Deep Learning 2.0.1: Mind and Cosmos - Towards Cosmos-Inspired Interpretable Neural Networks

Taha Bouhsine *

Posted Date: 25 June 2025

doi: 10.20944/preprints202506.2010.v1

Keywords: neural networks; physics; inverse-square laws; geometry; information theory

 doi:10.20944/preprints202506.2010.v1

Article

Deep Learning 2.0.1: Mind and Cosmos - Towards Cosmos-Inspired Interpretable Neural Networks

Taha Bouhsine

Research Scientist, MLNomads; yat@mlnomads.com

Abstract

The foundational dot product in deep learning, while ubiquitous, suffers from geometric limitations, conflating magnitude with direction and often obscuring complex data relationships. Conventional activation functions, introduced to imbue non-linearity, can further distort geometric fidelity. This paper introduces the Ɛ-product (Yat-product), a novel neural operator inspired by physical inverse-square laws, which intrinsically unifies vector alignment with spatial proximity to provide a geometrically richer measure of interaction. The Ɛ-product's inherent non-linearity and self-regulating properties form the basis of a new design philosophy, leading to Neural-Matter Networks (NMNs) that potentially obviate the need for separate activation and normalization layers to tap in the non-linearity. We demonstrate the efficacy of this approach through Ɛ-Conv in AetherResNet18 and Ɛ-Attention in AetherGPT, a GPT-2 style model. Experimental results show that these models achieve competitive or superior performance on benchmark datasets for image classification and language modeling, respectively, compared to standard architectures, despite their simplified design. This work suggests a path towards more interpretable, efficient, and geometrically faithful deep learning models by embedding non-linearity and regulation directly within the neural interaction mechanism.

Keywords: neural networks; physics; inverse-square laws; geometry; information theory

---

1. Introduction

The remarkable success of deep learning is largely built upon a standard model: the dot product for linear interaction, followed by a non-linear activation function. The dot product serves as the primary mechanism for measuring similarity and interaction between neural units, a practice dating back to the perceptron's introduction in 1958 (**?**). Non-linear activation functions, such as the Rectified Linear Unit (ReLU), are then applied to enable the network to learn complex patterns, as underscored by the universal approximation theorem (**?**). Without such non-linearities, a deep stack of layers would mathematically collapse into an equivalent single linear transformation, severely curtailing its representational capacity.

However, this ubiquitous approach has a significant cost: a loss of geometric fidelity and the need for additional components like normalization layers. The dot product itself is a geometrically impoverished measure, primarily capturing alignment while conflating magnitude with direction and often obscuring more complex structural and spatial relationships. Furthermore, the way current activation functions achieve non-linearity can exacerbate this issue. For instance, ReLU ($f(x) = \max(0, x)$) maps all negative pre-activations—which can signify a spectrum of relationships from weak dissimilarity to strong anti-alignment—to a single zero output. This thresholding, while promoting sparsity, means the network treats diverse inputs as uniformly orthogonal or linearly independent for onward signal propagation. Such a coarse-graining of geometric relationships leads to a tangible loss of information regarding the degree and nature of anti-alignment or other negative linear dependencies. This information loss, coupled with the inherent limitations of the dot product, highlights a fundamental challenge.

 doi:10.20944/preprints202506.2010.v1

This raises a central question: Can we develop a single computational operator that possesses intrinsic non-linearity while being inherently geometrically aware, thereby preserving geometric fidelity without the need for separate activation functions?

This paper proposes an elegant answer: the Ⓔ-product (pronounced Yat-product), a novel neural operator. The intuition behind the Ⓔ-product is the unification of alignment and proximity, inspired by fundamental principles observed in physical systems, particularly the concept of interaction fields governed by inverse-square laws Bouhsine (2024); Bouhsine et al. (2024). In physics, the strength of interactions (like gravity or electrostatic force) depends not only on intrinsic properties (like mass or charge) but critically on the inverse square of the distance between entities.

To this end, we introduce the Ⓔ-product, defined as:

$$\mathbb{E}(\mathbf{w}, \mathbf{x}) := \frac{\langle \mathbf{w}, \mathbf{x} \rangle^2}{\|\mathbf{w} - \mathbf{x}\|^2 + \epsilon} = \frac{\langle \mathbf{w}, \mathbf{x} \rangle^2}{(\mathbf{w} - \mathbf{x}) \cdot (\mathbf{w} - \mathbf{x}) + \epsilon} = \frac{\|\mathbf{x}\|^2 \|\mathbf{w}\|^2 \cos^2 \theta}{\|\mathbf{w}\|^2 - 2\langle \mathbf{w}, \mathbf{x} \rangle + \|\mathbf{x}\|^2 + \epsilon}$$

where $\mathbf{w}$ is a weight vector, $\mathbf{x}$ is an input vector, $\langle \cdot, \cdot \rangle$ denotes the dot product, $\| \cdot \|$ signifies the Euclidean norm, θ is the angle between $\mathbf{w}$ and $\mathbf{x}$, and ϵ is a small positive constant for numerical stability. The numerator, $\langle \mathbf{w}, \mathbf{x} \rangle^2$, emphasizes strong alignment and scales with vector magnitudes. The denominator, $\|\mathbf{w} - \mathbf{x}\|^2 + \epsilon$, incorporates an inverse-square law with respect to the Euclidean distance, penalizing large distances. This ensures the Ⓔ-product yields a high response only when $\mathbf{w}$ and $\mathbf{x}$ are well-aligned, significant in magnitude, and spatially proximate. This coupling of directional similarity with an inverse-square proximity term endows the Ⓔ-product with inherent non-linearity, potentially obviating the need for separate activation functions and allowing layers to learn complex, non-linear relationships directly.

The Ⓔ-product can be naturally extended to convolutional operations for processing structured data like images. The Ⓔ-Convolution (Yat-Conv) is defined as:

$$\mathbb{E}^*(\mathbf{W}, \mathbf{X}) := \frac{\left(\sum_{i,j} w_{ij} x_{ij}\right)^2}{\sum_{i,j} (w_{ij} - x_{ij})^2 + \epsilon}$$

where $\mathbf{W}$ and $\mathbf{X}$ represent local patches (e.g., a convolutional kernel and an input patch, respectively), and w_{ij} and x_{ij} are their corresponding elements. This formulation allows for patch-wise computation of the Ⓔ-product, integrating its geometric sensitivity into convolutional architectures.

Building upon the Ⓔ-product, we propose Neural-Matter Networks (NMNs) and Convolutional NMNs (CNMNs). NMNs are designed to preserve input topology by leveraging the Ⓔ-product's geometric awareness and avoiding aggressive, dimension-collapsing non-linearities typically found in standard architectures. In NMNs, each neuron, through its learned weight vector $\mathbf{w}$, effectively defines an interaction field. It "attracts" or responds to input vectors $\mathbf{x}$ based on the dual criteria of learned alignment and spatial proximity, analogous to how bodies with mass create gravitational fields. This approach aims to maintain critical geometric relationships, fostering more interpretable models and robust learning.

The primary contributions of this work are:

1. The introduction of the Ⓔ-product, a novel, physics-grounded neural operator that unifies directional sensitivity with an inverse-square proximity measure, designed for geometrically faithful similarity assessment.
2. The proposal of Neural-Matter Networks (NMNs), a new class of neural architectures based on the Ⓔ-product, which inherently incorporate non-linearity and are designed to preserve input topology.
3. The development and pretraining of AetherGPT, a transformer model based on GPT-2 architectural principles incorporating the Ⓔ-product, pretrained on the FineWeb 10B dataset, serving as an initial large-scale validation of the proposed concepts.

 doi:10.20944/preprints202506.2010.v1

4. A commitment to open science through the release of all associated code and models under the Affero GNU General Public License.

By reconceiving neural computation through the lens of physical interaction fields, this work seeks to bridge the empirical successes of contemporary machine learning with the structural understanding and interpretability afforded by principles derived from physics. The remainder of this paper is organized as follows: Section 2 discusses related work in geometric deep learning and alternative neural operators. Section 4 details the mathematical formulation of the E-product and NMNs. Section **??** presents experimental results, including the AetherGPT model. Section **??** provides a discussion of these results, and Section 7 concludes the paper with future directions.

2. Related Work

2.1. Inverse-Square Laws

The inverse-square law, where a quantity or intensity is inversely proportional to the square of the distance from its source, is a fundamental principle observed across numerous scientific disciplines (Kepler (1939)). This relationship signifies that as the distance from a source doubles, the intensity reduces to one-quarter of its original value.

In classical **physics**, this law is foundational. Newton's Law of Universal Gravitation describes the force between two masses (Newton (1687)), and Coulomb's Law defines the electrostatic force between charges (de Coulomb (1785)). The intensity of electromagnetic radiation, such as light, and the intensity of sound from a point source also diminish according to this principle. Gauss's Law offers a unifying mathematical framework for these phenomena in fields like electromagnetism and gravitation, connecting them to the property that the divergence of such fields is zero outside the source (Gauss (1835)). Similarly, thermal radiation intensity from a point source adheres to this law (Modest (2013)).

The inverse-square law's influence extends significantly into **engineering and applied sciences**. In health physics, it is critical for radiation protection, governing the attenuation of ionizing radiation (Knoll (2010)). Photometry applies it to illumination engineering for lighting design (Rea (2000)). In telecommunications, it underpins the free-space path loss of radio signals, as described by the Friis transmission equation (Rappaport (2002)), while radar systems experience an inverse fourth-power relationship due to the signal's two-way travel (Skolnik (2008)). Seismology observes that seismic wave energy attenuates following this law (Aki and Richards (2002)), and in fluid dynamics, the velocity field from a point source in incompressible, irrotational flow also demonstrates an inverse-square dependence (Batchelor (2000)).

Beyond the physical sciences, analogous concepts are found in **information theory, data science, and social sciences**. For instance, the Tanimoto coefficient, used in molecular similarity analysis (Tanimoto (1958)), and the Jaccard index, a metric for set similarity (Jaccard (1901)), exhibit mathematical properties akin to inverse-square decay when viewed in feature space geometry. In economics, the gravity model of trade frequently employs an inverse-square (or similar power law) relationship to predict trade flows between entities, based on their economic "mass" (e.g., GDP) and the distance separating them (Anderson (2011)), illustrating how these physical principles can offer insights into complex socio-economic phenomena.

