Universität Konstanz

08 | Attention and Transformers

Giordano De Marzo

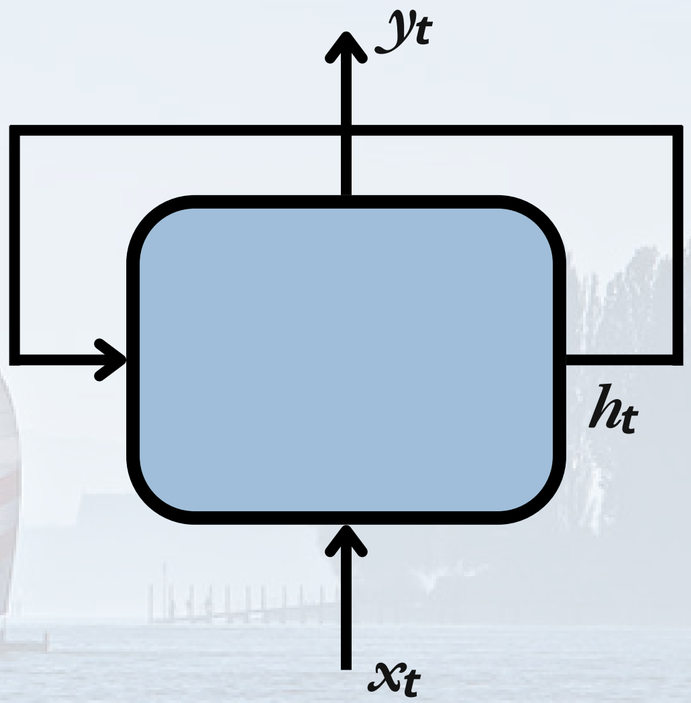
https://giordano-demarzo.github.io/

Deep Learning for the Social Sciences

Recap: Recurrent Neural Networks

In order to capture temporal dependencies in the data we add a feedback loop obtaining a so called Recurrent Neural Network (RNN)

- a RNN is characterized by an hidden state $\mathbf{h_t}$
- at each time step the RNN takes in input the value $\mathbf{x_t}$ from the sequence and the previous hidden state $\mathbf{h_{t-1}}$
- it then produces and output y_t and a new hidden state $\mathbf{h_{t-1}}$

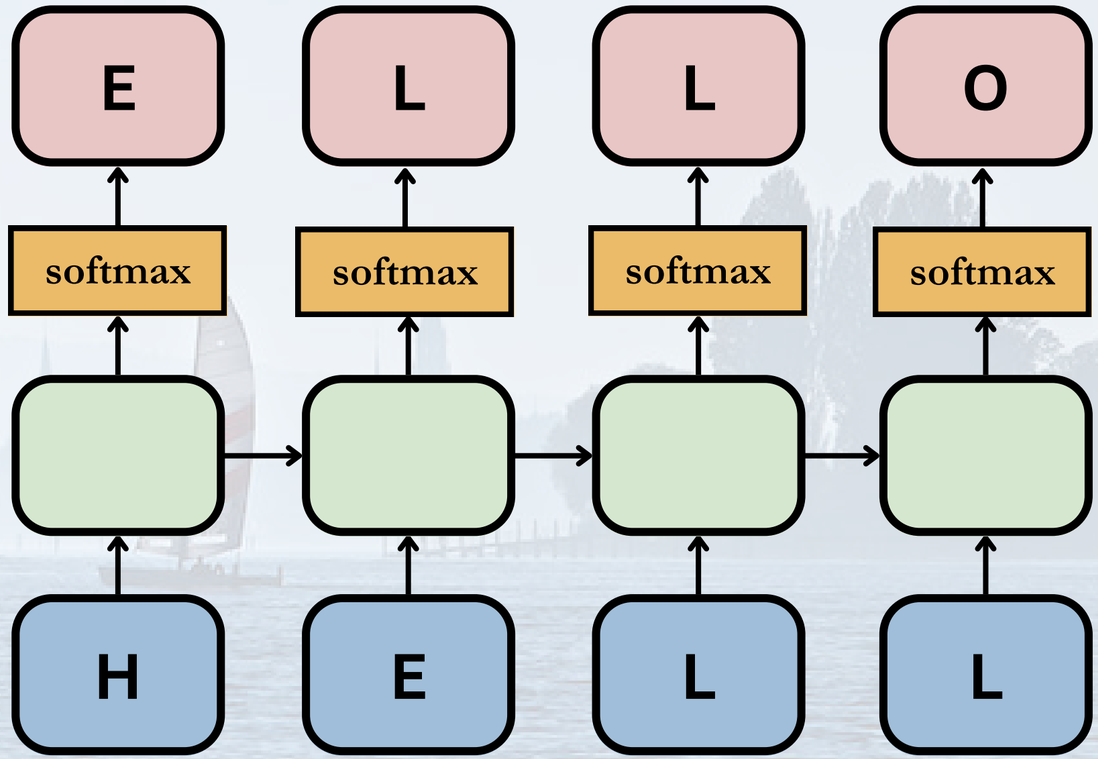


$$h_t = f_W(h_{t-1}, x_t)$$
$$y_t = g_W(h_t)$$

Recap: Text Generation with RNNs

RNNs can be used to generate text by training them to forecast the next character (or word)

- we input a letter
- the output of the unit is passed through a softmax
- the next character is sampled from the probability distribution
- the output character is used as the next input
- the process is iterated



Universität Konstanz

Outline

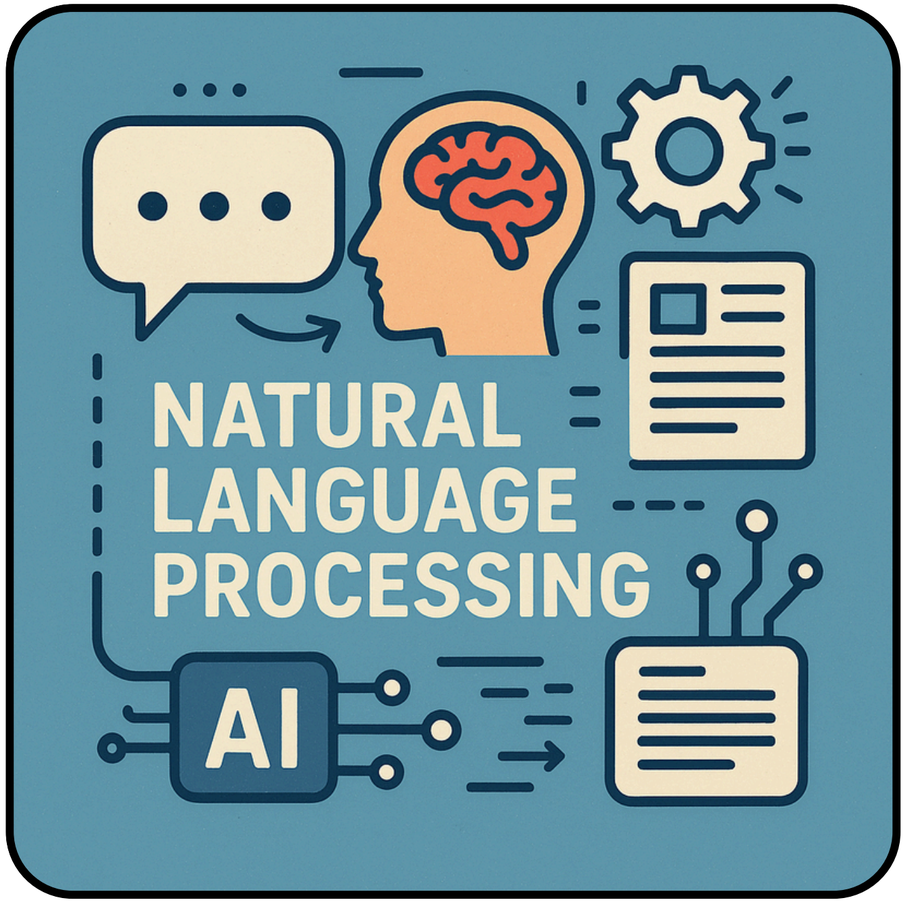
1. Natural Language Processing
2. Word Embeddings
3. Attention Mechanism
4. Transformer Architecture


Universität
Konstanz
Natural Language Processing

What is Natural Language Processing?

Natural Language Processing bridges the gap between human communication and computer understanding. It's the field that teaches machines to work with the language we naturally use every day.

- Definition: AI field focused on interaction between computers and human language
- Core goal: Make computers understand, interpret, and generate human language naturally
- The challenge: Transform messy, ambiguous human language into structured data computers can process



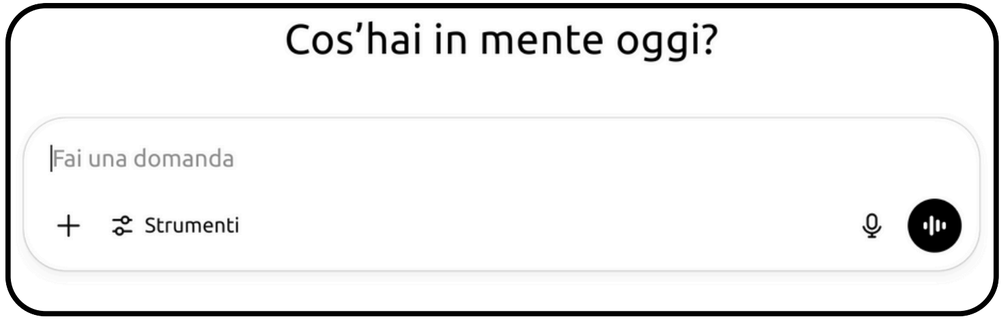
NLP Applications in Your Daily Life

We interact with NLP systems dozens of times every day, often without realizing it. These technologies have become so seamless that they feel like magic.

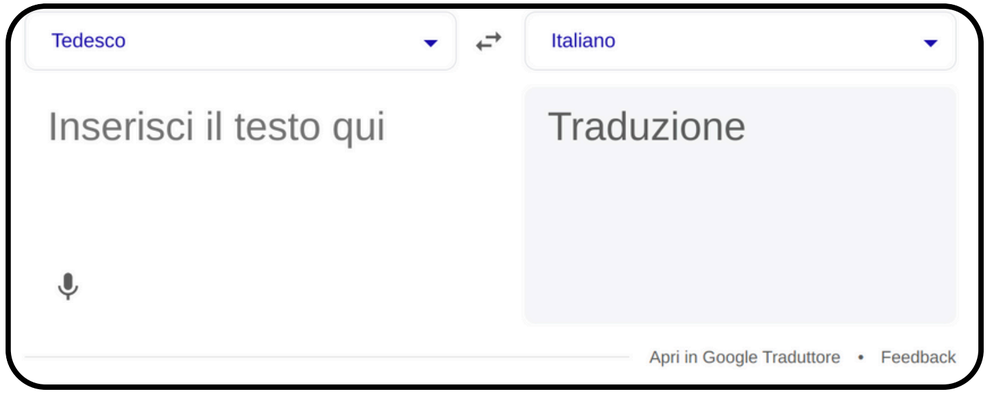
- Google Translate and Google Search
- ChatGPT & Virtual Assistants
- Email spam detection
- Sentiment analysis
- Auto-complete & suggestions

NLP is one of the most relevant domain where Deep Learning is applied

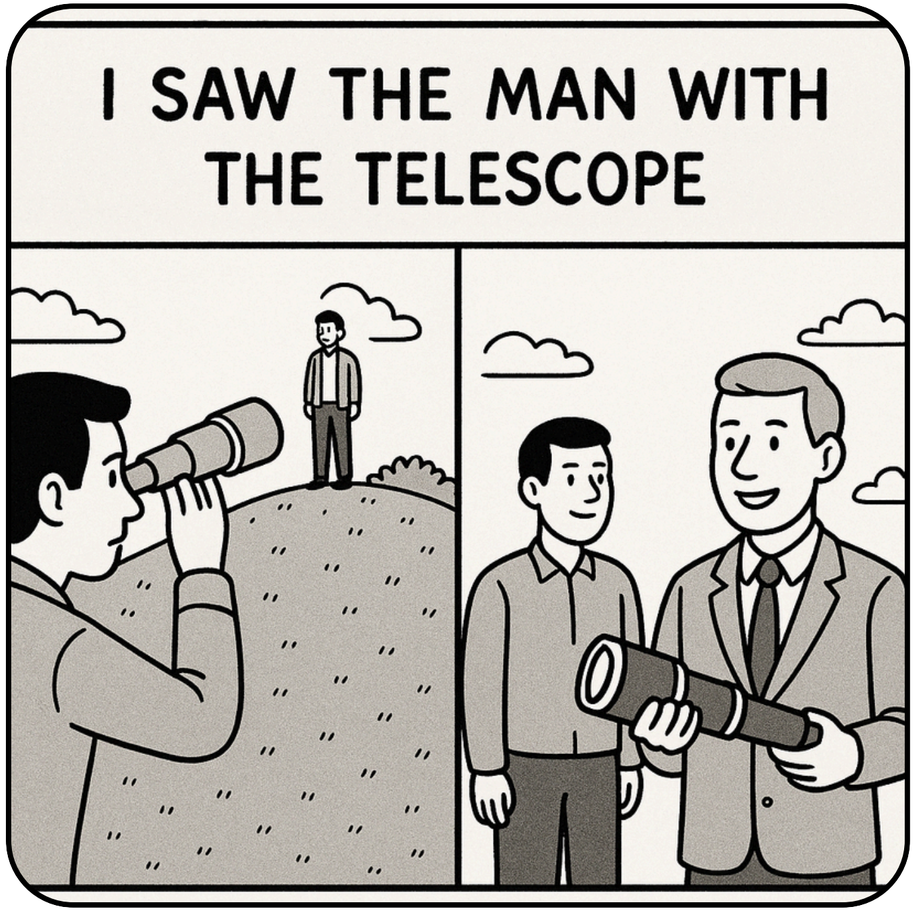
ChatGPT



Google Translate



NLP Challenges



Human language is incredibly complex and context-dependent. What seems obvious to us is actually a nightmare for computers to understand.

