

# Progressive Probabilistic Circuits

## Progressive probabilistische Schaltkreise

Masterarbeit von Uranik Berisha

Tag der Einreichung: 26. Mai 2023

Prüfer: Dr. Martin Mundt

Betreuer: M.Sc. Steven Braun

Darmstadt



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Fachbereich Informatik  
Artificial Intelligence and  
Machine Learning Lab  
(AIML)

---

# Zusammenfassung

---

In dieser Thesis untersuchen wir probabilistische Schaltkreise, speziell Summen-Produkt-Netzwerke (SPNs) und ihre potenzielle Anwendung in kontinuierlichen Lernszenarien.

Probabilistische Schaltkreise einschließlich Summen-Produkt-Netzwerke (SPNs) sind probabilistische Modelle, die in dem maschinellen Lernen für die Dichteschätzung von Daten verwendet werden. Sie funktionieren, indem sie Verteilungen durch Mischungen und Faktorisierungen kombinieren und dadurch komplexere Verteilungen erschaffen. Sie halten strukturelle Einschränkungen ein, die speziell darauf ausgelegt sind, um eine effiziente Inferenz von häufig verwendeten probabilistischen Abfragen zu ermöglichen. Diese Modelle haben sich als effiziente und interpretierbare Alternativen zu herkömmlichen Deep-Learning-Modellen in zahlreichen Aufgaben wie Klassifikationen gezeigt. Trotz ihrer nachgewiesenen Fähigkeiten bleibt die Anwendung von SPNs in kontinuierlichen Lernszenarien jedoch ein unerforschtes Gebiet.

Unsere Forschung zielt darauf ab, diese Lücke zu schließen, indem wir Modifikationen an der SPN-Struktur vornehmen, die von dem architektonischen Ansatz der progressiven neuronalen Netzwerke (PNNs) inspiriert sind, wo für jede Aufgabe ein neues Modell trainiert wird mit lateralen Verbindungen zwischen den Modellen, welche einen Wissenstransfer ermöglichen. Diese Modifikationen führen zu Progressiven Summen-Produkt-Netzwerken (PSPNs), die dazu entworfen sind, die Weitergabe von Wissen zu erleichtern und katastrophales Vergessen zu umgehen, zwei signifikante Herausforderungen im kontinuierlichen Lernen. Wir beweisen, dass unser Ansatz die strukturellen Einschränkungen der SPNs beibehält und so die Effizienz des Modells gewährleistet.

Darüber hinaus führen wir eine Reihe von Experimenten durch, um die Leistung unseres Modells im Vergleich zu einem standardmäßigen SPN zu bewerten. Die empirischen Ergebnisse zeigen, dass unser progressiver Wachstumsansatz, obwohl er bei PNNs erfolgreich ist, nicht ohne Weiteres auf SPNs übertragen werden kann. Trotz geringer Leistungsverbesserungen hinsichtlich Konvergenz und Klassifikationsgeschwindigkeit beobachtet wurden, benötigen die PSPNs zusätzliche optimierte Parameterinitialisierungen, um ein instabiles Training zu vermeiden. Wir führen diese Herausforderungen auf die Unterschiede zwischen Merkmal-transformierenden neuronalen Netzwerken und dichte-schätzenden SPNs zurück. Während Merkmale transformiert werden können, können Verteilungen nur neu kombiniert oder wiederverwendet werden, was die Weitergabe von Wissen einschränkt.

Insgesamt kommen wir zu dem Schluss, dass der progressive SPN in seiner jetzigen Form zu viele Einschränkungen aufweist, um in einem kontinuierlichen Lernszenario praktisch genutzt werden zu können. Nichtsdestotrotz dient diese Arbeit als Sprungbrett für zukünftige Forschung, um diese Herausforderungen zu bewältigen.

---

# Abstract

---

In this thesis, we explore probabilistic circuits, specifically Sum-Product Networks (SPNs), and their potential applications in continual learning scenarios.

Probabilistic circuits, including SPNs, are probabilistic models used in machine learning for data density estimation, which operate by combining distributions through mixtures and factorizations, thereby creating more complex ones. They maintain structural constraints specifically designed to enable tractable inference of commonly used probabilistic queries. These models have proven to serve as efficient and interpretable alternatives to conventional deep learning models in numerous tasks, such as classifications. Yet, despite their proven capabilities, the application of SPNs in continual learning settings remains an unexplored territory.

Our research aims to bridge this gap by introducing modifications to the SPN structure, inspired by the architectural approach of progressive neural networks (PNNs), where a new model is trained for each task with lateral connections between the models to allow for knowledge transfer. These modifications give rise to Progressive Sum-Product Networks (PSPNs), designed to facilitate forward transfer and circumvent catastrophic forgetting, two significant challenges in continual learning. We prove that our approach maintains the structural constraints of SPNs, thereby ensuring the model's tractability.

Furthermore, we conduct a series of experiments to evaluate the performance of our model against a standard SPN baseline. The empirical results reveal that the progressive growing approach, while successful in PNNs, does not translate seamlessly to SPNs. Although minor performance improvements were observed in terms of convergence and classification speed, the PSPNs require additional optimized parameter initializations to avoid unstable training. We attribute these challenges to the inherent differences between feature-transforming neural networks and density-estimating SPNs. While features can be transformed, distributions can only be recombined or reused, limiting the forward transfer.

Altogether we conclude, the progressive modeling approach, in its current form, presents too many limitations to be considered viable in a continual learning setting. However, this research serves as a stepping stone for future work to mitigate these challenges.

---

# Contents

---

|          |                                                                       |           |
|----------|-----------------------------------------------------------------------|-----------|
| <b>1</b> | <b>Introduction</b>                                                   | <b>1</b>  |
| 1.1      | Motivation . . . . .                                                  | 2         |
| 1.2      | Objectives . . . . .                                                  | 3         |
| 1.3      | Structure . . . . .                                                   | 3         |
| <b>2</b> | <b>Related Work</b>                                                   | <b>4</b>  |
| <b>3</b> | <b>Background</b>                                                     | <b>6</b>  |
| 3.1      | Probabilistic Circuits and Sum-Product Networks . . . . .             | 6         |
| 3.1.1    | Formal Definitions . . . . .                                          | 9         |
| 3.1.2    | Constraints . . . . .                                                 | 16        |
| 3.2      | Continual Learning . . . . .                                          | 19        |
| 3.3      | Progressive Neural Networks . . . . .                                 | 23        |
| <b>4</b> | <b>Method: Progressive Probabilistic Circuits</b>                     | <b>24</b> |
| 4.1      | Specification of the Setting . . . . .                                | 24        |
| 4.2      | Progressive Sum-Product Network . . . . .                             | 25        |
| 4.2.1    | Modeling . . . . .                                                    | 25        |
| 4.2.2    | Validity . . . . .                                                    | 28        |
| 4.2.3    | Proof of Validity for Progressive Sum-Product Networks . . . . .      | 29        |
| 4.2.4    | Leaf-Node Transformation . . . . .                                    | 30        |
| 4.2.5    | Training . . . . .                                                    | 31        |
| <b>5</b> | <b>Experiments: Exploration of Progressive Probabilistic Circuits</b> | <b>33</b> |
| 5.1      | General Evaluation Methods . . . . .                                  | 34        |
| 5.1.1    | Performance Evaluation . . . . .                                      | 34        |
| 5.1.2    | Transfer Evaluation . . . . .                                         | 34        |
| 5.2      | Exploration on Synthetic Data . . . . .                               | 35        |
| 5.2.1    | Setup . . . . .                                                       | 35        |
| 5.2.2    | Proof-of-Concept on Synthetic Data . . . . .                          | 40        |
| 5.2.3    | Transforming Leaf-Nodes . . . . .                                     | 44        |
| 5.2.4    | Copying Leaf-Nodes . . . . .                                          | 46        |
| 5.2.5    | Summary . . . . .                                                     | 48        |
| 5.3      | Exploration on Image Data . . . . .                                   | 49        |
| 5.3.1    | Setup . . . . .                                                       | 49        |
| 5.3.2    | Evaluation on Image Data . . . . .                                    | 51        |
| 5.3.3    | Parameter Search on PSPNs . . . . .                                   | 53        |
| 5.3.4    | Parameter Search on SPNs . . . . .                                    | 57        |
| 5.3.5    | Summary . . . . .                                                     | 59        |



|                                        |           |
|----------------------------------------|-----------|
| 5.4 Overall Analysis . . . . .         | 60        |
| <b>6 Conclusion</b>                    | <b>62</b> |
| <b>7 Limitations &amp; Future Work</b> | <b>63</b> |