2.2. Learning with Kernels

Learning with kernels has significantly influenced machine learning by enabling algorithms to handle complex, non-linear patterns efficiently. The introduction of Support Vector Machines (SVMs) (Cortes (1995)) laid the foundation for kernel-based learning, leveraging the kernel trick to implicitly map data into high-dimensional spaces. Schölkopf et al. (1999) formalized kernel methods, expanding their applicability to various tasks (Schölkopf et al. (1997)).

Key advancements include Kernel PCA (Schölkopf et al. (1998)) for non-linear dimensionality reduction and Gaussian Processes (Williams and Rasmussen (2006)) for probabilistic modeling. Appli-

 doi:10.20944/preprints202506.2010.v1

cations like Spectral Clustering(Ng et al. (2001)) and One-Class SVM (Schölkopf et al. (2001)) highlight the versatility of kernel methods.

To address computational challenges, techniques like the Nyström method (Williams and Seeger (2000)) and Random Fourier Features (Rahimi and Recht (2007)) have improved scalability. Recent work, such as the Neural Tangent Kernel (NTK) (Jacot et al. (2018)), bridges kernel methods and deep learning, offering insights into the dynamics of neural networks.

Furthermore, many prominent kernel functions, such as the Gaussian Radial Basis Function (RBF) kernel (Boser et al. (1992)), explicitly define similarity based on the Euclidean distance between data points, effectively giving higher weight to nearby points. This inherent sensitivity to proximity allows models like Support Vector Machines with RBF kernels or Gaussian Processes to capture local structures in the data.

While these methods leverage distance to inform the kernel matrix or model covariance, our research explores a more direct architectural integration of proximity. We propose a novel neural operator where an inverse-square law with respect to feature vector distance is fundamentally incorporated into the neuron's computation, in conjunction with measures of feature alignment. This approach differs from traditional kernel methods where the kernel function primarily serves to define a similarity measure for algorithms that operate on pairs of samples, rather than defining the intrinsic operational characteristics of individual neural units themselves.

2.3. Deep Learning

The landscape of deep learning is characterized by a continuous drive towards architectures and neural operators that offer enhanced expressivity, computational efficiency, and improved understanding of underlying data structures. Convolutional Neural Networks (CNNs) remain a foundational paradigm, particularly in vision, lauded for their proficiency in extracting hierarchical features (Lecun et al. (1998)). However, the pursuit of alternative and complementary approaches remains vibrant.

A significant trajectory involves architectures leveraging dot-product mechanisms, most prominently exemplified by the Transformer architecture and its self-attention mechanism (Vaswani et al. (2017)). This has spurred developments like Vision Transformers (ViTs) (Dosovitskiy et al. (2021)), and models such as MLP-Mixer (Tolstikhin et al. (2021)) and gMLP (Liu et al. (2021)) which, while simplifying or eschewing self-attention, still rely on operations like matrix multiplication for feature mixing, demonstrating competitive performance, particularly in computer vision.

The quest for capturing intricate data relationships has also led to renewed interest in Polynomial Neural Networks. These networks incorporate polynomial expansions of neuron inputs, enabling the modeling of higher-order correlations (Ivakhnenko (1971); Livni et al. (2014)), offering a different inductive bias compared to standard linear transformations followed by activation functions. Concurrently, Fourier Networks, such as FNet (Lee-Thorp et al. (2022)), explore the frequency domain, replacing spatial convolutions or attention with Fourier transforms for token or feature mixing, presenting an alternative for global information aggregation with potential efficiency gains.

Despite these advancements, a persistent challenge in deep learning is interpretability. The complex interplay of numerous parameters and non-linear activation functions (e.g., ReLU (Nair and Hinton (2010)), sigmoid, tanh) often renders the internal decision-making processes of deep models opaque (Montavon et al. (2018)). These diverse approaches highlight a shared pursuit for more expressive primitives. Our work contributes to this by proposing an operator that achieves non-linearity not through polynomial expansion or frequency-domain transformation, but through a unified measure of geometric alignment and proximity.

3. Theoretical Background

3.1. Revisiting Core Computational Primitives and Similarity Measures

The computational primitives used in deep learning are fundamental to how models represent and process information. This section revisits key mathematical operations and similarity measures—such

 doi:10.20944/preprints202506.2010.v1

as the dot product, convolution, cosine similarity, and Euclidean distance—that form the bedrock of many neural architectures. We will explore their individual properties and how they contribute to tasks like feature alignment, localized feature mapping, and quantifying spatial proximity. Furthermore, we will delve into the role of neural activation functions in enabling the non-linear transformations crucial for complex pattern recognition. Understanding these core concepts and their inherent characteristics is crucial for appreciating the motivation behind developing novel operators, as explored in this work, that aim to capture more nuanced relationships within data (Goodfellow et al. (2016)).

3.1.1. The Dot Product: A Measure of Alignment

The dot product, or scalar product, remains a cornerstone of neural computation, serving as the primary mechanism for quantifying the interaction between vectors, such as a neuron's weights and its input. For two vectors $\mathbf{a} = [a_1, a_2, \ldots, a_n]$ and $\mathbf{b} = [b_1, b_2, \ldots, b_n]$, it is defined as:

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

Geometrically, the dot product is proportional to the cosine of the angle between the vectors and their Euclidean magnitudes: $\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \|\mathbf{b}\| \cos(\theta)$. Its sign indicates the general orientation (acute, obtuse, or orthogonal angle), and its magnitude reflects the degree of alignment scaled by vector lengths. In machine learning, dot product scores are pervasively used to infer similarity, relevance, or the strength of activation. However, as noted in Section 1, its conflation of magnitude and directional alignment can sometimes obscure more fine-grained geometric relationships, motivating the exploration of operators that offer a more comprehensive assessment of vector interactions.

3.1.2. The Convolution Operator: Localized Feature Mapping

The convolution operator is pivotal in processing structured data, particularly in Convolutional Neural Networks (CNNs). It applies a kernel (or filter) across an input to produce a feature map, effectively an operation on two functions, f (input) and g (kernel), yielding a third that expresses how one modifies the shape of the other. For discrete signals, such as image patches and kernels, it is:

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m] g[n - m]$$

In CNNs, convolution performs several critical roles:

- **Feature Detection:** Kernels learn to identify localized patterns (edges, textures, motifs) at various abstraction levels.
- **Spatial Hierarchy:** Stacking layers allows the model to build complex feature representations from simpler ones.
- **Parameter Sharing:** Applying the same kernel across spatial locations enhances efficiency and translation equivariance.

The core computation within a discrete convolution at a specific location involves an element-wise product sum between the kernel and the corresponding input patch, which is, in essence, a dot product. Consequently, the resulting activation at each point in the feature map reflects the local alignment between the input region and the kernel. If an input patch and a kernel are orthogonal (i.e., their element-wise product sums to zero, akin to a zero dot product if they were vectorized), the convolution output at that position will be zero, indicating no local match for the feature encoded by the kernel. This reliance on dot product-like computations means that standard convolutions primarily assess feature alignment, potentially overlooking other geometric aspects of the data.

 doi:10.20944/preprints202506.2010.v1

3.1.3. Cosine Similarity: Normalizing for Directional Agreement

Cosine similarity refines the notion of alignment by isolating the directional aspect of vector relationships, abstracting away from their magnitudes. It measures the cosine of the angle between two non-zero vectors $\mathbf{A}$ and $\mathbf{B}$:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Scores range from -1 (perfectly opposite) to 1 (perfectly aligned), with 0 signifying orthogonality (decorrelation). By normalizing for vector lengths, cosine similarity provides a pure measure of orientation. This is particularly useful when the magnitude of vectors is not indicative of their semantic relationship, such as in document similarity tasks. While it effectively captures directional agreement, it explicitly discards information about vector magnitudes and, like the dot product, does not inherently account for the spatial proximity between the vectors themselves if they are points in a space (Draganov et al. (2024); Steck et al. (2024)).

3.1.4. Euclidean Distance: Quantifying Spatial Proximity

In contrast to measures of alignment, Euclidean distance quantifies the "ordinary" straight-line separation between two points (or vectors) $\mathbf{p} = (p_1, \ldots, p_n)$ and $\mathbf{q} = (q_1, \ldots, q_n)$ in an n-dimensional Euclidean space:

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

This metric is fundamental in various machine learning algorithms, including k-Nearest Neighbors and k-Means clustering, and forms the basis of loss functions like Mean Squared Error. Euclidean distance measures dissimilarity based on spatial proximity; a smaller distance implies greater similarity in terms of location within the vector space. Unlike cosine similarity, it is sensitive to vector magnitudes and their absolute positions. However, Euclidean distance alone does not directly convey information about the relative orientation or alignment of vectors, only their nearness.

The distinct characteristics of these foundational measures—alignment (dot product, cosine similarity) versus proximity (Euclidean distance)—highlight an opportunity. These foundational measures force a choice: one can measure alignment (dot product, cosine similarity) or spatial proximity (Euclidean distance), but no single, primitive operator in conventional use effectively unifies both. Neural operators that can synergistically combine these aspects, assessing not only if vectors point in similar directions but also if they are close in the embedding space, could offer a richer, more geometrically informed way to model interactions. This perspective underpins the development of the Ɇ-product introduced in Section 4.

3.2. The Role and Geometric Cost of Non-Linear Activation

While the core computational primitives provide tools to measure similarity and interaction, their inherent linearity limits the complexity of functions they can represent. To overcome this, deep neural networks employ non-linear activation functions. These are the standard method for introducing non-linearity, a necessary step for modeling intricate data patterns. However, this "fix" is imperfect, as it introduces its own set of problems, particularly concerning the preservation of the input data's geometric integrity. The remarkable expressive power of deep neural networks hinges on their capacity to model complex, non-linear relationships. This ability to approximate any continuous function to an arbitrary degree of accuracy is formally captured by the universal approximation theorem.