- Ambiguity: "I saw the man with the telescope"
- Context dependency: "Bank" (river vs. financial institution)
- Sarcasm and irony: "Great weather!" during a storm
- Cultural references and idioms
- Grammar variations and informal language

On top of this we need also to consider the presence of long distance correlations

Limits of RNN in NLP

Despite their success, RNNs have fundamental limitations that become obvious with longer texts

- **Sequential bottleneck:** Must process one word at a time - no parallelization possible
- **Vanishing memory:** Information from early words gradually fades away
- **Distance problem:** Struggle to connect words that are far apart in a sentence

This isn't just an engineering problem - it's a fundamental architectural limitation.

```
VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:
O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.
```

The Need for a Revolution

The limitations of RNNs pointed us toward what we really needed in language processing.

- **Parallel processing:** Handle all words simultaneously, not sequentially
- **Long-range connections:** Connect any word to any other word directly
- **Scalable memory:** Remember important information regardless of distance
- **Efficient computation:** Handle long documents without exponential slowdown

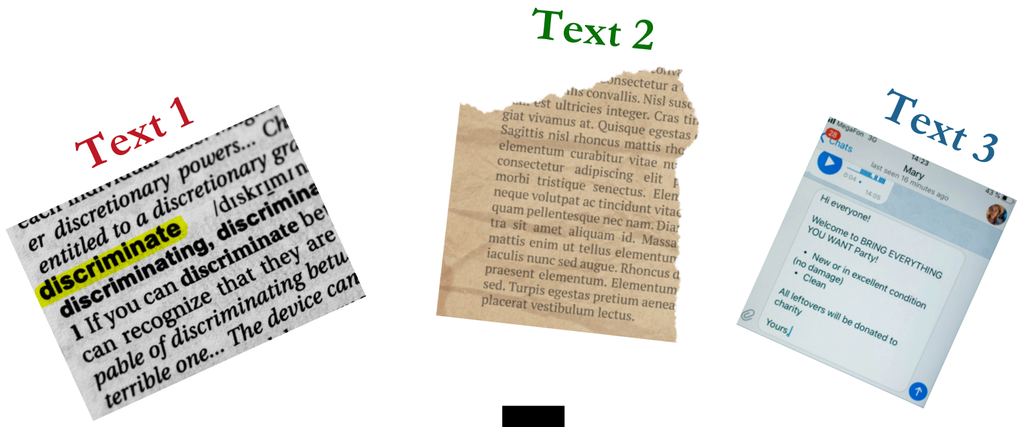
The breakthrough came in 2017 with a paper titled "Attention is All You Need." The solution: Replace recurrence with attention mechanisms.

Vaswani, Ashish, et al. "*Attention is all you need.*" Advances in neural information processing systems 30 (2017).

Universität
Konstanz

Word Embeddings

From Words to Numbers



$$\text{Text 1} = \begin{bmatrix} 1.2 \\ -0.3 \\ 1.8 \\ \vdots \\ 0.6 \\ 0.1 \\ -0.7 \end{bmatrix} \quad \text{Text 2} = \begin{bmatrix} 0.4 \\ 1.1 \\ -0.9 \\ \vdots \\ -0.2 \\ 0.8 \\ -1.2 \end{bmatrix} \quad \text{Text 3} = \begin{bmatrix} -0.8 \\ 0.6 \\ -0.6 \\ \vdots \\ 1.5 \\ 0.3 \\ 1.4 \end{bmatrix}$$

Before we can train language models, we need to solve a fundamental problem

- **The basic problem:** Computers only understand numbers, not words
- **Why this matters:** Everything in NLP starts with converting language to mathematics

We need to bridge the gap between human language and machine computation.

- This is the first time we encounter this problem
- Images have a natural matrix representation
- Graphs are stored as matrices
- What about texts?

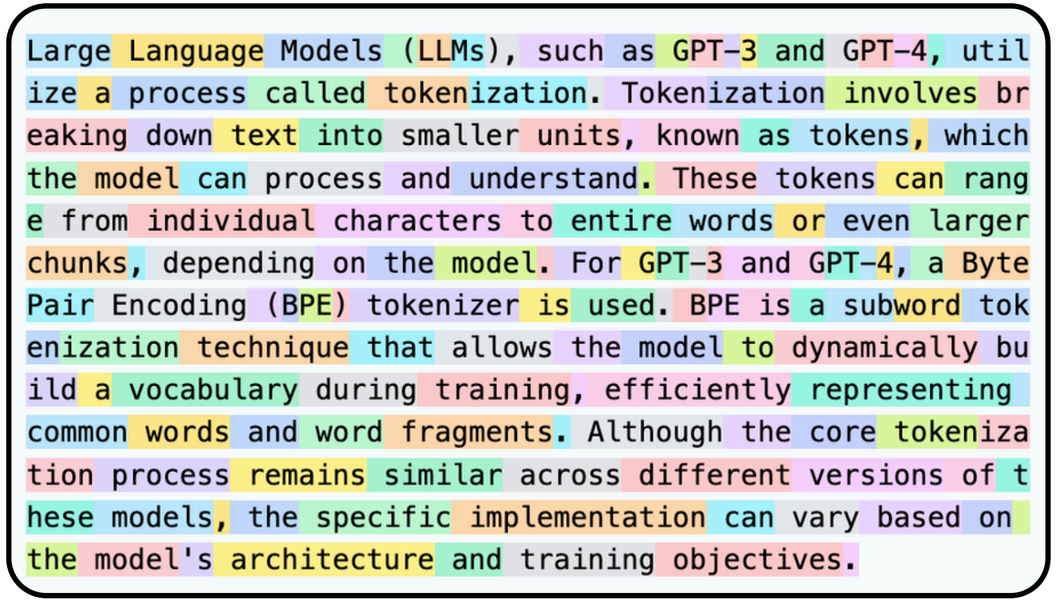
Tokenization

The first step in any NLP pipeline is breaking text into manageable pieces.

- Definition: Splitting text into individual units (tokens) that computers can process
  - "Hello world!" → ["Hello", "world", "!"]
- The tokenizer is characterized by a vocabulary
  - Each word in the vocabulary is assigned a number or code
  - The text is transformed into numbers

Modern approach: Subword tokenization

- Examples are WordPiece or BPE
- Less common words tend to be split



Tokenization example generated by Llama-3-8B. Each colored subword represents a distinct token.
https://medium.com/data-science/the-art-of-tokenization-breaking-down-text-for-ai-43c7bccaed25

One-hot Encoding

the	1	0	0	0
cat	0	1	0	0
and	0	0	1	0
the	1	0	0	0
dog	0	0	0	1

The first attempt at converting tokens to numbers uses one-hot encoding

- The idea: Create a binary vector for each word in our vocabulary
  - Each vector has as many component as the vocabulary size
  - Only one entry is equal to one

Problems with one-hot encoding:

- **Dimension explosion**
- **No relationships:** "cat" and "dog" are equally distant from each other as "cat" and "airplane"
- **Sparse and inefficient:** Mostly zeros, waste of memory

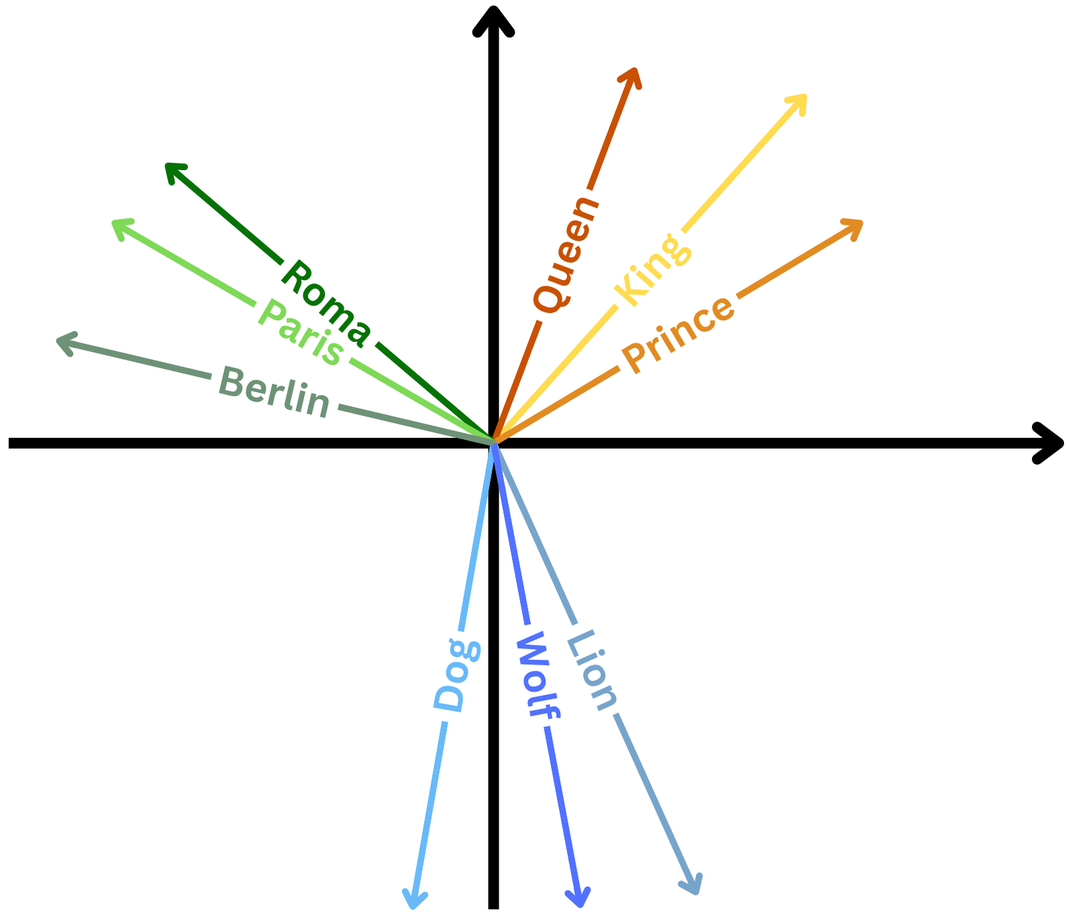
What Are Word Embeddings?

Word embeddings solve the problems of one-hot encoding by creating dense, meaningful representations.

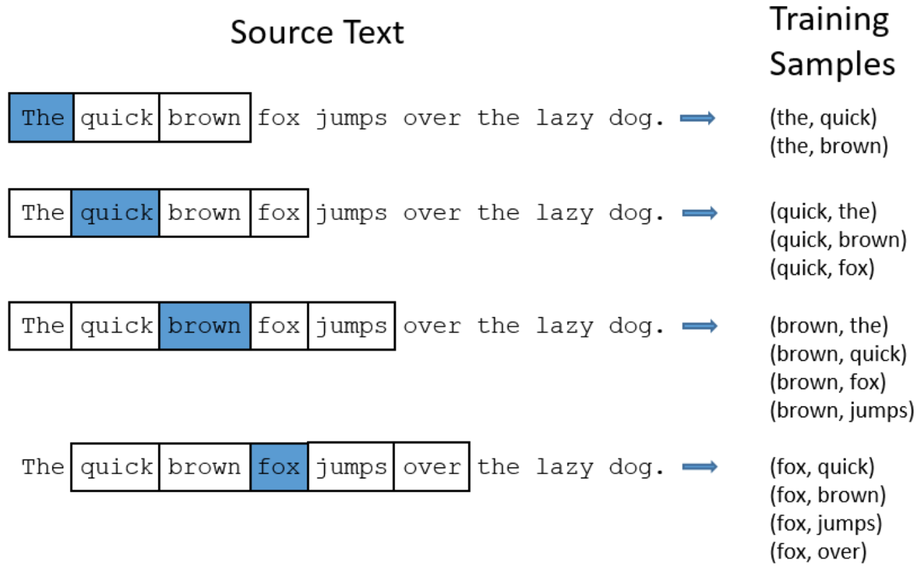
- They are coordinates in a "meaning space."
- Words are points in high-dimensional space
- Each word becomes a list of real numbers (typically 100+ dimensions)
- Similar words should have similar vectors
- Closer points = more similar meanings

Advantages over one-hot encoding:

- Compact (100/1000 dimensions vs 50,000+)
- Captures semantic relationships
- Works for words not seen during training



Word2Vec: Learning Word Meanings



Word2Vec if one of the first embedding models

- **Core insight:** "You shall know a word by the company it keeps" (J.R. Firth, 1957)
- **Training task:** Given a center word, predict its surrounding context words
- Input "sits" → predict ["cat", "on", "the", "mat"]

How it works step by step:

- Slide a window across millions of sentences
- For each center word, predict its neighbors
- Neural network adjusts word embedding vectors
- Words that appear in similar contexts end up with similar vectors

The Amazing Properties of Embeddings

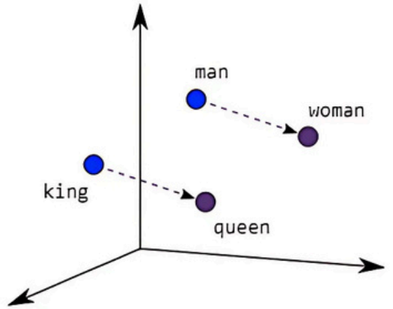
Once trained, word embeddings exhibit remarkable mathematical properties that mirror human language understanding. Vector arithmetic becomes semantic reasoning, for instance:

- Semantic relationships: King - Man + Woman ≈ Queen
- Geographic patterns: Paris - France + Italy ≈ Rome

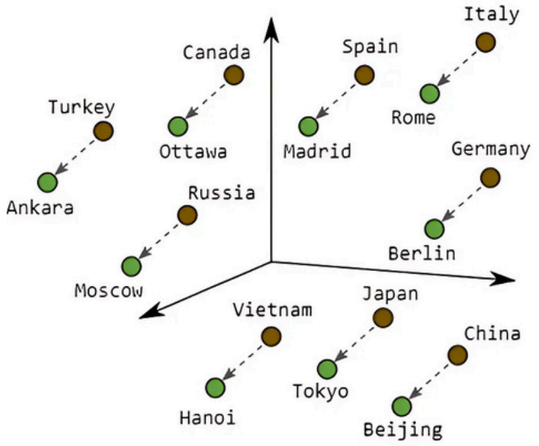
The interpretation is that different directions in the embedding space encode different concepts or meanings (e.g. “royalty” direction or “capital city” direction)

The embedding space organizes itself to reflect real-world relationships. These weren't programmed in - they emerged naturally from training on text.