---

## List of Figures

---

|      |                                                                                                                                                                                                |    |
|------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 3.1  | Overview of the background topics . . . . .                                                                                                                                                    | 7  |
| 3.2  | Example of a simple SPN . . . . .                                                                                                                                                              | 8  |
| 3.3  | Example of a discrete SPN . . . . .                                                                                                                                                            | 14 |
| 3.4  | Example of a continuous SPN . . . . .                                                                                                                                                          | 14 |
| 3.5  | Example of a sub-SPN . . . . .                                                                                                                                                                 | 15 |
| 3.6  | Overview of sub-fields of continual learning . . . . .                                                                                                                                         | 19 |
| 4.1  | Basic expansion of a simple PSPN . . . . .                                                                                                                                                     | 28 |
| 4.2  | Modified expansion incorporating the Leaf-Node Transformation of a simple PSPN . . . . .                                                                                                       | 31 |
| 5.1  | Examples of the Gaussian datasets used . . . . .                                                                                                                                               | 37 |
| 5.2  | Examples of the moon datasets used . . . . .                                                                                                                                                   | 38 |
| 5.3  | Structural template used for experiments with Gaussian data . . . . .                                                                                                                          | 39 |
| 5.4  | Structural template used for experiments with moon data . . . . .                                                                                                                              | 39 |
| 5.5  | Dataset used for the case study . . . . .                                                                                                                                                      | 41 |
| 5.6  | Induced SPN of the second column trained on the case study . . . . .                                                                                                                           | 42 |
| 5.7  | Decision boundaries of the case study . . . . .                                                                                                                                                | 43 |
| 5.8  | Induced SPN of the second column trained for longer on the case study . . . . .                                                                                                                | 43 |
| 5.9  | Induced SPN of the modified second column trained on the case study . . . . .                                                                                                                  | 45 |
| 5.10 | Comparison of the median loss and accuracy learning curves in Mean-Shift and Variance-Scaled test cases using moon data, for the baseline SPN, the base PSPN and PSPN with Leaf Copy . . . . . | 47 |
| 5.11 | Comparison of the accuracy learning curves in Independent and Fixed-Cluster test cases using moon data, for the baseline SPN, the base PSPN and PSPN with Leaf Copy . . . . .                  | 48 |
| 5.12 | Examples of the image datasets used . . . . .                                                                                                                                                  | 50 |
| 5.13 | Comparison of the mean loss curves for the baseline SPN and the base PSPN in the non intersecting test case . . . . .                                                                          | 52 |
| 5.14 | Comparison of the mean accuracy curves for the third column of selected search types and the baseline SPN in both test cases . . . . .                                                         | 56 |
| 5.15 | Comparison of the mean accuracy curves for the third column of selected search types applied on individual SPNs and the No Search SPN Baseline in both test cases . . . . .                    | 58 |
| 5.16 | CLEVA-Compass for Progressive SPNs . . . . .                                                                                                                                                   | 61 |

---

## List of Tables

---

|     |                                                                                                                                           |    |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------|----|
| 3.1 | The needed structural constraints for different queries . . . . .                                                                         | 6  |
| 5.1 | Median per-task metrics of PSPN and SPN on moon data . . . . .                                                                            | 40 |
| 5.2 | Mean transfer score of PSPN and SPN on moon data . . . . .                                                                                | 41 |
| 5.3 | Per-task metrics of the modified PSPN and SPN . . . . .                                                                                   | 44 |
| 5.4 | Mean per-task accuracy matrices of the SPN baseline and PSPN on two test cases. . . . .                                                   | 51 |
| 5.5 | Final reached accuracy of all columns for the different search types, in both test cases . . . . .                                        | 54 |
| 5.6 | Most copied column for selected search types . . . . .                                                                                    | 55 |
| 5.7 | Mean transfer score of the third column for select search types in both test cases. . . . .                                               | 55 |
| 5.8 | Mean transfer score of the third column for select search types applied on both PSPNs and SPNs in the non intersecting test case. . . . . | 58 |
| 5.9 | Mean transfer score of the third column for select search types applied on both PSPNs and SPNs in the intersecting test case. . . . .     | 58 |

---

## List of Conventions

---

With the intention of providing consistent notations and enhancing the readability, the following conventions hold throughout this thesis, if not stated otherwise:

|                                          |                                                                      |
|------------------------------------------|----------------------------------------------------------------------|
| $\mathbb{N}$                             | Set of natural numbers                                               |
| $\mathbb{R}$                             | Set of real numbers                                                  |
| $\mathcal{P}(X)$                         | Powerset of $X$                                                      |
| $X, Y$                                   | Single random variables                                              |
| $val(X)$                                 | Set of all possible values of $X$                                    |
| $x, y \in val(X)$                        | Single values of according uppercase random variables                |
| $\mathbf{X}, \mathbf{Y}$                 | Sets of random variables                                             |
| $val(\mathbf{X})$                        | Set of all possible sets of values of $\mathbf{X}$                   |
| $\mathbf{x}, \mathbf{y}$                 | Sets of values of according uppercase set of random variables        |
| $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ | Graph with vertices $\mathbf{V}$ and edges $\mathbf{E}$              |
| $N \in \mathbf{V}$                       | Single node or vertex                                                |
| $ch(N) \subseteq \mathbf{V}$             | Set of all children of node $N$                                      |
| $desc(N) \subseteq \mathbf{V}$           | Set of all descendants of node $N$                                   |
| $pa(N) \subseteq \mathbf{V}$             | Set of all parents of node $N$                                       |
| $\mathbf{S} \subset \mathbf{V}$          | Set of sum nodes                                                     |
| $\mathbf{P} \subset \mathbf{V}$          | Set of product nodes                                                 |
| $\mathbf{L} \subseteq \mathbf{V}$        | Set of leaf nodes                                                    |
| $\mathcal{S} = (\mathcal{G}, \Theta)$    | Sum-Product Network with Graph $\mathcal{G}$ and parameters $\Theta$ |

---

# 1 Introduction

---

By enabling machines to learn, recent research in machine learning (ML) and artificial intelligence (AI) has substantially reformed the way we live. Today applications of AI assist us in many parts of our everyday lives [1]. While some of these applications of AI are non-essential, for example, various recommendation systems for entertainment purposes [1], there are also many safety-critical applications where AI has helped significantly improve our quality of life. In healthcare, AI can improve the accuracy of medical diagnoses, increase the efficiency of treatments or assist with the research and development of new drugs [4]. For example, in late 2020, the artificial intelligence research laboratory DeepMind developed AlphaFold, a protein structure prediction tool facilitating research worldwide [54]. Considering these advancements, it is not far-fetched to assume that further research can develop AI systems that have the potential to tackle some of humanity's biggest challenges, like climate change, where already developed tools are currently being used to optimize renewable energy systems. [6]

However, most currently deployed models only address a limited set of tasks for which they are trained before the system is even deployed [1]. This limits the model to the knowledge gathered during the training, which can quickly become outdated as the environment changes and the data it was trained on becomes outdated. This may degrade the model's performance if it is not continually trained on the new environment [56]. A promising approach to these challenges is enabling systems to continue learning and acquiring new knowledge, adapting to dynamic data changes, and evolving over time [10]. This, furthermore, allows the model to make use of previous knowledge, thus accelerating learning and improving performance [33]. The area of machine learning applying this strategy is known as continual learning, with research that holds promising potential and constitutes another step toward general AI systems. [56]

