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

Taha Bouhsine *

Posted Date: 13 August 2025

doi: 10.20944/preprints202506.2010.v2

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

*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 standard dot product, foundational to deep learning, conflates magnitude and direction, limiting geometric expressiveness and often necessitating additional architectural components such as activation and normalization layers. We introduce the Yat-product (Yat-product), a novel neural operator inspired by physical inverse-square laws, which intrinsically unifies vector alignment and spatial proximity within a single, non-linear, and self-regulating computation. This operator forms the basis of Neural-Matter Networks (NMNs), a new class of architectures that embed non-linearity and normalization directly into the core interaction mechanism, obviating the need for separate activation or normalization layers. We demonstrate that NMNs, and their convolutional and attention-based extensions, achieve competitive or superior performance on benchmark tasks in image classification and language modeling, while yielding more interpretable and geometrically faithful representations. Theoretical analysis establishes the Yat-product as a positive semi-definite Mercer kernel with universal approximation and stable gradient properties. Our results suggest a new design paradigm for deep learning: by grounding neural computation in geometric and physical principles, we can build models that are not only efficient and robust, but also inherently interpretable. **Code Repository:** https://github.com/mlnomadpy/nmn.

**Keywords:** neural operators; geometric deep learning; interpretable AI; physics-inspired models; inverse-square law; kernel methods; neural-matter networks; non-linear neural computation; attention mechanisms; convolutional architectures; universal approximation; self-regularization; prototype learning; vortex phenomenon; space partitioning; explainable machine learning; deep learning theory; functional analysis; information geometry; dot product squared devided by euclidean distance squared

## 1. Introduction

Deep learning practitioners are often forced into a false dichotomy: to measure alignment (via the dot product) or to measure proximity (via Euclidean distance). Models that require sensitivity to both must rely on complex, multi-layered architectures to approximate this relationship. This work challenges that paradigm by introducing a primitive operator that unifies these concepts.

At the heart of this dichotomy lies the standard model underpinning most deep learning systems: 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 [1–6]. Non-linear activation functions, such as the Rectified Linear Unit (ReLU) [7–9], are then applied to enable the network to learn complex patterns, as underscored by the universal approximation theorem [3,4]. 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 [10–14]. Furthermore, the

 doi:10.20944/preprints202506.2010.v2



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.

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 Yat-product (pronounced Yat-product), a novel neural operator. The intuition behind the Yat-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 [15–20]. 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 Yat-product, defined as:

$$\text{Yat}(\mathbf{w}, \mathbf{x}) := \frac{\langle \mathbf{w}, \mathbf{x} \rangle^2}{\|\mathbf{w} - \mathbf{x}\|^2 + \epsilon} \tag{1}$$

where $\mathbf{w}$ is a weight vector, $\mathbf{x}$ is an input vector, and $\epsilon$ is a small positive constant for numerical stability. This operator, inspired by physical inverse-square laws, unifies alignment and proximity in a single, non-linear computation. See Section 3 for a detailed analysis, comparison with standard operators, and information-theoretic interpretation.

The Yat-product is intrinsically non-linear and self-regulating, and we prove that networks built from it are universal approximators (see Section 3 and Appendix G.6).

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

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

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 Yat-product, integrating its geometric sensitivity into convolutional architectures.

Building upon the Yat-product, we propose Neural-Matter Networks (NMNs) and Convolutional NMNs (CNMNs). NMNs are designed to preserve input topology by leveraging the Yat-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 Yat-product, a novel, physics-grounded neural operator that unifies directional sensitivity with an inverse-square proximity measure, designed for geometrically faithful similarity assessment.

 doi:10.20944/preprints202506.2010.v2



2. The proposal of Neural-Matter Networks (NMNs), a new class of neural architectures based on the Yat-product, which inherently incorporate non-linearity and are designed to preserve input topology.
3. 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.

## 2. Theoretical Background

### *2.1. 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 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 [5].

#### 2.1.1. 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:

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n \tag{3}$$

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.

#### 2.1.2. 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:

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m] g[n - m] \tag{4}$$

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.

 doi:10.20944/preprints202506.2010.v2



- **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.

#### 2.1.3. 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 $\mathbf{B}$:

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{5}$$

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 [13,14].

#### 2.1.4. 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:

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \tag{6}$$

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 Yat-product introduced in Section 3.

 doi:10.20944/preprints202506.2010.v2



*2.2. 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 [3,4,21,22])**.** *Let $\sigma$ 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 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*

$$F(x) = \sum_{i=1}^{N} v_i \sigma(w_i^T x + b_i) \tag{7}$$

*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.

2.2.1. 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 $\sigma$:

$$y = \sigma(\langle \mathbf{w}, \mathbf{x} \rangle + b), \tag{8}$$

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 $\sigma$ is zero:

$$\{\mathbf{x} \in \mathbb{R}^d \mid \langle \mathbf{w}, \mathbf{x} \rangle + b = 0\}. \tag{9}$$

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 [5].

 doi:10.20944/preprints202506.2010.v2



2.2.2. 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 $\phi$:

$$\mathbf{h} = \phi(W\mathbf{x} + \mathbf{b}). \tag{10}$$

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 $\phi$ 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.

2.2.3. 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 $\phi$, often crucial for breaking linearity, can significantly alter the topological and geometric structure of the data representation, potentially leading to information loss [5]. 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) [5].

However, the subsequent application of a typical non-linear activation $\phi$ 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 neuronś 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.v2



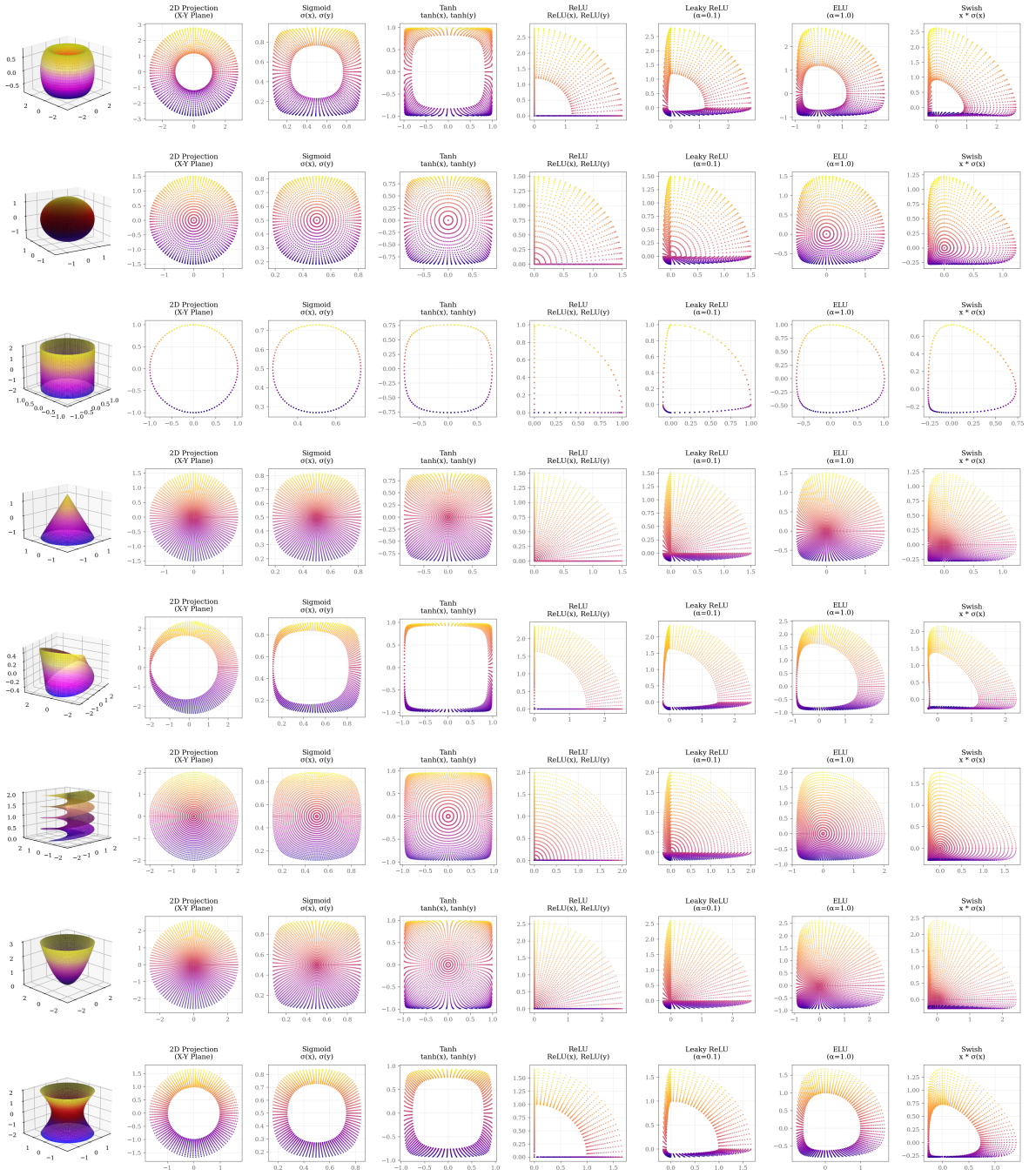



**Figure 1.** Illustration of how non-linear activation functions can distort the geometric structure of the input data manifold, leading to potential information loss. The original manifold (left) is transformed into a distorted representation after applying a non-linear activation functions.

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 $\phi$. 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 Yat-product.

 doi:10.20944/preprints202506.2010.v2



## 3. Methodology: A Framework for Geometry-Aware Computation

### *3.1. The Yat-Product: A Unified Operator for Alignment and Proximity*

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

The Yat-product is formally defined as $Yat(\mathbf{w}, \mathbf{x}) = \frac{(\mathbf{w}^\top \mathbf{x})^2}{\|\mathbf{w}-\mathbf{x}\|^2+\epsilon}$ [19,20]. 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 Yat-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.

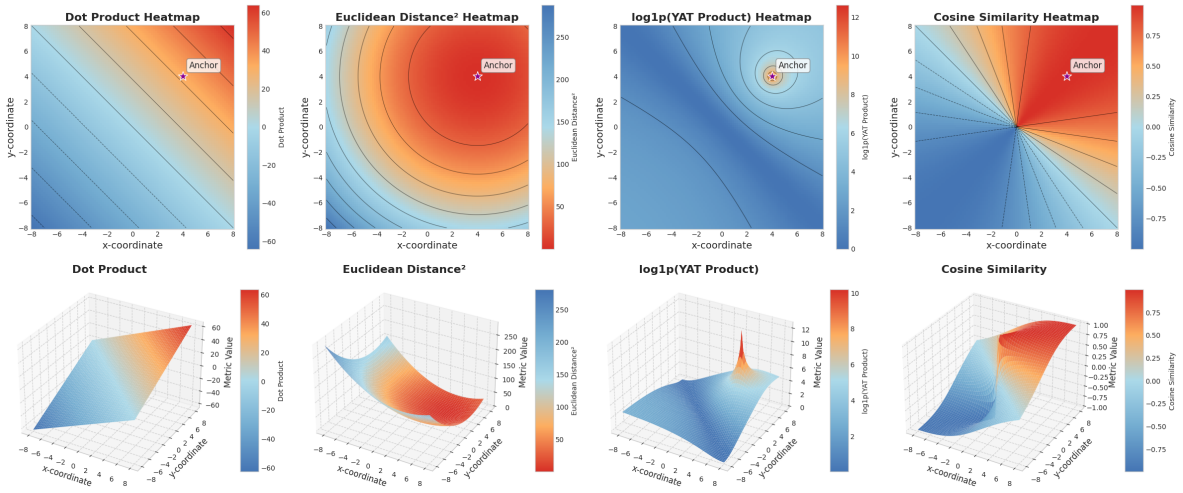



**Figure 2.** Visualization of the Yat-product's vector field in 2D and 3D spaces. The heatmaps illustrate how the Yat-product, unlike traditional similarity measures, creates a potential well around the weight vector $\mathbf{w}$, reflecting both alignment and proximity. This figure embodies the paper's philosophy of unifying geometric alignment and spatial closeness within a single neural operator, inspired by physical interaction fields. The resulting landscape demonstrates how the Yat-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 Yat-product creates a potential well around the weight vector $\mathbf{w}$, reflecting both alignment and proximity.

### *3.2. Comparison to Standard Similarity and Distance Metrics*

To further appreciate the unique characteristics of the Yat-product, it is instructive to compare it with other common similarity or distance metrics [13,14,23,24]:

- **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.

 doi:10.20944/preprints202506.2010.v2



- **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 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 $\mathbf{w}$.
- **Yat-Product ($\mathcal{K}_Yat(\mathbf{w}, \mathbf{x}) = \frac{(\mathbf{w}^\top \mathbf{x})^2}{\|\mathbf{w}-\mathbf{x}\|^2+\epsilon}$):** The Yat-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 Yat-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 Yat-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.

Beyond its geometric interpretation, the Yat-product has a profound connection to information theory when its arguments are probability distributions [25–27] (see Appendix G.7 for formal results). In this context, it can be viewed as a signal-to-noise ratio, where the "signal" $(\mathbf{p} \cdot \mathbf{q})^2$ measures distributional alignment and the "noise" $\|\mathbf{p} - \mathbf{q}\|^2$ quantifies their dissimilarity.

