
RPN: Reconciled Polynomial Network

Towards Unifying PGMs, Kernel SVMs, MLP and KAN

Jiawei Zhang

IFM Lab*

Department of Computer Science

University of California, Davis

jiawei@ifmlab.org

Project Website: <https://www.tinybig.org>

Github: <https://github.com/jwzhanggy/tinyBIG>

(Initial Version: July 9, 2024)



“With four parameters I can fit an elephant, and with five I can make him wiggle his trunk.”

— Von Neumann

Abstract

In this paper, we will introduce a novel deep model named Reconciled Polynomial Network (RPN) for deep function learning. RPN has a very general architecture and can be used to build models with various complexities, capacities, and levels of completeness, which all contribute to the correctness of these models. As indicated in the subtitle, RPN can also serve as the backbone to unify different base models into one canonical representation. This includes non-deep models, like probabilistic graphical models (PGMs) - such as Bayesian network and Markov network - and kernel support vector machines (kernel SVMs), as well as deep models like the classic multi-layer perceptron (MLP) and the recent Kolmogorov-Arnold network (KAN).

Technically, inspired by the Taylor’s Theorem, RPN proposes to disentangle the underlying function to be inferred into the inner product of a data expansion function and a parameter reconciliation function. Together with the remainder function, RPN accurately approximates the underlying functions that governs data distributions. The data expansion functions in RPN project data vectors from the input space to a high-dimensional intermediate space, specified by the expansion functions in definition. Meanwhile, RPN also introduces the parameter reconciliation functions to fabricate a small number of parameters into a higher-order

*© 2024 IFM Lab. All rights reserved. The RPN project and **TINYBIG** toolkit is developed and maintained by IFM Lab.

parameter matrix to address the “curse of dimensionality” problem caused by the data expansions. In the intermediate space, the expanded vectors are polynomially integrated and further projected into the low-dimensional output space via the inner product with the reconciled parameters generated by these parameter reconciliation functions. Moreover, the remainder functions provide RPN with additional complementary information to reduce potential approximation errors.

We conducted extensive empirical experiments on numerous benchmark datasets across multiple modalities, including continuous function datasets, discrete vision and language datasets, and classic tabular datasets, to investigate the effectiveness of RPN. The experimental results demonstrate that, RPN outperforms MLP and KAN with mean squared errors at least $\times 10^{-1}$ lower (and even $\times 10^{-2}$ lower in some cases) for continuous function fitting. On both vision and language benchmark datasets, using much less learnable parameters, RPN consistently achieves higher accuracy scores than Naive Bayes, kernel SVMs, MLP, and KAN for discrete image and text data classifications. In addition, equipped with the probabilistic data expansion functions, RPN learns better probabilistic dependency relationships among variables and outperforms other probabilistic models, including Naive Bayes, Bayesian networks, and Markov networks, for learning on classic tabular benchmark datasets.

Reconciled Polynomial Network (RPN) proposed in this paper provides the opportunity to represent and interpret current machine and deep learning models as sequences of vector space expansions and parameter reconciliations. These functions can all deliver concrete physical meanings about both the input data and model parameters. Furthermore, the application of simple inner-product and summation operations to these functions significantly enhances the interpretability of RPN. This paper presents not only empirical experimental investigations but also in-depth discussions on RPN, addressing its interpretations, merits, limitations, and potential future developments.

What’s more, to facilitate the implementation of RPN-based models, we have developed and released a toolkit named **TINYBIG**. This toolkit encompasses all functions, modules, and models introduced in this paper, accompanied by comprehensive documentation and tutorials. Detailed information about **TINYBIG** is available at the project’s GitHub repository and the dedicated project webpage, with their respective URLs provided above.

KEY WORDS: Function Learning; Data Expansion; Parameter Reconciliation; Remainder Function; Deep Learning

Contents

1	Introduction	5
2	Deep Function Learning	7
2.1	What is Deep Function Learning?	7
2.2	Deep Function Learning vs Deep Representation Learning	8
2.3	RPN vs Other Base Models	9
3	Notations and Background Knowledge on Taylor’s Theorem	11
3.1	Notation System	11
3.2	Taylor’s Theorem for Univariate Function Approximation	11
3.3	Taylor’s Theorem for Multivariate Function Approximation	12
3.4	Taylor’s Theorem based Machine Learning Models	13
4	RPN: Reconciled Polynomial Network for Deep Function Learning	14
4.1	RPN: Reconciled Polynomial Network	14
4.2	RPN Component Functions	14
4.3	Wide RPN: Multi-Head and Multi-Channel Model Architecture	15
4.4	Deep RPN: Multi-Layer Model Architecture	16
4.5	Versatile RPN: Nested and Extended Expansion Functions	16
4.6	Learning Correctness of RPN: Complexity, Capacity and Completeness	17
4.7	Learning Cost of RPN: Space, Time and Parameter Number	18
5	List of Expansion, Reconciliation and Remainder Functions for RPN Model	18
5.1	Data Expansion Functions	18
5.2	Parameter Reconciliation Functions	27
5.3	Remainder Functions	30
6	Unifying Existing Base Models with RPN Canonical Representation	32
6.1	Unifying MLP with RPN	32
6.2	Unifying KAN with RPN	33
6.3	Unifying Kernel SVM with RPN	34
6.4	Unifying PMs with RPN	35
7	Empirical Evaluations of RPN	38
7.1	Continuous Function Approximation	38
7.2	Discrete Image and Text Classification	43

7.3	Probabilistic Dependency Inference	51
8	Interpretations of the RPN Model Design	53
8.1	Theoretic Machine Learning Interpretations	53
8.2	Biological Neuroscience Interpretations	56
9	Intellectual Merits, Limitations and Future Work of RPN	58
9.1	Intellectual Merits of RPN	58
9.2	Limitations and Future Work of RPN	59
10	Related Work	60
10.1	Machine Learning and Deep Learning Base Models	60
10.2	Component Function Design	61
11	Conclusion	63
A	Appendix	71
A.1	Performance of RPN Variants for Continuous Function Fitting and Approximation	71
A.2	Ablation Studies of RPN on MNIST Dataset	74
A.3	Visualization of RPN Data Expansion and Parameter Reconciliation on MNIST Dataset	96
A.4	Licensing Rights of Using BioRender Created Contents in This Paper	105

1 Introduction

Over the past 70 years, the field of artificial intelligence has experienced dramatic changes in both the problems studied and the models used. With the emergence of new learning tasks, various machine learning models, each designed based on different prior assumptions, have been proposed to address these problems. As shown in Figure 1, we illustrate the timeline about three types of machine learning models that have dominated the field of artificial intelligence in the past 50 years, including probabilistic graphical models [37, 57, 39], support vector machines [12, 76, 8] and deep neural networks [66, 23]. Along with important technological breakthroughs, these models each had their moments of prominence and have been extensively explored and utilized in various research and application tasks related to data and learning nowadays. Besides these three categories of machine learning models, there are many other models (*e.g.*, the tree based models and clustering models) that do not fit into these categories, but we will not discuss them in this paper and will leave them for future investigation instead.

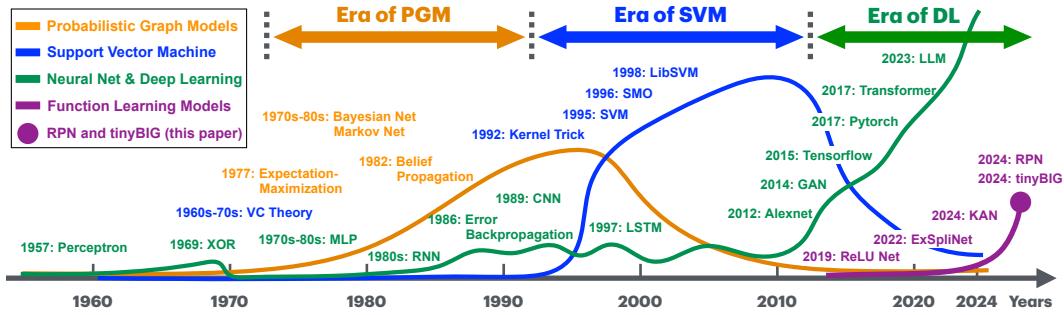


Figure 1: The timeline illustrates the development of various dominant machine learning base models over the past 70 years, with different colors representing different models. **Orange Color:** probabilistic graphical models (1980s to mid-2000s); **Blue Color:** support vector machine (mid 1990s to early 2010s); **Green Color:** deep learning models (mid-2010s to present); and **Purple Color:** deep function learning (2020s to present).

In this paper, we will introduce a novel deep model, namely **Reconciled Polynomial Network (RPN)**, that can potentially unify these different aforementioned base models into one shared representation. In terms of model architecture, RPN consists of three component functions: **data expansion function**, **parameter reconciliation function** and **remainder function**. Inspired by the Taylor’s theorem, RPN disentangles the input data from model parameters, and approximates the target functions to be inferred as the inner product of the data expansion function with the parameter reconciliation function, subsequently summed with the remainder function.

Based on architecture of RPN, inferring the diverse underlying mapping that governs data distributions (from inputs to outputs) is actually equivalent to inferring these three compositional functions. This inference process of the diverse data distribution mappings based on RPN is named as the **function learning task** in this paper. Specifically, the “function” term mentioned in the task name refers to not only the **mathematical function** components composing the RPN model but also the **cognitive function** of RPN as an intelligent system to relate input signals with desired output response. Function learning has been long-time treated as equivalent to the continuous function fitting and approximation for regression tasks only. Actually, in psychology and cognitive science, researchers have also used the function learning concept for modeling the mental induction process of stimulus-response relations of human and other intelligent subjects [9, 38], involving the acquisition of knowledge, manipulation of information and reasoning. In this paper, we argue that function learning is the most fundamental task in intelligent model learning, encompassing **continuous function approximation**, **discrete vision and language data recognition and prediction**, and **cognitive and logic dependency relation induction**. The following Section 2 will provide an

in-depth discussion of function learning and offer a comparative analysis of function learning with the currently prevailing paradigm of representation learning.

Determined by the definitions of the data expansion functions, RPN will project data vectors from the input space to an intermediate (higher-dimensional) space represented with new basis vectors. To address the “curse of dimensionality” issue stemming from the data expansions, the parameter reconciliation function in RPN fabricates a reduced set of parameters into a higher-order parameter matrix. These expanded data vectors are then polynomially integrated via the inner product with these generated reconciled parameters, which further projects these expanded data vectors back to the desired lower-dimensional output space. Moreover, the remainder function provides RPN with additional complementary information to further reduce potential approximation errors. All these component functions within RPN embody concrete physical meanings. These functions, coupled with the straightforward application of simple inner product and summation operators, provide RPN with greater interpretability compared to other existing base models.

RPN possesses a highly versatile architecture capable of constructing models with diverse complexities, capacities, and levels of completeness. In this paper, to provide RPN with greater modeling capabilities in design, we enable RPN to incorporate both a wide architecture featuring multi-heads and multi-channels (within each layer), as well as a deep architecture comprising multi-layers. Additionally, we further offer RPN with a more adaptable and lightweight mechanism for constructing models with comparable capabilities through the nested and extended data expansion functions. These powerful yet flexible design mechanisms provide RPN with greater modeling capability, enabling it to serve as the backbone for unifying various base models mentioned above into a single representation. This includes non-deep models, like probabilistic graphical models (PGMs) - such as Bayesian network [57] and Markov network [37] - and kernel support vector machines (kernel SVMs) [8], as well as deep models like the classic multi-layer perceptron (MLP) [66] and the recent Kolmogorov-Arnold network (KAN) [51].

To investigate the effectiveness of RPN for deep function learning tasks, this paper will present extensive empirical experiments conducted on numerous benchmark datasets. Given RPN’s status as a general base model for function learning, we evaluate its performance with datasets in various modalities, including numerical function datasets (for continuous function fitting and approximation), image and text datasets (for discrete vision and language data classification), and classic tabular datasets (for variable dependency relationship inference and induction). The experimental results demonstrate that, RPN outperforms MLP and KAN with mean squared errors at least $\times 10^{-1}$ lower (and even $\times 10^{-2}$ lower in some cases) on continuous function fitting tasks. On both vision and language benchmark datasets, using much less learnable parameters, RPN consistently achieves higher accuracy scores than Naive Bayes, kernel SVM, MLP, and KAN for these discrete data classifications. Moreover, equipped with probabilistic data expansion functions, RPN also learns better probabilistic dependency relationships among variables and outperforms probabilistic models, including Naive Bayes, Bayesian networks, and Markov networks, for learning on the tabular benchmark datasets.

We summarize the contributions of this paper as follows:

- **RPN for Deep Function Learning:** In this paper, we propose the task of “deep function learning” and introduce a novel deep function learning base model, *i.e.*, the Reconciled Polynomial Network (RPN). RPN has a versatile model architecture and attains superior modeling capabilities for diverse deep function learning tasks on various multi-modality datasets. Moreover, by disentangling input data from model parameters with the expansion, reconciliation and remainder functions, RPN achieves greater interpretability than existing deep and non-deep base models.
- **Component Functions:** In this paper, we introduce a tripartite set of compositional functions - data expansion, parameter reconciliation, and remainder functions - that serve as the building blocks for the RPN model. By strategically combining these component functions, we can construct a multi-head, multi-channel, and multi-layer architecture, enabling

RPN to address a wide spectrum of learning challenges across diverse function learning tasks.

- **Base Model Unification:** This paper demonstrates that RPN provides a unifying framework for several influential base models, including Bayesian networks, Markov networks, kernel SVMs, MLP, and KAN. We show that, through specific selections of component functions, each of these models can be unified into RPN’s canonical representation, characterized by the inner product of a data expansion function with a parameter reconciliation function, summed with a remainder function.
- **Experimental Investigations:** This paper presents a series of extensive empirical experiments conducted across numerous benchmark datasets for various deep function learning tasks, including numerical function fitting tasks, discrete image and language data classification tasks, and tabular data based dependency relation inference and induction tasks. The results demonstrate RPN’s consistently superior performance compared to other existing base models, providing strong empirical validations of our proposed model.
- **The TINYBIG Toolkit:** To facilitate the adoption, implementation and experimentation of RPN, we have released **TINYBIG**, a comprehensive toolkit for RPN model construction. **TINYBIG** offers a rich library of pre-implemented functions, including 25 categories of data expansion functions, 10 parameter reconciliation functions, and 5 remainder functions, along with the complete model framework and optimized model training pipelines. This integrated toolkit enables researchers to rapidly design, customize, and deploy RPN models across a wide spectrum of deep function learning tasks.

This paper provides a comprehensive investigation of the proposed Reconciled Polynomial Network model. The remaining parts of this paper will be organized as follows. In Section 2, we will first introduce the novel function learning concept and compare RPN with several existing base models. In Section 3, we will cover notations, task formulations, and essential background knowledge on Taylor’s theorem. In Section 4, we will provide detailed descriptions of RPN model’s architecture and design mechanisms. Our library of expansion, reconciliation, and remainder functions will be presented in Section 5. In Section 6, we demonstrate how RPN unifies and represents existing base models. The experimental evaluation of RPN’s performance on numerous benchmark datasets will be provided in Section 7. After that, we will discuss RPN’s interpretations from both machine learning and biological neuroscience perspectives in Section 8. In Section 9, we will critically discuss the merits, limitations and potential future works of RPN. Finally, we will introduce the related works in Section 10 and conclude this paper in Section 11.

2 Deep Function Learning

In this section, we will first introduce the concept of **deep function learning** task. After that, we will provide the detailed clarifications about how deep function learning differs from existing deep representation learning tasks. Based on this concept, we will further compare RPN, the deep function learning model proposed in this paper, with other existing non-deep and deep base models to illustrate their key differences.

2.1 What is Deep Function Learning?

As its name suggests, **deep function learning**, as the most fundamental task in machine learning, aims to build general deep models composed of a sequence of component functions that infer the relationships between inputs and outputs. These component functions define the mathematical projections across different data and parameter spaces. In deep function learning, without any prior assumptions about the data modalities, the corresponding input and output data can also appear in different forms, including but not limited to continuous numerical values (such as continuous functions), discrete categorical features (such as images and language data), probabilistic variables (defining the dependency relationships between inputs and outputs), and others.

DEFINITION 1 (Deep Function Learning): Formally, given the input and output spaces \mathbb{R}^m and \mathbb{R}^n , the underlying mapping that governs the data projection between these two spaces can be denoted as:

$$f : \mathbb{R}^m \rightarrow \mathbb{R}^n. \quad (1)$$

Deep function learning aims to build a model g as a composition of deep mathematical function sequences g_1, g_2, \dots, g_K to project data across different vector spaces, which can be represented as

$$g : \mathbb{R}^m \rightarrow \mathbb{R}^n, \text{ and } g = g_1 \circ g_2 \circ \dots \circ g_k, \quad (2)$$

where the \circ notation denotes the component function integration and composition operators. The component functions g_i can be defined on either input data or the model parameters.

For input $\mathbf{x} \in \mathbb{R}^m$, if the output generated by the model can approximate the desired output, i.e.,

$$g(\mathbf{x}|\mathbf{w}, \boldsymbol{\theta}) \approx f(\mathbf{x}), \quad (3)$$

then model is g can serve as an approximated mapping of f . Notations $\mathbf{w} \in \mathbb{R}^l$ and $\boldsymbol{\theta} \in \mathbb{R}^{l'}$ denote the learnable parameters and hyper-parameters of the function learning model, respectively.

Below, we will further clarify the distinctions between deep function learning and current deep model-based data representation learning tasks. After that, we will compare our RPN model, which is grounded in deep function learning, against other existing base models.

2.2 Deep Function Learning vs Deep Representation Learning

As mentioned previously, the function learning tasks and models examined in this paper encompass not only continuous function approximation, but also discrete data classification and the induction of dependency relations. Besides the literal differences indicated by their names - representation learning is data oriented but function learning is model oriented - deep function learning significantly differs from the current deep representation learning in several critical perspectives discussed below.

- **Generalizability:** Representation learning, to some extent, has contributed to the current fragmentation within the AI community, as data - the carrier of information - is collected, represented, and stored in disparate modalities. Existing deep models, specifically designed for certain modalities, tend to overfit to these modality-specific data representations in addition to learning the underlying information. Applying a model proposed for one modality to another typically necessitates significant architectural redesigns. Recently, there have been efforts to explore the cross-modal applicability of certain models, e.g., CNNs for language and Transformers for vision, but replicating such cross-modality migration explorations across all current and future deep models is extremely expensive and unsustainable. Furthermore, to achieve the future artificial general intelligence (AGI), the available data in a single modality is no longer sufficient for training larger models. Deep function learning, without any prior assumptions on data modalities, will pave the way for improving the model generalizability. These learned functions should demonstrate their generalizability and applicability to multi-modal data from the outset, during their design and investigation phases.
- **Interpretability:** Representation learning primarily aims to learn and extract latent patterns or salient features from data, aligning with the technological advancements in data science and big data analytics over the past two decades. However, the learned data representations often lack interpretable physical meanings, rendering most current AI models to be black boxes. In contrast, to realize the goal of explainable AI (xAI), greater emphasis must be placed on developing new model architectures with concrete physical meanings and mathematical interpretations in the future. The RPN based deep function learning, on the other hand, aims to learn compositional functions with inherent physical interpretability for building general-purpose models across various tasks, thereby bridging the interpretability gap of current and future deep models.

- **Reusability:** Representation learning converts input data into embedding vectors that can be stored in vector databases and reused in future applications (*e.g.*, in the retrieval-augmented generation (RAG) models). However, in practical real-world scenarios, the direct usability of such pre-computed embedding representations in vector databases can be quite limited, both in terms of use cases and transactional queries or operations. Moreover, as new data arrives, new architectures are designed, and new model checkpoints are updated in the dynamically evolving online and offline worlds, we may need to re-learn all these embedding representation vectors via fine-tuning or retraining from scratch to maintain consistency, greatly impacting the reusability of representation learning results. In contrast, function learning focuses on learning compositional functions for underlying mapping inference, whose disentangled design is inherently well-suited for reusability and continual learning in future AI systems.

As a special type of machine learning, both deep function learning and deep representation learning are focused on inferring the underlying distributions of data. In contrast to representation learning, deep function learning narrows down the model architecture to a sequence of concrete mathematical functions defined on both data and parameter spaces. The RPN based deep function learning model also disentangles data from parameters and aims to infer and learn these compositional functions, each bearing a concrete physical interpretation for mathematical projections between various data and parameter domains. In contrast, existing deep representation learning models inextricably mix data and parameters together, rendering model interpretability virtually impossible. Below, we will further illustrate the differences of RPN with several existing base models.

2.3 RPN vs Other Base Models

Figure 2 compares the RPN model proposed for deep function learning with several base models in terms of mathematical theorem foundations, formula representations, and model architectures. The top three Plots (a)-(c) describe the non-deep base models: Bayesian Networks, Markov Networks, and Kernel SVMs; while the Plots (d)-(i) at the bottom illustrate the architectures of deep base models: MLPs, KANs, and RPN. For MLPs and KANs, Plots (d)-(e) and (g)-(h) illustrate their two-layer and three-layer architectures, respectively. Similarly, for RPN, we present its one-layer and three-layer architectures in Plots (f) and (i).

Based on the plots shown in Figure 2, we can observe significant differences of RPN compared against these base models, which are summarized as follows:

- **RPN vs Non-Deep Base Models:** Examining the model plots, we observe that all these base model architectures can be represented as graph structures composed of variables and their relationships. The model architecture of the Markov network is undirected, while that of the Bayesian network is directed. Similarly, for MLP, KAN, and RPN, although we haven’t shown the variable connection directions, their model architecture are also directed, flowing from bottom to top. The model architectures of both Markov network and Bayesian network consist of variable nodes that correspond only to input features and output labels. In contrast, for kernel SVM, MLP, KAN, and RPN, their model architectures involve not only nodes representing inputs and outputs, but also those representing expansions and hidden layers. Both RPN and kernel SVM involve a data expansion function to project input data into a high-dimensional space. However, their approaches diverge thereafter. Kernel SVM directly defines parameters within this high-dimensional space to integrate expansion vectors into outputs. In contrast, RPN fabricates these high-dimensional parameters via a reconciliation function from a reduced set of parameters instead.
- **RPN vs Deep Base Models:** The difference between RPN and MLP is easy to observe. MLP involves neither data expansion nor parameter reconciliation. Instead, they apply activation functions to neurons after input integration, which is parameterized by neuron connection weights. Unlike MLPs with predefined, static activation functions, the recent

Non-Deep Model	Bayesian Network	Markov Network	Kernel SVM
Theorem	Bayes' Theorem and Probability Theory	Hammersley-Clifford Theorem and Probability Theory	Representer Theorem and Lagrange Duality Theory
Formula	$\log P(X_1, X_2, \dots, X_m) = \sum_{i=1}^m \log P(X_i, \Gamma(X_i)) - \log P(\Gamma(X_i))$	$\log P(X_1, X_2, \dots, X_m) = \sum_{i=1}^k \log \phi(X_{C_i}) - \log Z$	$g(\mathbf{x} \mathbf{w}, b) = \text{sign}(\mathbf{w}^\top \phi(\mathbf{x}) + b)$
Model Architecture	(a)	(b)	(c)
Deep Model	Multi-Layer Perceptron (MLP)	Kolmogorov-Arnold Network (KAN)	Reconciled Polynomial Network (RPN)
Theorem	Universal Approximation Theorem	Kolmogorov-Arnold Representation Theorem	Taylor's Theorem
Formula (Shallow)	$f(\mathbf{x}) \approx \sum_{i=1}^k a_i \sigma(\mathbf{w}_i \mathbf{x} + b_i)$	$f(\mathbf{x}) = \sum_{q=0}^{2m} \Phi_q(\sum_{p=1}^m \phi_{q,p}(x_p))$	$f(\mathbf{x}) = \langle \kappa(\mathbf{x}), \psi(\mathbf{w}) \rangle + \pi(\mathbf{x})$
Model Architecture (1-layer)	(d)	(e)	(f)
Formula (Deep)	$f(\mathbf{x}) \approx (\mathbf{W}_3 \circ \sigma_2 \circ \mathbf{W}_2 \circ \sigma_1 \circ \mathbf{W}_1)(\mathbf{x})$	$f(\mathbf{x}) = (\Phi_3 \circ \Phi_2 \circ \Phi_1)(\mathbf{x})$	$f(\mathbf{x}) = ((\mathbf{K}\Psi\Pi)_3 \circ (\mathbf{K}\Psi\Pi)_2 \circ (\mathbf{K}\Psi\Pi)_1)(\mathbf{x})$
Model Architecture (3-layer)	(g)	(h)	(i)

Figure 2: A comparison of RPN with Bayesian Network, Markov Network, Kernel SVM, MLP and KAN in terms of mathematical theorem foundation, formula and model architecture. In the plots, we represent the learnable parameters and functions in the red color, while the unlearnable/fixed ones are represented in the dark/gray colors instead. The inputs and outputs are represented with the solid circles, while the expansions are represented as the hollow circles instead.

KAN model proposes to learn the activation functions attached to neuron-neuron connections, with their outputs being directly summed together. RPN, in contrast, integrates the strengths of both kernel SVM and KAN: it employs the expansion function from kernel SVM and adopts learnable functions attached to neuron-to-neuron connections, similar to KANs. Meanwhile, different from kernel SVM, MLP and KAN, RPN introduces a novel approach to fabricate a large number of parameters from a small set. This technique helps address both the “curse of dimension” and the model generalization problems. We have briefly mentioned the “curse of dimension” problem before already, and will discuss about the model generalization issue later in Section 8 from the VC-theory perspective.

Here, we briefly compare these base models with RPN. In the following Section 6, after we introduce the RPN model architecture and the component functions, we will further discuss how to unify these base models into RPN’s canonical representations. More comprehensive information about these base models and other related work will also be provided in Section 10.

3 Notations and Background Knowledge on Taylor's Theorem

This section first introduces the notation system used throughout this paper. Based on the notations, we then briefly present Taylor's theorem as the preliminary knowledge of the RPN model, which will be introduced in the following Section 4.

3.1 Notation System

In the sequel of this paper, we will use the lower case letters (e.g., x) to represent scalars, lower case bold letters (e.g., \mathbf{x}) to denote column vectors, bold-face upper case letters (e.g., \mathbf{X}) to denote matrices and high-order tensors, and upper case calligraphic letters (e.g., \mathcal{X}) to denote sets. Given a matrix \mathbf{X} , we denote $\mathbf{X}(i, \cdot)$ and $\mathbf{X}(:, j)$ as its i_{th} row and j_{th} column, respectively. The (i_{th}, j_{th}) entry of matrix \mathbf{X} can be denoted as $\mathbf{X}(i, j)$. We use \mathbf{X}^\top and \mathbf{x}^\top to represent the transpose of matrix \mathbf{X} and vector \mathbf{x} . For vector \mathbf{x} , we represent its L_p -norm as $\|\mathbf{x}\|_p = (\sum_i |\mathbf{x}(i)|^p)^{\frac{1}{p}}$. The Frobenius norm of matrix \mathbf{X} is represented as $\|\mathbf{X}\|_F = (\sum_{i,j} |\mathbf{X}(i, j)|^2)^{\frac{1}{2}}$. The elementwise product of vectors \mathbf{x} and \mathbf{y} of the same dimension is represented as $\mathbf{x} \odot \mathbf{y}$, their inner product is represented as $\langle \mathbf{x}, \mathbf{y} \rangle$, and their Kronecker product is $\mathbf{x} \otimes \mathbf{y}$. The elementwise product and Kronecker product operators can also be applied to matrices \mathbf{X} and \mathbf{Y} as $\mathbf{X} \odot \mathbf{Y}$ and $\mathbf{X} \otimes \mathbf{Y}$, respectively.

3.2 Taylor's Theorem for Univariate Function Approximation

Taylor's theorem approximates a d -times differentiable function around a given point using polynomials up to degree d , commonly referred to as the d_{th} -order Taylor polynomial. In this section, we first introduce Taylor's theorem for univariate functions, which also generalizes to multivariate and vector valued functions. We will briefly describe its extension to multivariate functions in the subsequent Subsection 3.3, and then introduce the RPN model designed based on Taylor's theorem with vector valued functions in Section 4.

THEOREM 1 (*Taylor's Theorem*): *Let $d \geq 1$ be an integer and let function $f : \mathbb{R} \rightarrow \mathbb{R}$ be d times differentiable at the point $a \in \mathbb{R}$. As illustrated in Figure 3, then there exists a function $h_d : \mathbb{R} \rightarrow \mathbb{R}$ such that*

$$\begin{aligned} f(x) &= \frac{f(a)}{0!}(x-a)^0 + \frac{f'(a)}{1!}(x-a)^1 + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(d)}(a)}{d!}(x-a)^d + R_d(x), \\ &= \sum_{i=0}^d \frac{f^{(i)}(a)}{i!}(x-a)^i + R_d(x). \end{aligned} \tag{4}$$

In the equation, $R_d(x)$ is also normally called the “remainder” term and can be represented as

$$R_d(x) = h_d(x)(x-a)^d, \text{ where } \lim_{x \rightarrow a} h_d(x) = 0. \tag{5}$$

According to the above description of the Taylor's Theorem, the function output $f(x)$ can be represented as a summation of polynomials of high degrees of the $(x - a)$. What's more, in this paper, we propose to further disentangle the variable x from the given constant point a . Terms like $(x - a)^k$ can be decomposed into summations of polynomials in x alone, with a serving as the coefficients:

$$(x - a)^d = \binom{d}{0}(-a)^{d-0}x^0 + \binom{d}{1}(-a)^{d-1}x^1 + \cdots + \binom{d}{d}(-a)^{d-d}x^d. \tag{6}$$

Based on the decomposition, we can rewrite the above Equation (4) as follows:

$$f(x|a) = \langle \mathbf{x}, \mathbf{c} \rangle + R_d(x), \tag{7}$$

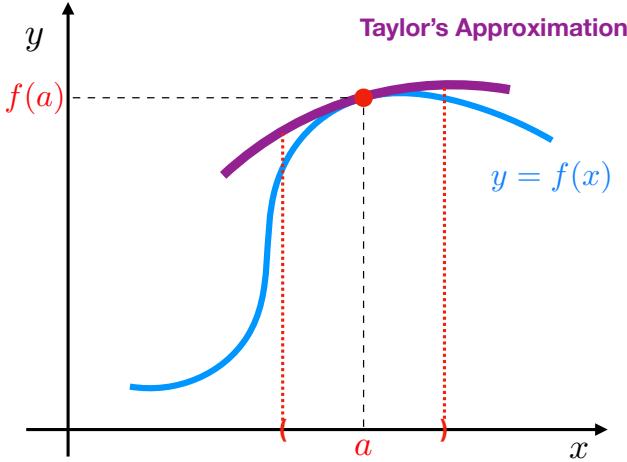


Figure 3: An illustration of Taylor's approximation of continuous functions.

where $\langle \cdot, \cdot \rangle$ denotes the inner product operator. The expanded data vector $\mathbf{x} = \kappa(x) = [x^0, x^1, x^2, \dots, x^d] \in \mathbb{R}^{d+1}$ contains the high-order polynomials of x , where the created coefficient vector $\mathbf{c} = \psi(a) = [c_0, c_1, c_2, \dots, c_d] \in \mathbb{R}^{d+1}$ has the same dimension as \mathbf{x} . Each coefficient term, such as c_i (where $i \in \{0, 1, \dots, d\}$), is fabricated with a as follows:

$$c_i = \sum_{j=i}^d \frac{f^{(j)}(a)}{j!} \binom{j}{i} (-a)^{j-i}. \quad (8)$$

The remainder term measure the error in approximating f with Taylor's polynomials. The representation illustrated in Equation (5) above is known as the “Peano Remainder”. In addition, mathematicians have introduced many different forms of remainder representations, some of which are listed below:

(a) Peano Remainder:

$$R_d(x) = h_d(x)(x - a)^d, \quad (9)$$

where $\lim_{x \rightarrow a} h_d(x) = 0$.

(c) Cauchy Remainder:

$$R_d(x) = \frac{f^{(d+1)}(\xi)}{d!} (x - \xi)^d (x - a),$$

for some ξ between a and x .

$$(11) \qquad \qquad \qquad (12)$$

(b) Lagrange Remainder:

$$R_d(x) = \frac{f^{(d+1)}(\xi)}{(d+1)!} (x - a)^{d+1}, \quad (10)$$

for some ξ between a and x .

(d) Schlömilch Remainder:

$$R_d(x) = \frac{f^{(d+1)}(\xi)}{d!} (x - \xi)^{d+1-p} \frac{(x - a)^p}{p},$$

for some $p > 0$ and ξ between a and x .

3.3 Taylor's Theorem for Multivariate Function Approximation

Representing multivariate continuous functions with Taylor's polynomials is more intricate. In this part, we use a multivariate function $f : \mathbb{R}^m \rightarrow \mathbb{R}$ as an example to illustrate how to disentangle the input variables and function parameters via Taylor's formula. Similar as the above single-variable function, assuming function f is d_{th} -time continuously differentiable at point $\mathbf{a} \in \mathbb{R}^m$, then for the inputs near the point can be approximated as

$$f(\mathbf{x}) = \sum_{|\alpha| \leq d} \frac{D^{|\alpha|} f(\mathbf{a})}{\alpha!} (\mathbf{x} - \mathbf{a})^\alpha + R_d(\mathbf{x}). \quad (13)$$

The notation $D^{|\alpha|}f$ denotes the $|\alpha|_{th}$ partial derivatives of function f and $R_d(\mathbf{x})$ denotes the remainder term:

$$\begin{cases} D^{|\alpha|}f = \frac{\partial^{|\alpha|}f}{\partial x_1^{\alpha_1} \cdots \partial x_m^{\alpha_m}} \\ R_d(\mathbf{x}) = \sum_{|\alpha|=d} h_\alpha(\mathbf{x})(\mathbf{x} - \mathbf{a})^\alpha \end{cases}, \text{ where } \begin{cases} |\alpha| = \alpha_1 + \alpha_2 + \cdots + \alpha_m; \\ \alpha! = \alpha_1! \alpha_2! \cdots \alpha_m!; \\ (\mathbf{x} - \mathbf{a})^\alpha = (x_1 - a_1)^{\alpha_1} \cdots (x_m - a_m)^{\alpha_m}; \\ \lim_{\mathbf{x} \rightarrow \mathbf{a}} h_\alpha(\mathbf{x}) = 0. \end{cases} \quad (14)$$

Similar to the single-variable case, the variables $\mathbf{x} = [x_1, x_2, \dots, x_m]$ involved in the multivariate polynomials can also be decoupled from the data point $\mathbf{a} = [a_1, a_2, \dots, a_m]$, leading to the following representation:

$$f(\mathbf{x}|\mathbf{a}) = \langle \bar{\mathbf{x}}, \mathbf{c} \rangle + R_d(\mathbf{x}), \quad (15)$$

where $\bar{\mathbf{x}}$ denotes the data expansion vector, and \mathbf{c} represents the created coefficient vector. Their detailed representations are provided as follows.

- *Data Expansion*: Function $\kappa : \mathbb{R}^m \rightarrow \mathbb{R}^D$ will expand the input vector $\mathbf{x} = [x_1, x_2, \dots, x_m]^\top$ to $\bar{\mathbf{x}} \in \mathbb{R}^D$ as follows:

$$\bar{\mathbf{x}} = \kappa(\mathbf{x}) = [\underbrace{1}_{1 \text{ term of order 0}}, \underbrace{x_1, x_2, \dots, x_m}_{m \text{ terms of order 1}}, \underbrace{x_1^2, x_1 x_2, \dots, x_m^2}_{m^2 \text{ terms of order 2}}, \dots, \underbrace{\dots, x_m^d}_{m^d \text{ terms of order } d}]^\top, \quad (16)$$

where the expansion output vector has a dimension of $D = \sum_{i=0}^d m^i$.

- *Parameter Fabrication*: Function $\psi : \mathbb{R}^m \rightarrow \mathbb{R}^D$ will fabricate the constant \mathbf{a} to a coefficient vector of dimension D as follows:

$$\mathbf{c} = \psi(\mathbf{a}) = [\underbrace{c_0}_{\text{coeff. of constant}}, \underbrace{c_1, c_2, \dots, c_m}_{\text{coeff. of terms with order 1}}, \underbrace{c_{1,1}, c_{1,2}, \dots, c_{m,m}, \dots}_{\text{coeff. of terms with order 2}}, \dots, \underbrace{\dots, c_{m,m}, \dots, c_m}_{\text{coeff. of terms with order } d}]^\top. \quad (17)$$

The coefficient vector \mathbf{c} has the same dimension as $\bar{\mathbf{x}}$, and the coefficient c_{i_1, i_2, \dots, i_k} corresponds to the polynomial term $x_{i_1} x_{i_2} \cdots x_{i_k}$. As to the specific representation of c_{i_1, i_2, \dots, i_k} , it can be obtained by decomposing the above Equations (13)-(14).

- *Lagrange Remainder*: The remainder $R_d(\mathbf{x})$ will include all the terms with order higher than d , which can reduce the approximation errors.

3.4 Taylor's Theorem based Machine Learning Models

In real-world problems, the underlying functional mappings are often more intricate, such as $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$ with multiple input variables and multiple outputs. Representing these functions with Taylor's polynomials requires more cumbersome derivations, and the coefficient fabrication outputs should be a two-dimensional matrix, such as $\psi(\mathbf{a}) \in \mathbb{R}^{n \times D}$. To avoid getting bogged down in unnecessary mathematical details, we will not repeat those derivations here.

In recent years, there has been a growing interest in designing machine learning and deep learning models based on Taylor's theorem. For binary data inputs, Zhang *et al.* [86] introduce the reconciled polynomial machine to unify shallow and deep learning models, which is also the prior work that this paper is based on. Balduzzi *et al.* [4] investigate the convergence and exploration in rectifier networks with neural Taylor approximations, while Chrysos *et al.* [11] propose a new class of function approximation method based on polynomial expansions. Zhao *et al.* [89] propose a generic neural architecture TaylorNet based on tensor decomposition to initialize the models, and Nivron *et al.* [56] introduce to incorporate use Taylor's expansion as a wrapper of transformer for the probabilistic predictions for time series and other random processes. Beyond time series and continuous

function approximation, Taylor’s expansion has found applications in reinforcement learning and computer vision. [72] investigates the application of Taylor’s expansions in reinforcement learning and introduces the Taylor expansion policy optimization to generalize prior work; and [61] introduces a simple augmentation to standard behavior cloning losses in the context of continuous control for Taylor series imitation learning. In image processing, [92] proposes to use Taylor’s formula to construct a novel framework for image restoration, and [81] proposes the Taylor neural net for image super-resolution.

Different from these prior work, since the underlying function f is unknown, we cannot directly employ the above derivations and Taylor’s expansions to define approximated polynomial representations of f for practical applications. Drawing inspiration from the approximation architecture delineated in Equation 7 and Equation 15, we propose a novel approach that defines distinct component functions to substitute the data vector, coefficient vector, and remainder terms. Additionally, as the input variable \mathbf{x} varies across instances, instead of manually selecting one single fixed constant \mathbf{a} , we propose to define it as multi-channel parameters and learn them instead. These innovations form the foundation of our proposed Reconciled Polynomial Network (our) model to be introduced in the following section.

4 RPN: Reconciled Polynomial Network for Deep Function Learning

Based on the preliminary background introduced above and inspired by the work of [86], we will introduce the Reconciled Polynomial Network (RPN) model for function learning in this section.

4.1 RPN: Reconciled Polynomial Network

Formally, given the underlying data distribution mapping $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$, we represent the RPN model proposed to approximate function f as follows:

$$g(\mathbf{x}|\mathbf{w}) = \langle \kappa(\mathbf{x}), \psi(\mathbf{w}) \rangle + \pi(\mathbf{x}), \quad (18)$$

where

- $\kappa : \mathbb{R}^m \rightarrow \mathbb{R}^D$ is named as the **data expansion function** and D is the target expansion space dimension.
- $\psi : \mathbb{R}^l \rightarrow \mathbb{R}^{n \times D}$ is named as the **parameter reconciliation function**, which is defined only on the parameters without any input data.
- $\pi : \mathbb{R}^m \rightarrow \mathbb{R}^n$ is named as the **remainder function**.

The architecture of RPN is also illustrated in Figure 4. The RPN model disentangles input data from model parameters through the expansion functions κ and reconciliation function ψ . More detailed information about all these components and modules mentioned in Figure 4 will be introduced in the following parts of this section.

4.2 RPN Component Functions

The **data expansion function** κ projects input data into a new space with different basis vectors, where the target vector space dimension D is determined when defining κ . In practice, the function κ can either expand or compress the input to a higher- or lower-dimensional space. The corresponding function, κ , can also be referred to as the data expansion function (if $D > m$) and data compression function (if $D < m$), respectively. Collectively, these can be unified under the term “**data transformation functions**”. In this paper, we focus on expanding the inputs to a higher-dimensional space, and will use the function names “data transformation” and “data expansion” interchangeably in the following sections.

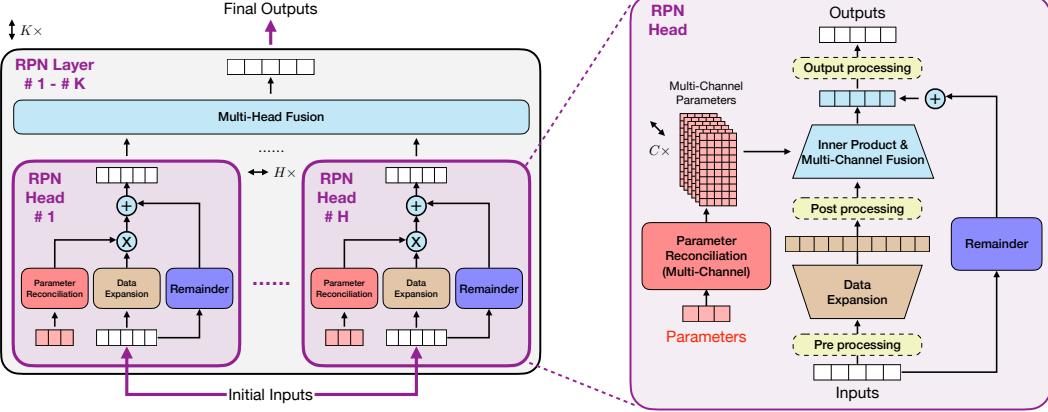


Figure 4: An illustration of the RPN framework. The left plot illustrates the multi-layer (K -layer) architecture of RPN. Each layer involves multi-head for function learning, whose outputs will be fused together. The right plot illustrates the detailed architecture of the RPN head, involving data expansion, multi-channel parameter reconciliation, remainder, and their internal operations. The components with yellow color in dashed lines denote the optional data processing functions (e.g., activation functions and norm functions) for the inputs, expansions and outputs.

Meanwhile, the **parameter reconciliation function** ψ adjusts the available parameter vector of length l by fabricating a new parameter matrix of size $n \times D$ to accommodate the expansion space dimension D defined by function κ . In most of the cases studied in this paper, the parameter vector length l is much smaller than the output matrix size $n \times D$, i.e., $l \ll n \times D$. Meanwhile, in practice, we can also define function ψ to fabricate a longer parameter vector into a smaller parameter matrix, i.e., $l > n \times D$. To unify these different cases, the data reconciliation function can also be referred to as the “**parameter fabrication function**”, and these function names will be used interchangeably in this paper.

Without specific descriptions, the **remainder function** π defined here is based solely on the input data \mathbf{x} . However, in practice, we also allow π to include learnable parameters for output dimension adjustment. In such cases, it should be rewritten as $\pi(\mathbf{x}|\mathbf{w}')$, where \mathbf{w}' is one extra fraction of the model’s learnable parameters. Together with the parameter vector \mathbf{w} (i.e., the input to the parameter reconciliation function ψ), they will define the complete set of learnable parameters for the model.

4.3 Wide RPN: Multi-Head and Multi-Channel Model Architecture

Similar to the Transformer with multi-head attention [78], as shown in Figure 4, the RPN model employs a multi-head architecture, where each head can disentangle the input data and model parameters using different expansion, reconciliation and remainder functions, respectively:

$$g(\mathbf{x}|\mathbf{w}, H) = \sum_{h=0}^{H-1} \left\langle \kappa^{(h)}(\mathbf{x}), \psi^{(h)}(\mathbf{w}^{(h)}) \right\rangle + \pi^{(h)}(\mathbf{x}), \quad (19)$$

where the superscript “ h ” indicates the head index and H denotes the total head number. By default, we use summation to combine the results from all these heads.

Moreover, in the RPN model shown in Figure 4, similar to convolutional neural networks (CNNs) employing multiple filters, we allow each head to have multiple channels of parameters applied to the same data expansion. For example, for the h_{th} head, we define its multi-channel parameters as $\mathbf{w}^{(h),0}, \mathbf{w}^{(h),1}, \dots, \mathbf{w}^{(h),C-1}$, where C denotes the number of channels. These parameters will be reconciled using the same parameter reconciliation function, as shown below:

$$g(\mathbf{x}|\mathbf{w}, H, C) = \sum_{h=0}^{H-1} \sum_{c=0}^{C-1} \left\langle \kappa^{(h)}(\mathbf{x}), \psi^{(h)}(\mathbf{w}^{(h),c}) \right\rangle + \pi^{(h)}(\mathbf{x}), \quad (20)$$

The multi-head, multi-channel design of the RPN model allows it to project the same input data into multiple different high-dimensional spaces simultaneously. Each head and channel combination may potentially learn unique features from the data. The unique parameters at different heads can have different initialized lengths, and each of them will be processed in unique ways to accommodate the expanded data. This multi-channel approach provides our model with more flexibility in model design. In the following parts of this paper, to simplify the notations, we will illustrate the model's functional components using a single-head, single-channel architecture by default. However, readers should note that these components to be introduced below can be extended to their multi-head, multi-channel designs in practical implementations.

4.4 Deep RPN: Multi-Layer Model Architecture

The wide model architecture introduced above provides RPN with greater capabilities for approximating functions with diverse expansions concurrently. However, such shallow architectures can be insufficient for modeling complex functions. In this paper, as illustrated in Figure 4, we propose to stack RPN layers on top of each other to build a deeper architecture, where the Equation (18) actually defines one single layer of the model. Formally, we can represent the deep RPN with multi-layers as follows:

$$\begin{cases} \text{Input: } \mathbf{h}_0 = \mathbf{x}, \\ \text{Layer 1: } \mathbf{h}_1 = \langle \kappa_1(\mathbf{h}_0), \psi_1(\mathbf{w}_1) \rangle + \pi_1(\mathbf{h}_0), \\ \text{Layer 2: } \mathbf{h}_2 = \langle \kappa_2(\mathbf{h}_1), \psi_2(\mathbf{w}_2) \rangle + \pi_2(\mathbf{h}_1), \\ \dots \quad \dots \dots \\ \text{Layer K: } \mathbf{h}_K = \langle \kappa_K(\mathbf{h}_{K-1}), \psi_K(\mathbf{w}_K) \rangle + \pi_K(\mathbf{h}_{K-1}), \\ \text{Output: } \hat{\mathbf{y}} = \mathbf{h}_K. \end{cases} \quad (21)$$

The subscripts used above denote the layer index. The dimensions of the outputs at each layer can be represented as a list $[d_0, d_1, \dots, d_{K-1}, d_K]$, where $d_0 = m$ and $d_K = n$ denote the input and the desired output dimensions, respectively. Therefore, if the component functions at each layer of our model have been predetermined, we can just use the dimension list $[d_0, d_1, \dots, d_{K-1}, d_K]$ to represent the architecture of the RPN model.

4.5 Versatile RPN: Nested and Extended Expansion Functions

The data expansion function introduced earlier projects the input data to a higher-dimensional space. There exist different ways to define the data expansion function, and a list of such basic expansion functions will be introduced in the following Section 5.1. The multi-head, multi-channel and multi-layer architecture also provides RPN with more capacity to build wider and deeper architectures for projecting input data to the desired target space. In addition to these designs, as illustrated in Figure 5, RPN also provides a more flexible and lightweight mechanism to build models with similar capacities via the nested and extended data expansion functions.

Nested expansions: Formally, given a list of n data expansion functions $\kappa_1 : \mathbb{R}^{d_0} \rightarrow \mathbb{R}^{d_1}$, $\kappa_2 : \mathbb{R}^{d_1} \rightarrow \mathbb{R}^{d_2}$, \dots , $\kappa_n : \mathbb{R}^{d_{n-1}} \rightarrow \mathbb{R}^{d_n}$, as shown in Plots (a)-(b) of Figure 5, the nested calls of these functions will project a data vector from the input space \mathbb{R}^{d_0} to the desired output space \mathbb{R}^{d_n} , defining the nested data expansion function $\kappa : \mathbb{R}^m \rightarrow \mathbb{R}^D$ as follows:

$$\kappa(\mathbf{x}) = \kappa_n(\kappa_{n-1}(\dots \kappa_2(\kappa_1(\mathbf{x})))) \in \mathbb{R}^D. \quad (22)$$

where the function input and output dimensions should be $d_0 = m$ and $d_n = D$.

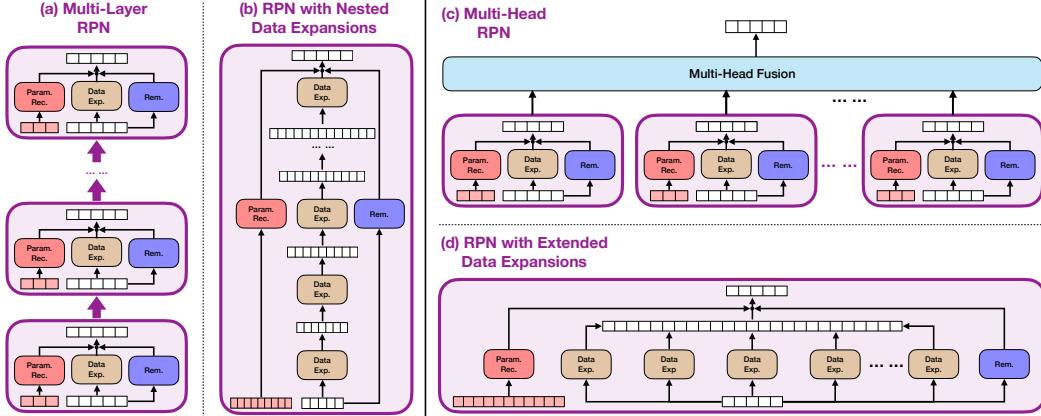


Figure 5: An illustration of the RPN layer with nested and extended data expansions. Plot (a): multi-layer RPN; Plot (b): single-layer RPN with nested data expansions; Plot (c): multi-head RPN; Plot (d): single-head RPN with extended data expansions.