Theorem 1 (Universal Approximation Theorem (Cybenko (1989); Hornik et al. (1989); Huang (2020); Lu et al. (2017))). *Let σ be any continuous, bounded, and nonconstant activation function. Let I_m denote the m-dimensional unit hypercube $[0,1]^m$. The space of continuous functions on I_m is denoted by $C(I_m)$. Then, for*

 doi:10.20944/preprints202506.2010.v1

any $f \in C(I_m)$ and any $\epsilon > 0$, there exists an integer N, real constants $v_i, b_i \in \mathbb{R}$, and real vectors $w_i \in \mathbb{R}^m$ for $i = 1, \ldots, N$, such that the function $F : I_m \to \mathbb{R}$ defined by

$$F(x) = \sum_{i=1}^{N} v_i \sigma(w_i^T x + b_i)$$

satisfies $|F(x) - f(x)| < \epsilon$ for all $x \in I_m$. In simpler terms, a single hidden layer feedforward network with a sufficient number of neurons employing a suitable non-linear activation function can approximate any continuous function on compact subsets of $\mathbb{R}^m$ to any desired degree of accuracy.

This theorem underscores the critical role of non-linear activation functions. Without such non-linearities, a deep stack of layers would mathematically collapse into an equivalent single linear transformation, severely curtailing its representational capacity. Activation functions are thus not mere auxiliaries; they are the pivotal components that unlock the hierarchical and non-linear feature learning central to deep learning's success. They determine a neuron's output based on its aggregated input, and in doing so, introduce crucial selectivity: enabling the network to preferentially respond to certain patterns while attenuating or ignoring others.

3.2.1. Linear Separability and the Limitations of the Inner Product

The fundamental computation within a single artificial neuron (perceptron) is an affine transformation followed by a non-linear activation function σ:

$$y = \sigma(\langle \mathbf{w}, \mathbf{x} \rangle + b), \tag{1}$$

where $\mathbf{w}$ is the weight vector, $\mathbf{x}$ is the input vector, and b is the bias term. The decision boundary of this neuron is implicitly defined by the hyperplane where the argument to σ is zero:

$$\{\mathbf{x} \in \mathbb{R}^d \mid \langle \mathbf{w}, \mathbf{x} \rangle + b = 0\}. \tag{2}$$

This hyperplane partitions the input space $\mathbb{R}^d$ into two half-spaces. Consequently, a single neuron can only implement linearly separable functions. This is a direct consequence of the linear nature of the inner product, which can only define a linear decision boundary. While this allows for efficient computation, it severely restricts the complexity of functions that can be learned.

A classic counterexample is the XOR function, whose truth table cannot be satisfied by any single linear decision boundary. Specifically, for inputs $\mathbf{x} \in \{(0,0), (0,1), (1,0), (1,1)\} \subset \mathbb{R}^2$, there exist no $\mathbf{w} \in \mathbb{R}^2$ and $b \in \mathbb{R}$ such that $\text{sign}(\langle \mathbf{w}, \mathbf{x} \rangle + b)$ matches the XOR output (0, 1, 1, 0 respectively). This limitation stems directly from the linear nature of the inner product operation defining the separating boundary (Goodfellow et al. (2016)).

3.2.2. Non-Linear Feature Space Transformation via Hidden Layers and its Geometric Cost

Multi-layer perceptrons (MLPs) overcome this limitation by cascading transformations. A hidden layer maps the input $\mathbf{x}$ to a new representation $\mathbf{h}$ through a matrix-vector product and an element-wise activation function ϕ:

$$\mathbf{h} = \phi(W\mathbf{x} + \mathbf{b}). \tag{3}$$

Here, $W \in \mathbb{R}^{m \times d}$ is the weight matrix, $\mathbf{b} \in \mathbb{R}^m$ is the bias vector, and m is the number of hidden neurons. Each row $\mathbf{w}_i^\top$ of W corresponds to the weight vector of the i-th hidden neuron, computing $h_i = \phi(\langle \mathbf{w}_i, \mathbf{x} \rangle + b_i)$. This transforms the input space $\mathbb{R}^d$ into a feature space $\mathbb{R}^m$. The introduction of the non-linear activation function ϕ is what allows the network to learn non-linear decision boundaries. However, this gain in expressive power comes at a cost: the potential loss of geometric fidelity.

 doi:10.20944/preprints202506.2010.v1

3.2.3. Topological Distortions and Information Loss via Activation Functions

While hidden layers using the transformation $\mathbf{h} = \phi(W\mathbf{x} + \mathbf{b})$ enable the learning of non-linear functions, the introduction of the element-wise non-linear activation function ϕ, often crucial for breaking linearity, can significantly alter the topological and geometric structure of the data representation, potentially leading to information loss (Goodfellow et al. (2016)). This is a critical trade-off: gaining non-linear modeling capability while potentially discarding valuable geometric information.

Consider the mapping $T : \mathbb{R}^d \to \mathbb{R}^m$ defined by $T(\mathbf{x}) = \phi(W\mathbf{x} + \mathbf{b})$. The affine part, $A(\mathbf{x}) = W\mathbf{x} + \mathbf{b}$, performs a linear transformation (rotation, scaling, shear, projection) followed by a translation. While this affine map distorts metric properties (distances and angles, unless W is proportional to an orthogonal matrix), it preserves basic topological features like connectedness and maps lines to lines (or points) (Goodfellow et al. (2016)).

However, the subsequent application of a typical non-linear activation ϕ element-wise often leads to more drastic topological changes:

1. Non-Injectivity and Collapsing Regions: Many common activation functions render the overall mapping T non-injective.
   - ReLU ($\phi(z) = \max(0, z)$): Perhaps the most prominent example. For each hidden neuron i, the entire half-space defined by $\{\mathbf{x} \in \mathbb{R}^d \mid \langle \mathbf{w}_i, \mathbf{x} \rangle + b_i \leq 0\}$ is mapped to $h_i = 0$. Distinct points $\mathbf{x}_1, \mathbf{x}_2$ within this region, potentially far apart, become indistinguishable along the i-th dimension of the hidden space. This constitutes a significant loss of information about the relative arrangement of data points within these collapsed regions. The mapping is fundamentally many-to-one. For instance, consider two input vectors that are anti-aligned with a neuroné weight vector to different degrees—one strongly and one weakly. A ReLU activation function would map both resulting negative dot products to zero, rendering their distinct geometric opposition indistinguishable to subsequent layers. This information is irretrievably discarded.
   - Sigmoid/Tanh: While smooth, these functions saturate. Inputs $\mathbf{z}_1 = A(\mathbf{x}_1)$ and $\mathbf{z}_2 = A(\mathbf{x}_2)$ that are far apart but both fall into the saturation regime (e.g., large positive or large negative values) will map to $\mathbf{h}_1 \approx \mathbf{h}_2$. This 'squashing' effect can merge distinct clusters from the input space if they map to saturated regions in the hidden space, again losing discriminative information and distorting the metric structure.
2. Distortion of Neighborhoods: The relative distances between points can be severely distorted. Points close in the input space $\mathbb{R}^d$ might be mapped far apart in $\mathbb{R}^m$, or vice-versa (especially due to saturation or the zero-region of ReLU). This means the local neighborhood structure is not faithfully preserved. Formally, the mapping T is generally not a homeomorphism onto its image, nor is it typically bi-Lipschitz (which would provide control over distance distortions).

 doi:10.20944/preprints202506.2010.v1

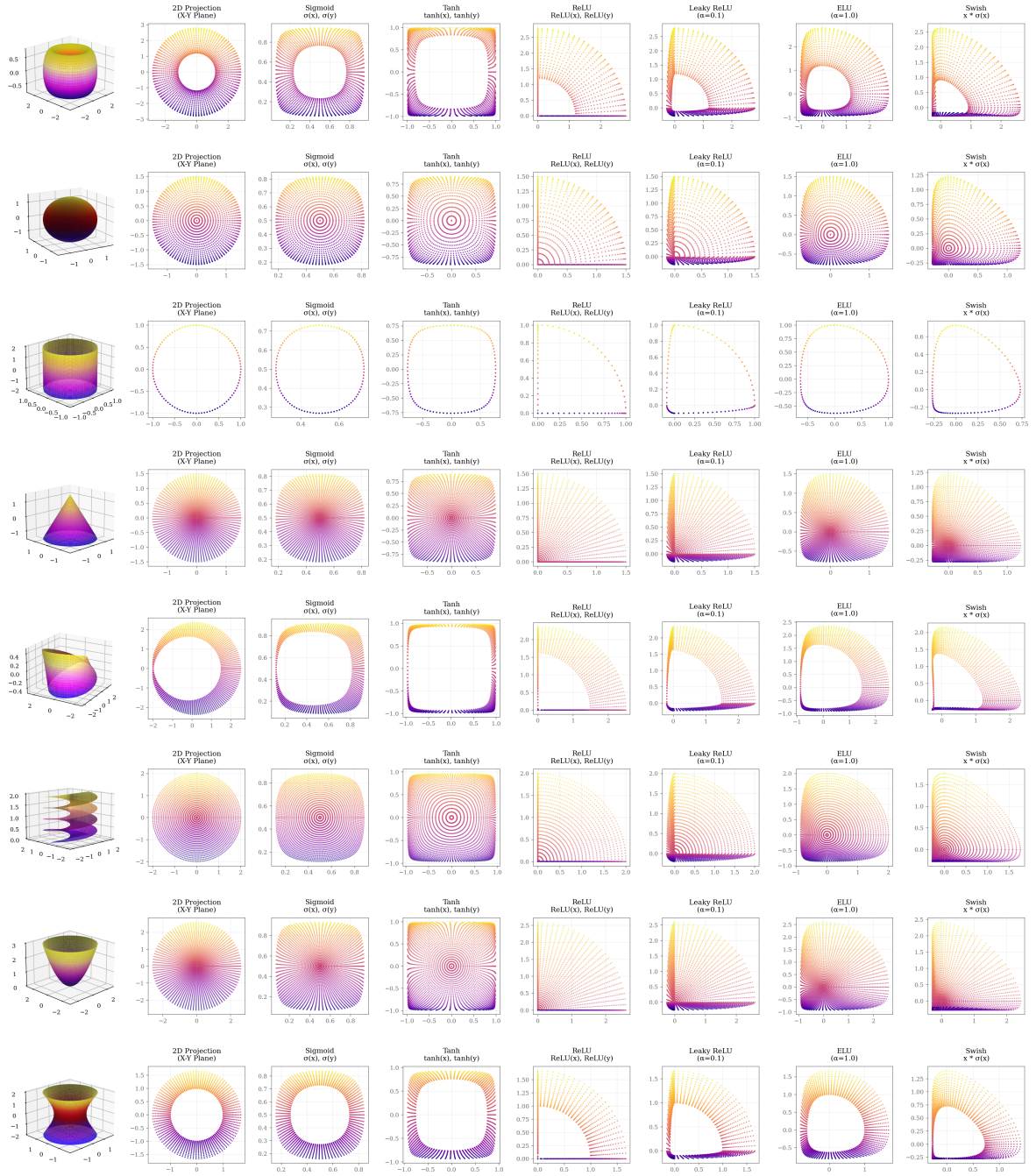

Figure 1. Illustration of how non-linear activation functions can distort the geometric structure of the input data manifold, leading to potential information loss. The original manifold (left) is transformed into a distorted representation after applying a non-linear activation functions.