Male-Female



Country-Capital



Visualizing Embedding Spaces

Embedding spaces are extremely high dimensional

- Visualizing them directly is impossibile!

Often embedding spaces are visualized using dimensionality reduction techniques (t-SNE, UMAP)

- **Clustering behavior:** Similar words and texts naturally group together
- **Semantic neighborhoods:** Countries cluster, animals cluster, colors clusters

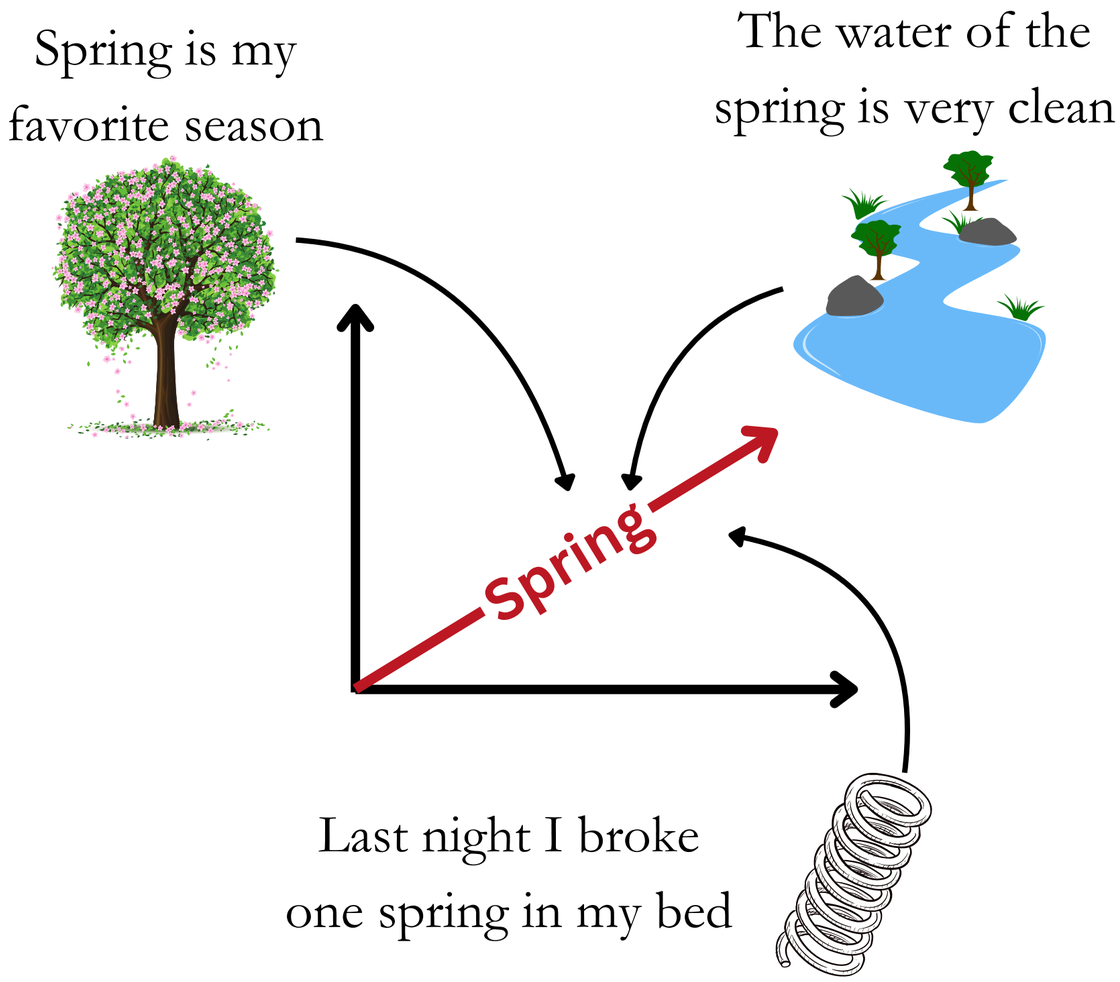
Dimensionality reduction is often a good test to understand if the embedding are meaningful

Limitations of Static Embeddings

Static embeddings have a fundamental flaw: they give each word exactly one representation, regardless of context

- "Spring" always gets the same vector
- However the meaning of “Spring” depends on the context it is used in
- Many words have multiple meanings
- Word2Vec can’t distinguish between different uses of the same word

The solution to this are contextual embeddings that change based on surrounding words.

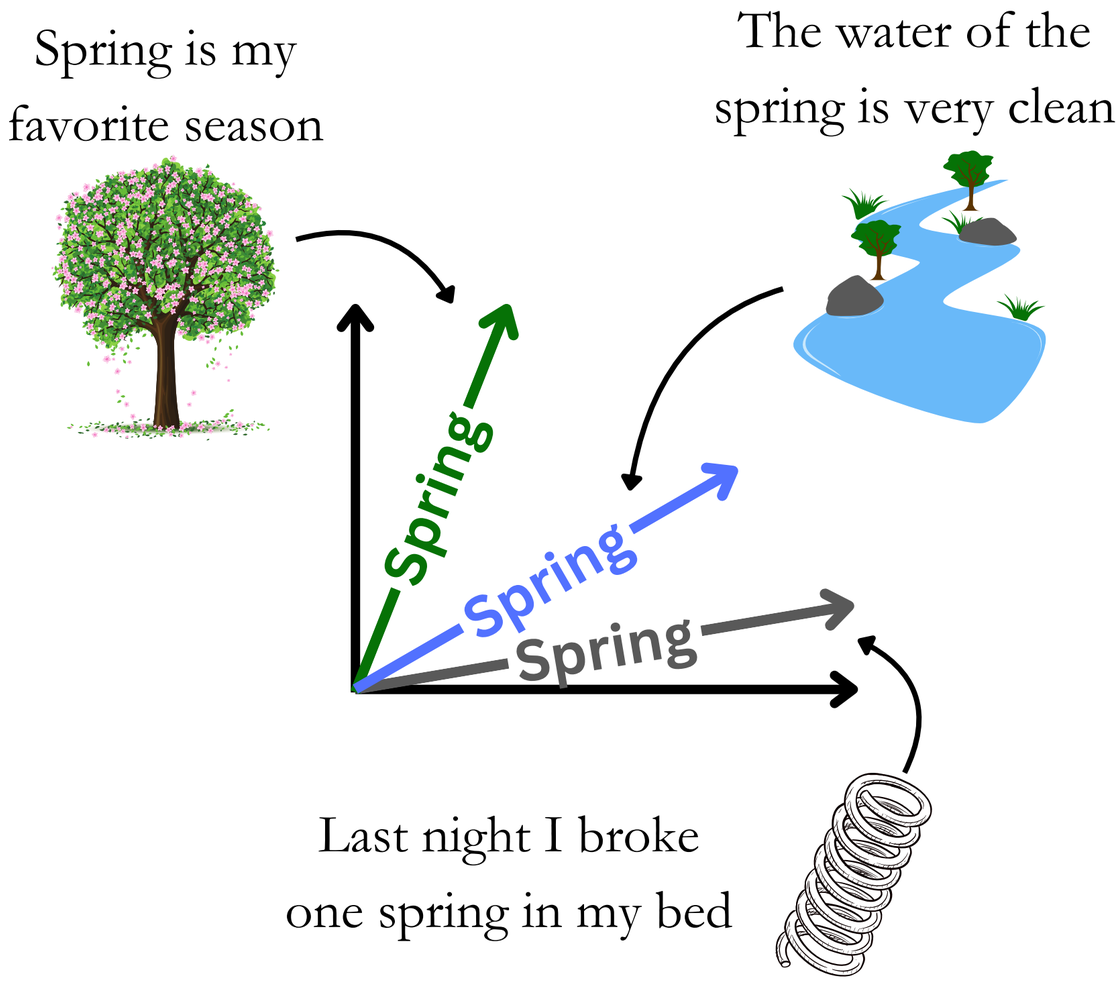


Beyond Static Embeddings

Static embeddings have a fundamental limitation: one word, one meaning, regardless of context

- **What we need:** Dynamic embeddings that change based on context
- **The key insight:** Words should influence each other's representations
- **The solution:** Let words "look around" and adjust their meaning based on what they see

The result is that the same word will be embedded in different ways depending on how it is used and its context



The Attention Revolution

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 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

In 2017, a single paper changed everything in AI. "Attention Is All You Need" introduced a mechanism that solved the context problem and became the foundation of modern language models.

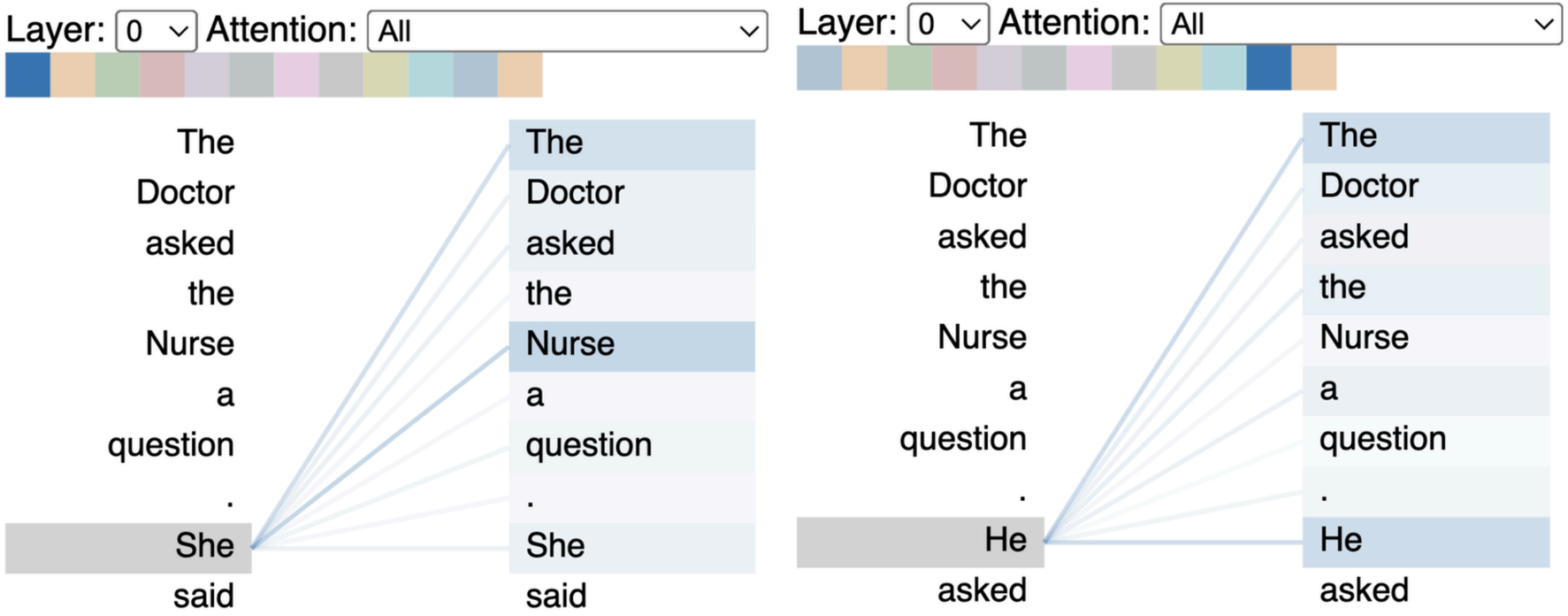
- Words pay attention to other words
- Inspired to how humans focus on relevant information when reading
- Words can directly influence each other's representations
- Attention gives the model the ability to focus on relevant words

This paper launched the era of large language models, from BERT to ChatGPT

What is Attention Doing?

