The linear algebra of large language models (LLM)

MATH 2740 – Mathematics of Data Science – Lecture 23

Julien Arino
julien.arino@umanitoba.ca

Department of Mathematics @ University of Manitoba

Fall 2026

The University of Manitoba campuses are located on original lands of Anishinaabeg, Ininew, Anisininew, Dakota and Dene peoples, and on the National Homeland of the Red River Métis. We respect the Treaties that were made on these territories, we acknowledge the harms and mistakes of the past, and we dedicate ourselves to move forward in partnership with Indigenous communities in a spirit of Reconciliation and collaboration.

Outline

The vector space model

Word embeddings and mathematical operations

The attention mechanism

The central idea: from words to vectors

- The fundamental challenge in Natural Language Processing (**NLP**) is to translate discrete, symbolic data (**words**) into a continuous, numerical format
- Why? Computers are built on arithmetic, not semantics. To process and generate language, we need a mathematical representation
- The **vector space model** is a key solution. It posits that we can represent words and documents as vectors in a multi-dimensional space, where the geometry of the space reflects semantic relationships
- Semantically similar words are mapped to nearby points in this space

The overall LLM process: a high-level overview

- Understanding how LLMs work involves two main phases:

  1. **Training phase**: The model **learns** the embedding vectors and the complex relationships between them from a vast amount of text data

  2. **Inference phase**: The trained model takes new text input, processes it, and generates an output, like predicting the next word or completing a sentence

- Both phases heavily rely on the linear algebra concepts we will explore

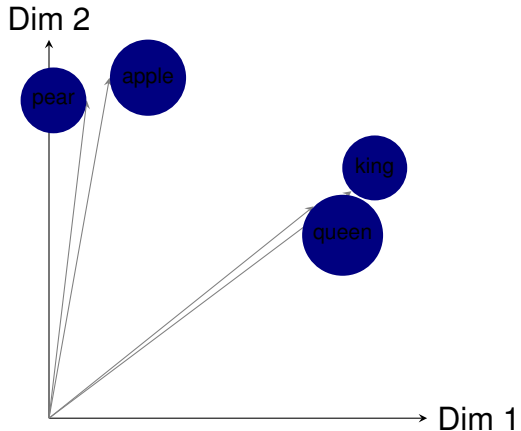
LLM process: the training phase (learning embeddings)

- **Input**: A massive corpus of text (e.g., books, articles, web pages)
- **Goal**: To fill the **embedding matrix** E with meaningful vectors, and to learn the parameters of the attention mechanisms and other layers
- **Mechanism**: The model is trained to perform tasks like predicting a masked word in a sentence or the next word in a sequence. Through this prediction task, it adjusts the values in the embedding matrix and other weight matrices
- **Mathematical Core**: This learning is an optimization problem solved using **gradient descent**. The model iteratively updates its parameters (the entries of matrices like E, W_Q, W_K, W_V) to minimize a **loss function** that measures prediction error

LLM process: the inference phase (generating text)

- **Input**: A new piece of text, often a prompt or query from a user
- **Process**:
  1. The input text is first **tokenized**, converting words into integer IDs
  2. These token IDs are then looked up in the **learned embedding matrix** to get their initial vectors
  3. These vectors are fed through the layers of the LLM (including attention mechanisms) to produce context-aware representations
  4. The model then uses these representations to predict the probability distribution over the next possible token in the vocabulary
  5. A token is selected from this distribution (e.g., the most probable one), added to the sequence, and the process repeats to generate further text
- **Output**: A coherent and contextually relevant text response

A visual metaphor for the vector space model



- Vectors for "king" and "queen" are close to each other
- Vectors for "apple" and "pear" are also close
- The distance between a fruit and royalty is large

What is a token?

