MGMT298D
Science and Strategy of AI

Week 6

Transformers and Large Language Models

Auyon Siddiq
UCLA Anderson School of Management

Product Sprint Details

Start thinking about your team now (3 or 4 people)

You will develop a *prototype* of an app that makes use of concepts from class, e.g.,

ML prediction tool
LLM chatbot agent
Computer vision

A specific and coherent use case with a clearly defined value proposition (even if it is narrow) is more important than flashiness

More details to be posted on BruinLearn tomorrow

Product Sprint Timeline

Week 8 lecture (May 18th): Overview of tools and deliverables

Week 9 lecture (Friday, May 22nd; Memorial Day make-up class): In-class lab session

11:59pm PT, May 27th: Apps go live

May 28th -- May 29th: Class preliminary voting

Week 10 lecture (June 1st): Top teams pitch in-class; final round of voting

Today's Class

Part 1: Word Embeddings

How do we represent words numerically in a way that captures their **meaning**?

Part 2: Attention

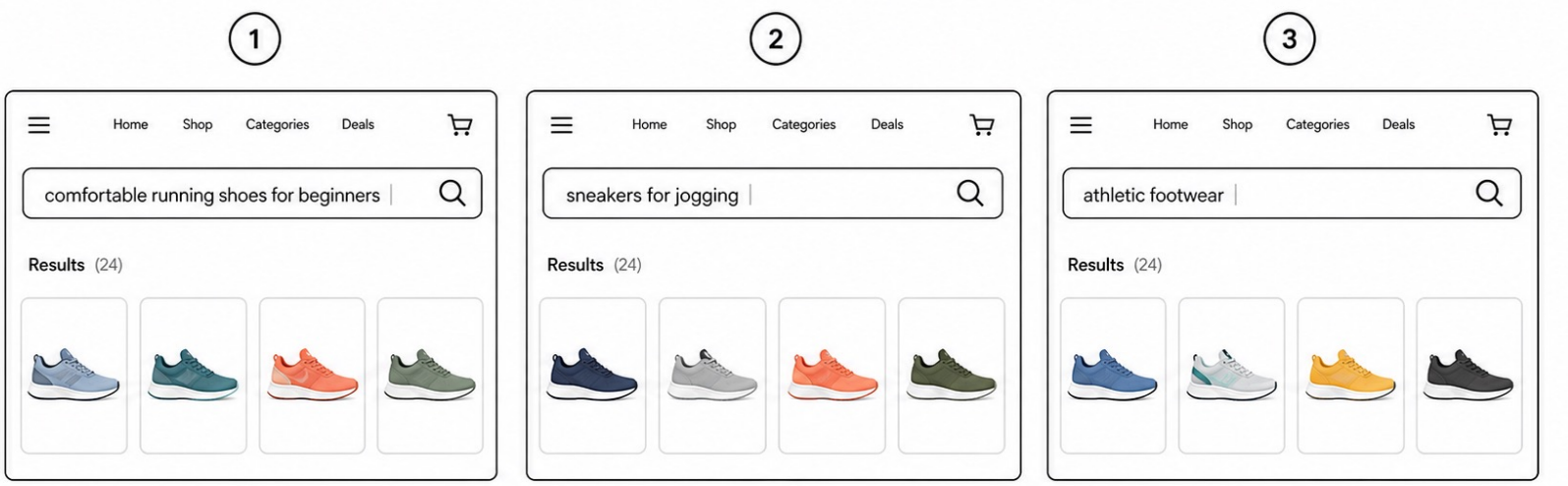
How does deep learning understand the **context** in which words are used?

Part 3: Transformers and LLMs

How do all the pieces come together in a deep neural network that can **understand** and **generate** language?

Part 1: Word Embeddings

The Word Representation Problem



All of these have the same intent but use different words. Neural networks can recognize them as equivalent. How?

The Word Representation Problem

ML models need numbers, not words

Naive approach: assign each word an arbitrary ID.

cat	ID = 1
dog	ID = 2
automobile	ID = 3
car	ID = 4

Problem: IDs do not encode meaning

car and ***automobile*** are synonyms but have unrelated IDs

cat and ***car*** are numerically close but semantically unrelated

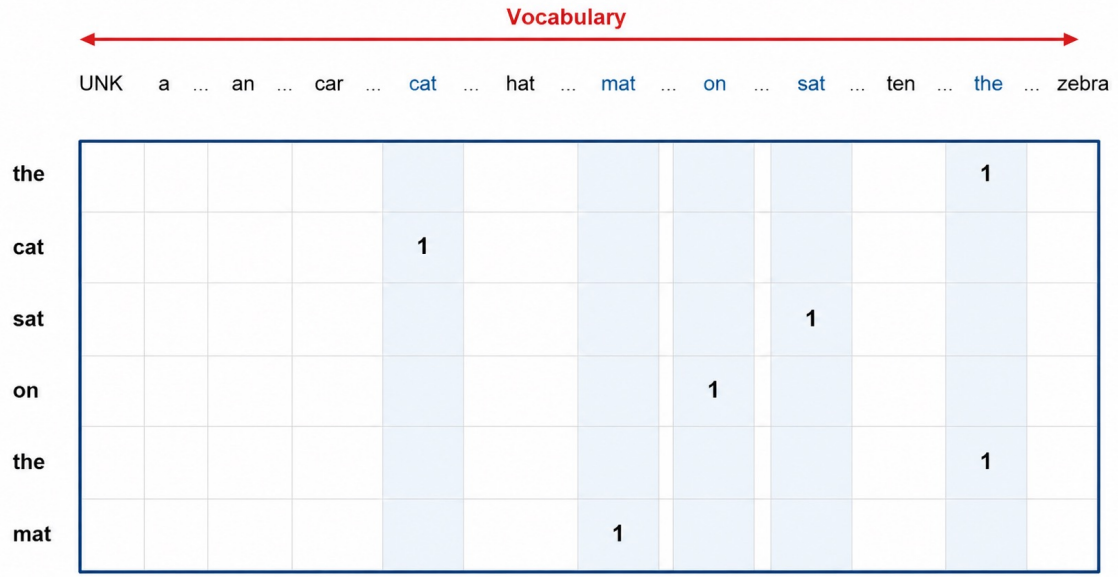
The Standard Approach

Vocabulary: All relevant words

Token: One word

Standard approach is to use “one-hot encoding”

What are some of the limitations of this approach?



Wouldn't it be nice if...

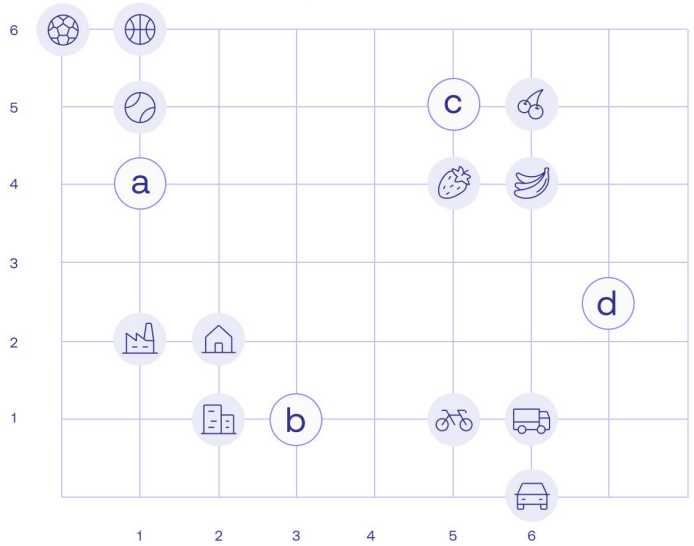
1. Each word (token) could be represented in lower dimensions?
2. We could capture the meaning of each word (numerically)?
3. Related words were closer together (numerically)?