Extended expansions: In addition to nesting these n expansion functions, as shown in Plots (c)-(d) of Figure 5, they can also be concatenated and applied concurrently, with their extended outputs allowing the model to leverage multiple expansion functions simultaneously. Formally, we can represent the extended data expansion function $\kappa : \mathbb{R}^m \rightarrow \mathbb{R}^D$ defined based on $\kappa_1 : \mathbb{R}^m \rightarrow \mathbb{R}^{d_1}$, $\kappa_2 : \mathbb{R}^m \rightarrow \mathbb{R}^{d_2}$, \dots , $\kappa_n : \mathbb{R}^m \rightarrow \mathbb{R}^{d_n}$ as follows:

$$\kappa(\mathbf{x}) = [\kappa_1(\mathbf{x}), \kappa_2(\mathbf{x}), \dots, \kappa_n(\mathbf{x})] \in \mathbb{R}^D, \quad (23)$$

where the extended expansion’s output dimension is equal to the sum of the output dimensions from all the individual expansion functions, *i.e.*, $D = \sum_{i=1}^n d_i$.

As illustrated in Figure 5, the nested expansion functions can define complex expansions akin to the multi-layer architecture of RPN mentioned above. Meanwhile, the extended expansion functions can define expansions similar to the multi-head architecture of RPN. Both nested and extended expansions allow for faster data expansions, circumventing cumbersome parameter inference and remainder function calculation, and can reduce the additional learning costs associated with training deep and wide architectures of our model. This flexibility afforded by nested and extended expansions provides us with greater versatility in designing the RPN model.

4.6 Learning Correctness of RPN: Complexity, Capacity and Completeness

The **learning correctness** of RPN is fundamentally determined by the compositions of its component functions, each contributing from different perspectives:

- **Model Complexity:** The data expansion function κ expands the input data by projecting its representations using basis vectors in the new space. In other words, function κ determines the upper bound of the RPN model’s complexity.
- **Model Capacity:** The reconciliation function ψ processes the parameters to match the dimensions of the expanded data vectors. The reconciliation function and parameters jointly determine the learning capacity and associated training costs of the RPN model.
- **Model Completeness:** The remainder function π completes the approximation as a residual term, governing the learning completeness of the RPN model.

In the following Section 5, we will introduce several different representations for the data expansion function κ , parameter reconciliation function ψ , and remainder function π . By strategically

combining these component functions, we can construct a multi-head, multi-channel, and multi-layer architecture, enabling RPN to address a wide spectrum of learning challenges across diverse learning tasks.

4.7 Learning Cost of RPN: Space, Time and Parameter Number

To analyze the learning costs of RPN, we can take a batch input $\mathbf{X} \in \mathbb{R}^{B \times m}$ of batch size B as an example, which will be fed to the RPN model with K layers, each with H heads and each head has C channels. Each head will project the data instance from a vector of length m to an expanded vector of length D and then further projected to the desired output of length n . Each channel reconciles parameters from length l to the sizes determined by both the expansion space and output space dimensions, *i.e.*, $n \times D$.

Based on the above hyper-parameters, assuming the input and output dimensions at each layer are comparable to m and n , then the space, time costs and the number of involved parameters in learning the RPN model are calculated as follows:

- **Space Cost:** The total space cost for data (including the inputs, expansions and outputs) and parameter (including raw parameters, fabricated parameters generated by the reconciliation function and optional remainder function parameters) can be represented as
$$\mathcal{O}(KH(B(\underbrace{\underbrace{m}_{\text{input}} + \underbrace{D}_{\text{expansion}} + \underbrace{n}_{\text{output}})}_{\text{space cost for data}}) + C(\underbrace{\underbrace{l}_{\text{raw param.}} + \underbrace{nD}_{\text{reconciled param.}})}_{\text{space cost for parameters}} + \underbrace{mn}_{\text{(optional) remainder param.}})).$$
- **Time Cost:** Depending on the expansion and reconciliation functions used for building RPN, the total time cost of RPN can be represented as
$$\mathcal{O}(KH(\underbrace{t_{exp}(m, D)}_{\text{time cost for data exp.}} + \underbrace{Ct_{rec}(l, D)}_{\text{time cost for param. rec.}} + \underbrace{CmnD}_{\text{time cost for inner product}} + \underbrace{mn}_{\text{(optional) time cost for remainder}})),$$
where notations $t_{exp}(m, D)$ and $t_{rec}(l, D)$ denote the expected time costs for data expansion and parameter reconciliation functions, respectively.
- **Learnable parameters:** The total number of parameters in RPN will be $\mathcal{O}(KHC_l + KHmn)$, where $\mathcal{O}(KHmn)$ denotes the optional parameter number used for defining the remainder function.

5 List of Expansion, Reconciliation and Remainder Functions for RPN Model

This section introduces the expansion, reconciliation, and remainder functions that can be used to design the RPN model, all of which have been implemented in the **TINYBIG** toolkit and are readily available. Readers seeking a concise overview can refer to Figure 6, which summarizes the lists of expansion, reconciliation and remainder functions to be introduced in this section.

5.1 Data Expansion Functions

The **data expansion function** determines the complexity of RPN. We will introduce several different data expansion functions below. In real-world practice, these individual data expansions introduced below can also be nested and extended to define more complex expansions, which provides more flexibility in the design of our RPN model.

5.1.1 Identity and Reciprocal Data Expansion

The simplest data expansion methods are the identity data expansion and reciprocal data expansion, which project the input data vector $\mathbf{x} \in \mathbb{R}^m$ onto itself and its reciprocal, potentially with minor transformations via some activation functions, as denoted below:

$$\kappa(\mathbf{x}) = \mathbf{x} \in \mathbb{R}^D, \text{ and } \kappa(\mathbf{x}) = \frac{1}{\mathbf{x}} \in \mathbb{R}^D, \quad (24)$$

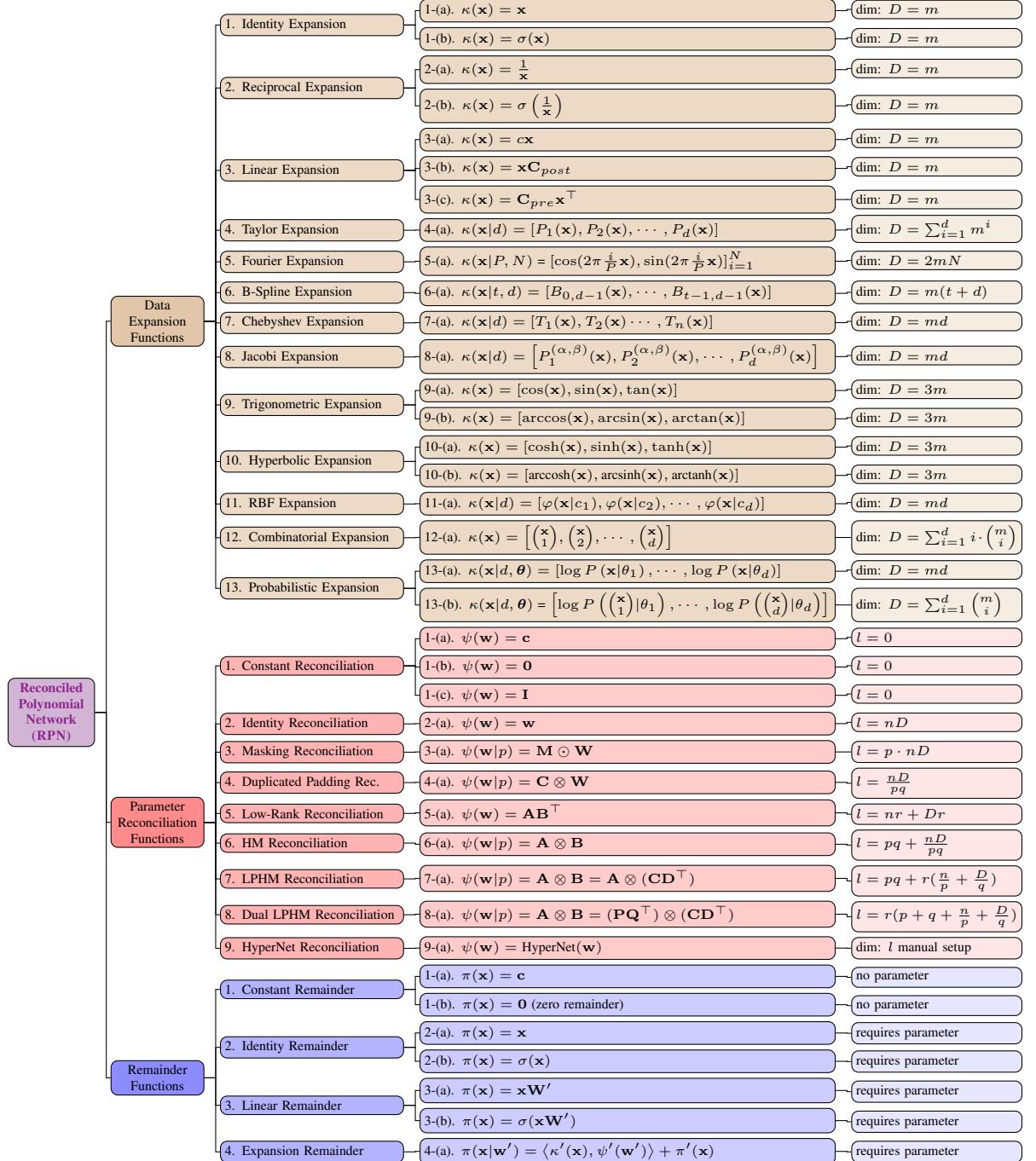


Figure 6: An overview of data expansion, parameter reconciliation, and remainder functions implemented in the **TINYBIG** toolkit for constructing the RPN model architecture.

or

$$\kappa(\mathbf{x}) = \sigma(\mathbf{x}) \in \mathbb{R}^D, \text{ and } \kappa(\mathbf{x}) = \sigma\left(\frac{1}{\mathbf{x}}\right) \in \mathbb{R}^D. \quad (25)$$

In the above equations, σ denotes an optional activation function (e.g., sigmoid, ReLU, SiLU) or normalization function (e.g., layer-norm, batch-norm, instance-norm). For both the identity and reciprocal expansion functions, their output dimension is equal to the input dimension, *i.e.*, $D = m$.

For all the other expansion functions introduced hereafter, as mentioned in the previous Figure 4, we can also apply the (optional) activation and normalization functions both before and after the expansion by default.

5.1.2 Linear Data Expansion

In certain cases, we may need to adjust the value scales of \mathbf{x} linearly without altering the basis vectors or the dimensions of the space. This can be accomplished through the linear data expansion function. Formally, the linear data expansion function projects the input data vector $\mathbf{x} \in \mathbb{R}^m$ onto itself via linear projection, as follows:

$$\kappa(\mathbf{x}) = c\mathbf{x} \in \mathbb{R}^D, \quad (26)$$

or

$$\kappa(\mathbf{x}) = \mathbf{x}\mathbf{C}_{post} \in \mathbb{R}^D, \quad (27)$$

or

$$\kappa(\mathbf{x}) = \mathbf{C}_{pre}\mathbf{x}^\top \in \mathbb{R}^D, \quad (28)$$

where the activation function or norm function σ is optional, and $c \in \mathbb{R}$, $\mathbf{C}_{post}, \mathbf{C}_{pre} \in \mathbb{R}^{m \times m}$ denote the provided constant scalar and linear transformation matrices, respectively. Linear data expansion will not change the data vector dimensions, and the output data vector dimension $D = m$.

5.1.3 Taylor's Polynomials based Data Expansions

Given a vector $\mathbf{x} = [x_1, x_2, \dots, x_m] \in \mathbb{R}^m$ of dimension m , the multivariate composition of order d defined based on \mathbf{x} can be represented as a list of potential polynomials composed by the product of the vector elements x_1, x_2, \dots, x_m , where the sum of the degrees equals d , i.e.,

$$P_d(\mathbf{x}) = [x_1^{d_1} x_2^{d_2} \cdots x_m^{d_m}]_{d_1, d_2, \dots, d_m \in \{0, 1, \dots, m\} \wedge \sum_{i=1}^m d_i = d}. \quad (29)$$

Some examples of the multivariate polynomials are provided as follows:

$$\begin{aligned} P_0(\mathbf{x}) &= [1] \in \mathbb{R}^1, \\ P_1(\mathbf{x}) &= [x_1, x_2, \dots, x_m] \in \mathbb{R}^m, \\ P_2(\mathbf{x}) &= [x_1^2, x_1x_2, x_1x_3, \dots, x_1x_m, x_2x_1, x_2^2, x_2x_3, \dots, x_mx_m] \in \mathbb{R}^{m^2}. \end{aligned} \quad (30)$$

We observe that the above representation of $P_2(\mathbf{x})$ may contain duplicated elements, e.g., x_1x_2 and x_2x_1 . However, this representation simplifies the implementation, and high-order polynomials can be recursively calculated using the Kronecker product operator based on the lower-order ones.

$1 \otimes 0$	$1 \otimes 3$	0	6
$1 \otimes 2$	$1 \otimes 1$	4	2
$3 \otimes 0$	$3 \otimes 3$	0	12
$3 \otimes 2$	$3 \otimes 1$	8	4

Figure 7: An illustration of Kronecker product on matrices \mathbf{A} and \mathbf{B} .

DEFINITION 2 (Kronecker product): Formally, as illustrated in Figure 7, given two matrices $\mathbf{A} \in \mathbb{R}^{p \times q}$ and $\mathbf{B} \in \mathbb{R}^{s \times t}$, the Kronecker product of \mathbf{A} and \mathbf{B} is defined as

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{1,1}\mathbf{B} & a_{1,2}\mathbf{B} & \cdots & a_{1,q}\mathbf{B} \\ a_{2,1}\mathbf{B} & a_{2,2}\mathbf{B} & \cdots & a_{2,q}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{p,1}\mathbf{B} & a_{p,2}\mathbf{B} & \cdots & a_{p,q}\mathbf{B} \end{bmatrix} \in \mathbb{R}^{ps \times qt}, \quad (31)$$

where the output will be a larger matrix with $p \times s$ rows and $q \times t$ columns.

The Kronecker product can also be applied to vectors, with the base polynomial vectors $P_0(\mathbf{x})$ and $P_1(\mathbf{x})$, the other Taylor's polynomials with higher orders can all be recursively defined as follows:

$$P_d(\mathbf{x}) = P_1(\mathbf{x}) \otimes P_{d-1}(\mathbf{x}), \text{ for } \forall d \geq 2. \quad (32)$$

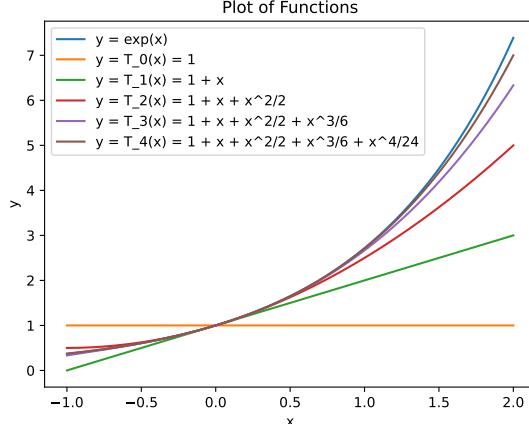
With the notation $P_d(\mathbf{x})$, we can define the *Taylor's polynomials* based data expansion function as the list of polynomial terms with orders no greater than d (where d is a hyper-parameter of the function) as follows:

$$\kappa(\mathbf{x}|d) = [P_1(\mathbf{x}), \dots, P_d(\mathbf{x})] \in \mathbb{R}^D. \quad (33)$$

Since RPN has a deep architecture, the hyper-parameter d is normally set to a small value (e.g., $d = 2$) to avoid excessively large expansions at each layer. Expanding the input data to a Taylor's polynomial of order 4 can be achieved either by stacking two layers of RPN layer or by nesting two Taylor's expansion functions with $d = 2$. Additionally, the base term $P_0(\mathbf{x})$ containing the constant value '1' can be subsumed by the bias term in the inner product implementation, and thus need not be explicitly included in the expansion. The output dimension will then be $D = \sum_{i=1}^d m^i$.

Taylor's polynomials are known to approximate functions very well, and an illustrative example of their approximation correctness on function $y = \exp(x)$ is provided below.

Example: In the right plot, we illustrate the approximation of function $f(x) = \exp(x)$ with Taylor's polynomials of different orders, where the notation $T_d(x)$ denotes the Taylor's polynomials with degrees up to d . According to the plot, increasing the degree allows the Taylor's polynomials to approximate the function $f(x)$ more accurately. Among all these illustrated Taylor's polynomials in the plot, $T_4(x) = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24}$ outperforms the others.



5.1.4 Fourier Series based Data Expansions

A Fourier series is an expansion of a periodic function into the sum of Fourier series. In mathematics, the Dirichlet-Jordan test gives sufficient conditions for a real-valued, periodic function to be equal to the sum of its Fourier series at a point of continuity. Fourier series can be represented in several forms, and in this paper, we will utilize the sine-cosine representation.

Based on the hyper-parameters P and N , we can represent the Fourier series based data expansion function κ for the input vector \mathbf{x} as follows:

$$\kappa(\mathbf{x}|P, N) = \left[\cos(2\pi \frac{1}{P}\mathbf{x}), \sin(2\pi \frac{1}{P}\mathbf{x}), \cos(2\pi \frac{2}{P}\mathbf{x}), \sin(2\pi \frac{2}{P}\mathbf{x}), \dots, \cos(2\pi \frac{N}{P}\mathbf{x}), \sin(2\pi \frac{N}{P}\mathbf{x}) \right] \in \mathbb{R}^D, \quad (34)$$

where the output dimension $D = 2mN$.

5.1.5 B-Splines based Data Expansion

Formally, a B-spline of degree $d + 1$ is defined as a collection of piecewise polynomial functions $\{B_{i,d}(x)\}_{i \in \{0, 1, \dots, t-1\}}$ of degree d over the variable x , which takes values from a pre-defined value range $[x_0, x_t]$. The value range $[x_0, x_t]$ is divided into smaller pieces by points $x_0, x_1, x_2, \dots, x_t$ sorted in a non-decreasing order, and these points are also known as the **knots**. These knots partition the value range $[x_0, x_t]$ into t disjoint intervals: $[x_0, x_1], [x_1, x_2], \dots, [x_{t-1}, x_t]$.

As to the specific representations of B-splines, they can be defined recursively based on the lower-degree terms according to the following equations:

Base B-splines with degree $d = 0$:

$$\{B_{0,0}(x), B_{1,0}(x), \dots, B_{t-1,0}(x)\}, \quad (35)$$

where

$$B_{i,0}(x) = \begin{cases} 1, & \text{if } x_i \leq x < x_{i+1}; \\ 0, & \text{otherwise.} \end{cases} \quad (36)$$

Higher-degree B-splines with $d > 0$:

$$\{B_{0,d}(x), B_{1,d}(x), \dots, B_{t-1,d}(x)\}, \quad (37)$$

where

$$B_{i,d}(x) = \frac{x - x_i}{x_{i+d} - x_i} B_{i,d-1}(x) + \frac{x_{i+d+1} - x}{x_{i+d+1} - x_{i+1}} B_{i+1,d-1}(x). \quad (38)$$

According to the representations, term $B_{i,d}(x)$ recursively defined above will have non-zero outputs if and only if the inputs lie within the value range $x_i \leq x < x_{i+d+1}$.

B-splines have been extensively used in curve-fitting and numerical differentiation of experimental data, including their recent application in the design of KAN [51]. In this paper, we define the B-spline-based data expansion function with degree d , which can be represented as follows:

$$\kappa(\mathbf{x}|d) = [B_{0,d}(\mathbf{x}), B_{1,d}(\mathbf{x}), \dots, B_{t-1,d}(\mathbf{x})] \in \mathbb{R}^D, \quad (39)$$

where the output dimension can be calculated as $D = m(t + d)$.

5.1.6 Chebyshev Polynomials based Data Expansion

In addition to B-splines, we observe several other similar basis functions recursively defined based on those of lower degrees, including Chebyshev polynomials and Jacobi polynomials.

Chebyshev polynomials have been demonstrated to be important in approximation theory for the solution of linear systems. They can be represented as two sequences of polynomials related to the cosine and sine functions (also known as the first-kind and second-kind), with recursive calculation equations that are quite similar to each other, differing only in scalar coefficients. In this paper, we will use the Chebyshev polynomials of the first kind (*i.e.*, defined based on the cosine function) and it can be represented with the following recursive equations.

Base cases $d = 0$ and $d = 1$:

$$T_0(x) = 1, \text{ and } T_1(x) = x \quad (40)$$

High-order cases with degree $d \geq 2$:

$$T_d(x) = 2x \cdot T_{d-1}(x) - T_{d-2}(x). \quad (41)$$

Based on the above representations, in this paper, we define the Chebyshev polynomials based data expansion function with degree hyper-parameter $d \geq 1$ as follows:

$$\kappa(\mathbf{x}|d) = [T_1(\mathbf{x}), T_2(\mathbf{x}), \dots, T_d(\mathbf{x})] \in \mathbb{R}^D. \quad (42)$$

Similar to the aforementioned Taylor's polynomial-based data expansions, since the output of $T_0(x)$ is a constant, it will not be included in the expansion function definition by default, and the output dimension will be $D = md$.

5.1.7 Jacobi Polynomials based Data Expansion

Different from the Chebyshev polynomial, the Jacobi polynomials have a more complicated recursive representation. Formally, the Jacobi polynomials parameterized by α and β of degree $d \geq 2$ on variable x can be represented as $P_d^{(\alpha,\beta)}(x)$, which can be recursively defined based on the lower-order cases:

$$\begin{aligned} P_d^{(\alpha,\beta)}(x) &= \frac{(2d + \alpha + \beta - 1) [(2d + \alpha + \beta)(2d + \alpha + \beta - 2)x + (\alpha^2 - \beta^2)]}{2d(d + \alpha + \beta)(2d + \alpha + \beta - 2)} P_{d-1}^{(\alpha,\beta)}(x) \\ &\quad - \frac{2(d + \alpha - 1)(d + \beta - 1)(2d + \alpha + \beta)}{2d(d + \alpha + \beta)(2d + \alpha + \beta - 2)} P_{d-2}^{(\alpha,\beta)}(x). \end{aligned} \quad (43)$$

As to the base case, we list some of them as follows:

$$\begin{aligned} P_0^{(\alpha,\beta)}(x) &= 1, \\ P_1^{(\alpha,\beta)}(x) &= (\alpha + 1) + (\alpha + \beta + 2) \frac{(x - 1)}{2}, \\ P_2^{(\alpha,\beta)}(x) &= \frac{(\alpha + 1)(\alpha + 2)}{2} + (\alpha + 2)(\alpha + \beta + 3) \frac{x - 1}{2} + \frac{(\alpha + \beta + 3)(\alpha + \beta + 4)}{2} \left(\frac{x - 1}{2} \right)^2. \end{aligned} \quad (44)$$

In this paper, we define the Jacobi polynomial based data expansion function with degree d as follows:

$$\kappa(\mathbf{x}|d) = [P_1^{(\alpha,\beta)}(\mathbf{x}), P_2^{(\alpha,\beta)}(\mathbf{x}), \dots, P_d^{(\alpha,\beta)}(\mathbf{x})] \in \mathbb{R}^D, \quad (45)$$

where the output dimension $D = md$.

The Jacobi polynomials belong to the family of classic orthogonal polynomials, where two different polynomials in the sequence are orthogonal to each other under some inner product. Meanwhile, the Gegenbauer polynomials form the most important class of Jacobi polynomials, and Chebyshev polynomial is a special case of the Gegenbauer polynomials. In addition to these, we will also gradually incorporate other classic orthogonal polynomials into our **TINYBIG** toolkit.

5.1.8 Hyperbolic Function and Trigonometric Function based Data Expansions

The Fourier series introduced above is actually an example of a trigonometric series. In addition to Fourier series, we also include several other types of trigonometric functions based data expansion approach in this paper, such as *hyperbolic functions*, shown as follows:

$$\kappa(\mathbf{x}) = [\cosh(\mathbf{x}), \sinh(\mathbf{x}), \tanh(\mathbf{x})] \in \mathbb{R}^D, \text{ where } D = 3m. \quad (46)$$

In addition to the hyperbolic functions, we can also define the data expansion function using inverse hyperbolic functions, trigonometric functions, and inverse trigonometric functions, as follows:

$$\kappa(\mathbf{x}) = [\operatorname{arccosh}(\mathbf{x}), \sinh(\mathbf{x}), \tanh(\mathbf{x})] \in \mathbb{R}^D; \quad (47)$$

$$\kappa(\mathbf{x}) = [\cos(\mathbf{x}), \sin(\mathbf{x}), \tan(\mathbf{x})] \in \mathbb{R}^D; \quad (48)$$

$$\kappa(\mathbf{x}) = [\arccos(\mathbf{x}), \arcsin(\mathbf{x}), \arctan(\mathbf{x})] \in \mathbb{R}^D, \quad (49)$$

where the output dimensions of these expansion functions are all $D = 3m$.

Unlike hyperbolic functions, trigonometric functions are periodic, and different input values of x may be projected to identical outputs, rendering them indistinguishable. This can potentially lead to

degraded performance. Nonetheless, the above trigonometric function-based data expansion function can be employed as intermediate layers or complementary heads within a layer to compose more complex functions and construct more powerful models.

5.1.9 Radial Basis Functions based Data Expansion

In mathematics, a radial basis function (RBF) is a real-valued function φ defined based on the distance between the input x and some fixed point, e.g., c , which can be represented as follows:

$$\varphi(x|c) = \varphi(x - c). \quad (50)$$

There are different ways to define the function φ in practice, two of which studied in this paper are shown as follows:

(a) Gaussian RBF:

$$\varphi(x|c) = e^{-(\epsilon(x-c))^2}, \quad (51)$$

where ϵ is a hyperparameter.

(b) Inverse Quadratic RBF:

$$\varphi(x|c) = \frac{1}{1 + (\epsilon(x-c))^2}, \quad (52)$$

where ϵ is a hyperparameter.

Given a set of d different fixed points, e.g., $\mathbf{c} = [c_1, c_2, \dots, c_d]$, a sequence of d different such RBF can be defined, which compare the input against these d different fixed points shown as follows:

$$\varphi(x|\mathbf{c}) = [\varphi(x|c_1), \varphi(x|c_2), \dots, \varphi(x|c_d)] \in \mathbb{R}^d. \quad (53)$$

With the above notations, we can represent the Gaussian RBF and Inverse Quadratic RBF based data expansion functions both with the following equation:

$$\kappa(\mathbf{x}) = \varphi(\mathbf{x}|\mathbf{c}) = [\varphi(\mathbf{x}|c_1), \varphi(\mathbf{x}|c_2), \dots, \varphi(\mathbf{x}|c_d)] \in \mathbb{R}^D, \quad (54)$$

where the output dimension will be $D = md$.

5.1.10 Combinatorial Data Expansion

Combinatorial data expansion function expands input data by enumerating the potential combinations of elements from the input vector, with the number of elements to be combined ranging from $1, 2, \dots, d$, where d is a hyper-parameter. Formally, given a data instance featured by a variable set $\mathcal{X} = \{X_1, X_2, \dots, X_m\}$ (here, we use the upper-case X_i to denote the variable of the i_{th} feature), we can represent the possible combinations of d terms selected from \mathcal{X} with notation:

$$\binom{\mathcal{X}}{d} = \{\mathcal{C} | \mathcal{C} \subset \mathcal{X} \wedge |\mathcal{C}| = d\}, \quad (55)$$

where \mathcal{C} denotes a subset of \mathcal{X} containing no duplicated elements and the size of the output set $\binom{\mathcal{X}}{d}$ will be equal to $\binom{m}{d}$. Some simple examples with $d = 1, d = 2$ and $d = 3$ are illustrated as follows:

$$\begin{aligned} d = 1 : \binom{\mathcal{X}}{1} &= \{\{X_i\} | X_i \in \mathcal{X}\}, \\ d = 2 : \binom{\mathcal{X}}{2} &= \{\{X_i, X_j\} | X_i, X_j \in \mathcal{X} \wedge X_i \neq X_j\}, \\ d = 3 : \binom{\mathcal{X}}{3} &= \{\{X_i, X_j, X_k\} | X_i, X_j, X_k \in \mathcal{X} \wedge X_i \neq X_j \wedge X_i \neq X_k \wedge X_j \neq X_k\}. \end{aligned} \quad (56)$$

By applying the above notations to concrete data instances, given a data instance with values $\mathbf{x} = [x_1, x_2, \dots, x_m]$ (the lower-case x_i denotes the feature value), we can also represent the combinations of d selected features from \mathbf{x} as $\binom{\mathbf{x}}{d}$, which can be used to define the combinatorial data expansion function as follows:

$$\kappa(\mathbf{x}) = \left[\binom{\mathbf{x}}{1}, \binom{\mathbf{x}}{2}, \dots, \binom{\mathbf{x}}{d} \right]. \quad (57)$$

Similar as the above Taylor's expansions, the output dimension of the combinatorial expansion will increase exponentially. Given an input data vector $\mathbf{x} \in \mathbb{R}^m$, with hyper-parameter d , its expansion output of combinations with up to d elements will be $\kappa(\mathbf{x}) \in \mathbb{R}^D$, where $D = \sum_{i=1}^d i \cdot \binom{m}{i}$.

5.1.11 Probabilistic Data Expansion

An important category of data expansion functions that surprisingly performed very well, even outperforming many of the extension approaches mentioned above, are probability density function based data expansions. Formally, in probability theory and statistics, a probability density function (PDF) is a function that describes the relative likelihood for a random variable to take on a given value within its sample space. Formally, given a probabilistic distribution parameterized by θ , we can represent its probability density function as

$$P(x|\theta) \in [0, 1], \text{ where } \int_x P(x|\theta) dx = 1. \quad (58)$$

Lots of probabilistic distributions have been proposed by mathematicians and statisticians, such as *Gaussian distribution* $\mathcal{N}(\mu, \sigma)$, *Exponential distribution* $\mathcal{E}(\lambda)$, *Laplace distribution* $\mathcal{L}(\mu, b)$, *Cauchy distribution* $\mathcal{C}(x_0, \gamma)$, *Chi-squared distribution* $\mathcal{X}^2(k)$ and *Gamma distribution* $\Gamma(k, \theta)$, etc. The PDFs of these distributions are also provided as follows:

(a) Gaussian Distribution:

$$P(x|\mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2},$$

where μ, σ are the mean and std parameters.

(59)

(b) Exponential Distribution:

$$P(x|\lambda) = \begin{cases} \lambda \exp^{-\lambda x} & \text{for } x \geq 0, \\ 0 & \text{otherwise,} \end{cases}, \quad (60)$$

where $\lambda > 0$ is the rate parameter.

(c) Laplace Distribution:

$$P(x|\mu, b) = \frac{1}{2b} \exp(-\frac{|x-\mu|}{b}),$$

where $\mu, b > 0$ are the location, scale parameters.

(61)

(d) Cauchy Distribution:

$$P(x|x_0, \gamma) = \frac{1}{\pi\gamma \left[1 + \left(\frac{x-x_0}{\gamma} \right)^2 \right]},$$

where x_0, γ are the location and scale parameters.

(62)

(e) Chi-Squared Distribution:

$$P(x|k) = \frac{1}{2^{\frac{k}{2}} \Gamma(\frac{k}{2})} x^{(\frac{k}{2}-1)} \exp^{-\frac{x}{2}}, \quad (63)$$

where $k \in \mathbb{N}^+$ is the dof parameter.

(f) Gamma Distribution:

$$P(x|k, \theta) = \frac{1}{\Gamma(k)\theta^k} x^{k-1} \exp^{-\frac{x}{\theta}},$$

where $k, \theta > 0$ are the shape and scale parameters.

(64)

When feeding inputs to a probability density function, its output is typically a very small number, and the curve of many distribution PDFs can be quite flat (*i.e.*, with a very small slope). In this paper, we propose using the log-likelihood to expand the input data instead, which makes it possible to unify the representations of probabilistic graphical models with RPN, more information of which will be introduced in Section 6.4.

In this paper, we introduce two different expansions based on the probabilistic distributions, *i.e.*, **naive probabilistic expansion** and **combinatorial probabilistic expansion** introduced as follows.

Naive Probabilistic Data Expansion

The naive probabilistic expansion assumes the input features are independent and directly applies the distribution PDFs to the input data vector to calculate the corresponding log-likelihood scores as

the outputs. To expand the inputs, a set of d identical (or different) distribution PDFs with different hyper-parameters can be used to concurrently calculate the log-likelihood scores. The concatenated outputs from these PDFs are then returned as the expansions.

Formally, given the input $\mathbf{x} \in \mathbb{R}^m$, the naive probabilistic function assuming all the elements in \mathbf{x} to be independent will compute the log-likelihood to sample each of the features in the instance according to certain distributions:

$$\kappa(\mathbf{x}|\boldsymbol{\theta}) = [\log P(\mathbf{x}|\theta_1), \log P(\mathbf{x}|\theta_2), \dots, \log P(\mathbf{x}|\theta_d)] \in \mathbb{R}^D, \quad (65)$$

where $\boldsymbol{\theta} = [\theta_1, \theta_2, \dots, \theta_d]$ denotes d different hyper-parameters used for the PDFs. For the input data vector of length m , it is easy to obtain its expansion output dimension via the above function will be $D = md$.

In addition to using a single distribution PDF (with different hyper-parameters), we also introduce a hybrid naive probabilistic expansion approach that simultaneously employs PDFs of different distributions. For example, $P_1(\cdot|\theta_1)$ denotes the normal distribution with mean/std denoted by θ_1 , while $P_2(\cdot|\theta_2)$ denotes the Cauchy distribution with location/scale denoted by θ_2 , and so on. The above expansion function can also be rewritten as follows:

$$\kappa(\mathbf{x}|\boldsymbol{\theta}) = [\log P_1(\mathbf{x}|\theta_1), \log P_2(\mathbf{x}|\theta_2), \dots, \log P_d(\mathbf{x}|\theta_d)] \in \mathbb{R}^D, \quad (66)$$

where d different distribution PDFs P_1, P_2, \dots, P_d are concatenated to define the expansion function here. The output dimension will remain the same as the above, *i.e.*, $D = md$.

For the PDF of the Gamma, Chi-square and Exponential distributions, they may require non-negative inputs (some may also require the input to be non-zero). Therefore, prior to feeding the input vector \mathbf{x} to their PDF for expansion, we need to pre-transform \mathbf{x} into positive vectors (with either activation functions or normalization functions).

Combinatorial Probabilistic Data Expansion

Based on the above combinatorial expansions and multivariate distributions, we can introduce the combinatorial probabilistic expansion function. Distinct from naive probabilistic data expansion functions, combinatorial probabilistic data expansion function considers the relationships among variables in the multivariate distribution PDFs, which can model complex data distributions better.

Formally, given the input data vector $\mathbf{x} \in \mathbb{R}^m$, the combinatorial probabilistic expansion function defined based on the multivariate distribution PDF can be represented as follows:

$$\kappa(\mathbf{x}|\boldsymbol{\theta}) = \left[\log P\left(\binom{\mathbf{x}}{1}|\theta_1\right), \log P\left(\binom{\mathbf{x}}{2}|\theta_2\right), \dots, \log P\left(\binom{\mathbf{x}}{d}|\theta_d\right) \right] \in \mathbb{R}^D, \quad (67)$$

where the output vector containing the log-likelihood has a dimension of $D = \sum_{i=1}^d \binom{m}{i}$. In real applications, the hyper-parameter d are usually set with a small number (*e.g.*, $d = 2$) to avoid extremely high-dimensional expansions. Different from the probability density functions used for naive probabilistic expansion, the above function $P(\cdot|\theta_d)$ is a multivariate probability density function with d variables. Notations θ_d denotes the hyper-parameters (*e.g.*, encompassing both the mean vector and covariance matrix if $P(\cdot|\theta_d)$ denotes the PDF of the multivariate normal distribution) of the distribution PDF for combinatorial terms with d elements.

This combinatorial probabilistic functions enable the unification of current deep learning models with classic probabilistic models (*e.g.*, Bayesian networks and Markov networks) into a single framework for data processing and learning. This is because the summation/subtraction of the output terms directly calculates the conditional and joint probabilities of the feature variables. Of course, the parameters of the distributions used above are pre-defined and frozen constants for the current expansion functions. In the future, we will investigate to learn the optimal distribution hyper-parameters instead for better data expansions.

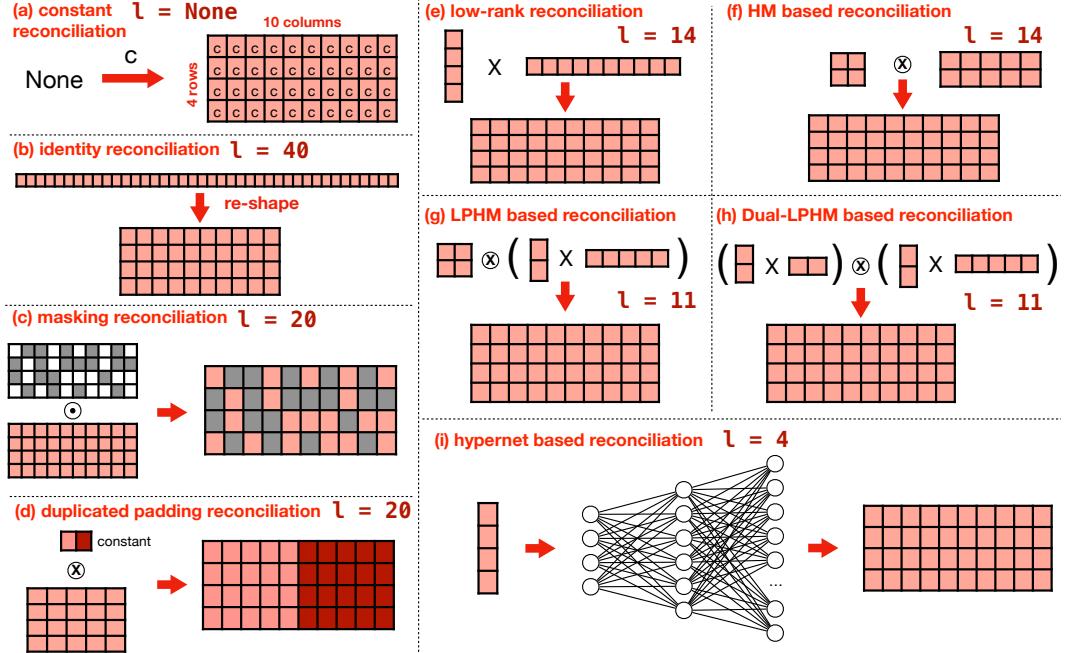


Figure 8: An illustration of different parameter reconciliation functions introduced in this paper. For each of the reconciliation function, we also indicate the number of required parameter length l to generate the desired parameter matrix of size 4×10 in the plots.

5.2 Parameter Reconciliation Functions

To approximate the underlying mapping $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$, the data expansion functions $\kappa : \mathbb{R}^m \rightarrow \mathbb{R}^D$ introduced above projects data instances from input dimension m to an intermediate space of dimension D , where $D > m$. When learning on such expanded data, directly applying existing models and learning approaches with similar parameter scales may suffer from the ‘‘curse of dimensionality’’ and overfitting issues, leading to practical failures. Instead of directly defining a parameter of a scale of D , we propose defining functions $\psi : \mathbb{R}^l \rightarrow \mathbb{R}^{n \times D}$ to fabricate a parameter vector $\mathbf{w} \in \mathbb{R}^l$ of length l to the target dimensions using advanced techniques, where $l \ll n \times D$.

This process is referred to as **parameter reconciliation** in this paper. The inner product of the expanded data vectors and the reconciled parameter matrices will project the data vectors from input dimension m to an intermediate dimension D and then back to the desired output dimension n . The parameter reconciliation function determines both the learning capacity and costs of the RPN model, and we will introduce several practical ways to fabricate the parameters in defining the parameter reconciliation functions in this section. In addition to the summary provided in Figure 6, we also illustrate the parameter reconciliation functions to be introduced here in Figure 8 as well.

5.2.1 Constant Parameter Reconciliation

The simplest parameter reconciliation function will be the constant parameter reconciliation, which projects any input parameters to constants (*e.g.*, zeros or ones) as follows:

$$\psi(\mathbf{w}|c) = c \cdot \mathbf{1}^{n \times D} = \mathbf{C} \in \mathbb{R}^{n \times D}, \quad (68)$$

where the output matrix \mathbf{C} of size $n \times D$ is filled with the provided constant c .

For constant parameter reconciliation, the input parameter \mathbf{w} is not required, which together with its dimension hyper-parameter l can both be set to *none* in implementation. If the output constant $\mathbf{C} = \mathbf{0}$ or $\mathbf{C} = \mathbf{1}$, we can also name the functions as **zero reconciliation** and **one reconciliation**,

respectively. Constant parameter reconciliation functions can accommodate outputs according to requirements. For example, we can set the output to be an identity matrix \mathbf{I} with dimensions $D \times D$, which can be used if and only if the layer input and output dimensions are identical, *i.e.*, $m = n$. We can name such a function as the **constant eye reconciliation** to differentiate it from the identity reconciliation defined below.

Constant reconciliation contributes almost nothing to model learning since it involves no parameters, but it provides our approach with substantial flexibility in representing and designing many models, such as the probabilistic models introduced later in the following Section 6.

5.2.2 Identity Parameter Reconciliation

Another simple parameter reconciliation function is the identity parameter reconciliation, which defines the identity reconciliation function $\psi : \mathbb{R}^l \rightarrow \mathbb{R}^{n \times D}$ as follows:

$$\psi(\mathbf{w}) = \text{reshape}(\mathbf{w}) = \mathbf{W}, \quad (69)$$

where the function will resize the parameters from vector \mathbf{w} of length $l = n \times D$ to the matrix \mathbf{W} of size $n \times D$.

Identity parameter reconciliation is straightforward and may work well for some expansion functions whose output dimension D is not very large. However, when used with expansion functions that produce a large output dimension (such as the high-order Taylor's polynomial expansions), the identity parameter reconciliation function may fail due to the "curse of dimensionality" issues. In such cases, the learning cost would also be extremely high.

5.2.3 Masking based Parameter Reconciliation

To mitigate the identified limitation of identity parameter reconciliation function, one prospective approach entails the initial definition of a parameter matrix denoted as $\mathbf{W} \in \mathbb{R}^{n \times D}$, with a subsequent strategic masking of a substantial proportion of its elements. This strategy is implemented to curtail the count of learnable parameters in \mathbf{W} to a reduced number of l :

$$\psi(\mathbf{w}) = (\mathbf{M} \odot \text{reshape}(\mathbf{w})) = (\mathbf{M} \odot \mathbf{W}) \in \mathbb{R}^{n \times D}, \quad (70)$$

where the term $\mathbf{M} \in \{0, 1\}^{n \times D}$ denotes the binary masking matrix only with l non-zero entries and \odot denotes the element-wise product operator. The notation \mathbf{w} is the vector representation of the parameter matrix \mathbf{W} .

This reconciliation operator is equivalent to: (a) first defining a parameter vector \mathbf{w} of length l , and (b) then scattering these l parameters to a larger matrix of size $n \times D$. Moreover, only these scattered parameters are learnable while all the remaining ones are constant zeros with no gradients. However, current programming toolkits such as PyTorch lack the capability to selectively assign gradients to specific entries within a tensor, rendering the above masking based reconciliation a more pragmatic choice for implementation.

What's more, to facilitate practical adoption, instead of pre-define the parameter dimension l , we advocate for the definition of the masking ratio $p \in [0, 1]$ as a hyper-parameter of the masking based reconciliation function instead. This parameter, in conjunction with the output dimensions $n \times D$, computes the requisite parameter vector dimension, given by $l = p \times n \times D$.

5.2.4 Duplicated Padding based Parameter Reconciliation

In addition to masking, an alternative straightforward approach for fabricating the parameter vector \mathbf{w} from length l to size $n \times D$ involves recursively duplicating \mathbf{w} and sequentially padding them to form the larger parameter matrix. Such a reconciliation function can be efficiently implemented using the matrix Kronecker product operator introduced before in Section 5.1.3.

Specifically, for the parameter vector $\mathbf{w} \in \mathbb{R}^l$ of length l , it can be reshaped into a matrix \mathbf{W} comprising s rows and t columns, where $l = s \times t$. Through the multiplication of \mathbf{W} with a

constant matrix $\mathbf{C} \in \mathbb{R}^{p \times q}$ populated with the constant value of ones, we can define the duplicated padding based parameter reconciliation function as follows:

$$\psi(\mathbf{w}) = \mathbf{C} \otimes \mathbf{W} = \begin{bmatrix} C_{1,1}\mathbf{W} & C_{1,2}\mathbf{W} & \cdots & C_{1,q}\mathbf{W} \\ C_{2,1}\mathbf{W} & C_{2,2}\mathbf{W} & \cdots & C_{2,q}\mathbf{W} \\ \vdots & \vdots & \ddots & \vdots \\ C_{p,1}\mathbf{W} & C_{p,2}\mathbf{W} & \cdots & C_{p,q}\mathbf{W} \end{bmatrix} \in \mathbb{R}^{ps \times qt}, \quad (71)$$

where $\mathbf{W} = \text{reshape}(\mathbf{w})$ and \otimes denotes the Kronecker product operator.

The resulting matrix will encompass $p \times q$ duplicates of the reshaped parameter matrix \mathbf{W} . By adjusting the dimensions of the constant matrix \mathbf{C} - specifically, p and q - we can ensure that $p \times s = n$, $q \times t = D$, and $l = s \times t$, aligning with the desired target parameter dimensions. Regarding the constant matrix \mathbf{C} , beyond being filled with all ones, its elements can also adopt a binary form, comprising zeros and ones. In this scenario, the output will exhibit sparsity, featuring only a few replicated copies of \mathbf{W} . This flexibility in matrix construction augments the versatility of RPN model design and learning process. Notably, the parameter length l is not predetermined but computed during the function definition as $l = st = \frac{n \times D}{pq}$, where p and q are the hyper-parameters of this reconciliation function to be set manually.

5.2.5 Low-Rank Parameter Reconciliation (LoRR)

The practice of low-rank adaption, as investigated in [33], is commonly employed in contemporary parameter-efficient fine-tuning (PEFT) methodologies for language models. Although our paper does not center on PEFT, and our RPN model doesn't incorporate adapters, the principles derived from existing low-rank adaption techniques can be leveraged to define the parameter reconciliation function, effectively addressing our current challenge. Consequently, we define this reconciliation method as LoRR (Low-Rank Reconciliation) in this paper.

Formally, given the parameter vector $\mathbf{w} \in \mathbb{R}^l$ and a rank hyper-parameter r , we partition \mathbf{w} into two sub-vectors and subsequently reshape them into two matrices $\mathbf{A} \in \mathbb{R}^{n \times r}$ and $\mathbf{B} \in \mathbb{R}^{D \times r}$, each possessing a rank of r . These two sub-matrices \mathbf{A} and \mathbf{B} help define the low-rank reconciliation function as follows:

$$\psi(\mathbf{w}) = \mathbf{AB}^\top \in \mathbb{R}^{n \times D}. \quad (72)$$

In implementation, similar to the aforementioned duplicated padding reconciliation, we will solely define r as the hyper-parameter, which in turn determines the desired parameter length l in accordance with the stated constraints. This necessitates imposing certain limitations on these dimension and rank parameters, specifically, $l = (n + D) \times r$.

5.2.6 Hypercomplex Multiplication (HM) based Parameter Reconciliation

The Kronecker product operator described above can also be directly applied to parameter for defining new reconciliation functions, *i.e.*, the hypercomplex parameter reconciliation function:

$$\psi(\mathbf{w}) = \mathbf{A} \otimes \mathbf{B} \in \mathbb{R}^{n \times D}. \quad (73)$$

Similar to the aforementioned low-rank reconciliation, both matrices \mathbf{A} and \mathbf{B} are derived from the parameter vector \mathbf{w} through partitioning and subsequent reshaping. However, instead of computing regular matrix multiplication as in low-rank reconciliation, hypercomplex multiplication-based reconciliation suggests computing the Kronecker product of these two parameter matrices instead.

In implementation, to reduce the number of hyper-parameters and accommodate the parameter dimensions, we can maintain the size of matrix \mathbf{A} as fixed by two hyper-parameters p and q , *i.e.*, $\mathbf{A} \in \mathbb{R}^{p \times q}$. Subsequently, the desired size of matrix \mathbf{B} can be directly calculated as $s \times t$, where $s = \frac{n}{p}$ and $t = \frac{D}{q}$. The hyper-parameters p and q need to be divisors of n and D , respectively. Since both \mathbf{A} and \mathbf{B} originate from \mathbf{w} , the desired parameter length defining \mathbf{w} can be obtained as $l = p \times q + \frac{n}{p} \times \frac{D}{q}$.

5.2.7 Low-Rank Parameterized Hypercomplex Multiplication (LPHM) based Parameter Reconciliation

In the aforementioned hypercomplex multiplication-based parameter reconciliation, ensuring that “parameters p and q divide both n and D ” results in a limited number of choices for p and q , typically leading to a small value (*e.g.*, $p = 4$ and $q = 8$). Consequently, the size of matrix \mathbf{B} and the parameter length l can exceed expectations. To further diminish the parameter count, inspired by [14], we can additionally transform matrix B into its low-rank representations during the reconciliation definition. Specifically,

$$\psi(\mathbf{w}) = \mathbf{A} \otimes \mathbf{B} = \mathbf{A} \otimes (\mathbf{S}\mathbf{T}^\top) \in \mathbb{R}^{n \times D}, \quad (74)$$

where $\mathbf{S} \in \mathbb{R}^{\frac{n}{p} \times r}$ and $\mathbf{T} \in \mathbb{R}^{\frac{D}{q} \times r}$ represent the low-rank matrices for composing \mathbf{B} . This parameter fabrication approach is coined as the Low-Rank Parameterized Hypercomplex Multiplication (LPHM) based parameter reconciliation, enabling further reduction of the required parameter vector length to $l = p \times q + r(\frac{n}{p} + \frac{D}{q})$.