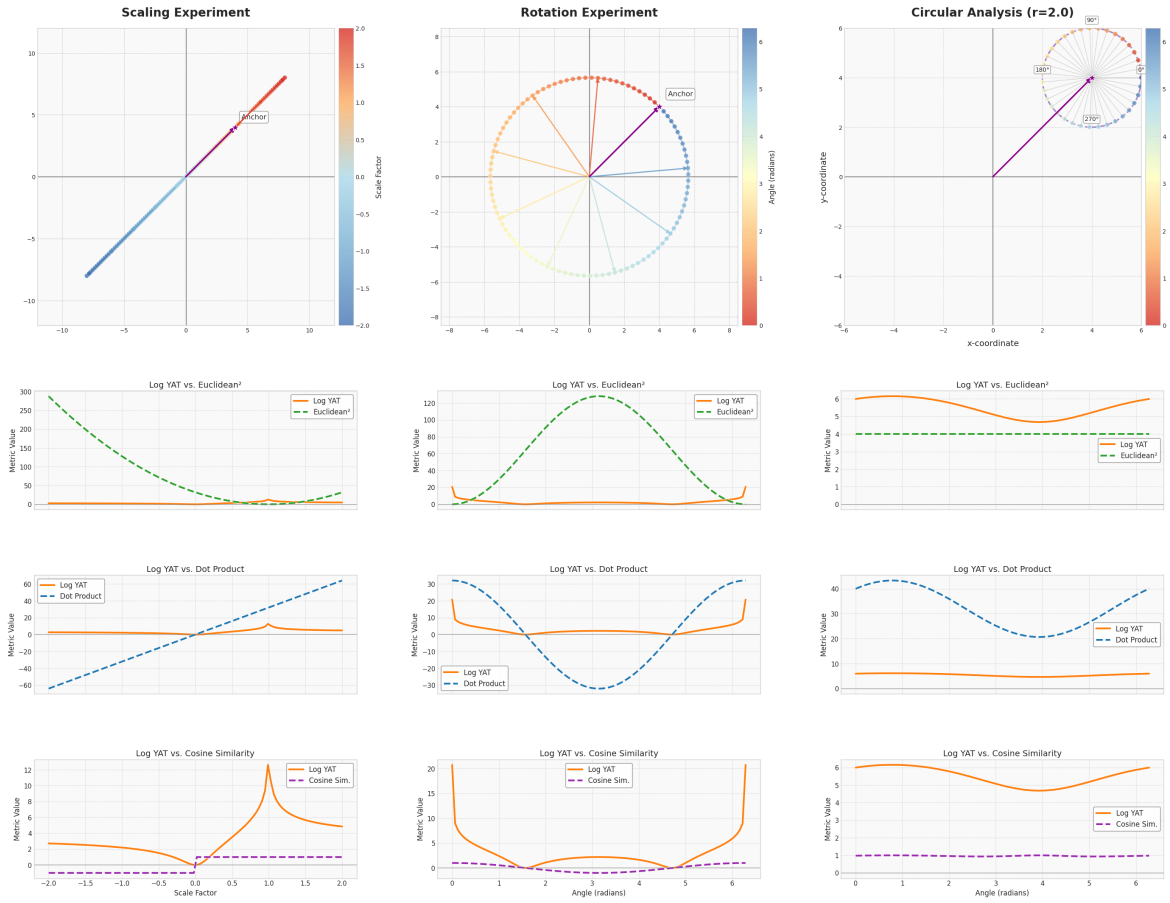



**Figure 3.** Comparison of the Yat-product with other metrics (dot product, cosine similarity, and Euclidean distance) in three distinct settings: (a) scaling vectors linearly by a factor $s$, (b) rotating the anchor vector, and (c) varying the distance of vectors around the anchor. The Yat-product's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

The Yat-product's ability to discern between aligned vectors at varying distances, as well as its sensitivity to the angle between vectors, is illustrated in Figure 2 and Figure 3. The vector field

 doi:10.20944/preprints202506.2010.v2



generated by the Yat-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 Yat-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.

### *3.3. Design Philosophy: Intrinsic Non-Linearity and Self-Regulation*

A central hypothesis underpinning our methodological choices is that the Yat-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 Yat-product (Section 1) then transcends simple similarity assessment; it functions as a sophisticated interaction potential, $\mathcal{K}_Yat(\mathbf{w}, \mathbf{x}) = \frac{(\mathbf{w}^\top \mathbf{x})^2}{\|\mathbf{w}-\mathbf{x}\|^2+\epsilon}$, 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 pairs apart in the embedding space, our Yat-product seeks a more nuanced relationship: 'positive' interactions (high Yat-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 Yat-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 Yat-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 Yat-product, coupled with the self-regulating properties suggested by its formulation (and formally proven in Appendix G.3), 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.

 doi:10.20944/preprints202506.2010.v2



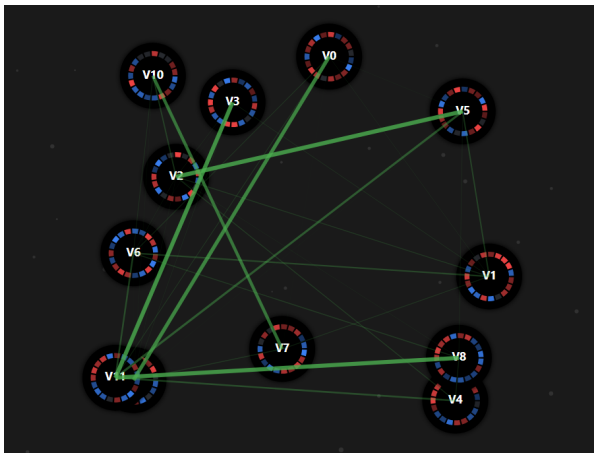



**Figure 4.** The Vectoverse: Conceptualizing neural computation where weight vectors ($\mathbf{w}$) and input vectors ($\mathbf{x}$) are akin to fundamental particles (vectoms). The interaction force between them is quantified by the Yat-product, which measures their alignment and proximity, defining a field of influence.

### *3.4. Core Building Blocks*

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

#### 3.4.1. The Neural Matter Network (NMN) Layer

The first and simplest application of the Yat-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 $\mathcal{K}Yat$-kernel (derived from the Yat-product, see Section 3.1) as the primary interaction mechanism, instead of the conventional linear projection ($\langle \mathbf{w}, \mathbf{x} \rangle$).

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:

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

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.
- $\mathcal{K}Yat(\mathbf{w}_i, \mathbf{x})$ represents the Yat-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.

A key theoretical guarantee for NMNs is their capacity for universal function approximation [3,4,21,22,28,29]. This is significant because, unlike traditional neural networks that depend on separate, often heuristically chosen, activation functions (e.g., ReLU, sigmoid) to introduce non-linearity, the approximation power of NMNs is an intrinsic property of the $\mathcal{K}Yat$-kernel itself. This finding validates the Yat-product as a self-contained computational primitive powerful enough to form the basis of expressive neural architectures, distinguishing NMNs from classical MLP-based designs and supporting our core hypothesis that effective, geometry-aware computation is possible without separate activation functions [10–12].

 doi:10.20944/preprints202506.2010.v2



#### 3.4.2. Convolutional Neural-Matter Networks (CNMNs) and the Yat-Convolution Layer

To extend the principles of Neural-Matter Networks (NMNs) (Section 3.4.1) and the Yat-product (Section 1) to spatially structured data like images, we introduce the Yat-Convolution (Yat-Conv) layer. This layer adapts the Yat-product to operate on local receptive fields, analogous to standard convolutional layers. The Yat-Conv operation is defined as:

$$(\text{Yat-Conv}(K, I))_{i,j} = Yat^*(K, I_{i,j}) = \frac{\langle K, I_{i,j} \rangle^2}{\|K - I_{i,j}\|^2 + \epsilon} \tag{11}$$

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.

#### 3.4.3. The Yat-Attention Mechanism

To extend the Yat-product's application to sequence modeling, we propose the Yat-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 Yat-product (Section 3.1). Given Query ($Q$), Key ($K$), and Value ($V$) matrices, Yat-Attention is computed as:

$$\text{Yat-Attention}(Q, K, V) = \text{softmax}\Big(s \cdot (QYatK^T)\Big)V \tag{12}$$

where the operation $QYatK^T$ signifies applying the Yat-product element-wise between query and key vectors (e.g., the $(i, j)$-th element is $Yat(q_i, k_j)$), and $s$ is a scaling factor.

### *3.5. 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 3.3. Key architectural distinctions driven by this philosophy include:

- **Fundamental Operator Replacement**: The standard dot product is replaced by the Yat-product. This is manifested as Yat-Convolution (Equation 11) in convolutional networks and Yat-Attention (Equation 12) in transformer-based models.
- **Feed-Forward Networks (FFNs)**: The FFNs within are constructed using NMN layers (Section 3.4.1) 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.

Additionally, in all NMN-based architectures, we use a scaling factor $s = \left(\frac{n}{\log(1+n)}\right)^{\alpha}$, where $n$ is the number of NMN units and $\alpha$ is a learnable parameter. This scaling is designed to adaptively control the overall magnitude of the layer outputs as a function of network width.

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 Yat-product is not limited to the architectures presented and holds potential for enhancing other neural network paradigms.

#### 3.5.1. Convolutional NMNs:

AetherResNet is a Convolutional Neural-Matter Network (CNMN) built by replacing all standard convolutions in a ResNet18 architecture with the Yat-Conv layers. Building upon the Yat-Conv layer, CNMNs adapt conventional convolutional architectures by employing the Yat-Conv layer as the primary feature extraction mechanism. The core idea is to leverage the geometric sensitivity and inherent non-linearity of the Yat-product within deep convolutional frameworks. Consistent with the philosophy of Section 3.3, AetherResNet omits Batch Normalization and activation functions [9,30–32].

 doi:10.20944/preprints202506.2010.v2



The design relies on the hypothesis that the Yat-product itself provides sufficient non-linearity and a degree of self-regulation.

In each CNMN residual block, we use a YatConv layer (with input dimension $n$ and output dimension $m$) followed by a linear Conv layer (with input and output dimension $m$), without any activation functions or normalization layers (see Fig. A15).

#### 3.5.2. YatFormer: AetherGPT

AetherGPT is a YatFormer model that uses Yat-Attention (from Section 3.4.3) for sequence interaction and NMN layers (from Section 3.4.1) in its feed-forward blocks. Building upon the Yat-Attention mechanism, which forms its cornerstone, we introduce YatFormer, a family of transformer-based 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 3.3, it omits standard normalization and activation layers.

In YatFormer/AetherGPT, we remove the projection layer after the attention mechanism, as the dot product between the attention map and the value matrix $V$ already serves as a linear projection (see Fig. A16). Furthermore, in the MLP block, we do not multiply the first layer's width by 4 as in standard transformers. Instead, we use a YatNMN layer with input and output dimensions equal to the embedding dimension, followed by a linear layer, also with input and output dimensions equal to the embedding dimension (see Fig. A17).

### *3.6. Output Processing for Non-Negative Scores*

The Yat-product and its derivatives, such as the $\mathcal{K}_Yat$-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).

Squashing functions for non-negative scores can be broadly categorized into two types:

- **Competitive (Vector-Normalizing) Functions:** These functions normalize a set of scores collectively, producing a distribution over the vector. Each output depends on the values of all dimensions, allowing for competitive interactions among them. This is useful for attention mechanisms or probability assignments where the sum of outputs is meaningful.
- **Individualistic (Per-Dimension) Functions:** These functions squash each score independently, without reference to other values in the vector. Each output depends only on its corresponding input, making them suitable for bounding or interpreting individual activations.

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. Additionally, softmax can suffer from numerical instability for large input values due to the exponentials.

Given these limitations and the non-negative nature of Yat-product scores, we consider alternative squashing functions more suited to this domain:

 doi:10.20944/preprints202506.2010.v2



- **softermax (Competitive):** 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:

$$\text{softermax}_n(x_k, \{x_i\}) = \frac{x_k^n}{\epsilon + \sum_i x_i^n} \tag{13}$$

  Unlike softmax, softermax does not use exponentials, which avoids numerical instability for large inputs and provides a more direct, interpretable translation of the underlying scores into a normalized distribution. 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 (Individualistic):** 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:

$$\text{soft-sigmoid}_n(x) = \frac{x^n}{1 + x^n} \tag{14}$$

  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 (Individualistic):** 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:

$$\text{soft-tanh}_n(x) = 2 \cdot (\text{soft-sigmoid}_n(x) - \tfrac{1}{2}) = \frac{x^n - 1}{1 + x^n} \tag{15}$$

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

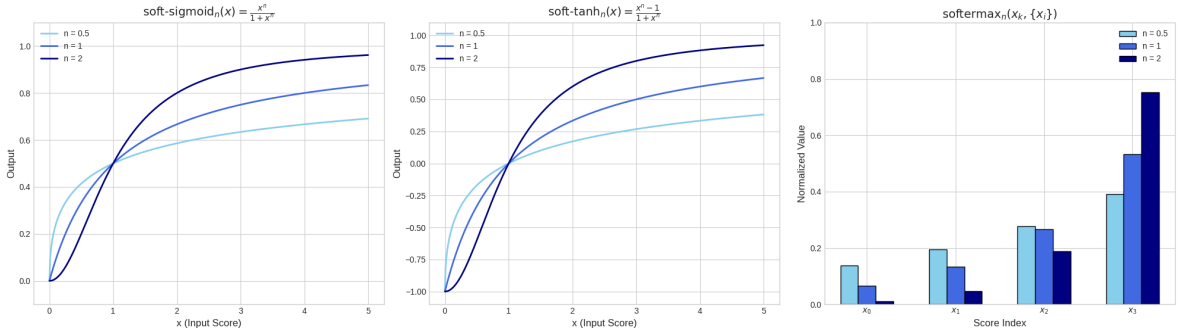



**Figure 5.** Visualization of the softermax, soft-sigmoid, and soft-tanh functions. These functions are designed to handle non-negative inputs from the Yat-product and its derivatives, providing appropriate squashing mechanisms that maintain sensitivity across the range of non-negative inputs.

These functions are particularly well-suited for the outputs of Yat-product-based computations, as they maintain sensitivity across the range of non-negative inputs while avoiding the pitfalls of standard activation functions [7–9].

The main role of these squashing functions can be categorized into two main categories:

- **Collective Communication and Space Splitting:** The softermax function allows for a comparative analysis of scores, reflecting their orthogonality and spatial proximity to an input vector. A higher score indicates that a vector is more aligned and closer to the input, while a lower score suggests greater orthogonality. This facilitates a competitive interaction where vectors vie for influence based on their geometric relationship with the input. The power parameter $n$, analogous to the temperature in softmax, controls the sharpness of the gravitational potential well's slope.
- **Individual Score Squashing:** The soft-sigmoid and soft-tanh functions are used to squash individual non-negative scores into a bounded range, typically $[0, 1)$ for soft-sigmoid and $[-1, 1)$

 doi:10.20944/preprints202506.2010.v2



for soft-tanh. They are particularly useful when the output needs to be interpreted as a probability or when a bounded response is required, as each score is processed independently of the others. The power parameter controls the steepness of the function, while the minimum value can be interpreted as an orthogonality score.

