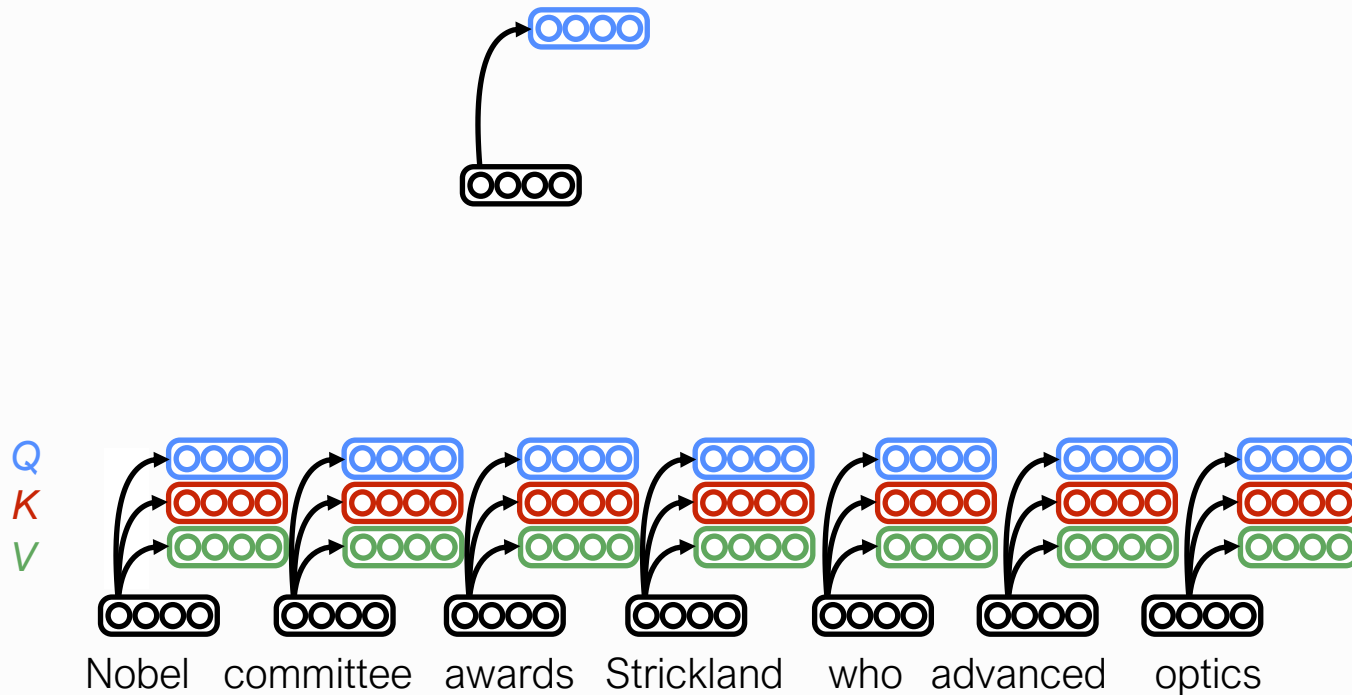


Transformer self-attention

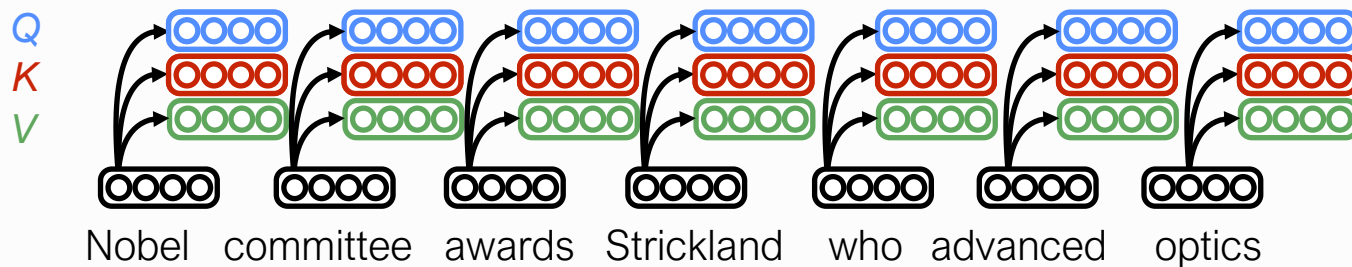
$$\text{Input Vector} \times \text{Attention Matrix} + \text{Bias} = \text{Output Vector}$$