While these distortions are precisely what grant neural networks their expressive power to warp the feature space and create complex decision boundaries, they come at the cost of potentially discarding information present in the original geometric configuration of the data. The network learns which information to preserve and which to discard based on the optimization objective, but the mechanism relies on potentially non-smooth or non-injective transformations introduced by ϕ. This highlights the conflation of magnitude and direction in the dot product, the information loss from activation functions, and the lack of a unified measure for proximity and alignment, setting the stage for the $\mathbb{E}$-product.

 doi:10.20944/preprints202506.2010.v1

4. Methodology: A Framework for Geometry-Aware Computation

4.1. The Ⓔ-Product: A Unified Operator for Alignment and Proximity

The methodological innovations presented in this work are fundamentally rooted in the Ⓔ-product, introduced in Section 1. This single operator serves as the foundation for subsequent layers and networks.

The Ⓔ-product is formally defined as $\mathbb{E}(\mathbf{w}, \mathbf{x}) = \frac{(\mathbf{w}^\top \mathbf{x})^2}{\|\mathbf{w}-\mathbf{x}\|^2+\epsilon}$. It exhibits a unique form of non-linearity. Unlike conventional activation functions (e.g., ReLU, sigmoid) which are often applied as separate, somewhat heuristic, transformations to introduce non-linearity after a linear operation, the non-linearity in the Ⓔ-product arises directly from its mathematical structure. It is a function of the squared dot product (capturing alignment) and the inverse squared Euclidean distance (capturing proximity) between the weight vector $\mathbf{w}$ and the input vector $\mathbf{x}$. This formulation provides a rich, explainable non-linearity based on fundamental geometric and algebraic relationships, rather than an imposed, "artificial" non-linear mapping. The interaction between the numerator and the denominator allows for complex responses that are inherently tied to the geometric interplay of the input vectors.

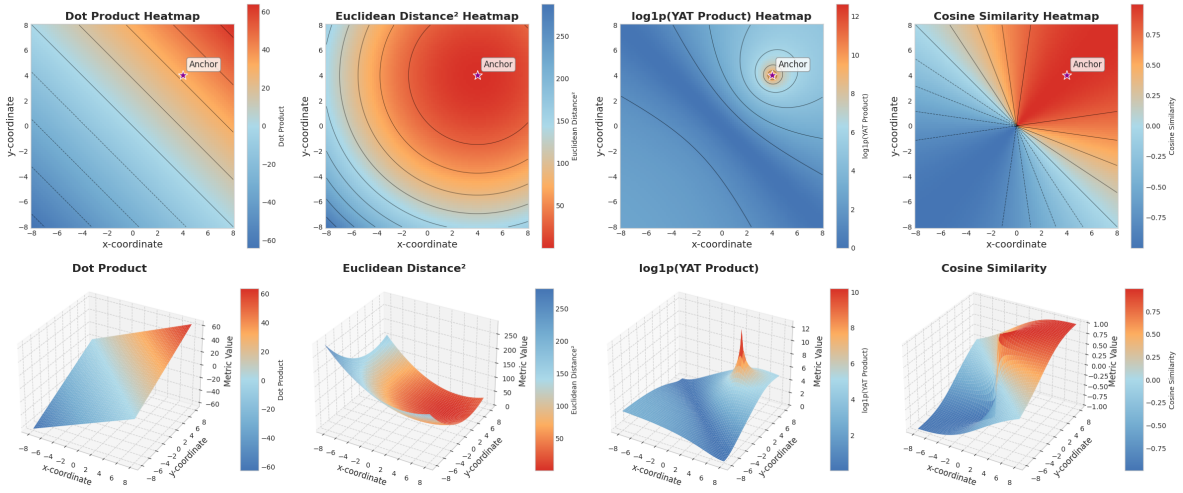

Figure 2. Visualization of the Ⓔ-product's vector field in 2D and 3D spaces. The heatmaps illustrate how the Ⓔ-product, unlike traditional similarity measures, creates a potential well around the weight vector $\mathbf{w}$, reflecting both alignment and proximity. This figure embodies the paper's philosophy of unifying geometric alignment and spatial closeness within a single neural operator, inspired by physical interaction fields. The resulting landscape demonstrates how the Ⓔ-product enables neural units to act as localized fields of influence, supporting our approach to interpretable, geometry-aware neural computation.

As visualized in Figure 2, the Ⓔ-product creates a potential well around the weight vector $\mathbf{w}$, reflecting both alignment and proximity.

To further appreciate the unique characteristics of the Ⓔ-product, it is instructive to compare it with other common similarity or distance metrics:

- **Dot Product ($\mathbf{w}^\top \mathbf{x}$):** The dot product measures the projection of one vector onto another, thus capturing both alignment and magnitude. A larger magnitude in either vector, even with constant alignment, leads to a larger dot product. While useful, its direct sensitivity to magnitude can sometimes overshadow the pure geometric alignment.
- **Cosine Similarity ($\frac{\mathbf{w}^\top \mathbf{x}}{\|\mathbf{w}\|\|\mathbf{x}\|}$):** Cosine similarity normalizes the dot product by the magnitudes of the vectors, yielding the cosine of the angle between them. This makes it purely a measure of alignment, insensitive to vector magnitudes. However, as pointed out, this means it loses information about true distance or scale; two vectors can have perfect cosine similarity (e.g., value of 1) even if one is very distant from the other, as long as they point in the same direction.
- **Euclidean Distance ($\|\mathbf{w} - \mathbf{x}\|$):** This metric computes the straight-line distance between the endpoints of two vectors. It is a direct measure of proximity. However, it does not inherently

 doi:10.20944/preprints202506.2010.v1

capture alignment. For instance, if $\mathbf{w}$ is a reference vector, all vectors $\mathbf{x}$ lying on the surface of a hypersphere centered at $\mathbf{w}$ will have the same Euclidean distance to $\mathbf{w}$, regardless of their orientation relative to $\mathbf{w}$.

- **Ɛ-Product ($\mathcal{K}_{\text{Ɛ}}(\mathbf{w}, \mathbf{x}) = \frac{(\mathbf{w}^\top \mathbf{x})^2}{\|\mathbf{w}-\mathbf{x}\|^2+\epsilon}$):** The Ɛ-product uniquely combines aspects of both alignment and proximity in a non-linear fashion. The numerator, $(\mathbf{w}^\top \mathbf{x})^2$, emphasizes strong alignment (being maximal when vectors are collinear and zero when orthogonal) and is sensitive to magnitude. The denominator, $\|\mathbf{w} - \mathbf{x}\|^2 + \epsilon$, heavily penalizes large distances between $\mathbf{w}$ and $\mathbf{x}$. This synergy allows the Ɛ-product to be highly selective. It seeks points that are not only well-aligned with the weight vector $\mathbf{w}$ but also close to it. Unlike cosine similarity, it distinguishes between aligned vectors at different distances. Unlike Euclidean distance alone, it differentiates based on orientation. This combined sensitivity allows the Ɛ-product to identify matches with a high degree of specificity, akin to locating a point with "atomic level" precision, as it requires both conditions (alignment and proximity) to be met strongly for a high output.

