Learnable Numerical Input Normalization for Tabular Representation Learning based on B-splines

Min-Kook Suh, Moonjung Eo, Ye Seul Sim, Woohyung Lim
LG AI Research
Seoul, Republic of Korea
{minkook.suh,moonj,ysl.sim,w.lim}@lgresearch.ai

Abstract

Input normalization of numerical features is essential for improving the performance and training stability of neural networks. This is particularly important in tabular deep learning, where the heterogeneity of features requires different normalization for each input feature. While various normalization techniques have been proposed, they are often specialized for specific input feature distributions and, more importantly, are not optimized for neural network training. In this paper, we propose a learnable input normalization technique based on B-splines, which provides a flexible and differentiable curve-based transformation for each feature. A key characteristic of our method is its novel loss function, which adapts the transformation based on the difficulty of each data point, thereby boosting model performance on challenging samples. We evaluate the proposed method on OpenML datasets using several popular neural network architectures, demonstrating that the learnable normalization consistently outperforms conventional techniques.

1 Introduction

Input normalization plays a crucial role in improving both model performance and training stability. This is especially important for tabular data, which is ubiquitous across many industries [4, 6, 12]. A key characteristic of tabular data is its highly heterogeneous nature, with features often having different scales and distributions. Therefore, applying tailored normalization to each input feature is essential. Unlike image or text data, where input features are typically more uniform, tabular data requires more precise normalization strategies to effectively handle its diverse feature distributions. Several input normalization techniques have been proposed, ranging from simple methods like min-max transformation to more complex approaches like quantile transformation. However, these methods often fail to account for the training procedure of neural network learning.

Recent research in tabular deep learning tends to rely on conventional normalization techniques, with limited efforts focused on developing novel methods. For instance, [10] employs quantile transformation across all cases, while [11] applies a min-max transformation. While selecting an existing method simplifies the process, identifying the best one among different normalization techniques can be time-consuming, and even then, the selected method may not guarantee optimal performance. An alternative approach proposed by [3] seeks to choose between standardization and quantile transformation based on the magnitude of low-frequency components using spectral analysis. However, this method does not introduce a new normalization technique, but rather focuses on selecting among pre-existing ones.

In this paper, we propose a novel method that parameterizes the transformation as a curve and optimizes it alongside neural network training. Since input normalization is applied to each feature individually, it can be viewed as a one-dimensional curve-fitting problem. Specifically, we utilize

Table Representation Learning Workshop at NeurIPS 2024.

B-splines to model these transformations. B-splines are particularly well-suited for this task due to their flexibility in modeling, as a B-spline of order $n + 1$ can approximate any n times differentiable curve by adjusting knot vectors and control points. Moreover, B-splines can control both the start and end points of the curve, making them ideal for input normalization, which maps arbitrary inputs with varying ranges to a pre-defined range.

Additionally, we introduce a loss function designed to make learning easier for challenging input ranges while compressing those that have already been learned. Specifically, input normalization can be seen as the task of distributing a limited range of normalized input across the raw input. For data points that are difficult to learn, assigning a wider range in the normalized input allows the network to allocate more capacity to these points, thereby facilitating easier learning. In contrast, a smaller range is allocated for data points that are easier to learn. This is achieved by increasing the derivative of the curve for data points with higher loss values, ensuring that a small region in the raw input corresponds to a larger region in the normalized input. Conversely, for data points with lower loss values, the curve's derivative decreases, allowing a larger region in the raw input to be mapped to a smaller region in the normalized input.

Main contributions of this paper are summarized as follows.

- We propose a learnable input normalization technique based on B-splines, which enhances neural network training efficiency for tabular data. By parameterizing the input transformation as a curve and optimizing it alongside network training, our method enables adaptive normalization for each feature, addressing the heterogeneity of tabular data.
- We introduce a novel loss function that dynamically adjusts the B-spline parameters based on the difficulty of each data point. This allows the network to allocate more capacity to challenging data points, ensuring an optimized transformation for both difficult and easily learnable data points.