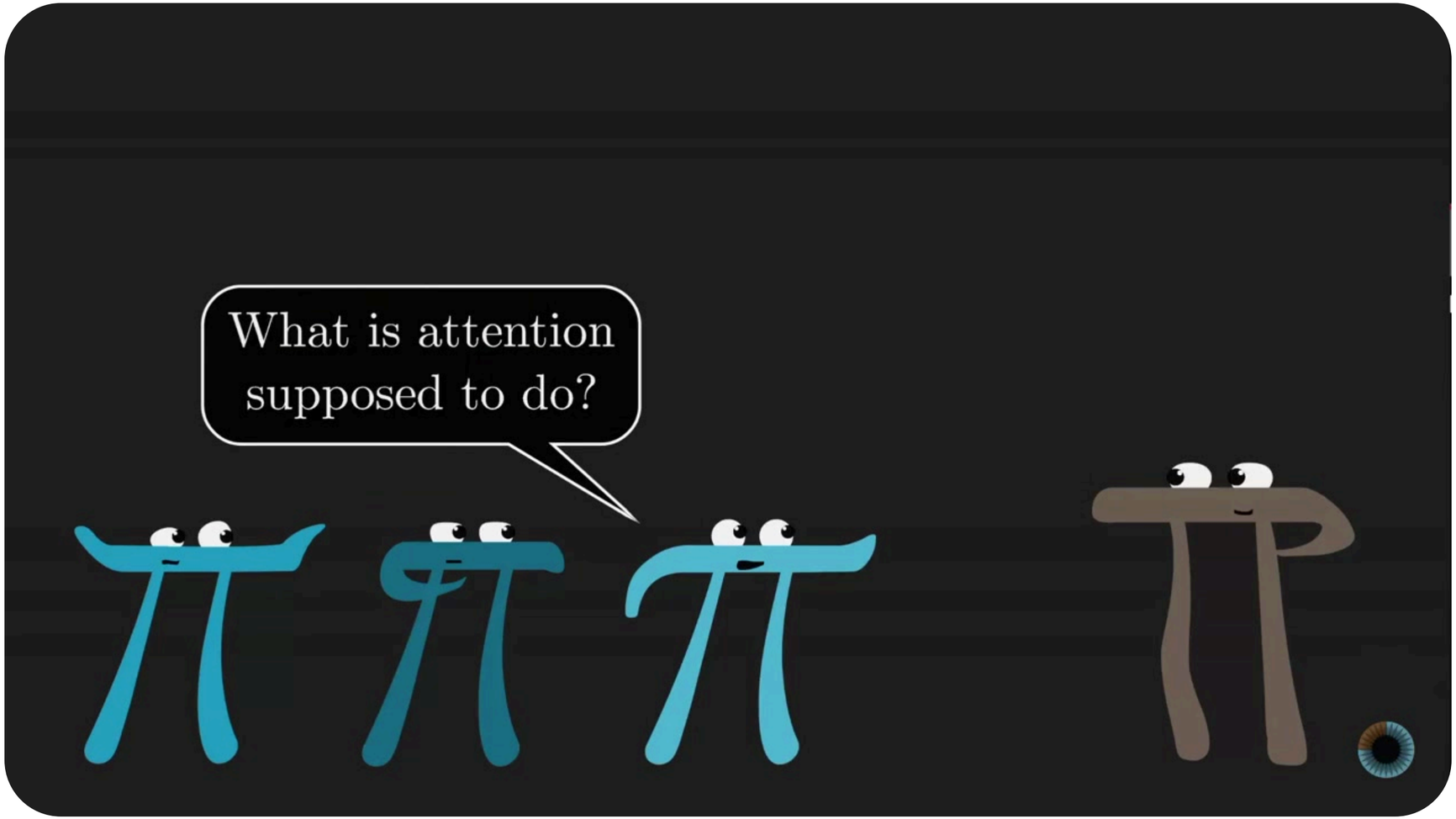
Attention creates connections between words, allowing them to influence each other's representations.

Instead of isolated meanings, words now share information based on relevance and context.



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

Example: Attention



Attention Mechanism

The Attention Mechanism in Short

Attention works in three logical steps that mirror how humans focus on relevant information

Step 1: Find the relevant words (Query ↔ Key matching)

- Each word asks "what should I pay attention to?" (Query)
- Each word advertises "here's what I can offer" (Key)
- We match queries with keys to find relevance (attention) scores

Step 2: Understand how relevant words should influence others (Values)

- Each word provides "here's the information I want to share" (Value)
- Values contain the actual content that will modify other words
- Think of values as the "message" each word wants to send

Step 3: Apply the modifications (Weighted combination)

- Use attention scores to blend information from relevant words
- Each word gets updated based on what it paid attention to
- Result: context-aware representations

Input Text and Initial Embeddings

In the beginning we start with just a simple raw text:

A fluffy blue creature roamed the verdant forest

The first step is converting each word from the input text into static embeddings, like with Word2Vec

- Starting point: Each word gets converted to its embedding vector
- Key insight: These are static embeddings - each word always starts with the same vector.
- What's coming: Attention will create dynamic, context-aware versions of these embeddings.

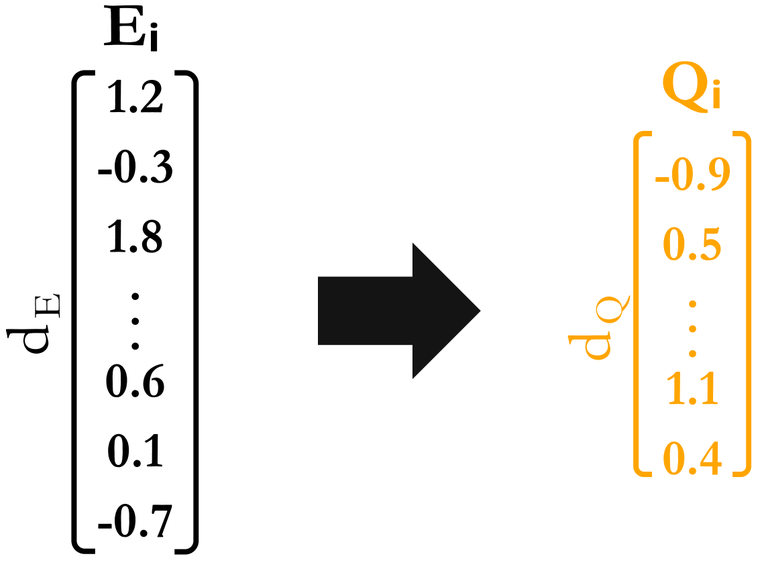
The magic happens when these static representations start talking to each other through attention.

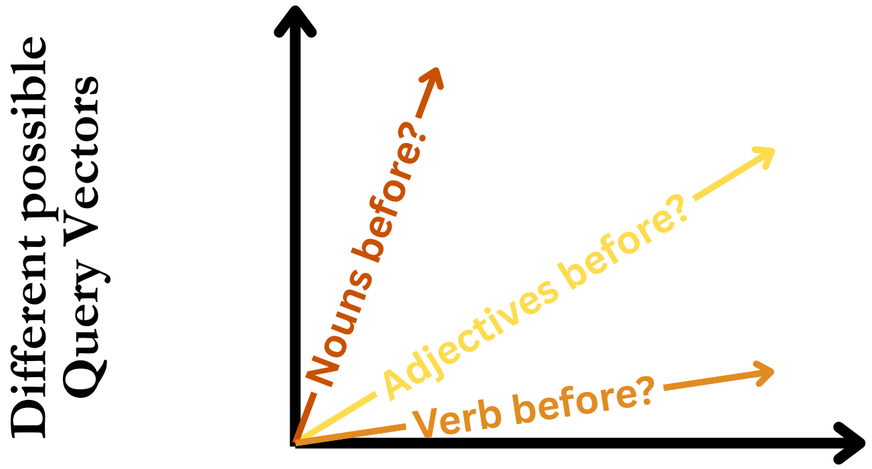
Query Vectors

Every word asks "**what information do I need from other words?**" The query vectors express these questions

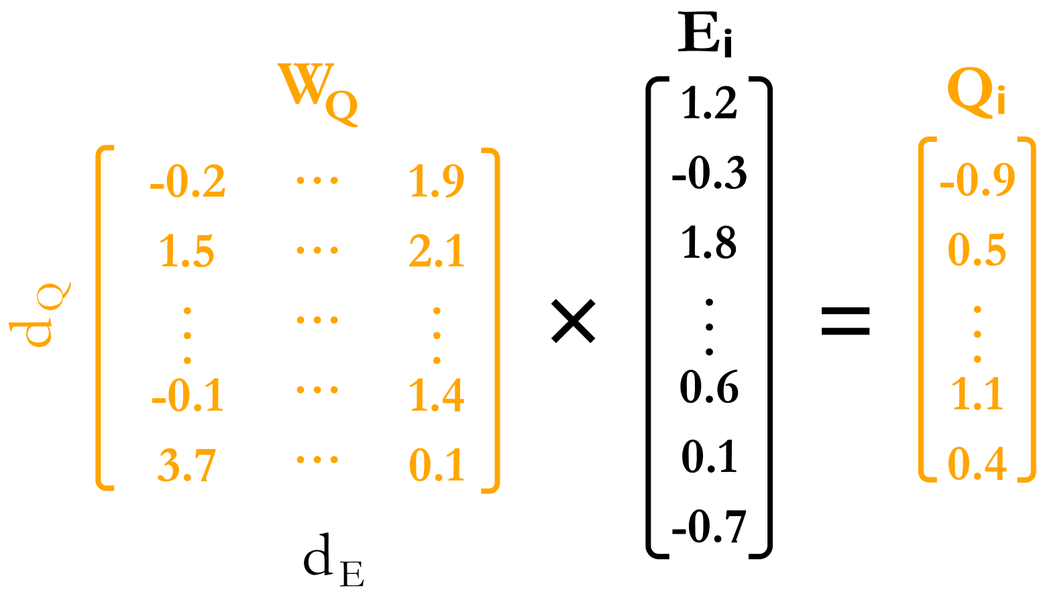
- **Purpose:** Each word creates a query vector to find relevant information
- **Example intuition:**
  - Verb "roamed" might query for: "show me the subject and object"
  - Noun "creature" might query for: "show me what adjectives refer to me"

Different words create different queries vectors based on what they need to understand. These vectors live in a smaller dimensional space





The Query Matrix



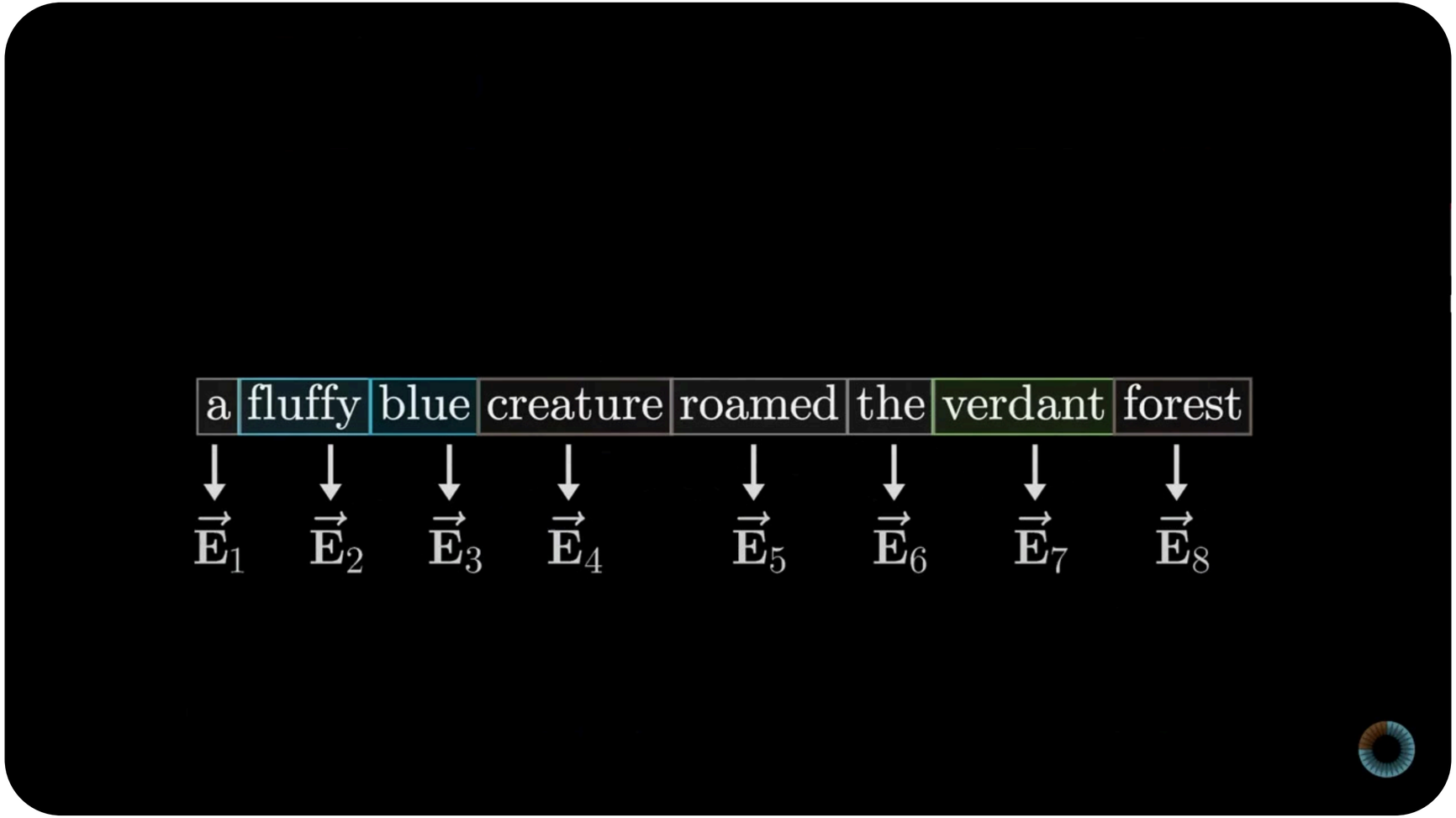
The Query Matrix is a matrix of learnable parameters that transforms initial embeddings into query vectors.

