# Mathematical Analysis of Artificial Neural Networks with Beta Function-Derived Activation Functions

Shahadat Ali[1], Mohammad Ibrahim Mir[2], Arham Hayat[3] and Raziya Sabri[4]

[1,2]Department of Mathematics, University of Kashmir, South Campus, Anantnag 192101, Jammu and Kashmir, India.

[3]Department of Computer Science and Engineering, Galgotias University, Gautam Buddha Nagar, Uttar Pradesh, India.

[4]Mathematics and Science Department, University Centre Weston, Weston super mare, BS23, United Kingdom.

E-mail: shahadatali6764@gmail.com; ibrahimmath80@gmail.com; ar.hayat7@gmail.com; dr.raziyasabri8@gmail.com

**Abstract**

In this paper,We study the map $B(z, \beta_0)$ with fixed $\beta_0 = 2$ as a hidden-layer activation: the first argument $z$ is an affine function of the input, compared in a *Safe* variant with $z$ clamped to $[0.1, 10]$ before evaluation and an *Unsafe* variant without clamping. Experiments use synthetic regression on $x \in [-5, 5]$ (e.g. $\sin x$, $\cos x$, $x \sin x$, $\tanh x$, $\mathrm{e}^{-x^2}$). The Beta is evaluated via SciPy on detached tensors, so training updates the readout layer only. We report mean squared error, training loss, and norms. The Safe variant yields lower test error and more stable traces in these experiments. The scope is the one-hidden-layer setup described in the paper.

**AMS Subject Classification (2020) :** 68T07, 33B15, 41A46

**Keywords:** Special Functions; Artificial Neural Network; Real-time implementation; Machine-Learning; Singular Point; Utilization.

## 1 Introduction

This paper studies the map $B(z, \beta_0)$ with fixed $\beta_0 = 2$ as a nonlinear activation in a feedforward neural network: each hidden unit uses one affine pre-activation $z$ as the first argument of $B$. The emphasis is on comparing a *safe* formulation that clamps $z$ before evaluation with an *unsafe* formulation that does not, so that forward values $B(z, \beta_0)$ differ when $z$ is near singular or ill-conditioned regions of the Beta function. Training is carried out on synthetic regression targets (e.g. $\sin x$, $\cos x$, $x \sin x$, $\tanh x$, $\mathrm{e}^{-x^2}$)

so that optimization and generalization can be compared under controlled conditions. From a mathematical viewpoint we treat scalar regression: given samples $(x_i, y_i)$ with $y_i = f_{\text{true}}(x_i)$, we learn a map $\hat{y} = F(x; \theta)$ whose architecture is specified in Section 4. The network applies a first linear layer (4.1), the activation $B(z_j, \beta_0)$ with fixed $\beta_0 = 2$, and a readout (4.3). Training minimizes the empirical risk $L(\theta)$ defined in (4.4) using Adam. The partial derivatives in (4.5) describe how $B(\alpha, \beta)$ depends on its first argument at fixed $\beta = \beta_0$; they explain why forward values $h_j = B(z_j, \beta_0)$ differ when $z_j$ is clamped versus not, hence different hidden features for the readout. As detailed in Section 4, the implementation evaluates the Beta in SciPy on detached tensors, so automatic differentiation does not propagate the loss gradient through $B$ to the first-layer weights; Adam updates chiefly the readout parameters in (4.3). The Beta function is not a standard choice for activations: it is non-monotone in general, can be costly to evaluate, and is undefined when $\alpha, \beta \leq 0$ in the usual integral sense, which complicates backpropagation. Related work includes neural approximation of special functions [4], Beta-basis and flexible transfer-function formulations, and analysis of learning near singularities [6]. We do not claim broad superiority over common activations such as ReLU or tanh on arbitrary tasks; the experiments isolate stability effects tied to singularities when a classical Beta map is used as the nonlinearity. Section 4 describes implementation; Sections 5–6 report metrics, figures, and conclusions.