2 Related Work

Despite its importance, input normalization in tabular deep learning has been largely overlooked. In many cases, researchers have simply opted to use one of the existing normalization methods without exploring new alternatives. For instance, [10], which is a benchmark study comparing the performance of various neural networks and gradient-boosted decision trees, exclusively uses the quantile transform. Similarly, [15], another benchmark study evaluating neural networks and gradient-boosted decision trees, utilizes quantile transformation and identity mapping. In addition, [11], which assesses the performance of various anomaly detection algorithms on tabular data, applies the min-max transform. [2, 8, 9, 13, 14, 16] also simply used one of identity mapping, standardization, and quantile transform.

A closely related work to our approach is [3], which emphasizes the importance of input normalization in tabular deep learning. Their approach leverages the fact that neural networks are particularly well-suited to learning smooth functions [17, 19]. In particular, they assume that input normalization techniques which produce smoother outputs lead to better performance in tabular neural networks and analyzed the correlation between the smoothness and performance. Based on this assumption, their method, named "selrank", selects between standardization and quantile transformation using spectral analysis. However, this technique still relies on conventional normalization methods, without introducing a new normalization approach specifically tailored for tabular data.

3 Proposed Method

3.1 Preliminary: B-splines

B-splines are defined by their order ($n+1$), knot vector (T), and control points (P). The output of a B-spline is calculated as a weighted sum of control points, with weights determined by the knot vector and the order of the spline. Specifically, a B-spline of order-$(n+1)$ is defined as:

$$B_{i,0}(x) := \begin{cases} 1, \text{if } t_i \leq x < t_{i+1} \\ 0, \text{otherwise} \end{cases} \tag{1}$$

$$B_{i,k}(x) := \frac{x - t_i}{t_{i+k} - t_i} B_{i,k-1}(x) + \frac{t_{i+k+1} - x}{t_{i+k+1} - t_{i+1}} B_{i+1,k-1}(x) \tag{2}$$

$$C(x) := \sum_i p_i B_{i,n}(x) \tag{3}$$

where t_i is the i-th element of T, p_i is the i-th element of P, and $C(x)$ denotes the output of the B-spline at input x.

Similarly, the derivative of a B-spline, which is crucial for understanding how the transformation adjusts across the input range, is defined as:

$$D_{i,0}(x) := \begin{cases} 1, \text{if } t_i \leq x < t_{i_1} \\ 0, \text{otherwise} \end{cases} \tag{4}$$

$$D_{i,k}(x) := \frac{1}{t_{i+k} - t_i} B_{i,k-1}(x) - \frac{1}{t_{i+k+1} - t_{i+1}} B_{i+1,k-1}(x) \tag{5}$$

$$+ \frac{x - t_i}{t_{i+k} - t_i} D_{i,k-1}(x) + \frac{t_{i+k+1} - x}{t_{i+k+1} - t_{i+1}} D_{i+1,k-1}(x) \tag{6}$$

$$\frac{dC}{dx}(x) := \sum_i p_i D_{i,n}(x) \tag{7}$$

3.2 Learnable Input Normalization based on B-splines

Input normalization can be viewed as a transformation function that maps raw input values to a predefined normalized range. In our approach, this transformation is represented as a parameterized B-spline, where the order $n+1$ is fixed as a hyperparameter, and the knot vector T and control points P are learned.

The knot vector must be non-decreasing due to the definition of B-splines, where each subsequent knot value must be greater than or equal to the previous one to ensure the continuity and smoothness of the spline. Similarly, the control points must also be non-decreasing to ensure that the input normalization function remains one-to-one, meaning that each unique input maps to a unique output without reversing the order of the input values. These constraints are enforced through a combination of softmax and cumulative sum (cumsum) functions, ensuring that both the knot vector and control points are non-decreasing while maintaining differentiability for learning. Specifically, they are expressed as:

$$T = [0; \text{cumsum}(\text{softmax}(T_{\text{logit}})] \tag{8}$$

$$P = [0; \text{cumsum}(\text{softmax}(P_{\text{logit}})] \tag{9}$$

However, as Eq. 8 restricts the knot vector to values between 0 and 1, we apply a simple min-max transformation to the raw input data before passing it through the B-spline. The final input normalization proposed in this paper is then given by:

$$y = C(M(x); n, T, P) \tag{10}$$

where $M(x)$ represents the min-max transformation applied to the raw input. The parameters T_{logit} and P_{logit} are learned through backpropagation.