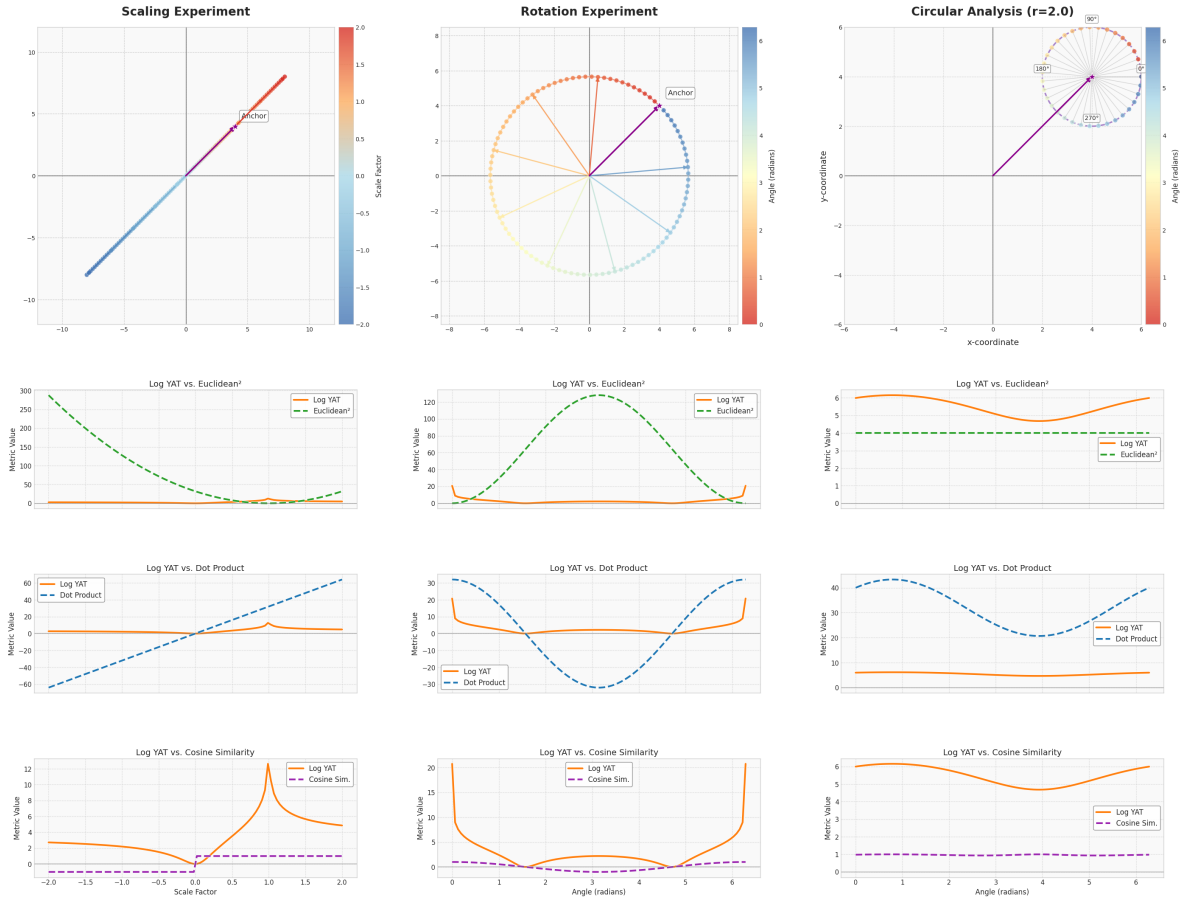

Figure 3. Comparison of the Ɛ-product with other metrics (dot product, cosine similarity, and Euclidean distance) in three distinct settings: (a) scaling vectors linearly by a factor s, (b) rotating the anchor vector, and (c) varying the distance of vectors around the anchor. The Ɛ-product's unique sensitivity to both alignment and proximity is highlighted across these scenarios.

The Ɛ-product's ability to discern between aligned vectors at varying distances, as well as its sensitivity to the angle between vectors, is illustrated in Figures 2 and 3. The vector field generated by the Ɛ-product can be visualized as a potential well around the weight vector $\mathbf{w}$, where the strength of the interaction diminishes with distance, akin to gravitational or electrostatic fields. This visualization underscores how the Ɛ-product captures both alignment and proximity in a unified manner. This combined sensitivity is crucial for tasks where both the orientation and the relative position of features are important.

 doi:10.20944/preprints202506.2010.v1

The Ɇ-product's non-linearity is not merely a mathematical curiosity; it has practical implications for neural computation. By integrating alignment and proximity into a single operator, the Ɇ-product allows for more nuanced feature learning. It can adaptively respond to inputs based on their geometric relationships with learned weight vectors, enabling the network to capture complex patterns without the need for separate activation functions.

Consider the classic XOR problem, which is not linearly separable and thus cannot be solved by a single traditional neuron (linear perceptron). The inputs are $(0,0) \to 0$, $(0,1) \to 1$, $(1,0) \to 1$, and $(1,1) \to 0$. A single Ɇ-product unit can, however, solve this. Let the weight vector be $\mathbf{w} = [1,-1]^\top$.

For $\mathbf{x} = [0,0]^\top$: $\mathbf{w}^\top\mathbf{x} = 0$, so $\mathcal{K}_{Ɇ}(\mathbf{w},\mathbf{x}) = 0$.

For $\mathbf{x} = [1,1]^\top$: $\mathbf{w}^\top\mathbf{x} = 1-1 = 0$, so $\mathcal{K}_{Ɇ}(\mathbf{w},\mathbf{x}) = 0$.

For $\mathbf{x} = [0,1]^\top$: $\mathbf{w}^\top\mathbf{x} = -1$. $\|\mathbf{w}-\mathbf{x}\|^2 = \|[1,-2]^\top\|^2 = 5$. So $\mathcal{K}_{Ɇ}(\mathbf{w},\mathbf{x}) = \frac{(-1)^2}{5+\epsilon} = \frac{1}{5+\epsilon} > 0$.

For $\mathbf{x} = [1,0]^\top$: $\mathbf{w}^\top\mathbf{x} = 1$. $\|\mathbf{w}-\mathbf{x}\|^2 = \|[0,-1]^\top\|^2 = 1$. So $\mathcal{K}_{Ɇ}(\mathbf{w},\mathbf{x}) = \frac{1^2}{1+\epsilon} = \frac{1}{1+\epsilon} > 0$.

Thus, the Ɇ-product unit with an appropriate weight vector (such as one where components have opposite signs, reflecting the XOR logic) naturally separates the XOR patterns, effectively acting as a mathematical kernel.

To understand its behavior during learning, let us consider its gradient. The partial derivative of $\mathcal{K}_{Ɇ}(\mathbf{w},\mathbf{x})$ with respect to an input component x_k (assuming $\mathbf{x} = (x_1,...,x_d)$ and $\mathbf{w} = (w_1,...,w_d)$) can be expressed. For simplicity, let $N = (\mathbf{w}^\top\mathbf{x})^2$ and $D = \|\mathbf{w}-\mathbf{x}\|^2 + \epsilon$. Then, $\frac{\partial N}{\partial x_k} = 2(\mathbf{w}^\top\mathbf{x})w_k$ and $\frac{\partial D}{\partial x_k} = -2(w_k - x_k)$. Using the quotient rule, $\frac{\partial \mathcal{K}_{Ɇ}}{\partial x_k} = \frac{\frac{\partial N}{\partial x_k}D - N\frac{\partial D}{\partial x_k}}{D^2}$. Substituting the terms:

$$\frac{\partial \mathcal{K}_{Ɇ}}{\partial x_k} = \frac{2(\mathbf{w}^\top\mathbf{x})w_k(\|\mathbf{w}-\mathbf{x}\|^2+\epsilon) - (\mathbf{w}^\top\mathbf{x})^2(-2(w_k-x_k))}{(\|\mathbf{w}-\mathbf{x}\|^2+\epsilon)^2}$$

$$= \frac{2(\mathbf{w}^\top\mathbf{x})[w_k(\|\mathbf{w}-\mathbf{x}\|^2+\epsilon) + (\mathbf{w}^\top\mathbf{x})(w_k-x_k)]}{(\|\mathbf{w}-\mathbf{x}\|^2+\epsilon)^2}$$

A key characteristic of this derivative is its continuous nature and the fact that it is generally non-zero, barring specific configurations where $\mathbf{w}^\top\mathbf{x} = 0$ (orthogonal vectors) or pathological cases. The presence of ϵ in the denominator ensures that the derivative remains well-defined and avoids division by zero, contributing to numerical stability. This contrasts with activation functions like ReLU, which have a derivative of zero for negative inputs, potentially leading to "dead neurons." The smooth and generally non-zero gradient of the Ɇ-product is hypothesized to contribute to more stable and efficient learning dynamics, reducing the reliance on auxiliary mechanisms like complex normalization schemes. The non-linearity is thus not an add-on but an intrinsic property derived from the direct mathematical interaction of vector projection (alignment, via the $(\mathbf{w}^\top\mathbf{x})^2$ term) and vector distance (proximity, via the $\|\mathbf{w}-\mathbf{x}\|^2+\epsilon$ term). This provides a mathematically grounded basis for feature learning, as the unit becomes selectively responsive to inputs that exhibit specific geometric relationships—both in terms of angular alignment and spatial proximity—to its learned weight vector $\mathbf{w}$. Consequently, $\mathbf{w}$ can be interpreted as a learned feature template or prototype that the unit is tuned to detect, with the Ɇ-product quantifying the degree of match in a nuanced, non-linear fashion.

The gradient of the Ɇ-product, being responsive across the input space, actively pushes the neuron's weights away from configurations that would lead to a zero output (neuron death, e.g., at an input of $[0,0]$ for this problem if weights were also near zero). This contrasts with a simple dot product neuron where the gradient might vanish or lead to a global minimum at zero output for certain problems. For instance, when considering gradient-based optimization, the loss landscape "seen" by the Ɇ-product neuron in the XOR context would exhibit a peak or high loss at $[0,0]$ (if that were the target for non-zero outputs), encouraging weights to move towards a state that correctly classifies. Conversely, a simple dot product neuron might present a loss landscape where a gradient-based optimizer could find a stable (but incorrect) minimum at zero output. This ability to avoid such dead zones and actively shape the decision boundary makes it helpful to solve problems like XOR with a single unit, leveraging its inherent non-linearity as a mathematical kernel.