Good news**:** All of this is possible by using word **embeddings**

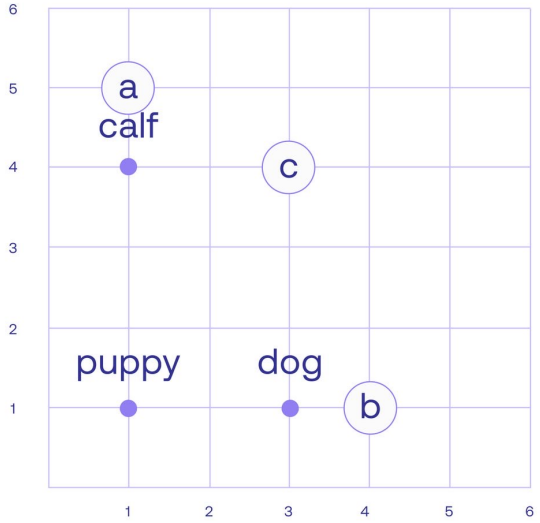
First, what embeddings are; second, how they are created

Embeddings

Embeddings Quiz 1:
Where would you put the word “apple”?



Embeddings Quiz 2:
Where would you put the word “cow”?

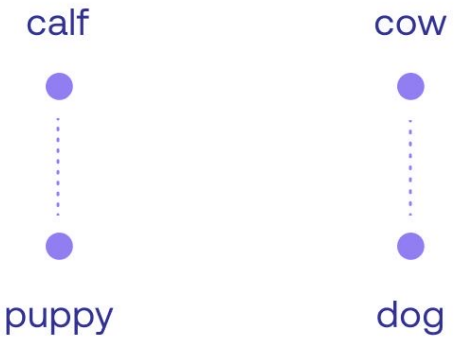


https://txt.cohere.com/sentence-word-embeddings/

Embeddings

A puppy is to a dog, like a calf is to a cow

A puppy is to calf, like a dog is to a cow



Embeddings

An embedding of a word is just a long vector of numbers:

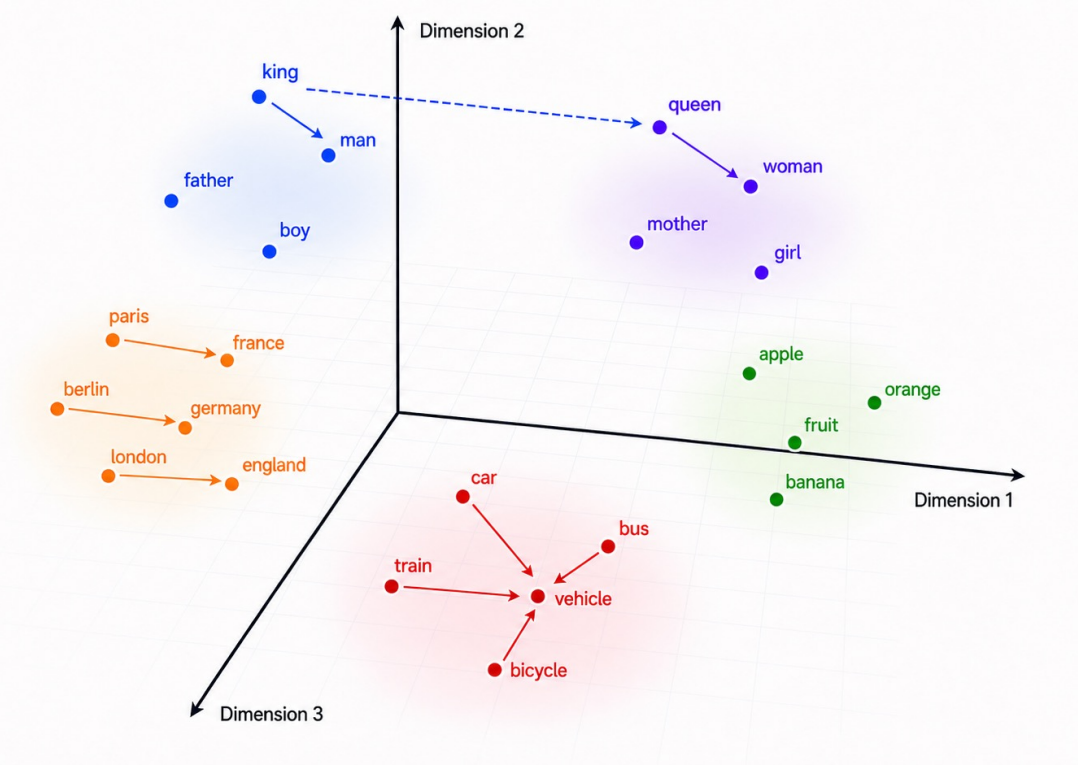
e.g., dog = [-0.21 0.3 1.2 -2.1 -0.52 ……]

An embedding can be interpreted as a point in a high-dimensional space (e.g., 256)

The point of embeddings is to allow a words meaning to be represented *spatially*

Embedding Visualization

Word Relationships in Embedding Space



The spatial relationship of embeddings reflects their meaning or usage

Semantic

italy - france + paris ≈ rome
pizza - usa + japan ≈ sushi
elbow - knee + leg ≈ arm

Morphological

children - child + person ≈ people
walking - running + ran ≈ walked

Social/Cultural

king - man + woman ≈ queen
doctor - man + woman ≈ nurse 🤔

Where Do Embeddings Come From?

“You shall know a word by the company it keeps.” — J. R. Firth

Where Do Embeddings Come From?

The acting in the ____ was superb

Likely: movie, film, play

Unlikely: refrigerator, legislature, molecule

Embedding models exploit this pattern: context is evidence for meaning.

Where Do Embeddings Come From?

Training data: a large text *corpus* such as Wikipedia, news, books, or web pages

Two major approaches to creating embeddings:

1. GloVe: learns from word co-occurrence statistics across the corpus – *how frequently does each pair of words appear together*?

2. Word2Vec: Predicts nearby words from a target word using a small neural network.

Key idea in both cases: Words used in similar contexts get similar vectors.

How GloVe Learns Embeddings

Co-occurrence: how many times two words appear near each other in a text window across the entire corpus.

GloVe starts with a giant word-by-word co-occurrence where $\mathbf{X}_{ij}$ counts how often word *j* appears near word *i*. E.g.,

$$\mathbf{X}_{(\text{radio, station})} = 10$$

$$\mathbf{X}_{(\text{radio, tv})} = 20$$

$$\mathbf{X}_{(\text{train, station})} = 5$$

$$\mathbf{X}_{(\text{radio, car})} = 1$$

$$\mathbf{X}_{(\text{train, car})} = 7$$

How GloVe Learns Embeddings

Next step is…regression!