# 2 Special Functions

Special functions are fundamental tools in various branches of science and engineering, including mathematics, physics, geometry, astronomy, biology, statistics and other fields (see [2, 5, 7, 8]) . In mathematics a special function is the function of a real or complex valued function. Special functions comprise a wide array of advanced mathematical functions, including transcendental functions—such as Bessel functions—and algebraic functions, such as various orthogonal polynomials. Alongside elementary functions like $x^n$, $e^x$, $\log x$, and $\sin x$, special functions including the Bessel, Gamma, Hypergeometric, Legendre, and Laguerre functions find extensive application in applied sciences. These functions are indispensable in many branches of mathematical physics and frequently appear in problems arising from mechanics, engineering, and various fields of applied science. The concept of representing special functions through series expansions dates back to the early 18th century. In 1703, James Bernoulli employed an infinite series to solve a differential equation—an early instance leading to the development of what we now recognize as Bessel functions. Although mathematicians such as Euler investigated these functions in the 17th century, a comprehensive and systematic study of Bessel functions did not emerge until 1824. During the 18th century, several major advancements were made. Among these were the development of the Gamma function and the theory of elliptic integrals. Around 1730, Leonhard Euler discovered many of the key properties of the Gamma function. In 1772, Euler extended his earlier work by evaluating the integral form of the Beta function, thereby establishing a strong and lasting connection between the Beta and Gamma functions. This breakthrough laid the foundation for deeper explorations into the nature and applications of special functions. Mathematical special functions, such as the Beta and Gamma functions, form a fascinating and essential area of study in both pure and applied mathematics.

While many of these functions have been investigated for centuries, the field continues to evolve with the development of new functions aimed at solving emerging mathematical problems. These special functions act as fundamental tools for constructing more complex mathematical models and theories. Among them, the Beta and Gamma functions stand out due to their rich properties and wide-ranging applicability. The Beta function, first examined by Euler, and the Gamma function—an elegant extension of the factorial to real and complex numbers—are central to numerous areas in analysis, number theory, probability, and mathematical physics.

### 2.1 Minimal Background: The Beta Function

For positive parameters $\alpha$ and $\beta$, the Beta function may be written as [1]

$$B(\alpha, \beta) = \int_0^1 t^{\alpha-1}(1-t)^{\beta-1}\, dt, \tag{2.1}$$

with symmetry $B(\alpha, \beta) = B(\beta, \alpha)$. It satisfies

$$B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}, \quad \alpha > 0,\ \beta > 0, \tag{2.2}$$

which is used for stable evaluation alongside the Gamma function. The Beta distribution on $[0, 1]$ uses $B(\alpha, \beta)$ as a normalizing constant; that probabilistic role is distinct from using $B(\cdot, \cdot)$ as an activation, but it explains why the same function appears in statistics and in machine learning [3].

## 3 Neural Networks and Activation Functions

Artificial neural networks (ANNs) [9] combine weighted sums of inputs with nonlinear activations so that hidden layers can approximate nonlinear maps. Figure 1 sketches a single computational unit: inputs $x_i$ are weighted, summed with a bias, and passed through an activation $\varphi(\cdot)$. In this article, the scalar case $x \in \mathbb{R}$ is used; each hidden unit forms one pre-activation $z_j$ affine in $x$ and applies $h_j = B(z_j, \beta_0)$ with fixed $\beta_0 = 2$, as in Section 4, followed by a linear readout. Biologically, the human brain contains billions of neurons, each composed of four primary components: dendrites, axon, soma (cell body), and synapses. Dendrites receive input signals, which are then aggregated by the soma. When the cumulative signal exceeds a certain threshold, it is transmitted through the axon to other neurons. Synapses control the strength of these inter-neuronal connections. In artificial models, dendrites, soma, and axons are analogously represented using activation functions and weighted inputs. This abstraction enables the construction of models capable of learning from data. The foundational mathematical model of a biological neuron was proposed by McCulloch and Pitts; the basic architecture of an artificial neuron is illustrated in Figure 1.