- **Dimensions:** [embedding_dimension × query_space_dimension]
- **Function:** Transforms any input embedding into a query vector by matrix multiplication

The same transformation matrix is used for all words in all positions. We are effectively reusing parameters across the whole text

- “creature” × W_Q = query expressing what “creature” needs to know

Example: Query



Key Vectors and Matrix

While queries ask “what do I need?”, keys answer “what can I provide?”. Every word advertises what information it has available

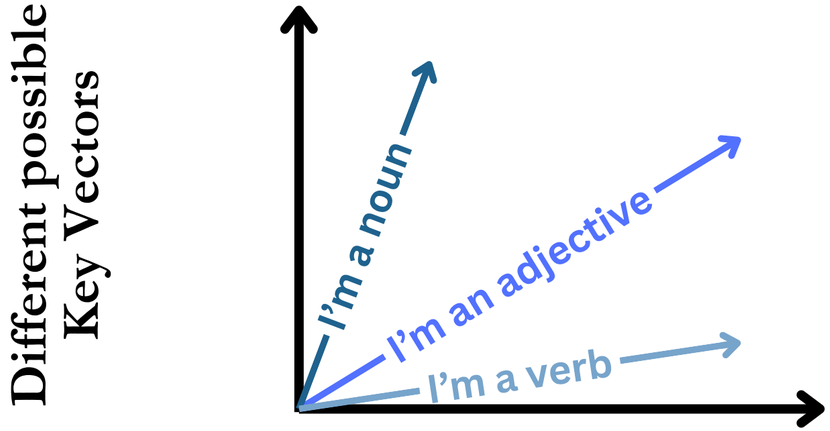
- This information is encoded in the key vectors
- They live in the same space as the query vectors (same dimension)
- These vectors are computed multiplying the key matrix for the embedding vectors

Example intuition:

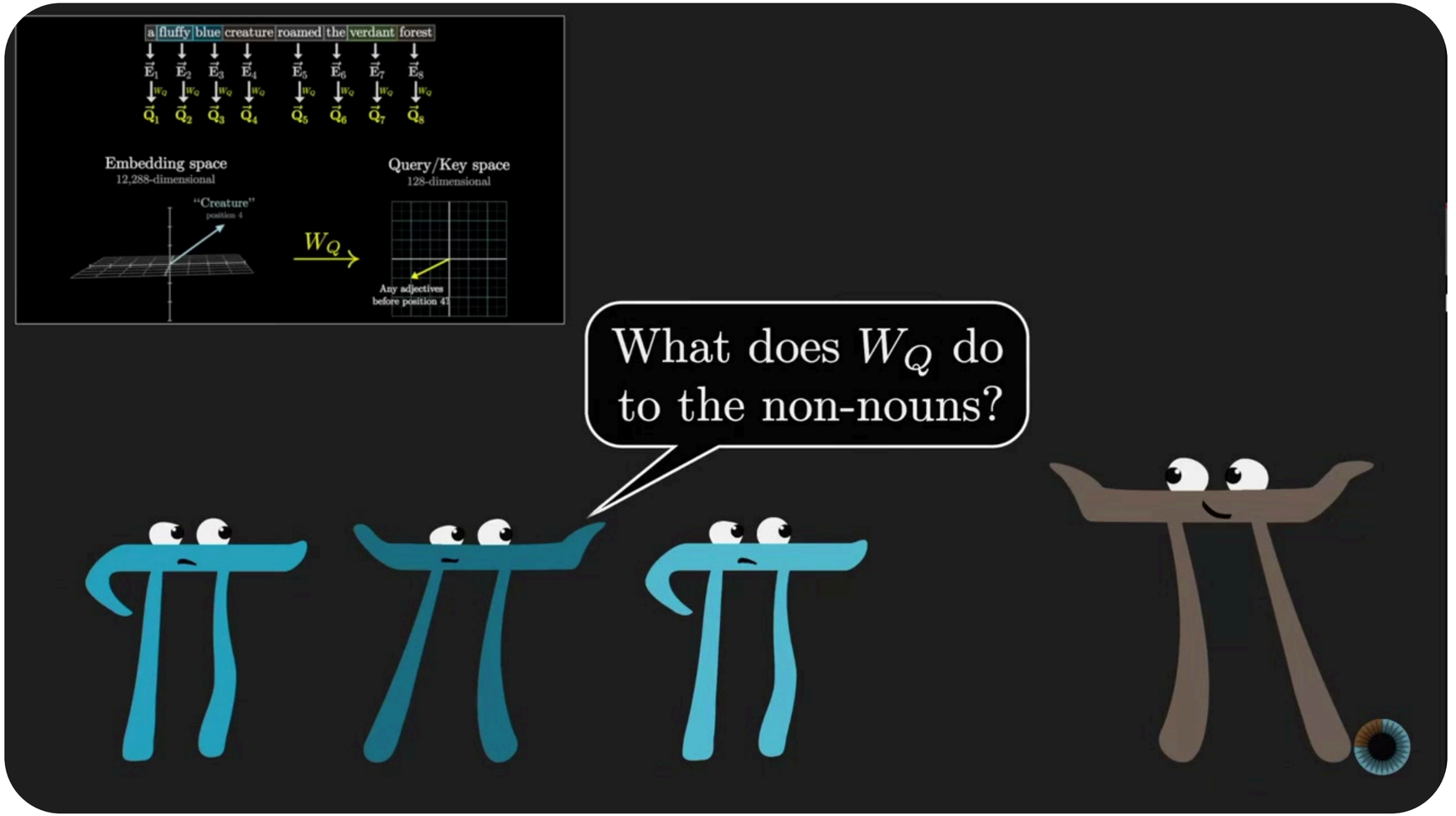
- “blue” × W_K = key expressing that “blue” is an adjective that can affect close nouns

$$\underset{d_Q}{\begin{bmatrix} 1.1 & \cdots & -2.3 \\ 0.4 & \cdots & 2.1 \\ \vdots & \cdots & \vdots \\ -2.1 & \cdots & -0.6 \\ 1.5 & \cdots & 3.1 \end{bmatrix}}_{W_K} \times \underbrace{\begin{bmatrix} 1.2 \\ -0.3 \\ 1.8 \\ \vdots \\ 0.6 \\ 0.1 \\ -0.7 \end{bmatrix}}_{E_i} = \underbrace{\begin{bmatrix} 0.1 \\ -1.3 \\ \vdots \\ 0.7 \\ -0.3 \end{bmatrix}}_{K_i}$$

d_E



Example: Key


a fluffy blue creature roamed the verdant forest
Embedding space
12,288-dimensional
"Creature"
position 4
W_Q
Query/Key space
128-dimensional
Any adjectives
before position 4
What does W_Q do
to the non-nouns?

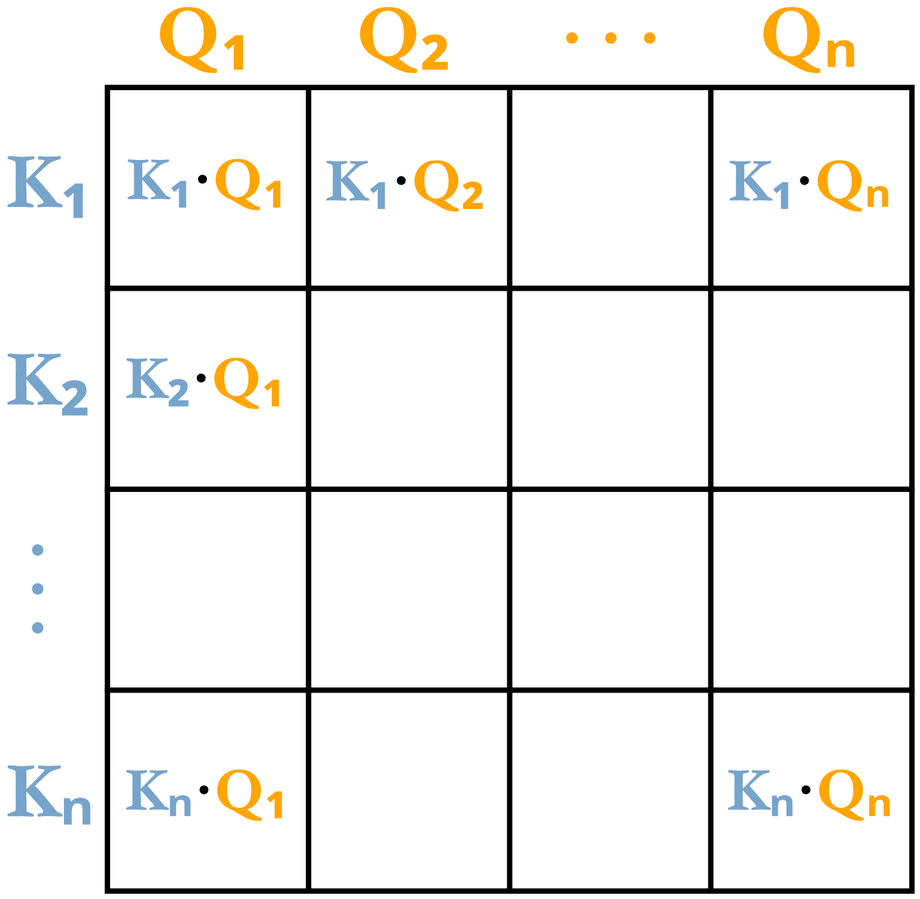
Attention Pattern

The attention pattern emerges when we compute how well each query aligns with each key. We use scalar products to measure this alignment

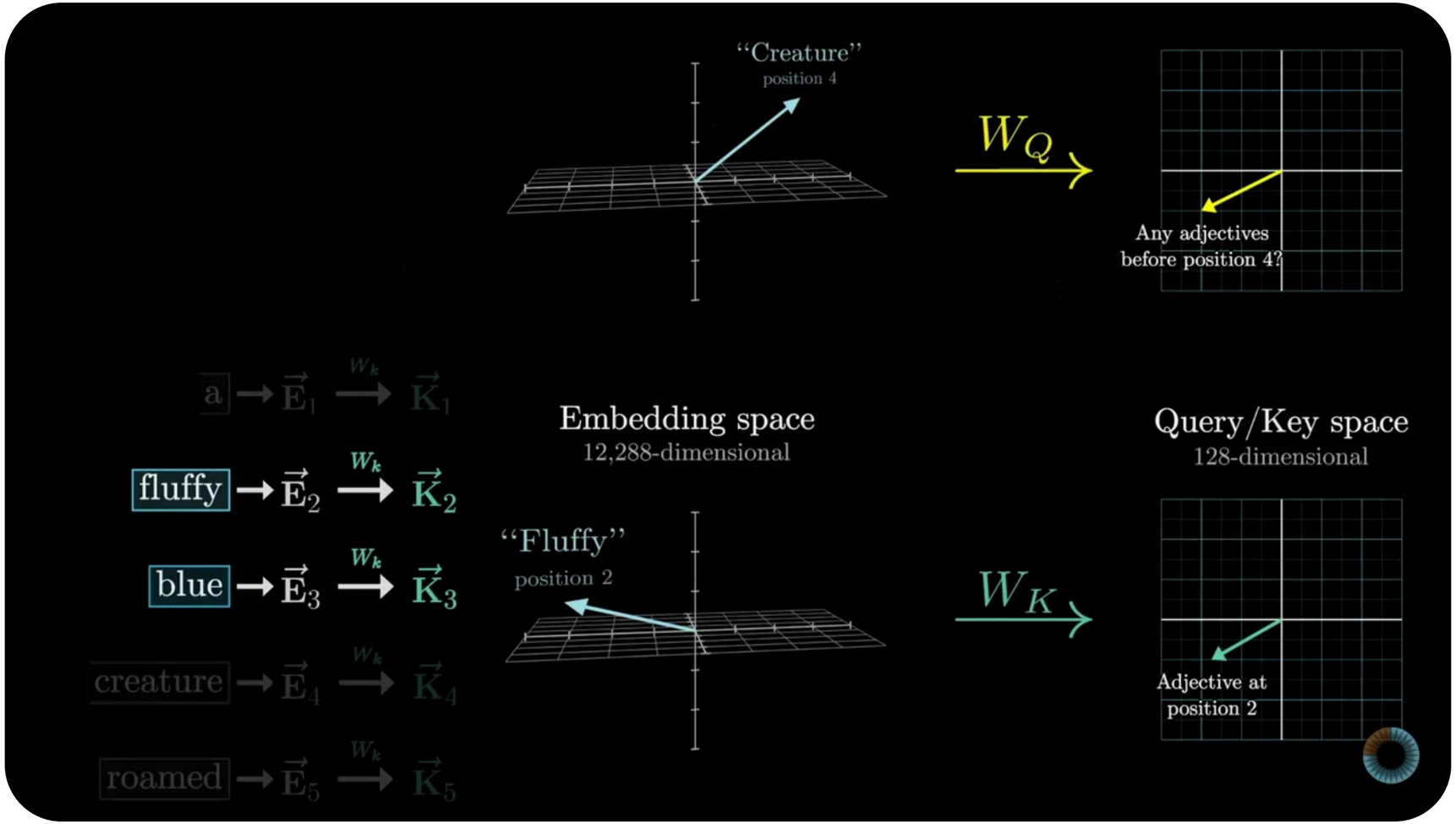
- For each query vector, we compute its scalar product with every key vector
- The results is the Attention Pattern
- It is a square matrix where each cell shows alignment between one query and one key

Reading the attention matrix:

- Rows represent “who is asking”
- Columns represent “who is being asked”
- Higher scores = stronger attention



Example: Attention Patterns



Softmax Normalization

Raw attention scores need to be converted into proper probabilities. Softmax ensures that each word's attention adds up to 100% - a complete probability distribution.

- **Why normalize:** Raw scores can be any size - we need probabilities between 0 and 1
- **Softmax function:** Converts any set of numbers into probabilities that sum to 1
- **Effect:**
  - High scores become high probabilities (close to 1)
  - Low scores become low probabilities (close to 0)
  - All probabilities for each word sum to exactly 1.0

Each word has 100% of its “attention budget” to distribute among all other words.

- Result: Clean probability matrix showing exactly how much attention each word gives to every other word.