3.3 Loss Function to Optimize B-Spline Parameters

Our loss function is designed to account for the difficulty of learning across different input ranges. In this formulation, the derivative $\frac{dC}{dx}$ indicates how much the normalized input stretches or compresses relative to the raw input. By increasing the derivative for data points with higher loss values, the network is able to allocate more capacity to difficult samples, making them easier to learn. Specifically, the loss is expressed as:

$$L_{spline}(x) = \frac{L(f(C(M(x); n, T, P)), y)}{\frac{dC}{dx}(M(x); n, T, P)} \tag{11}$$

where f represents the neural network, x is the raw input, and y is the corresponding training target for x. $L(f(C(M(x); n, T, P)), y)$ represents the training loss for x. This formulation enables the adjustment of the B-spline parameters T and P, ensuring that the model allocate more capacity to challenging data points, thereby improving learning efficiency.

3.4 Implementation Details

Incorporating a new loss function into the existing training process introduces the challenge of determining the appropriate balance between the original loss and the new loss, which requires tuning an additional hyperparameter. Additionally, it is crucial to ensure that the newly introduced loss function does not interfere with the learning of the network's original parameters. To avoid these issues, we utilize PyTorch's .detach() function, ensuring that the proposed loss does not interfere with the core learning process of the network:

$$L^{(d)}_{spline}(x) = \frac{L(f(C(M(x); n, T, P)), y).\text{detach()}}{\frac{dC}{dx}(M(x); n, T, P)} \tag{12}$$

$$L_{total} = L(f(C(M(x); n, T, P).\text{detach()}), y) + L^{(d)}_{spline}(x) \tag{13}$$

This guarantees that the primary loss trains only the network, while the spline loss exclusively updates the B-spline parameters. Another benefit of this approach is that allows the use of different optimization algorithms for updating the network and spline parameters independently. For the B-spline parameters, we specifically used the Adam optimizer with a fixed learning rate of 0.01, ensuring stable updates to the spline. The network parameters can be optimized using other suitable algorithms based on the task.

In addition to the proposed loss function and its implementation details, there are three important hyperparameters for the proposed learnable normalization.

1. Spline order $n + 1$: we set $n = 2$ to ensure that the transformation is differentiable across all regions.
2. Number of control points: it was empirically set to 16.
3. Min-max transformation margin δ: the knots generated using Eq. 8 cannot express the values 0 and 1, meaning the raw input's minimum and maximum values are not directly represented. To address this, a margin δ is applied to the min-max transformation, ensuring that the raw data falls within the range $[\delta, 1 - \delta]$ after transformation. We set $\delta = 0.1$.

Finally, to address the issue of out-of-range values in the test set when the min-max transformation is learned on the training data, we apply linear extrapolation to the B-spline for values outside the $[0, 1]$ range. This ensures that the model can handle inputs beyond the learned range in a linear manner.

More implementation details can be found in Appendix A.1.

4 Experiments

4.1 Experimental Settings

We conducted experiments on 26 diverse datasets from OpenML [18], selecting datasets based on the numbers of numerical features, dataset sizes, and target tasks. To evaluate the performance of the proposed input normalization method, we applied it to several neural network architectures

Table 1: Averaged normalized score of input normalization methods

Method	MLP	ResNet	FT-Transformer	T2G-Former
Quantile	0.398	<u>0.584</u>	0.714	0.703
Standardization	0.453	0.512	<u>0.716</u>	**0.712**
Min-max	**0.536**	0.562	0.683	<u>0.711</u>
Proposed	<u>0.524</u>	**0.636**	**0.745**	0.689

commonly used in tabular deep learning: MLP, ResNet, FT-Transformer [8], and T2G-Former [20]. Each network was trained using four different input normalization techniques: min-max scaling, standardization, quantile transformation, and our proposed learnable input normalization.