Although state-of-the-art AI models have fulfilled the promise of revolutionizing many different applications, and becoming an essential part of our lives, these profound changes involve many risks, especially when deploying these technologies in high-stake applications. These powerful technologies often lack the needed explainability to allow for transparent decision-making [4, 53]. For example, in 2018, it was discovered that an AI-powered hiring tool Amazon had been using discriminated against women, because it was trained on predominantly male resumes submitted to the company over the previous ten years[21]. Questions we therefore need to ask ourselves during the development of AI technology include the following:

1. What possible biases do systems hold, and how can we avoid them?
2. Can we retrace the system's decision-making process and understand its conclusions?
3. What are the consequences of a system failure, and who is responsible for failures? [53]

“There has been an increasing trend [...] to leverage machine learning (ML) for high-stakes [...] applications that deeply impact human lives [with] models [...] that do not explain their predictions in a way that humans can understand [lacking] transparency and accountability [...]. If we instead use models that are inherently interpretable, they provide their own explanations, which are faithful to what the model actually computes.” [48]

---

In order to allow for reasonable and effective regulation of AI systems, adequately address the aforementioned questions, and alleviate other similar concerns, a better understanding of the reasoning behind decisions made by AIs is needed. The step towards interpretable AI is therefore crucial for a critical examination of all kinds of ethical problems and, thus, a careful integration of AI into our society. [48]

A type of model that offers promising capabilities for transparent decision-making are probabilistic models. These models have demonstrated wide success across a diverse range of applications and offer several desirable qualities and advantages over traditional models, including:

- **Interpretability:** The use of probabilistic reasoning increases the understanding of the decision-making process. [38]
- **Robustness:** By employing robust statistical methods and robust probability distributions, probabilistic models are less susceptible to noise, improving their reliability. [9]

These abilities provide a way of understanding the decision-making process of probabilistic models, thereby addressing the aforementioned challenges. [20]

All in all, using AI to solve some of humanity’s biggest problems requires the combination of both previously mentioned factors. We need the advantages of continuously adapting models and probabilistic models combined into a single model.

---

## 1.1 Motivation

---

Probabilistic Circuits, a type of probabilistic models introduced in Section 3.1, have emerged as useful models to capture probability distributions by handling the modeling of complex distributions in graphs and enabling the inference of a wide range of probabilistic queries [5]. Specifically, Sum-Product Networks (SPNs), a type of Probabilistic Circuits introduced in 3.1.1, enforce certain constraints among Probabilistic Circuits to ensure tractable inference [45, 5], motivating the use of SPNs as representative models for general Probabilistic Circuits.

However, despite their advantages, one limitation of Probabilistic Circuits and SPNs is their missing support for continual learning applications. They require adaptations to handle incremental updates, which is crucial for real-world applications where data constantly changes and evolves. To provide this support for continual learning, this research was primarily inspired by the work on Progressive Neural Networks (PNNs) by Rusu, Rabinowitz, et al. [49], a continual learning model presented in Section 3.3, which introduced a new concept of networks capable of learning new tasks while retaining the knowledge of previously learned tasks.

We aimed to combine Probabilistic Circuits and PNNs, to develop Progressive Probabilistic Circuits that can efficiently and accurately reuse previously learned complex distributions to be used in continual learning settings, formally introduced in Section 3.2. We hope the progressive modeling approach employed by the PNNs can provide these necessary adaptations to develop practical Progressive Probabilistic Circuits.

---

## 1.2 Objectives

---

This thesis aims to investigate the adaptation of SPNs to continual learning. Our goal is to explore different modifications and extensions of SPNs to handle continual learning scenarios and ultimately research the progressive growing capabilities of Probabilistic Circuits. To do so, we will employ a range of modifications and evaluate their performance on continual learning tasks, beginning with simplistic synthetic data and later advancing to more complex tasks, such as image data. However, while this thesis aims to uncover new possibilities in continual learning methodologies, it is important to note that this is only an exploration and as such requires further research.

To assess to what extent SPNs are capable of progressive growth and whether differences between SPNs and traditional models facilitate or hinder the integration of SPNs within continual learning settings, we aim to answer the primary question: What is the impact of using Progressive Probabilistic Circuits on accuracy, convergence speed, and avoidance of forgetting in continual learning tasks?

Overall, we aim to contribute to the fields of continual learning and Probabilistic Circuits and take the first steps towards combining these fields and laying the groundwork for future research in this area, with the ultimate goal of creating more powerful interpretable machine learning systems.

---

## 1.3 Structure

---

The exploration of different approaches was conducted sequentially such that the results of the previous experiments could be taken into account. The thesis has consequently been structured to present the experiments, their results, and discussions accordingly.

The rest of this thesis is divided into the following five chapters. **Chapter 2** provides an overview of the state of the art by presenting related work in the field. **Chapter 3** presents the required background knowledge in a comprehensive overview of Probabilistic Circuits, more precisely SPNs, the general ideas of continual learning, and the Progressive Neural Networks, our approach is based on. The **Chapter 4** outlines the methods used in the research and experiments for this thesis. To that end, Progressive Sum-Product Networks (PSPNs), the extension of SPNs to our continual learning setting, are defined. Experiments are carried out, analyzed, and discussed sequentially in **Chapter 5**, starting with experiments on synthetic Gaussian data and continuing with more complex data. Furthermore, the main findings and the conclusion of the thesis are summarized in **Chapter 6**. Finally in **Chapter 7**, possible future research endeavors are identified and encouraged. This structure ensures a clear and comprehensive presentation of the research, the findings, and their implications.

---

## 2 Related Work

---

To provide the necessary research context and to justify the focus of this thesis, related literature is presented in the following sections. These related works span specific advancements in probabilistic circuits, continual learning, and their intersection – applying probabilistic models within the realm of continual learning.

Probabilistic circuits, introduced in Choi, Vergari, et al. [5], have sparked an interest due to its advantages of many attractive and efficient inference properties (see Section 1.1). Recently there have thus been substantial advancements that have increased their feasibility as deep models. These range from algorithmic to architectural modifications intended to address different challenges, such as their training, scalability, and expressiveness.

Until recently, algorithms were employed that can combine autonomous structure and parameter learning, such as LearnSPN introduced in 2013 by Gens and Pedro [13]. This fundamental algorithm serves as a blueprint for algorithms and can be adapted and expanded for various applications. The LearnSPN algorithm makes use of the independence assumptions made by product nodes and combines distributions of different clusters using sum nodes, which allows the algorithm to deduce a fitting structure from data samples alone. However, to employ PCs as deep models, the paradigm of closely related structure and parameter learning has been shifting. A recent trend is to generate the structures first and only perform parameter learning afterwards. EiNets [41], RAT-SPNs [43], and XPCs [11] are examples where these models rely on efficient randomly generated structures to increase their expressiveness through deep structures. Despite their random structure, these methods have demonstrated increased performance through their deep learning approach. [11]

For scaling, many different approaches have been proposed. SparsePCs [8], for example, introduce pruning and growing operations to make use of the sparsity in large PC structures. Pruning removes less used sub-networks, reducing the complexity of the model, while growing increases the model capacity by increasing the size of the latent space. This process decreases the unused capacity of PCs. On the other hand, the Latent Variable Distillation (LVD) pipeline [27] ensures the use of the entire capacity right away by providing additional supervision during training over the latent space by using Deep Generative Models, which, although less tractable, are more expressive. Another technique is to employ a weight-sharing technique, which further regularizes PCs. To do so, HyperSPNs [51] use smaller neural networks to generate the mixture weights, hence passing on their generalization and memory properties. All these approaches allow for bigger models, which allow for state-of-the-art performances on several measures. [8, 27, 51]

In terms of their expressiveness, Continuous Mixtures of Tractable Probabilistic Models [7] attempt to combine the expressivity of uncountable mixture models, such as variational autoencoders, and the tractable inference of PCs using discrete mixtures. Although their hybrid approach is analytically intractable, it allows for numerical integration through a finite set of integration points, which can be represented by a standard PC.