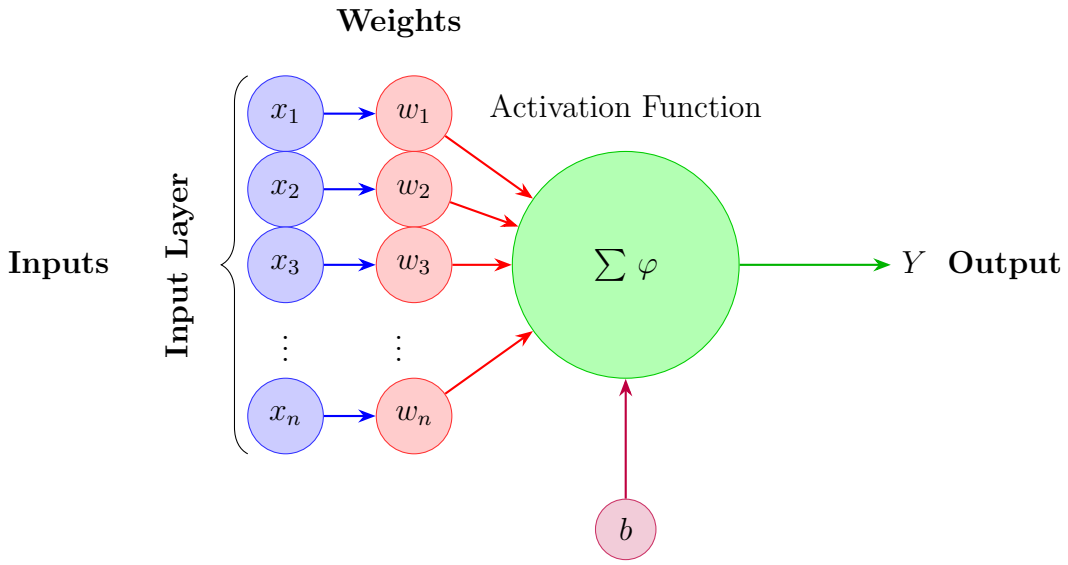



Figure 1: Basic structure of an artificial neuron [9].

# 4 Methodology

This section presents the methodology adopted in the study, including the data preparation and model development procedures, along with the main results and the corresponding predictions obtained from the proposed model. The performance and effectiveness of the proposed approach are also discussed in detail.

## 4.1 Mathematical formulation of the network

We approximate a target $y = f_{\text{true}}(x)$ on $x \in [-5, 5]$ with a network that maps a scalar input to a scalar output. Let the hidden layer have $m = 64$ units. For each unit $j = 1, \ldots, m$, a single affine pre-activation is formed from the input $x$,

$$z_j = w_j^{(1)} x + b_j^{(1)}. \tag{4.1}$$

The second argument of the Beta function is held fixed at $\beta_0 = 2$ (not learned). In the *Safe* variant, the first argument is clamped componentwise to $[\varepsilon_1, \varepsilon_2] = [0.1,\ 10.0]$ before evaluation; we write $\tilde{z}_j = \text{clamp}(z_j)$. In the *Unsafe* variant, $z_j$ is passed to $B$ without clamping. The hidden activation is

$$h_j(x) = B\big(\tilde{z}_j, \beta_0\big) \quad \text{(Safe)}, \qquad h_j(x) = B\big(z_j, \beta_0\big) \quad \text{(Unsafe)}, \qquad \beta_0 = 2. \tag{4.2}$$

The output layer is linear,

$$\hat{y}(x) = \sum_{j=1}^{m} w_j^{(2)}\, h_j(x) + b^{(2)}. \tag{4.3}$$

The training loss over $N$ samples is the mean squared error

$$L = \frac{1}{N} \sum_{i=1}^{N} \big(y_i - \hat{y}(x_i)\big)^2. \tag{4.4}$$

## 4.2 Derivatives and autograd in the implementation

The partial derivatives of $B(\alpha, \beta)$ with respect to its first argument (at fixed $\beta = \beta_0$) follow from (2.2). For $\alpha, \beta > 0$,

$$\frac{\partial B}{\partial \alpha} = B(\alpha, \beta)\big(\psi(\alpha) - \psi(\alpha + \beta)\big), \qquad \frac{\partial B}{\partial \beta} = B(\alpha, \beta)\big(\psi(\beta) - \psi(\alpha + \beta)\big), \tag{4.5}$$

where $\psi$ is the digamma function. With $\beta = \beta_0$ fixed, only $\partial B/\partial \alpha$ would enter the chain rule $\partial h_j/\partial z_j = (\partial B/\partial \alpha)|_{(\alpha,\beta)=(z_j,\beta_0)}$ if the map were fully differentiable end-to-end. When the Safe variant clamps $z_j$, the derivative of clamp is zero on the saturation segments and one in the interior, which caps $\partial h_j/\partial z_j$ in the idealized differentiable setting.

In the actual code, the Beta values are computed with SciPy's `scipy.special.beta` on NumPy arrays built from `x.detach().cpu().numpy()`, so the activation tensor is disconnected from the autograd graph that begins at the first linear layer. Consequently, `loss.backward()` does *not* propagate gradients through $B(z_j, \beta_0)$ to the weights $w_j^{(1)}$ or biases $b_j^{(1)}$: those parameters receive zero gradient from the loss. Only the readout layer in (4.3) (weights $w_j^{(2)}$ and bias $b^{(2)}$) is updated by Adam. The formulas (4.5) remain useful to interpret how $h_j$ depends on $z_j$ in the *forward* pass and why Safe versus Unsafe pre-activations produce different feature vectors $\mathbf{h}(x)$, which drives different readout-layer optimization; reported $\|\nabla_\theta L\|_2$ norms therefore reflect gradients with respect to the trainable subset of $\theta$, chiefly the output layer.