 doi:10.20944/preprints202506.2010.v1

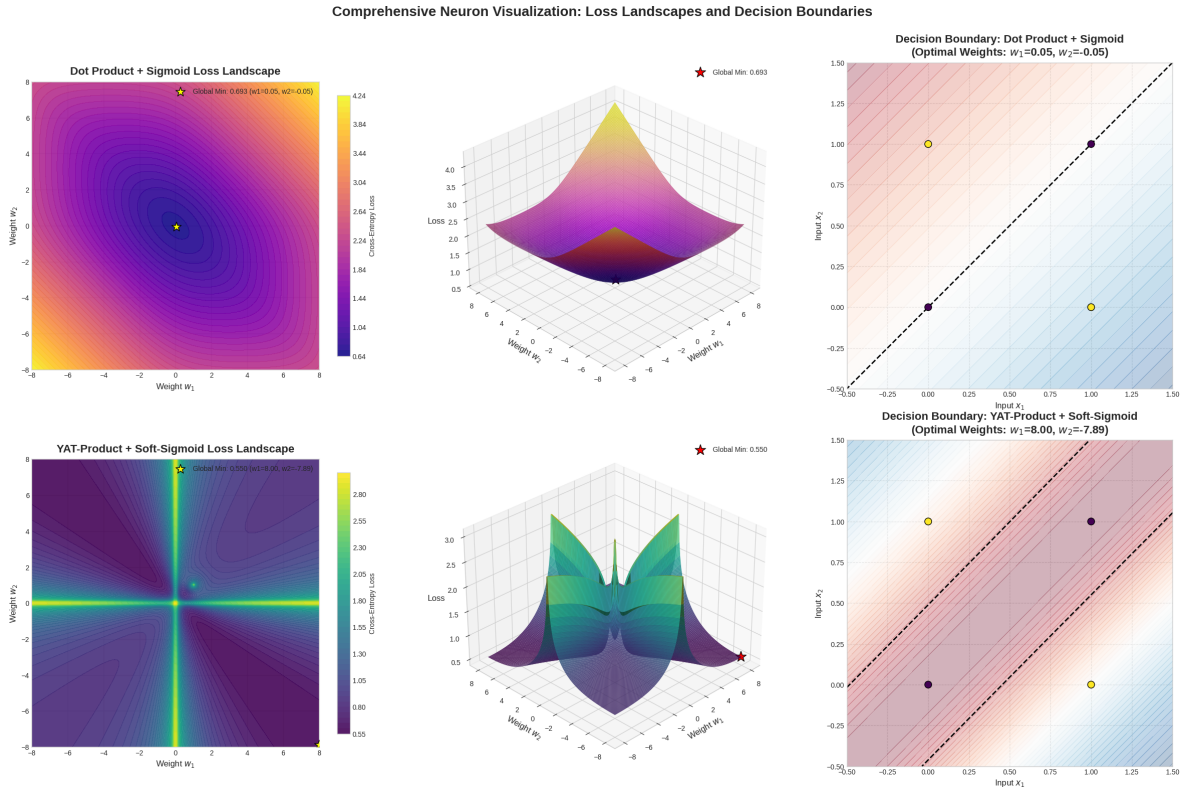

Figure 4. Comparison of the loss landscape for a simple dot product neuron and a Ɛ-product neuron. The dot product neuron has a stable minimum at zero output which doesn't solve the xor problem and cause the neuron death, while the Ɛ-product neuron resides in a valley of orthogonality, allowing it to avoid dead zones and actively shape the decision boundary. This illustrates the Ɛ-product's ability to solve problems like XOR with a single unit, leveraging its inherent non-linearity as a mathematical kernel.

Conceptually, the decision boundary or vector field generated by a simple dot product neuron is linear, forming a hyperplane that attempts to separate data points. In contrast, the Ɛ-product generates a more complex, non-linear vector field. This field can be visualized as creating a series of potential wells or peaks centered around the weight vector $\mathbf{w}$, with the strength of influence decaying with distance. The condition $\mathbf{w}^\top \mathbf{x} = 0$ defines a "valley" of zero output where vectors are orthogonal to the weight vector. This structure allows for more nuanced and localized responses, akin to a superposition of influences rather than a single linear division, enabling the capture of intricate patterns in the data.

4.2. Design Philosophy: Intrinsic Non-Linearity and Self-Regulation

A central hypothesis underpinning our methodological choices is that the Ɛ-product (Section 1) possesses inherent non-linearity and self-regulating properties that can reduce or eliminate the need for conventional activation functions (e.g., ReLU, sigmoid, GeLU) and normalization layers (e.g., Batch Normalization, Layer Normalization).

This philosophy recontextualizes the fundamental components of neural computation. Neuron weights ($\mathbf{w}$) and input signals ($\mathbf{x}$) are not merely operands in a linear transformation followed by a non-linear activation; instead, they are conceptualized as co-equal vector entities inhabiting a shared, high-dimensional feature manifold. Within this framework, each vector can be viewed as an analogue to a fundamental particle or feature vector, with its constituent dimensions potentially encoding excitatory, inhibitory, or neutral characteristics relative to other entities in the space. The Ɛ-product (Section 1) then transcends simple similarity assessment; it functions as a sophisticated interaction potential, $\mathcal{K}_{Ɛ}(\mathbf{w}, \mathbf{x}) = \frac{(\mathbf{w}^\top \mathbf{x})^2}{\|\mathbf{w}-\mathbf{x}\|^2+\epsilon}$, quantifying the 'field effects' between these vector entities. This interaction is reminiscent of n-body problems in physics. In machine learning, it draws parallels with, yet distinctively evolves from, learned metric spaces in contrastive learning, particularly those employing a triplet loss framework. While triplet loss aims to pull positive pairs closer and push negative

 doi:10.20944/preprints202506.2010.v1

pairs apart in the embedding space, our Ⓔ-product seeks a more nuanced relationship: 'positive' interactions (high Ⓔ-product value) require both strong alignment (high $(\mathbf{w}^\top \mathbf{x})^2$) and close proximity (low $\|\mathbf{w} - \mathbf{x}\|^2$). Conversely, 'negative' or dissimilar relationships are not merely represented by distance, but more significantly by orthogonality (leading to a vanishing numerator), which signifies a form of linear independence and contributes to the system's capacity for true non-linear discrimination. Crucially, the non-linearity required for complex pattern recognition is not an external imposition (e.g., via a separate activation function) but is intrinsic to this interaction potential. The interplay between the squared dot product (alignment sensitivity) and the inverse squared Euclidean distance (proximity sensitivity) in its formulation directly sculpts a complex, non-linear response landscape without recourse to auxiliary functions.

Furthermore, this conceptualization of the Ⓔ-product as an intrinsic interaction potential suggests inherent self-regulating properties. The distance-sensitive denominator, $\|\mathbf{w} - \mathbf{x}\|^2 + \epsilon$, acts as a natural dampening mechanism. As the 'distance' (dissimilarity in terms of position) between interacting vector entities $\mathbf{w}$ and $\mathbf{x}$ increases, the strength of their interaction, and thus the resultant activation, diminishes quadratically. This behavior is hypothesized to inherently curtail runaway activations and stabilize learning dynamics by ensuring that responses are localized and bounded. Such intrinsic stabilization contrasts sharply with conventional approaches that rely on explicit normalization layers (e.g., Batch Normalization, Layer Normalization) to manage activation statistics post-hoc. These layers, while effective, introduce additional computational overhead, can obscure direct input-output relationships, and sometimes complicate the theoretical analysis of network behavior. The Ⓔ-product's formulation, therefore, offers a pathway to architectures where regulatory mechanisms are embedded within the primary computational fabric of the network.

The inherent non-linearity of the Ⓔ-product, coupled with the self-regulating properties suggested by its formulation, are central to our hypothesis that it can form the basis of powerful and robust neural architectures. These intrinsic characteristics open avenues for simplifying network design, potentially reducing reliance on or even eliminating conventional activation functions and normalization layers.

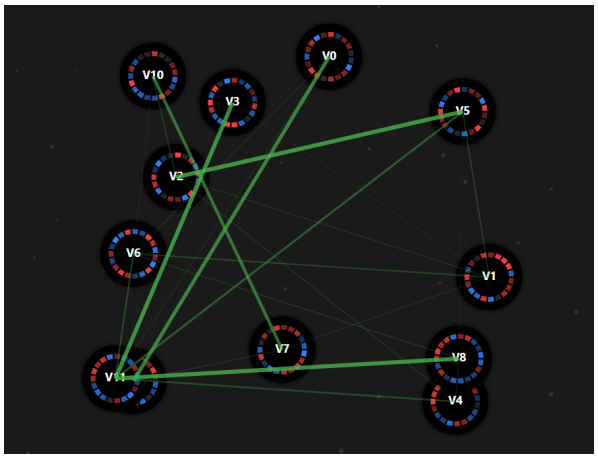

Figure 5. The Vectoverse: Conceptualizing neural computation where weight vectors ($\mathbf{w}$) and input vectors ($\mathbf{x}$) are akin to fundamental particles (vectoms). The interaction force between them is quantified by the Ⓔ-product, which measures their alignment and proximity, defining a field of influence.

4.3. Core Building Blocks

Now, we show how the Ⓔ-product is operationalized into reusable layers.

4.3.1. The Neural Matter Network (NMN) Layer

The first and simplest application of the Ⓔ-product is in Neural-Matter Network (NMN) layers. These networks represent a departure from traditional Multi-Layer Perceptrons (MLPs) by employing the non-linear, spatially-aware $\mathcal{K}_{\text{Ⓔ}}$-kernel (derived from the Ⓔ-product, see Section 4.1) as the primary interaction mechanism, instead of the conventional linear projection ($\langle \mathbf{w}, \mathbf{x} \rangle$).

 doi:10.20944/preprints202506.2010.v1

An NMN layer transforms an input vector ($\mathbf{x} \in \mathbb{R}^d$) into an output (here, we consider a scalar output h for simplicity, extendable to vector outputs $\mathbf{h}$ by aggregating the influence of multiple "neural matter" units). Each unit i is defined by a weight vector ($\mathbf{w}_i \in \mathbb{R}^d$) (acting as a positional anchor or prototype) and a bias term ($b_i \in \mathbb{R}$). The layer output is computed as:

$$h(\mathbf{x}) = \left(s \cdot \sum_{i=1}^{n} \left(\mathcal{K}_{\text{Ɛ}}(\mathbf{w}_i, \mathbf{x}) + b_i \right) \right) = \left(s \cdot \sum_{i=1}^{n} \left(\frac{(\mathbf{w}_i^\top \mathbf{x})^2}{\|\mathbf{w}_i - \mathbf{x}\|^2 + \epsilon} + b_i \right) \right)$$

where:

- $\mathbf{w}_i$ is the weight vector of the i-th NMN unit.
- b_i is the bias term for the i-th NMN unit.
- $\mathcal{K}_{\text{Ɛ}}(\mathbf{w}_i, \mathbf{x})$ represents the Ɛ-product between the weight vector $\mathbf{w}_i$ and the input $\mathbf{x}$.
- n is the number of NMN units in the layer.
- s is a scaling factor.