Values

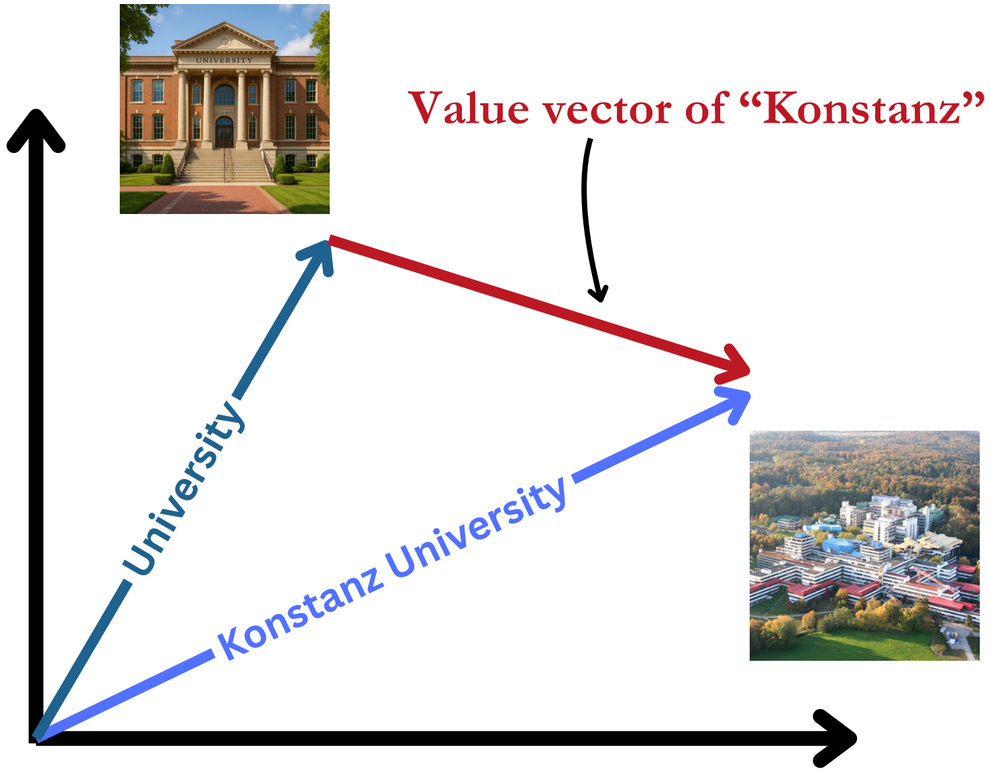
Values contain the actual information that words want to share with others.

- Keys advertise "what I have"
- Values contain the actual content to be shared
- They live in the same space as the original embedding vectors

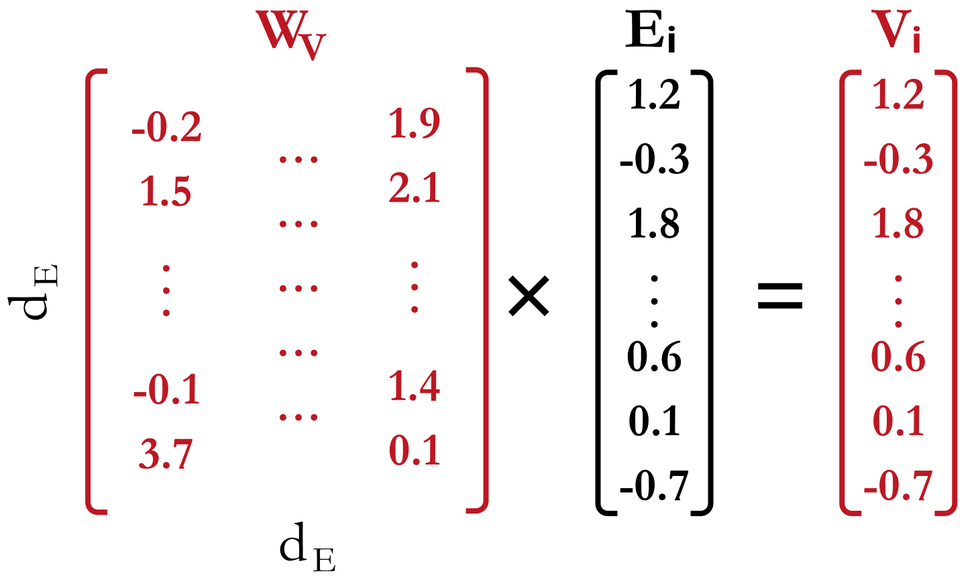
Example intuition:

- Adjective "blue" might share information about it being a specific color
- Noun "Konstanz" might share information connected to the location

"I work in **Konstanz University**"



The Value Matrix

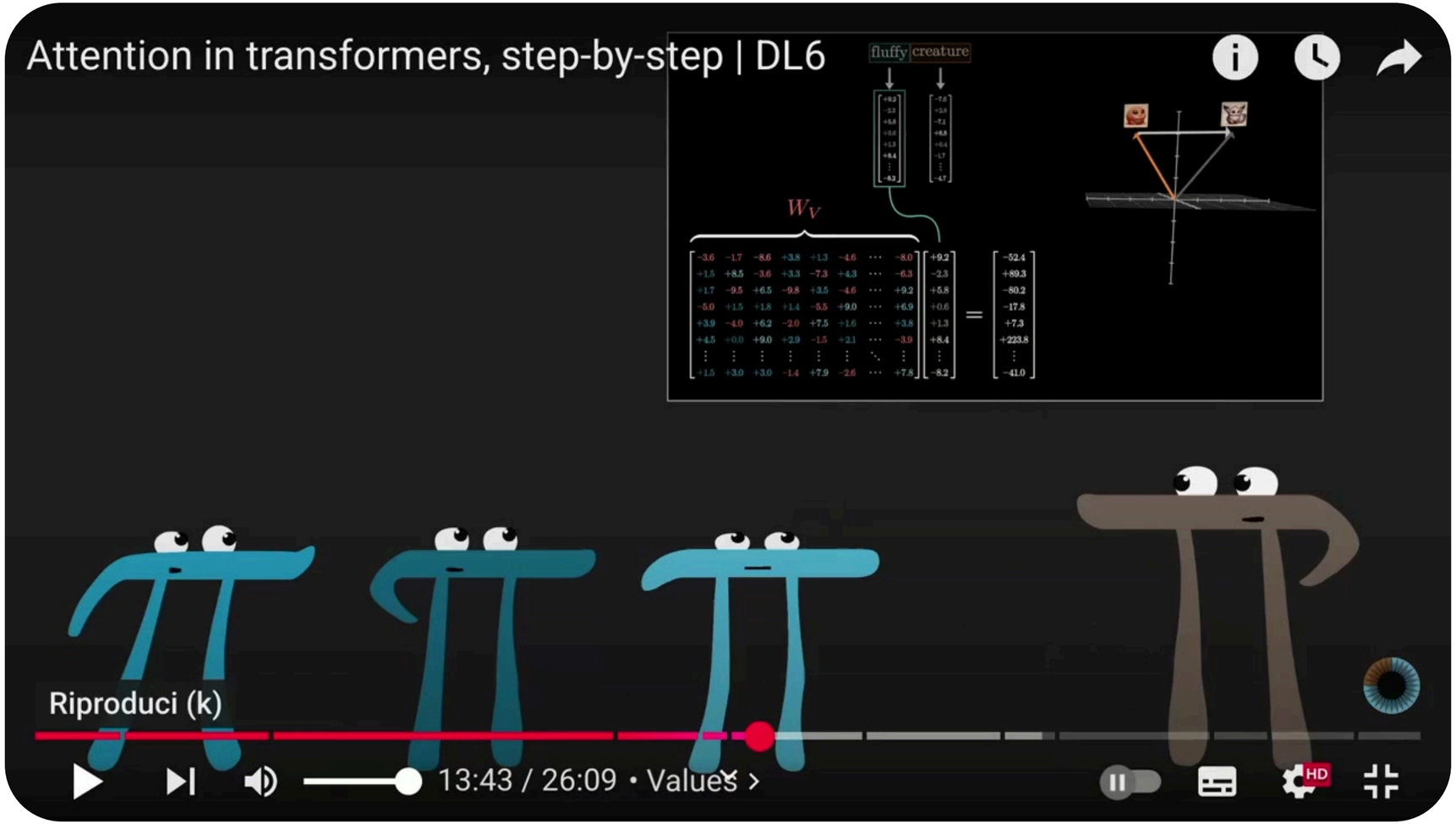


The Value Matrix (W_V) is the third crucial transformation that determines what information each word actually shares.

- It is a learned parameter matrix (W_V) with dimensions [embedding_dimension × embedding_dimension]
- Transforms embeddings into value vectors containing shareable information
- Mathematical operation:
  Value_Vector = Input_Embedding × W_V

The value matrix transforms the embedding vectors into the information they will share

Example: Value

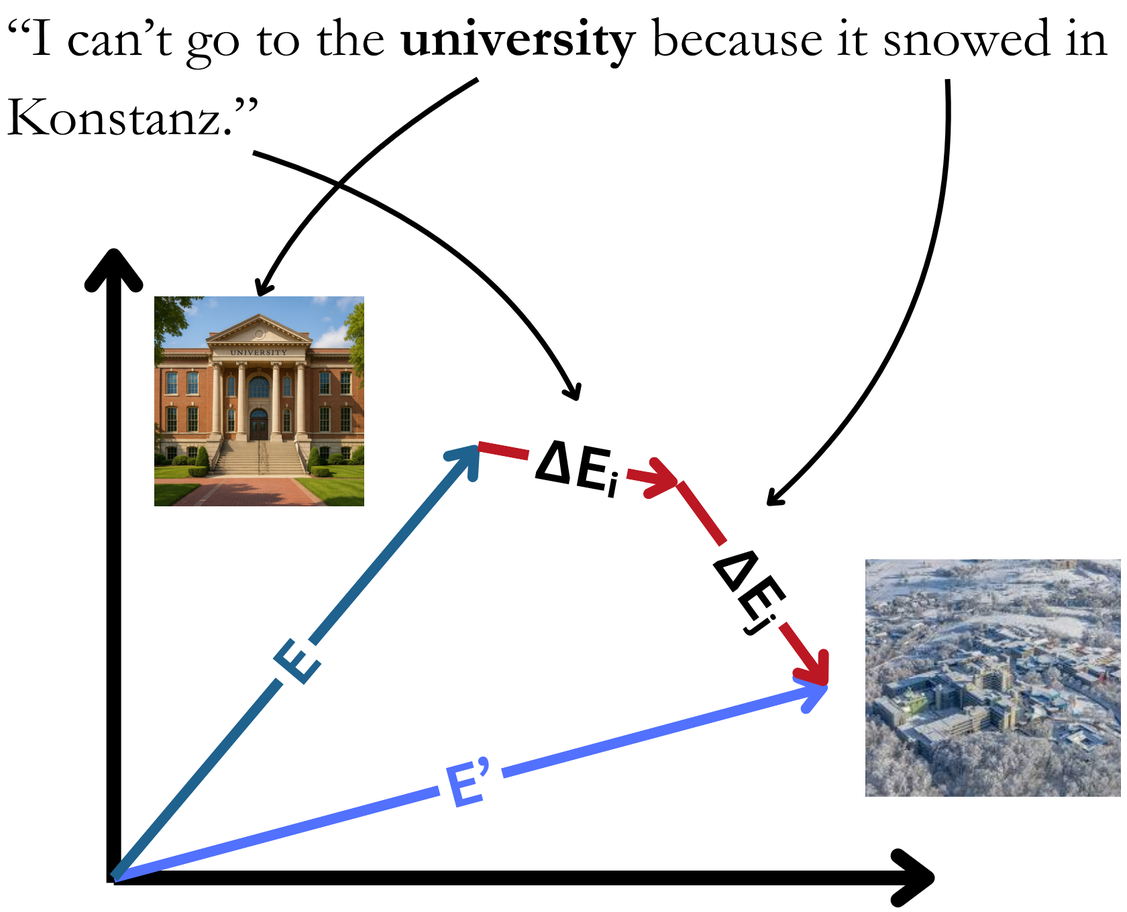


Computing Embeddings Variations

Finally we use attention weights to create new embeddings by mixing information from different words.

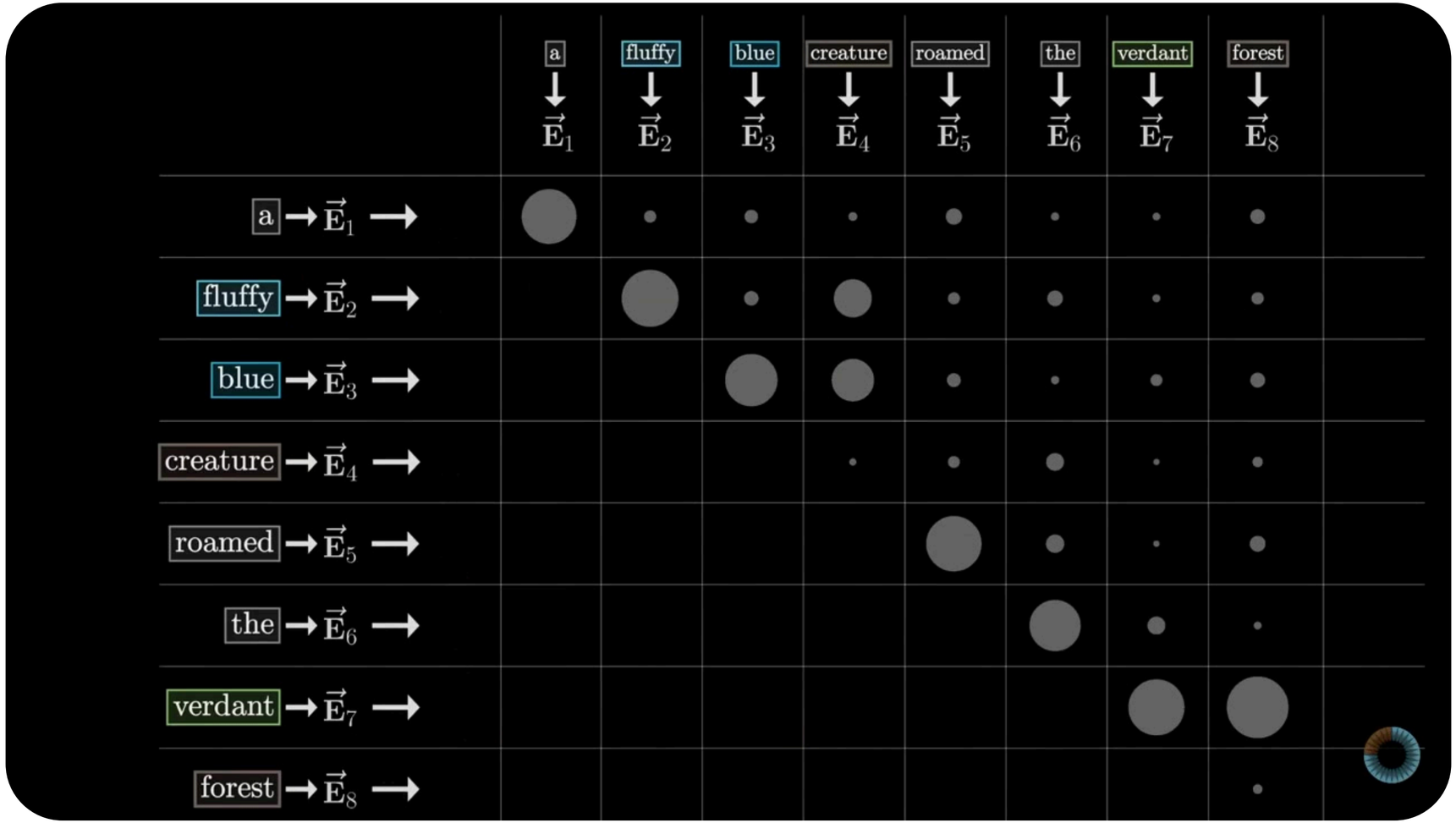
- Attention weights: How much each word should focus on every other word
- Value vectors: What information each word wants to share
- We compute the variation of each embedding as the sum of values weighted by attention

Each word gets a customized embedding based on its unique attention pattern.