## 4.3 Implementation summary

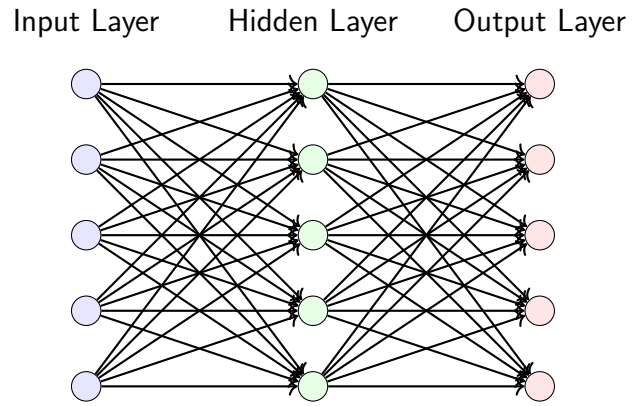
- **Objective:** We trained the shallow network above to compare the *Safe* (clamped first argument) and *Unsafe* (unclamped) evaluation of $B(z, \beta_0)$ with $\beta_0 = 2$ in the hidden layer. The synthetic targets are smooth on a bounded interval; the design isolates how forward stability of $B(z, \beta_0)$ affects the hidden features passed to the readout layer.

- **Beta activation in code:**

  - The activation is implemented as a subclass of `torch.nn.Module`. The forward pass applies `torch.clamp` to the hidden pre-activation tensor when `safe=True`, then evaluates `scipy.special.beta(x_np, 2.0)` so the second argument is fixed at 2. Tensors are converted to NumPy after `.detach()`, so the Beta is not differentiated by PyTorch.

  - Two variants differ only in whether the pre-activation is clamped to [0.1, 10.0] before $B(z, \beta_0)$:
    - *Safe:* clamp then `beta(x, 2.0)`.
    - *Unsafe:* no clamp; `beta(x, 2.0)` on raw linear outputs $z_j$.

- **Model architecture:** The architecture has one input node, one hidden layer of 64 neurons with the Beta activation described above, and one linear output. The same layout and parameter counts are used for both Safe and Unsafe runs so that differences are attributable to singularity handling.




- **Dataset:** For each target $f_{\text{true}}$ we form a uniform grid $\{x_i\}_{i=1}^{1000} \subset [-5, 5]$ and set $y_i = f_{\text{true}}(x_i)$. The primary experiment uses $f_{\text{true}}(x) = \sin x$; we then repeat the same training protocol for $f_{\text{true}}(x) = \cos x$, $x \sin x$, $\mathrm{e}^{-x^2}$, and $\tanh x$ to check that the comparison between Safe and Unsafe activations is not an artifact of a single target. The index set $\{1, \ldots, 1000\}$ is partitioned randomly into training indices $\mathcal{I}_{\text{tr}}$ ($|\mathcal{I}_{\text{tr}}| = 800$) and test indices $\mathcal{I}_{\text{te}}$ ($|\mathcal{I}_{\text{te}}| = 200$). All losses reported on the training stream use only $i \in \mathcal{I}_{\text{tr}}$ in (4.4); test MSE uses $i \in \mathcal{I}_{\text{te}}$.

- **Training setup:**
  - Optimizer: Adam (first and second moment estimates of $\nabla_\theta L$, with default-style hyperparameters as in the implementation).
  - Loss function: mean squared error, i.e. (4.4) restricted to the training split each epoch.
  - Epochs: 500 full passes over the training set.
  - Metrics tracked each epoch: value of $L$ on the training batch, $\|\nabla_\theta L\|_2$, and $\|\theta\|_2$ aggregated over weights.

- **Gradient and weight norms:** To connect optimization trajectories to the analysis in Section 5, we record the Euclidean norm of the full gradient vector, $\|\nabla_\theta L\|_2 = \left(\sum_k (\partial L/\partial \theta_k)^2\right)^{1/2}$, and the Euclidean norm of all network weights (or of the trainable subset). Large $\|\nabla_\theta L\|_2$ on the readout parameters indicates steep loss sensitivity there when hidden features $\mathbf{h}$ differ between Safe and Unsafe runs; monitoring $\|\theta\|_2$ shows whether the output layer scales compensate for poor conditioning of the forward Beta map.
  - L2 norm of all parameter gradients calculated each epoch.
  - L2 norm of model weights calculated to monitor parameter stability.

- **Evaluation Metrics:**
  - Final MSE on the test set.
  - Relative difference between predictions from safe and unsafe models.
- **Visualization:**
  - Predicted vs. Actual outputs.
  - Loss curves over epochs.
  - Gradient norm trends and detection of vanishing/exploding gradients.