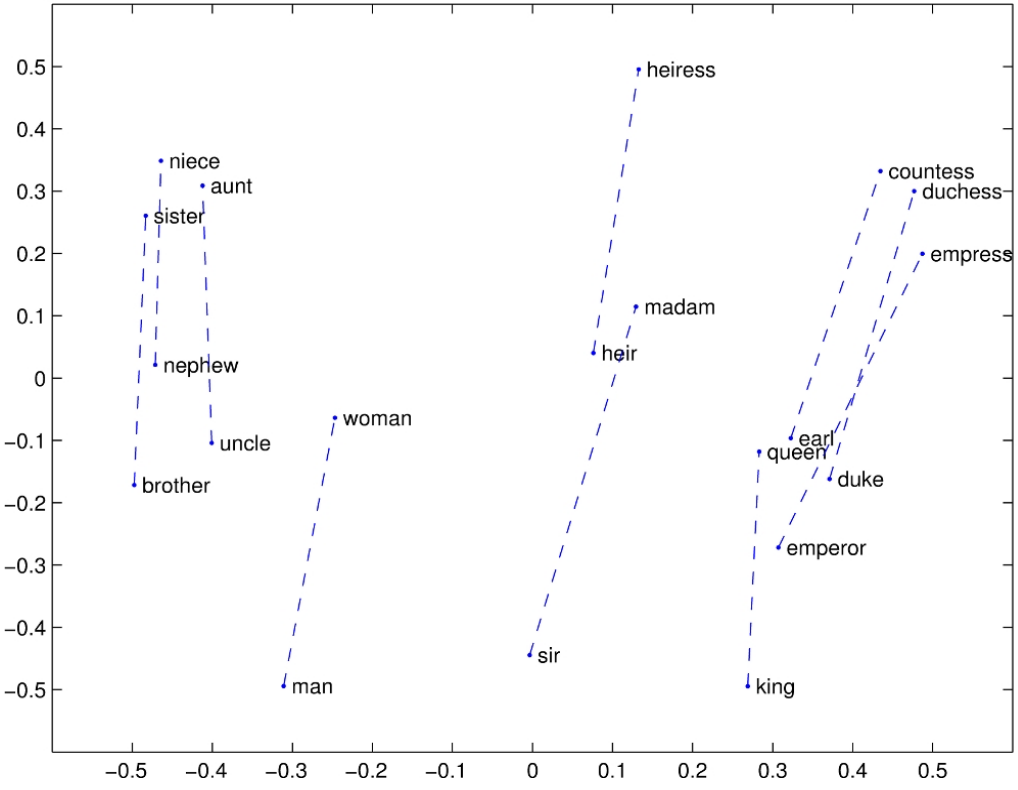
Choose vectors $\mathbf{w}_i$ and $\mathbf{w}_j$ and constants b_i and b_j that minimize prediction error of co-occurrence count across all word pairs.

$$\log(X_{ij}) \approx \mathbf{w}_i \cdot \mathbf{w}_j + b_i + b_j$$

The vector $\mathbf{w}_i$ becomes the embedding for word i

Words that appear in similar contexts get similar vectors

Real GloVe Embeddings



Embeddings Summary

Recall deep learning is all about learning useful numerical representations of raw data (images, words, audio, etc) that classification/prediction layers can deal with

Embeddings are exactly these representations

You can embed almost anything: a word, a sentence, a full page of text, an image, audio, video, a product, or a customer

Connection to computer vision**:** the output of convolutional blocks can be viewed as an embedding of an image.

Limitations of Raw Embeddings

Most word embeddings are trained on a large general corpus, not on a specific task

Embeddings capture broad statistical patterns in language, but your task may be need to use a word in a specific way

- *I deposited the check at the bank*
- *The children sat at the river bank*
- *The plane began to bank right*
- *I wouldn't bank on it*

Next challenge: How can we consider the *context* that the word appears in to improve its embedding?

Part 2: Attention

Context Matters

The problem with raw embeddings:

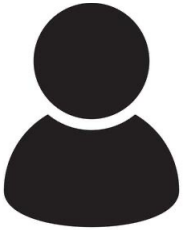
*"The animal didn't cross the street because **it** was too tired"*

*"The animal didn't cross the street because **it** was too wide"*

The embedding for **it** needs to pay “attention” to the surrounding words in the context.

Motivating Example: Classifying Yelp Reviews

Our goal is to predict # of stars from review text

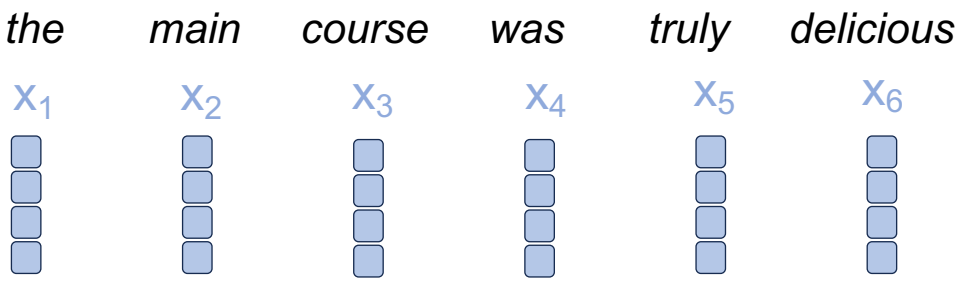


BruinFan298D
Los Angeles, CA

05/04/2026

The main course was truly delicious!

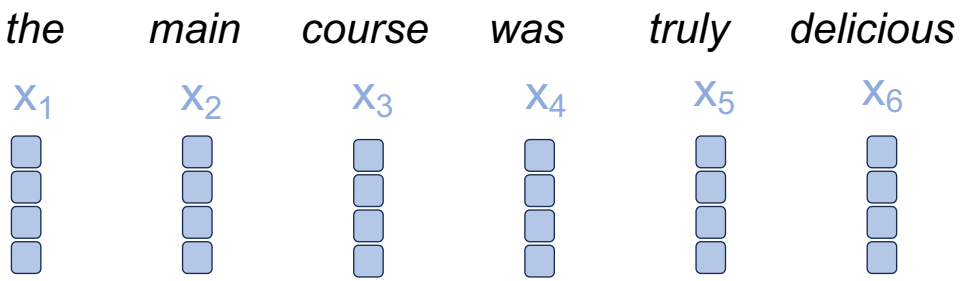
Context Matters



We can use stand-alone embeddings (like GloVe), but this ignores context.

Can we do better?

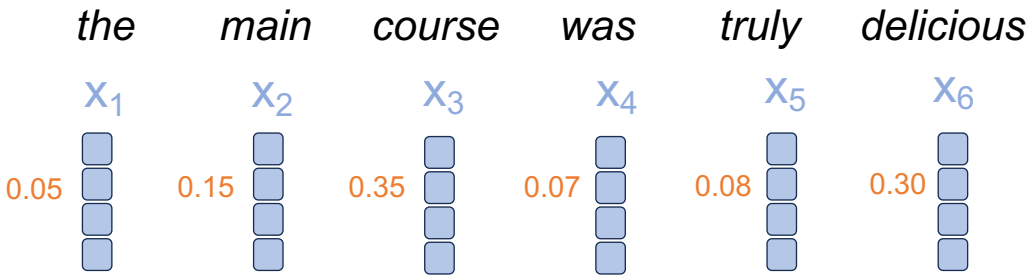
Contextual Embeddings



The big idea: *Update* the embedding of each word based on the “most relevant” words around it.

Which word is most relevant for **course**?

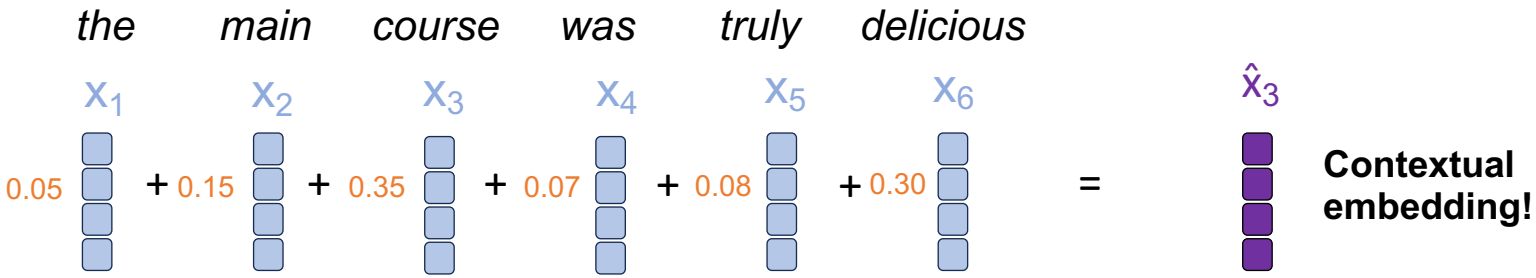
Contextual Embeddings



Suppose we could come up with weights for all the other tokens that reflect their related-ness to “course” in this particular context.