We evaluated which normalization method performed best on average for each network, measured by averaging their impact on model performance across multiple datasets. To fairly assess the performance of different normalization techniques, a rigorous hyperparameter search was essential. We conducted the hyperparameter search using Optuna [1]. The evaluation procedure consisted of the following steps:

1. **Initial hyperparameter search**: First, we applied quantile transformation for input normalization and conducted a thorough hyperparameter search for each neural network and dataset. During this step, we determined network-specific hyperparameters such as number of layers and hidden units.
2. **Normalization-specific hyperparameter tuning**: After determining the network-specific hyperparameters, we performed a separate hyperparameter search for training-related parameters (e.g., learning rate and weight decay) for each input normalization technique.
3. **Network training**: For each set of hyperparameters found, we repeated the training five times to ensure consistent evaluation.
4. **Repeated training**: Steps 2 to 3 were repeated for five times with different hyperparameter configurations, resulting in a total of 25 performance measurements per normalization technique.

Overall, our experiments involved of evaluating 4 normalization techniques across 26 datasets with 25 repeated training runs per configuration, yielding a total of 2600 individual evaluations.

To compare the performance across different datasets, we employed a normalized performance metric, where the highest achieved score on a given dataset was scaled to 1, and the lowest score to 0. This approach allowed us to average the performance of the normalization techniques across datasets with varying target tasks and levels of difficulty.

4.2 Quantitative Results

Table 1 presents the performance comparison of different input normalization techniques across various neural network architectures. The results highlight that the most effective normalization method depends on the network structure.

The performance of existing normalization methods, such as quantile, standardization, and min-max transform, varies depending on the network architecture. The most effective normalization method differs by architecture. For simpler architectures like MLP and ResNet, min-max transformation consistently provides better results, while for transformer-based architectures like FT-Transformer and T2G-Former, the quantile transformation tends to outperform other methods. Standardization generally achieved performance between that of min-max and quantile transformations across most architectures.

Our proposed learnable input normalization demonstrates robust and stable performance across all architectures. It consistently surpasses the performance of conventional methods or achieves results on par with the best-performing normalization method in each case. Detailed experimental results, including specific dataset performances, can be found in Appendix A.2.

4.3 Qualitative Results

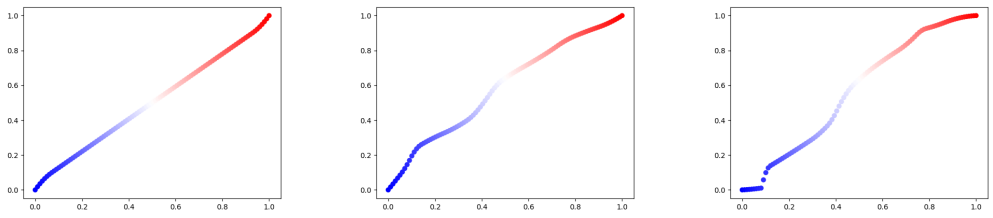
Figure 1 illustrates the evolution of the learned B-spline transformations and the corresponding training loss of data points throughout the training process. In Figures 1a we visualize the learned transformations, while Figures 1b display the training loss of individual data points. The images on the left side represent the early stages of training, where the B-spline closely resembles a min-max transform, as the network has not yet differentiated between data points based on their difficulty, resulting in uniformly high training loss across most data points.

As training progresses, shown in the right-side images, the network begins to distinguish between easy and difficult data points. For the more challenging data points with higher loss values, the B-spline allocates a larger input range, effectively adjusting its gradient to provide more capacity for these points. Meanwhile, the easier data points, which exhibit lower training loss, are compressed into a smaller input range. This adaptive learning behavior allows the B-spline transformation to prioritize and focus on the most challenging regions of the input space, leading to improved performance as the model dynamically allocates resources where they are needed most.

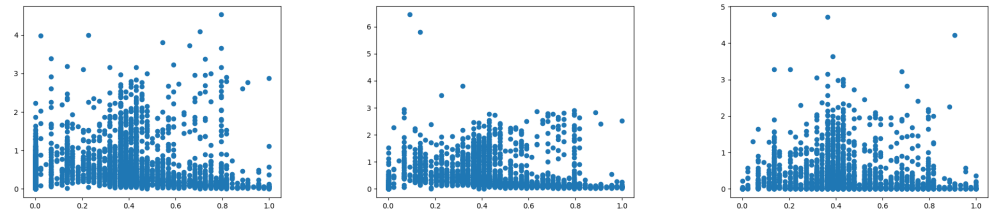
On the right side of Figures 1b, samples with high loss values are concentrated around raw input values of approximately 0.15, 0.4, and 0.8. Correspondingly, in Figures 1a, the curve exhibits a steeper gradient in these regions. This demonstrates that the proposed loss function and learnable normalization are functioning as designed, with the network allocating greater capacity to difficult data points by increasing the gradient in those areas. This behavior validates the effectiveness of our approach in dynamically adjusting the normalization based on the difficulty of the data.

