# How Linear Algebra Powers Artificial Intelligence

Michael Alexander Angkawijaya 13523102[1,2]
*Program Studi Teknik Informatika*
*Sekolah Teknik Elektro dan Informatika*
*Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia*
[1]13523102@std.stei.itb.ac.id, [2]angkawijayamichael@gmail.com

***Abstract*—Linear algebra is a fundamental component of artificial intelligence (AI), enabling the mathematical representation and transformation of data, optimization of neural networks, and implementation of applications across diverse domains. This paper explores how linear algebra underpins AI workflows, beginning with data representations like vectors, matrices, and tensors, and extending to neural network training through forward and backward propagation. Additionally, we examine applications in natural language processing, computer vision, and recommendation systems, highlighting the indispensable role of linear algebra in extracting patterns and reducing dimensionality. Finally, we discuss the computational challenges and future prospects of integrating linear algebra with emerging AI technologies, underscoring its pivotal role in the advancement of intelligent systems.**

***Keywords*—Linear Algebra, Artificial Intelligence, Optimization, Applications in AI**

## I. Introduction

Artificial Intelligence has come a long way since its inception in the middle of the 20th century. The first AI systems were based on symbolic reasoning, using logical systems and rule-based approaches to mimic human thought. While these methods were groundbreaking in their time, they suffered from limitations due to their reliance on structured data and inability to handle complex or unstructured information, such as images and natural language. In its evolution, AI made a strong turn from symbolic reasoning toward data-driven methodologies, including machine learning and deep learning. Central to this transformation has been an increasing reliance on linear algebra, now the mathematical backbone of modern AI.

Linear algebra provides the tools to represent, process, and analyze data in ways that were previously unattainable. In the early days of machine learning, only a few algorithms—basically relying on simple vector and matrix operations for recognizing patterns in datasets, including linear regression and support vector machines—were in vogue. However, with the rise of neural networks and deep learning, the importance of linear algebra has grown tremendously; it is heavily used to enable the large computations that are needed to train complex models. Current artificial intelligence frameworks, encompassing convolutional neural networks (CNNs) utilized for image recognition and transformers employed in natural language processing, fundamentally rely on sophisticated linear algebra methodologies to interpret extensive datasets and reveal complex relationships among them.

The foundational artificial intelligence and its contemporary equivalents diverge significantly in both the scope and sophistication of their application of linear algebra. Previous systems used linear algebra in a very limited way, constrained by both computational power and the type of problems they were designed to solve. However, powerful hardware makes linear algebra operations—like matrix multiplication, computation of eigenvectors, or Singular Value Decomposition (SVD)—possible on an enormously large scale. Such technological advances have enabled artificial intelligence to make sense of unstructured data, optimize the extraction of features, and represent complicated relationships in high-dimensional spaces.

This paper discusses the mechanisms by which linear algebra boosts artificial intelligence, tracing its impact from the very first steps of the development of machine-learning techniques up to the contemporary epoch of deep learning. Showing the critical mathematical underpinnings of huge leaps in AI, this discussion indicates the immense importance of linear algebra in framing the operations and applications of contemporary AI technologies.

## II. Data Representation and Transformations

### A. Data Representation

Data features are represented as vectors. Each data point is transformed into a vector in a high-dimensional space, and each dimension of the vector corresponds to a specific attribute or feature of the data. For instance, in image recognition, an n x n pixels of grayscale image is represented as ($p_{1,1}$, $p_{1,2}$, $p_{1,3}$, ..., $p_{n,n-2}$, $p_{n,n-1}$, $p_{n,n}$), where $p_{i,j}$ denotes the value of the pixel at row $i$, column $j$.

Feature vectors allow data to be processed mathematically and enable algorithms to analyze patterns and relationships between data points.

Datasets are represented as matrices. Each row corresponds to a data point, and each column corresponds to the vector feature. For example, a dataset with m samples and n features can be written as:

$$A = [a_{ij}] = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

Fig. 2.1 Dataset representation as matrix. (Source: [1])

In higher dimensions, data are represented as tensors. A common example is video data, represented as a 5D tensor with dimensions (batch size, frames, height, width, color channels).