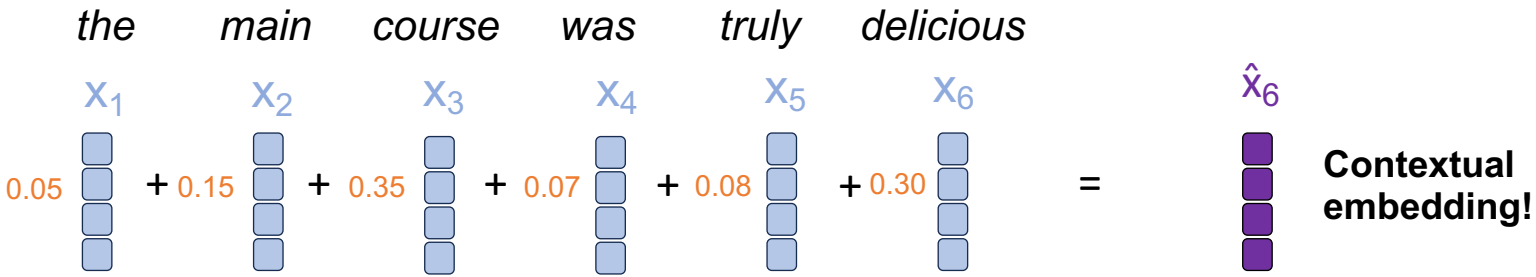
Clearly, “delicious” should have a high weight here.

Contextual Embeddings

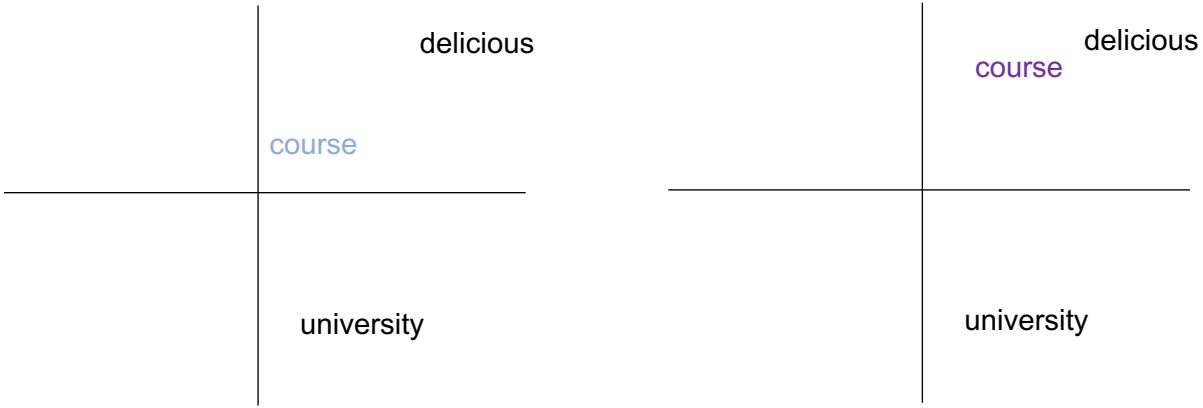


Then we can update the embedding for delicious by taking a weighted average:

Contextual Embeddings



What effect does this have? It pulls "course" closer to the location for "delicious" in the embedding space:



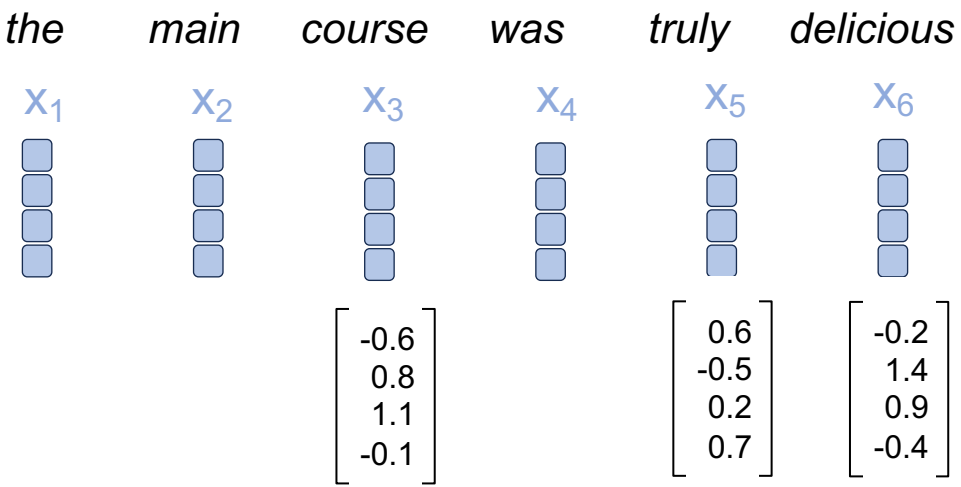
Contextual Embeddings

https://research.google/blog/transformer-a-novel-neural-network-architecture-for-language-understanding/

Where do attention weights come from?

Idea: Just use similarity of the embeddings themselves!

Intuition: Tokens that are close in the embedding space are semantically related, so we want them to attend strongly to each other



How to measure similarity between two embeddings? **Dot product:**

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \quad \mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 + a_3 b_3 + a_4 b_4$$

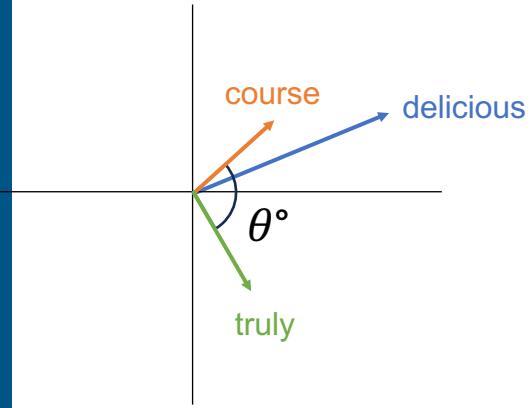
If embeddings **a** and **b** are "close", then **a**·**b** will be large.

Attention Weights

delicious course
$x_6 \cdot x_3 = \begin{bmatrix} -0.2 \\ 1.4 \\ 0.9 \\ -0.4 \end{bmatrix} \cdot \begin{bmatrix} -0.6 \\ 0.8 \\ 1.1 \\ -0.1 \end{bmatrix} = (-0.2)(-0.6) + (1.4)(0.8) + (0.9)(1.1) + (-0.4)(-0.1) = \mathbf{2.3}$

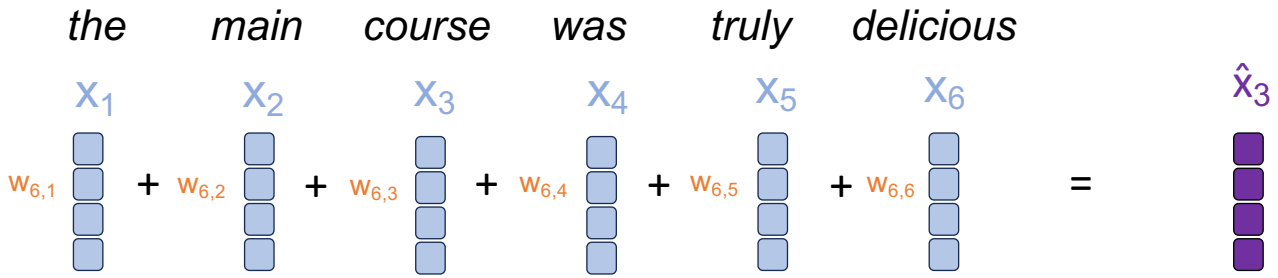
Larger dot product means **course** will attend more to **delicious** than **truly**

truly course
$x_5 \cdot x_3 = \begin{bmatrix} 0.6 \\ -0.5 \\ 0.2 \\ 0.7 \end{bmatrix} \cdot \begin{bmatrix} -0.6 \\ 0.8 \\ 1.1 \\ -0.1 \end{bmatrix} = (0.6)(-0.6) + (-0.5)(0.8) + (0.2)(1.1) + (0.7)(-0.1) = \mathbf{-0.6}$



After calculating all pairs of dot products, we run them through softmax to get positive weights that add up to 1.