# 5 Results and Analysis

This section interprets the numerical outcome of minimizing (4.4) for the two activations. Formally, at each epoch we evaluate the same loss functional on the training indices; after training we report the analogue on the test indices. The learned map depends on readout weights $\theta$ that receive gradients (the first linear layer is fixed under autograd because the Beta is evaluated on detached tensors; see Section 4). We emphasize qualitative agreement across targets: the Unsafe forward map uses $B(z_j, \beta_0)$ with unclamped $z_j$, so some $h_j$ can take extreme values compared with the Safe map; the readout layer then fits a harder regression, often with larger loss and less stable optimization traces, whereas the Safe path keeps $z_j$ in $[\varepsilon_1, \varepsilon_2]$ and yields smoother feature vectors $\mathbf{h}(x)$.
Two scalar summaries are used throughout: mean squared error on the training stream (often called training loss when plotted against epoch) and mean squared error on the held-out test points.

**Training loss.** Let $\mathcal{I}_{\mathrm{tr}}$ be the training index set. After each epoch,

$$L_{\mathrm{tr}} = \frac{1}{|\mathcal{I}_{\mathrm{tr}}|} \sum_{i \in \mathcal{I}_{\mathrm{tr}}} \big(y_i - \hat{y}(x_i)\big)^2,$$

which is the empirical risk (4.4) restricted to training data. Lower $L_{\mathrm{tr}}$ indicates smaller average squared residual on the sample used for updates.

**Test mean squared error.** Let $\mathcal{I}_{\mathrm{te}}$ denote the test indices. The test MSE is

$$\mathrm{MSE}_{\mathrm{te}} = \frac{1}{|\mathcal{I}_{\mathrm{te}}|} \sum_{i \in \mathcal{I}_{\mathrm{te}}} \big(y_i - \hat{y}(x_i)\big)^2.$$

Lower $\mathrm{MSE}_{\mathrm{te}}$ indicates better agreement between $f_{\mathrm{true}}$ and the learned $F(\cdot; \theta^*)$ on unseen $x$ in $[-5, 5]$.

## 5.1 Weight norms and gradient norms

Beyond $L_{\mathrm{tr}}$ and $\mathrm{MSE}_{\mathrm{te}}$, we inspect norms of $\theta$ and $\nabla_\theta L$ for the parameters that are actually updated (primarily the readout layer). Because the Beta is computed outside autograd, these norms do not reflect a full chain rule through (4.2) into $w_j^{(1)}$; they summarize optimization of the readout given different hidden features $\mathbf{h}$ produced by Safe versus Unsafe forward passes. We report the same Euclidean norms for **Beta (Safe)** and **Beta (With Singularity)**.

**Weight norms.** Let $\theta$ collect all scalar parameters. Its Euclidean norm $\|\theta\|_2$ measures overall parameter magnitude. In poorly conditioned activations, optimizers sometimes inflate $\|\theta\|_2$ to compensate for small effective gains per step.

$$\|\theta\|_2 = \sqrt{\sum_k \theta_k^2}$$

Figure 2 shows the evolution of $\|\theta\|_2$ over 500 epochs (all parameters are plotted, but only readout weights are updated by the loss). The Beta (Safe) model begins with a slightly higher norm (approximately 6.64) and demonstrates a steady, slow decay. The Beta (With Singularity) model shows a different trajectory, consistent with the readout layer adapting to different hidden features $B(z_j, \beta_0)$.

**Gradient norms.** The size of the gradient with respect to trainable parameters summarizes how strongly the loss responds to perturbations of the readout weights (and any other parameters that remain linked to the graph). Large $\|\nabla_\theta L\|_2$ in early epochs can signal steep loss landscapes when Unsafe forward features $\mathbf{h}$ are poorly scaled relative to the target.

$$\|\nabla_\theta L\|_2 = \sqrt{\sum_k \left(\frac{\partial L}{\partial \theta_k}\right)^2}$$

The gradient norm behavior, presented in Figure 3, compares $\|\nabla_\theta L\|_2$ for parameters that receive gradients (primarily the readout). The Beta (Safe) variant maintains smaller norms throughout training. The Beta (With Singularity) can exhibit much larger norms early on (e.g. peaks over $5 \times 10^6$ in our log), reflecting a harder least-squares problem for the readout when Unsafe forward features $\mathbf{h}$ are poorly scaled. Training loss and test MSE remain worse for Unsafe despite partial stabilization of norms in later epochs.

**Discussion:** This analysis confirms that the Beta (Safe) forward map yields more stable readout-layer optimization than the Unsafe map. The Unsafe path feeds $B(z_j, \beta_0)$ without clamping $z_j$, which harms fitting efficiency and final MSE under the same architecture.