5.2.8 Dual Low-Rank Parameterized Hypercomplex Multiplication (Dual LPHM) based Parameter Reconciliation

To provide RPN with more flexibility in defining the parameter reconciliation function, based on the above LPHM function, we further introduce the dual LPHM parameter reconciliation function allowing low-rank representations of both sub-matrices \mathbf{A} and \mathbf{B} , *i.e.*,

$$\psi(\mathbf{w}) = \mathbf{A} \otimes \mathbf{B} = (\mathbf{P}\mathbf{Q}^\top) \otimes (\mathbf{S}\mathbf{T}^\top) \in \mathbb{R}^{n \times D}, \quad (75)$$

where $\mathbf{P} \in \mathbb{R}^{p \times r}$ and $\mathbf{Q} \in \mathbb{R}^{q \times r}$ represent the low-rank matrices for composing \mathbf{A} , and $\mathbf{S} \in \mathbb{R}^{\frac{n}{p} \times r}$ and $\mathbf{T} \in \mathbb{R}^{\frac{D}{q} \times r}$ represent the low-rank matrices for composing \mathbf{B} . This parameter fabrication approach is named as the Dual Low-Rank Parameterized Hypercomplex Multiplication (Dual LPHM) based parameter reconciliation, which reduces the required parameter vector length to $l = r(p + q + \frac{n}{p} + \frac{D}{q})$.

5.2.9 HyperNets based Parameter Reconciliation

Besides these above matrix fabrication based reconciliations defined above, another viable approach for parameter reconciliation is through the utilization of hypernets [24, 52]. These works employ a hypernet model, such as a Multi-Layer Perceptron (MLP), to project the input parameter vector $\mathbf{w} \in \mathbb{R}^l$ from length l to a significantly higher-dimensional output:

$$\psi(\mathbf{w}) = \text{HyperNet}(\mathbf{w}) = \mathbf{W} \in \mathbb{R}^{n \times D}. \quad (76)$$

To circumvent the introduction of additional parameters and learning costs, we can initialize a hypernet model randomly and then freeze its parameters. Subsequently, we utilize this frozen hypernet model to define the aforementioned parameter reconciliation function. In this approach, the parameter l is not calculated automatically and needs to be manually set up as a hyper-parameter.

Compared to the aforementioned LoRR and Kronecker product-based reconciliation functions, hypernets offer greater flexibility in function definition but also entail higher computational costs, since the desired output parameter length $n \times D$ can be extremely high. We will delve into their performance through extensive experimental studies in the following sections.

5.3 Remainder Functions

The remainder function $\pi : \mathbb{R}^m \rightarrow \mathbb{R}^n$ plays a crucial role in ensuring the representation completeness of RPN. This function provides complementary information that may not be encompassed by the expansion and reconciliation functions alone. In this subsection, we will introduce several different remainder functions that can be employed to construct the RPN model. These remainder functions have also been summarized in Figure 6 as well.

5.3.1 Constant Remainder

The constant remainder function $\pi : \mathbb{R}^m \rightarrow \mathbb{R}^n$ just projects all inputs to a constant vector, *i.e.*,

$$\pi(\mathbf{x}) = \mathbf{c} \in \mathbb{R}^n, \quad (77)$$

where \mathbf{c} is a constant vector.

Specifically, when the output \mathbf{c} is $\mathbf{0}$, it can be referred to as the **zero remainder**. This function represents the simplest form of remainder, assuming that the data expansion function and parameter reconciling function already perform well in capture all necessary information about the underlying function already (a claim supported by forthcoming experimental results). Additionally, all constant remainder functions require no additional parameters.

5.3.2 Identity Remainder

Similar to residual learning techniques employed in contemporary deep learning models [26], we can define the remainder function $\pi : \mathbb{R}^m \rightarrow \mathbb{R}^n$ as an identity function based on the input. For instance, when $m = n$, we define the identity remainder function as follows:

$$\pi(\mathbf{x}) = \mathbf{x} \in \mathbb{R}^n, \text{ or } \pi(\mathbf{x}) = \sigma(\mathbf{x}) \in \mathbb{R}^n, \quad (78)$$

where notation σ denotes the (optional) activation function, which can be sigmoid, ReLU and the recent SiLU [19].

5.3.3 Linear Remainder

Meanwhile, when the dimensions of the input and output spaces differ, *i.e.*, $m \neq n$, we must introduce additional parameters into the remainder function to adjust for this mismatch. Here, we introduce the linear remainder function as follows:

$$\pi(\mathbf{x}) = \mathbf{x}\mathbf{W}' \in \mathbb{R}^n, \text{ or } \pi(\mathbf{x}) = \sigma(\mathbf{x}\mathbf{W}') \in \mathbb{R}^n. \quad (79)$$

Similarly, the notation σ denotes the optional activation function. Concurrently, the learnable parameter matrix $\mathbf{W}' \in \mathbb{R}^{m \times n}$ is used here for vector dimension adjustment. For the parameter \mathbf{W}' , we add the prime symbol to differentiate it from the parameter used in the reconciliation function.

5.3.4 Complementary Expansion based Remainder

For the majority of scenarios, the aforementioned remainder functions are sufficient for constructing RPN models that generally fulfill our requirements. However, incorporating learnable parameters into the remainder function enhances the flexibility and capacity of RPN models. In addition to the aforementioned simple forms of remainder functions, this paper also allows for the definition of the remainder function in an augmented manner, as a RPN head coupled with a zero remainder.

Formally, the complementary expansion based remainder function can be defined as follows:

$$\pi(\mathbf{x}|\mathbf{w}') = \langle \kappa'(\mathbf{x}), \psi'(\mathbf{w}') \rangle + \underbrace{\pi'(\mathbf{x})}_{\pi'(\mathbf{x})=\mathbf{0} \text{ by default}}. \quad (80)$$

To distinguish the notations used in defining the original RPN model, we will add the “prime” symbol to functions and parameters, indicating that they are defined within the complementary expansion. All the previously mentioned data expansion functions, parameter reconciliation functions, and remainder functions can be utilized to define the functions κ' , ψ' , and π' .

Since RPN itself permits multi-head and multi-channel model architecture design within each layer, the performance of the complementary expansion-based remainder function can be equivalently represented through a multi-head RPN layer assisted by the zero remainder. Meanwhile, this type

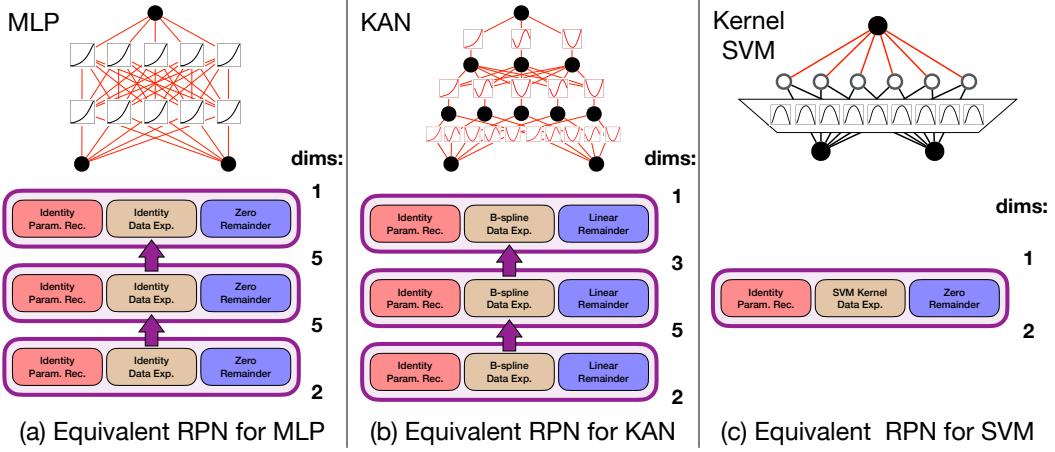


Figure 9: An illustration of representing MLP, KAN and kernel SVM with RPN.

of remainder function offers greater flexibility in model design, particularly for learning scenarios necessitating more potent remainders. At the same time, when employing the complementary expansion remainder function, it's customary to set the remainder function π' as the zero remainder by default. This ensures that it doesn't introduce unnecessary redundancy in model design by serving as another complementary expansion-based remainder.

6 Unifying Existing Base Models with RPN Canonical Representation

Building with the component functions outlined in the previous section, the RPN model has versatile model architecture and attains superior modeling capability. Through strategic combinations of these component functions, we can establish a multi-head, multi-channel, and multi-layer framework, providing a unified basis for representing several influential base models such as Bayesian network, Markov network, kernel SVM, MLP, and KAN.

In the previous Section 2, we have already provided the brief comparisons of RPN with these base models in terms of mathematical theorem foundation, formula and model architecture. This section further illustrates how, by selecting specific component functions, each of these models can be consolidated into RPN canonical representation, characterized by the inner product of a data expansion function with a parameter reconciliation function, complemented by a remainder function. The following Figures 9 and 10 also demonstrate the unified representations of these base models with RPN model.

6.1 Unifying MLP with RPN

In this subsection, we will introduce the Multi-Layer Perceptron (MLP) model designed based on the Universal Approximation Theorem, and discuss how to represent MLP into the unified representation with the RPN model.

6.1.1 Universal Approximation Theorem

Before talking about the MLP model and representing MLP with RPN, we will first introduce the Universal Approximation Theorem as follows.

THEOREM 2 (Universal Approximation Theorem): *Given a continuous multivariate $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$, we can approximate f with a series summation of function $\sigma : \mathbb{R} \rightarrow \mathbb{R}$. Function σ is not polynomial if and only if for every $m, n \in \mathbb{N}$ and $\epsilon > 0$, there exists $k \in \mathbb{N}$, $\mathbf{A} \in \mathbb{R}^{k \times m}$, $\mathbf{b} \in \mathbb{R}^k$ and $\mathbf{C} \in \mathbb{R}^{n \times k}$ defining function*

$$g(\mathbf{x}|\mathbf{w}) = \mathbf{C} \cdot \sigma(\mathbf{Ax} + \mathbf{b}), \quad (81)$$

that can approximate function f with an error no greater than ϵ , i.e.,

$$\sup_{\mathbf{x} \in \mathbb{R}^m} \|f(\mathbf{x}) - g(\mathbf{x}|\mathbf{w})\| < \epsilon. \quad (82)$$

The parameter vector \mathbf{w} covers all the aforementioned coefficient matrices \mathbf{A} , \mathbf{C} and bias vector \mathbf{b} .

6.1.2 Multi-Layer Perceptron (MLP)

Built upon the Universal Approximation Theorem, MLP proposes to approximate the function $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$ by stacking neuron layers on top of each other, where each neuron sums up the accumulated inputs and generates the output through an activation function $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ as follows:

$$g(\mathbf{x}|\mathbf{w}) = \mathbf{W}_1 \sigma(\mathbf{W}_2 \mathbf{x} + \mathbf{b}), \quad (83)$$

where $\mathbf{w} = (\mathbf{W}_1, \mathbf{W}_2, \mathbf{b})$ covers the matrices $\mathbf{W}_1 \in \mathbb{R}^{n \times k}$ and $\mathbf{W}_2 \in \mathbb{R}^{k \times m}$ as the weights and $\mathbf{b} \in \mathbb{R}^k$ as the bias. As to the activation function σ , many different types of activation functions have been proposed already, ranging from the classic binary-step and sigmoid function to the recent SiLU and dSiLU [19].

6.1.3 Representing MLP with RPN

The MLP model can be easily represented with RPN involving the *identity data expansion function*, *identity parameter reconciliation function* and *zero remainder function* introduced in the previous Section 5.

Specifically, as depicted in Plot (a) of Figure 9, we illustrate the representation of a three-layer MLP model at the top, and its corresponding representation with RPN involving three RPN-layers at the bottom, with the input and output dimensions indicated on the right-hand side.

- **RPN Layer 1:** A single-head, single-channel RPN layer consisting of (1) an identity data expansion function (with sigmoid as the optional output-processing function), (2) an identity parameter reconciliation function, and (3) a zero remainder function;
- **RPN Layer 2:** A single-head, single-channel RPN layer consisting of (1) an identity data expansion function (with sigmoid as the optional output-processing function), (2) an identity parameter reconciliation function, and (3) a zero remainder function;
- **RPN Layer 3:** A single-head, single-channel RPN layer consisting of (1) an identity data expansion function, (2) an identity parameter reconciliation function, and (3) a zero remainder function.

6.2 Unifying KAN with RPN

The Kolmogorov-Arnold Network (KAN) [51] is a new base model introduced recently in 2024, designed based on the Kolmogorov-Arnold RepresentationTheorem. Diverging from MLP's fixed activation functions, KAN suggests learning the activation functions for pairwise neuron connections using B-spline interpolation. Here, we will briefly introduce the Kolmogorov-Arnold Representation Theorem and subsequently discuss how to represent the KAN model architecture with RPN.

6.2.1 Kolmogorov-Arnold Representation Theorem

Kolmogorov-Arnold Representation Theorem posits that any multivariate continuous function can be expressed as a composition of the two-argument addition of continuous univariate functions.

THEOREM 3 (*Kolmogorov-Arnold Representation Theorem*): *Formally, given a continuous multivariate function on a bounded domain, e.g., $f : [0, 1]^m \rightarrow \mathbb{R}$, the function f can be written as a*

finite composition of continuous univariate functions and the binary operation of addition:

$$f(\mathbf{x}) = f([x_1, x_2, \dots, x_m]^\top) = \sum_{q=0}^{2m} \phi_q \left(\sum_{p=1}^m \phi_{q,p}(x_p) \right), \quad (84)$$

where $\phi_{q,p} : [0, 1] \rightarrow \mathbb{R}$ and $\phi_q : \mathbb{R} \rightarrow \mathbb{R}$.

6.2.2 Kolmogorov-Arnold Network (KAN)

Based on the Kolmogorov-Arnold Representation Theorem mentioned above, a recent paper [51] introduces the KAN model. The theorem imposes a specific constraint on the required function numbers, namely $2m+1$ and m . However, in the practical implementation of KAN, these constraints are relaxed, allowing the KAN model to employ a deep architecture by stacking multiple layers on top of each other to approximate the function $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$, as indicated below:

$$g(\mathbf{x}|\mathbf{w}) = \text{KAN}(\mathbf{x}) = \Phi^K \circ \Phi^{K-1} \circ \dots \circ \Phi^1(\mathbf{x}). \quad (85)$$

Here, the notation $\Phi^k = \{\phi_{i,j}^k\}_{i \in \{1, 2, \dots, d_{in}\}, j \in \{1, 2, \dots, d_{out}\}}$ represents a matrix of function $\phi_{i,j}^k$ with learnable parameters at the k_{th} layer of the model, and d_{in} , d_{out} are the corresponding input and output dimensions. Operator \circ denotes the function composition of sequential layers in KAN.

In implementation, the function ϕ with learnable parameters is formally defined as

$$\phi(x) = b(x) + \text{spline}(x), \text{ where } \begin{cases} b(x) = \text{SiLU}(x) = \frac{x}{1+\exp^{-x}}, \\ \text{spline}(x) = \sum_i w_i B_{i,d}(x). \end{cases} \quad (86)$$

In the above equation, the “spline(\cdot)” function is defined as a linear combination of B-splines of degree d , denoted by $\{B_{i,d}(x)\}_i$, where $\{w_i\}_i$ represents the set of learnable parameters. Meanwhile, the base function $b(\cdot)$ is defined as the SiLU function based on the inputs.

6.2.3 Representing KAN with RPN

Similar as MLP, the KAN model can also be easily represented with RPN involving the *B-spline data expansion function*, *identity parameter reconciliation function* and *linear remainder function* introduced in the previous Section 5.

Specifically, as depicted in Plot (b) of Figure 9, for the three-layer KAN model illustrated at the top, its corresponding representation with RPN involving three RPN-layers at the bottom:

- **RPN Layer 1:** A single-head, single-channel layer consisting of (1) a B-spline data expansion function, (2) an identity parameter reconciliation function, and (3) a linear remainder function (with SiLU as the activation function);
- **RPN Layer 2:** A single-head, single-channel layer consisting of (1) a B-spline data expansion function, (2) an identity parameter reconciliation function, and (3) a linear remainder function (with SiLU as the activation function);
- **RPN Layer 3:** A single-head, single-channel layer consisting of (1) a B-spline data expansion function, (2) an identity parameter reconciliation function, and (3) a linear remainder function (with SiLU as the activation function).

6.3 Unifying Kernel SVM with RPN

Support vector machine (SVM) is a renowned supervised machine learning model introduced for data classification and regression analysis. While SVM is adept at linearly separating instances, it can also perform non-linear classifications through the utilization of kernel tricks.

6.3.1 Support Vector Machine (SVM)

Support Vector Machine (SVM) is a non-probabilistic binary linear classifier model proposed based on statistical machine learning. Various extensions enable SVM to operate in multi-class and multi-label classification scenarios.

Below, we will introduce SVM within the classic binary classification learning settings and employ it to approximate the underlying mapping, represented as $f : \mathbb{R}^m \rightarrow \{-1, +1\}$ as follows:

$$g(\mathbf{x}|\mathbf{w}, b) = \text{sign}(\mathbf{w}^\top \mathbf{x} + b), \quad (87)$$

where $\text{sign}(\cdot)$ returns the polarity of the input term $\mathbf{w}^\top \mathbf{x} + b$, while \mathbf{w} , b denote the learnable parameters.

6.3.2 Kernel Tricks

The SVM model described above is highly effective for linear classification tasks. However, for handling nonlinear tasks, techniques such as the kernel trick have been introduced. The kernel trick is widely used in regression, classification, and PCA. It facilitates the embedding of the problem into higher-dimensional spaces, often even infinite-dimensional ones, without the need for an infinite amount of computational effort.

Formally, given a vector $\mathbf{x} \in \mathbb{R}^m$, the kernel trick introduces a feature mapping $\phi : \mathbb{R}^m \rightarrow \mathbb{R}^M$ to project the vector into a higher-dimensional space, denoted as $\phi(\mathbf{x}) \in \mathbb{R}^M$, where $M > m$. An illustrative example of such a mapping function used in kernel tricks is presented below.

EXAMPLE 1 For instance, given a vector $\mathbf{x} = [x_1, x_2]^\top \in \mathbb{R}^2$, a very simple mapping shown below will project \mathbf{x} from \mathbb{R}^2 to a high-order dimension \mathbb{R}^3 :

$$\phi([x_1, x_2]^\top) = [x_1^2, \sqrt{2}x_1x_2, x_2^2]^\top. \quad (88)$$

For data instances that are not linearly separable in the original space, projecting them using the mapping described above renders them linearly separable by the model represented in Equation (87) within the higher-dimensional space. The process of learning kernel SVM to obtain the parameters is beyond the scope of this paper, and will not be discussed here.

6.3.3 Representing Kernel SVM with RPN

Unlike MLP and KAN, which employ deep architectures, the SVM model is typically proposed with a shallow architecture consisting of a single layer. Specifically, for the kernel SVM illustrated in Plot (c) of Figure 9, it can be represented within RPN with a single layer:

- **RPN Layer:** A single-head, single-channel RPN layer consisting of (1) a data expansion function corresponding to the kernel function (*e.g.*, linear or RBF), (2) an identity parameter reconciliation function, and (3) a zero remainder function.

6.4 Unifying PMs with RPN

Probabilistic model denotes a broad family of statistical machine learning models build on probability theory. In this subsection, we will provide a brief introduction of the probabilistic models, followed by a discussion on representing these models using the RPN model.

6.4.1 Probabilistic Models (PMs)

Probabilistic models (PMs) assume the relationships among the variables can be effectively modeled with probability distributions. In PMs, we use the upper case notations, such as X_i and Y_j , to represent variables and lower case ones, such as x_i and y_j , to represent their respective values.

Formally, to infer the underlying mapping $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$, we represent the inputs and outputs as random variables X_1, X_2, \dots, X_m and Y_1, Y_2, \dots, Y_n , respectively. The potential value spaces of the input and output are denoted as $\mathcal{X} \subset \mathbb{R}^m$ and $\mathcal{Y} \subset \mathbb{R}^n$. Given an input instance $[x_1, x_2, \dots, x_m]^\top \in \mathcal{X}$, the underlying model f will project it to the outputs $[y_1, \dots, y_n]^\top \in \mathcal{Y}$ that maximize the following probability:

$$\max P(Y_1 = y_1, \dots, Y_n = y_n | X_1 = x_1, \dots, X_m = x_m), \quad (89)$$

where $P(\cdot|\cdot)$ denotes the conditional probability defined based on certain distributions.

We can calculate the above conditional probability by dividing the joint probabilities of the random variables. Furthermore, by applying the logarithm operator to the probabilities, we can rewrite the above conditional probability calculation as follows (simplifying the notations to include only two variables, X and Y):

$$\begin{aligned} \log P(Y = y | X = x) &= \log \left(\frac{P(Y = y \wedge X = x)}{P(X = x)} \right) \\ &= \log P(Y = y \wedge X = x) - \log P(X = x). \end{aligned} \quad (90)$$

The fundamental problem studied in PMs is how to calculate the joint probabilities $P(Y = y \wedge X = x)$ and $P(X = x)$, which involve multiple random variables. In PMs, we treat input and output random variables (*i.e.*, Y and X) equally. To simplify notations, we will just illustrate how to calculate joint probabilities with m random variables X_1, X_2, \dots, X_m below, noting that joint probabilities involving random variables from both X s and Y s can be calculated similarly.

6.4.2 Naive Bayes and Probabilistic Graphical Models (PGMs)

Numerous machine learning models fall under the category of PMs. Below, we will list three representative PMs: naive Bayes, Bayesian network, and Markov network, with Bayesian network and Markov network also commonly referred to as probabilistic graphical models. We will also demonstrate how to represent them using the RPN model.

Naive Bayes: The naive Bayes classifier operates under the assumption that, given the target class, the features of input data instances are conditionally independent. This assumption allows us to reformulate the above Equation (89) and Equation (90) as follows:

$$P(Y = y | X_1 = x_1, \dots, X_m = x_m) \propto P(Y = y) \prod_{i=1}^m P(X_i = x_i | Y = y), \quad (91)$$

and

$$\log P(Y = y | X_1 = x_1, \dots, X_m = x_m) \propto \log P(Y = y) + \sum_{i=1}^m \log P(X_i = x_i | Y = y). \quad (92)$$

The stringent independence assumption restricts the applicability of naive Bayes to a narrow range of scenarios. In contrast, both Bayesian network and Markov network, which will be introduced below shortly, aim to capture the dependency relationships among the feature variables instead.

Bayesian network: Bayesian network assumes that the relationships between variables can be represented as a directed acyclic graph, wherein each node (*i.e.*, the variables) possesses directed edges pointing to its children, indicating a direct influence or causal relationship between them. These networks encode conditional dependencies using directed edges, while the joint probability distribution is factored as a product of conditional probabilities.

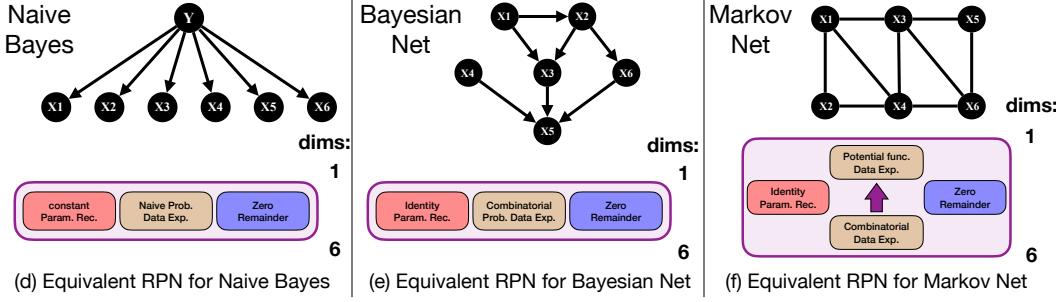


Figure 10: An illustration of representing different probabilistic models with RPN.

Formally, based on Bayesian network, the joint log-probability distribution $P(X_1, X_2, \dots, X_m)$ involving these m random variables can be written as:

$$\begin{aligned} \log P(X_1, X_2, \dots, X_m) &= \log \left(\prod_{i=1}^m P(X_i | \Gamma(X_i)) \right), \\ &= \sum_{i=1}^m \log P(X_i, \Gamma(X_i)) - \log P(\Gamma(X_i)), \end{aligned} \quad (93)$$

where $\Gamma(X_i)$ denotes the set of parent nodes of X_i in the probabilistic dependency DAG.

Markov network: Markov network, on the other hand, assumes that the relationships between variables can be represented as an undirected graph, where edges indicate a mutual dependency between nodes (*i.e.*, the variables), without implying any directionality or causation.

Formally, given the random variables X_1, X_2, \dots, X_m , Markov network defines their relations as an undirected graph G , which can be divided into cliques (fully connected subsets of nodes) $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_k\}$ and $\bigcup_{i=1}^k \mathcal{C}_i = \{X_1, X_2, \dots, X_m\}$. Based on such cliques, the joint log-probability defined on all the variables can be factored as follows:

$$\begin{aligned} \log P(X_1, X_2, \dots, X_m) &= \log \left(\frac{1}{Z} \prod_{i=1}^k \phi(X_{\mathcal{C}_i}) \right), \\ &= \sum_{i=1}^k \log \phi(X_{\mathcal{C}_i}) - \log Z, \end{aligned} \quad (94)$$

where $\log Z$ can be viewed as a constant. Regarding the *factor potential functions* in $\{\phi(X_{\mathcal{C}_i})\}_{i \in \mathcal{C}}$, several different methods exist for defining them, such as the *appearance count in the training set*, *exponential potential function* and *Gaussian potential function*.

6.4.3 Representing PMs with RPN

Similar to SVM, representing naive Bayes, Bayesian network and Markov network with RPN will also involve one single layer. However, because these probabilistic models are designed based on different assumptions, the component functions involved will also be distinct. As illustrated in Figure 10, these probabilistic models can be depicted using RPN as follows:

- **Naive Bayes:** A single-head, single-channel RPN layer consisting of (1) a naive probabilistic data expansion function, (2) a constant parameter reconciliation function (containing constant ones), and (3) a zero remainder function.
- **Bayesian Network:** A single-head, single-channel RPN layer consisting of (1) a combinatorial probabilistic data expansion function, (2) an identity parameter reconciliation function, and (3) a zero remainder function.

- **Markov Network:** A single-head, single-channel RPN layer consisting of (1) a nested data expansion function composed of the combinatorial expansion followed by the “factor potential function” based expansion function, (2) an identity parameter reconciliation function, and (3) a zero remainder function.

7 Empirical Evaluations of RPN

This section presents empirical evaluations of RPN across various deep function learning tasks with extensive experiments on real-world benchmark datasets. We examine several key performance aspects of RPN and organize our insightful findings as follows. In Section 7.1, we provide experimental investigations of RPN for continuous function learning on three datasets: elementary functions, composite functions, and Feynman functions. Section 7.2 evaluates RPN for discrete vision and language data classification, using image datasets (MNIST and CIFAR-10) and text datasets (IMDB, AGNews, and SST2). To assess RPN for probabilistic dependency relationship inference, Section 7.3 presents experiments on three classic tabular datasets: Iris Species, Pima Indians Diabetes, and Banknote. Throughout these subsections, we also provide experimental analysis of RPN in terms of convergence, parameter sensitivity, ablation studies, interpretation, and visualization for the specific deep function learning tasks.

7.1 Continuous Function Approximation

As previously described, RPN can serve as a base model for effective continuous function approximation. In this section, we investigate the empirical effectiveness of RPN using three continuous function datasets. We begin by introducing the datasets and experimental setups, followed by a detailed presentation of the experimental results and performance analysis.

7.1.1 Dataset Descriptions and Experiment Setups

Table 1: Statistics of continuous function datasets used in the experiments. For the Feynman function dataset, the input variable numbers can be different for different functions, which are not provided in the table. For each function in the dataset, we randomly generate 2,000 input-output pairs according to the input variable value ranges, which are partitioned into the training and testing sets according to the 50 : 50 ratio.

	Continuous Function Datasets		
	Elementary Functions	Composite Functions	Feynman Functions
Equ. #	17	17	100
Train #	1,000	1,000	1,000
Test #	1,000	1,000	1,000
Input Dim.	2	2	–
Output Dim.	1	1	1

Dataset Descriptions: Three continuous function datasets are used in our experiments to evaluate the performance of RPN against comparison methods MLP and KAN. The datasets are described below, with basic statistical information provided in Table 1.

- **Elementary Function Dataset:** We compose an elementary function dataset in this paper. The elementary function dataset comprises 17 elementary functions, each representing the simplest form of a multivariate function defined by two variables, x and y , with specific value ranges. These function ids, formulas and their corresponding input value ranges are provided in the first three columns of Table 2.
- **Composite Function Dataset:** Building upon these elementary functions, we created the composite function dataset by combining them through addition, multiplication, and nesting to form more complex functions. The 17 created composite functions and their input variable value ranges are presented in the first three columns of Table 3.

Table 2: Experimental results of continuous function approximation on the elementary function dataset. All these models are trained with 2,000 epochs, which guarantee their convergence, and the best testing scores achieved by all these methods within these 2,000 epochs are cherry-picked, aiming to eliminate the impacts of epoch selection on the result evaluation. All these models are all trained with 5 different random seeds, and the average scores together with the standard deviations are reported in the table. For the method names, RPN-Ext denotes RPN with extended expansion function (involving Taylor’s expansion and Bspline expansion), low-rank reconciliation function and Zero remainder function; while RPN-Nstd denotes RPN with nested expansion function (involving Taylor’s expansion and Bspline expansion), low-rank reconciliation function and Zero remainder function.

Eq.	Formula	Variables	MLP Architecture	MLP MSE	KAN Architecture	KAN MSE	RPN-Ext Architecture	RPN-Ext MSE	RPN-Nstd Architecture	RPN-Nstd MSE
E.0	$x + y$	$x, y \in (0, 1)$	[2, 5, 5, 1] param #: 51	6.25×10^{-7} $\pm 8.08 \times 10^{-7}$	[2, 2, 1, 1] param #: 63	4.23×10^{-7} $\pm 5.78 \times 10^{-7}$	[2, 2, 1, 1] param #: 47	8.40×10^{-8} $\pm 1.12 \times 10^{-7}$	[2, 2, 1, 1] param #: 547	1.93×10^{-8} $\pm 1.15 \times 10^{-8}$
E.1	$\frac{1}{(x+y)}$	$x, y \in (0, 1)$	[2, 5, 5, 1] param #: 51	7.64×10^{-1} $\pm 6.05 \times 10^{-1}$	[2, 2, 1, 1] param #: 63	3.32×10^{-2} $\pm 5.47 \times 10^{-2}$	[2, 2, 1, 1] param #: 47	8.67×10^{-2} $\pm 8.22 \times 10^{-2}$	[2, 2, 1, 1] param #: 547	1.03×10^{-1} $\pm 1.40 \times 10^{-1}$
E.2	$(x + y)^2$	$x, y \in (0, 1)$	[2, 5, 5, 1] param #: 51	1.89×10^{-3} $\pm 1.81 \times 10^{-3}$	[2, 2, 1, 1] param #: 63	1.32×10^{-6} $\pm 5.29 \times 10^{-7}$	[2, 2, 1, 1] param #: 47	2.56×10^{-7} $\pm 1.48 \times 10^{-7}$	[2, 2, 1, 1] param #: 547	5.06×10^{-8} $\pm 3.53 \times 10^{-8}$
E.3	$\exp(x + y)$	$x, y \in (0, 1)$	[2, 5, 5, 1] param #: 51	9.33×10^{-3} $\pm 1.14 \times 10^{-2}$	[2, 2, 1, 1] param #: 63	1.22×10^{-5} $\pm 1.10 \times 10^{-5}$	[2, 2, 1, 1] param #: 47	3.81×10^{-6} $\pm 6.56 \times 10^{-6}$	[2, 2, 1, 1] param #: 547	1.26×10^{-6} $\pm 1.37 \times 10^{-6}$
E.4	$\ln(x + y)$	$x, y \in (0, 1)$	[2, 5, 5, 1] param #: 51	1.26×10^{-3} $\pm 8.24 \times 10^{-4}$	[2, 2, 1, 1] param #: 63	3.95×10^{-5} $\pm 3.86 \times 10^{-5}$	[2, 2, 1, 1] param #: 47	7.05×10^{-5} $\pm 3.12 \times 10^{-5}$	[2, 2, 1, 1] param #: 547	1.80×10^{-5} $\pm 2.37 \times 10^{-5}$
E.5	$\sin(x + y)$	$x, y \in (0, 1)$	[2, 5, 5, 1] param #: 51	1.49×10^{-3} $\pm 2.50 \times 10^{-3}$	[2, 2, 1, 1] param #: 63	2.14×10^{-8} $\pm 9.06 \times 10^{-9}$	[2, 2, 1, 1] param #: 47	4.95×10^{-8} $\pm 4.39 \times 10^{-8}$	[2, 2, 1, 1] param #: 547	5.67×10^{-9} $\pm 3.20 \times 10^{-9}$
E.6	$\cos(x + y)$	$x, y \in (0, 1)$	[2, 5, 5, 1] param #: 51	2.12×10^{-2} $\pm 4.23 \times 10^{-2}$	[2, 2, 1, 1] param #: 63	2.20×10^{-7} $\pm 2.47 \times 10^{-7}$	[2, 2, 1, 1] param #: 47	1.25×10^{-7} $\pm 1.34 \times 10^{-7}$	[2, 2, 1, 1] param #: 547	5.93×10^{-9} $\pm 3.98 \times 10^{-9}$
E.7	$\tan(x + y)$	$x, y \in (0, 1)$	[2, 5, 5, 1] param #: 51	2.87×10^{-4} $\pm 2.97 \times 10^{-4}$	[2, 2, 1, 1] param #: 63	3.26×10^{-7} $\pm 3.27 \times 10^{-7}$	[2, 2, 1, 1] param #: 47	6.02×10^{-8} $\pm 4.46 \times 10^{-8}$	[2, 2, 1, 1] param #: 547	1.67×10^{-8} $\pm 2.32 \times 10^{-8}$
E.8	$\arcsin(x + y)$	$x, y \in (0, 0.5)$	[2, 5, 5, 1] param #: 51	3.84×10^{-4} $\pm 2.92 \times 10^{-4}$	[2, 2, 1, 1] param #: 63	2.61×10^{-7} $\pm 1.08 \times 10^{-7}$	[2, 2, 1, 1] param #: 47	1.65×10^{-6} $\pm 2.06 \times 10^{-6}$	[2, 2, 1, 1] param #: 547	5.16×10^{-7} $\pm 2.97 \times 10^{-7}$
E.9	$\arccos(x + y)$	$x, y \in (0, 0.5)$	[2, 5, 5, 1] param #: 51	1.26×10^{-2} $\pm 2.51 \times 10^{-2}$	[2, 2, 1, 1] param #: 63	2.35×10^{-6} $\pm 3.50 \times 10^{-6}$	[2, 2, 1, 1] param #: 47	4.49×10^{-5} $\pm 5.15 \times 10^{-5}$	[2, 2, 1, 1] param #: 547	3.73×10^{-7} $\pm 2.32 \times 10^{-7}$
E.10	$\arctan(x + y)$	$x, y \in (0, 0.5)$	[2, 5, 5, 1] param #: 51	3.09×10^{-5} $\pm 4.88 \times 10^{-5}$	[2, 2, 1, 1] param #: 63	6.09×10^{-8} $\pm 7.80 \times 10^{-8}$	[2, 2, 1, 1] param #: 47	4.86×10^{-9} $\pm 2.17 \times 10^{-9}$	[2, 2, 1, 1] param #: 547	3.02×10^{-9} $\pm 1.65 \times 10^{-9}$
E.11	$\sinh(x + y)$	$x, y \in (0, 1)$	[2, 5, 5, 1] param #: 51	2.42×10^{-3} $\pm 3.15 \times 10^{-3}$	[2, 2, 1, 1] param #: 63	8.96×10^{-7} $\pm 7.13 \times 10^{-7}$	[2, 2, 1, 1] param #: 47	1.99×10^{-7} $\pm 1.65 \times 10^{-7}$	[2, 2, 1, 1] param #: 547	5.43×10^{-8} $\pm 5.25 \times 10^{-8}$
E.12	$\cosh(x + y)$	$x, y \in (0, 1)$	[2, 5, 5, 1] param #: 51	1.62×10^{-3} $\pm 8.37 \times 10^{-4}$	[2, 2, 1, 1] param #: 63	8.77×10^{-7} $\pm 1.39 \times 10^{-7}$	[2, 2, 1, 1] param #: 47	7.00×10^{-7} $\pm 4.65 \times 10^{-7}$	[2, 2, 1, 1] param #: 547	4.47×10^{-8} $\pm 3.42 \times 10^{-8}$
E.13	$\tanh(x + y)$	$x, y \in (0, 1)$	[2, 5, 5, 1] param #: 51	8.35×10^{-4} $\pm 1.39 \times 10^{-3}$	[2, 2, 1, 1] param #: 63	3.10×10^{-8} $\pm 3.24 \times 10^{-8}$	[2, 2, 1, 1] param #: 47	5.73×10^{-8} $\pm 6.20 \times 10^{-8}$	[2, 2, 1, 1] param #: 547	4.55×10^{-9} $\pm 2.73 \times 10^{-9}$
E.14	$\text{arcsinh}(x + y)$	$x, y \in (0, 0.5)$	[2, 5, 5, 1] param #: 51	1.47×10^{-5} $\pm 2.21 \times 10^{-5}$	[2, 2, 1, 1] param #: 63	1.98×10^{-7} $\pm 2.27 \times 10^{-7}$	[2, 2, 1, 1] param #: 47	5.13×10^{-9} $\pm 3.94 \times 10^{-9}$	[2, 2, 1, 1] param #: 547	4.34×10^{-9} $\pm 2.82 \times 10^{-9}$
E.15	$\text{arccosh}(x + y)$	$x, y \in (0.5, 1)$	[2, 5, 5, 1] param #: 51	8.28×10^{-3} $\pm 1.57 \times 10^{-2}$	[2, 2, 1, 1] param #: 63	2.32×10^{-7} $\pm 7.08 \times 10^{-8}$	[2, 2, 1, 1] param #: 47	8.94×10^{-6} $\pm 4.55 \times 10^{-6}$	[2, 2, 1, 1] param #: 547	5.38×10^{-7} $\pm 5.62 \times 10^{-7}$
E.16	$\text{arctanh}(x + y)$	$x, y \in (0, 0.5)$	[2, 5, 5, 1] param #: 51	7.52×10^{-4} $\pm 2.55 \times 10^{-4}$	[2, 2, 1, 1] param #: 63	1.30×10^{-5} $\pm 1.64 \times 10^{-5}$	[2, 2, 1, 1] param #: 47	3.00×10^{-5} $\pm 3.47 \times 10^{-5}$	[2, 2, 1, 1] param #: 547	2.51×10^{-5} $\pm 3.04 \times 10^{-5}$

- Feynman Function Dataset:** To evaluate RPN’s effectiveness on real-world complex functions relevant to scientific research, we utilize the Feynman function dataset from [73]. Unlike [51], which simplifies the functions to dimensionless forms, we use the original Feynman functions² with their provided value ranges. The 27 functions used in our experiments, along with their input variable value ranges, are shown in the first three columns of Table 4.

All datasets used in these experiments have been made available in the **TINYBIG** toolkit, allowing readers to conduct follow-up experimental testing and result replication.

²<https://space.mit.edu/home/tegmark/aifeynman.html>

Table 3: Experimental results of continuous function approximation on the composite function dataset. The results are obtained in the same way and reported in the same format as the previous table on elementary functions.

Eq.	Formula	Variables	MLP Architecture	MLP MSE	KAN Architecture	KAN MSE	RPN-Ext Architecture	RPN-Ext MSE	RPN-Nstd Architecture	RPN-Nstd MSE
C.0	$\frac{(x+y)}{1/(x+y)}$	$x, y \in (0, 1)$	[2, 10, 10, 1] param #: 151	2.18×10^{-1} $\pm 2.85 \times 10^{-1}$	[2, 2, 2, 1] param #: 150	2.95×10^{-1} $\pm 4.85 \times 10^{-1}$	[2, 2, 2, 1] param #: 71	3.25×10^{-2} $\pm 3.34 \times 10^{-2}$	[2, 2, 2, 1] param #: 821	2.73×10^{-3} $\pm 3.32 \times 10^{-3}$
C.1	$\frac{(x+y)}{(x+y)^2}$	$x, y \in (0, 1)$	[2, 10, 10, 1] param #: 151	3.86×10^{-4} $\pm 2.40 \times 10^{-4}$	[2, 2, 2, 1] param #: 150	5.04×10^{-7} $\pm 2.07 \times 10^{-7}$	[2, 2, 2, 1] param #: 71	6.73×10^{-7} $\pm 3.04 \times 10^{-7}$	[2, 2, 2, 1] param #: 821	1.84×10^{-7} $\pm 1.19 \times 10^{-7}$
C.2	$\frac{(x+y)^2}{1+\exp(x+y)}$	$x, y \in (0, 1)$	[2, 10, 10, 1] param #: 151	2.99×10^{-3} $\pm 1.62 \times 10^{-3}$	[2, 2, 2, 1] param #: 150	1.61×10^{-6} $\pm 7.77 \times 10^{-7}$	[2, 2, 2, 1] param #: 71	1.33×10^{-6} $\pm 1.29 \times 10^{-6}$	[2, 2, 2, 1] param #: 821	1.54×10^{-6} $\pm 9.85 \times 10^{-7}$
C.3	$\exp(x+y) + \ln(x+y)$	$x, y \in (0, 1)$	[2, 10, 10, 1] param #: 151	2.97×10^{-2} $\pm 3.03 \times 10^{-3}$	[2, 2, 2, 1] param #: 150	4.53×10^{-5} $\pm 6.66 \times 10^{-5}$	[2, 2, 2, 1] param #: 71	1.17×10^{-4} $\pm 1.18 \times 10^{-4}$	[2, 2, 2, 1] param #: 821	3.86×10^{-6} $\pm 2.99 \times 10^{-6}$
C.4	$\frac{(x+y)^2}{\sin(x+y)}$	$x, y \in (0, 1)$	[2, 10, 10, 1] param #: 151	1.35×10^{-4} $\pm 6.03 \times 10^{-5}$	[2, 2, 2, 1] param #: 150	4.80×10^{-7} $\pm 6.21 \times 10^{-7}$	[2, 2, 2, 1] param #: 71	4.40×10^{-7} $\pm 4.59 \times 10^{-7}$	[2, 2, 2, 1] param #: 821	1.09×10^{-7} $\pm 6.68 \times 10^{-8}$
C.5	$\cos(x+y) + \arccos(x+y)$	$x, y \in (0, 0.5)$	[2, 10, 10, 1] param #: 151	5.69×10^{-4} $\pm 7.87 \times 10^{-4}$	[2, 2, 2, 1] param #: 150	2.28×10^{-7} $\pm 1.46 \times 10^{-7}$	[2, 2, 2, 1] param #: 71	1.48×10^{-6} $\pm 1.69 \times 10^{-6}$	[2, 2, 2, 1] param #: 821	7.24×10^{-8} $\pm 4.28 \times 10^{-8}$
C.6	$\exp(x+y) \times 1/(x+y)$	$x, y \in (0, 1)$	[2, 10, 10, 1] param #: 151	2.25×10^{-1} $\pm 2.86 \times 10^{-1}$	[2, 2, 2, 1] param #: 150	8.19×10^{-2} $\pm 1.46 \times 10^{-1}$	[2, 2, 2, 1] param #: 71	3.72×10^{-2} $\pm 3.95 \times 10^{-2}$	[2, 2, 2, 1] param #: 821	2.25×10^{-3} $\pm 2.49 \times 10^{-3}$
C.7	$\frac{(x+y)^2}{x \ln(x+y)}$	$x, y \in (0, 1)$	[2, 10, 10, 1] param #: 151	9.74×10^{-4} $\pm 1.24 \times 10^{-3}$	[2, 2, 2, 1] param #: 150	7.00×10^{-8} $\pm 3.38 \times 10^{-8}$	[2, 2, 2, 1] param #: 71	4.97×10^{-7} $\pm 5.01 \times 10^{-7}$	[2, 2, 2, 1] param #: 821	2.94×10^{-8} $\pm 1.37 \times 10^{-8}$
C.8	$\frac{(x+y)}{\sin(x+y)}$	$x, y \in (0, 1)$	[2, 10, 10, 1] param #: 151	3.80×10^{-5} $\pm 3.60 \times 10^{-5}$	[2, 2, 2, 1] param #: 150	3.03×10^{-8} $\pm 1.79 \times 10^{-8}$	[2, 2, 2, 1] param #: 71	5.70×10^{-8} $\pm 1.98 \times 10^{-8}$	[2, 2, 2, 1] param #: 821	1.64×10^{-8} $\pm 9.49 \times 10^{-9}$
C.9	$\exp(x+y) \times \ln(x+y)$	$x, y \in (0, 1)$	[2, 10, 10, 1] param #: 151	4.78×10^{-3} $\pm 2.58 \times 10^{-3}$	[2, 2, 2, 1] param #: 150	3.10×10^{-5} $\pm 2.39 \times 10^{-5}$	[2, 2, 2, 1] param #: 71	1.88×10^{-4} $\pm 2.20 \times 10^{-4}$	[2, 2, 2, 1] param #: 821	4.55×10^{-6} $\pm 3.99 \times 10^{-6}$
C.10	$\frac{\sin(x+y)}{\sinh(x+y)}$	$x, y \in (0, 1)$	[2, 10, 10, 1] param #: 151	1.82×10^{-4} $\pm 6.47 \times 10^{-5}$	[2, 2, 2, 1] param #: 150	1.19×10^{-7} $\pm 9.75 \times 10^{-8}$	[2, 2, 2, 1] param #: 71	3.40×10^{-7} $\pm 4.77 \times 10^{-7}$	[2, 2, 2, 1] param #: 821	3.58×10^{-8} $\pm 1.18 \times 10^{-8}$
C.11	$\arccos(x+y) \times \operatorname{arctanh}(x+y)$	$x, y \in (0, 0.5)$	[2, 10, 10, 1] param #: 151	1.08×10^{-4} $\pm 9.94 \times 10^{-5}$	[2, 2, 2, 1] param #: 150	2.75×10^{-7} $\pm 2.34 \times 10^{-7}$	[2, 2, 2, 1] param #: 71	3.57×10^{-7} $\pm 2.56 \times 10^{-7}$	[2, 2, 2, 1] param #: 821	4.87×10^{-7} $\pm 8.71 \times 10^{-7}$
C.12	$\exp(\frac{1}{(x+y)}) + \exp((x+y))$	$x, y \in (0, 0.5)$	[2, 10, 10, 1] param #: 151	1.07×10^{-1} $\pm 1.52 \times 10^{-1}$	[2, 2, 2, 1] param #: 150	3.81×10^{-4} $\pm 4.55 \times 10^{-4}$	[2, 2, 2, 1] param #: 71	3.74×10^{-5} $\pm 1.88 \times 10^{-5}$	[2, 2, 2, 1] param #: 821	7.17×10^{-5} $\pm 8.87 \times 10^{-5}$
C.13	$\exp(\sin(x+y) + \cos(x+y))$	$x, y \in (0, 1)$	[2, 10, 10, 1] param #: 151	2.35×10^{-3} $\pm 2.65 \times 10^{-3}$	[2, 2, 2, 1] param #: 150	3.62×10^{-7} $\pm 1.65 \times 10^{-7}$	[2, 2, 2, 1] param #: 71	6.97×10^{-7} $\pm 2.29 \times 10^{-7}$	[2, 2, 2, 1] param #: 821	8.56×10^{-8} $\pm 3.81 \times 10^{-8}$
C.14	$\ln((x+y)^2 + \exp(x+y))$	$x, y \in (0.5, 1)$	[2, 10, 10, 1] param #: 151	1.26×10^{-5} $\pm 1.72 \times 10^{-5}$	[2, 2, 2, 1] param #: 150	6.30×10^{-8} $\pm 7.82 \times 10^{-8}$	[2, 2, 2, 1] param #: 71	3.50×10^{-7} $\pm 3.89 \times 10^{-7}$	[2, 2, 2, 1] param #: 821	1.69×10^{-8} $\pm 2.42 \times 10^{-8}$
C.15	$\tan(\exp(x+y) + \ln(x+y))$	$x, y \in (0, 1)$	[2, 10, 10, 1] param #: 151	6.48×10^2 $\pm 4.50 \times 10^2$	[2, 2, 2, 1] param #: 150	1.74×10^3 $\pm 1.72 \times 10^3$	[2, 2, 2, 1] param #: 71	2.47×10^3 $\pm 4.39 \times 10^3$	[2, 2, 2, 1] param #: 821	1.17×10^4 $\pm 2.11 \times 10^4$
C.16	$\frac{1}{1+\exp(-x-y)}$	$x, y \in (0, 1)$	[2, 10, 10, 1] param #: 151	1.48×10^{-6} $\pm 1.69 \times 10^{-6}$	[2, 2, 2, 1] param #: 150	2.86×10^{-9} $\pm 1.75 \times 10^{-9}$	[2, 2, 2, 1] param #: 71	1.74×10^{-9} $\pm 1.09 \times 10^{-9}$	[2, 2, 2, 1] param #: 821	1.48×10^{-9} $\pm 1.32 \times 10^{-9}$

Experiment Setups: For the continuous function approximation task, we randomly generate 2000 input-output pairs for each function in the dataset. These pairs are divided into training and testing sets, with a 50 : 50 ratio. To ensure fair comparisons, each model is trained with five different random seeds over 2000 epochs. The best results encountered during training are selected to mitigate biases from epoch hyper-parameter selection. We use Mean Squared Error (MSE) as the evaluation metric. To account for variability due to random seed selection, we report the final evaluation results as “mean \pm std” of MSE scores obtained from the five random seeds. It allows us to provide a comprehensive and unbiased assessment of model performance across multiple runs.