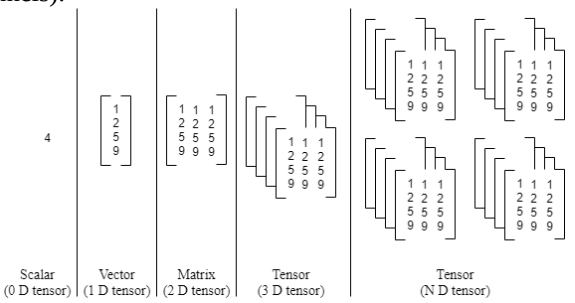


Fig. 2.2 Different types of tensors. (Source: https://www.analyticsvidhya.com/blog/2022/07/data-representation-in-neural-networks-tensor/)

In Natural Language Processing (NLP), words are also represented as vectors. One traditional way to represent words is by one-hot encoding. In this approach, the features correspond to all the words in the vocabulary, and each word is represented as a vector of zeros, except for a single '1' that marks the presence of the word, ensuring that each word has a unique representation. Another similar approach is with integer-encoding, encode each word using a unique number.

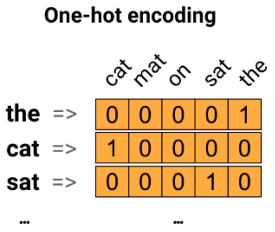


Fig. 2.4 One-hot encoding. (Source: https://www.tensorflow.org/text/guide/word_embeddings)

The problem with this approach is the dimensions of the vector. When dealing with large datasets of vocabulary, it creates vectors with as many dimensions as there are unique words in the vocabulary, resulting in extremely high-dimensional, sparse vectors that are computationally inefficient to store and process. Moreover, it doesn't capture the relationship between words, it lacks the context. Word embeddings will be discussed later below.

### *B. Linear Transformation*

To prepare the data for analysis, it undergoes linear transformation, such as scaling, rotation, shearing, and projection to manipulate and simplify its structure. Linear transformations are mathematically represented as matrix multiplication, where a transformation matrix A is applied to the input data x, transforming its vector space from one to another.

$$T(x) = Ax \qquad (1)$$

### *C. Dimensionality Reduction*

Beyond basic transformations, advanced techniques like Singular Value Decomposition (SVD) and Principal Component Analysis (PCA) leverage linear transformations to extract features and reduce dimensionality. SVD factorizes an m x n matrix M into matrices U, ∑, and T such that:

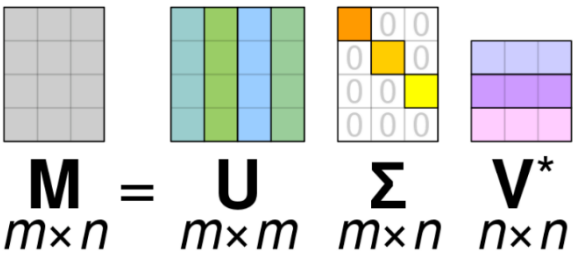


Fig. 2.3 Singular Value Decomposition. (Source:[2])

where:

**U**: An orthogonal matrix whose columns are the left singular vectors of M.

**Σ**: A diagonal matrix containing the singular values of M.

V: An orthogonal matrix whose rows are the right singular vectors of M.

By truncating the matrix **Σ**, taking only r highest singular value, SVD reduces the dimension of matrix M, allowing for more efficient storage and computation.

## III. Optimization and Model Training

Optimization and model training is a very crucial part in building powerful AI. The aim of the optimization and model training is to minimize loss, i.e., the error between the predictions and the real values. This section will be more focused on optimization and model training for neural networks.

### *A. Neural Network*

Neural networks consist of layers of interconnected nodes or neurons that process data to produce predictions. Neural networks typically consist of input layer, hidden layers, and output layer. Each connection between neurons is associated with a weight, which determines the strength of the connection, and a bias, which adjusts the output of the neuron. Each neuron calculates its output based on the weighted input it receives, a bias term, and a nonlinear activation function:

$$y = f(w_1x_1 + w_2x_2 + \cdots + w_nx_n + b) \qquad (2)$$

with:

$\boldsymbol{x_1, x_2, \dots, x_n}$: input features,

$\boldsymbol{w_1, w_2, \dots, w_n}$: weights for each input feature.