- In LLMs, a **token** is the fundamental unit of text. It can be a word, a sub-word, a character, or a special symbol
- The process of converting text into a sequence of tokens is called **tokenization**
- For example, the sentence "The cat sat on the mat" might be tokenized as:

  ‘["the", "cat", "sat", "on", "the", "mat"]‘

- Each token is assigned a unique integer index based on a predefined vocabulary. For a sentence, we get a sequence of integers:

  ‘[1, 2, 3, 4, 1, 5]‘

- This sequence of integers is the raw input that we must translate into our vector space

From tokens to vectors: the embedding matrix

- Our vocabulary of size V is a list of all unique tokens, e.g., "cat" $\rightarrow$ 1234, "dog" $\rightarrow$ 5678
- The central component of a large language model is the **embedding matrix**, denoted $E \in \mathbb{R}^{V \times d}$
- Each row of this matrix, $E_{i,:} \in \mathbb{R}^{d}$, is the **embedding vector** for the i-th token

Defining a vector space

- Recall that a **vector space** V over a field F (e.g., $\mathbb{R}$) is a set of vectors with two operations: vector addition and scalar multiplication
- In LLMs, our vectors are typically from $\mathbb{R}^d$, where d is the **embedding dimension**. This is the size of the vector used to represent each token. A larger d allows for more complex relationships to be encoded
- A vector $v \in \mathbb{R}^d$ is an ordered d-tuple: $v = [v_1, v_2, ..., v_d]^T$
- We use linear algebra to define key concepts like **distance** and **similarity** within this space. For a given word, its position in this space is determined by its relationship to all other words

How are embeddings learned?

- Unlike our simple 2D example, the embedding vectors in a real LLM are not assigned by hand
- These vectors are **learned** from vast amounts of text data during the model's training process
- The goal of the training is for the model to learn a vector space where words with similar meanings have similar vector representations. This is done by having the model predict masked words in a sentence or the next word in a sequence
- The numerical values in the embedding matrix E are initially random and are then optimized over many iterations using an algorithm called **gradient descent** to minimize a loss function
- The final, learned vectors encode a word's meaning, usage, and its relationship to other words based on the context it appeared in during training

The embedding lookup: a linear transformation

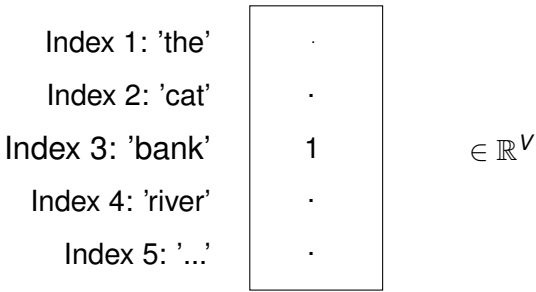
- The embedding process can be viewed as a linear transformation from a standard basis vector to the dense embedding space

- The embedding for token i is found by matrix-vector multiplication:

$$\boldsymbol{v}_i = E^T \boldsymbol{e}_i$$

  where $\boldsymbol{e}_i$ is a standard basis vector (a **one-hot vector** in LLM parlance)

- This is a computationally efficient lookup: for a vocabulary of size $V = 50000$ and a dimension $d = 768$, the matrix E would be of size 50000×768


Index 1: ’the’
Index 2: ’cat’
Index 3: ’bank’
Index 4: ’river’
Index 5: ’...’
1
$\in \mathbb{R}^V$

Measuring similarity: the dot product and cosine similarity

- How do we quantify the "closeness" of two word vectors, $\boldsymbol{u}$ and $\boldsymbol{v}$
- We use the **dot product** (or inner product):

$$\boldsymbol{u} \cdot \boldsymbol{v} = \sum_{i=1}^{d} u_i v_i = ||\boldsymbol{u}|| \cdot ||\boldsymbol{v}|| \cos(\theta)$$

- The dot product is maximized when vectors point in the same direction
- A more robust measure is **cosine similarity**, which normalizes for vector magnitude and is invariant to vector length:

$$\text{similarity}(\boldsymbol{u}, \boldsymbol{v}) = \frac{\boldsymbol{u} \cdot \boldsymbol{v}}{||\boldsymbol{u}|| \cdot ||\boldsymbol{v}||} = \cos(\theta)$$

- Cosine similarity is a scalar value in the range $[-1, 1]$. A value of 1 means identical direction (most similar), 0 means orthogonal (no relationship), and -1 means opposite direction