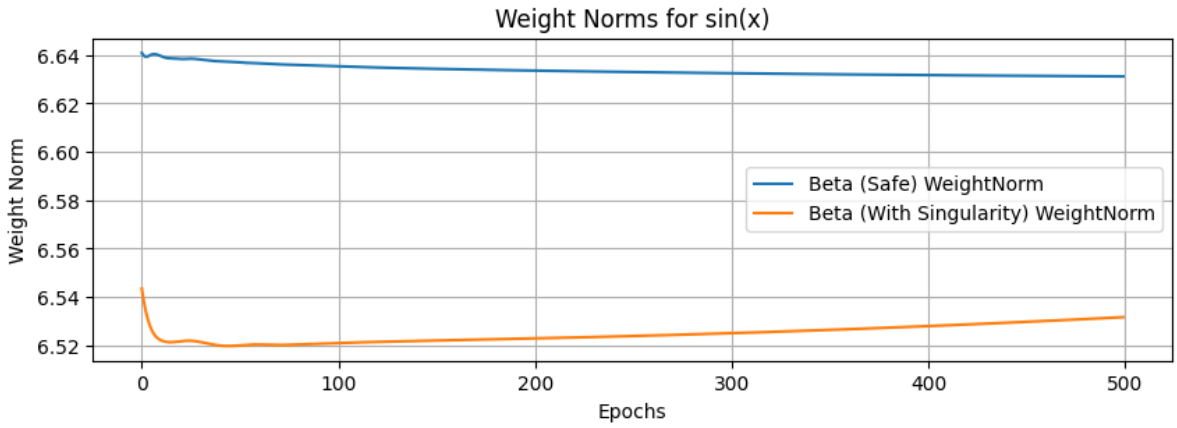



Figure 2: Weight Norms during training on $\sin(x)$ for Beta (Safe) and Beta (With Singularity).

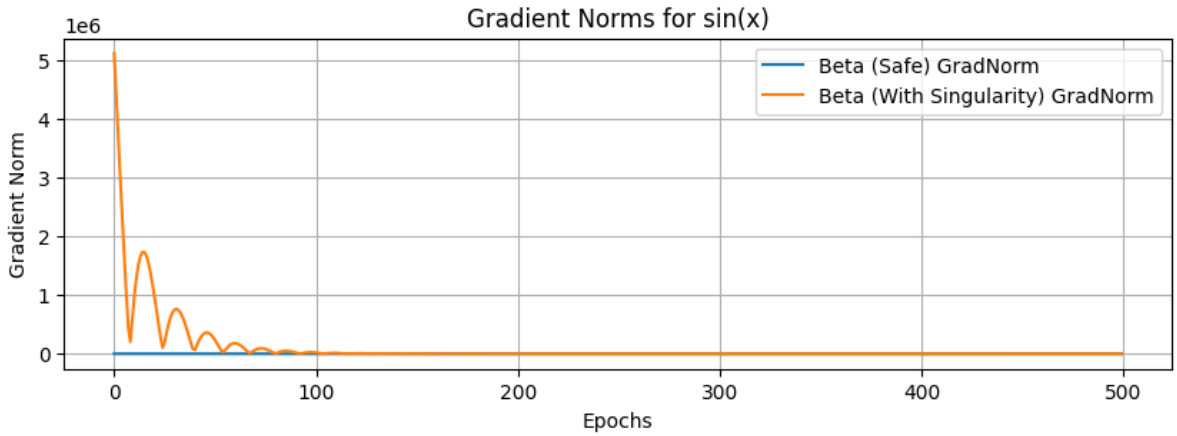



Figure 3: Gradient Norms during training on $\sin(x)$ for Beta (Safe) and Beta (With Singularity).

## 5.2 Our Result

Our setup of training the neural network with Unsafe version of Beta Function results in:

- Higher initial training loss and slower convergence.
- Significantly worse MSE on validation data.
- Unstable and large gradient norms during early training.
- Larger weight magnitudes as the model compensates for instability.

After analyzing these things, we can conclude that handling singularities in activation functions is crucial for stable and effective learning.

## 5.3 Training performance

Table 1 lists $L_{\text{tr}}$ at selected epochs for $f_{\text{true}}(x) = \sin x$. The Safe variant decreases monotonically in the table entries and reaches order $10^{-3}$ by epoch 400, whereas the Unsafe variant still has $L_{\text{tr}} \approx 0.14$ at the same depth—consistent with the gradient-norm behaviour and slower readout-layer fitting when Unsafe forward values $B(z_j, \beta_0)$ are poorly conditioned.

| Epoch | Beta (Safe) | Beta (With Singularity) |
|---|---|---|
| 0 | 15.8402 | 220511.4219 |
| 100 | 0.0314 | 4.1955 |
| 200 | 0.0153 | 0.1786 |
| 300 | 0.0090 | 0.1555 |
| 400 | 0.0054 | 0.1391 |

Table 1: Training loss $L_{\text{tr}}$ at selected epochs for $f_{\text{true}}(x) = \sin x$.