Attention Weights



Every token embedding is re-weighted based on the attention it pay to all other tokens

$$w_{3,i} = \frac{e^{(x_i \cdot x_3)}}{e^{(x_1 \cdot x_3)} + e^{(x_2 \cdot x_3)} + e^{(x_2 \cdot x_3)} + e^{(x_3 \cdot x_3)} + e^{(x_4 \cdot x_3)} + e^{(x_5 \cdot x_3)} + e^{(x_6 \cdot x_3)}}$$ = *"the attention token 3 pays to token i"*

$$\hat{x_3} = w_{3,1}x_1 + w_{3,2}x_2 + w_{3,3}x_3 + w_{3,4}x_4 + w_{3,5}x_5 + w_{3,6}x_6$$

15-min Break

Part 3: Transformers

Transformers

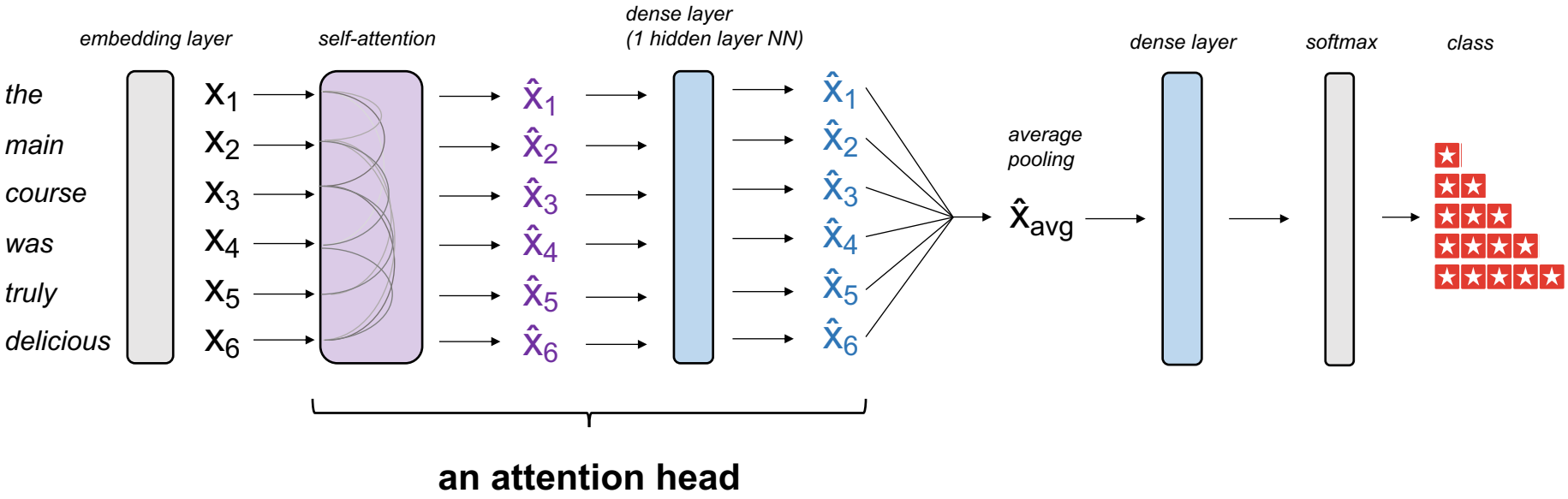
Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

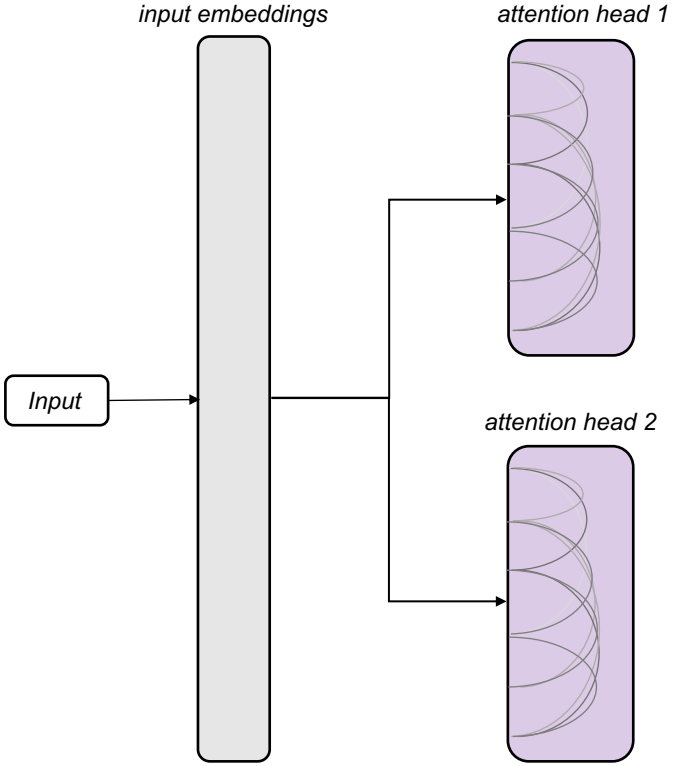
The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

Classifying with 1 Attention Head



Can we do better?

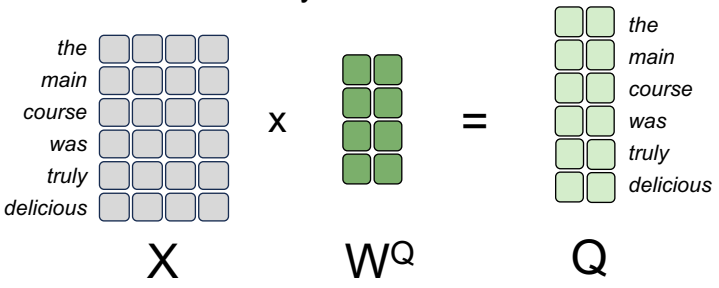
Multi-Headed Attention



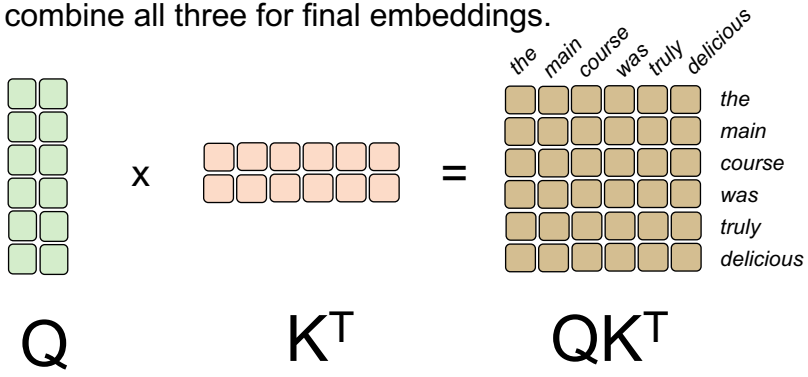
We can run attention heads in parallel. But how do we make each one learn something different?