### *3.7. Mathematical Guarantees of the Yat-Product and NMNs*

The Yat-product and the resulting Neural-Matter Networks (NMNs) are supported by several key mathematical properties, each formally proven in the appendices:

- **Mercer Kernel Property:** The Yat-product is a symmetric, positive semi-definite Mercer kernel, enabling its use in kernel-based learning methods (see Appendix G.2).
- **Universal Approximation:** NMNs with Yat-product activations can approximate any continuous function on a compact set, establishing their expressive power (see Appendix G.6).
- **Self-Regulation:** The output of a Yat-product neuron is naturally bounded and converges to a finite value as input magnitude increases, ensuring stable activations (see Appendix G.3).
- **Stable Gradient:** The gradient of the Yat-product with respect to its input vanishes for distant inputs, preventing large, destabilizing updates from outliers (see Appendix G.5).
- **Information-Theoretic Duality:** The Yat-product unifies geometric and information-theoretic notions of similarity and orthogonality, with formal theorems connecting it to KL divergence and cross-entropy (see Appendix G.7).

## 4. Results and Discussion

The Yat-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 Yat-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 Yat-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 $\mathcal{K}_Yat(\mathbf{w},\mathbf{x}) = 0$.

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

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

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

Thus, the Yat-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. We have formally proven that the Yat-product is a valid Mercer kernel in Appendix G.2 [33–38].

To understand its behavior during learning, we analyze its gradient. A key property for stable training is that the gradient with respect to the input, $\nabla_\mathbf{x}\mathcal{K}_{\text{Yat}}$, diminishes as the input $\mathbf{x}$ moves far from the weight vector $\mathbf{w}$. This ensures that distant outliers do not cause large, destabilizing updates. We have formally proven this property in Appendix G.5, demonstrating that $\lim_{\|\mathbf{x}\|\to\infty} \|\nabla_\mathbf{x}\mathcal{K}_{\text{Yat}}(\mathbf{w},\mathbf{x})\| = 0$.

The presence of $\epsilon$ in the denominator ensures that the derivative remains well-defined, avoiding division by zero and 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 Yat-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

 doi:10.20944/preprints202506.2010.v2



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 Yat-product quantifying the degree of match in a nuanced, non-linear fashion.

The gradient of the Yat-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 Yat-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.

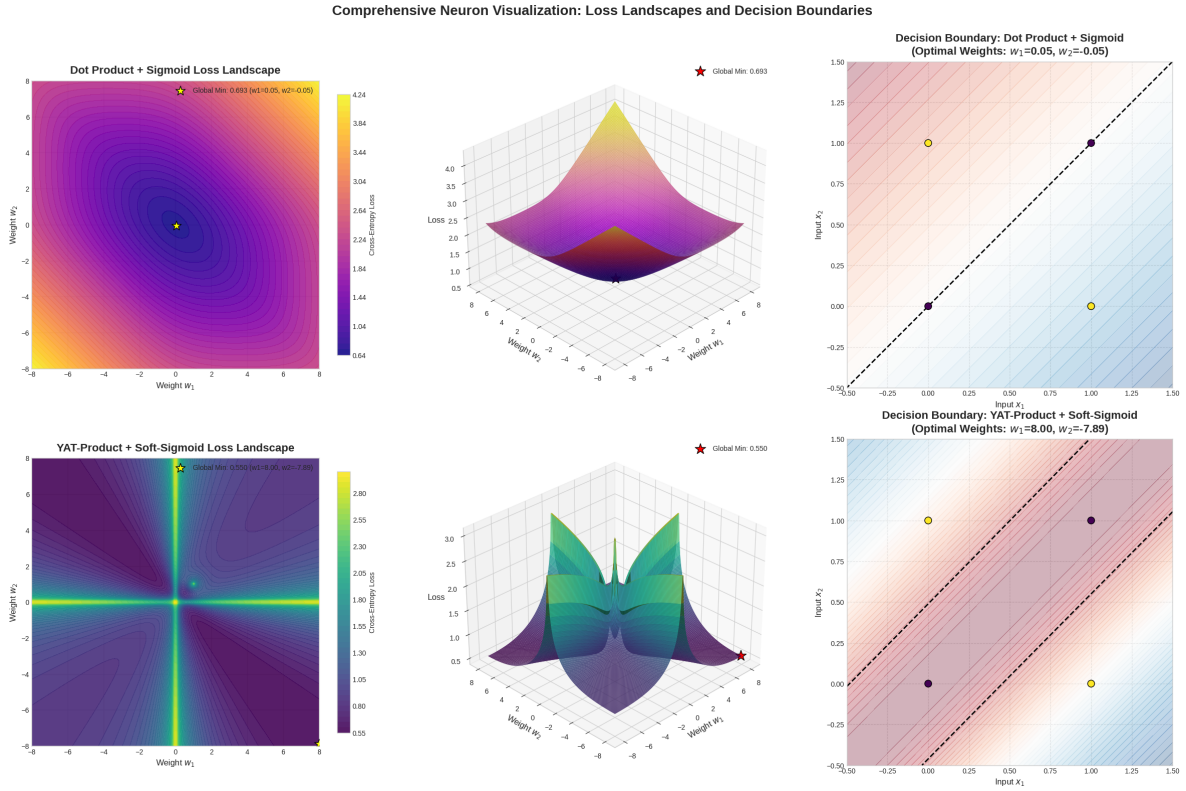



**Figure 6.** Comparison of the loss landscape for a simple dot product neuron and a Yat-product neuron. The dot product neuron has a stable minimum at zero output which doesn't solve the xor problem and cause the neuron death, while the Yat-product neuron resides in a valley of orthogonality, allowing it to avoid dead zones and actively shape the decision boundary. This illustrates the Yat-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 Yat-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.

 doi:10.20944/preprints202506.2010.v2



### 4.1. Your Neuron is a secret Vortex

We begin by analyzing the fundamental learning dynamics that emerge in both conventional and our proposed architectures. In artificial intelligence, competitive learning manifests in various forms, whether through linear classification using dot products or clustering using Euclidean distances. Both approaches involve partitioning the feature space between neurons, which can be conceptualized as prototype learning where each neuron claims a territorial "field" in the representation space.

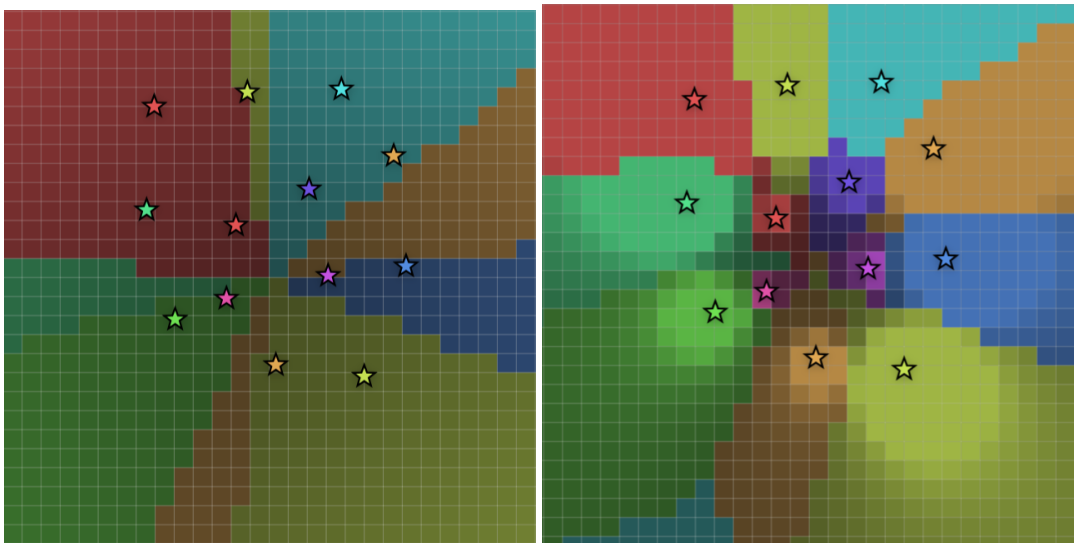


**Figure 7.** Comparison of decision boundaries formed by a conventional linear model (left) and our proposed Yat-product method (right). The conventional model's prototypes grow unbounded, while our method learns more representative prototypes that better capture class distributions.

In this experiment, we analyze the learning dynamics of a linear model on a synthetic dataset, comparing the formation of decision boundaries by conventional neurons employing standard dot products with those generated by our proposed Yat-product method.

In a conventional linear model, the logit for each class $i$ is computed as:

$$z_i = \mathbf{w}_i^T \mathbf{x} \tag{16}$$

where $\mathbf{w}_i$ is the weight vector (prototype) for class $i$, and $\mathbf{x}$ is the input vector. The softmax function then normalizes these logits into probabilities:

$$p_i = \frac{\exp(z_i)}{\sum_{j=1}^{C} \exp(z_j)} \tag{17}$$

The decision boundary between any two classes $i$ and $j$ forms a linear hyperplane defined by:

$$(\mathbf{w}_i - \mathbf{w}_j)^T \mathbf{x} = 0 \tag{18}$$

During training via gradient descent, each prototype $\mathbf{w}_i$ is updated to maximize its alignment with the data distributions of its assigned class. This optimization process often leads to an unbounded increase in prototype magnitudes, as $\|\mathbf{w}_i\| \to \infty$ directly amplifies the logit $z_i$, thereby increasing the model's confidence. However, the decision boundaries themselves remain linear hyperplanes, creating rigid geometric separations in the feature space.

 doi:10.20944/preprints202506.2010.v2



In contrast, the non-linear Yat-product allows neurons to learn more representative prototypes for each class, leading to the formation of more nuanced decision boundaries. For the Yat-product, the response of neuron $i$ to input $\mathbf{x}$ is given by:

$$z_i = \text{Yat}(\mathbf{w}_i, \mathbf{x}) = \frac{\langle \mathbf{w}_i, \mathbf{x} \rangle^2}{\|\mathbf{w}_i - \mathbf{x}\|^2 + \epsilon} \tag{19}$$

This formulation embodies the signal-to-noise ratio interpretation established in our theoretical framework (Appendix G.7), where the squared dot product $\langle \mathbf{w}_i, \mathbf{x} \rangle^2$ represents the "signal" of distributional alignment, and $\|\mathbf{w}_i - \mathbf{x}\|^2$ quantifies the "noise" of dissimilarity. The Yat-product thus provides a principled geometric measure that balances similarity and proximity in a theoretically grounded manner.

Similarly to conventional neurons, the Yat-product outputs are normalized using the softmax function:

$$p_i = \frac{\exp(z_i)}{\sum_{j=1}^{C} \exp(z_j)} = \frac{\exp\left(\frac{\langle \mathbf{w}_i, \mathbf{x} \rangle^2}{\|\mathbf{w}_i - \mathbf{x}\|^2 + \epsilon}\right)}{\sum_{j=1}^{C} \exp\left(\frac{\langle \mathbf{w}_j, \mathbf{x} \rangle^2}{\|\mathbf{w}_j - \mathbf{x}\|^2 + \epsilon}\right)} \tag{20}$$

This softmax normalization serves a crucial role in the competitive dynamics of Yat-product neurons. The softmax function acts as a transformation that maps from the real-valued Yat-product responses $z_i \in \mathbb{R}$ to a delta distribution $\delta_i$ in probability space. This softmax distribution over Yat-product scores can be interpreted as the *posterior responsibility* of each prototype (neuron) for the input, drawing a direct connection to Gaussian Mixture Models (GMMs) and expectation-maximization frameworks.

The softmax can also be viewed as computing a categorical distribution proportional to exponentiated log-likelihoods, which in this case derive from a geometric Yat-product similarity rather than traditional probabilistic assumptions. This bridges the gap between probabilistic views (such as EM algorithms and classification) and our geometric formulation, providing a principled foundation for the competitive dynamics.

As training progresses and the differences between Yat-product responses become more pronounced, the softmax transformation approaches a delta distribution, where the winning neuron (with the highest Yat-product response) approaches probability 1 while all others approach 0. This winner-take-all mechanism enables competitive learning dynamics where each neuron competes to "take over" regions of the input space based on their vortex-like attraction fields.

The decision boundary between two neurons with prototypes $\mathbf{w}_i$ and $\mathbf{w}_j$ is defined by the condition where their responses are equal:

$$\text{Yat}(\mathbf{w}_i, \mathbf{x}) = \text{Yat}(\mathbf{w}_j, \mathbf{x}) \tag{21}$$

Expanding this condition:

$$\frac{\langle \mathbf{w}_i, \mathbf{x} \rangle^2}{\|\mathbf{w}_i - \mathbf{x}\|^2 + \epsilon} = \frac{\langle \mathbf{w}_j, \mathbf{x} \rangle^2}{\|\mathbf{w}_j - \mathbf{x}\|^2 + \epsilon} \tag{22}$$

Cross-multiplying and rearranging:

$$\langle \mathbf{w}_i, \mathbf{x} \rangle^2 (\|\mathbf{w}_j - \mathbf{x}\|^2 + \epsilon) = \langle \mathbf{w}_j, \mathbf{x} \rangle^2 (\|\mathbf{w}_i - \mathbf{x}\|^2 + \epsilon) \tag{23}$$

This equation defines a complex, non-linear decision boundary that depends on both the alignment (through the squared dot products) and the proximity (through the squared distances) between the input and each prototype. Unlike the linear hyperplane formed by conventional dot product neurons, the Yat-product creates what we term a *vortex phenomenon* or gravitational potential well around each prototype.

 doi:10.20944/preprints202506.2010.v2



The space partitioning behavior of the Yat-product exhibits several key properties that create this vortex-like effect:

- **Gravitational Attraction**: The inverse-square relationship in the denominator creates a field where points are more strongly attracted to nearby prototypes, similar to gravitational fields in physics.
- **Alignment Amplification**: The squared dot product in the numerator creates a strong response for well-aligned inputs, while the vortex effect pulls data points toward the prototype center.
- **Bounded Potential Wells**: Each neuron creates a localized potential well with bounded depth, preventing the unbounded growth seen in linear neurons. This boundedness is theoretically guaranteed by the Minimal and Maximal Similarity Characterizations (Theorems A7 and A8), which establish that $0 \leq \text{Yat}(\mathbf{w}_i, \mathbf{x}) \leq \infty$ with well-defined extremal conditions.
- **Curved Decision Boundaries**: The resulting decision boundaries are non-linear curves that wrap around the data distribution, creating vortex-like territorial regions for each neuron.