While all of these advancements continue to push the limits of what is possible with PCs, there is no research on employing PCs in continual learning, also known as lifelong learning. This significant research area in

---

machine learning aims to create models that can learn and adapt over time. A crucial challenge in this field is to design models that learn new knowledge without forgetting previous knowledge, which is vital for application in real-world environments. While PCs have not been used in this setting, recent advancements have used many different approaches to address these challenges.

Architectural approaches dynamically add capacity, adjusting the model complexity based on its performance. This concept has been used by Dynamically Expandable Networks (DEN) [58], which expand the network when necessary by timestamping, splitting, and duplicating units. Despite its effectiveness, a downside to DENs is the high sensitivity of model performance to hyperparameters required for the dynamic expansion [58, 57]. Building on top of DENs, Reinforced Continual Learning [57] leverages reinforcement learning to search for the most suitable architecture for each incoming task, omitting the need for hyperparameters. This approach prevents catastrophic forgetting while outperforming existing continual learning alternatives.

Another method making up a considerable portion of the continual learning literature is the replay approach. These techniques typically contain a memory buffer to store data instances of previous tasks and attempt to prevent catastrophic forgetting by replaying these instances. However, they tend to require large working memory and may be unfeasible in real-world scenarios where data storage and replay are not viable [47]. Deep Generative Replay [52] follows a similar idea, but instead of storing past data instances, it generates data to mimic past tasks and rehearse previously learned knowledge. In their work, they pair a generator with a solver which replays and learns from generated data. This model can learn new tasks without forgetting previous knowledge and also teach other models by generating input and targets.

In addition to combining traditional deep learning methods with additional approaches, such as regularizations, [19, 26], some recent research focuses on continual learning approaches that leverage probabilistic models because of the possibilities for employing interpretable AI in complex real-world applications, which the potential of combining these two areas opens up.

Variational Continual Learning (VCL) [35] handles complex continual learning settings for both discriminative and generative models by combining online and Monte Carlo variational inference to approximate Bayesian inference. VCLs automatically avoid catastrophic forgetting and yield promising performances. A similar Bayesian perspective is given by Uncertainty-guided Continual Bayesian Neural Networks (UCB) [12]. It takes on a Bayesian perspective on the continual learning challenge, which uses uncertainty to select which knowledge to maintain and thereby again avoid catastrophic forgetting. More specifically, UCB uses uncertainty to select the learning rate, as well as to prune weights.

Yet, when it comes to explicitly applying PCs in the context of continual learning, we find that there is a gap in the literature. To our knowledge, no work has been done to specifically use SPNs in continual learning settings, which presents an opportunity for us to explore the potential capabilities of SPNs in this important area of machine learning. This thesis aims to address this gap by exploring methods of adapting SPNs to continual learning.

---

## 3 Background

---

This chapter serves as an introduction for the reader to key topics relevant to exploring modifications to SPNs for continual learning scenarios. It aims to build a foundation for relevant concepts, theories, and ideas that will be needed in the later sections of this thesis.

**Figure 3.1** provides a broad overview of the background topics discussed in this chapter, indicating their relations and relevance to the general setting and the development of PSPNs. We begin with a detailed explanation of Probabilistic Circuits, specifically SPNs, by introducing the needed definitions, constraints, and training methods. We follow with a definition of continual learning and its challenges, as well as common modeling approaches in continual learning needed to establish a setting for the exploration formally. Finally, we introduce the progressive modeling ideas used in Progressive Neural Networks and the basic definitions needed to develop PSPNs.

---

### 3.1 Probabilistic Circuits and Sum-Product Networks

---

Probabilistic Circuits are probabilistic models first introduced in Choi, Vergari, et al. [5] as a generalized framework for tractable probabilistic modeling. Abstractly, they capture probability distributions in the form of computational graphs, which depending on specific structural constraints, ensure the tractability of different probabilistic queries.

Many models developed before the introduction of PCs were designed to be tractable by satisfying different subsets of these constraints, enabling the according subset of these queries. PC, therefore, is a generic term that encompasses all these models under one framework [5]. The most prominent PCs are SPNs that satisfy

| Query Class                   | Com. | Con. | Dec. | Str.-Dec. | Det. | Mar.Det. |
|-------------------------------|------|------|------|-----------|------|----------|
| Marginal                      | ×    |      | ×    |           |      |          |
| Conditional                   | ×    |      | ×    |           |      |          |
| Moments of Distribution       | ×    |      | ×    |           |      |          |
| Maximum A-Posteriori          |      | ×    |      |           | ×    |          |
| Marginal Maximum A-Posteriori | ×    |      | ×    |           |      | ×        |
| Expectation                   | ×    |      | ×    | ×         |      |          |
| Kullback-Leibler Divergence   | ×    |      | ×    | ×         | ×    |          |

Table 3.1: This table is taken from Choi, Vergari, et al. [5] and shows the structural constraints needed for different types of tractable queries. The rows show the types of queries that can be performed, while the columns denote the following constraints: Completeness, Consistency, Decomposability, Structured-Decomposability, Determinism and Marginal Determinism.

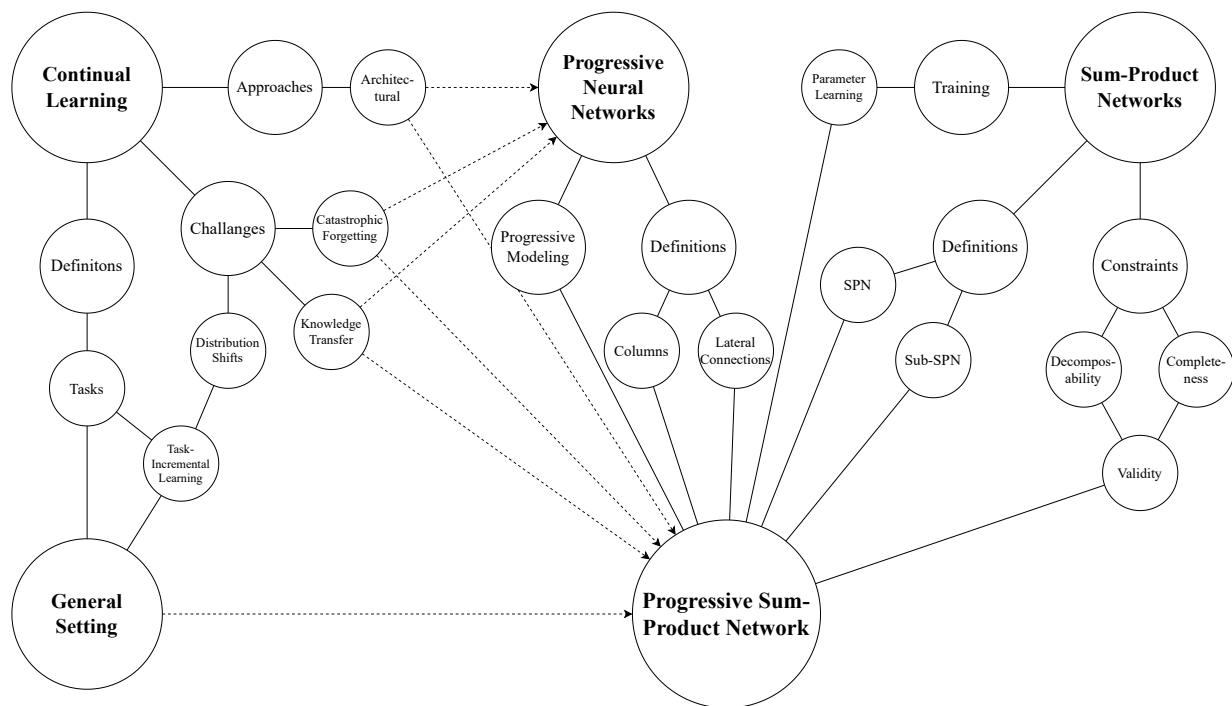


Figure 3.1: This figure shows an overview of the background topics and their relations to the setting of this thesis and the development of PSPNs. The core topics are represented in larger nodes, with their corresponding sub-sections depicted in connected smaller nodes and the individual concepts shown in the smallest nodes. Dashed arrows indicate the influence of one topic on another without strictly needing it for its definition. Note that the arrangement of nodes does not correspond to the sequence of the sections.