**b**: bias.

**f**: activation function (e.g., ReLU, sigmoid)

### B. Loss Function

As mentioned earlier, the aim of the optimization and model training is to minimize the difference between the network's predictions and the actual target values. This discrepancy is measured using a loss function. For example, in regression tasks, a common choice is the mean squared error (MSE):

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \quad (3)$$

where:
$\hat{\boldsymbol{y}}_{\boldsymbol{i}}$: Predicted value for the *i*-th sample,
$\boldsymbol{y}_{\boldsymbol{i}}$: Actual value for the *i*-th sample
**N**: number of samples

### C. Forward Propagation

Forward propagation is the process of passing input data through the network to calculate predictions and, subsequently, the loss. The value of a neuron is calculated with equation (2). At first iteration, the weight for each features is assigned randomly. The output of one layer would then become the input to the next layer, and so on, until the output layer. The loss then calculated using the loss function, summarizing how well the network is performing in its current state. The lower the value, the better the performance of the network.

### D. Backward Propagation

To improve the model, we need to calculate how each parameter (weights and biases) affects the overall loss function. This could be achieved through backward propagation, a method that uses the chain rule of calculus to compute gradients efficiently. Gradients are propagated backward through the network, layer by layer.

What we want to compute is how sensitive the loss to these weights variables, represented as the derivative of the loss, with respect to each of the weights. Using the chain rule, gradients at hidden layers are calculated recursively. Let the loss be *L*. The gradient with respect to $\hat{y}_i$ is:

$$\frac{\partial L}{\partial \hat{y}_i} = 2(y_i - \hat{y}_i) \quad (4)$$

The network's output $\hat{y}$ depends on the weights, biases, and activations in the preceding layer. For simplicity, consider a neuron with the output:

$$z = w_1 a_1 + w_2 a_2 + \cdots + w_n a_n + b$$

where *z* is the weighted sum, $a_i$ are the activations from the previous layer, and *b* is the bias. The neuron's activation is $\hat{y} = g(z)$, where *g* is the activation function.

The chain rule computes how the loss *L* changes with respect to each weight $w_i$:

$$\frac{\partial L}{\partial w_i} = \frac{\partial z}{\partial w_i} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial L}{\partial \hat{y}} \quad (5)$$

where $\frac{\partial z}{\partial w_i} = a_i$, the input to the neuron.

Thus, for each weight $w_i$:

$$\frac{\partial L}{\partial w_i} = 2(y_i - \hat{y}_i) \cdot g'(z) \cdot a_i \quad (6)$$

These gradients are stored in a vector called the gradient vector, which contains the partial derivatives of the loss with respect to every parameter.

### D. Gradient Descent

Once the gradients are computed, the weights and biases are updated:

$$w_i' = w_i - \eta \cdot \frac{\partial L}{\partial w_i} \quad (7)$$

where:
$\boldsymbol{\eta}$: learning rate
$\boldsymbol{w_i}'$: new weight for the i-th feature.

These process above, forward propagation, backward propagation, followed by parameter updates, constitutes one iteration. This iterative process involves repeating the following steps:

1. Forward propagation to compute the loss.
2. Backward propagation to calculate gradients.
3. Updating parameters based on the gradients.

## IV. Other Application of Linear Algebra in AI

### A. Natural Language Processing (NLP)

In NLP, linear algebra facilitates the transformation of language data into numerical representations. Word embeddings such as Word2Vec and GloVe utilize vector spaces to encode the semantic similarity between words. These embeddings are created by training on co-occurrence statistics of words in large corpora. Operations like matrix multiplication and singular value decomposition (SVD) help to optimize these representations. Moreover, in attention mechanisms like those in Transformers, matrix multiplications compute query, key, and value scores, enabling sophisticated context understanding.

### B. Computer Vision

In computer vision, images are treated as matrices where pixel values are elements. Convolutional neural networks (CNNs), a staple in this field, perform operations such as convolution (sliding filters across images) and pooling to extract features. Linear transformations like scaling and rotation are used to preprocess images, while matrix factorizations enable efficient computations in image compression and recognition tasks.