This vortex phenomenon allows each Yat-product neuron to create a territorial "field" in the representation space, where data points are pulled toward the dominant prototype based on both similarity and proximity metrics. The field each neuron occupies can indeed be considered a vortex, where the strength of attraction follows an inverse-square law, creating more natural and geometrically faithful decision boundaries.

The combination of the Yat-product's vortex-like attraction and the softmax's competitive normalization creates a powerful space partitioning mechanism. Each neuron's vortex field competes with others through the softmax transformation, and the neuron with the strongest local attraction (highest Yat-product response) wins that region. Over time, this leads to a natural tessellation of the input space, where each neuron's territory is defined by the regions where its vortex field dominates. The softmax transformation $\mathbb{R}^C \to \Delta^{C-1}$ (where $\Delta^{C-1}$ is the $(C-1)$-dimensional probability simplex) ensures that these territorial boundaries are sharp and well-defined, transforming the continuous real-valued responses into discrete delta distributions that clearly assign each input to its dominant neuron.

**Orthogonality and Competitive Dynamics:** The competitive learning behavior observed in practice is theoretically grounded in our Orthogonality-Entropy Connection. When two prototypes $\mathbf{w}_i$ and $\mathbf{w}_j$ develop disjoint support regions, they become Euclidean orthogonal ($\mathbf{w}_i \perp \mathbf{w}_j$), which corresponds to:

$$\text{Yat}(\mathbf{w}_i, \mathbf{w}_j) = 0 \quad \text{and} \quad H(\mathbf{w}_i, \mathbf{w}_j) = \infty \tag{24}$$

This geometric-probabilistic duality explains why neurons naturally develop specialized, non-overlapping representations during competitive learning. The infinite cross-entropy between orthogonal prototypes creates strong pressure for territorial separation, preventing the collapse to identical representations that can plague conventional competitive learning systems.

These prototypes are optimized to maximize parallelism and minimize distance to all points within their class distribution. When minimizing distance becomes challenging, the properties of the Yat-product enable the prototype to exist in a superposition state, prioritizing the maximization of parallelism over strict distance minimization.

### *4.2. Do you even MNIST bro?*

Having established the theoretical foundation of the vortex phenomenon in Section 4.1, we now validate these insights on the canonical MNIST dataset. This experiment serves as a bridge between our geometric theory and practical applications, demonstrating how the vortex-like territorial dynamics translate into improved prototype learning on real data.

In our MNIST experiments, the network consists of $C = 10$ neurons, each corresponding to one of the digit classes (0–9). Each neuron's prototype is represented as a vector $\mathbf{w}_i \in \mathbb{R}^{784}$, where $i = 1, \ldots, 10$. The input images $\mathbf{x} \in \mathbb{R}^{784}$ are obtained by flattening the original $28 \times 28$ pixel images, so

 doi:10.20944/preprints202506.2010.v2



each neuron's prototype $\mathbf{w}_i$ has the same dimensionality as the input, i.e., $\mathbf{w}_i = (w_{i,1}, w_{i,2}, \ldots, w_{i,784})$. This structure allows each neuron to learn class-specific features in the full image space.

The MNIST dataset provides an ideal testbed for examining the vortex phenomenon because its 10-class structure allows clear visualization of how different neurons compete for territorial control in the feature space. We specifically investigate whether the bounded attraction fields and territorial partitioning predicted by our theory manifest as improved prototype quality and learning dynamics in practice.

**Experimental Design:** We train both conventional linear classifiers and our Yat-product networks on MNIST, analyzing three key aspects that directly relate to the vortex phenomenon:

1. **Prototype Evolution Dynamics**: How do prototypes evolve during training under different competitive mechanisms?
2. **Territorial Boundary Formation**: Do we observe the predicted non-linear decision boundaries and vortex-like attraction fields?
3. **Representational Quality**: How does the theoretical prediction of bounded, concentrated prototypes translate to interpretability?

The prototype evolution during training reveals the fundamental differences between conventional unbounded growth and our bounded vortex dynamics. Figure 9 shows the final learned prototypes, providing striking empirical confirmation of our vortex theory. The conventional linear model produces prototypes that exhibit the unbounded growth predicted by our analysis—these prototypes become increasingly diffuse and less interpretable as they grow to maximize margin separation. The resulting digit representations are blurry and lack the fine-grained features necessary for robust classification.

In stark contrast, the Yat-product method produces prototypes that perfectly exemplify the bounded vortex fields described in our theory. Each digit prototype exhibits:

- **Localized Concentration**: Sharp, well-defined features that correspond to the bounded potential wells predicted by our Minimal and Maximal Similarity Characterizations (Theorems A7 and A8)
- **Class-Specific Territorial Structure**: Each prototype captures unique digit characteristics, reflecting the competitive territorial dynamics where each neuron's vortex field dominates specific regions of the input space
- **Geometric Fidelity**: The prototypes maintain geometric coherence with actual digit structure, confirming that the signal-to-noise ratio optimization preserves meaningful visual patterns

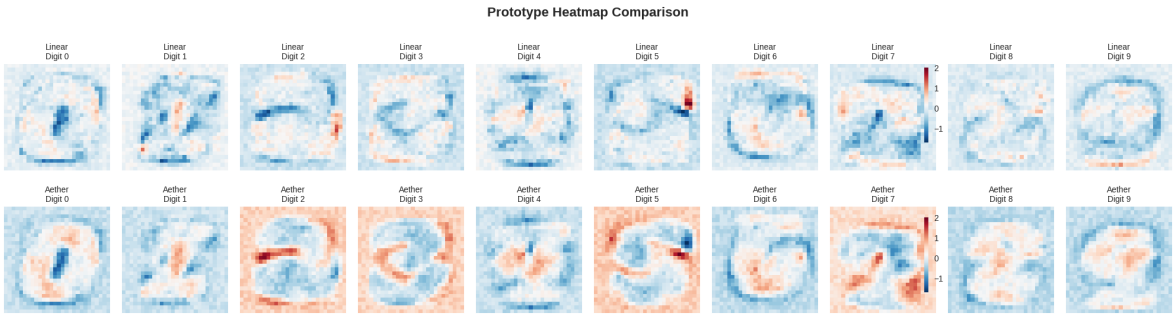



**Figure 8.** Prototypes learned by the conventional linear model (top) and our proposed Yat-product method (bottom) on the MNIST dataset. The prototypes from our method are more distinct and representative of the digit classes, capturing finer details and class-specific characteristics.

**Superposition and Prototype Inversion:** A unique property of the Yat-product neuron is its ability to exist in a superposition state, which can be empirically demonstrated by inverting the learned prototype. Specifically, if $\mathbf{w}$ is a learned prototype, we consider the effect of replacing $\mathbf{w}$ with $-\mathbf{w}$ (i.e.,

 doi:10.20944/preprints202506.2010.v2



multiplying by $-1$) at test time, without any retraining. For a conventional dot product neuron, this operation flips the sign of the logit:

$$z = \mathbf{w}^T\mathbf{x} \quad \longrightarrow \quad z' = (-\mathbf{w})^T\mathbf{x} = -z \tag{25}$$

This sign flip causes the softmax output to assign high probability to the incorrect class, resulting in a dramatic drop in accuracy (from 93% to nearly 0% in our MNIST experiments).

In contrast, for the Yat-product neuron, the response is:

$$z = \text{Yat}(\mathbf{w}, \mathbf{x}) = \frac{(\mathbf{w}^T\mathbf{x})^2}{\|\mathbf{w} - \mathbf{x}\|^2 + \epsilon} \tag{26}$$

Multiplying $\mathbf{w}$ by $-1$ leaves the numerator unchanged, since $(-\mathbf{w})^T\mathbf{x} = -\mathbf{w}^T\mathbf{x}$ and $(-\mathbf{w}^T\mathbf{x})^2 = (\mathbf{w}^T\mathbf{x})^2$. The denominator is also unchanged, as $\| - \mathbf{w} - \mathbf{x}\|^2 = \|\mathbf{w} + \mathbf{x}\|^2$, which is symmetric with respect to the data distribution. As a result, the Yat-product neuron's accuracy remains nearly unchanged (dropping only slightly from 92% to 89%), demonstrating its robustness to prototype inversion and its ability to represent solutions in a superposition state.

This property allows the Yat-product neuron to yield two valid solutions to the same dataset without retraining, a phenomenon not observed in conventional dot product neurons. The table below summarizes the empirical results:

**Table 1.** Test accuracy on MNIST before and after prototype inversion ($\mathbf{w} \to -\mathbf{w}$) for dot product and Yat-product (yat) neurons.

| Neuron Type | Original Prototype | Inverted Prototype ($\mathbf{w} \to -\mathbf{w}$) |
|---|---|---|
| Dot Product | 91.88% | ≈0.01% |
| Yat-Product (Yat) | 92.18% | 87.87% |

### *4.3. Aether-GPT2: The Last Unexplainable Language Model*

To demonstrate the versatility of our approach beyond vision tasks, we implement Aether-GPT2, incorporating the Yat-product architecture into the GPT2 framework for language modeling. We compare the perplexity scores between our Aether-GPT2 and the standard GPT2 architecture across multiple text corpora.

**Table 2.** Final validation loss comparison between GPT2 and Aether-GPT2 on 600m tokens of Fineweb.

| Dataset | GPT2 | Aether-GPT2 |
|---|---|---|
| Fineweb | 2.69 | 2.83 |

The results in Table 2 demonstrate that Aether-GPT2 achieves a validation loss competitive with the standard GPT-2 baseline. While the loss is marginally higher, it is critical to note that Aether-GPT2 attains this performance despite its simplified design, which entirely omits dedicated activation functions and normalization layers. This outcome highlights a promising trade-off between raw performance and architectural simplicity, efficiency, and the inherent interpretability afforded by the Yat-product. These findings establish Aether-GPT2 as a successful proof-of-concept, suggesting that the Yat-product can serve as a viable alternative to conventional neural network components.

 doi:10.20944/preprints202506.2010.v2



**Table 3.** Aether-GPT2 Experiment Card

| Parameter | Value |
|---|---|
| Optimizer | Novograd |
| Learning Rate | 0.003 |
| Batch Size | 32 |
| Embedding Dimension | 768 |
| MLP Dimension | 768 (No x4) |
| Vocabulary Size | 50,257 |
| Number of Heads | 12 |
| Number of Blocks | 12 |

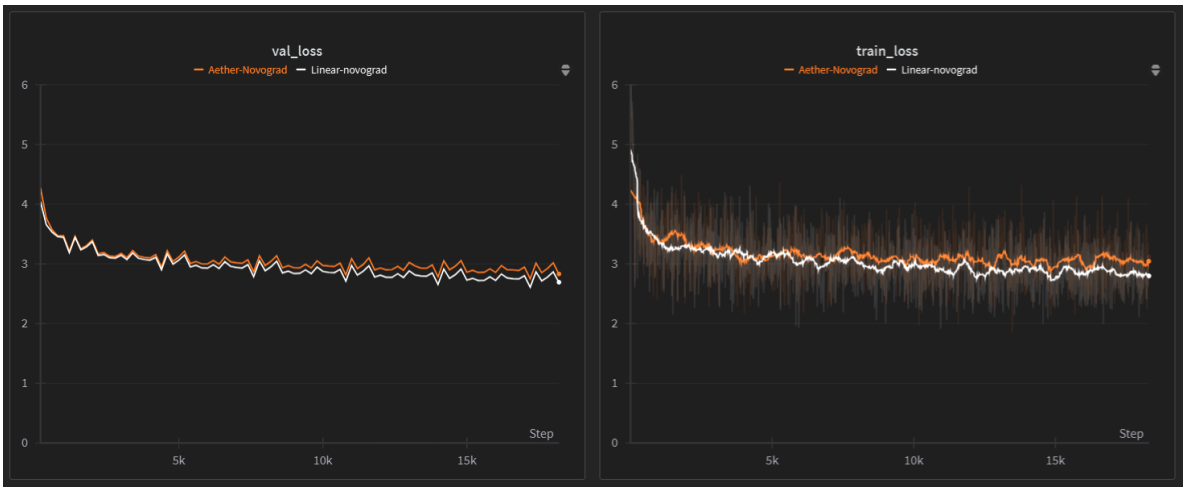



**Figure 9.** Loss Curve over the 600m tokens from fineweb trained on Kaggle TPU v3, Linear model is using standard GPT2 achitecture.

The results demonstrate that Aether-GPT2 consistently achieves close loss, indicating its ability to learn non-linearity without the need for activation functions.

The performance can be attributed to the Yat-product's ability to capture more nuanced relationships between tokens, allowing the model to better understand contextual dependencies and semantic similarities in natural language.

## 5. Related Work

### *5.1. 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 [15]. 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 [16], and Coulomb's Law defines the electrostatic force between charges [17]. 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 [18]. Similarly, thermal radiation intensity from a point source adheres to this law [39].

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 [40]. Photometry applies it to illumination engineering for lighting design [41]. In telecommunications, it underpins the free-space path loss of radio signals, as described by the Friis transmission equation

 doi:10.20944/preprints202506.2010.v2



[42], while radar systems experience an inverse fourth-power relationship due to the signal's two-way travel [43]. Seismology observes that seismic wave energy attenuates following this law [44], and in fluid dynamics, the velocity field from a point source in incompressible, irrotational flow also demonstrates an inverse-square dependence [45].

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 [23]), and the Jaccard index, a metric for set similarity [24], 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 [46], illustrating how these physical principles can offer insights into complex socio-economic phenomena.

### *5.2. 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) [37] laid the foundation for kernel-based learning, leveraging the kernel trick to implicitly map data into high-dimensional spaces. Schölkopf formalized kernel methods, expanding their applicability to various tasks [36].

Key advancements include Kernel PCA [47] for non-linear dimensionality reduction and Gaussian Processes [48] for probabilistic modeling. Applications like Spectral Clustering[49] and One-Class SVM [50] highlight the versatility of kernel methods.

To address computational challenges, techniques like the Nyström method [51] and Random Fourier Features [52] have improved scalability. Recent work, such as the Neural Tangent Kernel (NTK) [38], 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 [53], 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.