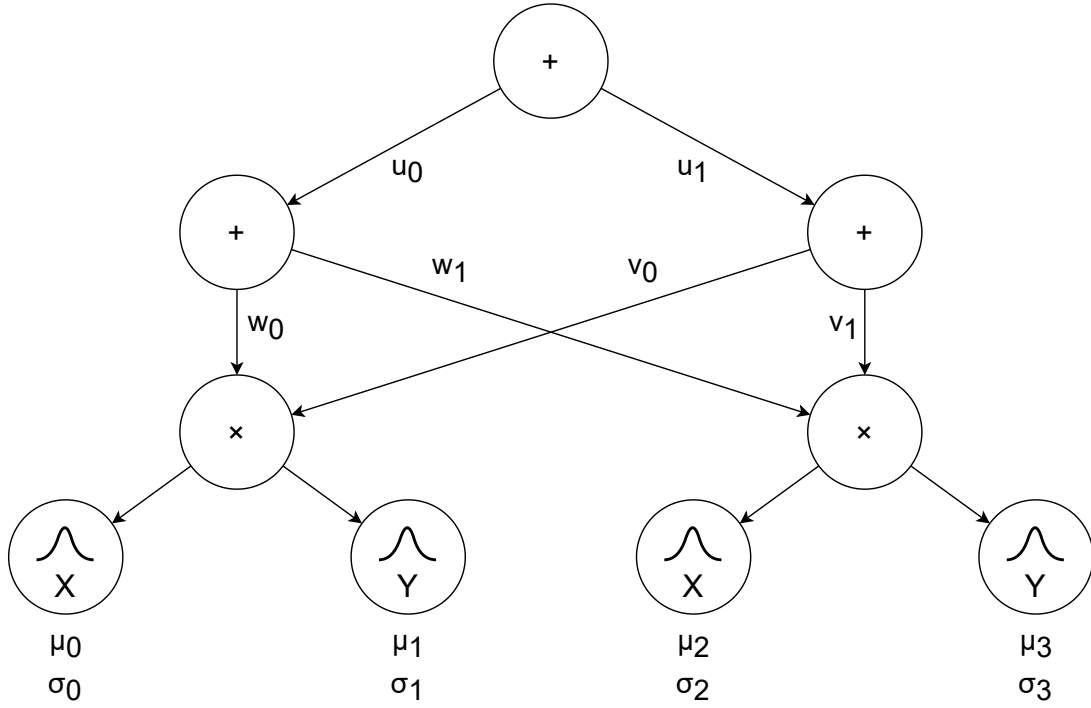


Figure 3.2: This figure shows an example of an SPN over two random variables  $X$  and  $Y$  with weights  $u_i, w_i$  and  $v_i$  and distribution parameters  $\mu_i$  and  $\sigma_i$  for  $i \in \{0, 1\}$ . Note that this SPN could be simplified further by combining the sum nodes into one.

the completeness (also known as smoothness) and decomposability constraints to allow for tractable inference of commonly used queries [50]. They were first introduced in 2011 by Poon and Domingos [45] as a deep architecture in the form of directed acyclic graphs with computational sum and product nodes and simple probability distributions as leaves:

- **Sum-Nodes** perform a normalized weighted sum across their child nodes, creating a mixture of the probability distribution represented by the child nodes.
- **Product-Nodes** calculate the product of all their child nodes, representing a factorization of the distribution into the distributions given by the child nodes.
- **Leaf-Nodes** represent simple parameterized probability distributions over specific sets of random variables. Common examples of such distributions are univariate or multivariate Gaussian distributions. [50]

These nodes are arranged in a graph structure that adheres to the abovementioned constraints [5]. Through this structure, SPNs encode and hierarchically build the desired probability distributions.

Figure 3.2 shows a simple SPN, which we will come back to throughout the thesis and is therefore presented as the main example for SPNs. The root node at the top represents the entirety of the probability distribution and is a sum node itself representing a mixture of the other two sum nodes, which themselves are mixtures of the underlying distributions of the two product nodes. Here it should be noted that, while such a structure can be valid, due to the linearity of weighted sums, the three sum nodes can be combined into a single one with weights  $z_0 = u_0 \cdot w_0 + u_1 \cdot v_0$  and  $z_1 = u_0 \cdot w_1 + u_1 \cdot v_0$ . The product node distributions are finally factorized

into two leaf distributions each, forming the four leaf nodes parameterized by two parameters each. The SPN thus has 10 free parameters in total, namely the distribution parameters mean  $\mu_i$  and standard deviation  $\sigma_i$  and the normalized sum weights  $z_i$  for  $i \in \{0, 1\}$ . The normalization refers to the weights of each sum node summing up to one  $z_0 + z_1 = u_0 + u_1 = w_0 + w_1 = v_0 + v_1 = 1$ , such that the weighted sum itself is a valid distribution mixture.

If samples from a given distribution are provided, various learning algorithms can be used to infer the structure and learn the parameters of the leaf nodes and the weights of the sum nodes [13]. While these learning algorithms include commonly used algorithms like gradient descent, SPNs can also be trained with several probabilistic algorithms like Expectation Maximization and Bayesian methods due to their probabilistic nature [50].

### 3.1.1 Formal Definitions

In this section, we will formally introduce SPNs and the accompanying terminology. For further references, see the work, which the information is mainly based upon Sanchez-Cauce, Paris, et al. [50], Poon and Domingos [45], and Peharz [40], as well as Gens and Pedro [13] and Peharz, Tschitschek, et al. [42], from which additional insights have been drawn from. Note that this thesis only focuses on an exploration of the basic capabilities of SPNs in the continual learning setting. Some concepts typically introduced in a more comprehensive discussion of SPNs are not relevant to this thesis and are thus omitted, e.g., concepts needed for marginal inferences, such as partial evidence and probability functions in general.

#### 3.1.1.1 Definitions: Values of Random Variables

Let  $\mathbf{X} = \{X_1, \dots, X_n\}$  be a set of random variables  $X_i$  for  $i \in \{1, \dots, n\}$  defined on an arbitrary (continuous or discrete) probability space  $\Omega$ .

1. We denote the set of all possible values  $x_i$  a random variable  $X_i$  assumes with  $val(X_i)$ .
2. We define the extension of  $val$  to all possible values  $\mathbf{x} = \{x_1, \dots, x_n\}$  of a set of random variables  $\mathbf{X}$  as:

$$val(\mathbf{X}) = \{\{x_1, \dots, x_n\} \mid x_i \in val(X_i) \ \forall i \in \{1, \dots, n\}\}$$

We refer to the elements  $\mathbf{x} \in val(\mathbf{X})$  as **(complete) evidence**.

3. For any given evidence  $\mathbf{x}$  the value assumed by  $X_i$  is denoted as  $\mathbf{x}[X_i] = x_i$

[50, 40]

Values of random variables refer to possible outcomes of a random variable and form the base set on which SPNs operate, and thus serve as the starting point of the upcoming definitions. Some known sets of values are:

1. Possible outcomes of a dice roll  $X_D$ :  $val(X_D) = \{1, 2, 3, 4, 5, 6\}$
2. Possible outcomes of a coin toss  $X_C$ :  $val(X_C) = \{H, T\}$

In many cases, single random variables are insufficient, and we need to consider multiple random variables to create a practical stochastic model. If, for example, we wanted to model a simultaneous roll of a dice and a coin toss, an outcome for the set of random variables  $\mathbf{X} = \{X_D, X_C\}$  could be expressed as  $\mathbf{x} = \{2, H\} \in \text{val}(\mathbf{X})$ . This outcome is called evidence, as it provides information about the underlying process. We use the term complete evidence to specify when no information is missing, which could be the case if, for example, one random variable has not assumed any value yet. The third definition offers a formal way to index the outcomes and access specific values, such as  $\mathbf{x}[X_D] = 2$  and  $\mathbf{x}[X_C] = H$ . To now complete the stochastic model, we need to assign probabilities to all possible outcomes, which is achieved using a probability distribution over the random variables.