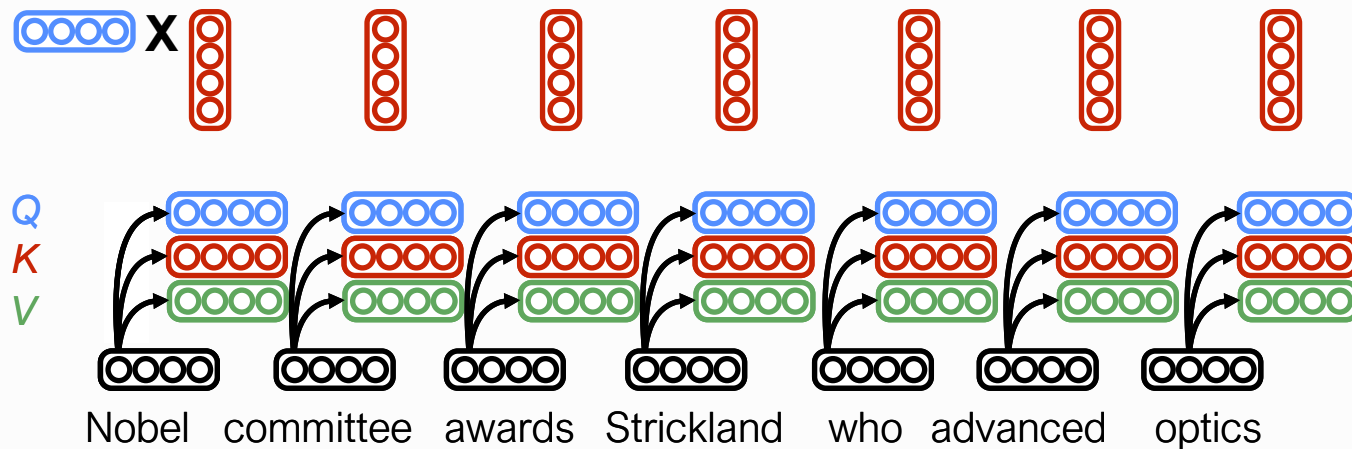
Transformer self-attention



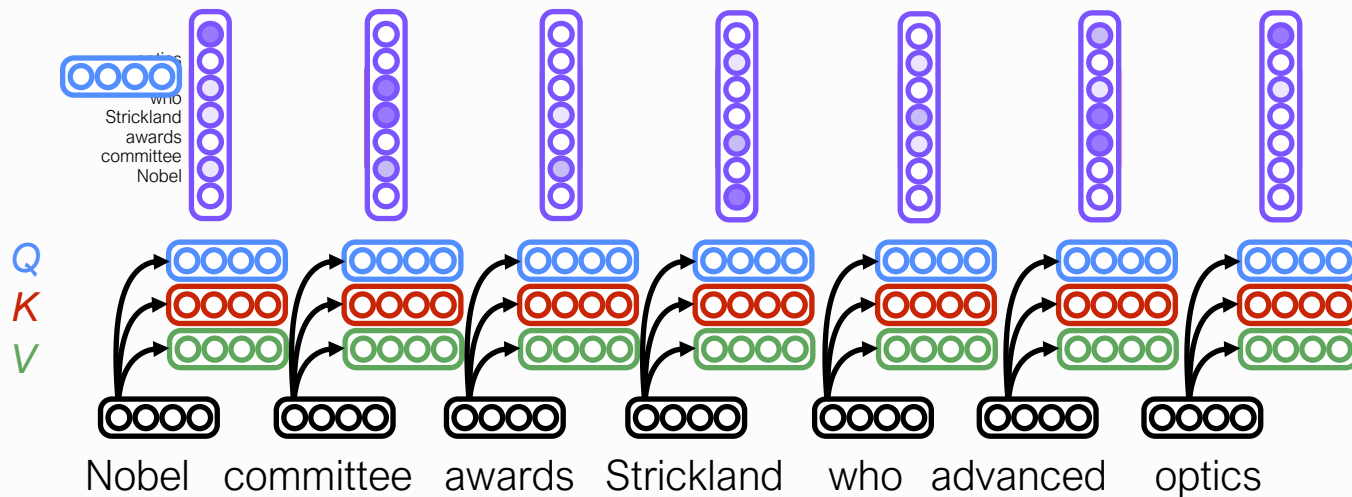
Transformer self-attention



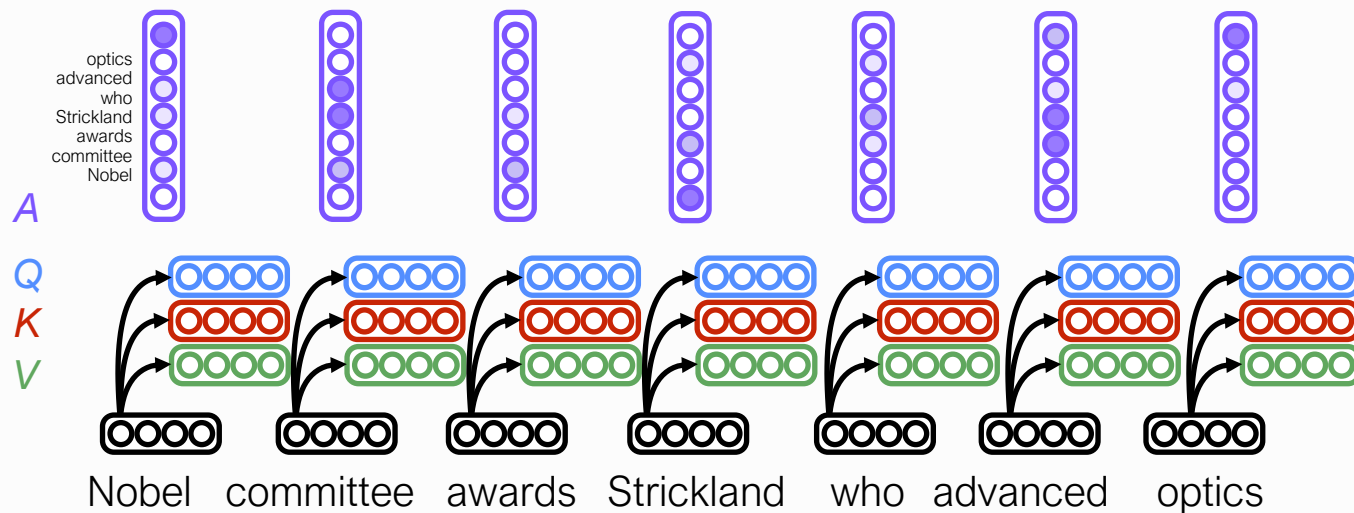
Transformer self-attention



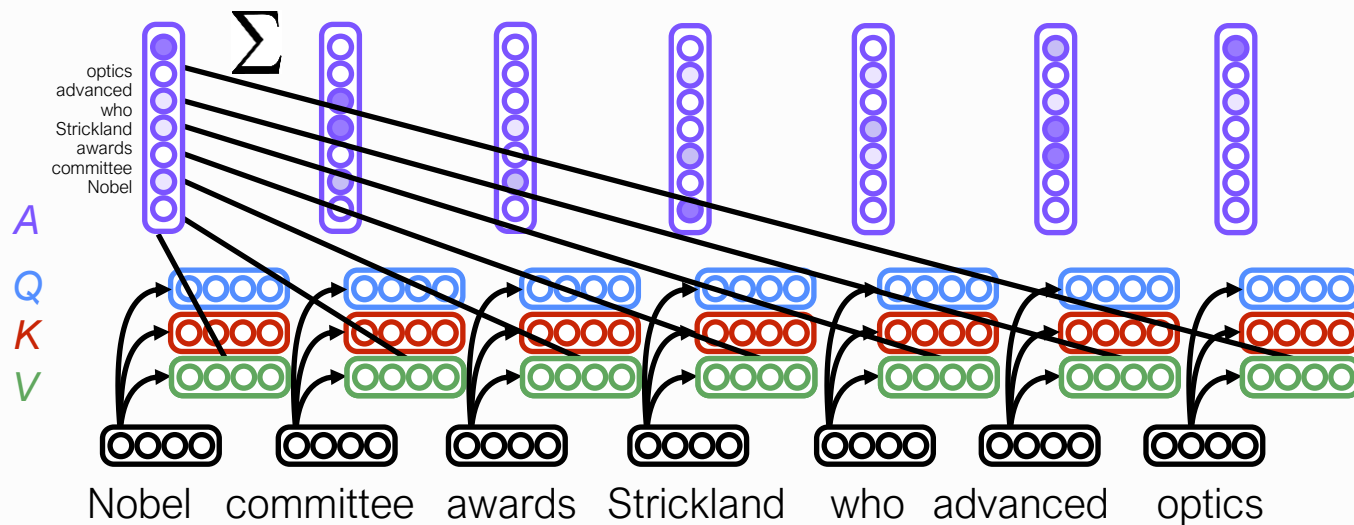
Transformer self-attention



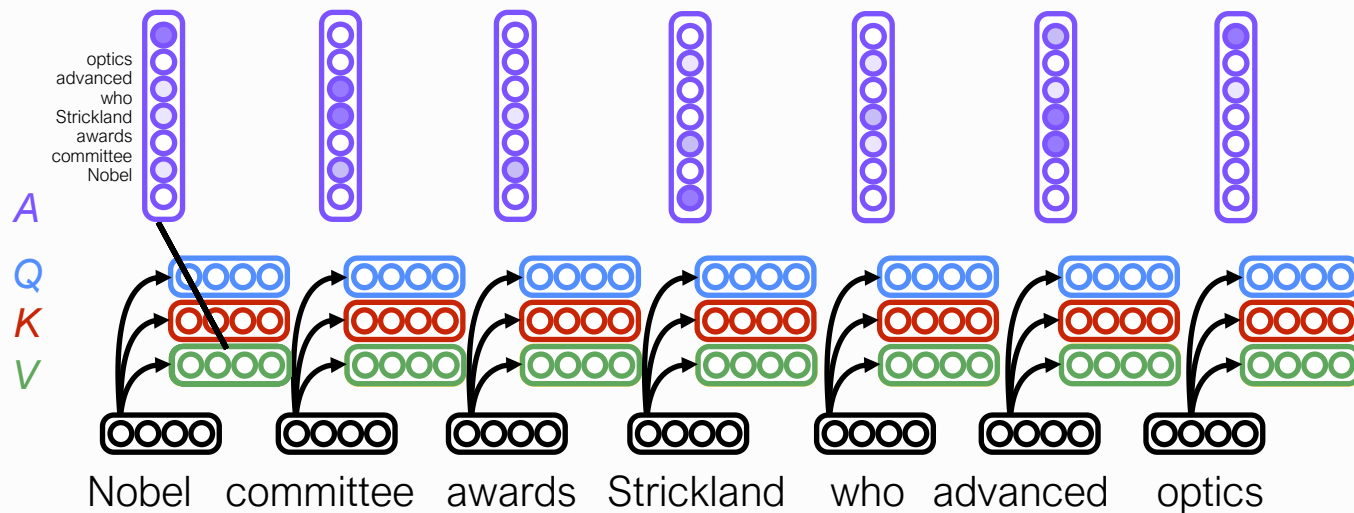
Transformer self-attention



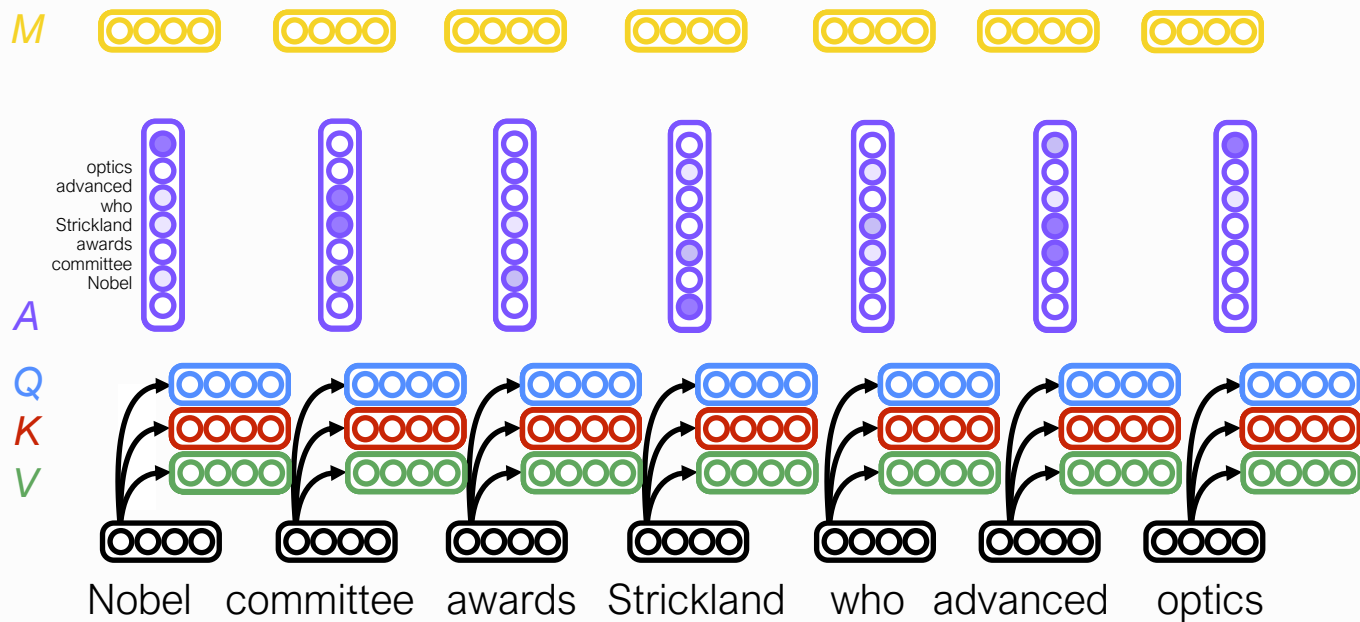
Transformer self-attention



Transformer self-attention



Transformer self-attention



Activity: work through self-attention

Calculate transformed output for one input word

- Example sentence: “we wash our cats” (don’t ask)
- Let’s just calculate the vector output, for one input word: “we”
- High-level points to remember before you get buried in the math:
 - Each token will have an output vector that integrates contextual information from other tokens in the sentence
 - Each token can play a role as a query, key, and value
- Parameters (learned through backpropagation) are assumed given:
 - W^Q, W^K, W^V

Computing Self-Attention, Step One: Compute Key, Query, and Value Vectors

d_x -dimensional
embeddings

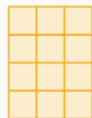
$d_k \times d_x$ -dimensional
Weight Matrices

d_k -dimensional vectors



x_1

$\times$



W^Q

$=$



q_1

queries



x_1

$\times$



W^K

$=$



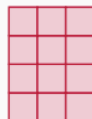
k_1

keys



x_1

$\times$



W^V

$=$



v_1

values

Dot product: vector $\cdot$ matrix

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} ax + by + cz \\ dx + ey + fz \\ gx + hy + iz \end{bmatrix}$$

Computing Self-Attention, Step One: Compute Key, Query, and Value Vectors

d_x -dimensional
embeddings

$d_k \times d_x$ -dimensional
Weight Matrices

d_k -dimensional vectors

$$x_1 = [3, 0, 1, -0.5]$$



x_1

$\times$

$$W^Q = \begin{bmatrix} 1.5 & 1 & 2 \\ 3 & -2 & 5 \\ 1 & 2 & -2 \\ 9 & 4 & 2 \end{bmatrix} =$$



q_1

queries



$$x_1 = [3, 0, 1, -0.5]$$



x_1

$\times$

$$W^K = \begin{bmatrix} 1 & 0.5 & 2 \\ -2 & 0.5 & 3 \\ 0.5 & 2 & -3 \\ 5 & 3 & 2 \end{bmatrix} =$$



k_1

keys

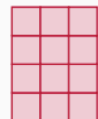


$$x_1 = [3, 0, 1, -0.5]$$



x_1

$\times$



W^V

$=$



v_1

values



Find q_1 and k_1

Computing Self-Attention, Step Two: Weighted Sum of Value Vectors

	We x_1	wash x_2	our x_3	cats x_4
query-key dot product	$q_1 \cdot k_1 =$	$q_1 \cdot k_2 =$	$q_1 \cdot k_3 =$	$q_1 \cdot k_4 =$
divide by $\sqrt{d_k}$ Assume $d_k = 64$				
softmax				
$\times$ value	$\times v_1$	$\times v_2$	$\times v_3$	$\times v_4$
sum	z_1			

$k_2 = [3, 4, 3]$
 $k_3 = [5, 2, 3]$
 $k_4 = [3, 2, 1]$

 $v_1 = [1, 0.5, -1]$
 $v_2 = [4, 5, -2]$
 $v_3 = [-3, 2, 2]$
 $v_4 = [1, 1, 6]$

Wrapping up

- Transformers are a high-performing NLP architecture based on self-attention
- Transformers produce one output vector per input token
- Output vectors from transformers integrate information from the surrounding tokens (self-attention)
- Self-attention computation is easily parallelizable