### *5.3. 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 [54]. 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 [55]. This has spurred developments like Vision Transformers (ViTs) [56], and models such as MLP-Mixer [57] and gMLP [58] which, while simplifying or eschewing self-attention, still rely on operations like matrix multiplication for feature mixing, demonstrating competitive performance, particularly in computer vision.

 doi:10.20944/preprints202506.2010.v2



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 [59,60], offering a different inductive bias compared to standard linear transformations followed by activation functions. Concurrently, Fourier Networks, such as FNet [61], 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 [7], sigmoid, tanh) often renders the internal decision-making processes of deep models opaque [27]. 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.

## 6. 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 Yat-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.

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

 doi:10.20944/preprints202506.2010.v2



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



## Appendix G Appendix

*Appendix G.1 Preliminary*

- **Cauchy-Schwarz Inequality** [62]: Used to bound the dot product and characterize equality conditions for identical vectors/distributions.
- **Properties of KL Divergence and Cross-Entropy** [63]: Used to show divergence for disjoint supports in probability distributions.
- **Mercer's Theorem** [33,64]: Establishes that a symmetric, positive semi-definite kernel defines a valid reproducing kernel Hilbert space (RKHS).
- **Schur Product Theorem** [62]: States that the Hadamard (element-wise) product of two positive semi-definite matrices is also positive semi-definite.
- **Bochner's Theorem** [65]: Characterizes translation-invariant kernels as positive definite if and only if their Fourier transform is non-negative.
- **Universal Approximation Theorem** [4,66]: Guarantees that neural networks with suitable activation functions or kernels can approximate any continuous function on a compact set.
- **Universality of Polynomial and Translation-Invariant Kernels** [35]: Used to argue that both the squared polynomial kernel and the translation-invariant kernel are universal.
- **Laplace Transform/Integral Representation** [67]: Used to express the inverse quadratic kernel as an integral over Gaussians, supporting the Bochner argument.

*Appendix G.2 Proof of Mercer's Condition for the Yat-product*

**Theorem A2.** *The Yat-product, defined as* $K(\mathbf{w}, \mathbf{x}) = \frac{\langle \mathbf{w}, \mathbf{x} \rangle^2}{\|\mathbf{w}-\mathbf{x}\|^2+\epsilon}$*, is a Mercer kernel.*

**Proof.** To prove that the Yat-product is a Mercer kernel, we must show that it is symmetric and positive semi-definite [33,64].

1. Symmetry

The kernel is defined as:

$$K(\mathbf{w}, \mathbf{x}) = \frac{\langle \mathbf{w}, \mathbf{x} \rangle^2}{\|\mathbf{w} - \mathbf{x}\|^2 + \epsilon} \tag{A27}$$

To check for symmetry, we evaluate $K(\mathbf{x}, \mathbf{w})$:

$$K(\mathbf{x}, \mathbf{w}) = \frac{\langle \mathbf{x}, \mathbf{w} \rangle^2}{\|\mathbf{x} - \mathbf{w}\|^2 + \epsilon} \tag{A28}$$

 doi:10.20944/preprints202506.2010.v2



Given that the dot product is commutative, $\langle \mathbf{w}, \mathbf{x} \rangle = \langle \mathbf{x}, \mathbf{w} \rangle$, and thus $\langle \mathbf{w}, \mathbf{x} \rangle^2 = \langle \mathbf{x}, \mathbf{w} \rangle^2$. Also, the squared Euclidean distance is symmetric: $\|\mathbf{w} - \mathbf{x}\|^2 = (\mathbf{w} - \mathbf{x}) \cdot (\mathbf{w} - \mathbf{x}) = \mathbf{w} \cdot \mathbf{w} - 2\mathbf{w} \cdot \mathbf{x} + \mathbf{x} \cdot \mathbf{x} = \|\mathbf{x} - \mathbf{w}\|^2$. Therefore, $K(\mathbf{w}, \mathbf{x}) = K(\mathbf{x}, \mathbf{w})$, and the kernel is symmetric.

2. Positive Semi-Definiteness

A kernel $K(\mathbf{w}, \mathbf{x})$ is positive semi-definite (PSD) if for any finite set of points $\{\mathbf{x}_i\}_{i=1}^N \subset \mathbb{R}^d$ and any coefficients $\{c_i\}_{i=1}^N \subset \mathbb{R}$, the following condition holds:

$$\sum_{i=1}^{N} \sum_{j=1}^{N} c_i c_j K(\mathbf{x}_i, \mathbf{x}_j) \geq 0$$

This is equivalent to stating that the Gram matrix $G$, where $G_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$, is positive semi-definite.

The proof of positive semi-definiteness is established by decomposing the kernel and leveraging the Schur product theorem [62] in conjunction with Bochner's theorem [65] for translation-invariant kernels. The proof proceeds by showing that the Yat-product is a product of two established Mercer kernels. Let the kernel be decomposed as:

$$K(\mathbf{w}, \mathbf{x}) = K_1(\mathbf{w}, \mathbf{x}) \cdot K_2(\mathbf{w}, \mathbf{x})$$

where:

- $K_1(\mathbf{w}, \mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle^2$
- $K_2(\mathbf{w}, \mathbf{x}) = \frac{1}{\|\mathbf{w} - \mathbf{x}\|^2 + \epsilon}$

The set of Mercer kernels is closed under pointwise product. If $K_1$ and $K_2$ are Mercer kernels, then for any set of points, their Gram matrices $G_1$ and $G_2$ are PSD. The Gram matrix for $K$ is the Hadamard (element-wise) product of $G_1$ and $G_2$. By the Schur product theorem [62], the Hadamard product of two PSD matrices is also PSD. Thus, if we can prove that $K_1$ and $K_2$ are Mercer kernels, their product $K$ must also be a Mercer kernel.

a) $K_1(\mathbf{w}, \mathbf{x})$ is a Mercer Kernel

The linear kernel $k_{\text{lin}}(\mathbf{w}, \mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle$ is a known Mercer kernel. As established, the set of Mercer kernels is closed under multiplication, so $K_1(\mathbf{w}, \mathbf{x}) = k_{\text{lin}}(\mathbf{w}, \mathbf{x}) \cdot k_{\text{lin}}(\mathbf{w}, \mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle^2$ is also a Mercer kernel.

b) $K_2(\mathbf{w}, \mathbf{x})$ is a Mercer Kernel

The kernel $K_2$ is a translation-invariant kernel, as it depends only on the difference $\mathbf{z} = \mathbf{w} - \mathbf{x}$. Let $k_2(\mathbf{z}) = (\|\mathbf{z}\|^2 + \epsilon)^{-1}$. By Bochner's theorem [65], a continuous translation-invariant kernel is positive definite if and only if its Fourier transform is non-negative.

The function $k_2(\mathbf{z})$ can be represented as an integral of a positive function (a Gaussian) using the identity $\frac{1}{A} = \int_0^\infty e^{-At} dt$ [67]:

$$\begin{aligned} k_2(\mathbf{z}) &= \frac{1}{\|\mathbf{z}\|^2 + \epsilon} \\ &= \int_0^\infty e^{-(\|\mathbf{z}\|^2 + \epsilon)t} dt \\ &= \int_0^\infty e^{-\epsilon t} e^{-t\|\mathbf{z}\|^2} dt \end{aligned}$$

The term $e^{-t\|\mathbf{z}\|^2}$ is proportional to the un-normalized density of a zero-mean Gaussian with variance $\sigma^2 = 1/(2t)$. The Fourier transform of a Gaussian is a Gaussian, which is always non-negative. Since $e^{-\epsilon t}$ is also non-negative for $t \geq 0$, the Fourier transform of $k_2(\mathbf{z})$ is an integral of non-negative functions, and is therefore itself non-negative. Thus, $K_2$ is a Mercer kernel.

Since both $K_1$ and $K_2$ are Mercer kernels, their product, $K(\mathbf{w}, \mathbf{x})$, is also a Mercer kernel.

 doi:10.20944/preprints202506.2010.v2



Thus, the Yat-product satisfies the conditions of symmetry and positive semi-definiteness, and is therefore a Mercer kernel. □

*Appendix G.3 Proof of Self-Regulation for the Yat-product*

**Theorem A3** (The Yat-Product is Naturally Self-Regulating)**.** *The output of a Yat-product neuron is bounded and converges to a finite value as the magnitude of the input vector approaches infinity.*

**Proof.** Let the Yat-product be defined as:

$$\text{Yat}(\mathbf{w}, \mathbf{x}) = \frac{\langle \mathbf{w}, \mathbf{x} \rangle^2}{\|\mathbf{w} - \mathbf{x}\|^2 + \epsilon} \tag{A29}$$

where $\mathbf{w}$ is a fixed weight vector and $\mathbf{x}$ is the input vector.

We want to analyze the behavior of $\text{Yat}(\mathbf{w}, \mathbf{x})$ as the magnitude of the input, $\|\mathbf{x}\|$, approaches infinity. We can represent any input vector $\mathbf{x}$ as $\mathbf{x} = k \cdot \mathbf{u}$, where $k = \|\mathbf{x}\|$ is its magnitude and $\mathbf{u}$ is a unit vector in the direction of $\mathbf{x}$. The limit can be expressed as $k \to \infty$.

Substituting $\mathbf{x} = k\mathbf{u}$ into the equation and using the properties of the dot product and norm, we get:

$$\begin{aligned} \text{Yat}(\mathbf{w}, k\mathbf{u}) &= \frac{\langle \mathbf{w}, k\mathbf{u} \rangle^2}{\|\mathbf{w} - k\mathbf{u}\|^2 + \epsilon} \\ &= \frac{k^2 \langle \mathbf{w}, \mathbf{u} \rangle^2}{\|\mathbf{w}\|^2 - 2\langle \mathbf{w}, k\mathbf{u} \rangle + \|k\mathbf{u}\|^2 + \epsilon} \\ &= \frac{k^2 \langle \mathbf{w}, \mathbf{u} \rangle^2}{\|\mathbf{w}\|^2 - 2k\langle \mathbf{w}, \mathbf{u} \rangle + k^2\|\mathbf{u}\|^2 + \epsilon} \end{aligned}$$

Since $\mathbf{u}$ is a unit vector, $\|\mathbf{u}\|^2 = 1$:

$$\text{Yat}(\mathbf{w}, k\mathbf{u}) = \frac{k^2 \langle \mathbf{w}, \mathbf{u} \rangle^2}{\|\mathbf{w}\|^2 - 2k\langle \mathbf{w}, \mathbf{u} \rangle + k^2 + \epsilon} \tag{A30}$$

To find the limit as $k \to \infty$, we divide the numerator and the denominator by the highest power of $k$, which is $k^2$:

$$\lim_{k \to \infty} \text{Yat}(\mathbf{w}, k\mathbf{u}) = \lim_{k \to \infty} \frac{\langle \mathbf{w}, \mathbf{u} \rangle^2}{\frac{\|\mathbf{w}\|^2}{k^2} - \frac{2\langle \mathbf{w}, \mathbf{u} \rangle}{k} + 1 + \frac{\epsilon}{k^2}} \tag{A31}$$

As $k \to \infty$, the terms with $k$ in the denominator approach zero:

$$\lim_{k \to \infty} \text{Yat}(\mathbf{w}, k\mathbf{u}) = \frac{\langle \mathbf{w}, \mathbf{u} \rangle^2}{0 - 0 + 1 + 0} = \langle \mathbf{w}, \mathbf{u} \rangle^2 \tag{A32}$$

By the definition of the dot product, $\langle \mathbf{w}, \mathbf{u} \rangle = \|\mathbf{w}\|\|\mathbf{u}\| \cos\theta = \|\mathbf{w}\| \cos\theta$, where $\theta$ is the angle between $\mathbf{w}$ and $\mathbf{u}$ (the direction of $\mathbf{x}$).

Therefore, the limit is:

$$\lim_{\|\mathbf{x}\| \to \infty} \text{Yat}(\mathbf{w}, \mathbf{x}) = (\|\mathbf{w}\| \cos\theta)^2 = \|\mathbf{w}\|^2 \cos^2\theta \tag{A33}$$

Since $\cos^2\theta$ is always between 0 and 1, the output of the Yat-product is bounded by $0 \leq \text{Yat}(\mathbf{w}, \mathbf{x}) \leq \|\mathbf{w}\|^2$. The limit is a finite value that depends only on the squared magnitude of the weight vector and the squared cosine of the angle between the weight and input vectors. This proves that the kernel is naturally self-regulating and does not diverge, even for inputs of arbitrarily large magnitude. □

 doi:10.20944/preprints202506.2010.v2



*Appendix G.4 Addressing Internal Covariate Shift*

**Theorem A4** (Asymptotic Independence of Score Statistics)**.** *Let $a = Yat(\mathbf{w}, \mathbf{x})$ be the score of a neuron for an input $\mathbf{x}$. Consider a mini-batch of inputs $\mathcal{B} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, where each input is represented as $\mathbf{x}_i = k_i \mathbf{u}_i$ with magnitude $k_i = \|\mathbf{x}_i\|$ and direction $\mathbf{u}_i$. Let $\mu_{\mathcal{B}}(a)$ and $\sigma^2_{\mathcal{B}}(a)$ denote the empirical mean and variance of the scores over the mini-batch. In the limit as $k_i \to \infty$ for all $i \in \{1, \ldots, N\}$, the mean and variance of the scores converge to values that are independent of the magnitudes $k_i$:*

$$\lim_{k_1,\ldots,k_N \to \infty} \mu_{\mathcal{B}}(a) = \|\mathbf{w}\|^2 \mathbb{E}_{\mathbf{u}\in\mathcal{U}}[\cos^2\theta(\mathbf{w}, \mathbf{u})]$$
$$\lim_{k_1,\ldots,k_N \to \infty} \sigma^2_{\mathcal{B}}(a) = \|\mathbf{w}\|^4 Var_{\mathbf{u}\in\mathcal{U}}[\cos^2\theta(\mathbf{w}, \mathbf{u})]$$

*where $\mathcal{U} = \{\mathbf{u}_1, \ldots, \mathbf{u}_N\}$ is the set of direction vectors and the expectation and variance are taken over this set. This demonstrates that the scores statistics are asymptotically decoupled from input magnitudes, thus mitigating internal covariate shift.*

**Proof.** Let a neuron in a neural network layer be defined by the Yat-product kernel, $a = \text{Yat}(\mathbf{w}, \mathbf{x})$, where $\mathbf{w}$ is the weight vector and $\mathbf{x}$ is the input vector from the previous layer. Internal Covariate Shift (ICS) refers to the change in the distribution of the input $\mathbf{x}$ during training, which in turn causes undesirable shifts in the distribution of the score $a$. We will demonstrate that the statistical moments of the score $a$ are asymptotically independent of the input magnitude $\|\mathbf{x}\|$, thus mitigating ICS.