## 5.4 Mean Squared Error (MSE)

After training, we evaluated the models using the Mean Squared Error (MSE) metric:

- **Beta (Safe)**: MSE = 0.0035
- **Beta (With Singularity)**: MSE = 5.1097

## 5.5 Observations

For $f_{\text{true}}(x) = \sin x$, the two networks differ only in whether the pre-activations $z_j$ are clamped before $B(z_j, \beta_0)$. The quantitative outcome is consistent with Section 4: the Unsafe map can feed $B(\cdot, \beta_0)$ with $z_j$ outside $[\varepsilon_1, \varepsilon_2]$, producing forward features $\mathbf{h}$ that are harder for the readout to match to $\sin x$, and optimization traces (Figure 3) reflect that mismatch. The Safe map restricts $z_j$ before evaluation, yielding better-scaled $h_j$ and lower $\text{MSE}_{\text{te}}$ on the same split, which matches Figure 4.

The results indicate a substantial degradation in both convergence speed and final accuracy when singularities are present in the activation. The *Beta (Safe)* configuration achieves rapid decay of $L_{\text{tr}}$ and low $\text{MSE}_{\text{te}}$, whereas *Beta (With Singularity)* begins with extremely large $L_{\text{tr}}$ and settles at a much higher test error. This aligns with the view that poorly conditioned first arguments $z_j$ to $B(z_j, \beta_0)$ disrupt stable fitting at the readout layer, which is particularly damaging for smooth regression targets such as $\sin x$ on $[-5, 5]$.

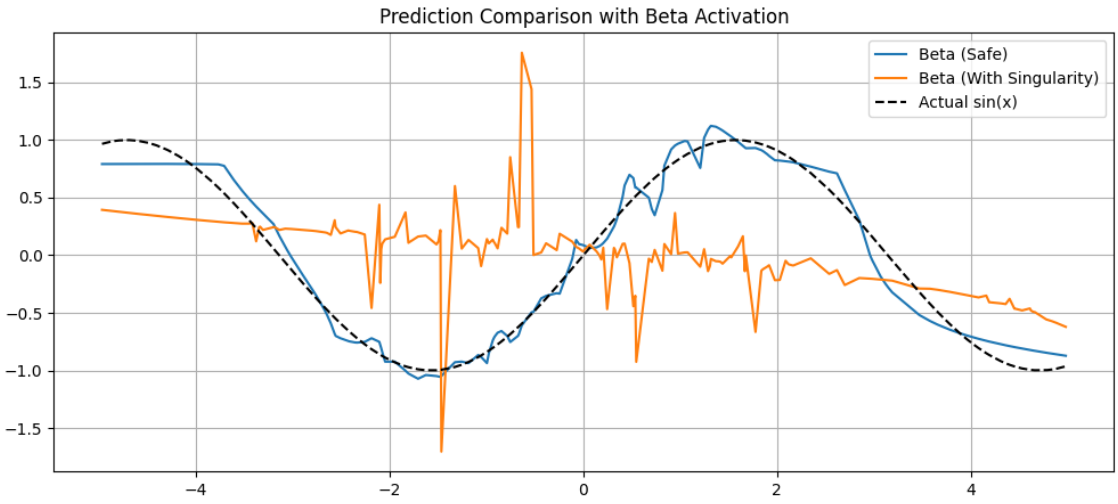
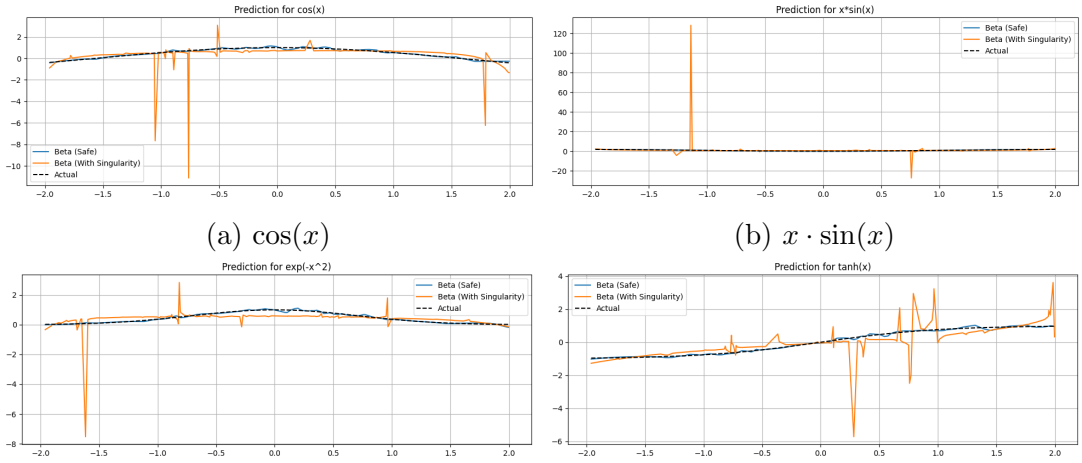