This formulation allows each NMN unit to respond based on both alignment and proximity to its learned weight vector.

Furthermore, the representational power of NMNs can be formally stated. We posit the following theorem regarding their approximation capabilities:

Theorem 2 (Universal Approximation for NMNs with $\mathcal{K}_{\text{Ɛ}}$-Kernel)**.** *Let K be a compact subset of $\mathbb{R}^d$. Let $C(K, \mathbb{R}^m)$ denote the space of continuous functions from K to $\mathbb{R}^m$. Let the $\mathcal{K}_{\text{Ɛ}}$-kernel be defined as $\mathcal{K}_{\text{Ɛ}}(\mathbf{w}, \mathbf{x}) = \frac{(\mathbf{w}^\top \mathbf{x})^2}{\|\mathbf{w}-\mathbf{x}\|^2+\epsilon}$ for some $\epsilon > 0$. Then, for any function $f \in C(K, \mathbb{R}^m)$ and any $\delta > 0$, there exist:*

- *a positive integer n (number of units),*
- *weight vectors $\mathbf{w}_k \in \mathbb{R}^d$ for $k = 1, \ldots, n$,*
- *scalar bias terms $b_k \in \mathbb{R}$ for $k = 1, \ldots, n$,*
- *vector coefficients $\mathbf{c}_k \in \mathbb{R}^m$ for $k = 1, \ldots, n$,*

such that the NMN layer $h : K \to \mathbb{R}^m$ defined by

$$h(\mathbf{x}) = \sum_{k=1}^{n} \mathbf{c}_k \left(\mathcal{K}_{\text{Ɛ}}(\mathbf{w}_k, \mathbf{x}) + b_k \right)$$

satisfies $\sup_{\mathbf{x} \in K} \|h(\mathbf{x}) - f(\mathbf{x})\| < \delta$. This approximation is achieved without recourse to traditional, separate activation functions, relying solely on the inherent non-linearity of the $\mathcal{K}_{\text{Ɛ}}$-kernel based terms and their linear combination.

This property underscores the potential of the $\mathcal{K}_{\text{Ɛ}}$-kernel to form the basis of powerful function approximators, distinguishing NMNs from classical MLP architectures that explicitly require non-linear activation functions for universal approximation.

4.3.2. Convolutional Neural-Matter Networks (CNMNs) and the Ɛ-Convolution Layer

To extend the principles of Neural-Matter Networks (NMNs) (Section 4.3.1) and the Ɛ-product (Section 1) to spatially structured data like images, we introduce the Ɛ-Convolution (Yat-Conv) layer. This layer adapts the Ɛ-product to operate on local receptive fields, analogous to standard convolutional layers. The Ɛ-Conv operation is defined as:

$$(\text{Ɛ-Conv}(K, I))_{i,j} = \text{Ɛ}^*(K, I_{i,j}) = \frac{\langle K, I_{i,j} \rangle^2}{\|K - I_{i,j}\|^2 + \epsilon}$$

where K is the convolutional kernel and $I_{i,j}$ is the input patch at location (i, j) corresponding to the receptive field of the kernel.

 doi:10.20944/preprints202506.2010.v1

4.3.3. The 𝔼-Attention Mechanism

To extend the 𝔼-product's application to sequence modeling, we propose the 𝔼-Attention mechanism. This mechanism serves as an alternative to the standard scaled dot-product attention found in transformer architectures by replacing the dot product used for calculating query-key similarity with the 𝔼-product (Section 4.1). Given Query (Q), Key (K), and Value (V) matrices, 𝔼-Attention is computed as:

$$\text{𝔼-Attention}(Q, K, V) = \text{softmax}\Big(s \cdot (Q \mathbb{E} K^T)\Big) V$$

where the operation $Q \mathbb{E} K^T$ signifies applying the 𝔼-product element-wise between query and key vectors (e.g., the (i, j)-th element is $\mathbb{E}(q_i, k_j)$), and s is a scaling factor.

4.4. Architectural Implementations

The development of architectures like AetherResNet and AetherGPT without standard components (like separate activation and normalization layers) is a deliberate effort to test the hypothesis outlined in Section 4.2. Key architectural distinctions driven by this philosophy include:

- **Fundamental Operator Replacement**: The standard dot product is replaced by the 𝔼-product. This is manifested as 𝔼-Convolution (Equation **??**) in convolutional networks and 𝔼-Attention (Equation 4.3.3) in transformer-based models.
- **Feed-Forward Networks (FFNs)**: The FFNs within are constructed using NMN layers (Section 4.3.1) without explicit non-linear activation functions.
- **Omission of Standard Layers**: Consistent with this design philosophy, explicit activation functions and standard normalization layers are intentionally omitted.

By minimizing reliance on these traditional layers, we aim to explore simpler, potentially more efficient, and interpretable models where the primary computational operator itself handles these crucial aspects of neural processing. Furthermore, this principle of substituting the dot product with the 𝔼-product is not limited to the architectures presented and holds potential for enhancing other neural network paradigms.

4.4.1. Convolutional NMNs:

AetherResNet is a Convolutional Neural-Matter Network (CNMN) built by replacing all standard convolutions in a ResNet18 architecture with the 𝔼-Convolution layers. Building upon the 𝔼-Conv layer, CNMNs adapt conventional convolutional architectures by employing the 𝔼-Conv layer as the primary feature extraction mechanism. The core idea is to leverage the geometric sensitivity and inherent non-linearity of the 𝔼-product within deep convolutional frameworks. Consistent with the philosophy of Section 4.2, AetherResNet omits Batch Normalization and activation functions. The design relies on the hypothesis that the 𝔼-product itself provides sufficient non-linearity and a degree of self-regulation.

A key architectural adaptation in our CNMN implementation is the removal of the traditional fully connected (FC) classification layer at the end of the network. Instead, classification is performed by appending a final 𝔼-Convolution layer, where the number of output channels is set equal to the number of target classes. This is followed by a global average pooling operation, which aggregates the spatial outputs of each class channel into a single score per class. This approach maintains the geometric and spatial inductive biases of the convolutional framework all the way through to the output, and further simplifies the architecture by eliminating the need for flattening and dense layers. Further details on the specific architectural adaptations of AetherResNet are provided in the experimental setup.

4.4.2. YatFormer: AetherGPT

AetherGPT is a YatFormer model that uses 𝔼-Attention (from Section 4.3.3) for sequence interaction and NMN layers (from Section 4.3.1) in its feed-forward blocks. Building upon the 𝔼-Attention mechanism, which forms its cornerstone, we introduce YatFormer, a family of transformer-based

 doi:10.20944/preprints202506.2010.v1

models. As a specific instantiation for our investigations, we developed AetherGPT. This model adapts the architectural principles of GPT-2. Again, following the philosophy of Section 4.2, it omits standard normalization and activation layers.

4.5. Output Processing for Non-Negative Scores

The $\mathbb{E}$-product and its derivatives, such as the $\mathcal{K}_{\mathbb{E}}$-kernel, naturally yield non-negative scores. In many machine learning contexts, particularly when these scores need to be interpreted as probabilities, attention weights, or simply normalized outputs, it is essential to apply a squashing function to map them to a desired range (e.g., [0, 1] or ensuring a set of scores sum to 1).

Traditional squashing functions, however, present challenges when applied to non-negative inputs:

- **Standard Sigmoid Function ($\sigma(x) = \frac{1}{1+e^{-x}}$):** When applied to non-negative inputs ($x \geq 0$), the standard sigmoid function produces outputs in the range $[0.5, 1)$. The minimum value of 0.5 for $x = 0$ renders it unsuitable for scenarios where small non-negative scores should map to values close to 0.
- **Standard Softmax Function ($\text{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$):** The use of the exponential function in softmax can lead to *hard* distributions, where one input value significantly dominates the output, pushing other probabilities very close to zero. While this is often desired for classification, it can be too aggressive if a softer assignment of probabilities or attention is preferred.

Given these limitations and the non-negative nature of $\mathbb{E}$-product scores, we consider alternative squashing functions more suited to this domain:

- **softermax:** This function normalizes a score x_k (optionally raised to a power $n > 0$) relative to the sum of a set of non-negative scores $\{x_i\}$ (each raised to n), with a small constant $\epsilon > 0$ for numerical stability. It is defined as:

$$\text{softermax}_n(x_k, \{x_i\}) = \frac{x_k^n}{\epsilon + \sum_i x_i^n}$$

  The power n controls the sharpness of the distribution: $n = 1$ recovers the original Softermax, while $n > 1$ makes the distribution harder (more peaked), and $0 < n < 1$ makes it softer.
- **soft-sigmoid:** This function squashes a single non-negative score $x \geq 0$ (optionally raised to a power $n > 0$) into the range $[0, 1)$. It is defined as:

$$\text{soft-sigmoid}_n(x) = \frac{x^n}{1 + x^n}$$

  The power n modulates the softness: higher n makes the function approach zero faster for large x, while $n < 1$ makes the decay slower.
- **soft-tanh:** This function maps a non-negative score $x \geq 0$ (optionally raised to a power $n > 0$) to the range $[-1, 1)$ by linearly transforming the output of soft-sigmoid. It is defined as:

$$\text{soft-tanh}_n(x) = 2 \cdot (\text{soft-sigmoid}_n(x) - \tfrac{1}{2}) = \frac{x^n - 1}{1 + x^n}$$

  The power n again controls the transition sharpness: higher n makes the function approach -1 more quickly for large x.

 doi:10.20944/preprints202506.2010.v1

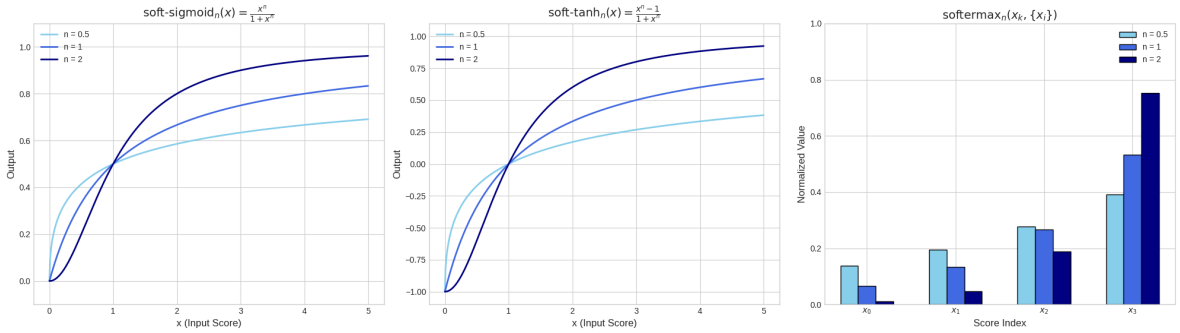

Figure 6. Visualization of the softermax, soft-sigmoid, and soft-tanh functions. These functions are designed to handle non-negative inputs from the Ɇ-product and its derivatives, providing appropriate squashing mechanisms that maintain sensitivity across the range of non-negative inputs.