From the proof in Section G.3, we have established the asymptotic behavior of the Yat-product for an input $\mathbf{x} = k\mathbf{u}$ where $k = \|\mathbf{x}\|$:

$$\lim_{k\to\infty} \text{Yat}(\mathbf{w}, k\mathbf{u}) = \|\mathbf{w}\|^2 \cos^2\theta \tag{A34}$$

where $\theta$ is the angle between the weight vector $\mathbf{w}$ and the input direction vector $\mathbf{u}$. For inputs with large magnitudes, which are a primary concern for training stability, the score can be approximated as:

$$\text{Yat}(\mathbf{w}, \mathbf{x}) \approx \|\mathbf{w}\|^2 \cos^2\theta \tag{A35}$$

Consider a mini-batch of $N$ inputs, $\mathcal{B} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$. The corresponding score are $a_i = \text{Yat}(\mathbf{w}, \mathbf{x}_i)$. The empirical mean of the scores over this mini-batch is:

$$\mu_{\mathcal{B}}(a) = \frac{1}{N}\sum_{i=1}^{N} a_i \tag{A36}$$

Assuming the inputs in the mini-batch have sufficiently large magnitudes, we can substitute the asymptotic approximation:

$$\mu_{\mathcal{B}}(a) \approx \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{w}\|^2 \cos^2\theta_i = \|\mathbf{w}\|^2 \cdot \frac{1}{N}\sum_{i=1}^{N} \cos^2\theta_i \tag{A37}$$

This can be expressed in terms of the empirical expectation over the mini-batch:

$$\mu_{\mathcal{B}}(a) \approx \|\mathbf{w}\|^2 \mathbb{E}_{\mathbf{x}\in\mathcal{B}}[\cos^2\theta(\mathbf{w}, \mathbf{x})] \tag{A38}$$

Similarly, the empirical variance of the scores is:

$$\sigma^2_{\mathcal{B}}(a) = \frac{1}{N}\sum_{i=1}^{N} (a_i - \mu_{\mathcal{B}}(a))^2 \tag{A39}$$

 doi:10.20944/preprints202506.2010.v2



Using the same approximation, the variance becomes:

$$\sigma^2_{\mathcal{B}}(a) \approx \mathbb{E}_{\mathbf{x}\in\mathcal{B}}[(\|\mathbf{w}\|^2\cos^2\theta - \|\mathbf{w}\|^2\mathbb{E}_{\mathbf{x}\in\mathcal{B}}[\cos^2\theta])^2] \tag{A40}$$

$$= \|\mathbf{w}\|^4\mathbb{E}_{\mathbf{x}\in\mathcal{B}}[(\cos^2\theta - \mathbb{E}_{\mathbf{x}\in\mathcal{B}}[\cos^2\theta])^2] \tag{A41}$$

$$= \|\mathbf{w}\|^4\Big(\mathbb{E}_{\mathbf{x}\in\mathcal{B}}[\cos^4\theta] - (\mathbb{E}_{\mathbf{x}\in\mathcal{B}}[\cos^2\theta])^2\Big) \tag{A42}$$

Crucially, both the empirical mean and variance of the scores are, in the large-magnitude limit, functions of the weight vector's magnitude $\|\mathbf{w}\|$ and the statistics of the angle $\theta$ between the weights and the inputs. They are independent of the input magnitudes $\|\mathbf{x}_i\|$.

During training, while the distribution of $\mathbf{x}$ (and thus the distribution of angles $\theta_i$) and the weight vector $\mathbf{w}$ evolve, the primary source of instability associated with ICS, namely, drastic fluctuations in the magnitudes of layer inputs, is filtered out. The evolution of the score distribution is governed by the more gradual changes in the learned weight vector and the angular structure of the data, rather than the raw input scales. This decoupling of score statistics from input magnitudes provides inherent stabilization, thus mitigating internal covariate shift. □

*Appendix G.5 Proof of Stable Learning for the Yat-product*

**Theorem A5** (The Yat-Product Ensures Stable Learning)**.** *The gradient of the Yat-product with respect to its input, $\nabla_{\mathbf{x}}Yat(\mathbf{w},\mathbf{x})$, approaches zero as the input vector $\mathbf{x}$ moves infinitely far from the weight vector $\mathbf{w}$.*

**Proof.** We aim to prove that the learning signal, represented by the gradient of the Yat-product with respect to the input $\mathbf{x}$, diminishes for inputs that are distant from the learned weight vector $\mathbf{w}$. This ensures that outliers do not cause large, destabilizing updates.

The Yat-product is defined as:

$$\text{Yat}(\mathbf{w},\mathbf{x}) = \frac{\langle\mathbf{w},\mathbf{x}\rangle^2}{\|\mathbf{w}-\mathbf{x}\|^2+\epsilon} = \frac{N(\mathbf{x})}{D(\mathbf{x})}$$

where $N(\mathbf{x}) = \langle\mathbf{w},\mathbf{x}\rangle^2$ and $D(\mathbf{x}) = \|\mathbf{w}-\mathbf{x}\|^2+\epsilon$.

Using the quotient rule for vector calculus, the gradient $\nabla_{\mathbf{x}}$Yat is:

$$\nabla_{\mathbf{x}}\text{Yat} = \frac{(\nabla_{\mathbf{x}}N)D - N(\nabla_{\mathbf{x}}D)}{D^2}$$

First, we compute the gradients of the numerator $N(\mathbf{x})$ and the denominator $D(\mathbf{x})$:

1. Gradient of the Numerator

$N(\mathbf{x}) = (\mathbf{w}^T\mathbf{x})^2$

$$\nabla_{\mathbf{x}}N(\mathbf{x}) = 2(\mathbf{w}^T\mathbf{x})\cdot\nabla_{\mathbf{x}}(\mathbf{w}^T\mathbf{x}) = 2\langle\mathbf{w},\mathbf{x}\rangle\mathbf{w}$$

2. Gradient of the Denominator

$D(\mathbf{x}) = \|\mathbf{w}-\mathbf{x}\|^2+\epsilon = (\mathbf{w}-\mathbf{x})^T(\mathbf{w}-\mathbf{x})+\epsilon$

$$\nabla_{\mathbf{x}}D(\mathbf{x}) = 2(\mathbf{w}-\mathbf{x})\cdot(-1) = -2(\mathbf{w}-\mathbf{x}) = 2(\mathbf{x}-\mathbf{w})$$

Substituting these into the quotient rule expression:

$$\nabla_{\mathbf{x}}\text{Yat} = \frac{(2\langle\mathbf{w},\mathbf{x}\rangle\mathbf{w})(\|\mathbf{w}-\mathbf{x}\|^2+\epsilon) - (\langle\mathbf{w},\mathbf{x}\rangle^2)(2(\mathbf{x}-\mathbf{w}))}{(\|\mathbf{w}-\mathbf{x}\|^2+\epsilon)^2}$$

To analyze the behavior for distant inputs, we examine the limit as $\|\mathbf{x}\|\to\infty$. Let $\mathbf{x} = k\mathbf{u}$, where $k = \|\mathbf{x}\|$ and $\mathbf{u}$ is a unit vector.

As $k\to\infty$:

 doi:10.20944/preprints202506.2010.v2



- $\langle \mathbf{w}, \mathbf{x} \rangle = k\langle \mathbf{w}, \mathbf{u} \rangle \sim \mathcal{O}(k)$
- $\|\mathbf{w} - \mathbf{x}\|^2 = \|\mathbf{w}\|^2 - 2k\langle \mathbf{w}, \mathbf{u} \rangle + k^2 \sim \mathcal{O}(k^2)$

Let's analyze the order of magnitude for the terms in the gradient's numerator:

- First term: $(2\langle \mathbf{w}, \mathbf{x} \rangle \mathbf{w})(\|\mathbf{w} - \mathbf{x}\|^2 + \epsilon) \sim \mathcal{O}(k) \cdot \mathcal{O}(k^2) = \mathcal{O}(k^3)$
- Second term: $(\langle \mathbf{w}, \mathbf{x} \rangle^2)(2(\mathbf{x} - \mathbf{w})) \sim \mathcal{O}(k^2) \cdot \mathcal{O}(k) = \mathcal{O}(k^3)$

The numerator as a whole is of order $\mathcal{O}(k^3)$.

The denominator is $(\|\mathbf{w} - \mathbf{x}\|^2 + \epsilon)^2 \sim (\mathcal{O}(k^2))^2 = \mathcal{O}(k^4)$.

Therefore, the magnitude of the gradient behaves as:

$$\|\nabla_{\mathbf{x}} \text{Yat}\| \sim \frac{\mathcal{O}(k^3)}{\mathcal{O}(k^4)} = \mathcal{O}\left(\frac{1}{k}\right)$$

As $k = \|\mathbf{x}\| \to \infty$, the magnitude of the gradient approaches zero:

$$\lim_{\|\mathbf{x}\| \to \infty} \|\nabla_{\mathbf{x}} \text{Yat}(\mathbf{w}, \mathbf{x})\| = 0$$

This proves that for inputs $\mathbf{x}$ that are very far from the weight vector $\mathbf{w}$, the gradient becomes vanishingly small. The learning process is therefore stable, as distant outliers will not exert a significant influence on the weight updates. □

*Appendix G.6 Proof of Universal Approximation Theorem for Yat-Product Networks*

**Theorem A6** (Universal Approximation Theorem for NMNs)**.** *Let $C(K)$ be the space of continuous functions on a compact set $K \subset \mathbb{R}^d$. A single-hidden-layer Neural-Matter Network (NMN) with Yat-product activation functions can approximate any function $f \in C(K)$ to any desired precision. That is, for any $f \in C(K)$ and any $\delta > 0$, there exists an NMN function $g(\mathbf{x}) = \sum_{i=1}^{N} c_i \text{Yat}(\mathbf{w}_i, \mathbf{x})$ such that $\sup_{\mathbf{x} \in K} |f(\mathbf{x}) - g(\mathbf{x})| < \delta$.*

**Proof.** The proof relies on the theory of universal kernels. A continuous kernel $K$ on a compact set $\mathcal{X}$ is defined as universal if its associated Reproducing Kernel Hilbert Space (RKHS), $\mathcal{H}_K$, is dense in the space of continuous functions $C(\mathcal{X})$ with respect to the uniform norm. The span of functions of the form $g(\mathbf{x}) = \sum_{i=1}^{N} c_i K(\mathbf{w}_i, \mathbf{x})$ is dense in $\mathcal{H}_K$. Therefore, if the Yat-product kernel is universal, the set of NMN functions is dense in $C(K)$, which proves the theorem.

A key result from kernel theory [35] states that the product of two universal kernels is also universal. We have previously shown in Section G.2 that the Yat-product kernel $K$ can be expressed as the pointwise product of two kernels:

$$K(\mathbf{w}, \mathbf{x}) = K_1(\mathbf{w}, \mathbf{x}) \cdot K_2(\mathbf{w}, \mathbf{x})$$

where:

- $K_1(\mathbf{w}, \mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle^2$ (the squared polynomial kernel)
- $K_2(\mathbf{w}, \mathbf{x}) = (\|\mathbf{w} - \mathbf{x}\|^2 + \epsilon)^{-1}$ (a translation-invariant kernel)

We will now show that both $K_1$ and $K_2$ are universal kernels on any compact set $K \subset \mathbb{R}^d$.

1. $K_1$ is a Universal Kernel

The polynomial kernel $k_p(\mathbf{w}, \mathbf{x}) = (\langle \mathbf{w}, \mathbf{x} \rangle + c)^p$ is known to be universal for any $p \geq 1$ and $c > 0$ [35]. Our kernel $K_1$ is a specific instance of the polynomial kernel family and is also universal on any compact subset of $\mathbb{R}^d$.

2. $K_2$ is a Universal Kernel

A sufficient condition for a continuous, translation-invariant kernel $k(\mathbf{z})$ (where $\mathbf{z} = \mathbf{w} - \mathbf{x}$) to be universal is that its Fourier transform must be strictly positive almost everywhere [65]. In the proof

 doi:10.20944/preprints202506.2010.v2



of Mercer's condition (Section G.2), we showed that $K_2$ has a non-negative Fourier transform via its integral representation:

$$K_2(\mathbf{w}, \mathbf{x}) = \int_0^\infty e^{-\epsilon t} e^{-t\|\mathbf{w}-\mathbf{x}\|^2} dt$$

The integrand is a strictly positive function for all $t \geq 0$. The integral of a strictly positive function is strictly positive. Therefore, the Fourier transform of $K_2$ is strictly positive everywhere, which is a stronger condition than required. Thus, $K_2$ is a universal kernel.

Since both $K_1$ and $K_2$ are universal kernels, their product, the Yat-product kernel $K(\mathbf{w}, \mathbf{x})$, is also universal. This implies that the span of functions generated by the NMN is dense in $C(K)$.

Therefore, for any continuous function $f \in C(K)$ and any $\delta > 0$, there exists an NMN function $g(\mathbf{x})$ such that $\sup_{\mathbf{x}\in K} |f(\mathbf{x}) - g(\mathbf{x})| < \delta$, which completes the proof. □

*Appendix G.7 Information-Geometric Foundations of the Yat-Product*

Appendix G.7.1 Definition and Geometric Interpretation

We consider probability distributions in the simplex $\Delta^{n-1} = \{\mathbf{p} \in \mathbb{R}^n_{\geq 0} : \sum_{i=1}^n p_i = 1\}$. While information geometry traditionally employs the Fisher metric, we establish a novel connection to Euclidean geometry through the Yat-product.