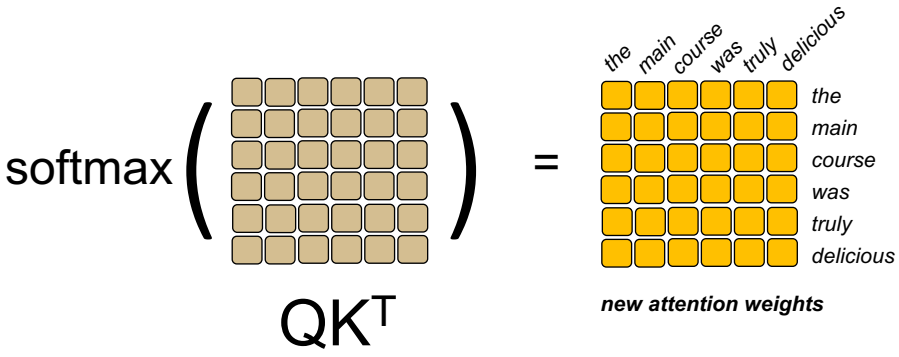
→ Introduce *weights* to each attention head

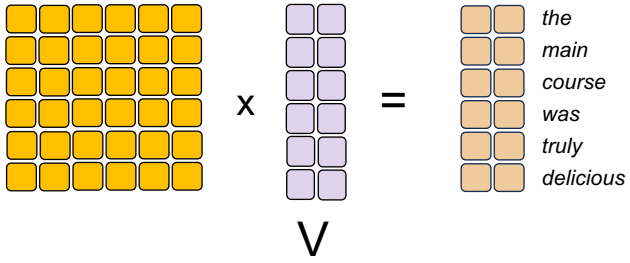
Make 3 copies of the input sequence and weight each one differently…



…then combine all three for final embeddings.

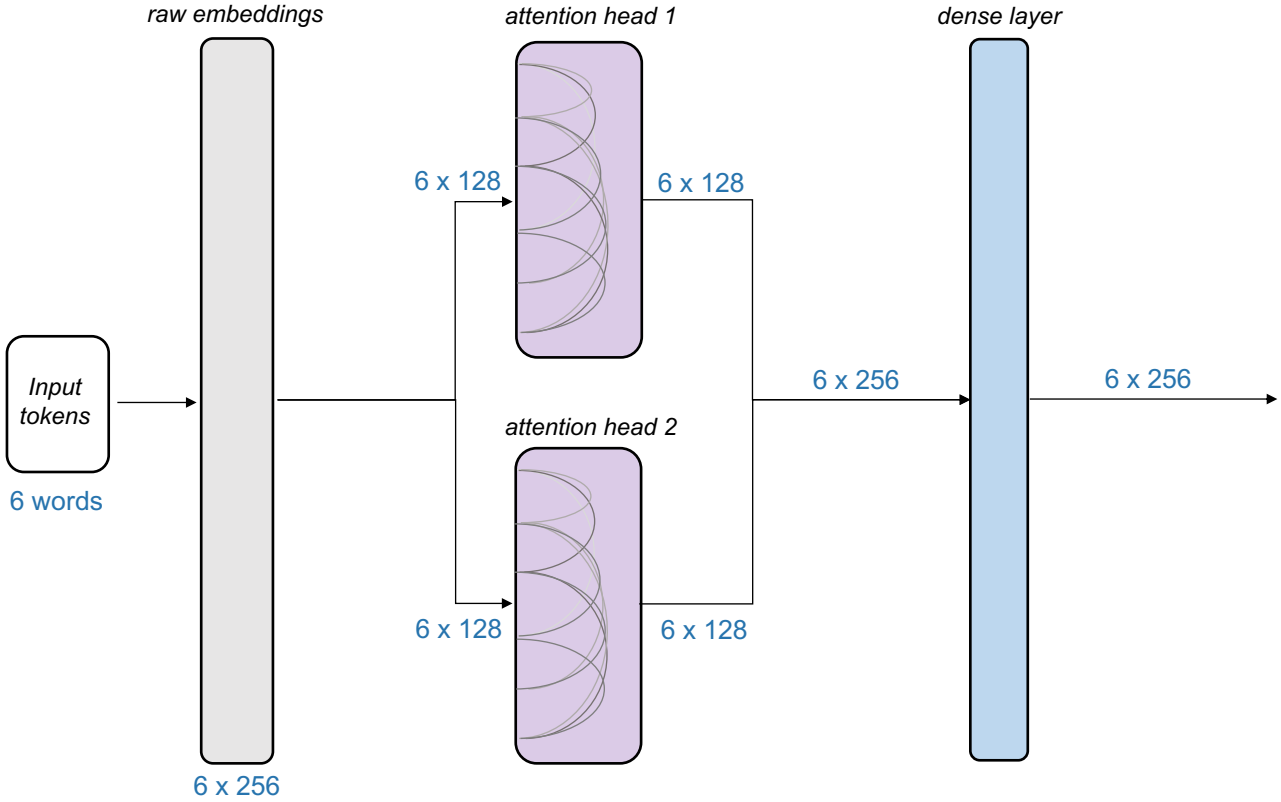






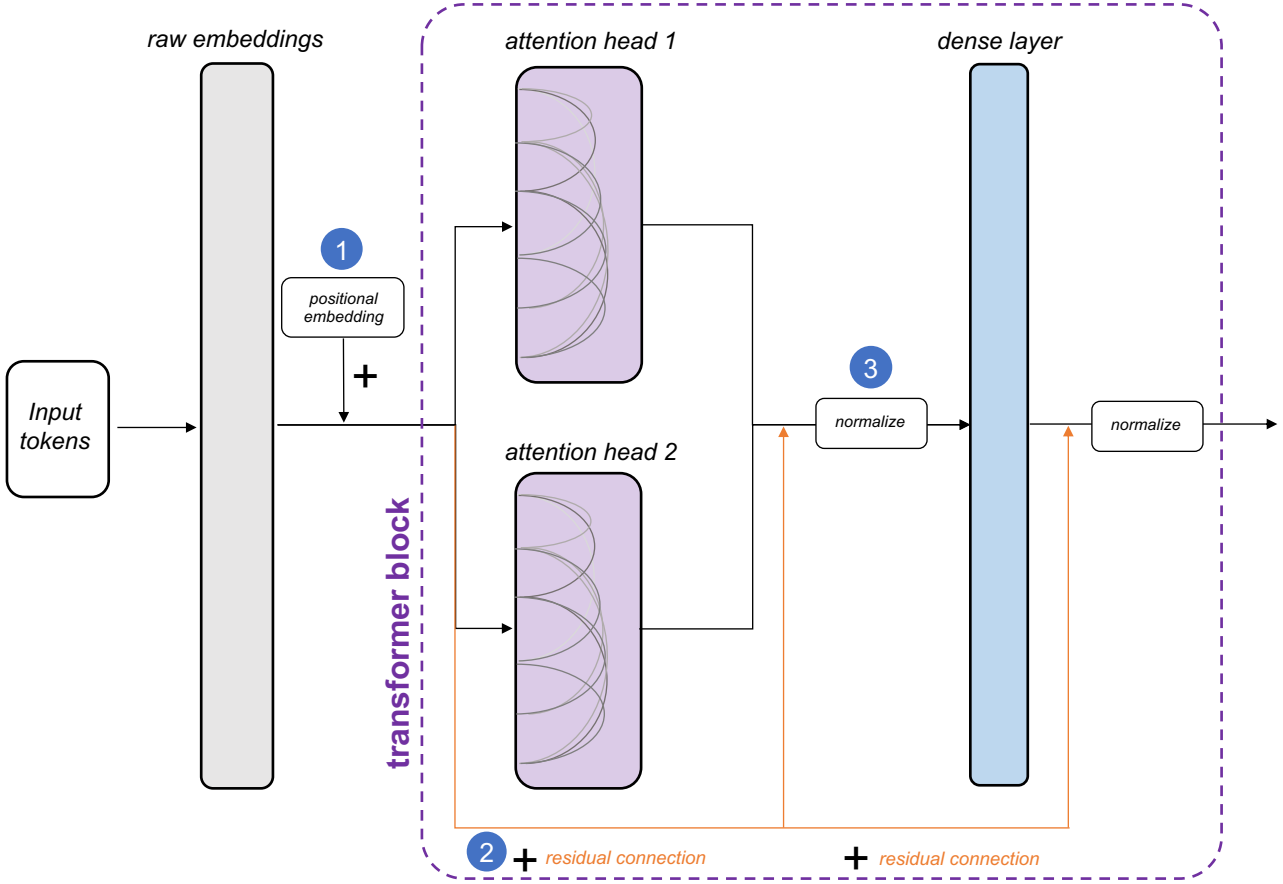
attention head output: new contextual embeddings

The Transformer Block



Example dimensions

The Transformer Block



Other improvements?