Figure 4: This is a chart showing the prediction chart of neural networks to approximate $\sin(x)$. The dashed line is the actual values, the blue line that closely traces the dashed line is the prediction made by the neural network that avoid singular points in its activation function, while the orange line that spreads in different direction and does not tract the actual output in any way is the neural networks prediction where we used singular points.

After $\sin(x)$, we tried implementing the same activation function on these functions, and got almost the same result:

- $\cos(x)$
- $x \cdot \sin(x)$
- $\exp(-x^2)$
- $\tanh(x)$

In each case the qualitative picture matches the $\sin x$ experiment: the Safe activation keeps $z_j$ inside the clamp interval before $B(z_j, \beta_0)$, whereas the Unsafe activation does not, leading to less favorable hidden features for the same readout architecture. The *Beta (Safe)* network therefore attains lower test error and cleaner predicted curves (Figure 5), reinforcing that the phenomenon is not tied to a single choice of $f_{\text{true}}$.




(a) $\cos(x)$ (b) $x \cdot \sin(x)$

(c) $\exp(-x^2)$ (d) $\tanh(x)$

Figure 5: Prediction performance of neural networks on various functions using Beta Function based activations with and without singularity handling.

As shown in Figure 4 and Figure 5, the network with the Unsafe activation produces hidden features $B(z_j, \beta_0)$ that are harder for the readout to combine into the target, whereas clamping $z_j$ before $B(z_j, \beta_0)$ yields better-scaled features and more accurate predictions.

# 6 Conclusion

In this work, we compared $B(z, \beta_0)$ with $\beta_0 = 2$ fixed, with and without clamping the pre-activation $z$, on synthetic regression tasks. The Safe variant produces more favorable hidden features for the linear readout and lower test MSE in our runs.

**Limitations.** The experiments are limited to synthetic one-dimensional targets and a single hidden layer. The implementation uses SciPy for $B$ on detached tensors, so the first linear layer does not receive loss gradients; a fully differentiable Beta activation in PyTorch would allow joint training of all layers. We do not claim superiority over common activations on large-scale or real-world tasks. Domain-specific applications would require separate validation.

**Funding:** This research was carried out independently, without any financial support from institutions or funding bodies, ensuring unbiased results and freedom from external influence.

**Data Availability Statement:** The datasets used in this study are available from the corresponding author upon reasonable request.

**Conflicts of Interest:** The author confirms that there are no conflicts of interest.

# References


[1] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions: With Formulas, Graphs, and Mathematical Tables*, Dover Publications, New York, 1972.

[2] Arakaparampil Mathai Mathai and Hansjoachim Haubold, *Special Functions for Applied Scientists*. Springer, 2008.

[3] Ş. Çelik and M. Korkmaz, Beta distribution and inferences about the beta functions, *Asian Journal of Science and Technology*, **7**(5), 2960–2970, May 2016.

[4] D.K. Shah, V.A. Vyawahare and S. Sadanand, Artificial neural network approximation of special functions: design, analysis and implementation, *International Journal of Dynamics and Control*, (2025).

[5] Milton Abramowitz and Irene A. Stegun, *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. 9th edn, Dover, New York, 1964.

[6] T. Nitta, Learning dynamics of the complex-valued neural network in the neighborhood of singular points, *Journal of Computer and Communications*, **2**, 27–32, 2014.

[7] Anatolij F. Nikiforov and Vasilij B. Uvarov, *Special Functions of Mathematical Physics: A Unified Introduction with Applications*. Springer, Birkhäuser Boston, 2013.

[8] Larry C. Andrews, *Special Functions of Mathematics for Engineers*. Oxford Science Publications, SPIE Optical Engineering Press, Bellingham, 1998.

[9] James M. Zurada, *Introduction to Artificial Neural Systems*. West Publishing Company, St. Paul, 1992.