7.1.2 The Main Results of RPN on Continuous Function Approximation

Tables 2, 3, and 4 present the main results of RPN compared to MLP and KAN on the elementary, composite, and Feynman function datasets, respectively. Each table shows MSE (mean \pm std) scores for each method, obtained using five random seeds. For the elementary and composite function datasets, we also provide the architecture and parameter counts of the compared methods.

For elementary functions, MLP uses a [2, 5, 5, 1] architecture with 51 parameters (including bias), while KAN uses [2, 2, 1, 1] with 5 input intervals divided by the knots, b-splines of order 3, and 63 parameters. For composite and Feynman functions, MLP uses [2, 10, 10, 1] with 151 parameters, and KAN uses [2, 2, 2, 1] with 150 parameters, 10 intervals and order 4. We compare two variants of RPN: (1) RPN-Ext using extended expansions (B-spline and Taylor’s), LoRR reconciliation (rank

Table 4: Experimental results of continuous function approximation on the Feynman function dataset. Method RPN-Ext uses extended expansion, low-rank reconciliation and linear remainder.

Eq.	Formula	Variables	MLP MSE	KAN MSE	RPN-Ext MSE
I.6.2	$\exp\left(-\frac{\theta^2}{2\sigma^2}\right)/\sqrt{2\pi}\sigma$	$\theta, \sigma \in [1, 3]$	7.17×10^{-5} $\pm 8.37 \times 10^{-5}$	3.20×10^{-7} $\pm 2.02 \times 10^{-7}$	1.63×10^{-6} $\pm 1.31 \times 10^{-6}$
I.6.2b	$\exp\left(-\frac{(\theta-\theta_1)^2}{2\sigma^2}\right)/\sqrt{2\pi}\sigma$	$\sigma, \theta, \theta_1 \in [1, 3]$	4.52×10^{-5} $\pm 1.90 \times 10^{-5}$	8.97×10^{-5} $\pm 1.77 \times 10^{-4}$	1.60×10^{-5} $\pm 1.25 \times 10^{-5}$
I.9.18	$\frac{G \cdot m_1 \cdot m_2}{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$	$G, m_1, m_2, x_2, y_2, z_2 \in [1, 2]$ $x_1, y_1, z_1 \in [3, 4]$	2.17×10^{-4} $\pm 7.42 \times 10^{-5}$	1.60×10^{-4} $\pm 9.60 \times 10^{-5}$	6.92×10^{-5} $\pm 2.42 \times 10^{-5}$
I.12.11	$q \cdot (E_f + B \cdot v \cdot \sin(\theta))$	$q, E_f, B, v, \theta \in [1, 5]$	9.81×10^0 $\pm 2.36 \times 10^0$	4.78×10^1 $\pm 2.05 \times 10^1$	1.36×10^1 $\pm 8.35 \times 10^0$
I.13.12	$G \cdot m_1 \cdot m_2 \left(\frac{1}{r_2} - \frac{1}{r_1}\right)$	$G, m_1, m_2, r_1, r_2 \in [1, 5]$	1.24×10^0 $\pm 2.83 \times 10^{-1}$	7.07×10^0 $\pm 3.25 \times 10^0$	8.47×10^{-1} $\pm 1.21 \times 10^{-1}$
I.15.3x	$\frac{(x-u \cdot t)}{\sqrt{1-u^2/c^2}}$	$x \in [5, 10], u \in [1, 2]$ $c \in [3, 20], t \in [1, 2]$	1.07×10^{-2} $\pm 9.00 \times 10^{-3}$	1.49×10^{-2} $\pm 4.94 \times 10^{-3}$	4.94×10^{-3} $\pm 1.57 \times 10^{-3}$
I.16.6	$\frac{(u+v)}{(1+u \cdot v/c^2)}$	$c, v, u \in [1, 5]$	3.39×10^{-3} $\pm 3.31 \times 10^{-3}$	8.47×10^{-3} $\pm 2.67 \times 10^{-3}$	8.68×10^{-4} $\pm 3.84 \times 10^{-4}$
I.18.4	$(m_1 \cdot r_1 + m_2 \cdot r_2)/(m_1 + m_2)$	$m_1, m_2, r_1, r_2 \in [1, 5]$	1.16×10^{-2} $\pm 1.13 \times 10^{-2}$	2.00×10^{-2} $\pm 5.02 \times 10^{-3}$	9.92×10^{-4} $\pm 4.82 \times 10^{-4}$
I.26.2	$\arcsin(n \cdot \sin(\theta_2))$	$n \in [0, 1], \theta_2 \in [1, 5]$	9.59×10^{-4} $\pm 3.33 \times 10^{-4}$	7.64×10^{-5} $\pm 1.12 \times 10^{-4}$	6.59×10^{-5} $\pm 5.13 \times 10^{-5}$
I.27.6	$\frac{1}{d_1 + \frac{n}{d_2}}$	$d_1, d_2, n \in [1, 5]$	5.40×10^{-4} $\pm 3.52 \times 10^{-4}$	8.22×10^{-4} $\pm 1.24 \times 10^{-4}$	4.98×10^{-4} $\pm 6.95 \times 10^{-4}$
I.29.16	$\sqrt{x_1^2 + x_2^2 - 2x_1 x_2 \cos(\theta_1 - \theta_2)}$	$x_1, x_2, \theta_1, \theta_2 \in [1, 5]$	1.16×10^{-1} $\pm 1.41 \times 10^{-1}$	1.40×10^0 $\pm 6.16 \times 10^{-1}$	7.27×10^{-2} $\pm 7.00 \times 10^{-2}$
I.30.3	$\text{Int}_0 \cdot \sin\left(n \frac{\theta}{2}\right)^2 / \sin\left(\frac{\theta}{2}\right)^2$	$\text{Int}_0, \theta, n \in [1, 5]$	2.17×10^0 $\pm 2.42 \times 10^{-1}$	2.13×10^0 $\pm 6.78 \times 10^{-1}$	1.58×10^0 $\pm 8.47 \times 10^{-1}$
I.30.5	$\arcsin\left(\frac{\lambda}{n \cdot d}\right)$	$\lambda \in [1, 2], d \in [2, 5], n \in [1, 5]$	2.45×10^{-4} $\pm 2.14 \times 10^{-4}$	6.26×10^{-5} $\pm 5.11 \times 10^{-5}$	6.83×10^{-6} $\pm 2.53 \times 10^{-6}$
I.37.4	$I_1 + I_2 + 2\sqrt{I_1 \cdot I_2} \cdot \cos(\delta)$	$I_1, I_2, \delta \in [1, 5]$	9.14×10^{-2} $\pm 5.89 \times 10^{-2}$	1.74×10^{-1} $\pm 1.43 \times 10^{-1}$	3.89×10^{-2} $\pm 3.06 \times 10^{-2}$
I.40.1	$n_0 \exp\left(-\frac{m \cdot g \cdot x}{(k_b \cdot T)}\right)$	$n_0, m, g, x, k_b, T \in [1, 5]$	8.90×10^{-3} $\pm 6.01 \times 10^{-3}$	1.09×10^{-2} $\pm 6.98 \times 10^{-3}$	2.13×10^{-3} $\pm 4.51 \times 10^{-4}$
I.44.4	$n \cdot k_b \cdot T \cdot \ln\left(\frac{V_2}{V_1}\right)$	$n, k_b, T, V_1, V_2 \in [1, 5]$	5.69×10^0 $\pm 1.06 \times 10^0$	2.58×10^1 $\pm 1.76 \times 10^1$	2.99×10^0 $\pm 5.68 \times 10^{-1}$
I.50.26	$x_1 \cdot (\cos(\omega t) + \alpha \cdot \cos(\omega t)^2)$	x_1, ω, t, α	1.42×10^0 $\pm 1.07 \times 10^0$	7.10×10^{-1} $\pm 2.03 \times 10^{-1}$	1.03×10^0 $\pm 7.47 \times 10^{-1}$
II.2.42	$\frac{\kappa \cdot (T_2 - T_1) \cdot A}{d}$	$\kappa, T_1, T_2, A, d \in [1, 3]$	8.73×10^{-1} $\pm 3.55 \times 10^{-1}$	1.09×10^0 $\pm 7.20 \times 10^{-1}$	6.98×10^{-1} $\pm 3.27 \times 10^{-1}$
II.6.15a	$\frac{3z P_d}{(4\pi e)^3} \sqrt{x^2 + y^2}$	$\epsilon, p_d, r, x, y, z \in [1, 3]$	2.07×10^{-3} $\pm 1.61 \times 10^{-3}$	6.36×10^{-4} $\pm 3.78 \times 10^{-4}$	9.29×10^{-5} $\pm 3.06 \times 10^{-5}$
II.11.17	$n_0 \left(1 + \frac{p_d \cdot E_f \cos \theta}{k_b \cdot T}\right)$	$n_0, k_b, T, \theta, p_d, E_f \in [1, 3]$	1.10×10^{-1} $\pm 1.35 \times 10^{-1}$	1.01×10^{-1} $\pm 7.75 \times 10^{-2}$	2.85×10^{-2} $\pm 3.35 \times 10^{-3}$
II.11.27	$\frac{n \cdot \alpha}{1 - \frac{n \cdot \alpha}{3}} e E_f$	$n, \alpha \in [0, 1], e, E_f \in [1, 2]$	2.23×10^{-2} $\pm 2.30 \times 10^{-2}$	8.43×10^{-5} $\pm 6.55 \times 10^{-5}$	9.93×10^{-5} $\pm 2.70 \times 10^{-5}$
II.35.18	$\frac{n_0}{\exp\left(\frac{\mu_m \cdot B}{k_b \cdot T}\right) + \exp\left(-\frac{\mu_m \cdot B}{k_b \cdot T}\right)}$	$n_0, k_b, T, \mu_m, B \in [1, 3]$	7.56×10^{-4} $\pm 2.87 \times 10^{-4}$	1.39×10^{-3} $\pm 2.38 \times 10^{-3}$	1.59×10^{-4} $\pm 1.08 \times 10^{-4}$
II.36.38	$\frac{\mu_m \cdot H}{k_b \cdot T} + \frac{\mu_m \cdot \alpha \cdot M}{e \cdot c^2 \cdot k_b \cdot T}$	$\mu_m, H, k_b, T, \alpha, e, c, M \in [1, 3]$	5.44×10^{-2} $\pm 3.07 \times 10^{-2}$	3.49×10^{-2} $\pm 9.15 \times 10^{-3}$	4.89×10^{-3} $\pm 2.59 \times 10^{-3}$
II.38.3	$\frac{Y \cdot A \cdot x}{d}$	$Y, A, d, x \in [1, 5]$	5.44×10^0 $\pm 8.81 \times 10^0$	1.38×10^0 $\pm 4.98 \times 10^{-1}$	1.24×10^{-1} $\pm 1.02 \times 10^{-1}$
III.9.52	$\frac{p_d \cdot E_f \cdot t \cdot \sin\left(\frac{(\omega - \omega_0)t}{2}\right)^2}{\frac{J_c}{2\pi} \cdot \left(\frac{(\omega - \omega_0)t}{2}\right)^2}$	$p_d, E_f, t, h \in [1, 3]$ $\omega, \omega_0 \in [1, 5]$	9.20×10^0 $\pm 1.59 \times 10^0$	1.88×10^1 $\pm 3.65 \times 10^0$	7.10×10^0 $\pm 2.92 \times 10^0$
III.10.19	$\mu_m \cdot \sqrt{B_x^2 + B_y^2 + B_z^2}$	$\mu_m, B_x, B_y, B_z \in [1, 5]$	4.17×10^{-1} $\pm 2.59 \times 10^{-1}$	2.69×10^{-1} $\pm 8.30 \times 10^{-2}$	3.08×10^{-2} $\pm 9.13 \times 10^{-3}$
III.17.37	$\beta \cdot (1 + \alpha \cdot \cos(\theta))$	$\beta, \alpha, \theta \in [1, 5]$	8.06×10^{-1} $\pm 5.30 \times 10^{-1}$	1.62×10^0 $\pm 1.12 \times 10^0$	2.66×10^{-1} $\pm 1.64 \times 10^{-1}$

1), and zero remainder; and (2) RPN-Nstd using nested expansions (B-spline and Taylor's), LoRR reconciliation (rank 1), and zero remainder. Both variants create higher-dimensional intermediate expanded vectors. RPN-Ext has fewer learnable parameters than MLP and KAN for elementary functions and less than half for composite and Feynman functions.

The results in Tables 2-4 show that RPN-Ext outperforms MLP, reducing MSE by at least $\times 10^{-1}$ across almost all these functions, with improvements up to $\times 10^{-5}$ for some elementary functions

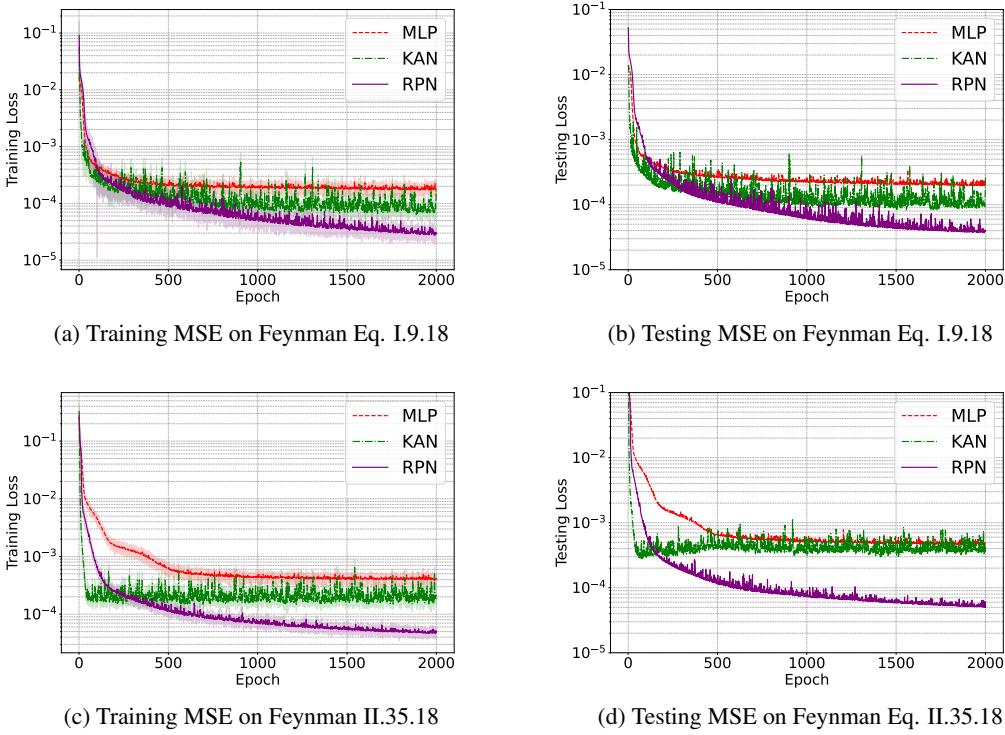


Figure 11: Training and testing MSE curves of MLP, KAN and RPN on fitting Feynman functions Eq. I.9.18 and Eq. II.35.18. The x axis denotes the training epochs and the y axes denote the training and testing MSE of RPN on these two functions, respectively.

(E.5, E.6) and $\times 10^{-4}$ for some composite functions (C.7, C.13). RPN-Ext with fewer parameters still achieves comparable or slightly better performance than KAN. Meanwhile, RPN-Nstd, with more parameters, significantly outperforms KAN, reducing MSE by at least $\times 10^{-1}$ and even $\times 10^{-2}$ for most elementary and composite functions. On the Feynman dataset, RPN-Ext with a linear remainder and half the parameters of MLP and KAN consistently performs well. It outperforms the other methods on 23 out of 27 functions, with improvements of at least $\times 10^{-1}$ and up to $\times 10^{-2}$ in some cases (e.g., Eq. I.18.4). MLP only performs best on Eq. I.12.11, while KAN excels with slightly more advantages on Eq. I.6.2, I.50.26, and II.11.27.

These results demonstrate RPN’s effectiveness in approximating both simple and complex continuous functions used in real-world scientific research discoveries. Additional experiments with different extension and remainder functions of RPN are also available in Tables 10, 11, and 12 in the Appendix Section A.1.

7.1.3 Model Learning Analysis

To further illustrate the training process of MLP, KAN, and RPN in approximating continuous functions, we randomly selected two Feynman functions: Eq. I.9.18 and Eq. II.35.18 (formulas available in Table 4). Figure 11 presents the model training and testing curves for each epoch. The RPN method shown in the plots uses extended expansion, low-rank reconciliation (rank=1), and a linear remainder function. Notably, this configuration of RPN has only half the learnable parameters of MLP and KAN.

The plots reveal that both MLP and KAN can approximate these functions with MSE on the scale of 10^{-4} . However, RPN, only using less than half of the learnable parameters, significantly outperforms MLP and KAN, achieving MSE on the scale of 10^{-5} . Moreover, RPN’s training and testing

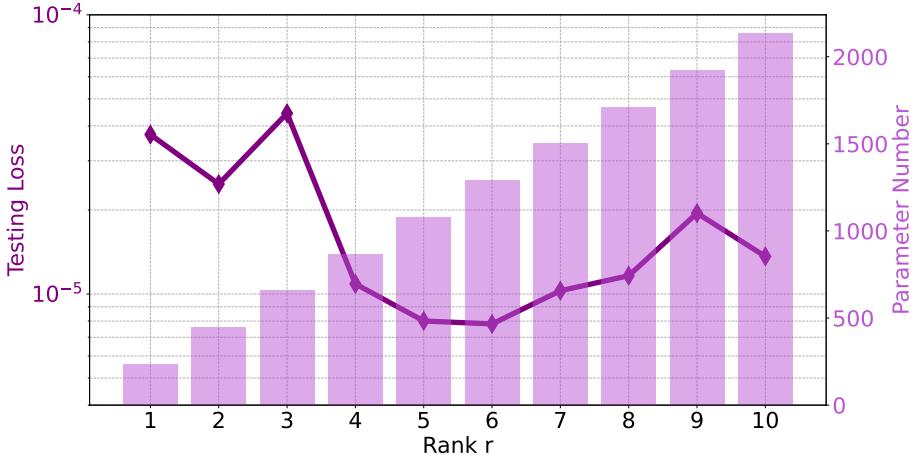


Figure 12: Analysis of rank parameter r for low-rank reconciliation function used in RPN on Feynman function Eq. I.9.18. The x axis: rank parameter r value; left y axis: testing MSE loss; and right y axis: learnable parameter number.

curves consistently descend with increasing epochs, in contrast to MLP and KAN. Remarkably, even without explicit parameter regularization, RPN does not exhibit overfitting problems. These results further demonstrate RPN’s superior performance and stability in continuous function approximation tasks.

7.1.4 Low-Rank Reconciliation Function Rank Parameter Selection

The RPN model analyzed previously used a low-rank reconciliation function with rank $r = 1$ by default, which reduced the number of parameters to half that of MLP and KAN for both composite and Feynman function datasets. However, in real-world applications where parameter constraints are less stringent, the rank parameter in the LoRR reconciliation function can be fine-tuned to further enhance RPN’s learning performance beyond what was reported in the previous tables.

Figure 12 illustrates the learning performance and parameter count of RPN when approximating Feynman Eq. I.9.18 with varying rank values. The testing loss curve shows that as the rank parameter r increases, the model’s testing loss initially decreases steadily, then increases, reaching its lowest point of 7.81×10^{-6} at rank $r = 6$. Concurrently, the number of parameters in RPN grows consistently from 235 at $r = 1$ to 134 at $r = 10$.

The following subsection on discrete data classifications will provide more detailed information on ablation studies of RPN with different expansion, reconciliation, and remainder functions.

7.2 Discrete Image and Text Classification

In addition to continuous function approximation tasks, this subsection examines the effectiveness of RPN for discrete data classification, encompassing both image and text classification. We organize this subsection as follows: First, we introduce the dataset descriptions and experimental setups. Next, we investigate the effectiveness of different data expansion techniques, parameter reconciliation methods, and remainder functions, as well as conduct detailed analyses of RPN in terms of model and reconciliation function hyper-parameters. Based on these insights, we apply RPN to classification tasks on benchmark datasets for both images and text. Finally, we visualize the data expansions and parameter reconciliation process to help interpret the learning process and results of RPN.

7.2.1 Dataset Descriptions and Experiment Setups

Table 5: Statistics of discrete image and text datasets. For the text datasets, we convert each text data instance to a bag-of-word vector rescaled by TF-IDF, whose dimensions are also provided in the table.

	Image Datasets		Text Datasets		
	MNIST	CIFAR-10	IMDB	AGNews	SST2
Train #	60,000	50,000	25,000	120,000	67,349
Test #	10,000	10,000	25,000	7,600	872
Input Dim.	28×28	$32 \times 32 \times 3$	26,964	25,985	10,325
Output Dim.	10	10	2	4	2

Dataset Descriptions: To demonstrate the generalizability and effectiveness of RPN, in this part, we will provide the experimental investigations of RPN for discrete data classification. Specifically, this subsection will focus on the experimental investigations on two categories of discrete datasets described below, whose basic statistical information is also provided in Table 5.

- **Image Datasets:** We use two benchmark datasets, MNIST and CIFAR-10, to investigate the performance of RPN for image classification. Images in MNIST are all in the grayscale, while those in CIFAR-10 are colored instead.
- **Text Datasets:** To examine the performance of RPN for text classification, we employ three text benchmark datasets: IMDB, AGNews, and SST2. The AGNews is a multi-class dataset, while IMDB and SST2 are both binary-class datasets.

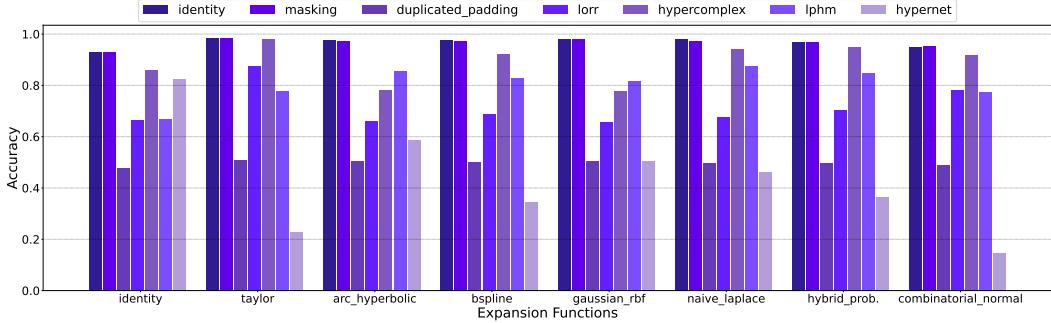
Experiment Setups: These image and text benchmark datasets have been pre-partitioned into training and testing sets, which will be used for all methods in our experiments. We preprocess the images by flattening and normalizing them before model training. For the text datasets, we use the sklearn TF-IDF vectorizer to preprocess the text inputs. The performance of all comparison methods, including RPN, on these datasets are evaluated by using Accuracy as the default metric.

7.2.2 Component Function Composition Analysis

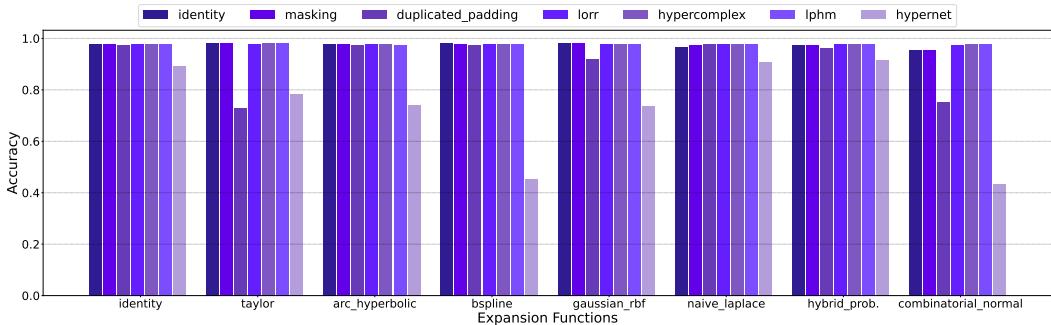
Before presenting the performance of RPN and other comparison baselines, we will first analyze the component functions involved in RPN to inform the design of an architecture that can achieve better performance. Figure 13 illustrates various combinations of the expansion, reconciliation, and remainder functions, as well as their performance on the MNIST dataset for image classification. Plot (a) corresponds to the zero remainder function, while Plot (b) corresponds to the linear remainder function. For both plots, the x axis denotes the expansion functions, and the bars in different colors denote different reconciliation functions, whose names are indicated in the plot legends.

The plots reveal several interesting observations for RPN’s performance. First of all, for RPN with the zero remainder function as illustrated in Plot (a), its performance will solely depend on the expansion and reconciliation functions. Meanwhile, among all these reconciliation functions, identity and masking reconciliation outperform others, while hypernet and duplicated padding-based reconciliation underperform expectations. For expansion functions, Taylor’s expansion, Gaussian RBF expansion, and naive Laplace distribution-based expansion slightly outperform the others. Notably, by comparing Plot (a) with Plot (b), we can observe that the linear remainder function dramatically improves performance for most expansion-reconciliation function pairs.

These observations provide valuable insights about the effectiveness of different component function combinations in RPN. They will also help guide the selection of optimal functions for improved performance of RPN to address the discrete data classification tasks studied in this paper.



(a) Ablation Studies on Expansion and Reconciliation Functions of RPN with Zero Remainder



(b) Ablation Studies on Expansion and Reconciliation Functions of RPN with Linear Remainder

Figure 13: An illustration of the learning performance of RPN with different expansion, reconciliation and remainder functions based on the MNIST dataset. The x axis denotes different expansion functions, and the y axis denotes the testing accuracy obtained by these methods. The bars with different colors denote different reconciliation functions with their names indicated in the legend. Plot (a): RPN with zero remainder function; and Plot (b): RPN with linear remainder function. For all the reconciliation functions, we just use their default hyper-parameters in the **TINYBIG** toolkit without any tuning.

7.2.3 Individual Component Function Analysis

Furthermore, to investigate the effectiveness of individual component functions, we also enumerate all potential combinations of these functions to build the RPN model and apply it to the MNIST dataset. For each individual function, we extract all these potential combinations involving them and obtain their testing accuracy scores. The median, mean, and max performance are provided in Figure 14. To exclude the performance boost created by the linear remainder, we use zero remainder by default for Plots (a)-(f), which will illustrate their expansion and reconciliation functions' effectiveness without remainders.

In Figure 14, we sort these component functions according to their max, median, and mean accuracy scores, respectively. Among the 21 expansion functions investigated, those obtaining the highest accuracy scores in Plot (a) are Taylor's expansion, Inverse Quadratic RBF-based expansion, Gaussian RBF-based expansion, Naive Laplace-based expansion, and B-spline-based expansion. In addition to the maximum accuracy scores, the median and mean accuracy-based ranking of the expansion functions in Plots (b)-(c) illustrates the robustness of their performance when combined with various reconciliation functions.

Regarding reconciliation functions, as illustrated in Plot (d), those achieving the highest accuracy scores include Identity reconciliation, Masking-based reconciliation, and Hypercomplex Multiplication-based reconciliation. As shown in Plots (d)-(f), consistent with Figure 13, the per-

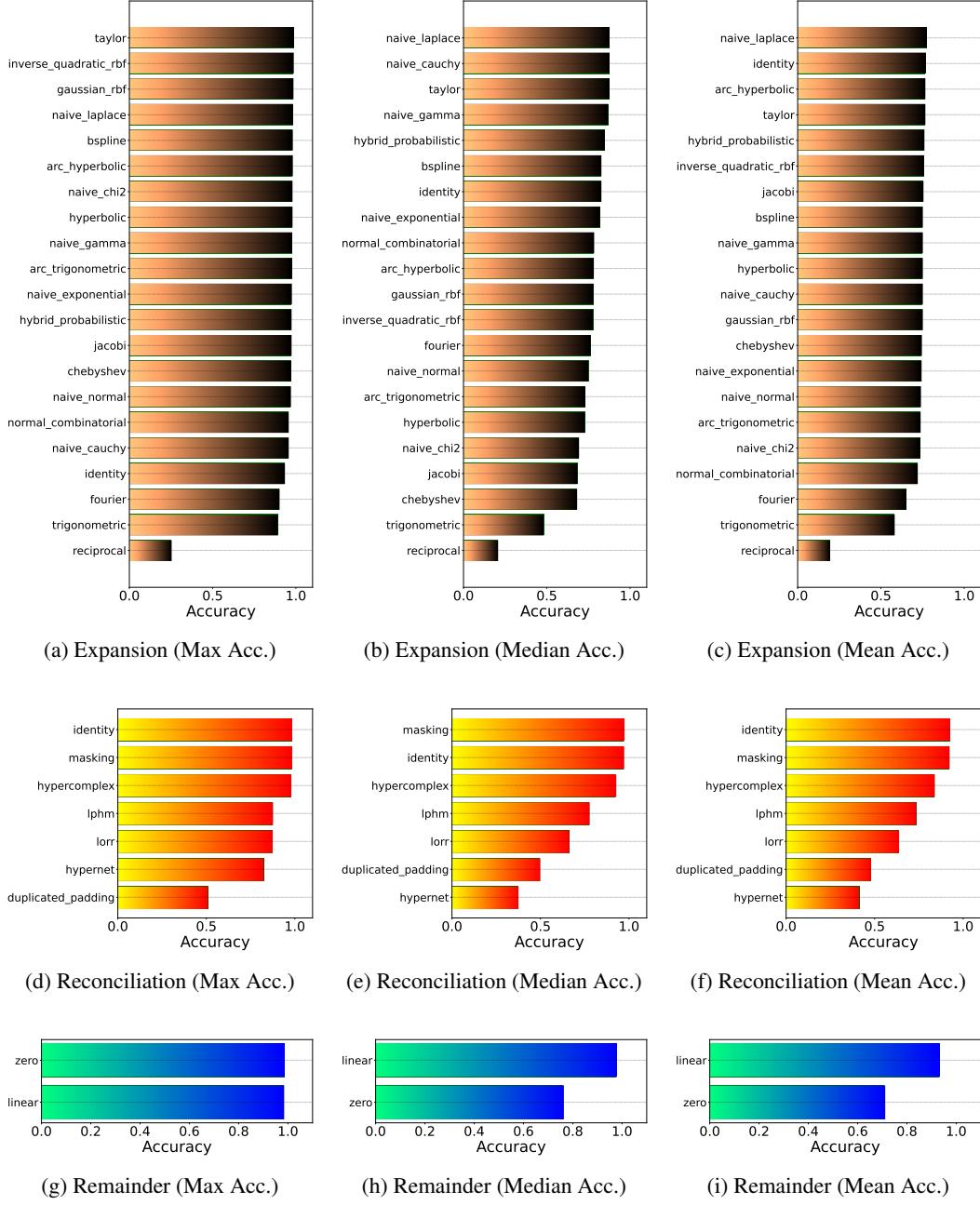


Figure 14: An illustration of the effectiveness of individual component functions in RPN based on the MNIST dataset. Plots (a)-(c): performance of expansion functions; Plots (d)-(f): performance of reconciliation functions; and Plots (g)-(i): performance of remainder functions. For each individual function, we calculate the max, median, and mean accuracy scores obtained by the compositions involving them in the plots.

formance of the hypernet and duplicated padding-based reconciliation is slightly below the expectations, especially compared with the other reconciliation functions. It's important to note that we use default hyper-parameters for all these reconciliation functions, and the lower scores for these functions reported here don't necessarily indicate their ineffectiveness. Below, we will also provide a tuning of the hyper-parameters for some of the reconciliation functions to demonstrate that they can

achieve comparable performance to the identity reconciliation functions with minor hyper-parameter tuning, while having far fewer learnable parameters.

For the remainder functions, according to Plots (g)-(i), linear remainder consistently outperforms zero remainder, demonstrating that remainder functions can provide complementary information to the learning results. In addition to these plots, more comprehensive results of the performance scores, time cost, and parameter number of these different component function compositions are provided in Tables 13-33 in the Appendix Section A.2. Readers may also refer to those tables for more detailed investigation results of RPN composed with different component functions.

7.2.4 Model Depth and Width Analysis

Table 6: An investigation of model depth and width on the performance of RPN.

	Model Layer Number (based on RPN with 1 head per layer)									
	1	2	3	4	5	6	7	8	9	10
Accuracy	0.9791	0.9849	0.9853	0.9855	0.9852	0.9849	0.9848	0.9842	0.9833	0.9830
Param. #	6.15M	39.43M	39.7M	39.96M	40.23M	40.49M	40.76M	41.03M	41.29M	41.56M
	Model Head Number (based on RPN with 4 layer)									
	1	2	3	4	5	6	7	8	9	10
Accuracy	0.9855	0.9853	0.9846	0.9867	0.9847	0.9852	0.9851	0.9862	0.9852	0.9856
Param. #	39.96M	79.92M	119.89M	159.85M	199.81M	239.77M	279.74M	319.7M	359.66M	399.62M

Beyond the shallow and narrow models analyzed previously, we also investigate the impacts of model depth and width on the performance of RPN. The results are illustrated in Table 6. For this analysis, we use RPN with Taylor’s expansion (order 2), identity reconciliation, and zero remainder as the default model architecture on the MNIST dataset.

The top part of Table 6 shows results for models with 1 head per layer, while increasing the number of layers from 1 to 10. Besides the input and output dimensions, for the hidden dimensions of the middle layers, we all use the default dimension 64 in this experiment. Except for RPN with depth 1, which achieves a best recorded testing accuracy of 0.979, all other depths obtain testing accuracy scores above 0.980. The highest recorded testing accuracy of 0.986 is achieved at depth 4.

Furthermore, for the RPN model with 4 layers, we investigate the impact of head number on model performance, as shown in the bottom part of the table. The performance of RPN with different numbers of heads is notably consistent and stable, achieving the highest scores with 4, 8, and 10 heads, respectively.

7.2.5 Reconciliation Function Hyper-Parameter Tuning Analysis

Prior to illustrating the main results of RPN and other comparison methods, we further investigate the effectiveness of the low-rank reconciliation (LoRR) function, whose initial performance was not as good as expected in the previous Plots (d)-(f) of Figure 14. Using the RPN model with Taylor’s expansion function (order 2) and zero remainder, we will tune the rank hyper-parameter in LoRR from 1 to 15. Figure 15 presents both the obtained testing accuracy and the introduced model parameter numbers. For comparison, we also include the performance of the identity reconciliation function in the plot, which are represented by a gray dashed horizontal line for testing score and a gray bar for parameter number.

The curve and bar plot reveal that LoRR with ranks 1, 2, and 3 yields lower performance than the identity reconciliation function. However, for ranks 4 and above, LoRR’s performance gradually approaches that of the identity reconciliation function while using significantly fewer parameters. This analysis demonstrates that with proper tuning, LoRR can achieve comparable accuracy to identity reconciliation. Similar performance have been observed for the other reconciliation functions as well. These insights will help in designing parameter-efficient RPN models especially for the discrete data classification tasks presented below.

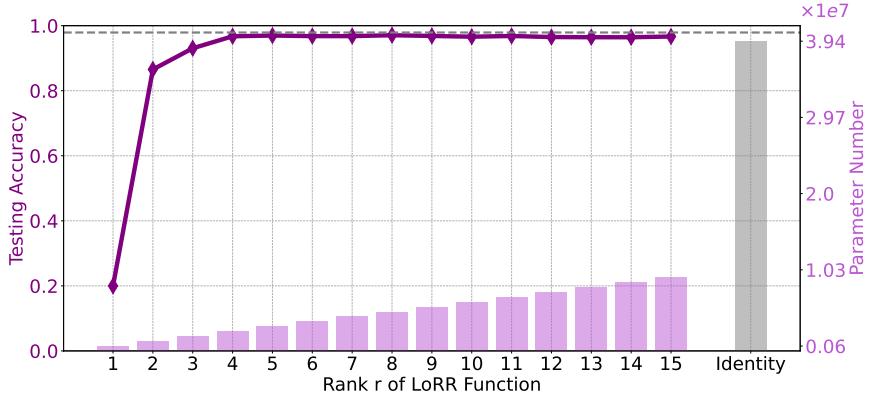


Figure 15: Analysis of rank parameter r for LoRR reconciliation function for RPN on MNIST dataset. Left y axis: Testing Accuracy; Right y axis: Learnable Parameter Number. For the dashed line and bar in gray color, they denote the testing accuracy (0.983) and parameter number (3.94×10^7) of the identity reconciliation function when applied to identical Taylor’s expansions in RPN.

7.2.6 The Main Results of RPN on Discrete Data Classification

Based on our above investigations and analyses, we present the main results of RPN on image and text benchmark datasets in Table 7. We compare RPN’s performance with several baseline models, including naive Bayes (based on the Gaussian and multinomial distributions), SVM (with Linear and RBF kernels), MLP, and KAN. Besides the accuracy scores obtained on the benchmark datasets, the table also specifies model configurations and architecture dimensions of these methods on different benchmark datasets.

In Table 7, the aforementioned Bayesian network and Markov network are excluded from this comparison due to their extremely time-consuming training process on larger-scale datasets. These models will be discussed and compared with RPN on smaller-sized tabular datasets for probabilistic relationship inference in the following subsection. For MLP, KAN, and RPN, we report “mean \pm std” accuracy obtained with 5 randomly picked random seeds. With their implementation via sklearn, the performance scores of naive Bayes and SVM are reported directly without standard deviations due to their stability across random states. RPN’s configurations vary by dataset: on MNIST, RPN uses Taylor’s expansion (order 2), identity reconciliation, zero remainder; on CIFAR-10, RPN uses identity expansion, masking-based reconciliation ($p=0.6$), zero remainder; and on IMDB, AGNews, SST, RPN uses Taylor’s expansion (order 2), low-rank reconciliation ($r=2$), zero remainder.

Except for MNIST, RPN uses much fewer learnable parameters than SVM, MLP, and KAN on these benchmark datasets due to masking and low-rank reconciliation functions. RPN outperforms baseline methods on most discrete benchmark datasets. On MNIST, it achieves 0.986 accuracy, significantly higher than naive Bayes, SVM, MLP, and KAN, demonstrating the effectiveness of Taylor’s expansions in feature extraction from image data. On CIFAR-10 and AGNews, RPN achieves comparable performance to the best baseline methods using fewer learnable parameters. And on IMDB and SST2, the performance of RPN slightly exceeds the other comparison methods. These obtained experimental results all showcase the superiority of RPN on discrete data classification tasks.

In addition to evaluating the performance of RPN, our experiments yielded several insightful observations about the comparison methods. Among all approaches, naive Bayes stands out as the fastest, generating results within seconds, while other methods may require hours for training. SVM (with RBF kernel) and MLP demonstrate more consistent performance across diverse datasets, showcasing their robustness. Notably, the recently proposed KAN exhibits significant limitations in the experiments. Not surprisingly, KAN uses substantially more parameters than other methods and

Table 7: Classification results of discrete data classification on the image and text benchmark datasets. Both naive Bayes and SVM are implemented with sklearn, whose performance is very stable. For MLP, KAN and RPN, we train them with 5 randomly selected random seeds. For each random seed, they are trained with 50 epochs, and the best testing scores achieved by all these methods within these 50 epochs are cherry-picked. For MLP, KAN and RPN, we report their average accuracy score together with the standard deviations in the table.

Models	Image Datasets		Text Datasets		
	MNIST	CIFAR10	IMDB	AGNews	SST2
Naive Bayes	5.56×10^{-1} (Gaussian)	2.98×10^{-1} (Gaussian)	6.30×10^{-1} (Gaussian)	8.14×10^{-1} (Gaussian)	7.31×10^{-1} (Gaussian)
	8.36×10^{-1} (Multinomial)	2.98×10^{-1} (Multinomial)	8.37×10^{-1} (Multinomial)	9.03×10^{-1} (Multinomial)	7.86×10^{-1} (Multinomial)
SVM	9.31×10^{-1} (Linear kernel)	3.50×10^{-1} (Linear kernel)	8.77×10^{-1} (Linear kernel)	9.18×10^{-1} (Linear kernel)	8.01×10^{-1} (Linear kernel)
	9.79×10^{-1} (RBF kernel)	5.44×10^{-1} (RBF kernel)	8.84×10^{-1} (RBF kernel)	9.25×10^{-1} (RBF kernel)	8.01×10^{-1} (RBF kernel)
MLP	9.82×10^{-1} $\pm 6.79 \times 10^{-4}$ [784, 512, 256, 10]	5.63×10^{-1} $\pm 1.67 \times 10^{-3}$ [784, 512, 256, 10]	8.85×10^{-1} $\pm 5.54 \times 10^{-3}$ [26964, 128, 32, 2]	9.21×10^{-1} $\pm 8.22 \times 10^{-4}$ [25985, 128, 32, 4]	8.05×10^{-1} $\pm 2.12 \times 10^{-3}$ [10325, 128, 32, 2]
KAN	9.75×10^{-1} $\pm 1.47 \times 10^{-3}$ [784, 64, 10]	5.27×10^{-1} $\pm 7.10 \times 10^{-4}$ [784, 64, 10]	5.00×10^{-1} $\pm 0.00 \times 10^0$ [26964, 128, 32, 2]	2.50×10^{-1} $\pm 0.00 \times 10^0$ [25985, 128, 32, 4]	4.91×10^{-1} $\pm 0.00 \times 10^0$ [10325, 128, 32, 2]
RPN	9.86×10^{-1} $\pm 8.70 \times 10^{-4}$ [784, 64, 64, 10]	5.61×10^{-1} $\pm 1.66 \times 10^{-3}$ [3072, 512, 256, 10]	8.86×10^{-1} $\pm 4.59 \times 10^{-4}$ [26964, 128, 32, 2]	9.19×10^{-1} $\pm 2.55 \times 10^{-3}$ [25985, 128, 4]	8.07×10^{-1} $\pm 1.72 \times 10^{-3}$ [10325, 128, 32, 2]

requires much longer training time. More critically, KAN fails to train effectively on text datasets using sparse vectorized data instances rescaled via bag-of-words and TF-IDF. These observations reveal major deficiencies in KAN’s model design not discovered nor reported in the previous paper [51], which may pose challenges for it in replacing MLP as a new base model for more complex learning scenarios.

All implementations of RPN in the above experiments have been included in the **TINYBIG** tutorials, allowing readers to rapidly reproduce the reported experimental scores.

7.2.7 Data Expansion and Reconciled Parameter Visualization

To interpret the learning process and obtained results, we illustrate the data expansion results along with the learned parameters of an image from MNIST in Figure 16. For this visualization, we build a 1-layered RPN with Taylor’s expansion (order 2), identity reconciliation, and zero remainder functions. From the testing set, we randomly pick one image with label 0. Prior to expansion, RPN flattens and normalizes the input image into a vector of length 784, using mean-std normalization (with mean: 0.1307 and std: 0.3081).

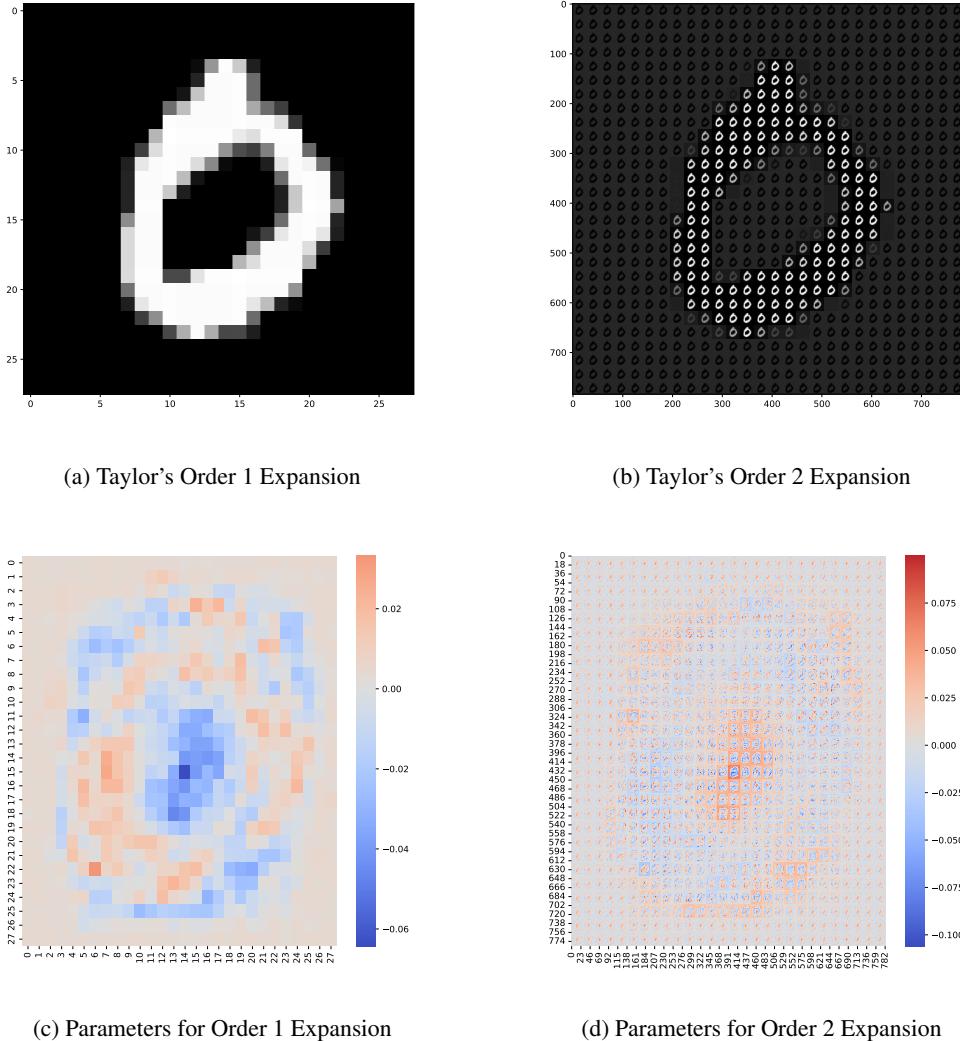


Figure 16: An illustration of image with label 0 from MNIST dataset. Plot (a): Taylor's order 1 expansion; Plot (b): Taylor's order 2 expansion; Plot (c): parameter corresponding to output neuron of label 0 for order 1 expansion; and Plot (d): parameter corresponding to output of label 0 for order 2 expansion.

The Taylor's expansion function extends the input image vector from length 784 to a much longer vector of length $784 + 784 \times 784$, representing Taylor's order 1 and order 2 expansions as shown in Plots (a) and (b), respectively. The RPN model correctly classifies this expanded vector. Plots (c) and (d) display the extracted parameters that project the expanded vector (including both order 1 and 2 expansions) to the output neuron corresponding to label 0.

Plots (a) and (b) demonstrate that the expansion output contains multiple copies of the input image, re-scaled by each pixel value. After mean-std normalization, pixels in the images may have both positive and negative values. As illustrated in Plot (b), pixels with large positive values will increase the image grayscales values, while negative values will invert the image grayscales from light to dark and vice versa. The parameters indicate that both order 1 and order 2 expansions contribute effectively to generating the correct output predictions.

Typically, we expect pixels denoting object location and shapes to be more important and assigned larger weights, which aligns with the parameters observed in Plot (c). However, Plot (d) reveals an unexpected observations: for pixel values in the order 2 expansions, particularly for image copies re-scaled by pixels in the central dark regions, they are assigned much larger weights. This suggests that, after expansions, the model can capture and utilize high-order signals from seemingly less important pixels which may deliver the correct classification results.

In addition to the images with label 0 illustrated in Figure 16, we also randomly selected images with labels 1 through 9 and visualized their expansions and learned parameters. These visualizations are presented in Figures 23-31 in Appendix Section A.3. Consistent patterns are observed across images with different labels. Readers may refer to these visualizations in the Appendix for more comprehensive information.

7.3 Probabilistic Dependency Inference

According to the descriptions introduced in the previous sections, equipped with the probabilistic expansion functions, RPN is capable to learn the probabilistic dependency relationships among the input variables (of both input features and output labels). In addition to the above continuous function approximation and discrete data classification tasks, in this part, we will investigate the effectiveness of RPN for the probabilistic dependency relationship inference task.

7.3.1 Datasets Description and Experiment Setups

Table 8: Statistics of classic machine learning tabular datasets. The datasets are partitioned into training and testing sets with 10-fold cross validation. The training and testing instance numbers for each fold are also reported in the table.

	Classic Machine Learning Tabular Datasets		
	Iris Species	Pima Indians Diabetes	Banknote
Instance #	150	768	1,372
Train #	135	691	1,234
Test #	15	77	137
Feature #	4	8	4
Class #	3	2	2

In this subsection, we will investigate the effectiveness of RPN in inferring probabilistic dependency relationships using three canonical tabular datasets. The basic statistical information of these datasets are summarized in Table 8. The Iris Species is a multi-class dataset, while the other two involve binary class instead. For all three datasets, we employ 10-fold cross-validation, allocating 90% of the instances to the training set and the remaining 10% to the testing set. To assess the classification performance of all comparative methods, we utilize Accuracy as our quantitative evaluation metric.

In the following sections, we will first present the quantitative results of RPN and other benchmark methods in classifying these tabular datasets. Subsequently, we will elucidate the learned probabilistic dependency relationships among feature and label variables, as inferred by these comparative methods, focusing on the Iris Species dataset as an illustrative example.

7.3.2 The Main Results of RPN on Classic Tabular Data Classification

Table 9 presents a comparative analysis of RPN against several established probabilistic methods, including naive Bayes (utilizing both Multinomial and Gaussian distributions), Bayesian networks, and Markov networks. We evaluate RPN using two distinct architectures: (1) RPN with a naive Laplace probabilistic expansion function, identity reconciliation function, and linear remainder function; and (2) RPN with a combinatorial Gaussian probabilistic expansion function, identity reconciliation function, and linear remainder function. The dataset is partitioned using the 10-fold cross-

Table 9: Classification results of classic tabular data classification with probabilistic models. For each method, we run them with 10 iterations based on the training and testing set partitioned via the 10-fold cross validation. The average and standard deviation of the obtained accuracy scores are reported in the table.