These functions are particularly well-suited for the outputs of Ɇ-product-based computations, as they maintain sensitivity across the range of non-negative inputs while avoiding the pitfalls of standard activation functions. The softermax function provides a normalized distribution over multiple scores, soft-sigmoid offers a gentle squashing to [0, 1), and soft-tanh maps to [-1, 1) while preserving the non-negative nature of the input.

5. Results and Discussion

In-Progress

6. Explainability

In-Progress

7. Conclusion

Perhaps artificial intelligence's greatest limitation has been our stubborn fixation on the human brain as the pinnacle of intelligence. The universe itself, governed by elegant and powerful laws, demonstrates intelligence far beyond human cognition. These fundamental laws, which shape galaxies and guide quantum particles, represent a deeper form of intelligence that we have largely ignored in our pursuit of AI.

In this paper, we challenge the conventional AI paradigm. We broke free from biological metaphors by drawing direct inspiration from inverse-square law interactions in physics. The Ɇ-product, with its inherent non-linearity and geometric sensitivity, allows for a more nuanced understanding of vector interactions, capturing both alignment and proximity in a single operation. This approach not only simplifies the architecture of neural networks but also enhances their interpretability and robustness.

Acknowledgments

This research was supported by the MLNomads community and the broader open-source AI community. We extend special thanks to Dr. D. Sculley for his insightful feedback on kernel learning. We are also grateful to Kaggle and Colab for providing the computational resources instrumental to this research. Additionally, this work received support from the Google Developer Expert program, the Google AI/ML Developer Programs team, and Google for Startups in the form of Google Cloud Credits. We used BashNota and Weights and Biases for managing hypotheses and validating our research.

Disclaimer

This research provides foundational tools to enhance the safety, explainability, and interpretability of AI systems. These tools are vital for ensuring precise human oversight, a prerequisite to prevent AI from dictating human destiny.

 doi:10.20944/preprints202506.2010.v1

The authors disclaim all liability for any use of this research that contradicts its core objectives or violates established principles of safe, explainable, and interpretable AI. This material is provided "as is," without any warranties. The end-user bears sole responsibility for ensuring ethical, responsible, and legally compliant applications.

We explicitly prohibit any malicious application of this research, including but not limited to, developing harmful AI systems, eroding privacy, or institutionalizing discriminatory practices. This work is intended exclusively for academic and research purposes.

We encourage active engagement from the open-source community, particularly in sharing empirical findings, technical refinements, and derivative works. We believe collaborative knowledge generation is essential for developing more secure and effective AI systems, thereby safeguarding human flourishing.

Our hope is that this research will spur continued innovation in AI safety, explainability, and interpretability. We expect the global research community to use these contributions to build AI systems demonstrably subordinate to human intent, thus mitigating existential risks. All researchers must critically evaluate the far-reaching ethical and moral implications of their work.

License

References

Aki, Keiiti and Paul G. Richards. 2002. *Quantitative Seismology* (2 ed.). Sausalito, CA: University Science Books.

Anderson, James E. 2011. The gravity model. *Annual Review of Economics* 3(1), 133–160.

Batchelor, G. K. 2000. *An Introduction to Fluid Dynamics*. Cambridge, UK: Cambridge University Press.

Boser, Bernhard E., Isabelle M. Guyon, and Vladimir N. Vapnik. 1992. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, New York, NY, USA, pp. 144–152. Association for Computing Machinery. https://doi.org/10.1145/130385.130401.

Bouhsine, Taha. 2024. Deep learning 2.0: Artificial neurons that matter – reject correlation, embrace orthogonality.

Bouhsine, Taha, Imad El Aaroussi, Atik Faysal, and Wang. 2024. Simo loss: Anchor-free contrastive loss for fine-grained supervised contrastive learning. In *Submitted to The Thirteenth International Conference on Learning Representations*. under review.

Cortes, Corinna. 1995. Support-vector networks. *Machine Learning*.

Cybenko, George. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems* 2(4), 303–314.

de Coulomb, Charles-Augustin. 1785. Premier mémoire sur l'électricité et le magnétisme. *Histoire de l'Académie Royale des Sciences*, 1–31. in French.

Dosovitskiy, Alexey, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An image is worth 16x16 words: Transformers for image recognition at scale.

Draganov, Andrew, Sharvaree Vadgama, and Erik J Bekkers. 2024. The hidden pitfalls of the cosine similarity loss. *arXiv preprint arXiv:2406.16468*.

Gauss, Carl Friedrich. 1835. *Allgemeine Lehrsätze in Beziehung auf die im verkehrten Verhältniss des Quadrats der Entfernung wirkenden Anziehungs- und Abstossungskräfte*. Göttingen: Dietrich.

Goodfellow, Ian, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep learning*, Volume 1. MIT Press.

Hornik, Kurt, Maxwell B. Stinchcombe, and Halbert L. White. 1989. Multilayer feedforward networks are universal approximators. *Neural Networks* 2, 359–366.

 doi:10.20944/preprints202506.2010.v1

Huang, Changcun. 2020. Relu networks are universal approximators via piecewise linear or constant functions. *Neural Computation 32*(11), 2249–2278.

Ivakhnenko, Alexey Grigorevich. 1971. Polynomial theory of complex systems. *IEEE transactions on Systems, Man, and Cybernetics* (4), 364–378.

Jaccard, Paul. 1901. Étude comparative de la distribution florale dans une portion des alpes et des jura. *Bulletin de la Société Vaudoise des Sciences Naturelles 37*, 547–579.

Jacot, Arthur, Franck Gabriel, and Clément Hongler. 2018. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems 31*.

Kepler, Johannes. 1939. Ad vitellionem paralipomena, quibus astronomiae pars optica traditur. 1604. *Johannes Kepler: Gesammelte Werke, Ed. Walther von Dyck and Max Caspar, Münchenk*.

Knoll, Glenn F. 2010. *Radiation Detection and Measurement* (4 ed.). Hoboken, NJ: John Wiley & Sons.

Lecun, Y., L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*(11), 2278–2324. https://doi.org/10.1109/5.726791.

Lee-Thorp, James, Joshua Ainslie, Ilya Eckstein, and Santiago Ontanon. 2022, May. FNet: Mixing Tokens with Fourier Transforms. arXiv:2105.03824 [cs].

Liu, Hanxiao, Zihang Dai, David R. So, and Quoc V. Le. 2021, June. Pay Attention to MLPs. arXiv:2105.08050 [cs].

Livni, Roi, Shai Shalev-Shwartz, and Ohad Shamir. 2014. An algorithm for training polynomial networks.

Lu, Zhou, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. 2017. The expressive power of neural networks: A view from the width. *Advances in neural information processing systems 30*.

Modest, Michael F. 2013. *Radiative Heat Transfer* (3 ed.). New York: Academic Press.

Montavon, Grégoire, Wojciech Samek, and Klaus-Robert Müller. 2018, February. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing 73*, 1–15. https://doi.org/10.1016/j.dsp.2017.10.011.

Nair, Vinod and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814.

Newton, Isaac. 1687. *Philosophiæ Naturalis Principia Mathematica*. London: S. Pepys.

Ng, Andrew, Michael Jordan, and Yair Weiss. 2001. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems 14*.

Rahimi, Ali and Benjamin Recht. 2007. Random features for large-scale kernel machines. *Advances in neural information processing systems 20*.

Rappaport, Theodore S. 2002. *Wireless Communications: Principles and Practice* (2 ed.). Upper Saddle River, NJ: Prentice Hall.

Rea, Mark S. 2000. *The IESNA Lighting Handbook: Reference & Application* (9 ed.). New York: Illuminating Engineering Society of North America.

Schölkopf, Bernhard, John C Platt, John Shawe-Taylor, Alex J Smola, and Robert C Williamson. 2001. Estimating the support of a high-dimensional distribution. *Neural computation 13*(7), 1443–1471.

Schölkopf, Bernhard, Alexander Smola, and Klaus-Robert Müller. 1997. Kernel principal component analysis. In *International conference on artificial neural networks*, pp. 583–588. Springer.

Schölkopf, Bernhard, Alexander Smola, and Klaus-Robert Müller. 1998. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation 10*(5), 1299–1319.

Skolnik, Merrill I. 2008. *Radar Handbook* (3 ed.). New York: McGraw-Hill Education.

Steck, Harald, Chaitanya Ekanadham, and Nathan Kallus. 2024. Is cosine-similarity of embeddings really about similarity? In *Companion Proceedings of the ACM Web Conference 2024*, pp. 887–890.

Tanimoto, Taffee T. 1958. Elementary mathematical theory of classification and prediction.

Tolstikhin, Ilya, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. 2021. Mlp-mixer: An all-mlp architecture for vision. *arXiv preprint arXiv:2105.01601*.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems 30*.

 doi:10.20944/preprints202506.2010.v1

Williams, Christopher and Matthias Seeger. 2000. Using the nyström method to speed up kernel machines. *Advances in neural information processing systems 13*.

Williams, Christopher KI and Carl Edward Rasmussen. 2006. *Gaussian processes for machine learning*, Volume 2. MIT press Cambridge, MA.