1 Positional embedding

Makes model aware of token positions.

2 Residual connection

Add original input embeddings to output of attention head and dense layer; passes original input forward.

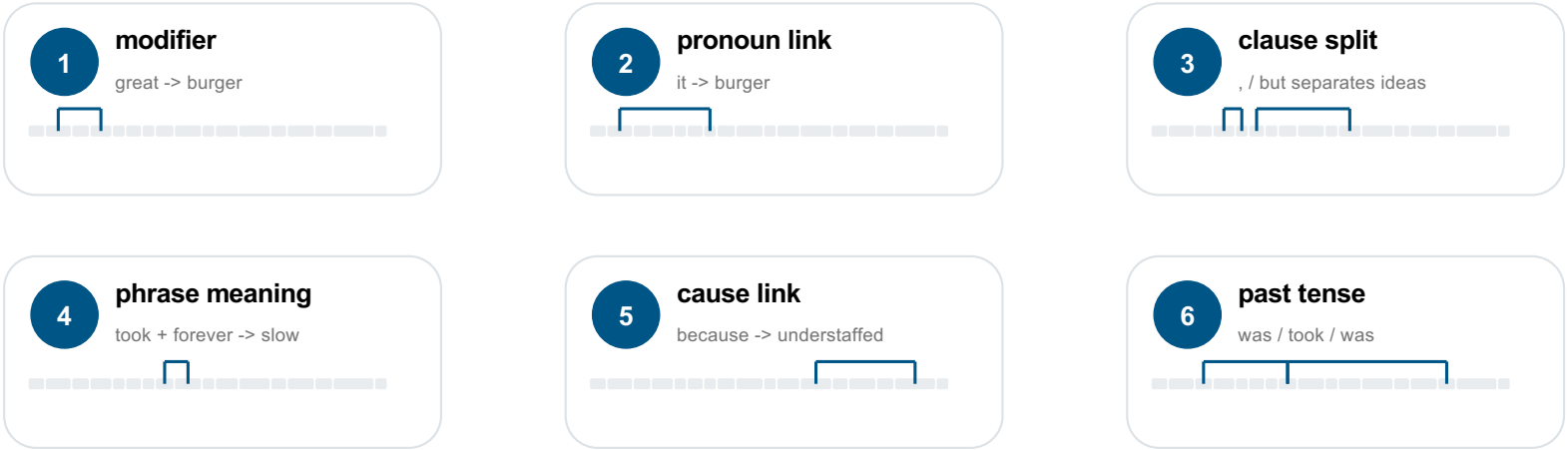
3 Normalization

Keeps values within a reasonable range to improve stability of gradients and backprop.

Why Use Multiple Heads?

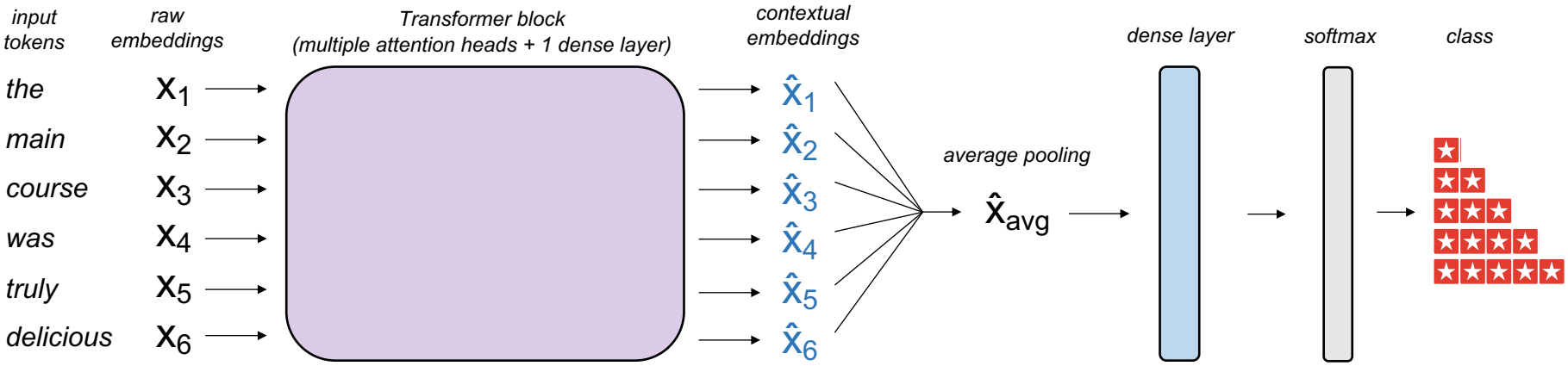
Each head learns a different attention pattern and specializes on what it detects

The	burger	was	great	,	but	it	took	forever	to	arrive	because	the	kitchen	was	understaffed	.



Multiple heads give the model many specialized “views” of the same text

Classification with 1 Transformer Block



Training Transformers

Same as any NN: Find weights by minimizing mismatch between true class label and model's probability for that label

Model's probability that training example *i* belongs to class *c*

$$\text{Loss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{ic}\,\log(p_{ic})$$

y=1 if training example *i* belongs to class *c*, 0 otherwise

Weights influence the assigned probability p, and are optimized through backprop algorithm

Yelp reviews example: Need lots of ($[\text{review text}]_i$, rating_i) training data

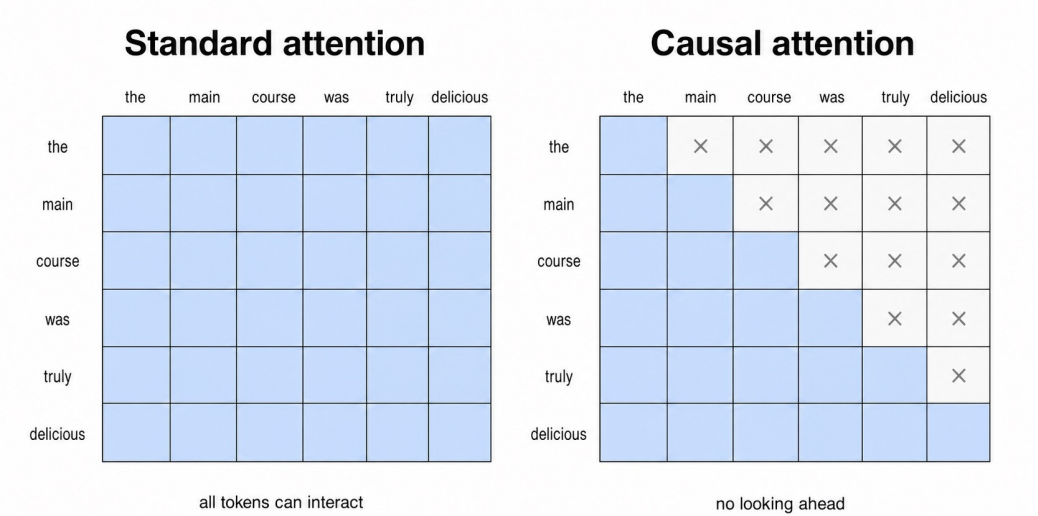
Large-Language Models: Next Token Prediction

Next Token Prediction

We considered prediction over only 5 classes (Yelp stars). But what about text?