5 Conclusion

In this paper, we proposed a learnable input normalization method tailored for tabular deep learning, using B-splines to model feature transformations. Our approach enables the network to learn optimal transformation parameters that improve training efficiency and performance. We also introduced a novel loss function that dynamically adjusts the input transformation based on the difficulty of learning specific data points, enhancing model adaptability.



(a) Evolution of learned B-spline



(b) Evolution of training loss

Figure 1: Qualitative results to visualize the evolution of learned transformations. Figures (a) show the learned transformation and Figures (b) show the training loss of data points. Images on the left side are captured at the early stage of training, and images on the right side are captured after the training finished.

Extensive experiments on OpenML datasets demonstrated the superiority of the proposed method compared to conventional input normalization techniques such as min-max scaling, standardization, and quantile transformation. The learnable input normalization approach consistently achieved better or comparable performance across various neural network architectures, demonstrating its robustness and versatility. Furthermore, our method requires minimal hyperparameter tuning, making it a practical solution for real-world tabular deep learning applications.

References

[1] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[2] S. Ö. Arik and T. Pfister. Tabnet: Attentive interpretable tabular learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 6679–6687, 2021.

[3] E. Beyazit, J. Kozaczuk, B. Li, V. Wallace, and B. Fadlallah. An inductive bias for tabular deep learning. *Advances in Neural Information Processing Systems*, 36, 2024.

[4] V. Borisov, T. Leemann, K. Seßler, J. Haug, M. Pawelczyk, and G. Kasneci. Deep neural networks and tabular data: A survey. *IEEE transactions on neural networks and learning systems*, 2022.

[5] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[6] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

[7] M. Feurer, J. N. van Rijn, A. Kadra, P. Gijsbers, N. Mallik, S. Ravi, A. Mueller, J. Vanschoren, and F. Hutter. Openml-python: an extensible python api for openml. *arXiv*, 1911.02490, 2020. URL `https://arxiv.org/pdf/1911.02490.pdf`.

[8] Y. Gorishniy, I. Rubachev, V. Khrulkov, and A. Babenko. Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems*, 34:18932–18943, 2021.

[9] Y. Gorishniy, I. Rubachev, and A. Babenko. On embeddings for numerical features in tabular deep learning. *Advances in Neural Information Processing Systems*, 35:24991–25004, 2022.

[10] L. Grinsztajn, E. Oyallon, and G. Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? *Advances in neural information processing systems*, 35: 507–520, 2022.

[11] S. Han, X. Hu, H. Huang, M. Jiang, and Y. Zhao. Adbench: Anomaly detection benchmark. *Advances in Neural Information Processing Systems*, 35:32142–32159, 2022.

[12] A. E. Johnson, T. J. Pollard, L. Shen, L.-w. H. Lehman, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. Anthony Celi, and R. G. Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3(1):1–9, 2016.

[13] K. Majmundar, S. Goyal, P. Netrapalli, and P. Jain. Met: Masked encoding for tabular data. *arXiv preprint arXiv:2206.08564*, 2022.

[14] A. Margeloiu, A. Bazaga, N. Simidjievski, P. Liò, and M. Jamnik. Tabmda: Tabular manifold data augmentation for any classifier using transformers with in-context subsetting. *arXiv preprint arXiv:2406.01805*, 2024.

[15] D. McElfresh, S. Khandagale, J. Valverde, V. Prasad C, G. Ramakrishnan, M. Goldblum, and C. White. When do neural nets outperform boosted trees on tabular data? *Advances in Neural Information Processing Systems*, 36, 2024.

[16] S. Onishi and S. Meguro. Rethinking data augmentation for tabular data in deep learning. *arXiv preprint arXiv:2305.10308*, 2023.

[17] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, and A. Courville. On the spectral bias of neural networks. In *International conference on machine learning*, pages 5301–5310. PMLR, 2019.