**Definition A1** (Yat-Product: Geometric Similarity Measure)**.** *For distinct distributions $\mathbf{p}, \mathbf{q} \in \Delta^{n-1}$, the Yat-product is defined as:*

$$Yat(\mathbf{p}, \mathbf{q}) := \frac{(\mathbf{p} \cdot \mathbf{q})^2}{\|\mathbf{p} - \mathbf{q}\|_2^2}$$

*where:*

- $\mathbf{p} \cdot \mathbf{q} = \sum_{i=1}^n p_i q_i$ *measures distributional alignment*
- $\|\mathbf{p} - \mathbf{q}\|_2^2 = \sum_{i=1}^n (p_i - q_i)^2$ *quantifies Euclidean dissimilarity*

*This ratio captures the tension between distributional agreement and geometric separation.*

**Remark A1** (Singularity and Invariance Properties)**.** *When $\mathbf{p} = \mathbf{q}$, we define $Yat(\mathbf{p}, \mathbf{q}) := \infty$ via the limit:*

$$\lim_{\mathbf{q}\to\mathbf{p}} Yat(\mathbf{p}, \mathbf{q}) = \infty$$

*reflecting maximal self-similarity. The Yat-product exhibits two key properties:*

1. ***Symmetry**: $Yat(\mathbf{p}, \mathbf{q}) = Yat(\mathbf{q}, \mathbf{p})$*
2. ***Scale Invariance**: Invariant under index permutation*

Appendix G.7.2 Extremal Similarity Theorems

**Theorem A7** (Minimal Similarity and Statistical Orthogonality)**.** *For distinct $\mathbf{p}, \mathbf{q} \in \Delta^{n-1}$:*

$$Yat(\mathbf{p}, \mathbf{q}) = 0 \iff \text{supp}(\mathbf{p}) \cap \text{supp}(\mathbf{q}) = \emptyset$$

*Moreover, this condition implies information-theoretic divergence:*

$$\text{KL}(\mathbf{p}\|\mathbf{q}) = \infty, \qquad \text{KL}(\mathbf{q}\|\mathbf{p}) = \infty, \qquad H(\mathbf{p}, \mathbf{q}) = \infty$$

**Proof.** ($\Rightarrow$) Assume $Yat(\mathbf{p}, \mathbf{q}) = 0$. Since $\mathbf{p} \neq \mathbf{q}$, $\|\mathbf{p} - \mathbf{q}\|_2^2 > 0$. Thus $(\mathbf{p} \cdot \mathbf{q})^2 = 0 \Rightarrow \sum p_i q_i = 0$. By non-negativity of probabilities, $p_i q_i = 0\ \forall i$, hence $\text{supp}(\mathbf{p}) \cap \text{supp}(\mathbf{q}) = \emptyset$.

($\Leftarrow$) Disjoint supports imply $\forall i : (p_i > 0 \Rightarrow q_i = 0)$ and vice versa. Thus $\mathbf{p} \cdot \mathbf{q} = 0$, so $Yat(\mathbf{p}, \mathbf{q}) = 0$.

The KL divergence $\text{KL}(\mathbf{p}\|\mathbf{q})$ contains terms $\log(p_i/q_i)$ where $p_i > 0$ and $q_i = 0$, causing divergence. Similar reasoning applies to $\text{KL}(\mathbf{q}\|\mathbf{p})$ and cross-entropy $H(\mathbf{p}, \mathbf{q})$ [63]. □

 doi:10.20944/preprints202506.2010.v2



**Theorem A8** (Maximal Similarity and Distributional Identity)**.** *For* $\mathbf{p}, \mathbf{q} \in \Delta^{n-1}$*:*

$$Yat(\mathbf{p}, \mathbf{q}) = \infty \iff \mathbf{p} = \mathbf{q}$$

*When satisfied, information-theoretic consistency holds:*

$$\mathrm{KL}(\mathbf{p} \| \mathbf{q}) = 0 \qquad \textit{and} \qquad H(\mathbf{p}, \mathbf{q}) = H(\mathbf{p})$$

**Proof.** ($\Rightarrow$) Suppose $Yat(\mathbf{p}, \mathbf{q}) \to \infty$. By Cauchy-Schwarz [62], $\mathbf{p} \cdot \mathbf{q} \leq \|\mathbf{p}\|_2 \|\mathbf{q}\|_2 \leq 1$. Since the numerator is bounded, $\|\mathbf{p} - \mathbf{q}\|_2^2 \to 0$, implying $\mathbf{p} = \mathbf{q}$.

($\Leftarrow$) For $\mathbf{p} = \mathbf{q}$, consider $\mathbf{q}^{(k)} \to \mathbf{p}$. Then:

$$\mathbf{p} \cdot \mathbf{q}^{(k)} \to \|\mathbf{p}\|_2^2 \geq \tfrac{1}{n} > 0 \quad (\text{since } \|\mathbf{p}\|_2^2 \geq \tfrac{1}{n} \text{ by Cauchy-Schwarz})$$

while $\|\mathbf{p} - \mathbf{q}^{(k)}\|_2^2 \to 0$, so $Yat(\mathbf{p}, \mathbf{q}^{(k)}) \to \infty$.

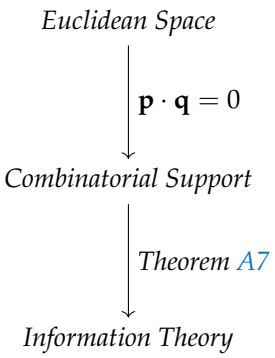
When $\mathbf{p} = \mathbf{q}$, $\log(p_i/q_i) = 0$ for all $i$, so $\mathrm{KL}(\mathbf{p} \| \mathbf{q}) = 0$. Cross-entropy reduces to entropy when distributions are identical. □

**Remark A2** (Duality of Orthogonality Concepts)**.** *The Yat-product unifies three distinct notions of orthogonality:*

$$\begin{aligned} \textit{Euclidean:}\ & \mathbf{p} \perp \mathbf{q} \iff \mathbf{p} \cdot \mathbf{q} = 0 \\ \textit{Combinatorial:}\ & \mathrm{supp}(\mathbf{p}) \cap \mathrm{supp}(\mathbf{q}) = \emptyset \\ \textit{Information-Theoretic:}\ & \mathrm{KL}(\mathbf{p} \| \mathbf{q}) = \infty \end{aligned}$$

*Theorem A7 establishes their equivalence through* $Yat(\mathbf{p}, \mathbf{q}) = 0$*. This contrasts with Fisher-based orthogonality, which depends on manifold curvature.*

**Remark A3** (Geometric-Information Duality)**.** *The Yat-product creates a bridge between geometric and probabilistic perspectives:*



 doi:10.20944/preprints202506.2010.v2



*Appendix G.8 Diagrams*

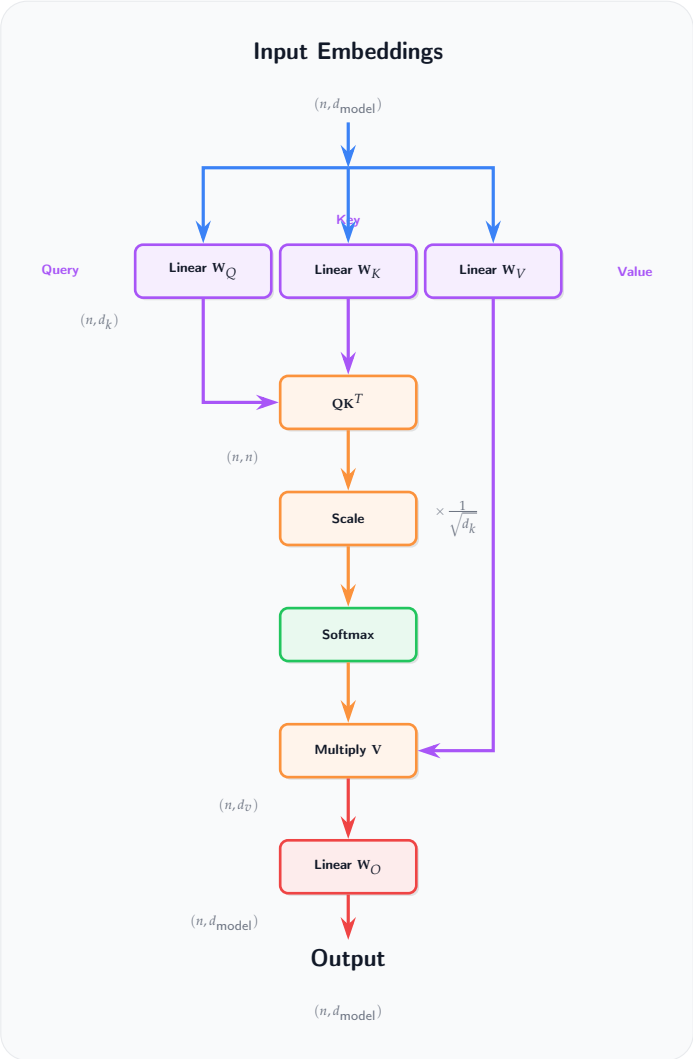



**Figure A10.** The core Scaled Dot-Product Attention calculation.

 doi:10.20944/preprints202506.2010.v2



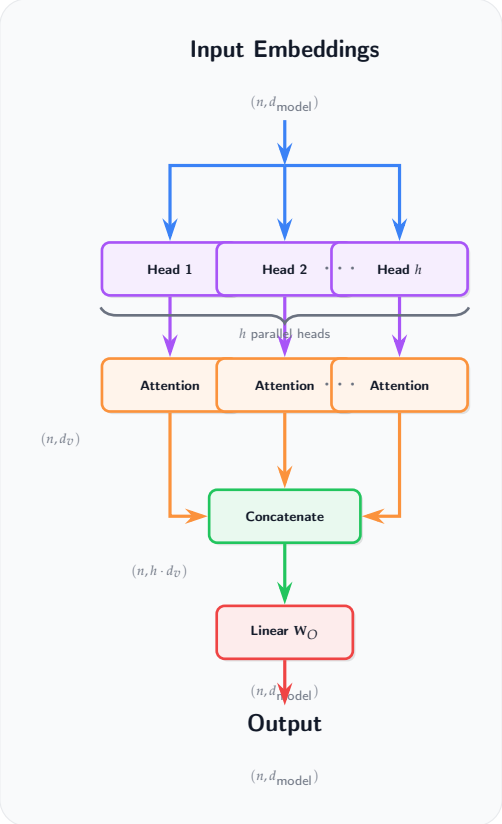



**Figure A11.** The Multi-Head Attention mechanism, which runs attention in parallel.

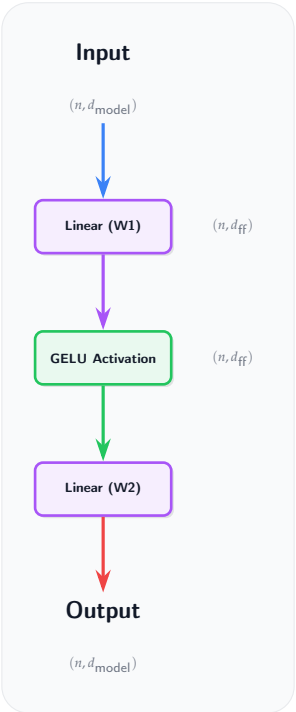



**Figure A12.** The position-wise Feed-Forward Network (MLP).

 doi:10.20944/preprints202506.2010.v2



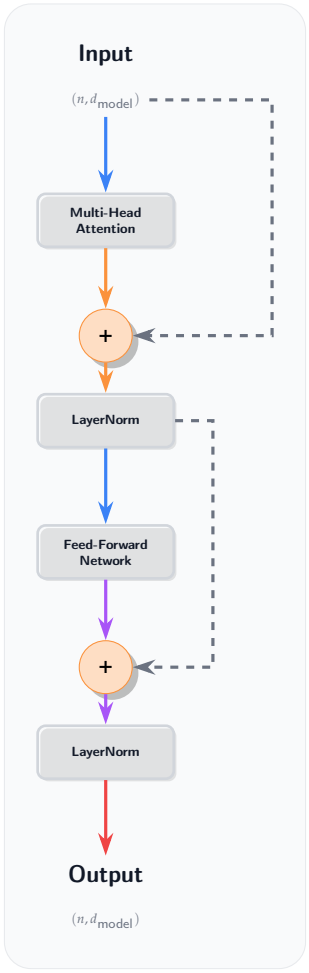



**Figure A13.** The complete Encoder block, showing how Multi-Head Attention and the FFN are combined using residual connections and layer normalization.

 doi:10.20944/preprints202506.2010.v2



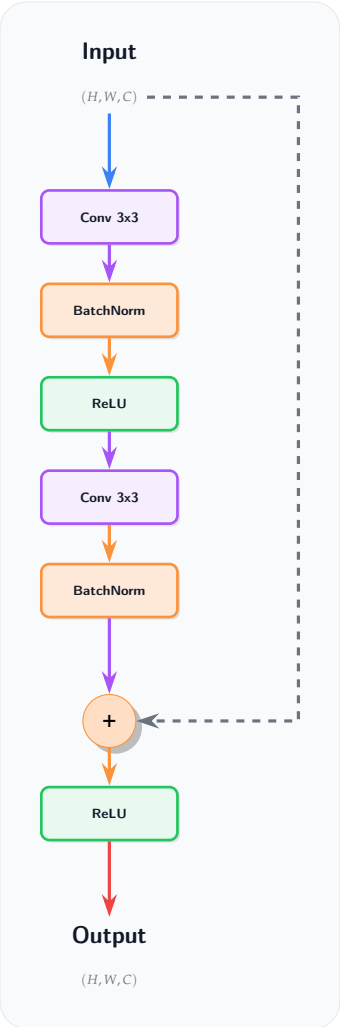



**Figure A14.** A standard ResNet "Basic Block" with a residual (skip) connection. This concept of bypassing layers is a precursor to the residual connections in Transformers.

 doi:10.20944/preprints202506.2010.v2



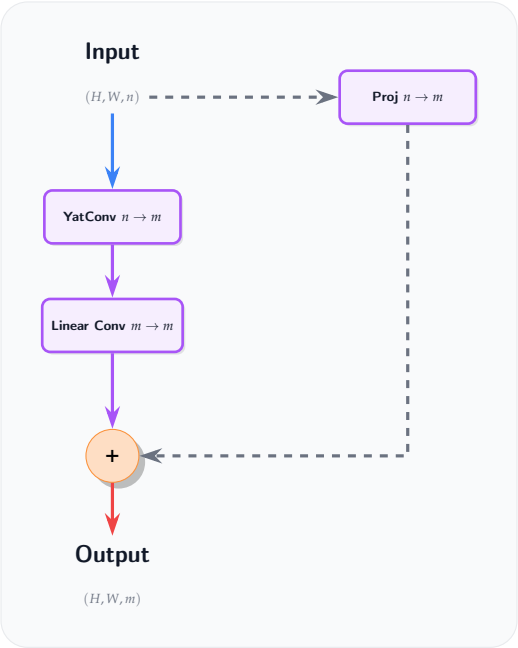



**Figure A15.** A CNMN Residual Block as used in AetherResNet: a YatConv layer ($n \to m$) followed by a linear Conv ($m \to m$), with no activation functions or normalization layers. The skip connection includes a projection if $n \neq m$.