	Classic Machine Learning Tabular Datasets		
	Iris Species	Pima Indians Diabetes	Banknote
Naive Bayes (Multinomial)	9.00×10^{-1} $\pm 1.12 \times 10^{-1}$	6.51×10^{-1} $\pm 4.17 \times 10^{-2}$	6.36×10^{-1} $\pm 4.61 \times 10^{-2}$
Naive Bayes (Gaussian)	9.53×10^{-1} $\pm 4.26 \times 10^{-2}$	7.56×10^{-1} $\pm 4.22 \times 10^{-2}$	8.39×10^{-1} $\pm 4.72 \times 10^{-2}$
Bayesian Network	9.53×10^{-1} $\pm 7.91 \times 10^{-2}$	6.64×10^{-1} $\pm 5.47 \times 10^{-2}$	9.24×10^{-1} $\pm 2.42 \times 10^{-2}$
Markov Network	9.20×10^{-1} $\pm 7.18 \times 10^{-2}$	7.09×10^{-1} $\pm 4.17 \times 10^{-2}$	9.02×10^{-1} $\pm 4.07 \times 10^{-2}$
RPN (Naive Prob. Exp.)	9.73×10^{-1} $\pm 3.26 \times 10^{-2}$	7.74×10^{-1} $\pm 3.91 \times 10^{-2}$	9.77×10^{-1} $\pm 2.27 \times 10^{-2}$
RPN (Comb. Prob. Exp)	9.67×10^{-1} $\pm 4.47 \times 10^{-2}$	7.80×10^{-1} $\pm 3.36 \times 10^{-2}$	9.79×10^{-1} $\pm 2.18 \times 10^{-2}$

validation, and the Accuracy scores for each method across these 10 iterations are reported as “mean \pm std” in the table.

The results indicate that RPN, employing both naive and combinatorial probabilistic expansion functions, consistently outperforms the other methods. Specifically, RPN with naive Laplace expansion and identity reconciliation functions demonstrates superior parameter learning and performance compared to naive Bayes methods. Furthermore, RPN with combinatorial Gaussian expansion and identity reconciliation functions exhibits enhanced capability in learning dependency relationships among feature and label variables, consistently surpassing the performance of Bayesian and Markov networks. These findings substantiate the efficacy of RPN in inferring the probabilistic dependency relationships of the feature variables and classifying the data instances.

7.3.3 Probabilistic Dependency Relationship Visualization

In addition to the quantitative effectiveness evaluations, we also visualize the relationships learned by the these different probabilistic methods on the Iris Species dataset, as illustrated in Figure 17. For the Bayesian network, we employ Hill Climb Search to learn the network structure from the training data, utilizing the Bayesian Information Criterion as the scoring function. For the Markov network, we need to manually define the variable clique and the factorization approach. In this experiment, all feature variables in the dataset are designated as the clique, with the factorization function based on data instance appearance counts. To facilitate this approach, float features are pre-partitioned and categorized into discrete bins. For RPN, we extract the elements with the k largest positive coefficients and k smallest negative coefficients to define the dependency graphs.

As shown in Plots (a) and (c) of Figure 17, the manually crafted variable relationships indicate that “Iris Species” (the label) is determined by all feature variables (“Sepal Length”, “Sepal Width”, “Petal Length”, and “Petal Width”) for both naive Bayes and Markov network models. The Markov network additionally considers relationships among the feature variables. In contrast, Bayesian network and RPN method, based on learned results from training data, extract different dependency relationships. In the Bayesian network, the “Iris Species” label is determined by “Petal Length |

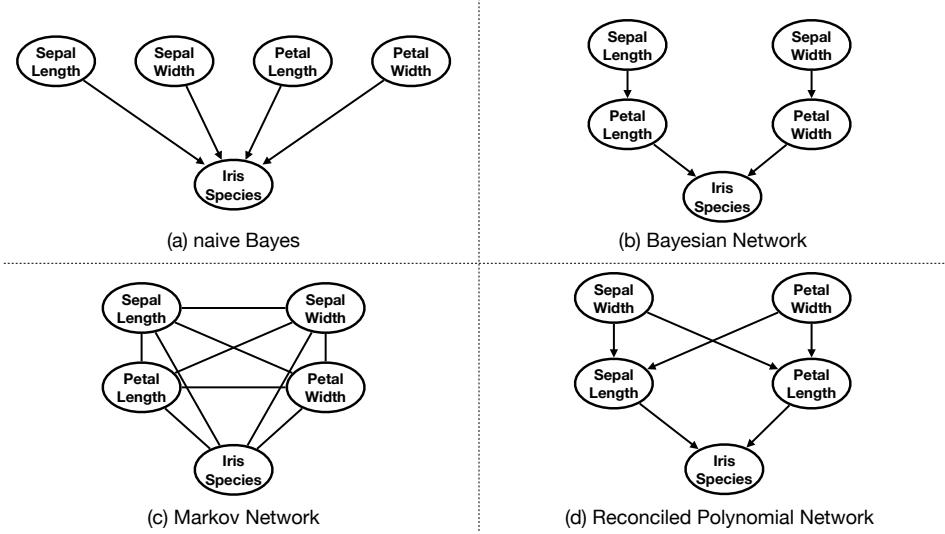


Figure 17: An illustration of dependency relationships inferred by different methods on the Iris Species dataset. In the plots, Sepal Length, Sepal Width, Petal Length and Petal Width denote the feature variables, and Iris Species denotes the label variable of the Iris dataset.

“Sepal Length” and “Petal Width | Sepal Width”. For RPN method, the label is determined by “Petal Length | (Petal Width, Sepal Width)” and “Sepal Length | (Petal Width, Sepal Width)”. We use the notation “ $A | B$ ” to represent the dependency of variable A on B, borrowing from conditional probability notation. The probabilistic dependency relationships learned by RPN method align more closely with the feature variable correlations for different label instances in the Iris dataset, as illustrated in Figure 18.

The visualized plots in Figure 17 reveal that RPN can learn more complex probabilistic dependency relationships among variables, effectively elucidating its learning process and classification results. However, we also observe certain limitations in the current RPN model. The static nature of the current probabilistic expansion functions, which rely on manually defined distributions and pre-provided distribution hyper-parameters, may present challenges when applied to more intricate probabilistic dependency relationships. Addressing this limitation will be a key focus in future work of RPN.

8 Interpretations of the RPN Model Design

In addition to the above empirical evaluations, in this section, we will discuss the interpretations of our model design from various perspectives, encompassing technical machine learning, and biological neuroscience. Through these discussions, we aim to illustrate the motivations and advantages inherent in RPN model design.

8.1 Theoretic Machine Learning Interpretations

As introduced in the previous section, the RPN model is composed of the data expansion function, parameter reconciliation function and the remainder function. Each of these components functions plays a crucial role in shaping the RPN model, striking a balance among model capacity, learning feasibility and performance robustness.

8.1.1 Understanding RPN from VC-Dimension

According to the Vapnik-Chervonenkis theory [75, 7], to evaluate the quality of the RPN model $g(\mathbf{x}|\mathbf{w})$, we compute the introduced errors by comparing $g(\mathbf{x}|\mathbf{w})$ against the underlying function

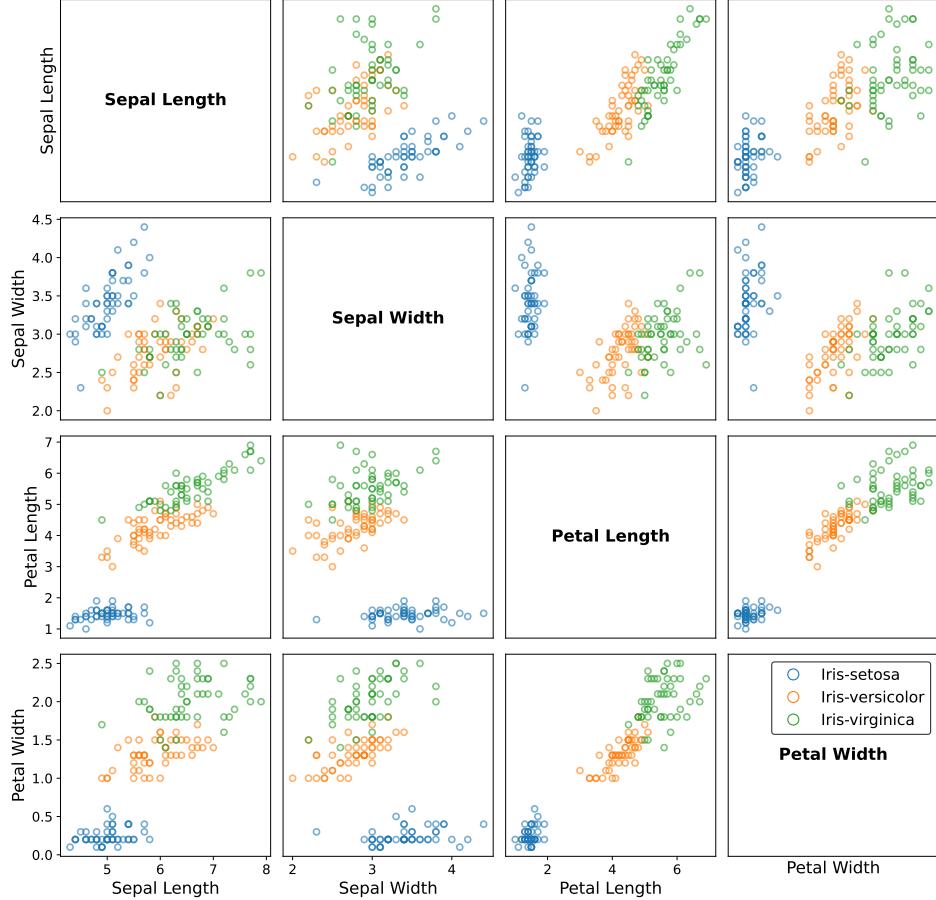


Figure 18: An illustration about the correlations of the pairwise features in the Iris dataset. The scatter dots in different colors denote the data instances of different classes.

$f(\mathbf{x})$ with the input data space $\mathcal{D} \subset \mathbb{R}^m$. This space encompasses all data instances, including those seen in the training set \mathcal{T} and unseen instances in $\mathcal{D} \setminus \mathcal{T}$:

$$\underbrace{\int_{\mathbf{x} \in \mathcal{D}} p(\mathbf{x}) \|g(\mathbf{x}|\mathbf{w}) - f(\mathbf{x})\| d\mathbf{x}}_{\text{overall error } \mathcal{L}} = \underbrace{\int_{\mathbf{x} \in \mathcal{T}} p(\mathbf{x}) \|g(\mathbf{x}|\mathbf{w}) - f(\mathbf{x})\| d\mathbf{x}}_{\text{empirical error } \mathcal{L}_{em}} + \underbrace{\int_{\mathbf{x} \in \mathcal{D} \setminus \mathcal{T}} p(\mathbf{x}) \|g(\mathbf{x}|\mathbf{w}) - f(\mathbf{x})\| d\mathbf{x}}_{\text{expected error } \mathcal{L}_{exp}}, \quad (95)$$

where $p(\mathbf{x})$ denotes the probability density function for drawing instance from the input data space \mathcal{D} and $\|\cdot\|$ denotes a norm measuring the difference between the outputs of $f(\mathbf{x})$ and $g(\mathbf{x}|\mathbf{w})$. As indicated by the equation, the **overall error** \mathcal{L} measuring the difference between the model $g(\mathbf{x}|\mathbf{w})$ and the underlying unknown mapping $f(\mathbf{x})$ consists of two parts: (1) the **empirical error** \mathcal{L}_{em} calculated on the training set \mathcal{T} , and (2) the **expected error** \mathcal{L}_{exp} calculated on unseen data instances from $\mathcal{D} \setminus \mathcal{T}$, both of which are closely related to the VC-dimension of the model.

As illustrated by the curves in Figure 19, as the VC-dimension of a model increases, the model will have a greater capacity (and complexity) and the training error will gradually decrease. Meanwhile, for complex models in higher VC-dimensions, the testing errors of the model on unseen data will first decrease, reaching the harmonious “sweet spot”, and then increase in a U-shape. In addition to the curves we borrowed from the VC-theory, we also add a red curve denoting the learning cost. As the VC-dimension increases and the model becomes more complex, without necessary regulations on the parameters, the learning cost of the model (in terms of time, space, data consumptions in

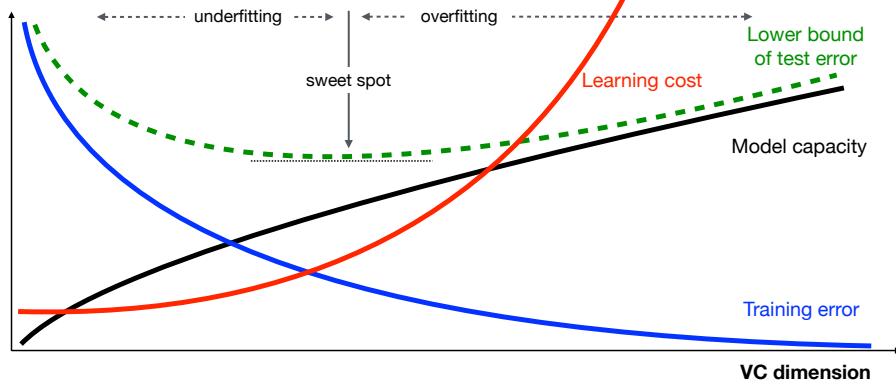


Figure 19: An illustration of VC dimension and model learning: the x-axis represents the VC-dimension, where the blue curve denotes the model training errors, the black curve represents the model capacity, and the green dashed curve signifies the lower bound of test error. The red curve, not previously present in VC theory, represents the added aspect of model learning and operating costs (e.g., required computational hardwares, time and space costs, and energy consumptions).

the training phase, as well as subsequent manual tuning for alignment and safety) will increase dramatically.

For RPN model, the VC-dimension is determined by both the data expansion function and parameter reconciliation function, and potentially the remainder function if it is non-zero. All these component functions collectively influence the model’s complexity and learning capacity. By projecting data instances through the expansion function into higher-dimensional space, we augment the model’s representational complexity. However, instead of directly training the model in this high-dimensional space, the parameter reconciliation function acts as a regulator of model complexity, enabling effective operation on data instances reduced from high-dimensional spaces while mitigating learning costs and guiding testing errors toward an optimal range. Of course, in real practice, the perfect balance between the expansion and reconciliation functions is hard to achieve, which may unnecessarily lead to learning errors and performance degradation, but the remainder function can compensate for potential deficiencies and improve the robustness of RPN.

8.1.2 Understanding RPN from Vector Space Projection

The layers in RPN model form a sequence of data projections across spaces. Initially, data is expanded from the input space to an intermediate higher-dimensional space via the expansion function, and subsequently, it is projected back to a lower-dimensional output space through the inner product with the reconciled parameters. This process enables the RPN model to effectively handle data instances that pose classification challenges in the original input space. By undergoing these projections, separation of the data instances in the output space can be much easier compared to the original input space.

As illustrated in Figure 20, we present a simplified example comprising data instances from two classes: blue circles represent the negative class, while red squares represent the positive class. The data expansion function projects these instances from the two-dimensional input space to an intermediate space, such as a three-dimensional space. Here, positive and negative instances are projected to the top and bottom regions of the newly created x_3 dimension, respectively. Rather than performing classification directly in this intermediate space, which would incur unnecessary high learning costs, our RPN model further projects the data instances back to the output space by excluding the x_2 dimension. This is accomplished through an inner product with the reconciled parameters. Separation of such instances in the output space is considerably easier than in the original input space.

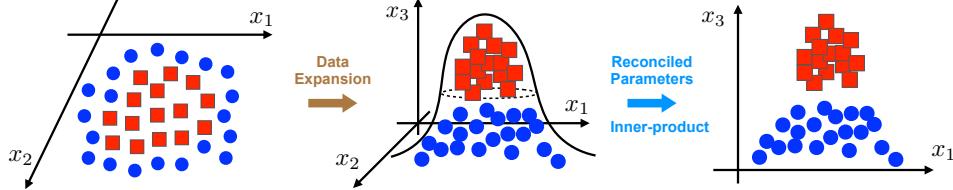


Figure 20: An Interpretations of RPN from vector space projection perspective.

Compared to MLP, the expansion function provides RPN with greater learning capacity to handle more complex tasks. Meanwhile, compared to KAN, the parameter reconciliation function allows RPN to fit the dataset with fewer parameters, thus making it more robust to overfitting and feasible for handling complex learning tasks. RPN and kernel SVM both operate within the expanded space. However, unlike kernel SVM, which directly trains the model with data instances in the expanded space, the reduction back to the output space for learning helps address the “curse-of-dimensionality” and high learning cost issues.

8.2 Biological Neuroscience Interpretations

Compared to the other existing base models mentioned in this paper, the RPN model introduced in this paper provides a closer approximation to current biological nervous systems. The expansion, reconciliation and remainder functions used in the design of the RPN model correspond to the functioning mechanisms of biological neurons.

8.2.1 Biological Neuron Structure and Membrane Potential

Unlike other existing neural models (including MLP and KAN, as well as their derivatives), RPN provides a more accurate modeling of the real-world biological neurons from the neuroscience perspective. Most readers interested in this paper may possess the basic background knowledge about neuroscience that supports the design of artificial neurons and neural network models. However, to help readers understand the RPN model better, we will briefly describe the structure and working mechanisms of biological neurons in this section.

As illustrated in Figure 21, the biological nervous systems for animal creatures (*e.g.*, mammals, birds, fishes, insects and even worms) consists of a large number of inter-connected biological neurons, which are highly specialized for the processing and transmitting cellular signals. These neurons’ cell body includes several important parts as shown in Figure 22 (a), including the *soma*, *nucleus*, *dendrite*, *axon*, *myelin sheath*, *Schwann cell*, *node of Ranvier* and *axon terminal*. The neuron’s axon can branch out and connect to a large number of downstream neurons’ dendrites at sites called *synapses*, whose zoomed-in structure is illustrated in Figure 22 (c). The neuron structure we show in the plot is called the “multipolar neuron”, which possesses a single axon and many dendrites. Besides this, as illustrated in Figure 21, there also exist many other neuron structures, such as “bipolar neuron”, “pseudo-unipolar neuron” and “unipolar neurons”, which will form neurons with different functions. Also neurons in real nervous systems are not arranged layer by layer in an orderly manner; their connections can be somewhat “chaotic”, with many “skip-layer” connections for faster signal transmission.

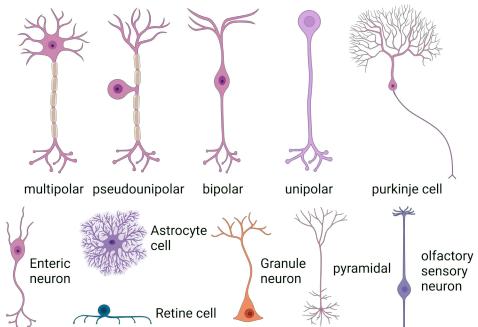


Figure 21: Different neuron structures.

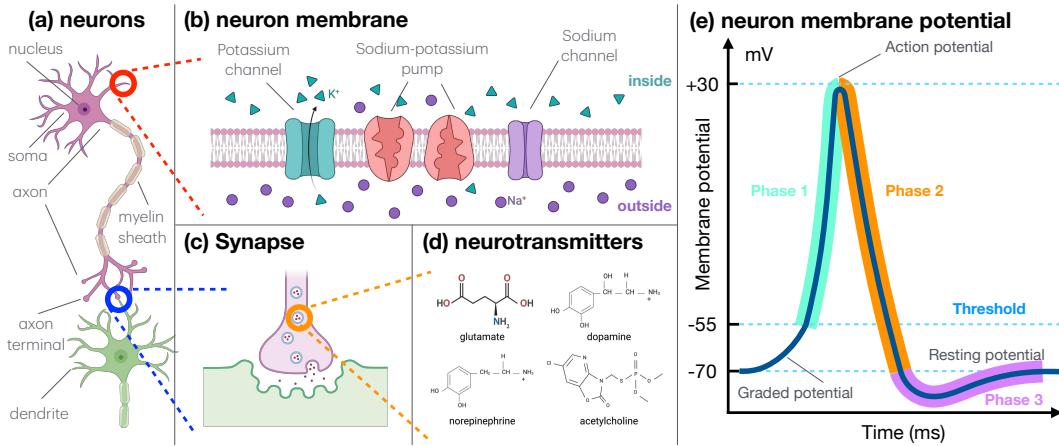


Figure 22: An illustration of connected neuron cells and the neuron cell membrane potential. For the neuron cell, we also provide a zoomed-in view to depict both the neuron membrane with its various channels allowing the ion propagation and the synapse connections releasing neurotransmitters that enable neurotransmissions.

These structures of biological neurons all contribute to the complex functioning of the creatures' nervous systems. Like all other animals' cells, the neuron cells are also enclosed in a plasma membrane, which has the structure of a lipid bilayer with many types of large protein molecules (*e.g.*, the ion channels and ion pumps) embedded in it as shown in Plot (b). Biological neurons are just like a "salty banana", with more sodium ions (purple circles in Plot (b)) outside the cell and more potassium ions (green triangles in Plot (b)) inside. The membrane serves as both an insulator and a diffusion barrier to the movement of ions. Ion pump proteins break down ATP for energy to actively push ions across the membrane and establish concentration gradients across the membrane; while the ion channels allow ions to move across the membrane down those concentration gradients. Ion pumps and ion channels are electrically equivalent to a set of batteries and resistors inserted in the membrane, creating a voltage between the two sides of the membrane, which is called the *membrane potential*. As illustrated in Plot (e), at the resting state, a neuron's membrane potential is sitting in millivolts, ranging from -40mV to -80mV .

The ion channels embedded in the neuron membrane are integral membrane proteins with a pore, which can be either open or closed for ion passage. Ion channels can be classified by how they respond to their environment: (a) ligand-gated channels open once receiving some certain types of chemical neurotransmitters, (b) voltage-dependent channels change their permeability according to the membrane potential, and (c) mechanically gated channels open due to physical distortion of the cell membrane (*e.g.*, sense of touch). When a channel is open, ions permeate through the channel pore down the transmembrane concentration gradient for that particular ion, such as sodium moves in and potassium moves out, causing the neuron membrane to change.

We all know that the "neurons receiving inputs greater than their inherent thresholds will be activated to transmit signals to surround neurons connected to them", but the details of such a process can be more complicated than readers may imagine. As illustrated in Plots (c) and (d), through the synapses, the neuron's axon terminal will release synaptic vesicles, each containing numerous chemical neurotransmitters that act on different connected dendrite branches to open ligand-gated channels. There exist various categories of neurotransmitters, *e.g.*, *amino acids*, *gasotransmitters*, *monoamines*, *trace amines*, *peptides*, *purines*, *catecholamines* and many others. Each category of neurotransmitter delivers different types of cellular signals to other connected neurons. If a neuron's dendrite receives sufficient chemical neurotransmitters from the connected axon terminals via the synapses, it will gradually open more ligand-gated channels, causing the nearby membrane potential to change.

Furthermore, once the membrane potential near the dendrites exceeds a certain threshold, *e.g.*, -55mV , the voltage-dependent sodium channels in the membrane will be abruptly triggered to open, allowing a large influx of sodium ions that rapidly increases the membrane potential near the dendrites from -55mV to $+30\text{mV}$, *i.e.*, phase 1 as shown in Plot (e). This triggers more sequential channels to open, producing a greater electric current across the cell membrane from dendrites to the axon, thereby activating the neuron to transmit signals to other neurons. As the membrane potential reaches its peak, voltage-sensitive potassium channels will open, which will dramatically increase the membrane's potassium permeability and drive the membrane voltage back down to -70mV , *i.e.*, phase 2 in Plot (e). Afterwards, these voltage-dependent sodium/potassium channels shut down, and ion pumps will take the control to gradually restore neurons to their resting potential, as shown in phase 3 of Plot (e).

8.2.2 Understanding RPN from Biological Neuroscience

We provide the detailed descriptions of biological neurons and their working mechanisms above to highlight the following key points to our readers:

- **Membrane Potentials of Neuron:** Biological neurons' membrane potential defines the neuron cell state, which is much more complex than the simple 0/1 (active/inactive) states previously understood. Neurons maintain their membrane potential states through a sophisticated biological mechanism (see Plot (b) in Figure 22). Neurons' membrane potentials typically form a periodic spiking curve (see Plot (e) in Figure 22).
- **Synapses of Neuron:** Biological neuron synapses transmit signals through physical media, with axon terminals releasing neurotransmitters of various categories (see Plots (c) and (d) in Figure 22), which act only on specific ion gates located on the dendrite branches. The combination of neurotransmitters and ligand-gated ion channels determines the effectiveness of signal transmission between neurons.
- **Connections of Neuron:** Biological neurons' structure are very complex and diverse, and each of them may have specific functions depending on their positions in the nervous system (see Figure 21). Biological neuron connections are extensive and often appear "chaotic". Besides being arranged in an orderly layer-by-layer manner, "skip-layer" neuron connections are common in nervous systems for faster and robust signal transmission.

The working mechanisms of biological neurons summarized above actually inspire the design of the RPN model in its current representation form. The data expansion function models the neuron's cell states by projecting them into a high-dimensional space rather than the original space. This approach allows us to model the complex state, including the spiking curve of neurons, moving beyond the simple binary 0/1 state or an integer in the range $[0, 1]$. The parameter expansion function adapts the parameters to accommodate the high-dimensional states of neurons, facilitating signal transmission to connected neurons. This models the fabrication of neurotransmitter-ligand-gated ion channels in forming synapses for signal transmission. Furthermore, the remainder function allows skip-layer data projection, mirroring the extensive connections of biological neurons for faster signal transmission.

9 Intellectual Merits, Limitations and Future Work of RPN

In this section, we will briefly discuss the merits and limitations of the RPN model in various aspects, which may also illustrate some potential future research opportunities on the RPN model.

9.1 Intellectual Merits of RPN

In this paper, we introduce the RPN model that can unify several machine learning and deep learning base models with a canonical representation. The RPN model has transformative impacts on the

current and future academic research and industrial development of machine learning, deep learning and artificial intelligence. We summarize the intellectual merits of RPN from several different perspectives as follows:

Theoretical Merits: The RPN model disentangles data from parameters and formalizes several machine learning and deep learning base model architectures as the inner product of expanded data instances with the reconciled parameters, derived according to the Taylor’s theorem. From the machine learning theory perspective, the data expansion functions are responsible for input data vector projection from one base space to another; and the parameter reconciliation functions will accommodate the parameters dimensions and regularize them to avoid overfitting according to the VC-theory. Meanwhile, from the biological neuroscience perspective, the data expansion functions models the complex states of neurons before, during and after the activations; and the parameter reconciliation functions fabricate the neurotransmitters and their ligand-gated ion channels to compose the synapses. The RPN model architecture makes it much easier to interpret the physical meanings of both the results and the learning process from both theoretic machine learning and biological neuroscience perspectives.

Technical Merits: The RPN model provides a unified representation for both classic machine learning models and the family of deep models under one framework. We use non-deep machine learning models, such as SVM and probabilistic models (including naive Bayes, Bayesian network and Markov network), and deep models, such as MLP and KAN, as examples to illustrate how they can be represented with RPN. Additionally, the multi-layer, multi-head, and multi-channel design of RPN provides significant flexibility for future model design and deployment in various applications. The unified representations in RPN greatly simplify the design and development of future machine learning, deep learning, and artificial intelligence models and systems.

Computational Merits: The disentanglement of data from parameters at each layer in RPN allows their computations to be separately routed to different chips, machines, and cloud platforms, protecting both data privacy and model parameter security. Additionally, RPN’s reliance on inner product computations facilitates easy parallelization across various computational platforms, systems, and hardware. The RPN model can be effectively trained using the conventional error backpropagation. Gradient computation and parameter updates can be performed locally with minimal message passing across different computational facilities, enabling the deployment of RPN in real-world large-scale systems and models with significantly reduced computational resources and energy consumption.

9.2 Limitations and Future Work of RPN

We have observed several limitations with the current RPN model based on our experiences in both framework development and experimental testing. These limitations can also highlight potential future research opportunities and suggest development directions for readers. We summarize these limitations from various perspectives as follows:

Modeling Limitations: While we aim to unify existing machine and deep learning base models with RPN, it’s important to note that there still exist a large number of models we haven’t covered or investigated regarding how to represent them with RPN. Examples include tree family models (*e.g.*, decision tree, gradient boosted trees, and random forest), ensemble learning models (*e.g.*, bagging, boosting, and stacking-based models), unsupervised learning models (*e.g.*, k-means, principal component analysis), and reinforcement learning models (*e.g.*, q-learning, policy gradient methods). Many of these models can potentially be represented with RPN; for instance, decision trees can be interpreted with conditional probabilities, and ensemble learning models can be modeled in RPN with the multi-head multi-channel mechanism. However, we leave the exploration of these models for future research endeavors.

Efficiency Limitations: The data expansion functions in RPN expand data instances to a higher dimension to better approximate the desired target function. Although we can utilize deep architectures to constrain the dimensionality of the expansion space at each layer, the intermediate

representations of data instances may still consume significant computational resources, and deep architectures can introduce higher calculation time overhead. For instance, if we plan to use RPN to build language models with longer context widths, certain expansion functions (*e.g.*, combinatorial expansion and those defined with the Kronecker product) may not be applicable. Furthermore, in the design of data expansion functions, parameter reconciliation functions, and remainder functions, we did not consider any techniques for optimizing performance on high-performance computing (HPC) systems, which could potentially address efficiency issues. We intend to investigate these areas in future work.

Learning Limitations: The training of the RPN model in this paper still relies on the error back-propagation algorithm, which has faced criticism in recent years due to its high computational costs, slow convergence, and sensitivity to hyperparameters and initializations. Additionally, the higher-dimensional expansions may lead to wider outputs for input data instances, potentially causing learning problems such as overfitting and performance robustness issues. However, the disentanglement of parameters from data into different functions at each layer presents an opportunity to explore new model training algorithms, a discussion which is not included in this paper. We consider to explore and design new learning algorithms for RPN as one of the most important directions for future research.

10 Related Work

This section briefly discusses the other existing work related to the RPN model introduced in this paper, including the existing machine learning and deep learning base models, and the techniques utilized for designing the component functions in RPN.

10.1 Machine Learning and Deep Learning Base Models

In this paper, we compare RPN with two main types of machine learning base models, *i.e.*, probabilistic models and support vector machine, and two main types of deep learning base models, *i.e.*, multi-layer perceptron and Kolmogorov-Arnold network. We will briefly introduce their development history and some important papers that made critical technical breakthroughs about them below.

10.1.1 Probabilistic Models

In classic machine learning, lots of models have been proposed based on probability theory. Among them, naive Bayes [36], based on the assumption that the features are conditionally independent given the target class, is one of the most well-known linear probabilistic classifiers designed based on the Bayes' theorem. Different from naive Bayes, Markov network [6] and Bayesian network [59] models the dependency relationships among the variables as an inter-connected variable network. A random field can be said to be a Markov random field if it satisfy the Markov property [67]. Markov network, also known as Markov random field and Markov model, was proposed by Besag [6] to represent the spatial interactions of lattice systems based on the Hammersley-Clifford Theorem. As one of the pioneers of Bayesian networks and probabilistic models, in his book [59], Pearl introduced the networks for plausible inference for probabilistic reasoning in intelligent systems. For calculating and updating the likelihood in the Bayesian network, Pearl proposed the Belief Propagation algorithm in [58]. Expectation-maximization algorithm initially introduced in [15] was used for probabilistic graphical model training afterwards as proposed in [29]. All these aforementioned probabilistic models have comprehensively introduced in the book written by Koller and Friedman [39].

10.1.2 Support Vector Machines

Support vector machine (SVM), as a supervised max-margin model, was mainly developed by Vladimir Vapnik and his colleagues. In [12, 77], Cortes and Vapnik introduced support-vector net-

work as a binary classification model. With the kernel tricks [8], Boser, Guyon and Vapnik proposed to project input vectors that are non-linearly separable into a high-dimensional space, within which the data instances can be separated with a linear decision surface. For the training of SVM with kernel tricks, [8] addressed the optimization problem within the dual space instead to identify the support vectors closest to the decision boundary. Meanwhile, as the training data size increases, the costs of training SVM will increase dramatically. Platt [62] proposed the Sequential Minimal Optimization (SMO) algorithm which breaks the quadratic programming (QP) problem in SVM training into a series of smallest possible QP problems, which made fast SVM training possible. Besides classification tasks, Vapnik, Golowich and Smola also reported the performance of SVM for function approximation, regression estimation and signal processing in [76]. Another follow-up work [5] used support vector machine for the clustering task. All these model and training algorithms related to SVM were implemented by Chang and Lin in the LIBSVM package [10].

10.1.3 Multi-layer Perceptron

Multi-layer perceptron (MLP) denotes the perceptron model [63] with a multi-layered architecture built based on the MP-neurons [53]. To address the non-linear XOR problem proposed by Minsky [54], hidden layers were added to the single-layered perceptron to compose a multi-layered network with the feedforward architecture by Rosenblatt [34]. The deep MLP learned with stochastic gradient descent was proposed by Amari in [1], which introduced a five-layered feedforward network with two learning layers. As to the theoretic foundation, Cybenko introduced the universal approximation theorem in [13] and the multilayer perceptron with hidden layers was demonstrated to be a universal approximator [31]. For the training of deep MLP models, the modern version of back-propagation algorithm designed based on the chain-rule was published by Linnainmaa in his master thesis [49], and was later evaluated with experimental analysis by Rumelhart, Hinton and Williams in [65]. Nowadays, MLP still serves as the base model for most of the current deep learning models.

10.1.4 Kolmogorov-Arnold Network

Unlike MLPs with predefined, static activation functions, the recent Kolmogorov-Arnold network (KAN) model [51] proposes to learn the activation functions attached to neuron-neuron connections. KAN was designed based on the Kolmogorov-Arnold superposition theorem [40, 22] and models multivariate continuous functions as a superposition of the two-argument addition of continuous functions of one variable. Prior to the KAN mode, many other prior works [47, 69, 41] have proposed to build neural networks based on the Kolmogorov-Arnold theorem already. Based on the Kolmogorov-Arnold superposition theorem, [55] investigates the error bounds of deep ReLU networks; [25] studies the expressive power of ReLU deep neural networks in function approximation; and [20] investigates the interpretability of spline-based neural networks. Besides the continuous function approximation, [3] studied the Kolmogorov spline network for image processing. In addition, the Kolmogorov-Arnold superposition theorem has also been demonstrated to be capable to address the curse of dimension when approximating high-dimensional functions [43].

10.2 Component Function Design

In this paper, we introduce a tripartite set of compositional component functions - data expansion, parameter reconciliation, and remainder functions - that serve as the building blocks for the RPN model. These component functions' designing and definitions are closely related to the techniques used for data augmentation, parameter-efficient learning and residual learning. We will briefly introduce the related work of these topics below.

10.2.1 Data Augmentation

In addition to the kernel tricks [8] and Taylor's expansion based machine learning preliminary work introduced in the previous Section 3.4, some data expansion functions introduced in this paper are also inspired by the data augmentation techniques proposed for deep learning model training. Data

augmentation [93] is a frequently used approach for model learning with incomplete or imbalanced data, which creates new data instances via minor modifications on existing data. Some frequently used data augmentation techniques used in current deep models include geometric transformation [42, 84], space transformation [17, 50], noise injection [17] and adversarial instance creation [21]. Frequently used geometric transformation techniques include flipping, rotation, scaling, translation and reflection in the input space, which may improve the models in combating against the overfitting issues [42]. The space transformation and noise injection techniques introduced in [17] performances the data augmentation not in the input space but in a learned feature space instead [50]. Specifically, via adding noise, interpolation and extrapolation, the data augmentation can improve model learning performance. In addition, in [21] and other follow-up works [68, 80, 90], adversarial data augmentation was utilized to improve the model generalization and robustness. A comprehensive survey of related data augmentation techniques used for image and language data and models is has also been provided in [84, 93].

10.2.2 Parameter-Efficient Learning

Many of the parameter reconciliation functions introduced in this paper are actually defined based on the current parameter-efficient fine-tuning (PEFT) techniques proposed for language models. PEFT offers an effective solution by reducing the number of fine-tuning parameters and memory usage while achieving comparable performance to full fine-tuning. Current PEFT methods can be roughly categorized into several types, such as adapter-based methods [32, 48], parameter masking based methods [91, 71], low-rank matrix decomposition based methods [33, 74]. Adapter based fine-tuning methods propose to freeze pre-trained model parameters and introduce new extra trainable parameters for task-specific fine-tuning [32, 48, 60, 64]. Besides adding these new parameters as adapters, researchers also propose to pre-pend these new parameters to the input data as prompts instead, such as [45, 44]. The masking based fine-tuning methods reduce the number of fine-tuned parameters by selecting a subset of pre-trained parameters critical to downstream tasks while discarding unimportant ones. The parameter masking can be applied to either the pre-trained parameters [91, 71] or the delta parameters [2, 83]. Based on the adapters, LoRA (Low-Rank Adaptation) [33] and its derivatives [74, 88, 94] propose to decompose the parameter matrix into the product of low-rank sub-matrices instead. Instead of composing the parameters into low-rank matrix products, [85] proposes to further reduce the parameters with the hypercomplex multiplication of small-sized sub-matrices instead; [28] integrates both LoRA with the hypercomplex multiplication for parameter matrix decomposition. A recent survey [82] provides a more comprehensive introduction about the latest parameter-efficient model learning methods for pre-trained language models.

10.2.3 Residual Learning

The remainder function derived from Taylor’s expansions plays a very similar role in the RPN model as the skip-layer connections used in residual learning. Cross-layer residual connections [70, 26] have been extensively in building current deep learning models, which greatly improves the model stability, robustness and trainability. Prior to the ResNet, Schmidhuber et al. proposed the Highway networks [70] that allow unimpeded information flow across several layers for building deep models with gradient based training algorithms. Inspired by Highway network, He et al. introduced the ResNet [26] for building deep CNN models to achieve about 28% improvement on the CoCo dataset. Several follow-up work [27, 79, 46, 35] investigate the interpretation and theoretic analysis of the ResNet and residual learning. In [27], He et al. analyze the propagation formulation behind the residual building block in both forward and backward propagations; [79] interprets residual networks as a collection of many paths of differing length, where the shortest paths are leveraged during training; [46] interprets ResNet layers as iterative refinement of learned representations; and [35] investigates that residual connections naturally encourage features of residual blocks to move along the negative gradient of loss. In addition to CNN, residual learning and the skip-layer highway connections have become a standard paradigm in deep learning. It has been extensively used in the follow-up deep models, including Transformer [78], BERT [16], Graph-BERT [87], ViT [18], Stable Diffusion [30] and almost all the recent large language models.

11 Conclusion

In this paper, we introduce the Reconciled Polynomial Network (RPN) as a novel base model for deep function learning tasks. By incorporating multi-layers, multi-heads and multi-channels, RPN has a versatile model architecture and attains superior modeling capability for diverse deep function learning tasks on various multi-modality data. What's more, through specific selections of component functions - including data expansion, parameter reconciliation and remainder functions - RPN provides a unifying framework for several influential base models into a canonical representation. To demonstrate the effectiveness of RPN, this paper presents the extensive empirical experiments conducted across numerous benchmark datasets for various deep function learning tasks. The results consistently demonstrate RPN's superior performance compared to other base models.

Compared with other existing machine learning and deep learning base models, RPN presents significant advantages across several critical dimensions, including generalizability, interpretability, and reusability. Without any prior assumptions on data modalities, RPN serves as a general model applicable to multi-modal data from the outset. Furthermore, through the disentanglement of data from parameters using the expansion, reconciliation, and remainder functions, RPN achieves greater interpretability compared to existing base models. Moreover, the component functions designed and learned in RPN is inherently well-suited for reusability and continual learning. In addition to the empirical evaluations via experiments on benchmark datasets, this paper also discusses the interpretations of the RPN model design from both technical machine learning and biological neuroscience perspectives.

To facilitate the adoption, implementation and experimentation of RPN, we have released a comprehensive toolkit named **TINYBIG** in this paper. The **TINYBIG** toolkit provides a rich library of pre-implemented component functions introduced in this paper, which allows researchers to rapidly design, customize, and effectively deploy RPN models across various learning tasks.

Acknowledgments and Disclosure of Funding

This work is partially supported by NSF through grants IIS-1763365 and IIS-2106972.

References

- [1] Shunichi Amari. A theory of adaptive pattern classifiers. *IEEE Transactions on Electronic Computers*, EC-16(3):299–307, 1967. doi: 10.1109/PGEC.1967.264666.
- [2] Alan Ansell, E. Ponti, Anna Korhonen, and Ivan Vulic. Composable sparse fine-tuning for cross-lingual transfer. In *Annual Meeting of the Association for Computational Linguistics*, 2021. URL <https://api.semanticscholar.org/CorpusID:238856900>.
- [3] Information Association, Pierre-Emmanuel Leni, Yohan Fougerolle, and Frederic Truchetet. *The Kolmogorov Spline Network for Image Processing*, pages 54–78. 01 2013. ISBN 9781466639959. doi: 10.4018/978-1-4666-3994-2.ch004.
- [4] David Balduzzi, Brian McWilliams, and Tony Butler-Yeoman. Neural taylor approximations: Convergence and exploration in rectifier networks. *ArXiv*, abs/1611.02345, 2016. URL <https://api.semanticscholar.org/CorpusID:17611960>.
- [5] Asa Ben-Hur, David Horn, Hava T. Siegelmann, and Vladimir Vapnik. Support vector clustering. *J. Mach. Learn. Res.*, 2:125–137, mar 2002. ISSN 1532-4435.
- [6] Julian Besag. Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, 36(2):pp. 192–236, 1974. ISSN 00359246. URL <http://www.jstor.org/stable/2984812>.
- [7] Anselm Blumer, A. Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the vapnik-chervonenkis dimension. *J. ACM*, 36(4):929–965, oct 1989. ISSN 0004-5411. doi: 10.1145/76359.76371. URL <https://doi.org/10.1145/76359.76371>.
- [8] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, page 144–152, New York, NY, USA, 1992. Association for Computing Machinery. ISBN 089791497X. doi: 10.1145/130385.130401. URL <https://doi.org/10.1145/130385.130401>.
- [9] J Douglas Carroll. Functional learning: The learning of continuous functional mappings relating stimulus and response continua. *ETS Research Bulletin Series*, 1963(2):i–144, 1963.
- [10] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- [11] Grigoris G. Chrysos, Stylianos Moschoglou, Giorgos Bouritsas, Jiankang Deng, Yannis Panagakis, and Stefanos Zafeiriou. Deep polynomial neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44:4021–4034, 2020. URL <https://api.semanticscholar.org/CorpusID:219980501>.
- [12] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, sep 1995. ISSN 0885-6125. doi: 10.1023/A:1022627411411. URL <https://doi.org/10.1023/A:1022627411411>.
- [13] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, December 1989. ISSN 0932-4194. doi: 10.1007/BF02551274. URL <http://dx.doi.org/10.1007/BF02551274>.
- [14] Joe Davison. Compacter: Efficient low-rank hypercomplex adapter layers. In *Neural Information Processing Systems*, 2021. URL <https://api.semanticscholar.org/CorpusID:235356070>.

- [15] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B*, 39:1–38, 1977. URL <http://web.mit.edu/6.435/www/Dempster77.pdf>.
- [16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics*, 2019. URL <https://api.semanticscholar.org/CorpusID:52967399>.
- [17] Terrance Devries and Graham W. Taylor. Dataset augmentation in feature space. *ArXiv*, abs/1702.05538, 2017. URL <https://api.semanticscholar.org/CorpusID:15530352>.
- [18] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ArXiv*, abs/2010.11929, 2020. URL <https://api.semanticscholar.org/CorpusID:225039882>.
- [19] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks : the official journal of the International Neural Network Society*, 107:3–11, 2017. URL <https://api.semanticscholar.org/CorpusID:6940861>.
- [20] Daniele Fakhry, Emanuele Fakhry, and Hendrik Speleers. Exsplinet: An interpretable and expressive spline-based neural network. *Neural networks : the official journal of the International Neural Network Society*, 152:332–346, 2022. URL <https://api.semanticscholar.org/CorpusID:248505713>.
- [21] Yaroslav Ganin, E. Ustinova, Hana Ajakan, Pascal Germain, H. Larochelle, François Laviolette, Mario Marchand, and Victor S. Lempitsky. Domain-adversarial training of neural networks. In *Journal of machine learning research*, 2015. URL <https://api.semanticscholar.org/CorpusID:2871880>.
- [22] Alexander B. Givental, Boris A. Khesin, Jerrold E. Marsden, Alexander N. Varchenko, Victor A. Vassiliev, Oleg Ya. Viro, and Vladimir M. Zakalyukin, editors. *On functions of three variables*, pages 5–8. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-642-01742-1. doi: 10.1007/978-3-642-01742-1_2. URL https://doi.org/10.1007/978-3-642-01742-1_2.
- [23] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. <http://www.deeplearningbook.org>.
- [24] David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. *ArXiv*, abs/1609.09106, 2016. URL <https://api.semanticscholar.org/CorpusID:208981547>.
- [25] Juncai He. On the optimal expressive power of relu dnns and its application in approximation with kolmogorov superposition theorem. *ArXiv*, abs/2308.05509, 2023. URL <https://api.semanticscholar.org/CorpusID:260775833>.
- [26] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015. URL <https://api.semanticscholar.org/CorpusID:206594692>.
- [27] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, 2016. URL <https://api.semanticscholar.org/CorpusID:6447277>.

- [28] Xuehai He, Chunyuan Li, Pengchuan Zhang, Jianwei Yang, and Xin Eric Wang. Parameter-efficient model adaptation for vision transformers. In *AAAI Conference on Artificial Intelligence*, 2022. URL <https://api.semanticscholar.org/CorpusID:254247258>.
- [29] David Heckerman, Dan Geiger, and David M. Chickering. Learning bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3):197–243, 1995. doi: 10.1023/A:1022623210503. URL <https://doi.org/10.1023/A:1022623210503>.
- [30] Jonathan Ho, Ajay Jain, and P. Abbeel. Denoising diffusion probabilistic models. *ArXiv*, abs/2006.11239, 2020. URL <https://api.semanticscholar.org/CorpusID:219955663>.
- [31] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. ISSN 0893-6080. doi: [https://doi.org/10.1016/0893-6080\(89\)90020-8](https://doi.org/10.1016/0893-6080(89)90020-8). URL <https://www.sciencedirect.com/science/article/pii/0893608089900208>.
- [32] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. *ArXiv*, abs/1902.00751, 2019. URL <https://api.semanticscholar.org/CorpusID:59599816>.
- [33] J. Edward Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *ArXiv*, abs/2106.09685, 2021. URL <https://api.semanticscholar.org/CorpusID:235458009>.
- [34] A G Ivakhnenko and V G Lapa. *Cybernetics and forecasting techniques*. Modern analytic and computational methods in science and mathematics. North-Holland, New York, NY, 1967. URL <https://cds.cern.ch/record/209675>. Trans. from the Russian, Kiev, Naukova Dumka, 1965.
- [35] Stanislaw Jastrzebski, Devansh Arpit, Nicolas Ballas, Vikas Verma, Tong Che, and Yoshua Bengio. Residual connections encourage iterative inference. *ArXiv*, abs/1710.04773, 2017. URL <https://api.semanticscholar.org/CorpusID:3803599>.
- [36] George H. John and Pat Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, UAI’95, page 338–345, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. ISBN 1558603859.
- [37] Ross Kindermann and Laurie Snell. *Markov random fields and their applications*, volume 1. American Mathematical Society, 1980. ISBN 978-0-8218-5001-5. doi: <http://dx.doi.org/10.1090/conm/001>.
- [38] Kyunghee Koh and David E Meyer. Function learning: induction of continuous stimulus-response relations. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 17(5):811, 1991.
- [39] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009. ISBN 0262013193.
- [40] A. K. Kolmogorov. On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition. *Doklady Akademii Nauk SSSR*, 114:369–373, 1957.
- [41] Mario Köppen. On the training of a kolmogorov network. In José R. Dorronsoro, editor, *Artificial Neural Networks — ICANN 2002*, pages 474–479, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. ISBN 978-3-540-46084-8.

- [42] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS’12, page 1097–1105, Red Hook, NY, USA, 2012. Curran Associates Inc.
- [43] Ming-Jun Lai and Zhaiming Shen. The kolmogorov superposition theorem can break the curse of dimensionality when approximating high dimensional functions. *ArXiv*, abs/2112.09963, 2021. URL <https://api.semanticscholar.org/CorpusID:245334285>.
- [44] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Conference on Empirical Methods in Natural Language Processing*, 2021. URL <https://api.semanticscholar.org/CorpusID:233296808>.
- [45] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, abs/2101.00190, 2021. URL <https://api.semanticscholar.org/CorpusID:230433941>.
- [46] Qianli Liao and Tomaso A. Poggio. Bridging the gaps between residual learning, recurrent neural networks and visual cortex. *ArXiv*, abs/1604.03640, 2016. URL <https://api.semanticscholar.org/CorpusID:16045384>.
- [47] Ji-Nan Lin and Rolf Unbehauen. On the realization of a kolmogorov network. *Neural Comput.*, 5(1):18–20, jan 1993. ISSN 0899-7667. doi: 10.1162/neco.1993.5.1.18. URL <https://doi.org/10.1162/neco.1993.5.1.18>.
- [48] Zhaojiang Lin, Andrea Madotto, and Pascale Fung. Exploring versatile generative language model via parameter-efficient transfer learning. In *Findings*, 2020. URL <https://api.semanticscholar.org/CorpusID:215415943>.
- [49] Seppo Linnainmaa. The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors. *Master’s Thesis (in Finnish), Univ. Helsinki*, 1970.
- [50] Bo Liu, Mandar Dixit, Roland Kwitt, and Nuno Vasconcelos. Feature space transfer for data augmentation. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9090–9098, 2018. URL <https://api.semanticscholar.org/CorpusID:23302696>.
- [51] Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljacic, Thomas Y. Hou, and Max Tegmark. Kan: Kolmogorov-arnold networks. *ArXiv*, abs/2404.19756, 2024. URL <https://api.semanticscholar.org/CorpusID:269457619>.
- [52] Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. In *Annual Meeting of the Association for Computational Linguistics*, 2021. URL <https://api.semanticscholar.org/CorpusID:235309789>.
- [53] Warren Mcculloch and Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:127–147, 1943.
- [54] M. Minsky and S. Papert. *Perceptrons*. MIT Press, Cambridge, MA, 1969.
- [55] Hadrien Montanelli and Haizhao Yang. Error bounds for deep relu networks using the kolmogorov-arnold superposition theorem. *Neural networks : the official journal of the International Neural Network Society*, 129:1–6, 2019. URL <https://api.semanticscholar.org/CorpusID:195750815>.