[18] J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo. Openml: networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013. doi: 10.1145/2641190.2641198. URL `http://doi.acm.org/10.1145/2641190.264119`.

[19] Z.-Q. J. Xu, Y. Zhang, T. Luo, Y. Xiao, and Z. Ma. Frequency principle: Fourier analysis sheds light on deep neural networks. *arXiv preprint arXiv:1901.06523*, 2019.

[20] J. Yan, J. Chen, Y. Wu, D. Z. Chen, and J. Wu. T2g-former: organizing tabular features into relation graphs promotes heterogeneous feature interaction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 10720–10728, 2023.

A Appendix

A.1 Implementation

Learnable B-spline in PyTorch

```
def bspline(
    p: int,
    X: Tensor,
    k: Tensor,
    t: Tensor,
    eps: float = 1e-6,
) -> Tensor:
    """
    Computes the coordinates of points on multiple B-spline curves using PyTorch.

    Parameters:
    p (int): The degree of the B-spline.
    X (Tensor): Control points, a 2D tensor of shape [L, N], where L is the number of B-spline curves,
                and N is the number of control points per curve.
    k (Tensor): Knot vectors, a 2D tensor of shape [L, N - p + 1]. The function clamps each curve by
                duplicating the first and last knot values p times.
    t (Tensor): Parameter values, a 2D tensor of shape [B, L], where L is the number of B-spline curves,
                and B is the number of parameter values to evaluate per curve.

    Returns:
    Tensor: Coordinates of the points on the B-spline curves at the given parameter values,
            a tensor of shape [B, L].
    """
    # Input validation
    assert p >= 1, f"B-spline degree p must be at least 1, but got p={p}"
    assert X.ndim == 2, f"X must be a 2D tensor [L, N], but got shape {X.shape}"
    assert k.ndim == 2, f"k must be a 2D tensor [L, N - p + 1], but got shape {k.shape}"
    assert k.shape[1] == X.shape[1] - p + 1
    assert (
        k == k.sort().values
    ).all(), "Knot vectors must be sorted in non-decreasing order"
    assert t.ndim == 2, f"t must be a 2D tensor [B, L], but got shape {t.shape}"
    assert (
        X.shape[0] == k.shape[0] == t.shape[1]
    ), f"Mismatch in num curves: X.shape[0]={X.shape[0]}, k.shape[0]={k.shape[0]}, t.shape[1]={t.shape[1]}"

    X = X.unsqueeze(2)  # Shape [L, N, 1]
    t = t.T.unsqueeze(2)  # Transpose and shape [L, B, 1] for easier batch processing

    # Backup original t values and clamp them to the range [0, 1 - eps]
    t_unclamp = torch.clone(t)
    t = torch.clamp(t, min=0, max=1 - eps)

    # Clamp the knot vectors by duplicating the first and last values `p` times
    k = torch.cat(
        [
            k[:, 0].unsqueeze(1).expand(-1, p),  # Clamp start
            k,
            k[:, -1].unsqueeze(1).expand(-1, p),  # Clamp end
```

```
        ],
        dim=1,
    )
    k = k.unsqueeze(1)  # Reshape to [L, 1, N + 2p]

    # Initialize basis function and derivative function values
    b = X.new_zeros((X.shape[0], t.shape[1], X.shape[1] + p))
    d = X.new_zeros((X.shape[0], t.shape[1], X.shape[1] + p))

    # Basis functions for degree 0
    for i in range(X.shape[1] + p):
        b[..., i] = (k[..., i] <= t[..., 0]) & (t[..., 0] < k[..., i + 1])

    # Iteratively compute basis functions and derivatives for degrees 1 to p
    for i in range(1, p + 1):
        left_denom = k[..., i:-1] - k[..., : -i - 1]  # Denominator for the left term
        right_denom = k[..., 1 + i :] - k[..., 1:-i]  # Denominator for the right term

        # Avoid NaNs by setting zero denominators to 1
        left_denom[left_denom == 0] = 1
        right_denom[right_denom == 0] = 1

        # Recurrence relation for derivatives
        d_term1 = (t - k[..., : -i - 1]) / left_denom * d[..., :-1]
        d_term2 = (k[..., i + 1 :] - t) / right_denom * d[..., 1:]
        d_term3 = 1 / left_denom * b[..., :-1]
        d_term4 = 1 / right_denom * b[..., 1:]

        d = d_term1 + d_term2 + d_term3 - d_term4

        # Recurrence relation for B-spline basis functions
        b_term1 = (t - k[..., : -i - 1]) / left_denom * b[..., :-1]
        b_term2 = (k[..., i + 1 :] - t) / right_denom * b[..., 1:]

        b = b_term1 + b_term2

    # Compute points on the B-spline curve
    point = b @ X
    deriv = d @ X

    # Handle out-of-bound `t` values by extrapolating using linear approximation
    k = k.unsqueeze(2)  # Shape [L, 1, N + 2p, 1]
    X = X.unsqueeze(2)  # Shape [L, N, 1, 1]

    left_out_mask = t_unclamp < k[..., 0]
    right_out_mask = t_unclamp >= k[..., -1]

    # Linear extrapolation for out-of-bound values
    left_extrap = X[:, 0] + deriv * (t_unclamp - t)
    right_extrap = X[:, -1] + deriv * (t_unclamp - t)

    # Replace out-of-bound points with extrapolated values
    point = torch.where(left_out_mask, left_extrap, point)
    point = torch.where(right_out_mask, right_extrap, point)

    point = point.squeeze(2).T  # Reshape to [B, L]
    deriv = deriv.squeeze(2).T  # Reshape to [B, L]

    return point, deriv

def cumsum_softmax(v, dim=1):
    v = F.softmax(v, dim)
    v = v.cumsum(dim)
    shape = list(v.shape)
    shape[dim] = 1
    z = v.new_zeros(shape)
    v = torch.cat([z, v], dim)
    return v

node_vector = cumsum_softmax(node_logits)
knot_vector = cumsum_softmax(knot_logits)
Yp, Yd = bspline(spline_order, node_vector, knot_vector, X)
```