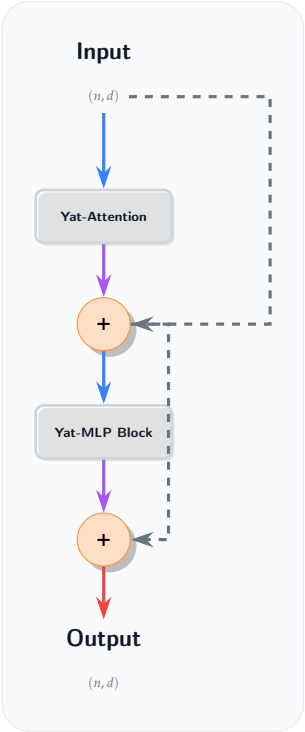



**Figure A16.** A YatFormer transformer block as used in AetherGPT: Yat-Attention followed by a Yat-MLP block, each with residual connections. No projection after attention, and no normalization layers.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 August 2025 doi:10.20944/preprints202506.2010.v2



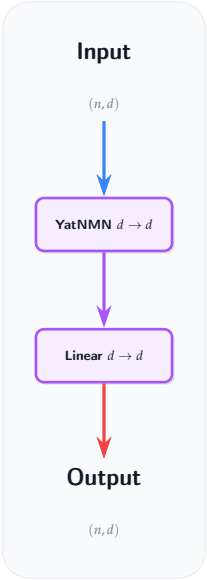



**Figure A17.** The Yat-MLP block as used in AetherGPT: a YatNMN layer ($d \to d$) followed by a linear layer ($d \to d$), with no expansion of the hidden dimension.

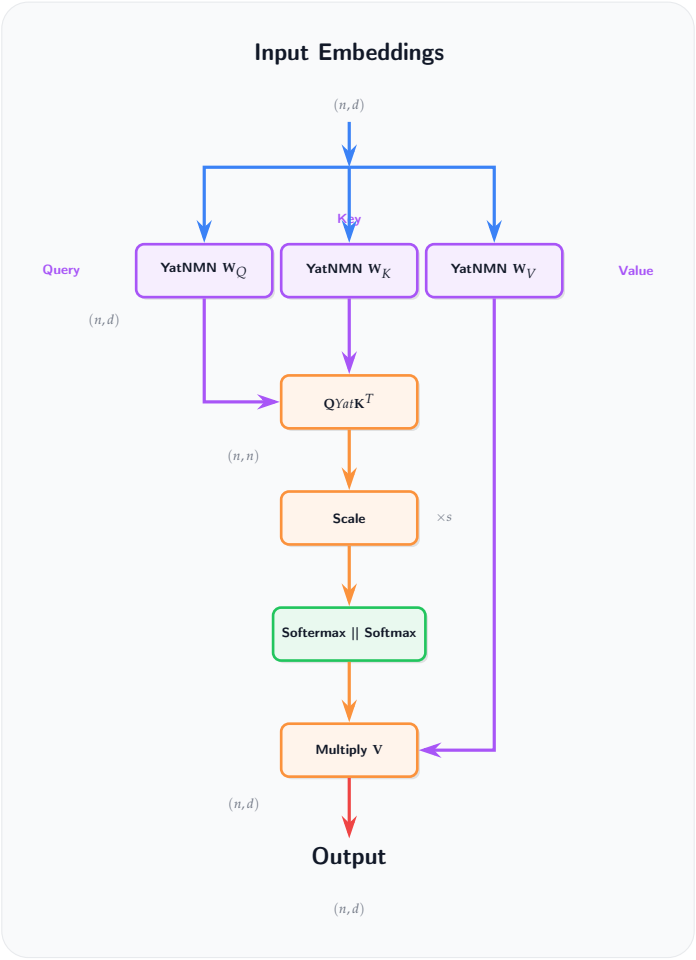



**Figure A18.** The Yat-Attention block as used in AetherGPT: input embeddings are projected to queries, keys, and values; Yat-product similarity is computed between queries and keys; softermax is applied; and the output is the weighted sum of values. No projection layer after attention.

 doi:10.20944/preprints202506.2010.v2



## References


1. Rosenblatt, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review* **1958**, *65*, 386.
2. McCulloch, W.S.; Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics* **1943**, *5*, 115–133.
3. Hornik, K.; Stinchcombe, M.; White, H. Multilayer feedforward networks are universal approximators. *Neural networks* **1989**, *2*, 359–366.
4. Cybenko, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems* **1989**, *2*, 303–314.
5. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep learning*; Vol. 1, MIT Press, 2016.
6. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *nature* **2015**, *521*, 436–444.
7. Nair, V.; Hinton, G.E. Rectified linear units improve restricted boltzmann machines. In Proceedings of the Proceedings of the 27th international conference on machine learning (ICML-10), 2010, pp. 807–814.
8. Hendrycks, D.; Gimpel, K. Gaussian Error Linear Units (GELUs), 2023, [arXiv:cs.LG/1606.08415].
9. Klambauer, G.; Unterthiner, T.; Mayr, A.; Hochreiter, S. Self-normalizing neural networks. *Advances in neural information processing systems* **2017**, *30*.
10. Bronstein, M.M.; Bruna, J.; LeCun, Y.; Szlam, A.; Vandergheynst, P. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine* **2017**, *34*, 18–42.
11. Bronstein, M.M.; Bruna, J.; Cohen, T.; Veličković, P. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges, 2021, [arXiv:cs.LG/2104.13478].
12. Balestriero, R.; Humayun, A.I.; Baraniuk, R.G. On the geometry of deep learning. *NOTICES OF THE AMERICAN MATHEMATICAL SOCIETY* **2025**, *72*.
13. Steck, H.; Ekanadham, C.; Kallus, N. Is cosine-similarity of embeddings really about similarity? In Proceedings of the Companion Proceedings of the ACM Web Conference 2024, 2024, pp. 887–890.
14. Draganov, A.; Vadgama, S.; Bekkers, E.J. The hidden pitfalls of the cosine similarity loss. *arXiv preprint arXiv:2406.16468* **2024**.
15. Kepler, J. Ad Vitellionem paralipomena, quibus astronomiae pars optica traditur. 1604. *Johannes Kepler: Gesammelte Werke, Ed. Walther von Dyck and Max Caspar, Münchenk* **1939**.
16. Newton, I. *Philosophiæ Naturalis Principia Mathematica*; S. Pepys: London, 1687.
17. de Coulomb, C.A. Premier mémoire sur l'électricité et le magnétisme. *Histoire de l'Académie Royale des Sciences* **1785**, pp. 1–31. in French.
18. Gauss, C.F. *Allgemeine Lehrsätze in Beziehung auf die im verkehrten Verhältniss des Quadrats der Entfernung wirkenden Anziehungs- und Abstossungskräfte*; Dietrich: Göttingen, 1835.
19. Bouhsine, T. Deep Learning 2.0: Artificial Neurons That Matter – Reject Correlation, Embrace Orthogonality, 2024, [arXiv:cs.LG/2411.08085].
20. Bouhsine, T.; Aaroussi, I.E.; Faysal, A.; Wang. SimO Loss: Anchor-Free Contrastive Loss for Fine-Grained Supervised Contrastive Learning. In Proceedings of the Submitted to The Thirteenth International Conference on Learning Representations, 2024. under review.
21. Lu, Z.; Pu, H.; Wang, F.; Hu, Z.; Wang, L. The expressive power of neural networks: A view from the width. *Advances in neural information processing systems* **2017**, *30*.
22. Huang, C. ReLU networks are universal approximators via piecewise linear or constant functions. *Neural Computation* **2020**, *32*, 2249–2278.
23. Tanimoto, T.T. Elementary mathematical theory of classification and prediction **1958**.
24. Jaccard, P. Étude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bulletin de la Société Vaudoise des Sciences Naturelles* **1901**, *37*, 547–579.
25. Rolls, E.T.; Treves, A. The neuronal encoding of information in the brain. *Progress in neurobiology* **2011**, *95*, 448–490.
26. Barlow, H.B.; et al. Possible principles underlying the transformation of sensory messages. *Sensory communication* **1961**, *1*, 217–233.
27. Montavon, G.; Samek, W.; Müller, K.R. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing* **2018**, *73*, 1–15. https://doi.org/10.1016/j.dsp.2017.10.011.
28. Montufar, G.F.; Pascanu, R.; Cho, K.; Bengio, Y. On the number of linear regions of deep neural networks. *Advances in neural information processing systems* **2014**, *27*.
29. Ba, J.; Caruana, R. Do deep nets really need to be deep? *Advances in neural information processing systems* **2014**, *27*.

 doi:10.20944/preprints202506.2010.v2




30. Pezeshki, M.; Kaba, S.O.; Bengio, Y.; Courville, A.; Precup, D.; Lajoie, G. Gradient Starvation: A Learning Proclivity in Neural Networks, 2021, [arXiv:cs.LG/2011.09468].
31. Wan, L.; Zeiler, M.; Zhang, S.; Le Cun, Y.; Fergus, R. Regularization of neural networks using dropconnect. In Proceedings of the International conference on machine learning. PMLR, 2013, pp. 1058–1066.
32. Rumelhart, D.E.; Zipser, D. Competitive learning. *Cognitive science* **1985**, *9*, 75–112.
33. Mercer, J. Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character* **1909**, *209*, 415–446.
34. Hofmann, T.; Schölkopf, B.; Smola, A.J. Kernel methods in machine learning **2008**.
35. Micchelli, C.A.; Xu, Y.; Zhang, H. Universal Kernels. *Journal of Machine Learning Research* **2006**, *7*.
36. Schölkopf, B.; Smola, A.; Müller, K.R. Kernel principal component analysis. In Proceedings of the International conference on artificial neural networks. Springer, 1997, pp. 583–588.
37. Cortes, C. Support-Vector Networks. *Machine Learning* **1995**.
38. Jacot, A.; Gabriel, F.; Hongler, C. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems* **2018**, *31*.
39. Modest, M.F. *Radiative Heat Transfer*, 3 ed.; Academic Press: New York, 2013.
40. Knoll, G.F. *Radiation Detection and Measurement*, 4 ed.; John Wiley & Sons: Hoboken, NJ, 2010.
41. Rea, M.S. *The IESNA Lighting Handbook: Reference & Application*, 9 ed.; Illuminating Engineering Society of North America: New York, 2000.
42. Rappaport, T.S. *Wireless Communications: Principles and Practice*, 2 ed.; Prentice Hall: Upper Saddle River, NJ, 2002.
43. Skolnik, M.I. *Radar Handbook*, 3 ed.; McGraw-Hill Education: New York, 2008.
44. Aki, K.; Richards, P.G. *Quantitative Seismology*, 2 ed.; University Science Books: Sausalito, CA, 2002.
45. Batchelor, G.K. *An Introduction to Fluid Dynamics*; Cambridge University Press: Cambridge, UK, 2000.
46. Anderson, J.E. The Gravity Model. *Annual Review of Economics* **2011**, *3*, 133–160.
47. Schölkopf, B.; Smola, A.; Müller, K.R. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation* **1998**, *10*, 1299–1319.
48. Williams, C.K.; Rasmussen, C.E. *Gaussian processes for machine learning*; Vol. 2, MIT press Cambridge, MA, 2006.
49. Ng, A.; Jordan, M.; Weiss, Y. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems* **2001**, *14*.
50. Schölkopf, B.; Platt, J.C.; Shawe-Taylor, J.; Smola, A.J.; Williamson, R.C. Estimating the support of a high-dimensional distribution. *Neural computation* **2001**, *13*, 1443–1471.
51. Williams, C.; Seeger, M. Using the Nyström method to speed up kernel machines. *Advances in neural information processing systems* **2000**, *13*.
52. Rahimi, A.; Recht, B. Random features for large-scale kernel machines. *Advances in neural information processing systems* **2007**, *20*.
53. Boser, B.E.; Guyon, I.M.; Vapnik, V.N. A training algorithm for optimal margin classifiers. In Proceedings of the Proceedings of the Fifth Annual Workshop on Computational Learning Theory, New York, NY, USA, 1992; COLT '92, p. 144–152. https://doi.org/10.1145/130385.130401.
54. Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* **1998**, *86*, 2278–2324. https://doi.org/10.1109/5.726791.
55. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. *Advances in neural information processing systems* **2017**, *30*.
56. Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, 2021, [arXiv:cs.CV/2010.11929].
57. Tolstikhin, I.; Houlsby, N.; Kolesnikov, A.; Beyer, L.; Zhai, X.; Unterthiner, T.; Yung, J.; Steiner, A.; Keysers, D.; Uszkoreit, J.; et al. MLP-Mixer: An all-MLP Architecture for Vision, 2021. arXiv:2105.01601 [cs].
58. Liu, H.; Dai, Z.; So, D.R.; Le, Q.V. Pay Attention to MLPs, 2021. arXiv:2105.08050 [cs].
59. Ivakhnenko, A.G. Polynomial theory of complex systems. *IEEE transactions on Systems, Man, and Cybernetics* **1971**, pp. 364–378.
60. Livni, R.; Shalev-Shwartz, S.; Shamir, O. An Algorithm for Training Polynomial Networks, 2014, [arXiv:cs.LG/1304.7045].

 doi:10.20944/preprints202506.2010.v2




61. Lee-Thorp, J.; Ainslie, J.; Eckstein, I.; Ontanon, S. FNet: Mixing Tokens with Fourier Transforms, 2022. arXiv:2105.03824 [cs].
62. Horn, R.A.; Johnson, C.R. *Matrix Analysis*; Cambridge University Press, 2012.
63. Cover, T.M.; Thomas, J.A. *Elements of Information Theory*; Wiley-Interscience, 2006.
64. Schölkopf, B.; Smola, A.J. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*; MIT Press, 2002.
65. Rudin, W. *Functional Analysis*; McGraw-Hill, 1991.
66. Hornik, K. Approximation capabilities of multilayer feedforward networks. *Neural Networks* **1991**, *4*, 251–257.
67. Rudin, W. *Real and Complex Analysis*; McGraw-Hill, 1987.