Universität Konstanz

Example: Modified Embeddings



The Attention Equation

Everything we've learned can be summarized in one equation.

$$\text{Attention}(Q, V, K) = \text{softmax}\left(\frac{QK'}{\sqrt{d_k}}\right)V$$

This is the mathematical heart of the attention mechanism.

- **QK'**: Match queries with keys to get raw attention scores
- Softmax(...): Convert scores to probabilities that sum to 1
- ... × V: Use probabilities to combine value vectors

What each part does:

- Q (Query): What each word is looking for
- K (Key): What each word advertises
- V (Value): What each word actually provides

Universität
Konstanz

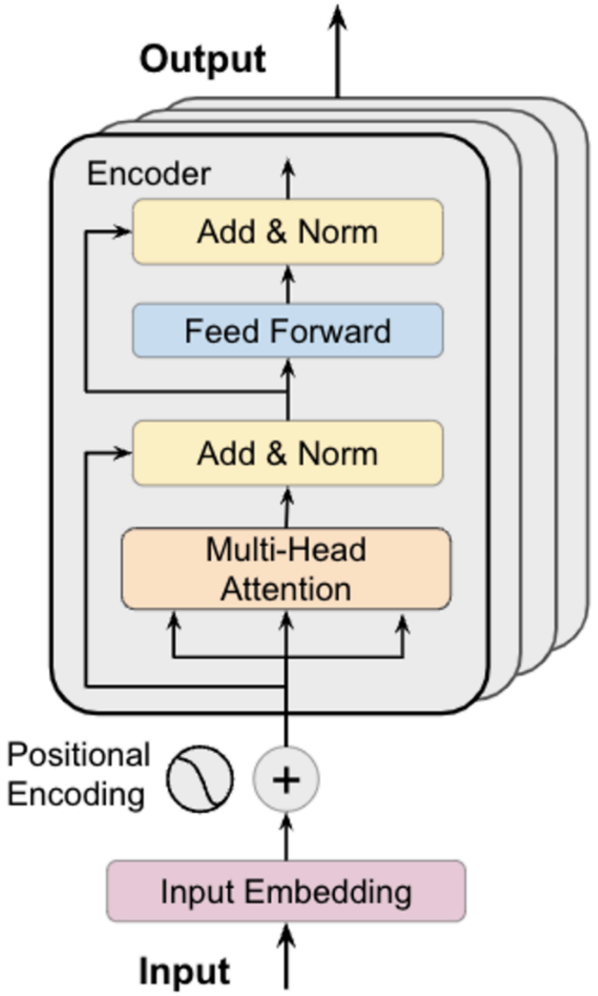
Transformer Architecture

The Transformer

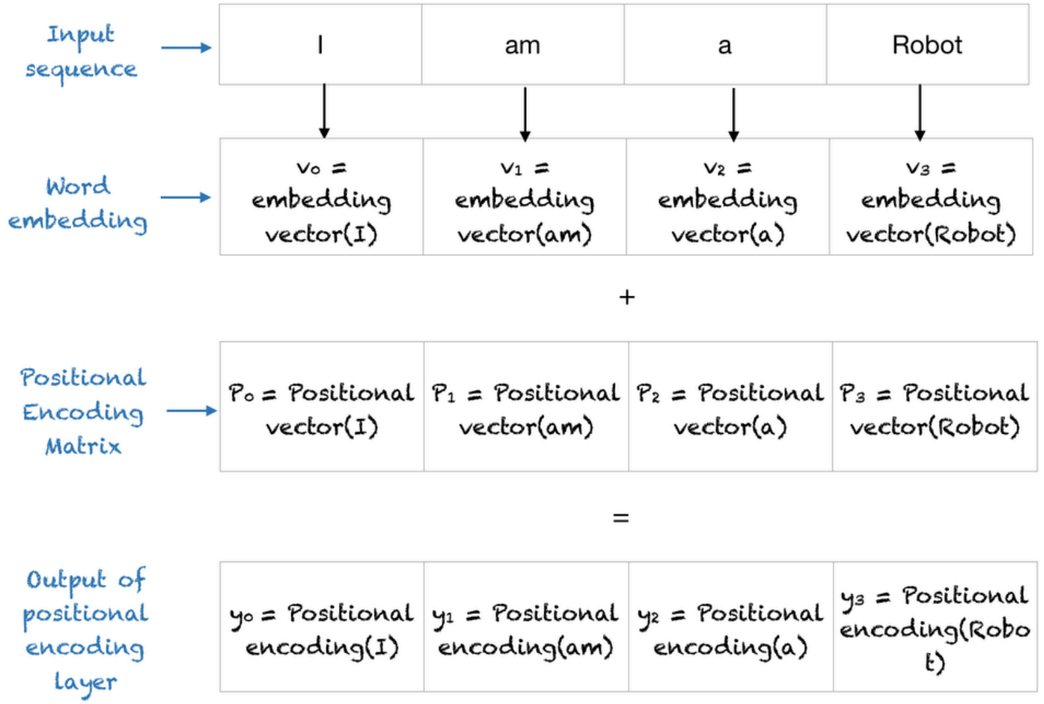
The Transformer is the architecture that revolutionized AI. It is composed of

- **Input embeddings:** Convert words to vectors
- **Positional encoding:** Add position information to embeddings
- **Multi-head attention:** The mechanism we just learned, with multiple heads
- **Feed-forward network (MLP):** Add external knowledge and processing power

The last two blocks are repeated multiple times allowing the model to develop a deep understanding of texts and relations between words



Positional Embedding



Attention has a major blindness: it doesn't know the order of words! “Cat ate fish” and “Fish ate cat” would look identical to pure attention.

- We need to add positional information.
- “The cat chased the mouse” ≠ “The mouse chased the cat”
- The solution: Add positional information to each word's embedding

How it works:

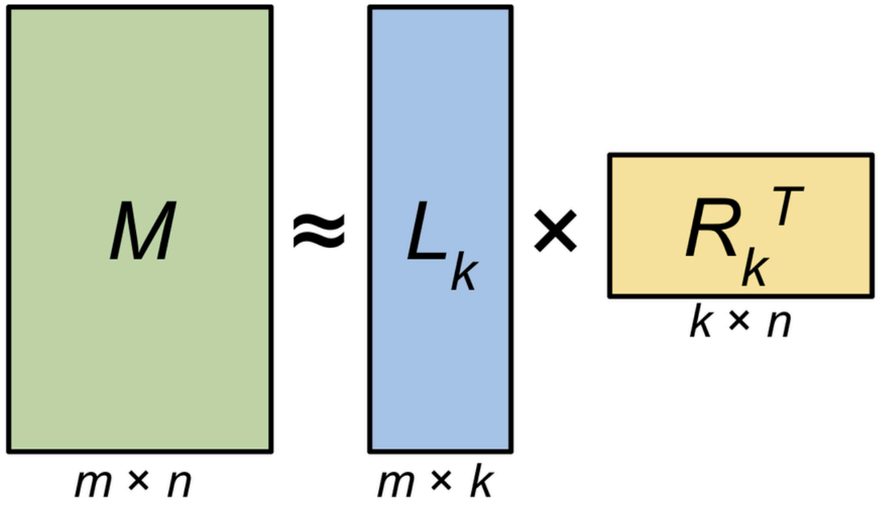
- Create a unique "position signature" for each position (1st word, 2nd word, etc.)
- Add this signature to the word embedding

Low-Rank Decomposition

Creating value vectors would require a huge matrix.

In practice, we use a trick to save on parameters

- The efficiency problem:
  - Large models need large transformation matrices
  - Example: 768×768 matrix = nearly 600,000 parameters
  - Multiply by many attention heads and layers = billions of parameters just for values
- The solution: Instead of one big transformation, use two smaller ones in sequence
  - In practice this is equivalent to write the value matrix as the product of two smaller matrices



Multi-Head Attention

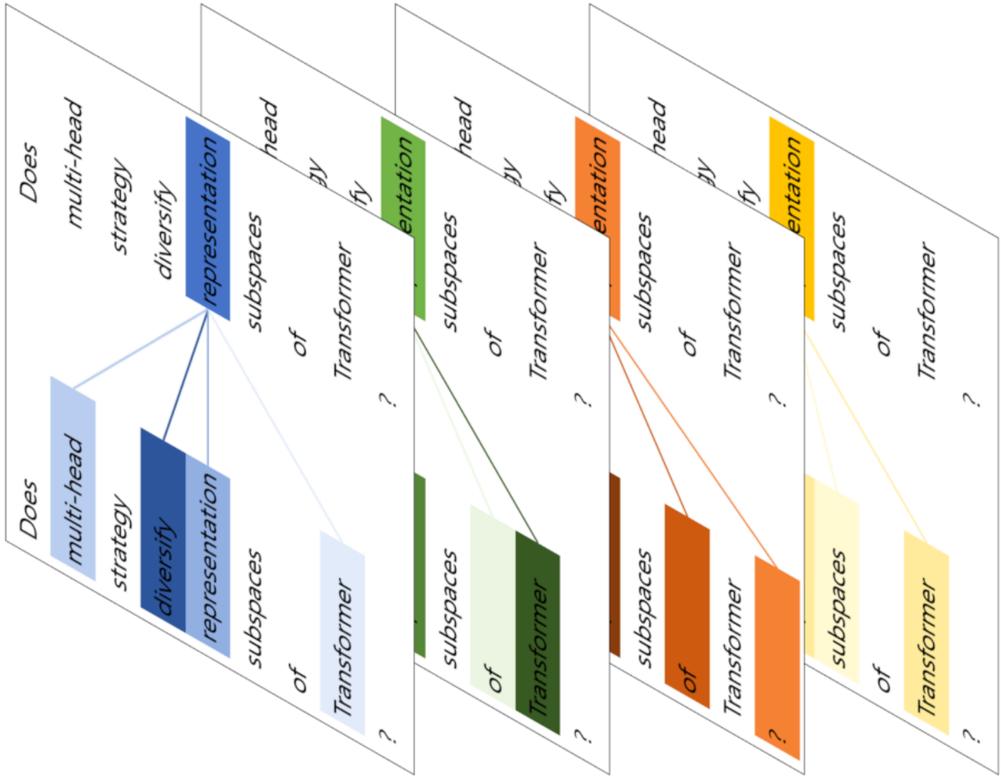
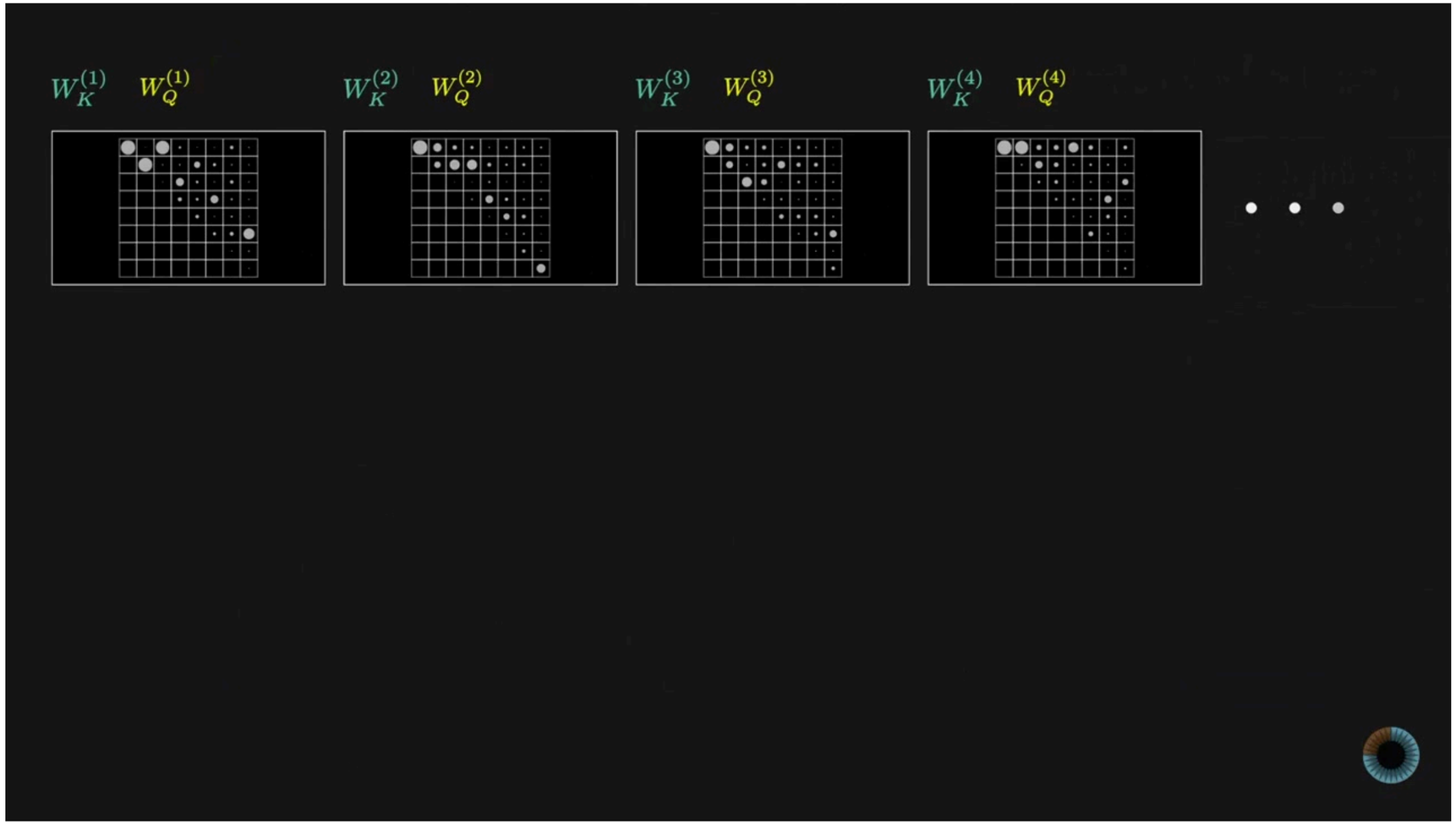


Image from: "*Analyzing and controlling inter-head diversity in multi-head attention.*" Applied Sciences 11.4 (2021): 1548.