### 3.1.1.2 Definitions: Probability Distributions

Let  $\mathbf{X}$  be a set of random variables defined on a discrete probability space  $\Omega$ .

A **probability mass function (PMF)** on  $\mathbf{X}$  is defined as a function,  $p : \text{val}(\mathbf{X}) \rightarrow \mathbb{R}$  such that:

1.  $\forall \mathbf{x} \in \text{val}(\mathbf{X}) : p(\mathbf{x}) \geq 0$
2.  $\sum_{\mathbf{x} \in \text{val}(\mathbf{X})} p(\mathbf{x}) = 1$

Now, let  $\mathbf{X}$  be a set of random variables defined on a continuous probability space  $\Omega$ .

A **probability density function (PDF)** on  $\mathbf{X}$  is defined analogously i.e.  $p : \text{val}(\mathbf{X}) \rightarrow \mathbb{R}$ , such that:

1.  $\forall \mathbf{x} \in \text{val}(\mathbf{X}) : p(\mathbf{x}) \geq 0$
2.  $\int_{\text{val}(\mathbf{X})} p(\mathbf{x}) d\mathbf{x}$

[50]

For general probability distributions, it should be noted that the term is used independent of the specific underlying probability space. Nonetheless, the definitions must consider the nature of the probability space. This is why in the context of a *discrete* probability space, a PMF assigns a probability to each possible value of  $\mathbf{X}$  and the distribution is defined as a non-negative function whose sum over all potential values normalizes to 1. For a *continuous* probability space, it is not possible to assign reasonable probabilities to individual values because there are infinite possibilities. Instead, a PDF describes a normalized density assigning probabilities to ranges of values and is, therefore, not defined by a sum over all values but by an integral over the input space, which is equal to one.

Revisiting the example of possible outcomes from a coin toss  $X_C$ , we describe the probability distribution using a PMF defined as  $p(X_C = H) = 0.6$  and  $p(X_C = T) = 0.4$ , assuming an unfair coin. More generally, examples of probability distributions include:

- Bernoulli distributions as PMFs:  $f(x) = px + (1 - p)(1 - x)$
- Normal/Gaussian distributions as PDFs:  $f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$

In this thesis, for the sake of simplicity, we will focus on univariate distributions, mainly using Gaussian distributions.

### 3.1.1.3 Remark: Combining Probability Distributions

Probability distributions can be combined in specific ways to construct new, more complex distributions. Here we introduce two essential methods:

1. **Mixture of Probability Distributions:** Given probability distributions  $\mathbf{p} = p_1, \dots, p_n$  over the same set of random variables  $\mathbf{X}$  and non-negative weights  $w_1, \dots, w_n$ , such that  $\sum_{i=1}^n w_i = 1$ , the function:

$$p_{\Sigma}(\mathbf{x}) = \sum_{i=1}^n w_i \cdot p_i(\mathbf{x})$$

is also a valid probability function. This operation, which creates a weighted sum of probability distributions, is called a **mixture** of probability distributions.

2. **Factorization of Probability Distributions:** Given probability distributions  $\mathbf{p} = \{p_1, \dots, p_n\}$  over disjoint sets of random variables  $\mathbf{X}_1, \dots, \mathbf{X}_n$ , the function:

$$p_{\Pi}(\mathbf{x}) = \prod_{i=1}^n p_i(\mathbf{x})$$

is also a valid probability function over the union  $\mathbf{X} = \bigcup_{i=1}^n \mathbf{X}_i$ . This operation, which multiplies different probability distributions, is referred to as a **factorization** of probability distributions.

[50]

These properties show the composability of probability distributions, provided they are defined over appropriate sets of random variables. This trivially allows for repeated alternation between mixtures and factorizations to construct more complex distributions. While it is not strictly necessary to alternate between mixtures and factorizations, repeated consecutive uses of either operation can be simplified into a single mixture or factorization due to the linearity of these operations as seen in **Figure 3.2**. Therefore, we can restrict the combinations to alternating uses of sums and products.

Crucially, both mixtures and factorizations maintain tractability, ensuring the tractability of the constructed distribution if the base probability distributions are tractable, which is a desirable property for computational efficiency. Moreover, these methods enable a more flexible and versatile way to build PMFs by combining so-called indicator functions rather than using pre-defined PMFs.

### 3.1.1.4 Definition: Indicator Function

Let  $\mathbf{X} = \{X_1, \dots, X_n\}$  be a set of random variables over a discrete probability space and  $X \in \mathbf{X}$ . Furthermore, let  $x \in \text{val}(X)$  be a value  $X$  can assume.

We define the indicator function  $\mathbb{1}_{X=x}$  as follows:

$$\mathbb{1}_{X=x} : \text{val}(\mathbf{X}) \longrightarrow \{0, 1\}, \quad \mathbb{1}_{X=x}(\mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{x}[X] = x \\ 0, & \text{else} \end{cases}$$

[40]

The advantage of indicator functions in the context of SPNs lies in their computational efficiency, as an indicator function is a simple binary function that outputs a binary value indicating whether the random variable  $X$  takes on the specified value  $x$ . Since indicator functions satisfy the definitions of PMFs, they can be used to construct arbitrary PMFs via the mixture and factorization operations discussed earlier in Remark 3.1.1.3.

This offers much flexibility, as it does not require the assumption of any specific family of distributions. For instance, the Bernoulli distribution for the coin toss example  $X_C$  can be constructed using indicator functions as follows:  $p(x) = p \cdot \mathbb{1}_{X_C=H} + (1-p) \cdot \mathbb{1}_{X_C=T}$ . For  $p = 0.6$ , this equation describes the probability distribution of the unfair coin toss we previously discussed. With this theoretical background on distributions, we are now prepared to continue with the formal definition and structure of SPNs.

### 3.1.1.5 Definition: Sum-Product Networks over Discrete Probability Spaces

When describing the structure of SPNs, it is important to establish conventions for naming various graph components, to provide a clear understanding of the nodes and their corresponding roles. By convention, a graph with vertices given as  $\mathbf{V} = \mathbf{S} \cup \mathbf{P} \cup \mathbf{L}$  can be partitioned into sum nodes  $\mathbf{S}$ , product nodes  $\mathbf{P}$ , and leaf nodes  $\mathbf{L}$ . While these terms have yet to be formally defined, these conventions allow us to distinguish between them to describe their functionality. We use  $N \in \mathbf{V}$  to refer to general nodes in the graph, allowing us to make structural arguments without specifying what type of node. The structural notation includes terms such as  $ch(N)$ ,  $desc(N)$ , and  $pa(N)$  that describe the child nodes, all descendants, and the parent nodes, respectively.

Let  $\mathbf{X}$  be a set of random variables over a discrete probability space and  $\mathbf{w}$  be a set of normalized parameters, i.e.,  $\sum_{w \in \mathbf{w}} w = 1$ .

Additionally, let  $\mathcal{G} = (\mathbf{V}, \mathbf{E})$  be a rooted acyclic directed graph with nodes  $N \in \mathbf{V} = \mathbf{S} \cup \mathbf{P} \cup \mathbf{L}$ , which can be categorized into three types of functions:

1.  $N = S: val(\mathbf{X}) \rightarrow [0, 1] \in \mathbf{S}$ :

$$S(\mathbf{x}) = \sum_{C \in ch(S)} w_{S,C} \cdot C(\mathbf{x})$$

Where  $w_{S,C} \in \mathbf{w}$  denotes the weight of the edge from node  $S$  to  $C$ .