A.2 Detailed Experimental Results

This section presents the normalized performance of various input normalization methods. Table 2-5 correspond to the MLP, ResNet, FT-Transformer, and T2G-Former architectures, respectively. The "Task Type" column indicates the dataset type, where M represents multi-class classification, B

represents binary classification, and R represents regression. The "Dataset ID" column provides the key for accessing each dataset through the OpenML-Python library [7]. The dataset labeled with ID CA is an exception and can be accessed using the scikit-learn library [5].

Table 2: Normalized performance of MLP with different input normalization methods

Dataset ID	Task type	Quantile	Standardization	Min-max	Proposed
14	M	0.000	0.361	0.312	0.244
18	M	0.646	1.000	0.874	0.924
22	M	0.986	0.606	0.769	0.697
23	M	0.621	0.184	0.000	0.833
29	B	0.477	0.942	0.767	1.000
31	B	0.167	0.000	0.191	0.048
36	M	0.634	0.000	0.211	0.282
42	M	0.000	0.640	0.800	0.640
46	M	0.140	0.190	0.095	0.000
151	B	0.743	0.216	0.000	0.367
182	M	0.315	0.777	0.750	0.505
188	M	0.772	0.086	0.670	0.721
189	R	0.166	0.000	0.115	0.102
194	R	0.690	0.046	1.000	0.518
210	R	0.461	0.986	1.000	0.780
337	B	1.000	0.844	1.000	0.635
344	R	0.000	0.566	0.343	0.170
372	M	0.000	0.146	0.045	0.639
422	R	0.989	0.874	0.827	0.923
444	B	0.000	0.000	0.125	0.375
44122	B	0.000	0.869	0.905	0.865
44124	B	0.671	0.215	0.638	0.776
44133	R	0.000	0.981	0.994	0.940
44134	R	0.133	0.700	0.000	0.113
45012	R	0.421	0.538	0.940	0.000
CA	R	0.325	0.000	0.575	0.533
Average		0.398	0.453	**0.536**	<u>0.524</u>

Table 3: Normalized performance of ResNet with different input normalization methods