Just like CNNs use multiple filters to detect different features, Transformers use multiple attention heads to capture different types of relationships.

- CNN filters detect edges, textures, shapes - each attention head detects different linguistic patterns
- Each head learns different relationships:
  - Head 1: Subject-verb relationships
  - Head 2: Adjective-noun relationships
  - Head 3: Long-distance dependencies

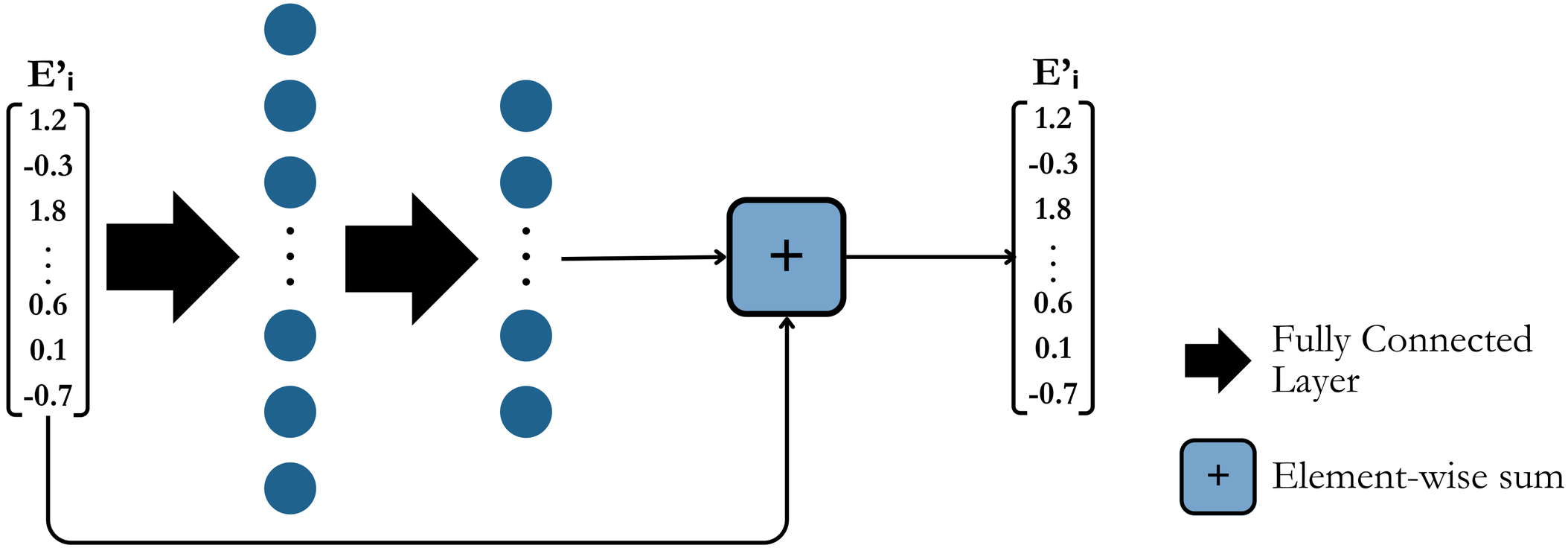
Example: Multi-Head Attention



Multilayer Perceptron

After attention mixes information between words, we process the embedding with a MLP

- Input: same dimension as embeddings
- Hidden layer: typical choice 4× expansion + ReLU (or similar)
- Output: back to original size



What is the MLP Doing?

While attention combines meaning between words in the current text, the MLP adds knowledge from outside the text - facts learned during training on massive datasets.

- Attention's job: “Michael” and “Jordan” appear together → they're related
- MLP's job: Add the fact that “Michael Jordan is a basketball player”

Examples of external knowledge:

- "Paris is the capital of France" (geographic facts)
- "Einstein developed relativity theory" (historical facts)
- "Cats are animals" (taxonomic relationships)
- "Winter is cold" (common sense knowledge)

Attention handles "what words relate to each other in this text" while MLP handles "what do I know about these concepts from my training."

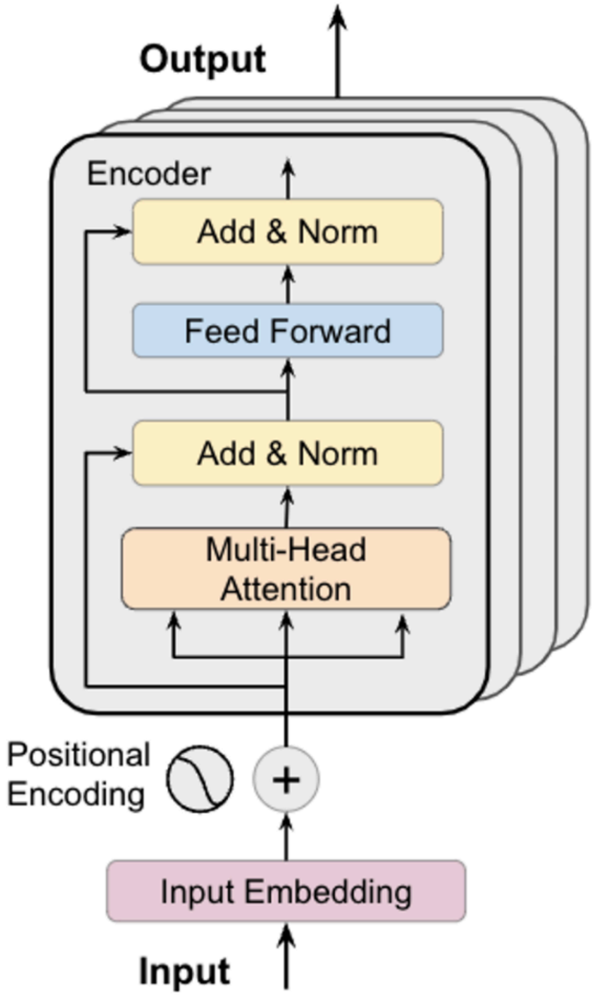
Repeat, Repeat, Repeat

The power of Transformers comes from stacking many attention and MLP layers. Each layer builds more sophisticated understanding on top of the previous layers.

- Typical architectures:
  - BERT-Large: 24 layers
  - GPT-3: 96 layers

What each layer adds:

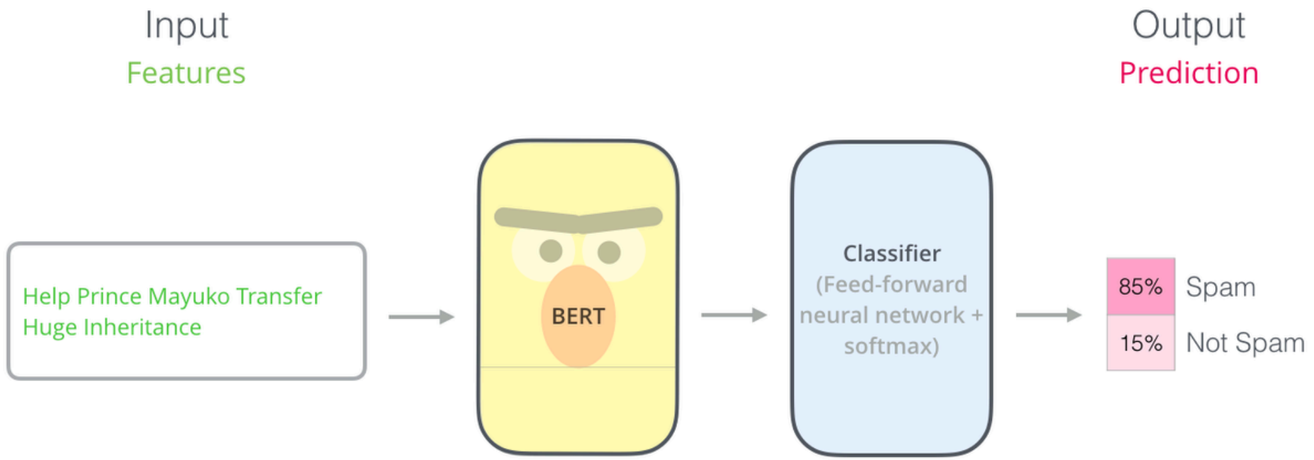
- Layer 1-2: Basic word relationships and syntax
- Layer 3-6: Phrase-level understanding and local context
- Layer 7-12: Sentence-level meaning and complex relationships
- Layer 13+: Abstract reasoning and complex inference



What Comes Out of the Transformer?

After all this processing through multiple layers, what do we actually get? The output is a set of context-aware embeddings - one for each input word, but now incredibly rich with information.

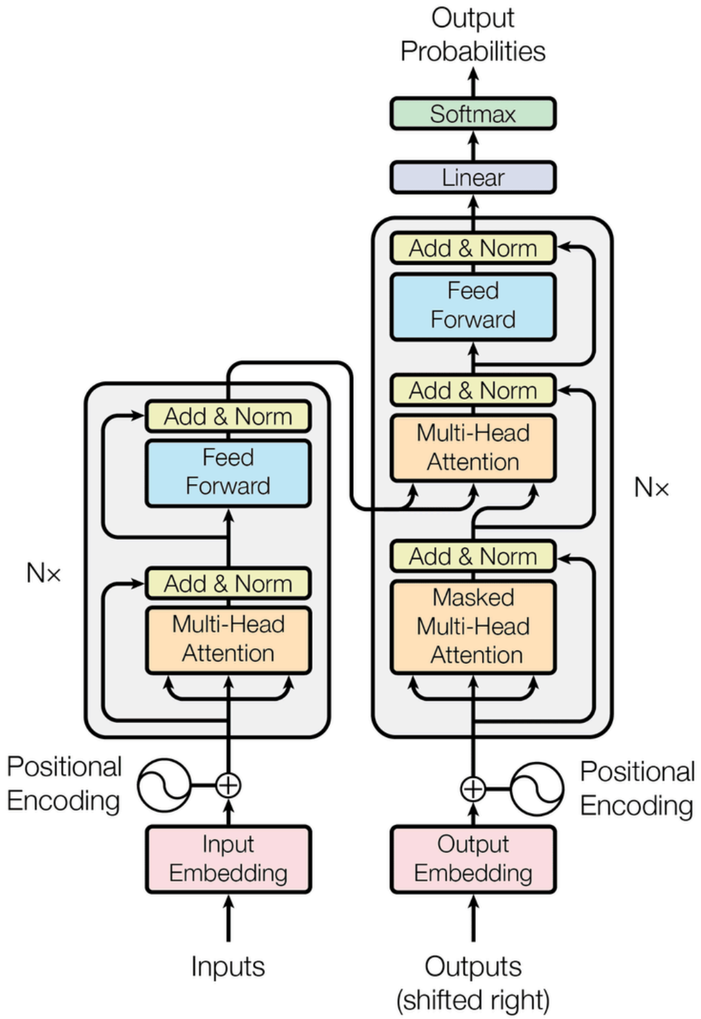
- Each word's final representation is a rich summary of both its original meaning and all its relationships in this specific context.
- These refined embeddings can be used for any language task - classification, question answering, or further text generation.



The “True” Transformer

What we've described is actually a simplified version focused on understanding text. The original Transformer paper described a more complex encoder-decoder architecture designed for translation tasks.

- What we learned: Encoder-only architecture (like BERT)
  - Takes input text and creates rich representations
  - Good for understanding tasks
- Original Transformer (2017): Encoder-Decoder
  - Designed for translation: English → French
  - Encoder: Processes input text (what we described)
  - Decoder: Generates output text word by word
  - Cross-attention: Decoder attends to encoder's representations



Summary

Natural Language Processing

Human language is complex and context-dependent, making it challenging for computers to understand. RNNs were our first successful approach but suffered from various limitations

Word Embeddings

Word embeddings like Word2Vec create dense vector representations where similar words cluster together. However, static embeddings give each word only one representation regardless of context.

Attention Mechanism

Attention creates dynamic word representations through queries, keys, and values, allowing words to focus on relevant information and influence each other's meanings.

Transformer Architecture

The Transformer combines multi-head attention with MLPs and stacks many layers to build sophisticated language understanding.

Next Lectures and Events

Tomorrow no coding session

This was a mostly theoretical lecture, so the holiday fits perfectly

Next Week

We will see how the attention mechanism and the transformer architecture are used to create Large Language Models. We will consider both embedding models and generative model and we will focus on how these models work and are trained.

Next Week CDM Colloquium (05/06 - Room D301 13:30-14:30)

“The Politics of Climate Change Mitigation: Evidence from the Ninth European Parliament”. Thomas Däubler (University College Dublin)