Example: word embeddings in 2D

- Let's consider a toy example in $\mathbb{R}^2$ with a vocabulary of three words: "cat", "dog", and "apple"
- Assume embedding vectors are

$$\boldsymbol{v}_{\text{cat}} = \begin{bmatrix} 0.8 \\ 0.6 \end{bmatrix}, \quad \boldsymbol{v}_{\text{dog}} = \begin{bmatrix} 0.7 \\ 0.7 \end{bmatrix}, \quad \boldsymbol{v}_{\text{apple}} = \begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix}$$

- First, compute their norms:

$$||\boldsymbol{v}_{\text{cat}}|| = \sqrt{0.8^2 + 0.6^2} = 1$$

$$||\boldsymbol{v}_{\text{dog}}|| = \sqrt{0.7^2 + 0.7^2} \approx 0.99$$

$$||\boldsymbol{v}_{\text{apple}}|| = \sqrt{(-0.5)^2 + 0.5^2} \approx 0.71$$

Example: word embeddings in 2D (continued)

- Now, the dot products:

$$\boldsymbol{v}_{\text{cat}} \cdot \boldsymbol{v}_{\text{dog}} = (0.8)(0.7) + (0.6)(0.7) = 0.56 + 0.42 = 0.98$$

$$\boldsymbol{v}_{\text{cat}} \cdot \boldsymbol{v}_{\text{apple}} = (0.8)(-0.5) + (0.6)(0.5) = -0.4 + 0.3 = -0.1$$

- And finally, the similarities:

$$\text{sim}(\text{cat}, \text{dog}) = \frac{0.98}{1 \cdot \sqrt{0.98}} \approx 0.99 \quad \text{(very similar)}$$

$$\text{sim}(\text{cat}, \text{apple}) = \frac{-0.1}{1 \cdot \sqrt{0.5}} \approx -0.14 \quad \text{(not similar)}$$

Vector arithmetic: encoding analogies

- The true power of these embeddings is their capacity for vector arithmetic, which reveals latent linear relationships
- The famous example:

$$\vec{\textbf{king}} - \vec{\textbf{man}} + \vec{\textbf{woman}} \approx \vec{\textbf{queen}}$$

- This demonstrates that the vector difference $\vec{\textbf{king}} - \vec{\textbf{man}}$ represents the "gender" or "royalty" dimension. Adding it to $\vec{\textbf{woman}}$ yields the vector for "queen"
- This property is a direct consequence of the linear algebraic structure learned during training. It shows that the model has learned to identify and represent meaningful axes in the vector space

The problem of context: a mathematical view

- A simple word embedding is context-independent. The vector for "bank" is always the same, whether in "river bank" or "financial bank."
- LLMs need a mechanism to produce **context-dependent** representations
- The **attention mechanism** computes a new representation for each token by creating a **weighted average** of all other token vectors in the sequence
- The weights are determined by the relevance of each token to the current token

Attention in three matrices: Q, K, and V

- For a sequence of token embeddings, we stack them into a matrix $X \in \mathbb{R}^{n \times d}$, where n is the sequence length
- For each token's input embedding $\vec{x}_i \in \mathbb{R}^d$, we project it into three different vector spaces using three trainable weight matrices:
  - **Query matrix** $W_Q \in \mathbb{R}^{d \times d_k}$
  - **Key matrix** $W_K \in \mathbb{R}^{d \times d_k}$
  - **Value matrix** $W_V \in \mathbb{R}^{d \times d_v}$
- The matrices Q, K, and V are then calculated via matrix multiplication:

$$Q = XW_Q \quad K = XW_K \quad V = XW_V$$

- $Q, K \in \mathbb{R}^{n \times d_k}$ and $V \in \mathbb{R}^{n \times d_v}$. This is a simple linear transformation for all tokens in the sequence simultaneously

The scaled dot-product attention equation

- The core of attention is computing the attention scores and applying them to the Value vectors
- The score matrix is a result of a massive dot product operation between every Query vector and every Key vector
- This is efficiently computed as a single matrix multiplication: QK^T
- The full equation for the output matrix $Z \in \mathbb{R}^{n \times d_v}$ is:

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

Dissecting the equation: the attention scores

- Let's consider a simple example with a sequence of two words, "river bank". Let's use 2D vectors for simplicity
- Let the input embeddings be: $\vec{x}_{\text{river}} = \begin{bmatrix} 0.5 \\ 0.2 \end{bmatrix}$ and $\vec{x}_{\text{bank}} = \begin{bmatrix} 0.8 \\ 0.6 \end{bmatrix}$. So

$$X = \begin{bmatrix} 0.5 & 0.2 \\ 0.8 & 0.6 \end{bmatrix}$$

- Assume the weight matrices are:

$$W_Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad W_K = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad W_V = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

- Then $Q = X$, $K = X$, $V = X$. We compute QK^T:

$$QK^T = \begin{bmatrix} 0.5 & 0.2 \\ 0.8 & 0.6 \end{bmatrix} \begin{bmatrix} 0.5 & 0.8 \\ 0.2 & 0.6 \end{bmatrix} = \begin{bmatrix} (0.5)(0.5) + (0.2)(0.2) & (0.5)(0.8) + (0.2)(0.6) \\ (0.8)(0.5) + (0.6)(0.2) & (0.8)(0.8) + (0.6)(0.6) \end{bmatrix}$$

$$= \begin{bmatrix} 0.29 & 0.52 \\ 0.52 & 1 \end{bmatrix}$$

- The element $(1, 2)$ is the dot product of the query for "river" and the key for

Dissecting the equation: scaling and softmax

- The next step is scaling and applying the softmax function to the score matrix
- The scale factor $\sqrt{d_k} = \sqrt{2} \approx 1.414$.

$$\frac{QK^T}{\sqrt{d_k}} = \frac{1}{\sqrt{2}} \begin{bmatrix} 0.29 & 0.52 \\ 0.52 & 1 \end{bmatrix} \approx \begin{bmatrix} 0.205 & 0.368 \\ 0.368 & 0.707 \end{bmatrix}$$

- The softmax is applied row-wise. For the first row (the weights for "river"):

$$\text{softmax}([0.205, 0.368]) = \left[\frac{e^{0.205}}{e^{0.205} + e^{0.368}}, \frac{e^{0.368}}{e^{0.205} + e^{0.368}} \right] \approx [0.46, 0.54]$$

- The resulting weight matrix A is:

$$A = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} \right) \approx \begin{bmatrix} 0.46 & 0.54 \\ 0.46 & 0.54 \end{bmatrix}$$

Dissecting the equation: final output

- The final step is to multiply the weight matrix A by the Value matrix V
- The output is the new context-aware representation $Z = AV$:

$$Z = \begin{bmatrix} 0.46 & 0.54 \\ 0.46 & 0.54 \end{bmatrix} \begin{bmatrix} 0.5 & 0.2 \\ 0.8 & 0.6 \end{bmatrix}$$

- The first row of Z is the new vector for "river":

$$\vec{z}_{\text{river}} = (0.46)\boldsymbol{v}_{\text{river}} + (0.54)\boldsymbol{v}_{\text{bank}} = 0.46 \begin{bmatrix} 0.5 \\ 0.2 \end{bmatrix} + 0.54 \begin{bmatrix} 0.8 \\ 0.6 \end{bmatrix} = \begin{bmatrix} 0.23 \\ 0.092 \end{bmatrix} + \begin{bmatrix} 0.432 \\ 0.324 \end{bmatrix} = \begin{bmatrix} 0.6 \\ 0.4 \end{bmatrix}$$

- The new vector for "river" is a **linear combination** of the two original value vectors. Its position in the space is now influenced by the vector for "bank", thereby incorporating context

The importance of the Transformer architecture

- The attention mechanism is a key component of the **Transformer architecture**
- This architecture allows for parallel processing of all tokens in a sequence, a major improvement over older models like Recurrent Neural Networks (**RNNs**) that process tokens one-by-one
- The entire Transformer block is a sequence of linear algebraic operations:
  1. Linear projection of input embeddings ($X \to Q, K, V$)
  2. Matrix multiplication for attention scores (QK^T)
  3. Scaling, softmax, and a final matrix multiplication ($Z = AV$)
  4. Further linear transformations (feed-forward layers) and residual connections (vector addition)
- The power of LLMs lies in the scale and repeated application of these fundamental linear algebra concepts