Dataset ID	Task type	Quantile	Standardization	Min-max	Proposed
14	M	0.199	0.429	0.365	0.314
18	M	0.470	0.000	0.227	0.389
22	M	0.552	0.787	0.643	0.574
23	M	0.407	0.308	0.218	0.387
29	B	0.174	0.430	0.000	0.000
31	B	0.464	0.900	0.407	0.488
36	M	0.887	0.155	0.000	0.169
42	M	0.592	0.800	0.912	0.784
46	M	0.971	0.972	0.987	0.961
151	B	0.930	0.147	0.209	0.954
182	M	0.935	0.658	0.685	1.000
188	M	0.345	0.487	0.000	0.325
189	R	0.802	0.776	0.791	0.818
194	R	0.761	0.082	0.406	0.560
210	R	0.323	0.143	0.401	0.427
337	B	0.823	0.865	0.906	0.781
344	R	0.530	0.873	0.786	0.688
372	M	0.587	0.593	0.549	0.642
422	R	0.849	0.000	0.844	0.895
444	B	0.850	0.838	0.925	0.787
44122	B	0.209	0.944	0.960	0.827
44124	B	0.847	0.000	0.026	0.691
44133	R	0.351	0.926	0.925	0.892
44134	R	0.412	0.305	0.739	0.589
45012	R	0.552	0.594	0.932	0.840
CA	R	0.354	0.291	0.763	0.755
Average		<u>0.584</u>	0.512	0.562	**0.636**

Table 4: Normalized performance of FT-Transformer with different input normalization methods

Dataset ID	Task type	Quantile	Standardization	Min-max	Proposed
14	M	0.845	0.542	0.880	0.812
18	M	0.111	0.293	0.283	0.419
22	M	0.455	1.000	0.841	0.892
23	M	0.816	0.551	0.723	0.723
29	B	0.372	0.256	0.221	0.453
31	B	0.833	1.000	0.842	0.761
36	M	0.915	0.549	0.704	1.000
42	M	0.832	0.448	0.480	0.688
46	M	0.982	0.990	0.947	0.988
151	B	1.000	0.541	0.398	0.604
182	M	0.750	0.500	0.451	0.457
188	M	0.944	0.812	1.000	0.761
189	R	0.857	0.646	0.856	0.805
194	R	0.612	0.814	0.427	0.384
210	R	0.012	0.587	0.736	0.595
337	B	0.177	0.844	0.000	0.312
344	R	0.691	0.886	0.869	0.819
372	M	0.949	0.951	0.944	0.940
422	R	1.000	0.892	0.898	0.897
444	B	0.938	0.738	0.663	0.925
44122	B	0.336	0.958	0.963	0.969
44124	B	1.000	0.396	0.664	0.820
44133	R	0.636	1.000	0.436	0.937
44134	R	0.698	1.000	0.650	0.734
45012	R	0.862	0.716	1.000	0.798
CA	R	0.946	0.697	0.882	0.868
Average		0.714	<u>0.716</u>	0.683	**0.745**

Table 5: Normalized performance of T2G-Former with different input normalization methods

Dataset ID	Task type	Quantile	Standardization	Min-max	Proposed
14	M	0.828	0.886	1.000	0.980
18	M	0.611	0.298	0.457	0.495
22	M	0.000	0.397	0.002	0.146
23	M	1.000	0.980	0.932	0.911
29	B	0.430	0.430	0.012	0.390
31	B	0.861	0.804	0.550	0.873
36	M	0.958	0.282	0.423	0.669
42	M	0.752	0.864	1.000	0.640
46	M	0.996	0.973	1.000	0.985
151	B	0.977	0.620	0.383	0.389
182	M	0.380	0.000	0.356	0.370
188	M	0.650	0.716	0.670	0.480
189	R	0.969	0.839	0.961	1.000
194	R	0.088	0.617	0.646	0.000
210	R	0.000	0.537	0.527	0.521
337	B	0.542	0.948	0.818	0.245
344	R	0.726	0.949	0.930	1.000
372	M	0.975	0.987	1.000	0.971
422	R	1.000	0.968	0.970	0.949
444	B	0.962	0.875	0.875	1.000
44122	B	0.820	0.939	1.000	0.969
44124	B	0.895	0.286	0.402	0.454
44133	R	0.615	0.997	0.992	0.987
44134	R	0.629	0.963	0.637	0.546
45012	R	0.776	0.698	0.958	0.943
CA	R	0.846	0.653	0.984	1.000
Average		0.703	**0.712**	<u>0.711</u>	0.689