- [56] Omer Nivron, Raghul Parthipan, and Damon Wischik. Taylorformer: Probabilistic predictions for time series and other processes. *ArXiv*, abs/2305.19141, 2023. URL <https://api.semanticscholar.org/CorpusID:258967520>.
- [57] J. Pearl. Bayesian networks: A model of self-activated memory for evidential reasoning. In *Proc. of Cognitive Science Society (CSS-7)*, 1985.
- [58] Judea Pearl. Reverend bayes on inference engines: a distributed hierarchical approach. In *Proceedings of the Second AAAI Conference on Artificial Intelligence*, AAAI'82, page 133–136. AAAI Press, 1982.
- [59] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988. ISBN 1558604790.
- [60] Jonas Pfeiffer, Andreas Rücklé, Clifton A. Poth, Aishwarya Kamath, Ivan Vulic, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. Adapterhub: A framework for adapting transformers. *ArXiv*, abs/2007.07779, 2020. URL <https://api.semanticscholar.org/CorpusID:220525782>.
- [61] Daniel Pfrommer, Thomas Zhang, Stephen Tu, and N. Matni. Tasil: Taylor series imitation learning. *ArXiv*, abs/2205.14812, 2022. URL <https://api.semanticscholar.org/CorpusID:249191504>.
- [62] J. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines, 1998. URL <http://citeseer.ist.psu.edu/platt98sequential.html>.
- [63] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958. ISSN 0033-295X. doi: 10.1037/h0042519. URL <http://dx.doi.org/10.1037/h0042519>.
- [64] Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. Adapterdrop: On the efficiency of adapters in transformers. In *Conference on Empirical Methods in Natural Language Processing*, 2020. URL <https://api.semanticscholar.org/CorpusID:225040886>.
- [65] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning internal representations by error propagation*, page 318–362. MIT Press, Cambridge, MA, USA, 1986. ISBN 026268053X.
- [66] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986. URL <https://api.semanticscholar.org/CorpusID:205001834>.
- [67] David Sherrington and Scott Kirkpatrick. Solvable model of a spin-glass. *Physical Review Letters*, 35:1792+, 12 1975. doi: 10.1103/PhysRevLett.35.1792.
- [68] Aman Sinha, Hongseok Namkoong, and John C. Duchi. Certifying some distributional robustness with principled adversarial training. *arXiv: Machine Learning*, 2017. URL <https://api.semanticscholar.org/CorpusID:13900194>.
- [69] David A. Sprecher and Sorin Draghici. Space-filling curves and kolmogorov superposition-based neural networks. *Neural Netw.*, 15(1):57–67, jan 2002. ISSN 0893-6080. doi: 10.1016/S0893-6080(01)00107-1. URL [https://doi.org/10.1016/S0893-6080\(01\)00107-1](https://doi.org/10.1016/S0893-6080(01)00107-1).
- [70] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *ArXiv*, abs/1505.00387, 2015. URL <https://api.semanticscholar.org/CorpusID:14786967>.
- [71] Yi-Lin Sung, Varun Nair, and Colin Raffel. Training neural networks with fixed sparse masks. *ArXiv*, abs/2111.09839, 2021. URL <https://api.semanticscholar.org/CorpusID:244345839>.

- [72] Yunhao Tang, Michal Valko, and Rémi Munos. Taylor expansion policy optimization. *ArXiv*, abs/2003.06259, 2020. URL <https://api.semanticscholar.org/CorpusID:212718119>.
- [73] S. M. Udrescu and Max Tegmark. Ai feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6, 2019. URL <https://api.semanticscholar.org/CorpusID:167217655>.
- [74] Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobyzev, and Ali Ghodsi. Dylora: Parameter-efficient tuning of pre-trained models using dynamic search-free low-rank adaptation. *ArXiv*, abs/2210.07558, 2022. URL <https://api.semanticscholar.org/CorpusID:252907428>.
- [75] V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability & Its Applications*, 16(2):264–280, 1971. doi: 10.1137/1116025. URL <https://doi.org/10.1137/1116025>.
- [76] Vladimir Vapnik, Steven E. Golowich, and Alex Smola. Support vector method for function approximation, regression estimation and signal processing. In *Proceedings of the 9th International Conference on Neural Information Processing Systems*, NIPS’96, page 281–287, Cambridge, MA, USA, 1996. MIT Press.
- [77] Vladimir N. Vapnik. The support vector method. In Wulfram Gerstner, Alain Germond, Martin Hasler, and Jean-Daniel Nicoud, editors, *Artificial Neural Networks — ICANN’97*, pages 261–271, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg. ISBN 978-3-540-69620-9.
- [78] Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Neural Information Processing Systems*, 2017. URL <https://api.semanticscholar.org/CorpusID:13756489>.
- [79] Andreas Veit, Michael J. Wilber, and Serge J. Belongie. Residual networks are exponential ensembles of relatively shallow networks. *ArXiv*, abs/1605.06431, 2016. URL <https://api.semanticscholar.org/CorpusID:18359848>.
- [80] Riccardo Volpi, Hongseok Namkoong, Ozan Sener, John C. Duchi, Vittorio Murino, and Silvio Savarese. Generalizing to unseen domains via adversarial data augmentation. *ArXiv*, abs/1805.12018, 2018. URL <https://api.semanticscholar.org/CorpusID:44178122>.
- [81] Pengxu Wei, Ziwei Xie, Guanbin Li, and Liang Lin. Taylor neural network for real-world image super-resolution. *IEEE Transactions on Image Processing*, 32:1942–1951, 2023. doi: 10.1109/TIP.2023.3255107.
- [82] Lingling Xu, Haoran Xie, Si-Zhao Joe Qin, Xiaohui Tao, and Fu Lee Wang. Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment. *ArXiv*, abs/2312.12148, 2023. URL <https://api.semanticscholar.org/CorpusID:266362573>.
- [83] Runxin Xu, Fuli Luo, Zhiyuan Zhang, Chuanqi Tan, Baobao Chang, Songfang Huang, and Fei Huang. Raise a child in large language model: Towards effective and generalizable fine-tuning. *ArXiv*, abs/2109.05687, 2021. URL <https://api.semanticscholar.org/CorpusID:237491053>.
- [84] Suorong Yang, Wei-Ting Xiao, Mengcheng Zhang, Suhan Guo, Jian Zhao, and Shen Furao. Image data augmentation for deep learning: A survey. *ArXiv*, abs/2204.08610, 2022. URL <https://api.semanticscholar.org/CorpusID:248240105>.

- [85] Aston Zhang, Yi Tay, Shuai Zhang, Alvin Chan, Anh Tuan Luu, Siu Cheung Hui, and Jie Fu. Beyond fully-connected layers with quaternions: Parameterization of hypercomplex multiplications with 1/n parameters. *ArXiv*, abs/2102.08597, 2021. URL <https://api.semanticscholar.org/CorpusID:231942691>.
- [86] Jiawei Zhang, Limeng Cui, and Fisher B. Gouza. Reconciled polynomial machine: A unified representation of shallow and deep learning models. *ArXiv*, abs/1805.07507, 2018. URL <https://api.semanticscholar.org/CorpusID:29158959>.
- [87] Jiawei Zhang, Haopeng Zhang, Li Sun, and Congying Xia. Graph-bert: Only attention is needed for learning graph representations. *ArXiv*, abs/2001.05140, 2020. URL <https://api.semanticscholar.org/CorpusID:210698881>.
- [88] Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning. 2023. URL <https://api.semanticscholar.org/CorpusID:266435293>.
- [89] Hongjue Zhao, Yizhuo Chen, Dachun Sun, Yingdong Hu, Kaizhao Liang, Yanbing Mao, Lui Sha, and Huajie Shao. Taylornet: A taylor-driven generic neural architecture. 2023. URL <https://openreview.net/forum?id=tDNGHd0Qmz0>.
- [90] Long Zhao, Ting Liu, Xi Peng, and Dimitris N. Metaxas. Maximum-entropy adversarial data augmentation for improved generalization and robustness. *ArXiv*, abs/2010.08001, 2020. URL <https://api.semanticscholar.org/CorpusID:223953563>.
- [91] Mengjie Zhao, Tao Lin, Martin Jaggi, and Hinrich Schütze. Masking as an efficient alternative to finetuning for pretrained language models. In *Conference on Empirical Methods in Natural Language Processing*, 2020. URL <https://api.semanticscholar.org/CorpusID:216553665>.
- [92] Man Zhou, Zeyu Xiao, Xueyang Fu, Aiping Liu, Gang Yang, and Zhiwei Xiong. Unfolding taylor’s approximations for image restoration. *ArXiv*, abs/2109.03442, 2021. URL <https://api.semanticscholar.org/CorpusID:237440410>.
- [93] Yue Zhou, Chenlu Guo, Xu Wang, Yi Chang, and Yuan Wu. A survey on data augmentation in large model era. *ArXiv*, abs/2401.15422, 2024. URL <https://api.semanticscholar.org/CorpusID:267311830>.
- [94] Bojia Zi, Xianbiao Qi, Lingzhi Wang, Jianan Wang, Kam-Fai Wong, and Lei Zhang. Delta-lora: Fine-tuning high-rank parameters with the delta of low-rank matrices. *ArXiv*, abs/2309.02411, 2023. URL <https://api.semanticscholar.org/CorpusID:261556652>.

A Appendix

A.1 Performance of RPN Variants for Continuous Function Fitting and Approximation

Descriptions: In the following Tables 10-12, we provide the experimental results of RPN on the elementary function, composite function and Feynman function datasets, respectively. Several different variants of RPN are compared:

- **RPN-Exd-LoRR-Linear:** The RPN model with extended expansion function, involving Taylor’s expansion function and Bspline expansion function, low-rank reconciliation function and linear remainder function.
- **RPN-Exd-LoRR-Zero:** The RPN model with similar component function as RPN-Exd-LoRR-Linear, but replace the remainder with zero remainder function instead.
- **RPN-Nstd-LoRR-Linear:** The RPN model with nested expansion function, involving Taylor’s expansion function and Bspline expansion function, low-rank reconciliation function and linear remainder function. The nested expansion will dramatically increase the intermediate expansion space dimensions, it may involve more learnable parameters in the model.
- **RPN-Nstd-LoRR-Zero:** The RPN model with similar component function as RPN-Nstd-LoRR-Linear, but replace the remainder with zero remainder function instead.

Table 10: Elementary Function Fitting with RPN Variants.

Eq.	RPN (Extended Expansion-LoRR-Linear)	RPN (Extended Expansion-LoRR-Zero)	RPN (Nested Expansion-LoRR-Linear)	RPN (Nested Expansion-LoRR-Zero)
E.0	$1.92 \times 10^{-8} \pm 1.30 \times 10^{-8}$	$8.40 \times 10^{-8} \pm 1.12 \times 10^{-7}$	$1.35 \times 10^{-8} \pm 6.69 \times 10^{-9}$	$1.93 \times 10^{-8} \pm 1.15 \times 10^{-8}$
E.1	$6.42 \times 10^{-2} \pm 3.08 \times 10^{-2}$	$8.67 \times 10^{-2} \pm 8.22 \times 10^{-2}$	$1.00 \times 10^{-1} \pm 1.22 \times 10^{-1}$	$1.03 \times 10^{-1} \pm 1.40 \times 10^{-1}$
E.2	$4.08 \times 10^{-7} \pm 4.65 \times 10^{-7}$	$2.56 \times 10^{-7} \pm 1.48 \times 10^{-7}$	$9.90 \times 10^{-8} \pm 5.91 \times 10^{-8}$	$5.06 \times 10^{-8} \pm 3.53 \times 10^{-8}$
E.3	$5.51 \times 10^{-7} \pm 4.69 \times 10^{-7}$	$3.81 \times 10^{-6} \pm 6.56 \times 10^{-6}$	$1.08 \times 10^{-6} \pm 1.53 \times 10^{-6}$	$1.26 \times 10^{-6} \pm 1.37 \times 10^{-6}$
E.4	$4.64 \times 10^{-5} \pm 4.69 \times 10^{-5}$	$7.05 \times 10^{-5} \pm 3.12 \times 10^{-5}$	$1.87 \times 10^{-5} \pm 2.34 \times 10^{-5}$	$1.80 \times 10^{-5} \pm 2.37 \times 10^{-5}$
E.5	$7.28 \times 10^{-8} \pm 6.79 \times 10^{-8}$	$4.95 \times 10^{-8} \pm 4.39 \times 10^{-8}$	$3.99 \times 10^{-9} \pm 1.85 \times 10^{-9}$	$5.67 \times 10^{-9} \pm 3.20 \times 10^{-9}$
E.6	$4.32 \times 10^{-8} \pm 2.43 \times 10^{-8}$	$1.25 \times 10^{-7} \pm 1.34 \times 10^{-7}$	$1.41 \times 10^{-8} \pm 8.39 \times 10^{-9}$	$5.93 \times 10^{-9} \pm 3.98 \times 10^{-9}$
E.7	$9.25 \times 10^{-8} \pm 5.53 \times 10^{-8}$	$6.02 \times 10^{-8} \pm 4.46 \times 10^{-8}$	$6.88 \times 10^{-9} \pm 4.46 \times 10^{-9}$	$1.67 \times 10^{-8} \pm 2.32 \times 10^{-8}$
E.8	$2.05 \times 10^{-6} \pm 2.22 \times 10^{-6}$	$1.65 \times 10^{-6} \pm 2.06 \times 10^{-6}$	$1.26 \times 10^{-6} \pm 6.89 \times 10^{-7}$	$5.16 \times 10^{-7} \pm 2.97 \times 10^{-7}$
E.9	$3.57 \times 10^{-6} \pm 3.84 \times 10^{-6}$	$4.49 \times 10^{-5} \pm 5.15 \times 10^{-5}$	$1.48 \times 10^{-6} \pm 8.78 \times 10^{-7}$	$3.73 \times 10^{-7} \pm 2.32 \times 10^{-7}$
E.10	$8.42 \times 10^{-9} \pm 7.09 \times 10^{-9}$	$4.86 \times 10^{-9} \pm 2.17 \times 10^{-9}$	$1.13 \times 10^{-9} \pm 2.69 \times 10^{-10}$	$3.02 \times 10^{-9} \pm 1.65 \times 10^{-9}$
E.11	$1.56 \times 10^{-7} \pm 9.03 \times 10^{-8}$	$1.99 \times 10^{-7} \pm 1.65 \times 10^{-7}$	$8.83 \times 10^{-8} \pm 7.14 \times 10^{-8}$	$5.43 \times 10^{-8} \pm 5.25 \times 10^{-8}$
E.12	$5.28 \times 10^{-7} \pm 6.84 \times 10^{-7}$	$7.00 \times 10^{-7} \pm 4.65 \times 10^{-7}$	$1.11 \times 10^{-7} \pm 9.78 \times 10^{-8}$	$4.47 \times 10^{-8} \pm 3.42 \times 10^{-8}$
E.13	$2.61 \times 10^{-8} \pm 1.79 \times 10^{-8}$	$5.73 \times 10^{-8} \pm 6.20 \times 10^{-8}$	$4.23 \times 10^{-9} \pm 2.15 \times 10^{-9}$	$4.55 \times 10^{-9} \pm 2.73 \times 10^{-9}$
E.14	$4.94 \times 10^{-9} \pm 5.74 \times 10^{-9}$	$5.13 \times 10^{-9} \pm 3.94 \times 10^{-9}$	$1.32 \times 10^{-9} \pm 3.41 \times 10^{-10}$	$4.34 \times 10^{-9} \pm 2.82 \times 10^{-9}$
E.15	$4.85 \times 10^{-6} \pm 2.36 \times 10^{-6}$	$8.94 \times 10^{-6} \pm 4.55 \times 10^{-6}$	$4.75 \times 10^{-7} \pm 2.04 \times 10^{-7}$	$5.38 \times 10^{-7} \pm 5.62 \times 10^{-7}$
E.16	$2.61 \times 10^{-5} \pm 2.20 \times 10^{-5}$	$3.00 \times 10^{-5} \pm 3.47 \times 10^{-5}$	$4.21 \times 10^{-5} \pm 4.52 \times 10^{-5}$	$2.51 \times 10^{-5} \pm 3.04 \times 10^{-5}$

Table 11: Composite Function Fitting with RPN Variants.

Eq.	RPN (Extended Expansion-LoRR-Linear)	RPN (Extended Expansion-LoRR-Zero)	RPN (Nested Expansion-LoRR-Linear)	RPN (Nested Expansion-LoRR-Zero)
C.0	$1.18 \times 10^{-1} \pm 1.20 \times 10^{-1}$	$3.25 \times 10^{-2} \pm 3.34 \times 10^{-2}$	$6.45 \times 10^{-3} \pm 3.49 \times 10^{-3}$	$2.73 \times 10^{-3} \pm 3.32 \times 10^{-3}$
C.1	$3.71 \times 10^{-7} \pm 3.64 \times 10^{-7}$	$6.73 \times 10^{-7} \pm 3.04 \times 10^{-7}$	$1.76 \times 10^{-7} \pm 1.20 \times 10^{-7}$	$1.84 \times 10^{-7} \pm 1.19 \times 10^{-7}$
C.2	$1.23 \times 10^{-6} \pm 8.21 \times 10^{-7}$	$1.33 \times 10^{-6} \pm 1.29 \times 10^{-6}$	$2.40 \times 10^{-6} \pm 3.59 \times 10^{-6}$	$1.54 \times 10^{-6} \pm 9.85 \times 10^{-7}$
C.3	$9.36 \times 10^{-5} \pm 7.99 \times 10^{-5}$	$1.17 \times 10^{-4} \pm 1.18 \times 10^{-4}$	$1.03 \times 10^{-5} \pm 1.57 \times 10^{-5}$	$3.86 \times 10^{-6} \pm 2.99 \times 10^{-6}$
C.4	$4.40 \times 10^{-7} \pm 3.09 \times 10^{-7}$	$4.40 \times 10^{-7} \pm 4.59 \times 10^{-7}$	$1.56 \times 10^{-7} \pm 1.02 \times 10^{-7}$	$1.09 \times 10^{-7} \pm 6.68 \times 10^{-8}$
C.5	$4.95 \times 10^{-6} \pm 5.14 \times 10^{-6}$	$1.48 \times 10^{-6} \pm 1.69 \times 10^{-6}$	$3.70 \times 10^{-7} \pm 3.73 \times 10^{-7}$	$7.24 \times 10^{-8} \pm 4.28 \times 10^{-8}$
C.6	$6.53 \times 10^{-2} \pm 5.58 \times 10^{-2}$	$3.72 \times 10^{-2} \pm 3.95 \times 10^{-2}$	$6.25 \times 10^{-3} \pm 5.42 \times 10^{-3}$	$2.25 \times 10^{-3} \pm 2.49 \times 10^{-3}$
C.7	$6.67 \times 10^{-7} \pm 3.73 \times 10^{-7}$	$4.97 \times 10^{-7} \pm 5.01 \times 10^{-7}$	$7.72 \times 10^{-8} \pm 7.22 \times 10^{-8}$	$2.94 \times 10^{-8} \pm 1.37 \times 10^{-8}$
C.8	$3.78 \times 10^{-7} \pm 2.98 \times 10^{-7}$	$5.70 \times 10^{-8} \pm 1.98 \times 10^{-8}$	$1.40 \times 10^{-8} \pm 6.12 \times 10^{-9}$	$1.64 \times 10^{-8} \pm 9.49 \times 10^{-9}$
C.9	$8.77 \times 10^{-5} \pm 5.73 \times 10^{-5}$	$1.88 \times 10^{-4} \pm 2.20 \times 10^{-4}$	$5.59 \times 10^{-6} \pm 3.63 \times 10^{-6}$	$4.55 \times 10^{-6} \pm 3.99 \times 10^{-6}$
C.10	$4.68 \times 10^{-7} \pm 3.11 \times 10^{-7}$	$3.40 \times 10^{-7} \pm 4.77 \times 10^{-7}$	$5.30 \times 10^{-8} \pm 3.02 \times 10^{-8}$	$3.58 \times 10^{-8} \pm 1.18 \times 10^{-8}$
C.11	$9.17 \times 10^{-7} \pm 9.75 \times 10^{-7}$	$3.57 \times 10^{-7} \pm 2.56 \times 10^{-7}$	$2.02 \times 10^{-7} \pm 1.09 \times 10^{-7}$	$4.87 \times 10^{-7} \pm 8.71 \times 10^{-7}$
C.12	$3.63 \times 10^{-5} \pm 2.25 \times 10^{-5}$	$3.74 \times 10^{-5} \pm 1.88 \times 10^{-5}$	$4.11 \times 10^{-5} \pm 4.54 \times 10^{-5}$	$7.17 \times 10^{-5} \pm 8.87 \times 10^{-5}$
C.13	$2.83 \times 10^{-6} \pm 2.03 \times 10^{-6}$	$6.97 \times 10^{-7} \pm 2.29 \times 10^{-7}$	$2.45 \times 10^{-7} \pm 1.35 \times 10^{-7}$	$8.56 \times 10^{-8} \pm 3.81 \times 10^{-8}$
C.14	$3.45 \times 10^{-7} \pm 4.58 \times 10^{-7}$	$3.50 \times 10^{-7} \pm 3.89 \times 10^{-7}$	$2.22 \times 10^{-8} \pm 1.04 \times 10^{-8}$	$1.69 \times 10^{-8} \pm 2.42 \times 10^{-8}$
C.15	$2.56 \times 10^3 \pm 4.35 \times 10^3$	$2.47 \times 10^3 \pm 4.39 \times 10^3$	$1.20 \times 10^4 \pm 2.18 \times 10^4$	$1.17 \times 10^4 \pm 2.11 \times 10^4$
C.16	$3.42 \times 10^{-8} \pm 2.79 \times 10^{-8}$	$1.74 \times 10^{-9} \pm 1.09 \times 10^{-9}$	$2.10 \times 10^{-9} \pm 1.58 \times 10^{-9}$	$1.48 \times 10^{-9} \pm 1.32 \times 10^{-9}$

Table 12: Feynman Function Fitting with RPN Variants.

Eq.	RPN (Extended Expansion-LoRR-Linear)	RPN (Extended Expansion-LoRR-Zero)	RPN (Nested Expansion-LoRR-Linear)	RPN (Nested Expansion-LoRR-Zero)
F.0	$1.63 \times 10^{-6} \pm 1.31 \times 10^{-6}$	$8.48 \times 10^{-5} \pm 6.93 \times 10^{-5}$	$5.45 \times 10^{-7} \pm 5.88 \times 10^{-7}$	$1.41 \times 10^{-4} \pm 4.19 \times 10^{-6}$
F.1	$1.60 \times 10^{-5} \pm 1.25 \times 10^{-5}$	$5.02 \times 10^{-5} \pm 4.33 \times 10^{-6}$	$3.97 \times 10^{-5} \pm 4.80 \times 10^{-5}$	$7.01 \times 10^{-4} \pm 2.40 \times 10^{-5}$
F.2	$6.92 \times 10^{-5} \pm 2.42 \times 10^{-5}$	$6.59 \times 10^{-5} \pm 1.01 \times 10^{-5}$	$9.99 \times 10^{-5} \pm 1.32 \times 10^{-5}$	$3.29 \times 10^{-3} \pm 1.69 \times 10^{-4}$
F.3	$1.36 \times 10^1 \pm 8.35 \times 10^0$	$5.02 \times 10^1 \pm 1.31 \times 10^1$	$1.20 \times 10^2 \pm 9.14 \times 10^1$	$4.76 \times 10^2 \pm 4.21 \times 10^1$
F.4	$8.47 \times 10^{-1} \pm 1.21 \times 10^{-1}$	$5.66 \times 10^0 \pm 9.33 \times 10^{-1}$	$2.00 \times 10^1 \pm 7.81 \times 10^0$	$3.67 \times 10^1 \pm 5.25 \times 10^0$
F.5	$4.94 \times 10^{-3} \pm 1.57 \times 10^{-3}$	$7.72 \times 10^{-2} \pm 1.29 \times 10^{-1}$	$1.28 \times 10^{-2} \pm 6.23 \times 10^{-3}$	$2.20 \times 10^0 \pm 2.36 \times 10^{-2}$
F.6	$8.68 \times 10^{-4} \pm 3.84 \times 10^{-4}$	$3.34 \times 10^{-3} \pm 3.00 \times 10^{-4}$	$2.37 \times 10^{-2} \pm 2.07 \times 10^{-2}$	$4.16 \times 10^{-1} \pm 1.81 \times 10^{-2}$
F.7	$9.92 \times 10^{-4} \pm 4.82 \times 10^{-4}$	$2.47 \times 10^{-3} \pm 2.05 \times 10^{-4}$	$2.62 \times 10^{-2} \pm 1.95 \times 10^{-2}$	$3.32 \times 10^{-1} \pm 9.83 \times 10^{-3}$
F.8	$6.59 \times 10^{-5} \pm 5.13 \times 10^{-5}$	$2.75 \times 10^{-2} \pm 1.90 \times 10^{-2}$	$4.91 \times 10^{-5} \pm 7.83 \times 10^{-5}$	$1.42 \times 10^{-1} \pm 2.49 \times 10^{-3}$
F.9	$4.98 \times 10^{-4} \pm 6.95 \times 10^{-4}$	$1.27 \times 10^{-2} \pm 2.49 \times 10^{-2}$	$4.83 \times 10^{-4} \pm 1.78 \times 10^{-4}$	$3.92 \times 10^{-2} \pm 1.99 \times 10^{-3}$
F.10	$7.27 \times 10^{-2} \pm 7.00 \times 10^{-2}$	$7.13 \times 10^{-1} \pm 2.95 \times 10^{-2}$	$1.23 \times 10^0 \pm 5.38 \times 10^{-1}$	$2.95 \times 10^0 \pm 2.31 \times 10^{-1}$
F.11	$1.58 \times 10^0 \pm 8.47 \times 10^{-1}$	$2.61 \times 10^0 \pm 9.01 \times 10^{-1}$	$2.88 \times 10^0 \pm 4.33 \times 10^{-1}$	$4.33 \times 10^0 \pm 3.73 \times 10^{-1}$
F.12	$6.83 \times 10^{-6} \pm 2.53 \times 10^{-6}$	$1.08 \times 10^{-5} \pm 6.83 \times 10^{-6}$	$7.39 \times 10^{-6} \pm 6.04 \times 10^{-6}$	$5.45 \times 10^{-3} \pm 6.94 \times 10^{-4}$
F.13	$3.89 \times 10^{-2} \pm 3.06 \times 10^{-2}$	$4.56 \times 10^{-1} \pm 3.37 \times 10^{-1}$	$6.00 \times 10^{-1} \pm 4.57 \times 10^{-1}$	$4.12 \times 10^0 \pm 2.04 \times 10^{-2}$
F.14	$2.13 \times 10^{-3} \pm 4.51 \times 10^{-4}$	$3.08 \times 10^{-3} \pm 6.18 \times 10^{-4}$	$4.08 \times 10^{-2} \pm 2.46 \times 10^{-2}$	$1.95 \times 10^{-1} \pm 1.57 \times 10^{-2}$
F.15	$2.99 \times 10^0 \pm 5.68 \times 10^{-1}$	$1.80 \times 10^1 \pm 1.06 \times 10^0$	$9.47 \times 10^1 \pm 4.85 \times 10^1$	$2.25 \times 10^2 \pm 1.42 \times 10^1$
F.16	$1.03 \times 10^0 \pm 7.47 \times 10^{-1}$	$1.66 \times 10^0 \pm 5.41 \times 10^{-1}$	$9.40 \times 10^{-1} \pm 6.44 \times 10^{-1}$	$2.33 \times 10^0 \pm 2.13 \times 10^{-1}$
F.17	$6.98 \times 10^{-1} \pm 3.27 \times 10^{-1}$	$3.71 \times 10^0 \pm 8.11 \times 10^{-1}$	$1.39 \times 10^1 \pm 9.53 \times 10^0$	$4.18 \times 10^1 \pm 6.45 \times 10^0$
F.18	$9.29 \times 10^{-5} \pm 3.06 \times 10^{-5}$	$3.20 \times 10^{-4} \pm 1.19 \times 10^{-4}$	$4.08 \times 10^{-3} \pm 4.96 \times 10^{-3}$	$1.20 \times 10^{-2} \pm 2.10 \times 10^{-3}$
F.19	$2.85 \times 10^{-2} \pm 3.35 \times 10^{-3}$	$1.42 \times 10^{-1} \pm 5.50 \times 10^{-3}$	$4.10 \times 10^{-2} \pm 6.58 \times 10^{-3}$	$5.06 \times 10^{-1} \pm 9.98 \times 10^{-2}$
F.20	$9.93 \times 10^{-5} \pm 2.70 \times 10^{-5}$	$1.18 \times 10^{-4} \pm 2.03 \times 10^{-5}$	$6.98 \times 10^{-5} \pm 3.37 \times 10^{-5}$	$8.50 \times 10^{-5} \pm 8.81 \times 10^{-6}$
F.21	$1.59 \times 10^{-4} \pm 1.08 \times 10^{-4}$	$3.90 \times 10^{-4} \pm 5.99 \times 10^{-5}$	$2.43 \times 10^{-4} \pm 2.52 \times 10^{-4}$	$1.37 \times 10^{-2} \pm 7.85 \times 10^{-4}$
F.22	$4.89 \times 10^{-3} \pm 2.59 \times 10^{-3}$	$6.61 \times 10^{-3} \pm 1.97 \times 10^{-3}$	$2.18 \times 10^{-2} \pm 1.05 \times 10^{-2}$	$1.89 \times 10^{-1} \pm 1.87 \times 10^{-2}$
F.23	$1.24 \times 10^{-1} \pm 1.02 \times 10^{-1}$	$3.98 \times 10^{-1} \pm 7.39 \times 10^{-2}$	$1.11 \times 10^0 \pm 7.73 \times 10^{-1}$	$4.58 \times 10^1 \pm 2.09 \times 10^0$
F.24	$7.10 \times 10^0 \pm 2.92 \times 10^0$	$1.17 \times 10^1 \pm 7.65 \times 10^{-1}$	$1.02 \times 10^2 \pm 4.24 \times 10^1$	$1.31 \times 10^2 \pm 3.35 \times 10^0$
F.25	$3.08 \times 10^{-2} \pm 9.13 \times 10^{-3}$	$5.17 \times 10^{-2} \pm 1.40 \times 10^{-3}$	$4.99 \times 10^0 \pm 6.20 \times 10^0$	$2.45 \times 10^1 \pm 1.98 \times 10^0$
F.26	$2.66 \times 10^{-1} \pm 1.64 \times 10^{-1}$	$6.78 \times 10^0 \pm 2.06 \times 10^0$	$8.10 \times 10^{-1} \pm 3.09 \times 10^{-1}$	$1.33 \times 10^1 \pm 5.73 \times 10^{-1}$

A.2 Ablation Studies of RPN on MNIST Dataset

Descriptions: In this section, we provide the performance of RPN with different component function combinations on the MNIST dataset. We enumerate different combinations of the expansion, reconciliation and remainder functions studied in the previous Figures 13-14 in Section 7.2. For each of these combinations, we provide the testing accuracy, model parameter number and time cost of training RPN. In addition, since RPN allows the optional pre-processing and post-processing functions to the expansion function, we also investigate the impacts of such optional functions in the table, where the layer-norm is analyzed as the default function.

Specifically, the following Tables 13-33 will be organized according to the data expansion functions used in composing RPN. Each table presents the results learned for each individual expansion function combined with different reconciliation and remainder functions. The reported results in these tables have also been summarized in the previous Figures 13-14 already.

Table 13: Performance analysis of RPN with the identity expansion function and different reconciliation and remainder functions on the MNIST dataset.

Expansion	Reconciliation	Remainder	Parameter#	Pre-Layer-Norm	Post-Layer-Norm	Time Cost	Accuracy
identity	identity	zero	50816	False	False	1.44×10^3	9.27×10^{-1}
				True	False	1.37×10^3	9.30×10^{-1}
				False	True	1.39×10^3	9.30×10^{-1}
				True	True	9.19×10^2	9.30×10^{-1}
		linear	101632	False	False	7.66×10^2	9.77×10^{-1}
				True	False	7.72×10^2	9.76×10^{-1}
				False	True	7.76×10^2	9.76×10^{-1}
				True	True	8.14×10^2	9.77×10^{-1}
	masking	zero	50816	False	False	1.04×10^3	9.27×10^{-1}
				True	False	1.03×10^3	9.30×10^{-1}
				False	True	1.05×10^3	9.30×10^{-1}
				True	True	1.04×10^3	9.30×10^{-1}
		linear	101632	False	False	9.21×10^2	9.79×10^{-1}
				True	False	9.25×10^2	9.78×10^{-1}
				False	True	9.28×10^2	9.78×10^{-1}
				True	True	9.14×10^2	9.78×10^{-1}
	duplicated-padding	zero	12704	False	False	1.04×10^3	4.77×10^{-1}
				True	False	1.04×10^3	4.78×10^{-1}
				False	True	1.04×10^3	4.78×10^{-1}
				True	True	8.92×10^2	4.78×10^{-1}
		linear	63520	False	False	9.12×10^2	9.73×10^{-1}
				True	False	9.14×10^2	9.73×10^{-1}
				False	True	9.16×10^2	9.73×10^{-1}
				True	True	8.57×10^2	9.74×10^{-1}
	lorr	zero	1844	False	False	8.85×10^2	6.63×10^{-1}
				True	False	8.89×10^2	6.45×10^{-1}
				False	True	8.94×10^2	6.45×10^{-1}
				True	True	8.98×10^2	6.41×10^{-1}
		linear	52660	False	False	8.07×10^2	9.78×10^{-1}
				True	False	8.13×10^2	9.77×10^{-1}
				False	True	8.18×10^2	9.77×10^{-1}
				True	True	8.16×10^2	9.78×10^{-1}
	hypercomplex	zero	12712	False	False	8.85×10^2	8.61×10^{-1}
				True	False	8.90×10^2	8.60×10^{-1}
				False	True	8.92×10^2	8.60×10^{-1}
				True	True	8.96×10^2	8.61×10^{-1}
		linear	63528	False	False	8.06×10^2	9.78×10^{-1}
				True	False	8.13×10^2	9.78×10^{-1}
				False	True	8.16×10^2	9.78×10^{-1}
				True	True	8.17×10^2	9.78×10^{-1}
	lphm	zero	930	False	False	8.96×10^2	6.37×10^{-1}
				True	False	9.02×10^2	6.68×10^{-1}
				False	True	9.05×10^2	6.68×10^{-1}
				True	True	9.03×10^2	6.67×10^{-1}
		linear	51746	False	False	8.26×10^2	9.77×10^{-1}
				True	False	8.36×10^2	9.77×10^{-1}
				False	True	8.32×10^2	9.77×10^{-1}
				True	True	8.44×10^2	9.78×10^{-1}
	hypernet	zero	512	False	False	9.05×10^2	8.26×10^{-1}
				True	False	1.10×10^3	7.24×10^{-1}
				False	True	1.10×10^3	7.24×10^{-1}
				True	True	1.11×10^3	6.97×10^{-1}
		linear	51328	False	False	9.01×10^2	8.94×10^{-1}
				True	False	1.08×10^3	8.68×10^{-1}
				False	True	1.08×10^3	8.68×10^{-1}
				True	True	1.08×10^3	8.67×10^{-1}

Table 14: Performance analysis of RPN with the reciprocal expansion function and different reconciliation and remainder functions on the MNIST dataset.

Expansion	Reconciliation	Remainder	Parameter#	Pre-Layer-Norm	Post-Layer-Norm	Time Cost	Accuracy
reciprocal	identity	zero	50816	False	False	1.51×10^3	2.51×10^{-1}
				True	False	1.62×10^3	1.73×10^{-1}
				False	True	1.56×10^3	1.24×10^{-1}
				True	True	1.65×10^3	1.30×10^{-1}
		linear	101632	False	False	1.58×10^3	6.59×10^{-1}
				True	False	1.59×10^3	3.09×10^{-1}
				False	True	1.61×10^3	9.63×10^{-1}
				True	True	1.62×10^3	9.61×10^{-1}
	masking	zero	50816	False	False	1.69×10^3	2.07×10^{-1}
				True	False	1.69×10^3	1.51×10^{-1}
				False	True	1.70×10^3	1.11×10^{-1}
				True	True	1.71×10^3	1.19×10^{-1}
		linear	101632	False	False	1.66×10^3	3.74×10^{-1}
				True	False	1.67×10^3	3.86×10^{-1}
				False	True	1.68×10^3	9.65×10^{-1}
				True	True	1.70×10^3	9.70×10^{-1}
	duplicated-padding	zero	12704	False	False	1.68×10^3	2.11×10^{-1}
				True	False	1.72×10^3	1.51×10^{-1}
				False	True	1.70×10^3	1.28×10^{-1}
				True	True	1.68×10^3	1.35×10^{-1}
		linear	63520	False	False	1.65×10^3	3.37×10^{-1}
				True	False	1.70×10^3	1.50×10^{-1}
				False	True	1.68×10^3	7.26×10^{-1}
				True	True	1.68×10^3	9.68×10^{-1}
	lorr	zero	1844	False	False	1.64×10^3	1.56×10^{-1}
				True	False	1.65×10^3	1.34×10^{-1}
				False	True	1.66×10^3	1.26×10^{-1}
				True	True	1.67×10^3	1.24×10^{-1}
		linear	52660	False	False	1.61×10^3	9.28×10^{-1}
				True	False	1.64×10^3	6.13×10^{-1}
				False	True	1.64×10^3	9.75×10^{-1}
				True	True	1.65×10^3	9.76×10^{-1}
	hypercomplex	zero	12712	False	False	1.64×10^3	1.60×10^{-1}
				True	False	1.66×10^3	1.52×10^{-1}
				False	True	1.67×10^3	1.36×10^{-1}
				True	True	1.67×10^3	1.40×10^{-1}
		linear	63528	False	False	1.64×10^3	7.17×10^{-1}
				True	False	1.62×10^3	6.08×10^{-1}
				False	True	1.65×10^3	7.79×10^{-1}
				True	True	1.67×10^3	8.49×10^{-1}
	lphm	zero	930	False	False	1.70×10^3	1.25×10^{-1}
				True	False	1.72×10^3	1.34×10^{-1}
				False	True	1.71×10^3	1.34×10^{-1}
				True	True	1.72×10^3	1.33×10^{-1}
		linear	51746	False	False	1.67×10^3	8.63×10^{-1}
				True	False	1.67×10^3	8.37×10^{-1}
				False	True	1.70×10^3	9.28×10^{-1}
				True	True	1.73×10^3	9.76×10^{-1}
	hypernet	zero	512	False	False	1.20×10^3	1.12×10^{-1}
				True	False	1.36×10^3	1.65×10^{-1}
				False	True	1.36×10^3	9.40×10^{-2}
				True	True	1.38×10^3	2.30×10^{-1}
		linear	51328	False	False	1.16×10^3	1.05×10^{-1}
				True	False	1.38×10^3	1.61×10^{-1}
				False	True	1.37×10^3	7.64×10^{-1}
				True	True	1.44×10^3	7.98×10^{-1}

Table 15: Performance analysis of RPN with the taylor expansion function and different reconciliation and remainder functions on the MNIST dataset.

Expansion	Reconciliation	Remainder	Parameter#	Pre-Layer-Norm	Post-Layer-Norm	Time Cost	Accuracy
taylor	identity	zero	39429760	False	False	2.28×10^3	9.83×10^{-1}
				True	False	2.35×10^3	9.85×10^{-1}
				False	True	2.54×10^3	9.85×10^{-1}
				True	True	2.57×10^3	9.85×10^{-1}
		linear	39480576	False	False	2.26×10^3	9.81×10^{-1}
				True	False	2.26×10^3	9.81×10^{-1}
				False	True	2.47×10^3	9.82×10^{-1}
				True	True	2.47×10^3	9.82×10^{-1}
	masking	zero	39429760	False	False	3.10×10^3	9.83×10^{-1}
				True	False	3.11×10^3	9.85×10^{-1}
				False	True	3.21×10^3	9.85×10^{-1}
				True	True	3.10×10^3	9.85×10^{-1}
		linear	39480576	False	False	3.02×10^3	9.82×10^{-1}
				True	False	3.01×10^3	9.82×10^{-1}
				False	True	3.07×10^3	9.83×10^{-1}
				True	True	3.02×10^3	9.82×10^{-1}
	duplicated-padding	zero	9857440	False	False	2.28×10^3	5.08×10^{-1}
				True	False	2.25×10^3	5.09×10^{-1}
				False	True	2.44×10^3	5.10×10^{-1}
				True	True	2.50×10^3	5.09×10^{-1}
		linear	9908256	False	False	2.17×10^3	7.28×10^{-1}
				True	False	2.17×10^3	5.27×10^{-1}
				False	True	2.33×10^3	5.24×10^{-1}
				True	True	2.39×10^3	5.22×10^{-1}
	lorr	zero	1239348	False	False	2.11×10^3	8.65×10^{-1}
				True	False	2.09×10^3	8.69×10^{-1}
				False	True	2.27×10^3	8.30×10^{-1}
				True	True	2.31×10^3	8.75×10^{-1}
		linear	1290164	False	False	1.99×10^3	9.79×10^{-1}
				True	False	2.01×10^3	9.78×10^{-1}
				False	True	2.23×10^3	9.79×10^{-1}
				True	True	2.23×10^3	9.78×10^{-1}
	hypercomplex	zero	9857448	False	False	2.21×10^3	9.78×10^{-1}
				True	False	2.24×10^3	9.79×10^{-1}
				False	True	2.41×10^3	9.78×10^{-1}
				True	True	2.42×10^3	9.79×10^{-1}
		linear	9908264	False	False	2.16×10^3	9.82×10^{-1}
				True	False	2.11×10^3	9.81×10^{-1}
				False	True	2.29×10^3	9.81×10^{-1}
				True	True	2.29×10^3	9.81×10^{-1}
	lphm	zero	619682	False	False	2.18×10^3	7.74×10^{-1}
				True	False	2.21×10^3	7.72×10^{-1}
				False	True	2.33×10^3	7.72×10^{-1}
				True	True	2.34×10^3	7.76×10^{-1}
		linear	670498	False	False	2.06×10^3	9.80×10^{-1}
				True	False	2.05×10^3	9.79×10^{-1}
				False	True	2.18×10^3	9.80×10^{-1}
				True	True	2.17×10^3	9.80×10^{-1}
	hypernet	zero	512	False	False	5.12×10^3	1.92×10^{-1}
				True	False	5.13×10^3	1.93×10^{-1}
				False	True	3.84×10^3	1.54×10^{-1}
				True	True	3.83×10^3	2.28×10^{-1}
		linear	51328	False	False	4.95×10^3	6.31×10^{-1}
				True	False	4.95×10^3	7.72×10^{-1}
				False	True	5.07×10^3	7.62×10^{-1}
				True	True	5.07×10^3	7.83×10^{-1}

Table 16: Performance analysis of RPN with the fourier expansion function and different reconciliation and remainder functions on the MNIST dataset.

Expansion	Reconciliation	Remainder	Parameter#	Pre-Layer-Norm	Post-Layer-Norm	Time Cost	Accuracy
fourier	identity	zero	508160	False	False	1.85×10^3	1.06×10^{-1}
				True	False	1.85×10^3	8.95×10^{-1}
				False	True	1.74×10^3	1.07×10^{-1}
				True	True	1.76×10^3	8.97×10^{-1}
		linear	558976	False	False	1.71×10^3	1.05×10^{-1}
				True	False	1.72×10^3	9.47×10^{-1}
				False	True	1.72×10^3	1.07×10^{-1}
				True	True	1.73×10^3	9.45×10^{-1}
	masking	zero	508160	False	False	1.80×10^3	1.09×10^{-1}
				True	False	1.79×10^3	8.87×10^{-1}
				False	True	1.81×10^3	1.06×10^{-1}
				True	True	1.82×10^3	8.75×10^{-1}
		linear	558976	False	False	1.77×10^3	5.21×10^{-1}
				True	False	1.77×10^3	9.57×10^{-1}
				False	True	1.81×10^3	3.58×10^{-1}
				True	True	1.80×10^3	9.48×10^{-1}
	duplicated-padding	zero	127040	False	False	1.80×10^3	1.11×10^{-1}
				True	False	1.81×10^3	4.53×10^{-1}
				False	True	1.85×10^3	1.07×10^{-1}
				True	True	1.82×10^3	4.46×10^{-1}
		linear	177856	False	False	1.78×10^3	2.49×10^{-1}
				True	False	1.77×10^3	9.59×10^{-1}
				False	True	1.80×10^3	1.94×10^{-1}
				True	True	1.82×10^3	8.72×10^{-1}
	lorr	zero	17108	False	False	1.76×10^3	6.17×10^{-1}
				True	False	1.75×10^3	4.26×10^{-1}
				False	True	1.77×10^3	6.11×10^{-1}
				True	True	1.78×10^3	5.12×10^{-1}
		linear	67924	False	False	1.72×10^3	9.64×10^{-1}
				True	False	1.75×10^3	9.79×10^{-1}
				False	True	1.73×10^3	9.36×10^{-1}
				True	True	1.73×10^3	9.77×10^{-1}
	hypercomplex	zero	127048	False	False	1.76×10^3	7.62×10^{-1}
				True	False	1.76×10^3	7.05×10^{-1}
				False	True	1.74×10^3	7.61×10^{-1}
				True	True	1.75×10^3	7.18×10^{-1}
		linear	177864	False	False	1.70×10^3	9.77×10^{-1}
				True	False	1.70×10^3	9.77×10^{-1}
				False	True	1.72×10^3	9.77×10^{-1}
				True	True	1.73×10^3	9.78×10^{-1}
	lphm	zero	8562	False	False	1.75×10^3	7.00×10^{-1}
				True	False	1.73×10^3	5.52×10^{-1}
				False	True	1.75×10^3	8.06×10^{-1}
				True	True	1.75×10^3	5.73×10^{-1}
		linear	59378	False	False	1.72×10^3	9.76×10^{-1}
				True	False	1.72×10^3	9.77×10^{-1}
				False	True	1.71×10^3	9.77×10^{-1}
				True	True	1.73×10^3	9.77×10^{-1}
	hypernet	zero	512	False	False	8.85×10^2	1.07×10^{-1}
				True	False	1.24×10^3	1.19×10^{-1}
				False	True	1.24×10^3	1.07×10^{-1}
				True	True	1.28×10^3	1.17×10^{-1}
		linear	51328	False	False	8.64×10^2	2.81×10^{-1}
				True	False	1.14×10^3	1.90×10^{-1}
				False	True	1.19×10^3	2.90×10^{-1}
				True	True	1.17×10^3	2.01×10^{-1}

Table 17: Performance analysis of RPN with the hyperbolic expansion function and different reconciliation and remainder functions on the MNIST dataset.

Expansion	Reconciliation	Remainder	Parameter#	Pre-Layer-Norm	Post-Layer-Norm	Time Cost	Accuracy
hyperbolic	identity	zero	152448	False	False	1.86×10^3	2.55×10^{-1}
				True	False	1.88×10^3	9.72×10^{-1}
				False	True	1.85×10^3	9.39×10^{-1}
				True	True	1.86×10^3	9.75×10^{-1}
		linear	203264	False	False	1.74×10^3	3.11×10^{-1}
				True	False	1.74×10^3	9.73×10^{-1}
				False	True	1.72×10^3	9.26×10^{-1}
				True	True	1.75×10^3	9.75×10^{-1}
	masking	zero	152448	False	False	1.90×10^3	8.95×10^{-1}
				True	False	1.89×10^3	9.69×10^{-1}
				False	True	1.87×10^3	9.58×10^{-1}
				True	True	1.89×10^3	9.71×10^{-1}
		linear	203264	False	False	1.76×10^3	8.29×10^{-1}
				True	False	1.77×10^3	9.74×10^{-1}
				False	True	1.74×10^3	9.44×10^{-1}
				True	True	1.76×10^3	9.76×10^{-1}
	duplicated-padding	zero	38112	False	False	1.72×10^3	2.68×10^{-1}
				True	False	1.68×10^3	5.02×10^{-1}
				False	True	1.71×10^3	4.93×10^{-1}
				True	True	1.70×10^3	5.02×10^{-1}
		linear	88928	False	False	1.66×10^3	4.08×10^{-1}
				True	False	1.71×10^3	9.68×10^{-1}
				False	True	1.68×10^3	9.53×10^{-1}
				True	True	1.68×10^3	9.76×10^{-1}
	lorr	zero	5236	False	False	1.64×10^3	6.66×10^{-1}
				True	False	1.67×10^3	6.55×10^{-1}
				False	True	1.64×10^3	6.63×10^{-1}
				True	True	1.67×10^3	6.51×10^{-1}
		linear	56052	False	False	1.62×10^3	9.71×10^{-1}
				True	False	1.66×10^3	9.77×10^{-1}
				False	True	1.64×10^3	9.75×10^{-1}
				True	True	1.64×10^3	9.77×10^{-1}
	hypercomplex	zero	38120	False	False	1.65×10^3	7.43×10^{-1}
				True	False	1.68×10^3	7.44×10^{-1}
				False	True	1.67×10^3	7.75×10^{-1}
				True	True	1.68×10^3	7.75×10^{-1}
		linear	88936	False	False	1.64×10^3	9.74×10^{-1}
				True	False	1.65×10^3	9.79×10^{-1}
				False	True	1.63×10^3	9.77×10^{-1}
				True	True	1.68×10^3	9.78×10^{-1}
	lphm	zero	2626	False	False	1.70×10^3	7.25×10^{-1}
				True	False	1.72×10^3	7.16×10^{-1}
				False	True	1.70×10^3	7.27×10^{-1}
				True	True	1.70×10^3	7.09×10^{-1}
		linear	53442	False	False	1.66×10^3	9.71×10^{-1}
				True	False	1.68×10^3	9.75×10^{-1}
				False	True	1.66×10^3	9.71×10^{-1}
				True	True	1.67×10^3	9.75×10^{-1}
	hypernet	zero	512	False	False	7.44×10^2	9.80×10^{-2}
				True	False	9.16×10^2	6.11×10^{-1}
				False	True	9.04×10^2	9.80×10^{-2}
				True	True	9.09×10^2	5.36×10^{-1}
		linear	51328	False	False	7.38×10^2	9.80×10^{-2}
				True	False	9.11×10^2	6.25×10^{-1}
				False	True	8.93×10^2	9.80×10^{-2}
				True	True	8.67×10^2	6.73×10^{-1}

Table 18: Performance analysis of RPN with the arc-hyperbolic expansion function and different reconciliation and remainder functions on the MNIST dataset.