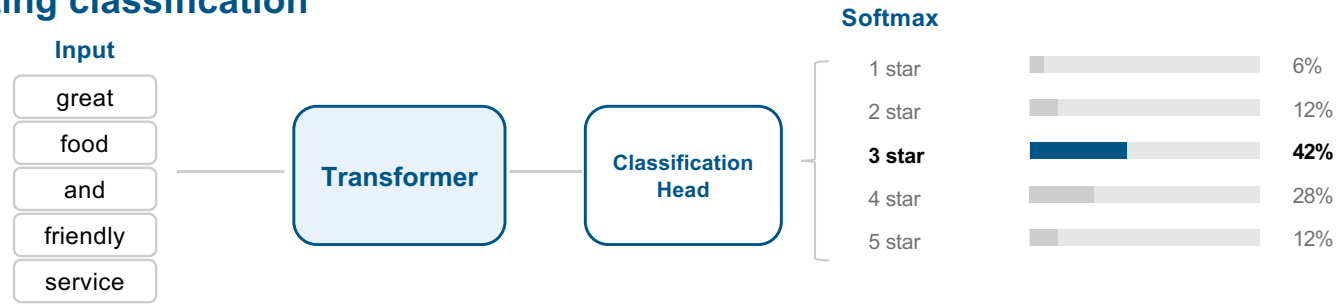
LLMs are based on **next-token prediction**

Requires adjusting attention mechanism to prevent “looking ahead”; no weights for later tokens



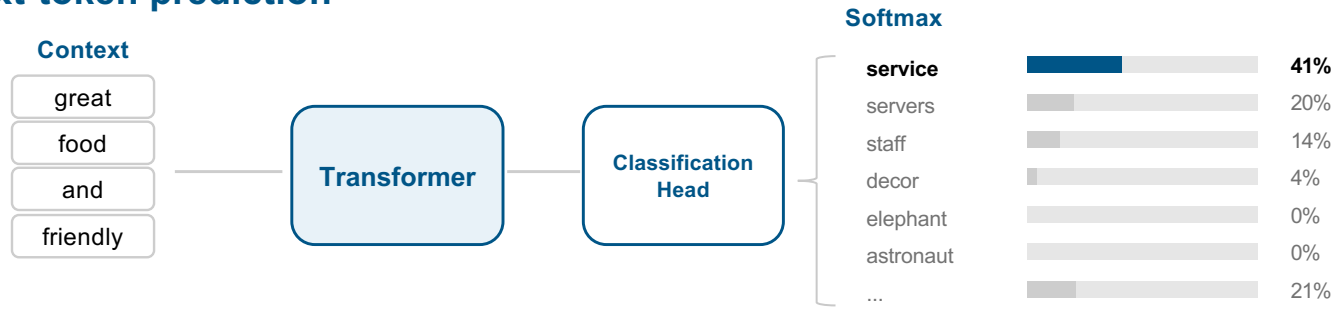
Next Token Prediction is Just Classification

Rating classification



softmax over a small set of classes

Next-token prediction

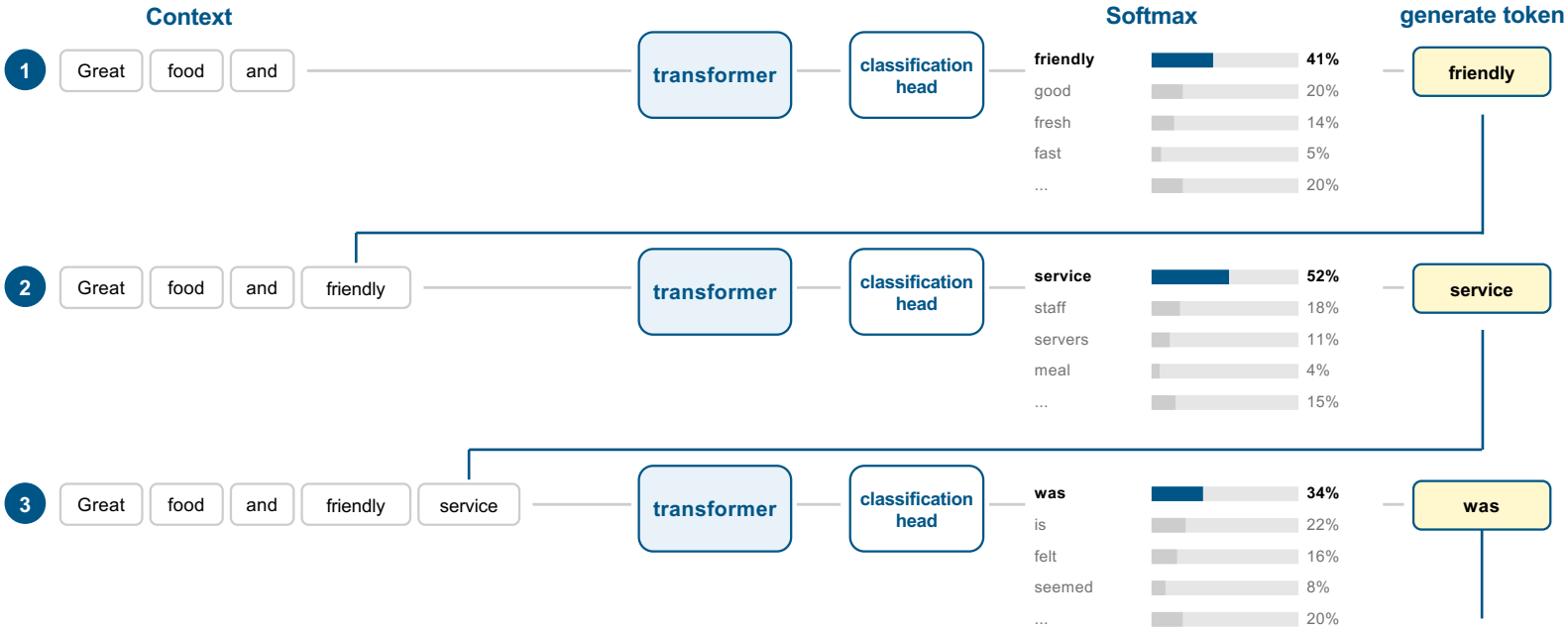


classes are all vocabulary tokens

Exact same transformer, different classification head

Text Generation

Predict, append, repeat



Each inference pass predicts one next token, appends it to the context, then runs the transformer again

Why Does Next-Token Prediction Work So Well?

Next token prediction is not the end goal, but just an extremely convenient training objective

Predicting the next token accurately forces the LLM to implicitly build an internal model of:

- language structure
- facts and domain knowledge
- relationship between concepts
- patterns of reasoning

Examples:

Gerrymandering can convert fewer votes into more	->	**seats**
When interest rates rise, existing bond prices usually	->	**fall**
To treat type 2 diabetes, doctors often prescribe	->	**metformin**

By focusing on next-token prediction, we naturally get billions of training examples from ordinary text

Transformers in Practice

Early GPT models were all stacks of transformer blocks

	GPT-1 (2018)	GPT-2 (2019)	GPT-3 (2020)
Transformer blocks	12	48	96
Attention heads per block	12	25	96
Embedding dimension	768	1,600	12,288
Dims / head	64	64	128
Weight parameters	117M	1.5B	175B
Context window	512 tokens	1,024 tokens	2,048 tokens
Training tokens	~0.8B	~10B	~300B
Training data	BookCorpus (~5 GB)	WebText (40 GB)	Common Crawl + books + Wikipedia (570 GB)


colab

Assignment 6

Glossary (1/2)

Token	The basic unit of text a model reads or generates.
Vocabulary	The full set of tokens the model can recognize or generate.
Corpus	A large collection of text used to learn language patterns.
Embedding	A numerical representation of a token learned from data.
Contextual Embedding	A token representation that changes based on the surrounding words.

Glossary (2/2)

Attention Weight	A learned score indicating how much one token uses information from another token.
Transformer	The architecture behind modern LLMs, built around attention-based processing.
Context Window	The maximum number of tokens a transformer model can process in one input sequence, including the prompt and generated tokens kept in context.