### C. Recommendation Systems

Recommendation systems use linear algebra to model user-item interactions. Matrix factorization techniques like singular value decomposition (SVD) decompose a large user-item matrix into latent factors that capture user preferences and item attributes. These decompositions enable accurate predictions of unseen interactions, powering recommendation engines in platforms like Netflix and Amazon.

### D. Autonomous Systems

Linear algebra also powers autonomous systems, such as self-driving cars and robotics, through transformations

of sensor data. For example, rotation matrices and quaternions are used to process lidar and camera inputs, enabling accurate positioning and navigation.

### *E. Reinforcement Learning*

In reinforcement learning, linear algebra is central to optimizing policies and value functions. Operations such as solving linear systems and eigenvalue computations are used to evaluate Markov decision processes (MDPs) and approximate optimal solutions.

By applying these mathematical tools, AI systems extract insights, recognize patterns, and provide actionable outcomes across diverse fields.

## V. Challenges and Future of Linear Algebra

As AI models scale and diversify, new challenges emerge in applying linear algebra effectively.

### *A. Computational Challenges*

The growing size of datasets and models necessitates efficient linear algebra operations. Issues like high memory usage, computational overhead, and numerical stability are pressing concerns.

### *B. Algorithm Innovations*

Emerging techniques aim to reduce the computational burden of linear algebra in AI. For instance, randomized linear algebra methods approximate matrix decomposition faster and with less memory. Sparse matrix operations, which ignore zero elements, also provide efficiency gains.

### *C. Hardware Acceleration*

Modern hardware, such as graphics processing units (GPUs), tensor processing units (TPUs), and quantum computers, is designed to accelerate linear algebra computations. These technologies enable AI to scale to massive datasets and complex models, unlocking new possibilities.

### *D. Future Prospects*

Quantum computing holds promise for revolutionizing linear algebra computations, offering exponential speed-ups for matrix operations. This could significantly enhance AI's ability to handle high-dimensional data and complex models.

## V. Conclusion

Linear Algebra is the foundation of Artificial Intelligence, providing the mathematical tools needed for data representation, model training, and real-world applications. From tensors and embeddings to the iterative optimization of neural networks, linear algebra enables AI systems to analyze patterns, make predictions, and continuously improve. As AI continues to evolve, the role of linear algebra will remain central, driving advancements in efficiency and capability. By learning the foundations of AI in terms of linear algebra, it helps understanding the concepts of AI at a basic level, especially in the fields of Machine Learning, Deep Learning, and Neural Networks.

## VII. Acknowledgment

The author would like to express sincere gratitude to God Almighty for blessings and grace in writing this paper. The author would also like to thank Dr. Ir. Rinaldi Munir, M.T., for being a very supportive lecturer of IF2123 Linear and Geometry Algebra.

## References

[1] R. Munir, "Review Matriks," IF2123 Aljabar Linear dan Geometri, 2023. Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-01-Review-Matriks-2023.pdf. Accessed: December 27, 2024.

[2] R. Munir, "Singular Value Decomposition (Bagian 2)," IF2123 Aljabar Linear dan Geometri, 2023. Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-22-Singular-value-decomposition-Bagian2-2023.pdf. Accessed: December 27, 2024.

[3] K. Rani, "LINEAR ALGEBRA AS THE MATHEMATICAL FOUNDATION OF ARTIFICIAL INTELLIGENCE: CONCEPTS, APPLICATIONS, AND FUTURE PROSPECTS". 2024. Source: https://ijeshonline.com/index.php/ijesh/article/view/32. Accessed: December 27, 2024.

[4] M. Mhaske, B. Gidhad, K. Maniyar and G. Ghuge, "The Role of Linear Algebra in Developing Machine Learning Solutions," 2024 3rd International Conference for Advancement in Technology (ICONAT), GOA, India, 2024, pp. 1-5.

[5] Shafi, "Linear Algebra- How it is used in AI?", 2020. Source: https://medium.com/analytics-vidhya/linear-algebra-how-uses-in-artificial-intelligence-2e1e001c65. Accessed: December 28, 2024.

[6] 3Blue1Brown,

[7] Artem Kirsanov,

## Statement

I hereby declare that the paper I wrote is my own writing, not an adaptation or translation of someone else's paper, and is not plagiarized.

Bandung, 2 January 2025

Michael Alexander Angkawijaya
13523102