Expansion	Reconciliation	Remainder	Parameter#	Pre-Layer-Norm	Post-Layer-Norm	Time Cost	Accuracy
arc-hyperbolic	identity	zero	152448	False	False	1.44×10^3	9.72×10^{-1}
				True	False	1.59×10^3	9.61×10^{-1}
				False	True	1.51×10^3	9.74×10^{-1}
				True	True	1.63×10^3	9.76×10^{-1}
		linear	203264	False	False	1.59×10^3	9.70×10^{-1}
				True	False	1.61×10^3	9.79×10^{-1}
				False	True	1.59×10^3	9.75×10^{-1}
				True	True	1.63×10^3	9.79×10^{-1}
	masking	zero	152448	False	False	1.68×10^3	9.73×10^{-1}
				True	False	1.71×10^3	9.59×10^{-1}
				False	True	1.70×10^3	9.73×10^{-1}
				True	True	1.70×10^3	9.73×10^{-1}
		linear	203264	False	False	1.67×10^3	9.75×10^{-1}
				True	False	1.68×10^3	9.77×10^{-1}
				False	True	1.65×10^3	9.74×10^{-1}
				True	True	1.68×10^3	9.78×10^{-1}
	duplicated-padding	zero	38112	False	False	1.68×10^3	5.01×10^{-1}
				True	False	1.70×10^3	4.99×10^{-1}
				False	True	1.70×10^3	4.99×10^{-1}
				True	True	1.71×10^3	5.03×10^{-1}
		linear	88928	False	False	1.64×10^3	9.69×10^{-1}
				True	False	1.71×10^3	9.74×10^{-1}
				False	True	1.66×10^3	9.73×10^{-1}
				True	True	1.68×10^3	9.72×10^{-1}
	lorr	zero	5236	False	False	1.66×10^3	6.57×10^{-1}
				True	False	1.66×10^3	6.25×10^{-1}
				False	True	1.67×10^3	6.61×10^{-1}
				True	True	1.67×10^3	6.34×10^{-1}
		linear	56052	False	False	1.62×10^3	9.76×10^{-1}
				True	False	1.65×10^3	9.76×10^{-1}
				False	True	1.64×10^3	9.77×10^{-1}
				True	True	1.66×10^3	9.78×10^{-1}
	hypercomplex	zero	38120	False	False	1.66×10^3	7.76×10^{-1}
				True	False	1.68×10^3	7.74×10^{-1}
				False	True	1.70×10^3	7.81×10^{-1}
				True	True	1.69×10^3	7.77×10^{-1}
		linear	88936	False	False	1.65×10^3	9.77×10^{-1}
				True	False	1.66×10^3	9.78×10^{-1}
				False	True	1.68×10^3	9.78×10^{-1}
				True	True	1.67×10^3	9.77×10^{-1}
	lphm	zero	2626	False	False	1.69×10^3	8.33×10^{-1}
				True	False	1.71×10^3	7.19×10^{-1}
				False	True	1.69×10^3	8.57×10^{-1}
				True	True	1.71×10^3	7.19×10^{-1}
		linear	53442	False	False	1.66×10^3	9.75×10^{-1}
				True	False	1.71×10^3	9.75×10^{-1}
				False	True	1.69×10^3	9.75×10^{-1}
				True	True	1.69×10^3	9.75×10^{-1}
	hypernet	zero	512	False	False	7.51×10^2	1.34×10^{-1}
				True	False	9.26×10^2	2.70×10^{-1}
				False	True	9.11×10^2	5.64×10^{-1}
				True	True	8.95×10^2	5.87×10^{-1}
		linear	51328	False	False	7.42×10^2	3.90×10^{-1}
				True	False	8.61×10^2	3.39×10^{-1}
				False	True	8.82×10^2	7.27×10^{-1}
				True	True	8.78×10^2	7.41×10^{-1}

Table 19: Performance analysis of RPN with the trigonometric expansion function and different reconciliation and remainder functions on the MNIST dataset.

Expansion	Reconciliation	Remainder	Parameter#	Pre-Layer-Norm	Post-Layer-Norm	Time Cost	Accuracy
trigonometric	identity	zero	152448	False	False	1.80×10^3	2.42×10^{-1}
				True	False	1.86×10^3	4.87×10^{-1}
				False	True	1.84×10^3	1.08×10^{-1}
				True	True	1.89×10^3	8.89×10^{-1}
		linear	203264	False	False	1.78×10^3	1.85×10^{-1}
				True	False	1.80×10^3	5.95×10^{-1}
				False	True	1.78×10^3	4.27×10^{-1}
				True	True	1.78×10^3	9.09×10^{-1}
	masking	zero	152448	False	False	2.00×10^3	3.07×10^{-1}
				True	False	2.04×10^3	3.98×10^{-1}
				False	True	2.01×10^3	1.26×10^{-1}
				True	True	2.04×10^3	8.71×10^{-1}
		linear	203264	False	False	1.92×10^3	2.83×10^{-1}
				True	False	1.92×10^3	5.66×10^{-1}
				False	True	1.92×10^3	7.25×10^{-1}
				True	True	1.91×10^3	9.48×10^{-1}
	duplicated-padding	zero	38112	False	False	2.00×10^3	1.52×10^{-1}
				True	False	2.04×10^3	2.78×10^{-1}
				False	True	2.04×10^3	1.13×10^{-1}
				True	True	2.06×10^3	4.81×10^{-1}
		linear	88928	False	False	1.89×10^3	1.49×10^{-1}
				True	False	1.89×10^3	3.29×10^{-1}
				False	True	1.90×10^3	4.40×10^{-1}
				True	True	1.92×10^3	9.70×10^{-1}
	lorr	zero	5236	False	False	1.82×10^3	2.17×10^{-1}
				True	False	1.84×10^3	1.64×10^{-1}
				False	True	1.86×10^3	1.41×10^{-1}
				True	True	1.85×10^3	3.80×10^{-1}
		linear	56052	False	False	1.76×10^3	8.57×10^{-1}
				True	False	1.79×10^3	7.39×10^{-1}
				False	True	1.78×10^3	9.75×10^{-1}
				True	True	1.77×10^3	9.75×10^{-1}
	hypercomplex	zero	38120	False	False	1.86×10^3	2.31×10^{-1}
				True	False	1.88×10^3	2.36×10^{-1}
				False	True	1.85×10^3	1.25×10^{-1}
				True	True	1.86×10^3	6.21×10^{-1}
		linear	88936	False	False	1.74×10^3	8.58×10^{-1}
				True	False	1.78×10^3	4.98×10^{-1}
				False	True	1.77×10^3	9.73×10^{-1}
				True	True	1.75×10^3	9.74×10^{-1}
	lphm	zero	2626	False	False	1.89×10^3	1.88×10^{-1}
				True	False	1.93×10^3	2.02×10^{-1}
				False	True	1.91×10^3	1.29×10^{-1}
				True	True	1.92×10^3	4.40×10^{-1}
		linear	53442	False	False	1.78×10^3	8.72×10^{-1}
				True	False	1.86×10^3	7.87×10^{-1}
				False	True	1.84×10^3	5.97×10^{-1}
				True	True	1.86×10^3	9.75×10^{-1}
	hypernet	zero	512	False	False	7.50×10^2	1.06×10^{-1}
				True	False	8.92×10^2	1.56×10^{-1}
				False	True	8.93×10^2	1.08×10^{-1}
				True	True	9.19×10^2	3.75×10^{-1}
		linear	51328	False	False	7.31×10^2	1.40×10^{-1}
				True	False	8.78×10^2	2.05×10^{-1}
				False	True	8.68×10^2	3.67×10^{-1}
				True	True	8.52×10^2	4.35×10^{-1}

Table 20: Performance analysis of RPN with the arc-trigonometric expansion function and different reconciliation and remainder functions on the MNIST dataset.

Expansion	Reconciliation	Remainder	Parameter#	Pre-Layer-Norm	Post-Layer-Norm	Time Cost	Accuracy
arc-trigonometric	identity	zero	152448	False	False	8.83×10^2	9.66×10^{-1}
				True	False	1.45×10^3	9.52×10^{-1}
				False	True	1.45×10^3	9.68×10^{-1}
				True	True	1.42×10^3	9.72×10^{-1}
		linear	203264	False	False	8.00×10^2	9.66×10^{-1}
				True	False	8.02×10^2	9.78×10^{-1}
				False	True	8.02×10^2	9.67×10^{-1}
				True	True	8.06×10^2	9.79×10^{-1}
	masking	zero	152448	False	False	1.07×10^3	9.73×10^{-1}
				True	False	1.07×10^3	9.52×10^{-1}
				False	True	1.07×10^3	9.73×10^{-1}
				True	True	1.06×10^3	9.68×10^{-1}
		linear	203264	False	False	9.49×10^2	9.73×10^{-1}
				True	False	9.51×10^2	9.77×10^{-1}
				False	True	9.50×10^2	9.76×10^{-1}
				True	True	9.52×10^2	9.77×10^{-1}
	duplicated-padding	zero	38112	False	False	1.05×10^3	5.02×10^{-1}
				True	False	1.06×10^3	4.96×10^{-1}
				False	True	1.06×10^3	4.96×10^{-1}
				True	True	1.06×10^3	5.01×10^{-1}
		linear	88928	False	False	9.27×10^2	9.69×10^{-1}
				True	False	9.35×10^2	9.71×10^{-1}
				False	True	9.34×10^2	9.73×10^{-1}
				True	True	9.34×10^2	9.75×10^{-1}
	lorr	zero	5236	False	False	9.11×10^2	6.31×10^{-1}
				True	False	9.16×10^2	6.02×10^{-1}
				False	True	9.14×10^2	6.55×10^{-1}
				True	True	9.16×10^2	6.46×10^{-1}
		linear	56052	False	False	8.39×10^2	9.76×10^{-1}
				True	False	8.44×10^2	9.76×10^{-1}
				False	True	8.43×10^2	9.77×10^{-1}
				True	True	8.46×10^2	9.78×10^{-1}
	hypercomplex	zero	38120	False	False	9.08×10^2	7.75×10^{-1}
				True	False	9.13×10^2	7.65×10^{-1}
				False	True	9.09×10^2	7.76×10^{-1}
				True	True	9.14×10^2	7.79×10^{-1}
		linear	88936	False	False	8.35×10^2	9.77×10^{-1}
				True	False	8.43×10^2	9.77×10^{-1}
				False	True	8.39×10^2	9.77×10^{-1}
				True	True	8.45×10^2	9.77×10^{-1}
	lphm	zero	2626	False	False	9.28×10^2	7.28×10^{-1}
				True	False	9.29×10^2	6.89×10^{-1}
				False	True	9.32×10^2	7.30×10^{-1}
				True	True	9.31×10^2	7.29×10^{-1}
		linear	53442	False	False	7.97×10^2	9.74×10^{-1}
				True	False	8.15×10^2	9.75×10^{-1}
				False	True	8.08×10^2	9.76×10^{-1}
				True	True	8.29×10^2	9.74×10^{-1}
	hypernet	zero	512	False	False	7.56×10^2	1.26×10^{-1}
				True	False	9.33×10^2	1.15×10^{-1}
				False	True	9.35×10^2	5.01×10^{-1}
				True	True	9.51×10^2	5.29×10^{-1}
		linear	51328	False	False	7.51×10^2	7.23×10^{-1}
				True	False	8.73×10^2	6.92×10^{-1}
				False	True	9.01×10^2	6.32×10^{-1}
				True	True	8.70×10^2	6.11×10^{-1}

Table 21: Performance analysis of RPN with the bspline expansion function and different reconciliation and remainder functions on the MNIST dataset.

Expansion	Reconciliation	Remainder	Parameter#	Pre-Layer-Norm	Post-Layer-Norm	Time Cost	Accuracy
bspline	identity	zero	406528	False	False	2.01×10^3	9.28×10^{-1}
				True	False	2.05×10^3	9.72×10^{-1}
				False	True	2.02×10^3	9.80×10^{-2}
				True	True	2.07×10^3	9.78×10^{-1}
		linear	457344	False	False	1.95×10^3	9.64×10^{-1}
				True	False	1.94×10^3	9.76×10^{-1}
				False	True	1.96×10^3	9.80×10^{-2}
				True	True	1.98×10^3	9.80×10^{-1}
	masking	zero	406528	False	False	2.20×10^3	9.39×10^{-1}
				True	False	2.20×10^3	9.63×10^{-1}
				False	True	2.23×10^3	9.80×10^{-2}
				True	True	2.24×10^3	9.72×10^{-1}
		linear	457344	False	False	2.08×10^3	9.70×10^{-1}
				True	False	2.09×10^3	9.78×10^{-1}
				False	True	2.12×10^3	8.22×10^{-1}
				True	True	2.12×10^3	9.79×10^{-1}
	duplicated-padding	zero	101632	False	False	2.23×10^3	4.86×10^{-1}
				True	False	2.21×10^3	4.99×10^{-1}
				False	True	2.22×10^3	9.80×10^{-2}
				True	True	2.23×10^3	5.02×10^{-1}
		linear	152448	False	False	2.06×10^3	9.74×10^{-1}
				True	False	2.04×10^3	9.71×10^{-1}
				False	True	2.03×10^3	2.72×10^{-1}
				True	True	2.06×10^3	7.65×10^{-1}
	lorr	zero	13716	False	False	2.02×10^3	6.72×10^{-1}
				True	False	2.05×10^3	6.22×10^{-1}
				False	True	2.04×10^3	6.87×10^{-1}
				True	True	2.05×10^3	6.39×10^{-1}
		linear	64532	False	False	1.95×10^3	9.77×10^{-1}
				True	False	1.94×10^3	9.77×10^{-1}
				False	True	1.92×10^3	9.76×10^{-1}
				True	True	1.93×10^3	9.77×10^{-1}
	hypercomplex	zero	101640	False	False	2.03×10^3	7.64×10^{-1}
				True	False	2.03×10^3	9.24×10^{-1}
				False	True	2.02×10^3	7.73×10^{-1}
				True	True	2.04×10^3	7.74×10^{-1}
		linear	152456	False	False	1.94×10^3	9.77×10^{-1}
				True	False	1.98×10^3	9.77×10^{-1}
				False	True	1.94×10^3	9.77×10^{-1}
				True	True	1.95×10^3	9.77×10^{-1}
	lphm	zero	6866	False	False	2.11×10^3	8.12×10^{-1}
				True	False	2.10×10^3	8.12×10^{-1}
				False	True	2.09×10^3	8.27×10^{-1}
				True	True	2.13×10^3	7.00×10^{-1}
		linear	57682	False	False	1.99×10^3	9.78×10^{-1}
				True	False	2.00×10^3	9.78×10^{-1}
				False	True	1.99×10^3	9.78×10^{-1}
				True	True	1.97×10^3	9.78×10^{-1}
	hypernet	zero	512	False	False	9.19×10^2	1.00×10^{-1}
				True	False	1.23×10^3	2.89×10^{-1}
				False	True	1.23×10^3	1.00×10^{-1}
				True	True	1.27×10^3	3.44×10^{-1}
		linear	51328	False	False	8.88×10^2	1.00×10^{-1}
				True	False	1.09×10^3	4.51×10^{-1}
				False	True	1.16×10^3	1.00×10^{-1}
				True	True	1.11×10^3	4.48×10^{-1}

Table 22: Performance analysis of RPN with the chebyshev expansion function and different reconciliation and remainder functions on the MNIST dataset.

Expansion	Reconciliation	Remainder	Parameter#	Pre-Layer-Norm	Post-Layer-Norm	Time Cost	Accuracy
chebyshev	identity	zero	254080	False	False	3.68×10^3	8.45×10^{-1}
				True	False	3.70×10^3	9.68×10^{-1}
				False	True	3.69×10^3	9.65×10^{-1}
				True	True	3.71×10^3	9.68×10^{-1}
	linear	304896	304896	False	False	3.50×10^3	8.47×10^{-1}
				True	False	3.51×10^3	9.67×10^{-1}
				False	True	3.51×10^3	9.74×10^{-1}
				True	True	3.51×10^3	9.75×10^{-1}
	masking	zero	254080	False	False	3.55×10^3	8.30×10^{-1}
				True	False	3.56×10^3	9.61×10^{-1}
				False	True	3.55×10^3	9.55×10^{-1}
				True	True	3.57×10^3	9.64×10^{-1}
		linear	304896	False	False	3.50×10^3	8.43×10^{-1}
				True	False	3.53×10^3	9.62×10^{-1}
				False	True	3.53×10^3	9.71×10^{-1}
				True	True	3.52×10^3	9.71×10^{-1}
	duplicated-padding	zero	63520	False	False	3.57×10^3	3.81×10^{-1}
				True	False	3.57×10^3	4.99×10^{-1}
				False	True	3.56×10^3	4.99×10^{-1}
				True	True	3.57×10^3	4.98×10^{-1}
		linear	114336	False	False	3.33×10^3	4.65×10^{-1}
				True	False	3.31×10^3	4.99×10^{-1}
				False	True	3.31×10^3	9.03×10^{-1}
				True	True	3.29×10^3	9.18×10^{-1}
	lorr	zero	8628	False	False	3.26×10^3	4.93×10^{-1}
				True	False	3.23×10^3	6.06×10^{-1}
				False	True	3.25×10^3	6.37×10^{-1}
				True	True	3.23×10^3	6.10×10^{-1}
		linear	59444	False	False	3.08×10^3	9.34×10^{-1}
				True	False	3.07×10^3	9.62×10^{-1}
				False	True	3.07×10^3	9.78×10^{-1}
				True	True	3.07×10^3	9.78×10^{-1}
	hypercomplex	zero	63528	False	False	3.01×10^3	7.97×10^{-1}
				True	False	3.01×10^3	9.37×10^{-1}
				False	True	3.01×10^3	9.24×10^{-1}
				True	True	3.01×10^3	9.37×10^{-1}
		linear	114344	False	False	3.01×10^3	9.44×10^{-1}
				True	False	3.01×10^3	9.60×10^{-1}
				False	True	3.01×10^3	9.78×10^{-1}
				True	True	3.02×10^3	9.79×10^{-1}
	lphm	zero	4322	False	False	3.03×10^3	6.63×10^{-1}
				True	False	3.02×10^3	6.79×10^{-1}
				False	True	3.02×10^3	6.80×10^{-1}
				True	True	3.04×10^3	6.75×10^{-1}
		linear	55138	False	False	3.03×10^3	9.76×10^{-1}
				True	False	3.03×10^3	9.77×10^{-1}
				False	True	3.03×10^3	9.76×10^{-1}
				True	True	3.04×10^3	9.77×10^{-1}
	hypernet	zero	512	False	False	1.97×10^3	5.92×10^{-2}
				True	False	3.02×10^3	3.48×10^{-1}
				False	True	3.02×10^3	5.04×10^{-1}
				True	True	3.06×10^3	5.01×10^{-1}
		linear	51328	False	False	1.95×10^3	7.34×10^{-2}
				True	False	2.90×10^3	3.38×10^{-1}
				False	True	2.99×10^3	7.38×10^{-1}
				True	True	2.94×10^3	7.71×10^{-1}

Table 23: Performance analysis of RPN with the jacobi expansion function and different reconciliation and remainder functions on the MNIST dataset.

Expansion	Reconciliation	Remainder	Parameter#	Pre-Layer-Norm	Post-Layer-Norm	Time Cost	Accuracy
jacobi	identity	zero	254080	False	False	2.00×10^3	8.67×10^{-1}
				True	False	1.71×10^3	9.70×10^{-1}
				False	True	1.87×10^3	9.66×10^{-1}
				True	True	1.57×10^3	9.70×10^{-1}
		linear	304896	False	False	1.40×10^3	8.69×10^{-1}
				True	False	1.40×10^3	9.67×10^{-1}
				False	True	1.40×10^3	9.76×10^{-1}
				True	True	1.40×10^3	9.74×10^{-1}
	masking	zero	254080	False	False	1.41×10^3	8.28×10^{-1}
				True	False	1.41×10^3	9.62×10^{-1}
				False	True	1.41×10^3	9.58×10^{-1}
				True	True	1.42×10^3	9.65×10^{-1}
		linear	304896	False	False	1.40×10^3	8.30×10^{-1}
				True	False	1.41×10^3	9.62×10^{-1}
				False	True	1.41×10^3	9.71×10^{-1}
				True	True	1.41×10^3	9.72×10^{-1}
	duplicated-padding	zero	63520	False	False	1.41×10^3	3.97×10^{-1}
				True	False	1.41×10^3	4.99×10^{-1}
				False	True	1.41×10^3	4.98×10^{-1}
				True	True	1.42×10^3	4.99×10^{-1}
		linear	114336	False	False	1.41×10^3	4.39×10^{-1}
				True	False	1.41×10^3	4.96×10^{-1}
				False	True	1.41×10^3	9.22×10^{-1}
				True	True	1.42×10^3	8.81×10^{-1}
	lorr	zero	8628	False	False	1.40×10^3	3.18×10^{-1}
				True	False	1.41×10^3	5.45×10^{-1}
				False	True	1.40×10^3	6.40×10^{-1}
				True	True	1.41×10^3	6.09×10^{-1}
		linear	59444	False	False	1.40×10^3	9.29×10^{-1}
				True	False	1.40×10^3	9.22×10^{-1}
				False	True	1.40×10^3	9.78×10^{-1}
				True	True	1.41×10^3	9.77×10^{-1}
	hypercomplex	zero	63528	False	False	1.41×10^3	7.11×10^{-1}
				True	False	1.41×10^3	9.38×10^{-1}
				False	True	1.41×10^3	9.30×10^{-1}
				True	True	1.42×10^3	9.34×10^{-1}
		linear	114344	False	False	1.41×10^3	9.46×10^{-1}
				True	False	1.41×10^3	9.56×10^{-1}
				False	True	1.41×10^3	9.78×10^{-1}
				True	True	1.41×10^3	9.78×10^{-1}
	lphm	zero	4322	False	False	1.42×10^3	6.62×10^{-1}
				True	False	1.42×10^3	6.77×10^{-1}
				False	True	1.41×10^3	6.69×10^{-1}
				True	True	1.42×10^3	6.83×10^{-1}
		linear	55138	False	False	1.41×10^3	9.75×10^{-1}
				True	False	1.35×10^3	9.76×10^{-1}
				False	True	1.40×10^3	9.76×10^{-1}
				True	True	1.27×10^3	9.76×10^{-1}
	hypernet	zero	512	False	False	1.99×10^3	5.86×10^{-2}
				True	False	3.03×10^3	3.75×10^{-1}
				False	True	3.02×10^3	4.97×10^{-1}
				True	True	3.00×10^3	5.56×10^{-1}
		linear	51328	False	False	2.00×10^3	7.28×10^{-2}
				True	False	3.02×10^3	4.15×10^{-1}
				False	True	3.03×10^3	7.81×10^{-1}
				True	True	3.03×10^3	7.75×10^{-1}

Table 24: Performance analysis of RPN with the gaussian-rbf expansion function and different reconciliation and remainder functions on the MNIST dataset.

Expansion	Reconciliation	Remainder	Parameter#	Pre-Layer-Norm	Post-Layer-Norm	Time Cost	Accuracy
gaussian-rbf	identity	zero	508160	False	False	8.74×10^2	9.82×10^{-2}
				True	False	1.43×10^3	9.79×10^{-1}
				False	True	1.45×10^3	1.07×10^{-1}
				True	True	1.37×10^3	9.81×10^{-1}
		linear	558976	False	False	7.90×10^2	7.95×10^{-1}
				True	False	7.92×10^2	9.82×10^{-1}
				False	True	7.97×10^2	4.52×10^{-1}
				True	True	7.96×10^2	9.82×10^{-1}
	masking	zero	508160	False	False	1.06×10^3	8.75×10^{-1}
				True	False	1.06×10^3	9.78×10^{-1}
				False	True	1.06×10^3	8.03×10^{-1}
				True	True	1.05×10^3	9.81×10^{-1}
		linear	558976	False	False	9.36×10^2	9.25×10^{-1}
				True	False	9.39×10^2	9.79×10^{-1}
				False	True	9.38×10^2	7.27×10^{-1}
				True	True	9.44×10^2	9.81×10^{-1}
	duplicated-padding	zero	127040	False	False	1.05×10^3	9.80×10^{-2}
				True	False	1.05×10^3	5.04×10^{-1}
				False	True	1.06×10^3	9.80×10^{-2}
				True	True	1.05×10^3	5.03×10^{-1}
		linear	177856	False	False	9.22×10^2	6.56×10^{-1}
				True	False	9.26×10^2	9.21×10^{-1}
				False	True	9.29×10^2	7.21×10^{-1}
				True	True	9.28×10^2	8.33×10^{-1}
	lorr	zero	17108	False	False	8.99×10^2	6.49×10^{-1}
				True	False	9.00×10^2	6.17×10^{-1}
				False	True	9.03×10^2	6.57×10^{-1}
				True	True	9.06×10^2	6.43×10^{-1}
		linear	67924	False	False	8.27×10^2	9.79×10^{-1}
				True	False	8.31×10^2	9.78×10^{-1}
				False	True	8.35×10^2	9.79×10^{-1}
				True	True	8.33×10^2	9.78×10^{-1}
	hypercomplex	zero	127048	False	False	9.00×10^2	7.74×10^{-1}
				True	False	9.03×10^2	7.78×10^{-1}
				False	True	9.04×10^2	7.71×10^{-1}
				True	True	9.06×10^2	7.78×10^{-1}
		linear	177864	False	False	8.28×10^2	9.78×10^{-1}
				True	False	8.31×10^2	9.77×10^{-1}
				False	True	8.34×10^2	9.79×10^{-1}
				True	True	8.34×10^2	9.77×10^{-1}
	lphm	zero	8562	False	False	9.18×10^2	7.06×10^{-1}
				True	False	9.16×10^2	8.15×10^{-1}
				False	True	9.23×10^2	7.12×10^{-1}
				True	True	9.19×10^2	8.15×10^{-1}
		linear	59378	False	False	8.18×10^2	9.77×10^{-1}
				True	False	8.33×10^2	9.78×10^{-1}
				False	True	8.22×10^2	9.79×10^{-1}
				True	True	8.41×10^2	9.77×10^{-1}
	hypernet	zero	512	False	False	8.33×10^2	1.03×10^{-1}
				True	False	1.20×10^3	2.83×10^{-1}
				False	True	1.21×10^3	1.03×10^{-1}
				True	True	1.21×10^3	5.06×10^{-1}
		linear	51328	False	False	8.27×10^2	1.02×10^{-1}
				True	False	1.18×10^3	3.35×10^{-1}
				False	True	1.17×10^3	1.15×10^{-1}
				True	True	1.12×10^3	7.36×10^{-1}

Table 25: Performance analysis of RPN with the inverse-quadratic-rbf expansion function and different reconciliation and remainder functions on the MNIST dataset.

Expansion	Reconciliation	Remainder	Parameter#	Pre-Layer-Norm	Post-Layer-Norm	Time Cost	Accuracy
inverse-quadratic-rbf	identity	zero	508160	False	False	1.46×10^3	3.18×10^{-1}
				True	False	1.61×10^3	9.81×10^{-1}
				False	True	1.55×10^3	9.48×10^{-1}
				True	True	1.63×10^3	9.80×10^{-1}
	linear	558976	False	False	1.60×10^3	8.79×10^{-1}	
			True	False	1.62×10^3	9.80×10^{-1}	
			False	True	1.58×10^3	9.53×10^{-1}	
			True	True	1.61×10^3	9.81×10^{-1}	
	masking	zero	508160	False	False	1.68×10^3	9.14×10^{-1}
				True	False	1.70×10^3	9.80×10^{-1}
				False	True	1.71×10^3	9.63×10^{-1}
				True	True	1.73×10^3	9.82×10^{-1}
		linear	558976	False	False	1.67×10^3	9.30×10^{-1}
				True	False	1.67×10^3	9.79×10^{-1}
				False	True	1.69×10^3	9.68×10^{-1}
				True	True	1.69×10^3	9.82×10^{-1}
	duplicated-padding	zero	127040	False	False	1.70×10^3	2.99×10^{-1}
				True	False	1.69×10^3	5.04×10^{-1}
				False	True	1.69×10^3	5.00×10^{-1}
				True	True	1.72×10^3	5.03×10^{-1}
		linear	177856	False	False	1.66×10^3	4.78×10^{-1}
				True	False	1.69×10^3	7.92×10^{-1}
				False	True	1.67×10^3	6.93×10^{-1}
				True	True	1.69×10^3	8.60×10^{-1}
	lorr	zero	17108	False	False	1.66×10^3	6.45×10^{-1}
				True	False	1.66×10^3	6.61×10^{-1}
				False	True	1.65×10^3	6.63×10^{-1}
				True	True	1.66×10^3	6.46×10^{-1}
		linear	67924	False	False	1.64×10^3	9.78×10^{-1}
				True	False	1.64×10^3	9.78×10^{-1}
				False	True	1.63×10^3	9.78×10^{-1}
				True	True	1.65×10^3	9.78×10^{-1}
	hypercomplex	zero	127048	False	False	1.67×10^3	7.77×10^{-1}
				True	False	1.68×10^3	7.75×10^{-1}
				False	True	1.67×10^3	7.76×10^{-1}
				True	True	1.67×10^3	7.76×10^{-1}
		linear	177864	False	False	1.62×10^3	9.78×10^{-1}
				True	False	1.66×10^3	9.77×10^{-1}
				False	True	1.65×10^3	9.79×10^{-1}
				True	True	1.67×10^3	9.78×10^{-1}
	lphm	zero	8562	False	False	1.70×10^3	6.98×10^{-1}
				True	False	1.70×10^3	7.64×10^{-1}
				False	True	1.71×10^3	7.15×10^{-1}
				True	True	1.70×10^3	8.25×10^{-1}
		linear	59378	False	False	1.66×10^3	9.78×10^{-1}
				True	False	1.70×10^3	9.78×10^{-1}
				False	True	1.69×10^3	9.78×10^{-1}
				True	True	1.72×10^3	9.76×10^{-1}
	hypernet	zero	512	False	False	8.44×10^2	1.83×10^{-1}
				True	False	1.08×10^3	2.54×10^{-1}
				False	True	1.10×10^3	1.39×10^{-1}
				True	True	1.04×10^3	5.52×10^{-1}
		linear	51328	False	False	8.26×10^2	2.16×10^{-1}
				True	False	1.12×10^3	2.40×10^{-1}
				False	True	1.08×10^3	3.15×10^{-1}
				True	True	1.11×10^3	7.73×10^{-1}

Table 26: Performance analysis of RPN with the normal expansion function and different reconciliation and remainder functions on the MNIST dataset.

Expansion	Reconciliation	Remainder	Parameter#	Pre-Layer-Norm	Post-Layer-Norm	Time Cost	Accuracy
normal	identity	zero	304896	False	False	1.17×10^3	9.66×10^{-1}
				True	False	1.37×10^3	1.14×10^{-1}
				False	True	1.40×10^3	1.31×10^{-1}
				True	True	1.42×10^3	8.73×10^{-1}
		linear	355712	False	False	1.12×10^3	9.69×10^{-1}
				True	False	1.37×10^3	3.42×10^{-1}
				False	True	1.40×10^3	1.00×10^{-1}
				True	True	1.43×10^3	9.30×10^{-1}
	masking	zero	304896	False	False	1.32×10^3	9.66×10^{-1}
				True	False	1.34×10^3	2.12×10^{-1}
				False	True	1.35×10^3	6.67×10^{-1}
				True	True	1.41×10^3	9.20×10^{-1}
		linear	355712	False	False	1.18×10^3	9.65×10^{-1}
				True	False	1.48×10^3	6.70×10^{-1}
				False	True	1.51×10^3	7.35×10^{-1}
				True	True	1.57×10^3	9.53×10^{-1}
	duplicated-padding	zero	76224	False	False	1.32×10^3	4.88×10^{-1}
				True	False	1.46×10^3	1.14×10^{-1}
				False	True	1.47×10^3	1.28×10^{-1}
				True	True	1.53×10^3	4.30×10^{-1}
		linear	127040	False	False	1.23×10^3	7.01×10^{-1}
				True	False	1.53×10^3	1.55×10^{-1}
				False	True	1.56×10^3	1.68×10^{-1}
				True	True	1.60×10^3	4.97×10^{-1}
	lorr	zero	10324	False	False	1.19×10^3	6.38×10^{-1}
				True	False	1.50×10^3	6.09×10^{-1}
				False	True	1.51×10^3	6.57×10^{-1}
				True	True	1.58×10^3	6.89×10^{-1}
		linear	61140	False	False	1.14×10^3	9.77×10^{-1}
				True	False	1.57×10^3	9.69×10^{-1}
				False	True	1.59×10^3	9.77×10^{-1}
				True	True	1.61×10^3	9.76×10^{-1}
	hypercomplex	zero	76232	False	False	1.12×10^3	9.36×10^{-1}
				True	False	1.55×10^3	9.44×10^{-1}
				False	True	1.55×10^3	9.30×10^{-1}
				True	True	1.57×10^3	9.26×10^{-1}
		linear	127048	False	False	7.62×10^2	9.77×10^{-1}
				True	False	1.59×10^3	9.73×10^{-1}
				False	True	1.62×10^3	9.77×10^{-1}
				True	True	1.64×10^3	9.76×10^{-1}
	lphm	zero	5170	False	False	7.68×10^2	5.28×10^{-1}
				True	False	1.59×10^3	6.50×10^{-1}
				False	True	1.60×10^3	7.49×10^{-1}
				True	True	1.62×10^3	7.49×10^{-1}
		linear	55986	False	False	7.70×10^2	9.78×10^{-1}
				True	False	1.64×10^3	9.76×10^{-1}
				False	True	1.68×10^3	9.77×10^{-1}
				True	True	1.78×10^3	9.77×10^{-1}
	hypernet	zero	512	False	False	1.64×10^3	3.51×10^{-1}
				True	False	1.84×10^3	2.64×10^{-1}
				False	True	1.82×10^3	1.03×10^{-1}
				True	True	1.84×10^3	1.89×10^{-1}
		linear	51328	False	False	1.65×10^3	4.43×10^{-1}
				True	False	1.91×10^3	5.96×10^{-1}
				False	True	1.85×10^3	1.73×10^{-1}
				True	True	1.94×10^3	6.23×10^{-1}

Table 27: Performance analysis of RPN with the cauchy expansion function and different reconciliation and remainder functions on the MNIST dataset.

Expansion	Reconciliation	Remainder	Parameter#	Pre-Layer-Norm	Post-Layer-Norm	Time Cost	Accuracy
cauchy	identity	zero	304896	False	False	8.42×10^4	9.26×10^{-1}
				True	False	1.98×10^3	1.14×10^{-1}
				False	True	1.91×10^3	9.33×10^{-1}
				True	True	1.83×10^3	9.45×10^{-1}
		linear	355712	False	False	6.79×10^3	9.48×10^{-1}
				True	False	1.99×10^3	9.16×10^{-1}
				False	True	1.98×10^3	9.46×10^{-1}
				True	True	1.83×10^3	9.50×10^{-1}
	masking	zero	304896	False	False	6.83×10^3	9.48×10^{-1}
				True	False	1.77×10^3	2.13×10^{-1}
				False	True	1.76×10^3	9.53×10^{-1}
				True	True	1.80×10^3	9.50×10^{-1}
		linear	355712	False	False	2.47×10^3	9.59×10^{-1}
				True	False	1.78×10^3	9.21×10^{-1}
				False	True	1.81×10^3	9.55×10^{-1}
				True	True	1.82×10^3	9.66×10^{-1}
	duplicated-padding	zero	76224	False	False	1.43×10^3	4.93×10^{-1}
				True	False	1.76×10^3	1.14×10^{-1}
				False	True	1.77×10^3	4.94×10^{-1}
				True	True	1.82×10^3	4.86×10^{-1}
		linear	127040	False	False	1.51×10^3	6.59×10^{-1}
				True	False	1.79×10^3	4.96×10^{-1}
				False	True	1.82×10^3	7.28×10^{-1}
				True	True	1.85×10^3	7.87×10^{-1}
	lorr	zero	10324	False	False	1.47×10^3	6.59×10^{-1}
				True	False	1.75×10^3	6.60×10^{-1}
				False	True	1.75×10^3	6.58×10^{-1}
				True	True	1.83×10^3	6.59×10^{-1}
		linear	61140	False	False	1.54×10^3	9.76×10^{-1}
				True	False	1.78×10^3	9.76×10^{-1}
				False	True	1.81×10^3	9.77×10^{-1}
				True	True	1.91×10^3	9.75×10^{-1}
	hypercomplex	zero	76232	False	False	1.52×10^3	9.42×10^{-1}
				True	False	1.86×10^3	9.41×10^{-1}
				False	True	1.82×10^3	9.33×10^{-1}
				True	True	1.92×10^3	9.41×10^{-1}
		linear	127048	False	False	1.57×10^3	9.78×10^{-1}
				True	False	1.98×10^3	9.77×10^{-1}
				False	True	1.94×10^3	9.79×10^{-1}
				True	True	1.85×10^3	9.78×10^{-1}
	lphm	zero	5170	False	False	1.58×10^3	8.53×10^{-1}
				True	False	1.82×10^3	8.75×10^{-1}
				False	True	1.83×10^3	8.76×10^{-1}
				True	True	1.85×10^3	8.66×10^{-1}
		linear	55986	False	False	1.60×10^3	9.77×10^{-1}
				True	False	1.81×10^3	9.78×10^{-1}
				False	True	1.85×10^3	9.77×10^{-1}
				True	True	1.94×10^3	9.77×10^{-1}
	hypernet	zero	512	False	False	1.81×10^3	3.54×10^{-1}
				True	False	2.00×10^3	1.32×10^{-1}
				False	True	2.04×10^3	1.14×10^{-1}
				True	True	2.29×10^3	2.93×10^{-1}
		linear	51328	False	False	1.82×10^3	8.62×10^{-1}
				True	False	2.18×10^3	5.80×10^{-1}
				False	True	2.17×10^3	4.32×10^{-1}
				True	True	2.15×10^3	7.06×10^{-1}

Table 28: Performance analysis of RPN with the chi2 expansion function and different reconciliation and remainder functions on the MNIST dataset.

Expansion	Reconciliation	Remainder	Parameter#	Pre-Layer-Norm	Post-Layer-Norm	Time Cost	Accuracy
chi2	identity	zero	101632	False	False	1.36×10^3	9.72×10^{-1}
				True	False	1.32×10^3	9.57×10^{-1}
				False	True	1.31×10^3	9.75×10^{-1}
				True	True	1.28×10^3	9.75×10^{-1}
		linear	152448	False	False	1.26×10^3	9.74×10^{-1}
				True	False	1.30×10^3	9.72×10^{-1}
				False	True	1.30×10^3	9.76×10^{-1}
				True	True	1.33×10^3	9.76×10^{-1}
	masking	zero	101632	False	False	1.14×10^3	9.68×10^{-1}
				True	False	1.31×10^3	9.54×10^{-1}
				False	True	1.29×10^3	9.74×10^{-1}
				True	True	1.33×10^3	9.72×10^{-1}
		linear	152448	False	False	1.22×10^3	9.74×10^{-1}
				True	False	1.32×10^3	9.71×10^{-1}
				False	True	1.33×10^3	9.78×10^{-1}
				True	True	1.37×10^3	9.77×10^{-1}
	duplicated-padding	zero	25408	False	False	1.17×10^3	5.02×10^{-1}
				True	False	1.31×10^3	4.93×10^{-1}
				False	True	1.30×10^3	5.00×10^{-1}
				True	True	1.37×10^3	4.99×10^{-1}
		linear	76224	False	False	1.23×10^3	9.64×10^{-1}
				True	False	1.34×10^3	9.63×10^{-1}
				False	True	1.35×10^3	9.75×10^{-1}
				True	True	1.45×10^3	9.70×10^{-1}
	lorr	zero	3540	False	False	1.15×10^3	6.33×10^{-1}
				True	False	1.31×10^3	6.19×10^{-1}
				False	True	1.29×10^3	6.44×10^{-1}
				True	True	1.49×10^3	6.42×10^{-1}
		linear	54356	False	False	1.21×10^3	9.78×10^{-1}
				True	False	1.43×10^3	9.77×10^{-1}
				False	True	1.43×10^3	9.76×10^{-1}
				True	True	1.47×10^3	9.75×10^{-1}
	hypercomplex	zero	25416	False	False	1.18×10^3	9.33×10^{-1}
				True	False	1.45×10^3	9.22×10^{-1}
				False	True	1.40×10^3	9.46×10^{-1}
				True	True	1.34×10^3	9.46×10^{-1}
		linear	76232	False	False	1.23×10^3	9.78×10^{-1}
				True	False	1.43×10^3	9.77×10^{-1}
				False	True	1.41×10^3	9.77×10^{-1}
				True	True	1.35×10^3	9.78×10^{-1}
	lphm	zero	1778	False	False	1.23×10^3	6.29×10^{-1}
				True	False	1.29×10^3	6.29×10^{-1}
				False	True	1.31×10^3	6.31×10^{-1}
				True	True	1.48×10^3	6.93×10^{-1}
		linear	52594	False	False	1.26×10^3	9.77×10^{-1}
				True	False	1.32×10^3	9.77×10^{-1}
				False	True	1.35×10^3	9.76×10^{-1}
				True	True	1.59×10^3	9.76×10^{-1}
	hypernet	zero	512	False	False	1.28×10^3	1.59×10^{-1}
				True	False	1.50×10^3	2.06×10^{-1}
				False	True	1.46×10^3	3.95×10^{-1}
				True	True	1.55×10^3	3.53×10^{-1}
		linear	51328	False	False	1.40×10^3	7.39×10^{-1}
				True	False	1.49×10^3	9.01×10^{-1}
				False	True	1.46×10^3	8.25×10^{-1}
				True	True	1.45×10^3	8.41×10^{-1}

Table 29: Performance analysis of RPN with the gamma expansion function and different reconciliation and remainder functions on the MNIST dataset.

Expansion	Reconciliation	Remainder	Parameter#	Pre-Layer-Norm	Post-Layer-Norm	Time Cost	Accuracy
gamma	identity	zero	304896	False	False	1.58×10^3	9.55×10^{-1}
				True	False	1.65×10^3	9.32×10^{-1}
				False	True	1.73×10^3	9.47×10^{-1}
				True	True	1.59×10^3	9.14×10^{-1}
		linear	355712	False	False	1.71×10^3	9.61×10^{-1}
				True	False	1.67×10^3	9.45×10^{-1}
				False	True	1.75×10^3	9.59×10^{-1}
				True	True	1.64×10^3	9.53×10^{-1}
	masking	zero	304896	False	False	1.89×10^3	9.73×10^{-1}
				True	False	1.56×10^3	9.62×10^{-1}
				False	True	1.62×10^3	9.70×10^{-1}
				True	True	1.62×10^3	9.69×10^{-1}
		linear	355712	False	False	1.89×10^3	9.72×10^{-1}
				True	False	1.62×10^3	9.62×10^{-1}
				False	True	1.68×10^3	9.71×10^{-1}
				True	True	1.68×10^3	9.68×10^{-1}
	duplicated-padding	zero	76224	False	False	1.70×10^3	4.91×10^{-1}
				True	False	1.59×10^3	1.14×10^{-1}
				False	True	1.63×10^3	4.86×10^{-1}
				True	True	1.65×10^3	4.82×10^{-1}
		linear	127040	False	False	1.69×10^3	7.77×10^{-1}
				True	False	1.63×10^3	8.20×10^{-1}
				False	True	1.69×10^3	8.45×10^{-1}
				True	True	1.69×10^3	9.69×10^{-1}
	lorr	zero	10324	False	False	1.64×10^3	6.62×10^{-1}
				True	False	1.58×10^3	6.35×10^{-1}
				False	True	1.62×10^3	6.45×10^{-1}
				True	True	1.93×10^3	6.40×10^{-1}
		linear	61140	False	False	1.68×10^3	9.77×10^{-1}
				True	False	1.63×10^3	9.76×10^{-1}
				False	True	1.66×10^3	9.77×10^{-1}
				True	True	1.75×10^3	9.76×10^{-1}
	hypercomplex	zero	76232	False	False	1.65×10^3	9.44×10^{-1}
				True	False	2.00×10^3	9.39×10^{-1}
				False	True	2.05×10^3	9.52×10^{-1}
				True	True	1.42×10^3	9.49×10^{-1}
		linear	127048	False	False	1.71×10^3	9.79×10^{-1}
				True	False	1.59×10^3	9.79×10^{-1}
				False	True	1.53×10^3	9.78×10^{-1}
				True	True	1.46×10^3	9.78×10^{-1}
	lphm	zero	5170	False	False	1.70×10^3	8.67×10^{-1}
				True	False	1.40×10^3	8.66×10^{-1}
				False	True	1.43×10^3	7.69×10^{-1}
				True	True	1.60×10^3	7.40×10^{-1}
		linear	55986	False	False	1.73×10^3	9.76×10^{-1}
				True	False	1.44×10^3	9.78×10^{-1}
				False	True	1.50×10^3	9.78×10^{-1}
				True	True	1.73×10^3	9.78×10^{-1}
	hypernet	zero	512	False	False	1.81×10^3	1.96×10^{-1}
				True	False	1.87×10^3	2.11×10^{-1}
				False	True	1.83×10^3	3.31×10^{-1}
				True	True	2.03×10^3	3.30×10^{-1}
		linear	51328	False	False	1.80×10^3	8.51×10^{-1}
				True	False	1.93×10^3	7.62×10^{-1}
				False	True	1.95×10^3	7.49×10^{-1}
				True	True	1.93×10^3	7.64×10^{-1}

Table 30: Performance analysis of RPN with the laplace expansion function and different reconciliation and remainder functions on the MNIST dataset.

Expansion	Reconciliation	Remainder	Parameter#	Pre-Layer-Norm	Post-Layer-Norm	Time Cost	Accuracy
laplace	identity	zero	304896	False	False	1.15×10^3	9.65×10^{-1}
				True	False	7.45×10^4	9.77×10^{-1}
				False	True	7.91×10^4	9.77×10^{-1}
				True	True	7.94×10^4	9.80×10^{-1}
		linear	355712	False	False	1.10×10^3	9.68×10^{-1}
				True	False	1.51×10^3	8.88×10^{-1}
				False	True	1.54×10^3	9.40×10^{-1}
				True	True	1.53×10^3	9.57×10^{-1}
	masking	zero	304896	False	False	1.30×10^3	9.71×10^{-1}
				True	False	1.47×10^3	1.14×10^{-1}
				False	True	1.48×10^3	9.49×10^{-1}
				True	True	1.52×10^3	9.61×10^{-1}
		linear	355712	False	False	1.17×10^3	9.74×10^{-1}
				True	False	1.52×10^3	9.26×10^{-1}
				False	True	1.54×10^3	9.54×10^{-1}
				True	True	1.56×10^3	9.66×10^{-1}
	duplicated-padding	zero	76224	False	False	1.31×10^3	4.98×10^{-1}
				True	False	1.50×10^3	1.14×10^{-1}
				False	True	1.51×10^3	4.87×10^{-1}
				True	True	1.55×10^3	4.75×10^{-1}
		linear	127040	False	False	1.21×10^3	6.98×10^{-1}
				True	False	1.54×10^3	5.58×10^{-1}
				False	True	1.57×10^3	9.48×10^{-1}
				True	True	1.59×10^3	9.76×10^{-1}
	lorr	zero	10324	False	False	1.19×10^3	6.76×10^{-1}
				True	False	1.50×10^3	6.78×10^{-1}
				False	True	1.50×10^3	6.59×10^{-1}
				True	True	1.53×10^3	6.49×10^{-1}
		linear	61140	False	False	1.11×10^3	9.77×10^{-1}
				True	False	1.52×10^3	9.77×10^{-1}
				False	True	1.54×10^3	9.77×10^{-1}
				True	True	1.60×10^3	9.77×10^{-1}
	hypercomplex	zero	76232	False	False	1.14×10^3	9.31×10^{-1}
				True	False	1.56×10^3	9.33×10^{-1}
				False	True	1.57×10^3	9.37×10^{-1}
				True	True	1.65×10^3	9.43×10^{-1}
		linear	127048	False	False	7.30×10^2	9.80×10^{-1}
				True	False	1.67×10^3	9.78×10^{-1}
				False	True	1.71×10^3	9.78×10^{-1}
				True	True	1.77×10^3	9.80×10^{-1}
	lphm	zero	5170	False	False	7.34×10^2	8.71×10^{-1}
				True	False	1.75×10^3	8.65×10^{-1}
				False	True	1.75×10^3	8.69×10^{-1}
				True	True	1.72×10^3	8.76×10^{-1}
		linear	55986	False	False	7.46×10^2	9.77×10^{-1}
				True	False	1.74×10^3	9.77×10^{-1}
				False	True	1.76×10^3	9.78×10^{-1}
				True	True	1.87×10^3	9.77×10^{-1}
	hypernet	zero	512	False	False	1.63×10^3	4.64×10^{-1}
				True	False	1.89×10^3	1.16×10^{-1}
				False	True	1.86×10^3	1.07×10^{-1}
				True	True	1.95×10^3	3.07×10^{-1}
		linear	51328	False	False	1.63×10^3	9.08×10^{-1}
				True	False	1.88×10^3	4.78×10^{-1}
				False	True	1.86×10^3	3.58×10^{-1}
				True	True	1.88×10^3	7.86×10^{-1}

Table 31: Performance analysis of RPN with the exponential expansion function and different reconciliation and remainder functions on the MNIST dataset.