2.  $N = P: val(\mathbf{X}) \rightarrow [0, 1] \in \mathbf{P}$ :

$$P(\mathbf{x}) = \prod_{C \in ch(P)} C(\mathbf{x})$$

3.  $N = L: val(\mathbf{X}) \rightarrow \{0, 1\} \in \mathbf{L}$ :

$$L(\mathbf{x}) = \mathbb{1}_{X=x}(\mathbf{x})$$

We define the **Sum-Product Network** as the tuple  $\mathcal{S} = (\mathcal{G}, \mathbf{w})$

[40, 45]

The definition differentiates between the three types of nodes: sum nodes, product nodes, and leaf nodes. While leaf nodes do not have children, as the name suggests, sum and product nodes are inner nodes that require their child nodes for their definition. As such the nodes are defined recursively as functions using,

their child nodes. This construction is well-defined because the definition of each node only uses its own children. In a finite ( $|V| < \infty$ ) directed acyclic graph, there are always leaf nodes, defined independently of other nodes, forming the base case for the recursive definition. Because of this structure, we can take any node  $N \in \mathbf{V}$  in the graph and get a valid probability distribution from its function, denoted  $p_N$ . The root node  $R$  has no parent and initiates the computation of the entire SPN. Hence we denote  $p_R$  as  $\mathcal{S}(\mathbf{x})$ . Following the previously mentioned remark, sum nodes can be interpreted as mixtures of distributions, product nodes as factorizations over independent distributions, and leaf nodes as base distributions. While discrete SPNs can use any PMFs as leaf nodes, we use indicator functions, since these can be combined to simulate other PMFs. Through these combinations, the SPN, in this case, constructs a discrete distribution. Therefore, such an SPN is more specifically named an SPN over a discrete probability space.

### 3.1.1.6 Remark: Discrete and Continuous SPNs

The construction of SPNs can naturally be extended to continuous probability spaces. Since the combination of PMFs always yields a PMF itself, and the combination of PDFs always results in a PDF, we need to adjust the base distributions. Instead of using indicator functions, we must adapt leaf nodes to use PDFs.

### 3.1.1.7 Definition: Sum-Product Networks

Let  $\mathbf{X}$  now be a set of random variables over a continuous probability space and  $\Theta = \mathbf{w} \cup \theta$  be a new set of parameters.

We define a **Sum-Product Network over a continuous probability space**  $\mathcal{S} = (\mathcal{G}, \Theta)$ , analogous to Definition 3.1.1.5, using the new set of random variables  $\mathbf{X}$  and changing the leaves to:

3. For  $N = L: \text{val}(\mathbf{X}) \rightarrow [0, 1] \in \mathbf{L}$ :

$$L(\mathbf{x}) = p_{\theta}(\mathbf{x})$$

Where  $\theta$  denotes the parameters of the distribution.

[40]

Building upon the definition of an SPN over a discrete probability space, as defined in Definition 3.1.1.5, we define an SPN over a continuous probability space by using probability density functions  $p_{\theta}$ . In this case, the set of parameters  $\Theta$  includes both the normalized weights of the edges  $\mathbf{w}$  and the parameters  $\theta$  of the probability distribution, if it is a parameterized distribution. From now on, we will not differentiate between the discrete and continuous cases and only refer to SPNs in general, it will be clear from the context which case we are dealing with.

To better illustrate the concepts explained so far, we consider a few examples, such as the previous example of the binomial distribution. We can construct a corresponding representation with the SPN shown in Figure 3.3. In this SPN, a mixture of indicator functions, with weights corresponding to the probabilities of the outcomes. Another example we already considered was Figure 3.2, an SPN with probability distributions as leaf nodes. Recalling that consecutive applications of the same operation can be simplified, the Remark 3.1.1.3 allows us to simplify the SPN structure without changing the distribution: Once again,  $x_0$  and  $x_1$  represent combined weights or parameters from the more complex SPN. This simplification can compact the SPN representation and thus help analyze the structure and understand the underlying distribution better.

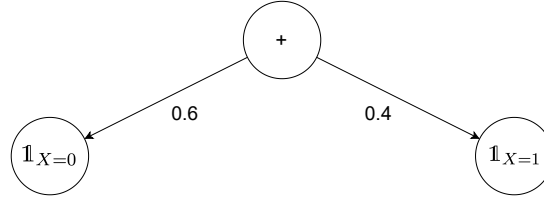


Figure 3.3: This figure shows an example of a discrete SPN over the random variable  $X$  with indicator functions as leaves and weights 0.6 and 0.4.

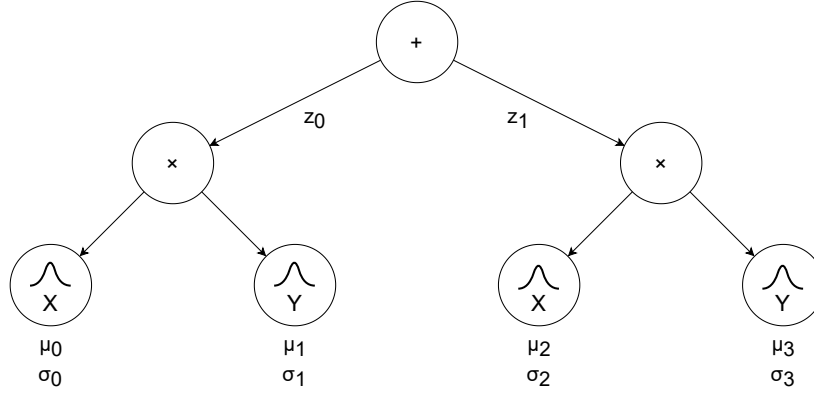


Figure 3.4: This figure shows an example of a continuous SPN over the random variables  $X$  and  $Y$  with parameterized PDFs as leaves with parameters  $\mu_i$  and  $\sigma_i$  as well as sum weights  $z_i$  for  $i \in \{0, 1\}$ . Note that this is the simplified SPNs of [Figure 3.2](#)

While SPNs primarily serve for data density estimation, they can also be used for classifications. This is done by implicitly representing the class variables through multiple root nodes, each corresponding to a specific class conditional likelihood. These can then be used for classification by calculating the posterior probabilities through Bayesian inference. Additionally, if we want to model the entire data distributions and transform the SPN back into a rooted directed acyclic graph again, we can use a sum node to marginalize the class variables by using their priors as weights. We refer to the class conditional likelihood nodes as a **classification head** and implicitly add a root node if necessary for any formal construction. If instead we want to only model one class, we can use the so called Sub-SPN of the given class root node. To do so, it can be useful to define these substructures formally, which we also need for the definition of the constraints.

### 3.1.1.8 Definition: Sub-Sum-Product Networks

Let  $\mathcal{S} = (\mathcal{G}, \Theta)$  with  $\mathcal{G} = (\mathbf{V}, \mathbf{E})$  be an SPN.

For any node  $N \in \mathbf{V}$  we define the **sub-SPN rooted at  $N$**  as an SPN  $\mathcal{S}_N = (\mathcal{G}_N, \Theta_N)$  with the rooted sub-graph  $\mathcal{G}_N$  induced by  $\text{desc}(N)$  and the trivial restriction of the parameters  $\Theta_N$  on the induced sub-graph  $\mathcal{G}_N$ .

[40]

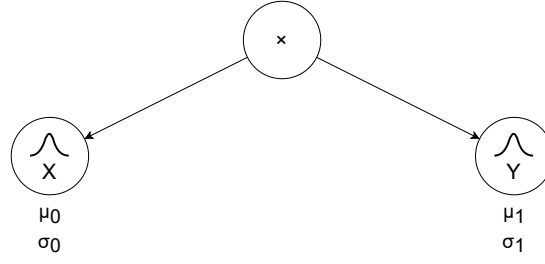


Figure 3.5: This figure shows an example of a sub-SPN of the left product node in the SPN in Figure 3.2

Since an SPN is a directed acyclic graph, any node in the graph and its descendants also form a rooted directed acyclic graph. Therefore, every defined sub-SPN is also an SPN, which represents a probability distribution. Similar to general SPNs, we denote the probability distribution of a sub-SPN rooted at node  $N$  as  $\mathcal{S}_N$ .