Expansion	Reconciliation	Remainder	Parameter#	Pre-Layer-Norm	Post-Layer-Norm	Time Cost	Accuracy
exponential	identity	zero	304896	False	False	5.36×10^4	9.53×10^{-1}
				True	False	1.36×10^3	9.54×10^{-1}
				False	True	1.30×10^3	9.61×10^{-1}
				True	True	1.16×10^3	9.54×10^{-1}
		linear	355712	False	False	3.28×10^4	9.57×10^{-1}
				True	False	1.37×10^3	9.56×10^{-1}
				False	True	1.42×10^3	9.61×10^{-1}
				True	True	1.21×10^3	9.60×10^{-1}
	masking	zero	304896	False	False	5.48×10^3	9.71×10^{-1}
				True	False	1.23×10^3	9.67×10^{-1}
				False	True	1.19×10^3	9.69×10^{-1}
				True	True	1.19×10^3	9.70×10^{-1}
		linear	355712	False	False	5.53×10^3	9.65×10^{-1}
				True	False	1.21×10^3	9.71×10^{-1}
				False	True	1.26×10^3	9.72×10^{-1}
				True	True	1.25×10^3	9.70×10^{-1}
	duplicated-padding	zero	76224	False	False	2.61×10^3	4.92×10^{-1}
				True	False	1.18×10^3	4.63×10^{-1}
				False	True	1.21×10^3	4.88×10^{-1}
				True	True	1.21×10^3	4.89×10^{-1}
		linear	127040	False	False	7.05×10^2	9.66×10^{-1}
				True	False	1.23×10^3	9.72×10^{-1}
				False	True	1.27×10^3	9.64×10^{-1}
				True	True	1.26×10^3	9.76×10^{-1}
	lorr	zero	10324	False	False	1.43×10^3	6.52×10^{-1}
				True	False	1.17×10^3	6.27×10^{-1}
				False	True	1.21×10^3	6.47×10^{-1}
				True	True	1.19×10^3	6.51×10^{-1}
		linear	61140	False	False	1.46×10^3	9.78×10^{-1}
				True	False	1.22×10^3	9.75×10^{-1}
				False	True	1.24×10^3	9.77×10^{-1}
				True	True	1.24×10^3	9.76×10^{-1}
	hypercomplex	zero	76232	False	False	1.40×10^3	9.38×10^{-1}
				True	False	1.17×10^3	9.35×10^{-1}
				False	True	1.20×10^3	9.36×10^{-1}
				True	True	1.21×10^3	9.42×10^{-1}
		linear	127048	False	False	1.45×10^3	9.77×10^{-1}
				True	False	1.22×10^3	9.78×10^{-1}
				False	True	1.26×10^3	9.78×10^{-1}
				True	True	1.31×10^3	9.77×10^{-1}
	lphm	zero	5170	False	False	1.45×10^3	6.33×10^{-1}
				True	False	1.21×10^3	6.43×10^{-1}
				False	True	1.23×10^3	8.19×10^{-1}
				True	True	1.05×10^3	8.00×10^{-1}
		linear	55986	False	False	1.49×10^3	9.78×10^{-1}
				True	False	1.24×10^3	9.78×10^{-1}
				False	True	1.27×10^3	9.77×10^{-1}
				True	True	1.05×10^3	9.78×10^{-1}
	hypernet	zero	512	False	False	1.58×10^3	3.44×10^{-1}
				True	False	1.57×10^3	3.09×10^{-1}
				False	True	1.57×10^3	3.20×10^{-1}
				True	True	1.63×10^3	3.43×10^{-1}
		linear	51328	False	False	1.62×10^3	8.37×10^{-1}
				True	False	1.60×10^3	7.84×10^{-1}
				False	True	1.66×10^3	7.57×10^{-1}
				True	True	1.54×10^3	7.75×10^{-1}

Table 32: Performance analysis of RPN with the hybrid-probabilistic expansion function and different reconciliation and remainder functions on the MNIST dataset.

Expansion	Reconciliation	Remainder	Parameter#	Pre-Layer-Norm	Post-Layer-Norm	Time Cost	Accuracy
hybrid-probabilistic	identity	zero	304896	False	False	1.47×10^3	9.67×10^{-1}
				True	False	2.26×10^3	8.08×10^{-1}
		linear	355712	False	True	2.17×10^3	9.53×10^{-1}
				True	True	2.39×10^3	9.42×10^{-1}
	masking	zero	304896	False	False	1.61×10^3	9.74×10^{-1}
				True	False	3.51×10^3	9.29×10^{-1}
				False	True	3.01×10^3	9.56×10^{-1}
				True	True	3.58×10^3	9.43×10^{-1}
	duplicated-padding	zero	76224	False	False	1.77×10^3	9.70×10^{-1}
				True	False	2.20×10^3	9.37×10^{-1}
				False	True	2.12×10^3	9.67×10^{-1}
				True	True	2.34×10^3	9.61×10^{-1}
	lorr	zero	10324	False	False	1.81×10^3	4.96×10^{-1}
				True	False	1.93×10^3	1.14×10^{-1}
				False	True	3.46×10^3	4.90×10^{-1}
				True	True	4.76×10^3	4.71×10^{-1}
	hypercomplex	zero	127040	False	False	1.77×10^3	9.61×10^{-1}
				True	False	4.40×10^3	6.23×10^{-1}
				False	True	3.84×10^3	8.93×10^{-1}
				True	True	3.78×10^3	6.92×10^{-1}
	lphm	zero	5170	False	False	1.62×10^3	6.69×10^{-1}
				True	False	3.87×10^3	6.58×10^{-1}
				False	True	3.54×10^3	6.55×10^{-1}
				True	True	3.91×10^3	7.03×10^{-1}
	hypernet	zero	76232	False	False	1.67×10^3	9.78×10^{-1}
				True	False	3.64×10^3	9.77×10^{-1}
				False	True	3.03×10^3	9.75×10^{-1}
				True	True	3.38×10^3	9.76×10^{-1}
	hypernet	linear	127048	False	False	1.62×10^3	9.39×10^{-1}
				True	False	3.46×10^3	9.42×10^{-1}
				False	True	2.50×10^3	9.47×10^{-1}
				True	True	3.62×10^3	9.48×10^{-1}
	hypernet	zero	512	False	False	1.68×10^3	9.80×10^{-1}
				True	False	5.08×10^3	9.78×10^{-1}
				False	True	3.19×10^3	9.78×10^{-1}
				True	True	5.16×10^3	9.77×10^{-1}
	hypernet	linear	55986	False	False	1.66×10^3	8.11×10^{-1}
				True	False	3.38×10^3	8.47×10^{-1}
				False	True	2.40×10^3	8.43×10^{-1}
				True	True	9.21×10^3	8.20×10^{-1}
	hypernet	zero	512	False	False	1.67×10^3	9.77×10^{-1}
				True	False	9.92×10^3	9.77×10^{-1}
				False	True	7.77×10^3	9.76×10^{-1}
				True	True	1.06×10^4	9.78×10^{-1}
	hypernet	linear	51328	False	False	1.98×10^3	3.66×10^{-1}
				True	False	2.22×10^3	2.81×10^{-1}
				False	True	2.33×10^3	3.38×10^{-1}
				True	True	2.29×10^3	2.91×10^{-1}
	hypernet	zero	512	False	False	1.83×10^3	9.16×10^{-1}
				True	False	3.47×10^3	4.84×10^{-1}
				False	True	3.50×10^3	5.74×10^{-1}
				True	True	3.64×10^3	4.82×10^{-1}

Table 33: Performance analysis of RPN with the normal-combinatorial expansion function and different reconciliation and remainder functions on the MNIST dataset.

Expansion	Reconciliation	Remainder	Parameter#	Pre-Layer-Norm	Post-Layer-Norm	Time Cost	Accuracy
normal-combinatorial	identity	zero	19714880	False	False	4.31×10^4	9.49×10^{-1}
				True	False	4.30×10^4	1.14×10^{-1}
				False	True	7.14×10^4	1.00×10^{-1}
				True	True	7.13×10^4	6.96×10^{-1}
		linear	19765696	False	False	4.30×10^4	9.54×10^{-1}
				True	False	4.30×10^4	1.14×10^{-1}
				False	True	7.12×10^4	1.14×10^{-1}
				True	True	7.15×10^4	8.35×10^{-1}
	masking	zero	19714880	False	False	4.36×10^4	9.53×10^{-1}
				True	False	4.39×10^4	1.14×10^{-1}
				False	True	4.30×10^4	1.02×10^{-1}
				True	True	4.30×10^4	7.94×10^{-1}
		linear	19765696	False	False	4.33×10^4	9.53×10^{-1}
				True	False	4.26×10^4	1.14×10^{-1}
				False	True	4.29×10^4	1.14×10^{-1}
				True	True	4.31×10^4	8.56×10^{-1}
	duplicated-padding	zero	4928720	False	False	4.86×10^4	4.88×10^{-1}
				True	False	4.87×10^4	1.14×10^{-1}
				False	True	4.75×10^4	9.91×10^{-2}
				True	True	4.76×10^4	4.14×10^{-1}
		linear	4979536	False	False	4.86×10^4	7.53×10^{-1}
				True	False	4.86×10^4	1.14×10^{-1}
				False	True	4.75×10^4	1.07×10^{-1}
				True	True	4.76×10^4	4.67×10^{-1}
	lorr	zero	619748	False	False	4.26×10^4	6.05×10^{-1}
				True	False	4.26×10^4	5.15×10^{-1}
				False	True	5.79×10^4	7.41×10^{-1}
				True	True	5.80×10^4	7.82×10^{-1}
		linear	670564	False	False	4.25×10^4	9.75×10^{-1}
				True	False	4.24×10^4	5.88×10^{-1}
				False	True	5.78×10^4	9.71×10^{-1}
				True	True	5.80×10^4	9.75×10^{-1}
	hypercomplex	zero	4928728	False	False	3.23×10^4	1.14×10^{-1}
				True	False	2.90×10^4	1.14×10^{-1}
				False	True	6.04×10^4	9.20×10^{-1}
				True	True	6.04×10^4	7.88×10^{-1}
		linear	4979544	False	False	2.49×10^4	9.81×10^{-1}
				True	False	3.37×10^4	9.60×10^{-1}
				False	True	6.03×10^4	9.78×10^{-1}
				True	True	6.03×10^4	9.77×10^{-1}
	lphm	zero	309882	False	False	2.49×10^4	1.14×10^{-1}
				True	False	3.34×10^4	7.39×10^{-1}
				False	True	6.18×10^4	7.76×10^{-1}
				True	True	6.18×10^4	7.75×10^{-1}
		linear	360698	False	False	3.01×10^4	9.78×10^{-1}
				True	False	2.81×10^4	9.63×10^{-1}
				False	True	6.20×10^4	9.76×10^{-1}
				True	True	6.18×10^4	9.74×10^{-1}
	hypernet	zero	512	False	False	3.58×10^4	9.29×10^{-2}
				True	False	4.27×10^4	1.13×10^{-1}
				False	True	6.53×10^4	1.03×10^{-1}
				True	True	6.53×10^4	1.44×10^{-1}
		linear	51328	False	False	3.60×10^4	9.14×10^{-2}
				True	False	4.30×10^4	8.27×10^{-1}
				False	True	6.54×10^4	1.43×10^{-1}
				True	True	6.53×10^4	4.33×10^{-1}

A.3 Visualization of RPN Data Expansion and Parameter Reconciliation on MNIST Dataset

Descriptions: The following Figures 23-31 present the visualizations of the expanded images with labels 1 through 9. Besides the image data, we also illustrate the learned parameters of RPN corresponding to these expansions as well.

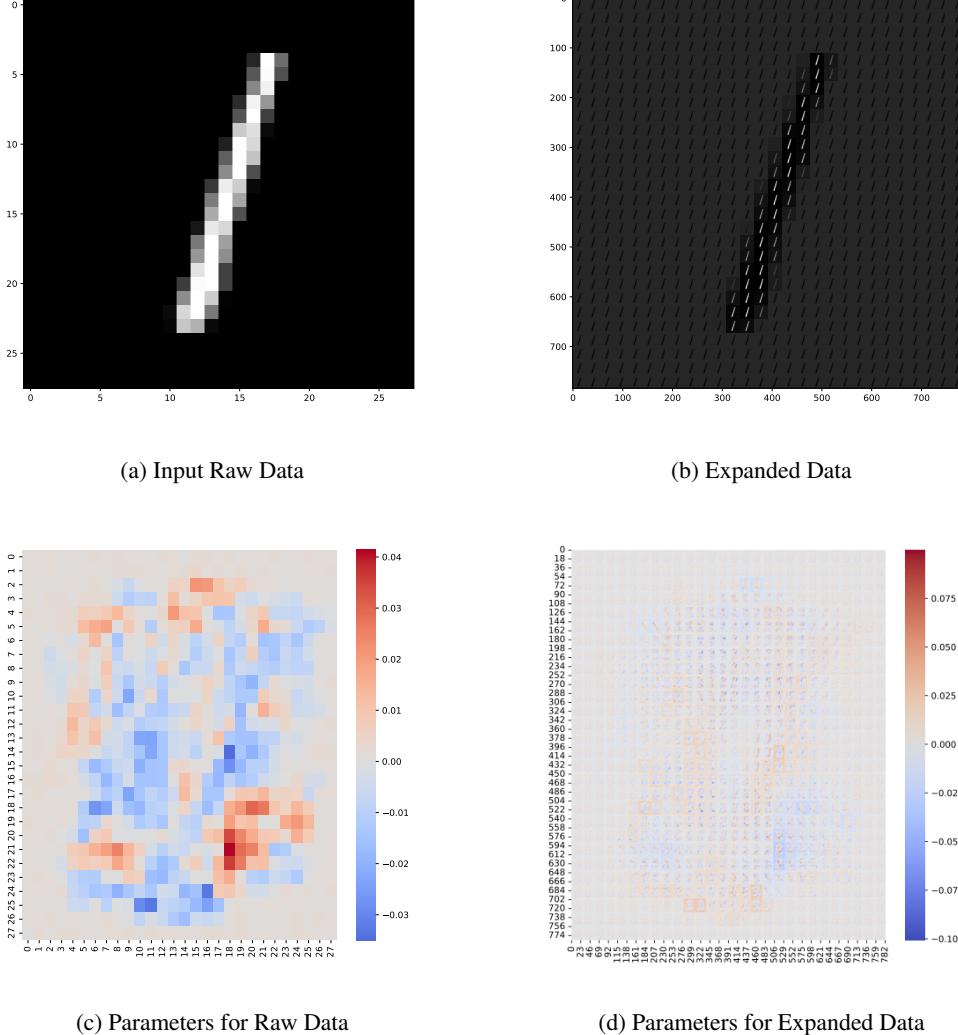


Figure 23: An illustration of an image with label 1 randomly selected from the MNIST dataset. Plot (a): raw image data; Plot (b): expanded data; Plot (c): parameter corresponding to output neuron of label 1 for raw image data; and Plot (d): parameter corresponding to output neuron of label 1 for expanded image data.

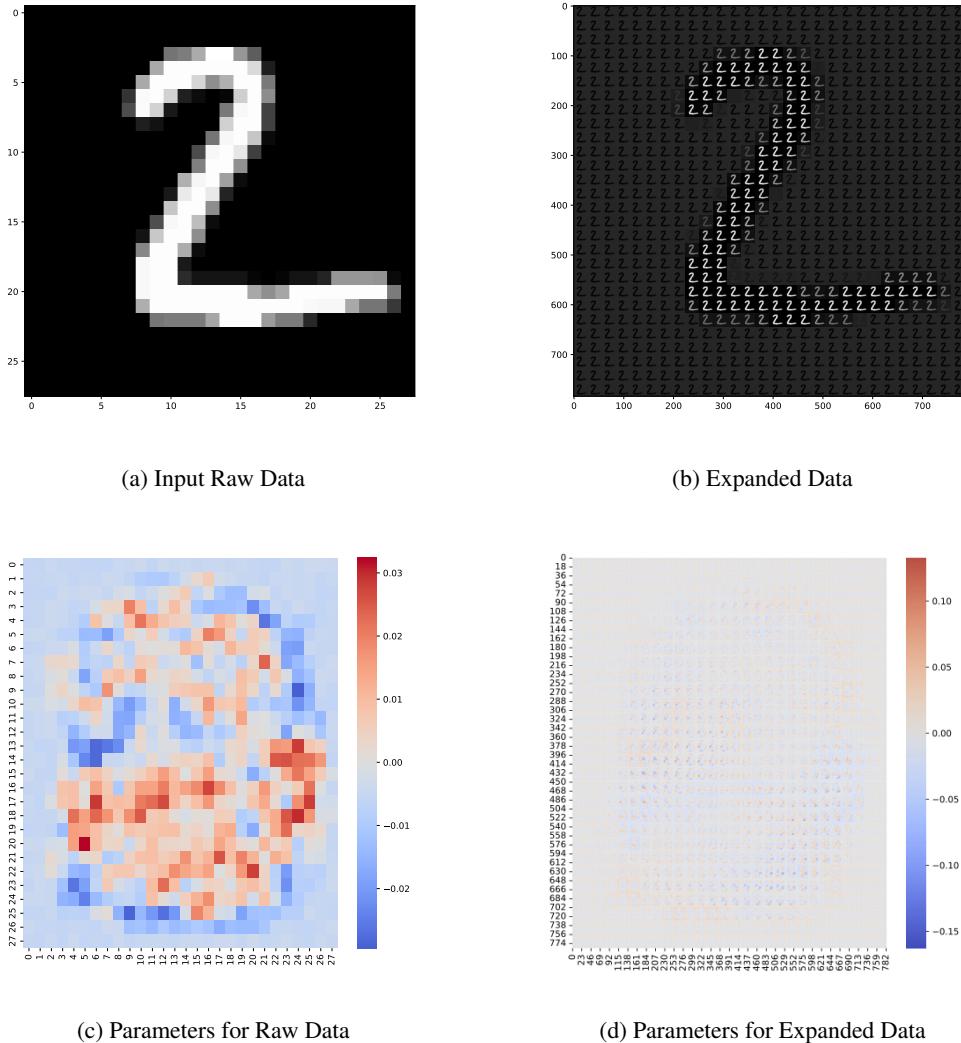
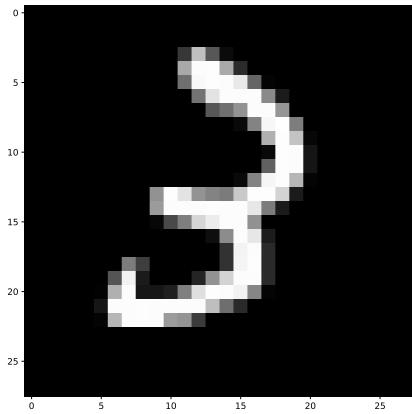
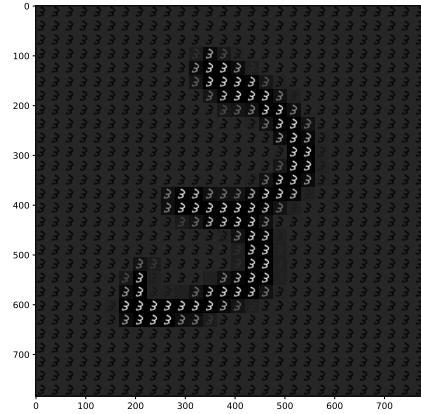


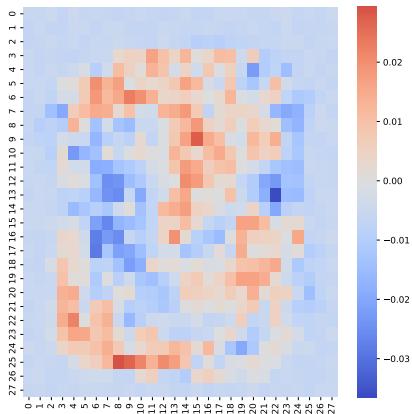
Figure 24: An illustration of an image with label 2 randomly selected from the MNIST dataset. Plot (a): raw image data; Plot (b): expanded data; Plot (c): parameter corresponding to output neuron of label 2 for raw image data; and Plot (d): parameter corresponding to output neuron of label 2 for expanded image data.



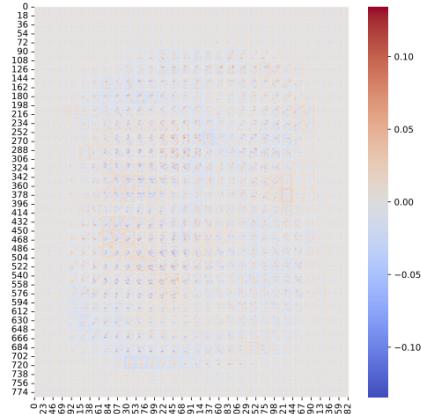
(a) Input Raw Data



(b) Expanded Data



(c) Parameters for Raw Data



(d) Parameters for Expanded Data

Figure 25: An illustration of an image with label 3 randomly selected from the MNIST dataset. Plot (a): raw image data; Plot (b): expanded data; Plot (c): parameter corresponding to output neuron of label 3 for raw image data; and Plot (d): parameter corresponding to output neuron of label 3 for expanded image data.

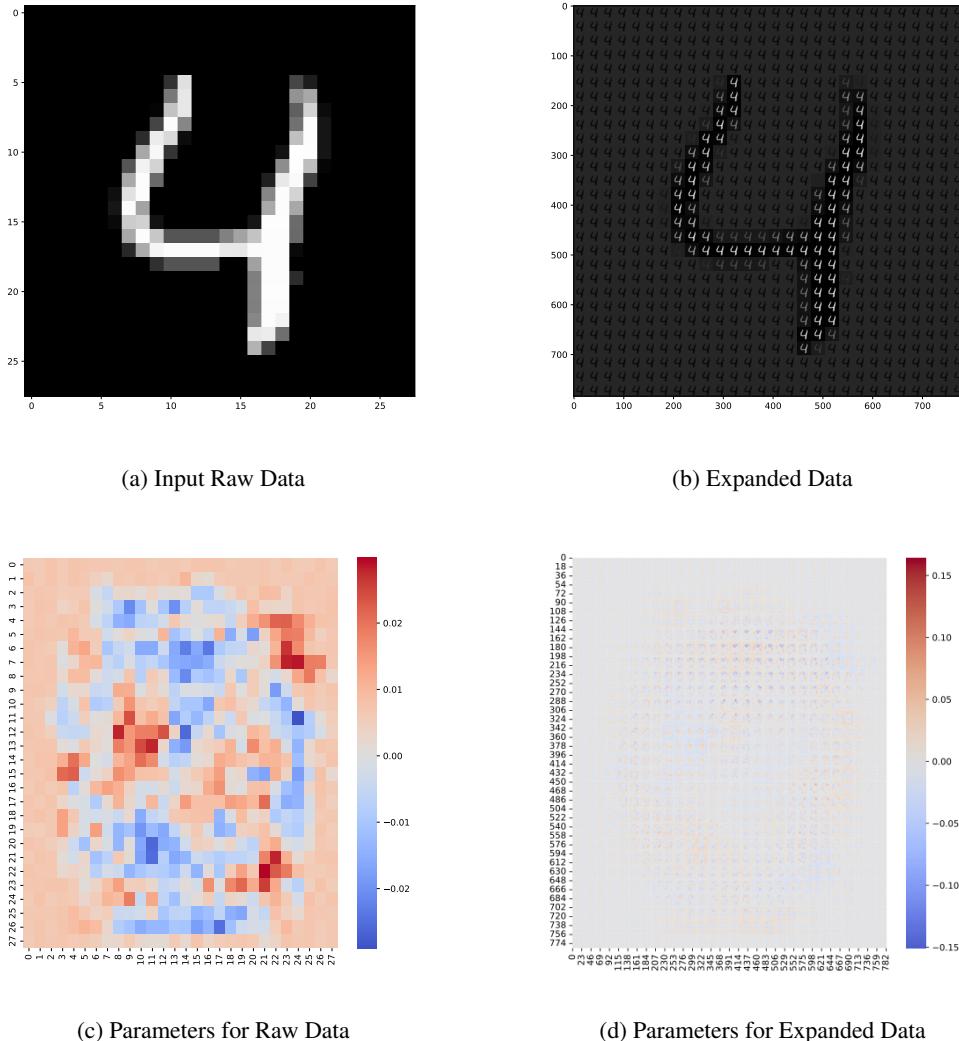
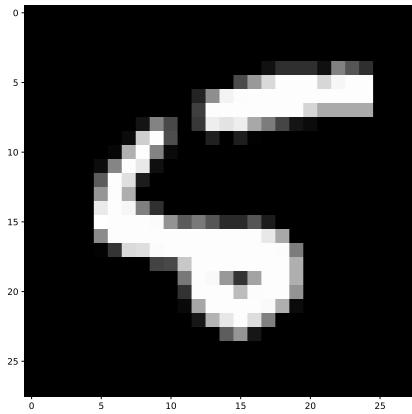
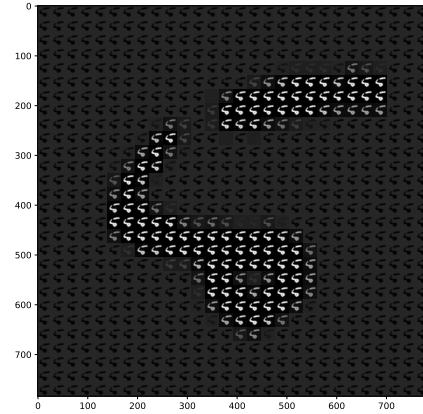


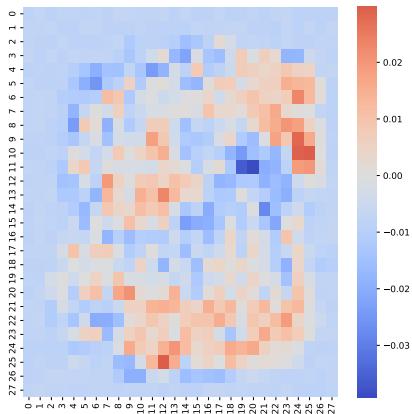
Figure 26: An illustration of an image with label 4 randomly selected from the MNIST dataset. Plot (a): raw image data; Plot (b): expanded data; Plot (c): parameter corresponding to output neuron of label 4 for raw image data; and Plot (d): parameter corresponding to output neuron of label 4 for expanded image data.



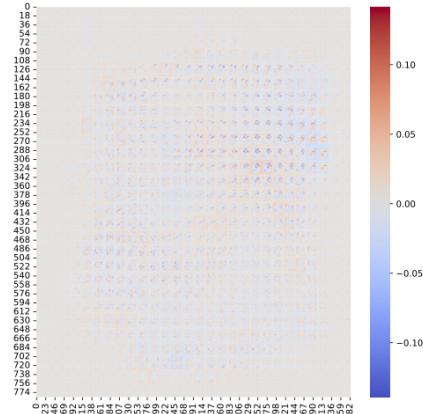
(a) Input Raw Data



(b) Expanded Data



(c) Parameters for Raw Data



(d) Parameters for Expanded Data

Figure 27: An illustration of an image with label 5 randomly selected from the MNIST dataset. Plot (a): raw image data; Plot (b): expanded data; Plot (c): parameter corresponding to output neuron of label 5 for raw image data; and Plot (d): parameter corresponding to output neuron of label 5 for expanded image data.

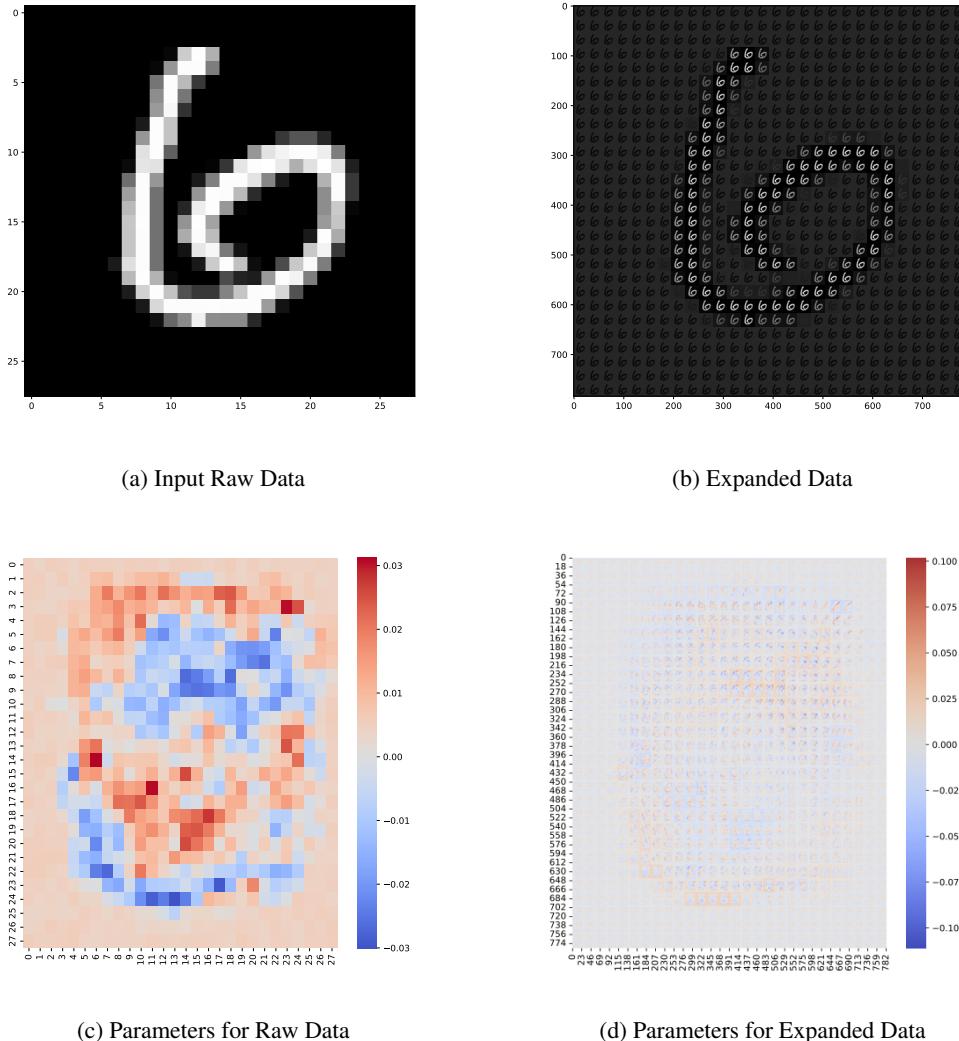
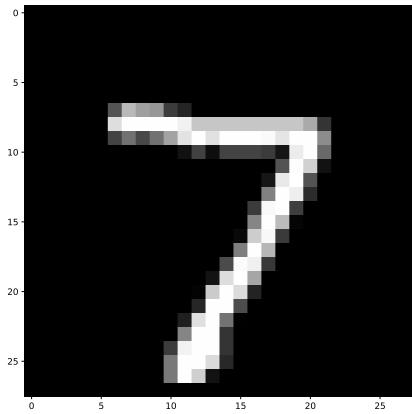
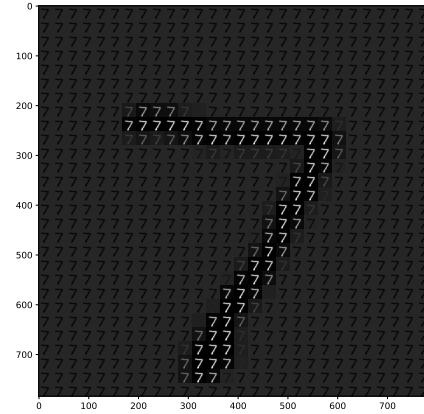


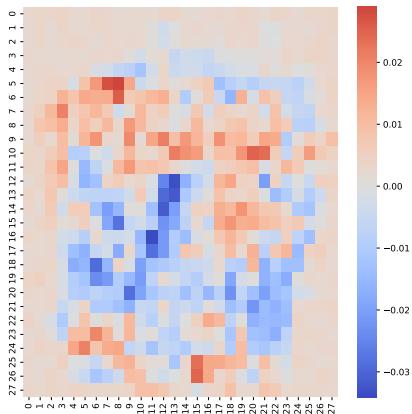
Figure 28: An illustration of an image with label 6 randomly selected from the MNIST dataset. Plot (a): raw image data; Plot (b): expanded data; Plot (c): parameter corresponding to output neuron of label 6 for raw image data; and Plot (d): parameter corresponding to output neuron of label 6 for expanded image data.



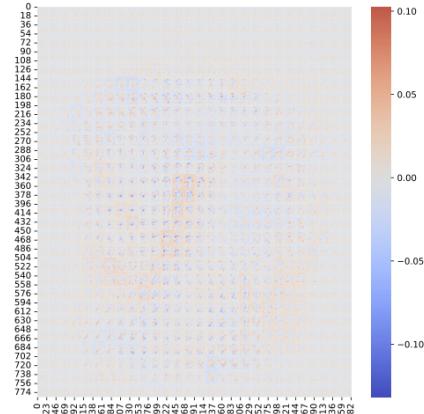
(a) Input Raw Data



(b) Expanded Data

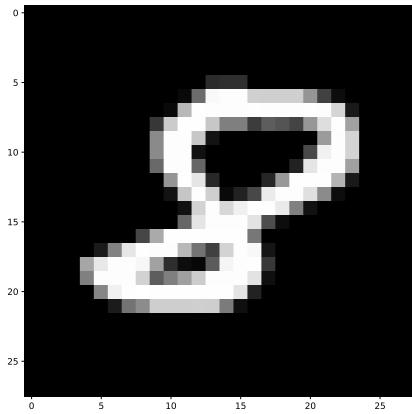


(c) Parameters for Raw Data

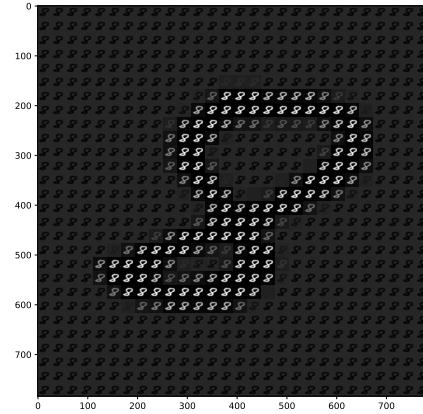


(d) Parameters for Expanded Data

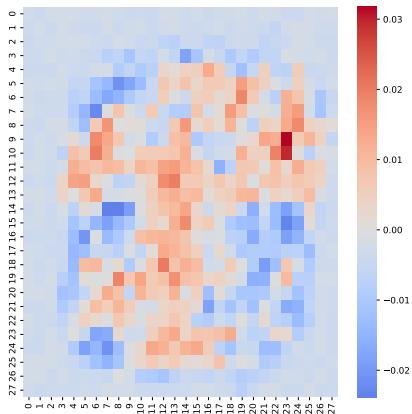
Figure 29: An illustration of an image with label 7 randomly selected from the MNIST dataset. Plot (a): raw image data; Plot (b): expanded data; Plot (c): parameter corresponding to output neuron of label 7 for raw image data; and Plot (d): parameter corresponding to output neuron of label 7 for expanded image data.



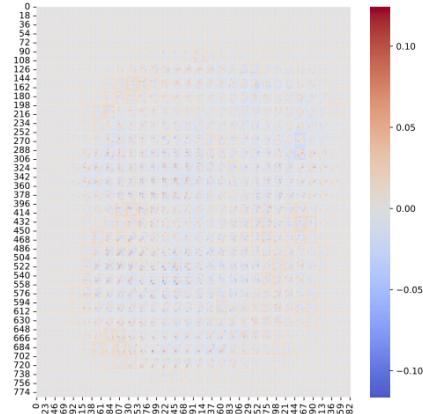
(a) Input Raw Data



(b) Expanded Data

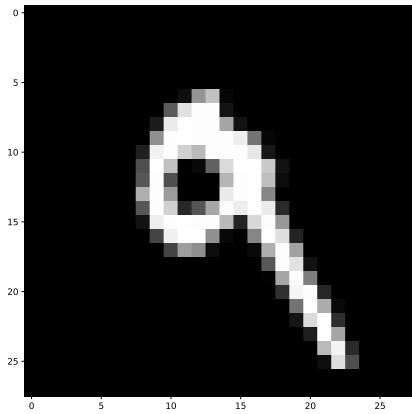


(c) Parameters for Raw Data

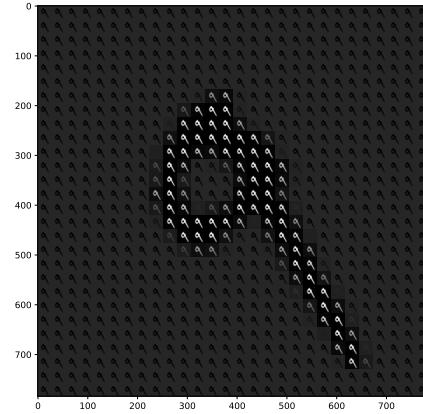


(d) Parameters for Expanded Data

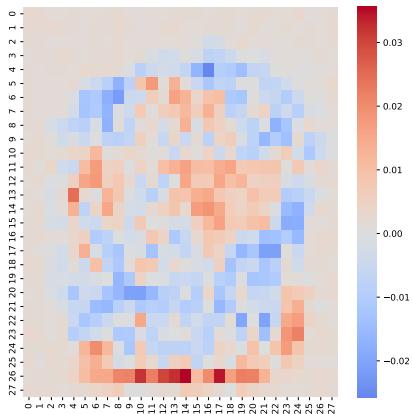
Figure 30: An illustration of an image with label 8 randomly selected from the MNIST dataset. Plot (a): raw image data; Plot (b): expanded data; Plot (c): parameter corresponding to output neuron of label 8 for raw image data; and Plot (d): parameter corresponding to output neuron of label 8 for expanded image data.



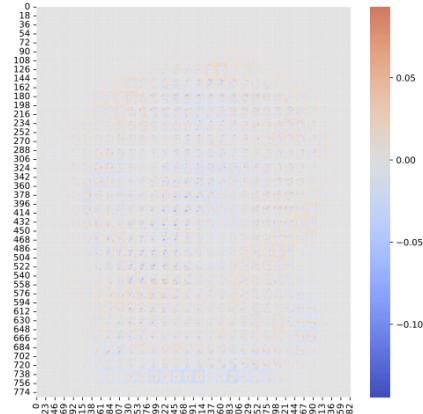
(a) Input Raw Data



(b) Expanded Data



(c) Parameters for Raw Data



(d) Parameters for Expanded Data

Figure 31: An illustration of an image with label 9 randomly selected from the MNIST dataset. Plot (a): raw image data; Plot (b): expanded data; Plot (c): parameter corresponding to output neuron of label 9 for raw image data; and Plot (d): parameter corresponding to output neuron of label 9 for expanded image data.

A.4 Licensing Rights of Using BioRender Created Contents in This Paper

Descriptions: For the bio-medical image contents used in the previous Figures 21-22 in Section 8, we have obtained the permissions to use them in publications. The licensing rights granting confirmation letters from BioRender for using these generated contents are attached in the following pages.

Confirmation of Publication and Licensing Rights

May 15th, 2024
Science Suite Inc.

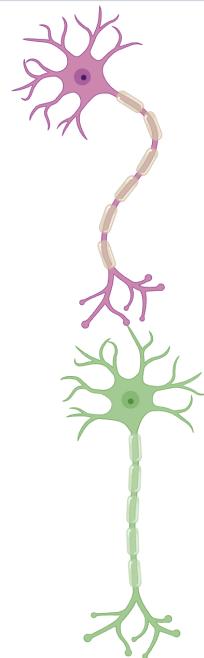
Subscription: *Institution*
Agreement number: AN26TMJDKE
Journal name: arxiv

To whom this may concern,

This document is to confirm that Jiawei Zhang has been granted a license to use the BioRender content, including icons, templates and other original artwork, appearing in the attached completed graphic pursuant to BioRender's [Academic License Terms](#). This license permits BioRender content to be sublicensed for use in journal publications.

All rights and ownership of BioRender content are reserved by BioRender. All completed graphics must be accompanied by the following citation: "Created with BioRender.com".

BioRender content included in the completed graphic is not licensed for any commercial uses beyond publication in a journal. For any commercial use of this figure, users may, if allowed, recreate it in BioRender under an Industry BioRender Plan.



For any questions regarding this document, or other questions about publishing with BioRender refer to our [BioRender Publication Guide](#), or contact BioRender Support at support@biorender.com.

Confirmation of Publication and Licensing Rights

May 15th, 2024
Science Suite Inc.

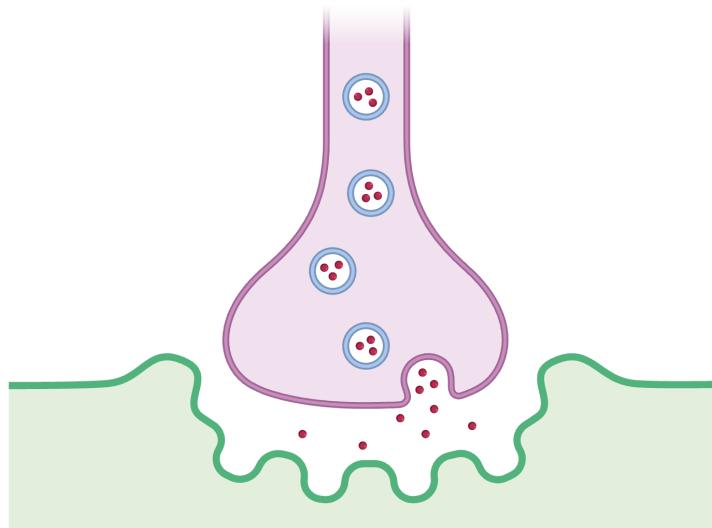
Subscription: *Institution*
Agreement number: *FK26TMK5JY*
Journal name: *arxiv*

To whom this may concern,

This document is to confirm that Jiawei Zhang has been granted a license to use the BioRender content, including icons, templates and other original artwork, appearing in the attached completed graphic pursuant to BioRender's [Academic License Terms](#). This license permits BioRender content to be sublicensed for use in journal publications.

All rights and ownership of BioRender content are reserved by BioRender. All completed graphics must be accompanied by the following citation: "Created with BioRender.com".

BioRender content included in the completed graphic is not licensed for any commercial uses beyond publication in a journal. For any commercial use of this figure, users may, if allowed, recreate it in BioRender under an Industry BioRender Plan.



For any questions regarding this document, or other questions about publishing with BioRender refer to our [BioRender Publication Guide](#), or contact BioRender Support at support@biorender.com.

Confirmation of Publication and Licensing Rights

May 15th, 2024
Science Suite Inc.

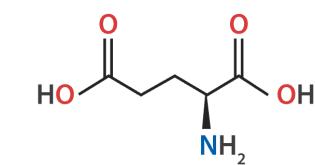
Subscription: Institution
Agreement number: BD26TMLQQI
Journal name: arxiv

To whom this may concern,

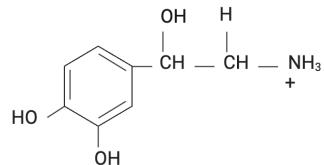
This document is to confirm that Jiawei Zhang has been granted a license to use the BioRender content, including icons, templates and other original artwork, appearing in the attached completed graphic pursuant to BioRender's [Academic License Terms](#). This license permits BioRender content to be sublicensed for use in journal publications.

All rights and ownership of BioRender content are reserved by BioRender. All completed graphics must be accompanied by the following citation: "Created with BioRender.com".

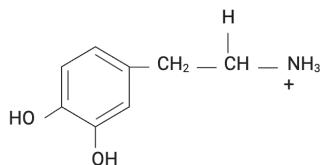
BioRender content included in the completed graphic is not licensed for any commercial uses beyond publication in a journal. For any commercial use of this figure, users may, if allowed, recreate it in BioRender under an Industry BioRender Plan.



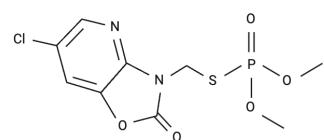
glutamate



dopamine



norepinephrine



acetylcholine

For any questions regarding this document, or other questions about publishing with BioRender refer to our [BioRender Publication Guide](#), or contact BioRender Support at support@biorender.com.

Confirmation of Publication and Licensing Rights

May 15th, 2024
Science Suite Inc.

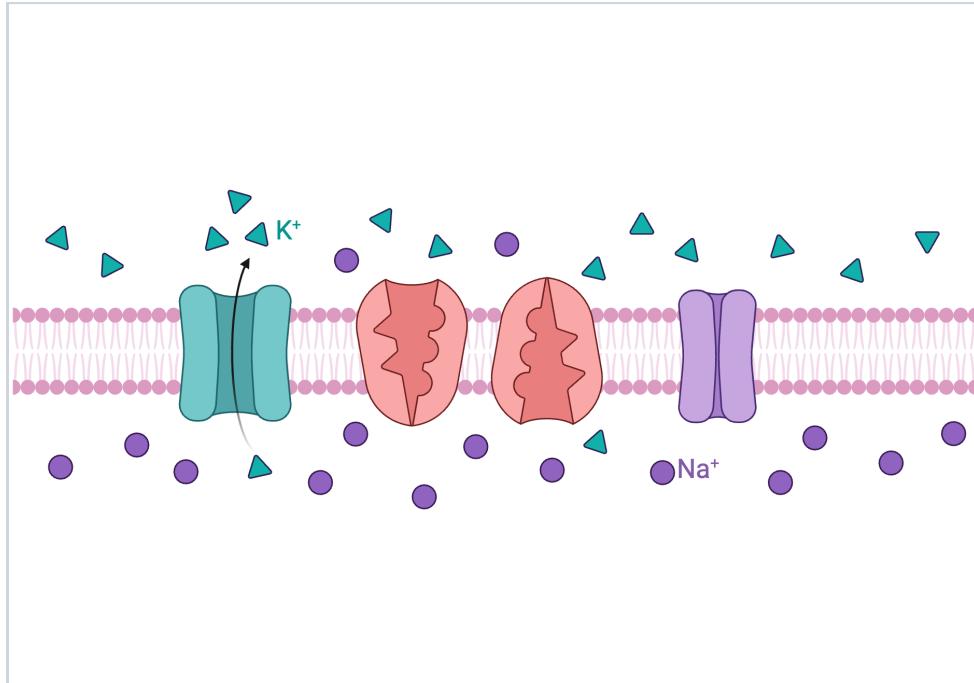
Subscription: Institution
Agreement number: WA26TNN08G
Journal name: arxiv

To whom this may concern,

This document is to confirm that Jiawei Zhang has been granted a license to use the BioRender content, including icons, templates and other original artwork, appearing in the attached completed graphic pursuant to BioRender's [Academic License Terms](#). This license permits BioRender content to be sublicensed for use in journal publications.

All rights and ownership of BioRender content are reserved by BioRender. All completed graphics must be accompanied by the following citation: "Created with BioRender.com".

BioRender content included in the completed graphic is not licensed for any commercial uses beyond publication in a journal. For any commercial use of this figure, users may, if allowed, recreate it in BioRender under an Industry BioRender Plan.



For any questions regarding this document, or other questions about publishing with BioRender refer to our [BioRender Publication Guide](#), or contact BioRender Support at support@biorender.com.

Confirmation of Publication and Licensing Rights

May 15th, 2024
Science Suite Inc.

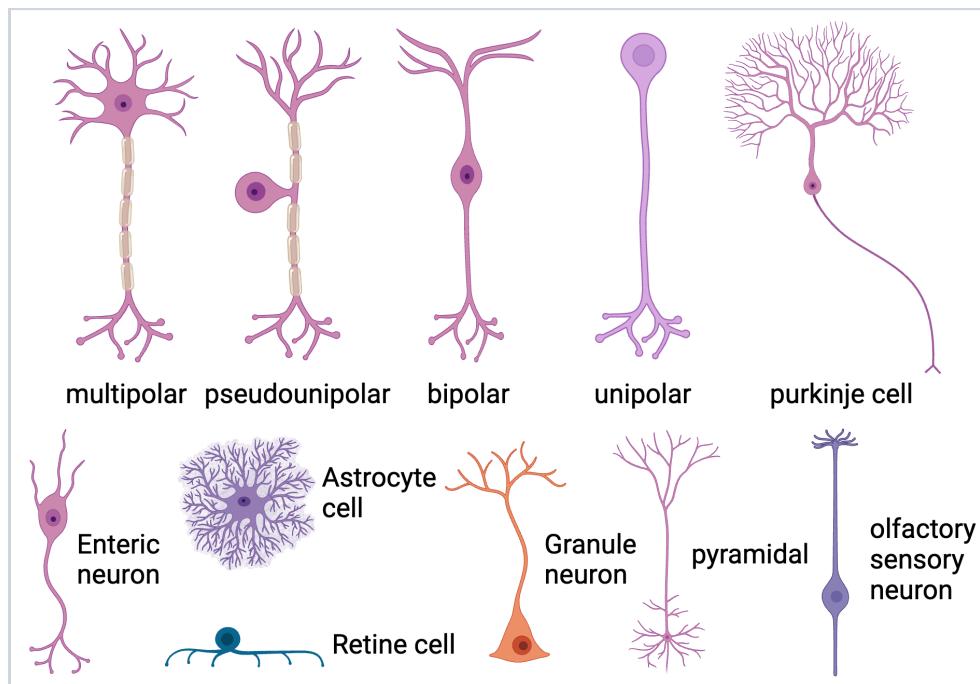
Subscription: *Institution*
Agreement number: LJ26TOMY8Y
Journal name: arxiv

To whom this may concern,

This document is to confirm that Jiawei Zhang has been granted a license to use the BioRender content, including icons, templates and other original artwork, appearing in the attached completed graphic pursuant to BioRender's [Academic License Terms](#). This license permits BioRender content to be sublicensed for use in journal publications.

All rights and ownership of BioRender content are reserved by BioRender. All completed graphics must be accompanied by the following citation: "Created with BioRender.com".

BioRender content included in the completed graphic is not licensed for any commercial uses beyond publication in a journal. For any commercial use of this figure, users may, if allowed, recreate it in BioRender under an Industry BioRender Plan.



For any questions regarding this document, or other questions about publishing with BioRender refer to our [BioRender Publication Guide](#), or contact BioRender Support at support@biorender.com.