Working with sub-SPNs provides a way to analyze and understand the overall structure of an SPN. It can improve the interpretability of the SPN by breaking down the network into smaller components. For example, the sub-SPN in Figure 3.5 is a part of the SPN shown in Figure 3.2. By analyzing that this sub-SPN represents a multivariate Gaussian distribution with no covariance, we can infer that the parent SPN must be a mixture of two multivariate Gaussian distributions. To define the formal constraints of SPNs, we still need the following definition of scopes.

### 3.1.1.9 Definition: Scope

Let  $\mathcal{S} = (\mathcal{G}, \Theta)$  with  $\mathcal{G} = (\mathbf{V}, \mathbf{E})$  be an SPN over random variables  $\mathbf{X}$  with nodes  $\mathbf{V} = \mathbf{S} \cup \mathbf{P} \cup \mathbf{L}$ .

For any node  $N \in \mathbf{V}$  we recursively define the **scope** as:

$$sc : \mathbf{V} \longrightarrow \mathcal{P}(\mathbf{X})^1, \quad sc(N) = \begin{cases} \{X\}, & \text{if } N \in \mathbf{L} \text{ and } N \text{ is defined over } X \\ \bigcup_{C \in ch(N)} sc(C), & \text{else} \end{cases}$$

[40]

This recursive definition describes the scope of a given node as the set of random variables over which the node operates by joining the scopes of all child nodes. With analogous arguments as for the recursive Definition 3.1.1.5 of SPNs, we argue that the scope is also well-defined. By identifying the SPN with its root node, this concept of the scope can be extended to SPNs. We represent this as  $sc(\mathcal{S}) := sc(R)$ . We can further extend this concept to probability distributions since each node in an SPN also represents a probability distribution.

In the case of the discrete example SPN (see Figure 3.3), the scope is  $sc(\mathcal{S}) = \mathbf{X}$ , since all child nodes of the sum root node have the same scope,  $\mathbf{X}$ . In the second general example (see Figure 3.4), where the PDFs for the leaf nodes, are defined over the random variables  $X$  and  $Y$ , we can deduce the scope of the SPN to be  $sc(\mathcal{S}) = \{X, Y\}$ . With an understanding of the concept of scopes, we can now define the constraints needed.

<sup>1</sup> $\mathcal{P}(\mathbf{X})$  denotes the power set of  $\mathbf{X}$

### 3.1.2 Constraints

Until now, we have considered arbitrary directed acyclic graphs and defined SPNs on them. However, it is crucial to note that not all structures yield valid probability distributions due to requirements set by Remark 3.1.1.3 for the random variables over which the distributions are defined. For instance, a mixture must be constructed over the same set of random variables.

#### 3.1.2.1 Definition: Completeness

Let  $\mathcal{S} = (\mathcal{G}, \Theta)$  with  $\mathcal{G} = (\mathbf{V}, \mathbf{E})$  be an SPN with nodes  $\mathbf{V} = \mathbf{S} \cup \mathbf{P} \cup \mathbf{L}$

We call sum nodes  $S \in \mathbf{S}$  **complete** if:

$$\forall C, C' \in ch(S): sc(C) = sc(C')$$

We call the SPN  $\mathcal{S}$  **complete** if:

$$\forall S \in \mathbf{S}: S \text{ is complete}$$

[40]

This constraint, also referred to as smoothness in other Probabilistic Circuits (see Section 3.1), enforces the requirement for mixtures for single sum nodes. Since mixtures are defined over the same random variables, for a sum node to represent a mixture, its children nodes must define their distributions over the same scope. Assuming the children nodes represent valid probability distributions, this ensures that the parent sum node also represents a valid probability distribution. Consequently, for an entire SPN to compute a probability distribution, all sum nodes must fulfill this completeness constraint, which motivates the definition of completeness for entire SPNs.

While the completeness constraint assures that distributions under sum nodes are defined over the same set of random variables, it does not guarantee that factorizations are defined over distributions with disjoint sets of random variables, as described in Remark 3.1.1.3. To ensure this, we introduce another constraint, which enforces that requirement.

#### 3.1.2.2 Definition: Decomposability

Let  $\mathcal{S} = (\mathcal{G}, \Theta)$  with  $\mathcal{G} = (\mathbf{V}, \mathbf{E})$  be an SPN with nodes  $\mathbf{V} = \mathbf{S} \cup \mathbf{P} \cup \mathbf{L}$

We call product nodes  $P \in \mathbf{P}$  **decomposable** if:

$$\forall C, C' \in ch(P), \text{ such that } C \neq C': sc(C) \cup sc(C') = \emptyset$$

We call the SPN  $\mathcal{S}$  **decomposable** if all its product nodes are decomposable

[40]

Analogously to the completeness constraint, the decomposability constraint guarantees the requirement of disjoint sets of random variables since each child node under a product node must have a unique scope. This translates to each child having to represent a distribution defined over a set of random variables distinct from those of its siblings. Therefore, for an SPN to represent a valid probability distribution, all product nodes must fulfill the decomposability constraint, again justifying the definition of a decomposable SPN. However, it is worth noting that the decomposability constraint is stricter than necessary for an SPN to represent a valid probability distribution.

### 3.1.2.3 Remark: Consistency

Let  $\mathcal{S} = (\mathcal{G}, \Theta)$  with  $\mathcal{G} = (\mathbf{V}, \mathbf{E})$  be an SPN with nodes  $\mathbf{V} = \mathbf{S} \cup \mathbf{P} \cup \mathbf{L}$

We call product nodes  $P \in \mathbf{P}$  **consistent** if:

$$\forall C, C' \in \text{ch}(P), \text{ such that } C \neq C' \text{ it holds that: } \mathbb{1}_{X=x} \in \text{desc}(C) \implies \mathbb{1}_{X=x'} \notin \text{desc}(C'), \text{ if } x \neq x'$$

[40]

This definition implies that no variable can appear with one realization in one child and a different realization in another child. It is important to note that this definition would need to be adjusted for continuous SPNs, but that is outside the scope of this discussion. It is apparent that for child nodes with disjoint scopes, no two random variables can have conflicting realizations. Therefore, decomposability, by definition, implies consistency:

$$\forall P \in \mathbf{P}: P \text{ is decomposable} \implies P \text{ is consistent}$$

Given that the consistency condition is less strict than decomposability, SPNs fulfilling consistency instead of decomposability could potentially allow for the representation of more distributions or more efficient representations of the same distributions, providing advantages over decomposable SPNs. However, as shown in a theorem presented in Peharz, Tschitschek, et al. [42], the benefits of this increased flexibility are not significant (e.g., exponentially more compact representations). Therefore, despite the consistency constraint, as originally introduced in the original work by Poon and Domingos [45], being strictly weaker than the decomposability constraint, it offers no significant practical advantage. In fact, it further complicates both the definition and the implementation, which is why in practice, the decomposability constraint is favored over the consistency constraint due to its simplicity [50].

Having established the definitions of completeness (or smoothness) and decomposability, we can formally complete the general SPNs considered so far to those that compute probability distributions.

### 3.1.2.4 Definition: Validity

Let  $\mathcal{S} = (\mathcal{G}, \Theta)$

We call  $\mathcal{S}$  **valid** if:

$$\exists \text{ probability distribution } p : \text{val}(\mathcal{S}) \longrightarrow \mathbb{R}, \text{ such that: } \forall x \in \text{val}(\mathcal{S}): \mathcal{S}(x) = p(x)$$

[40]

---

This defines the desired SPNs we aim to explore, namely valid SPNs. It is worth noting that this validity does not impose any structural assumptions on the SPN. We can replace any node in the SPN with a distribution over the same scope without affecting its validity. An example of a valid SPN is the binomial SPN, as there exists a distribution it can represent, specifically, the according binomial distribution. However, while valid SPNs are the ones we are interested in, the definition of validity alone does not provide a practical way to construct such SPNs or verify the validity of existing SPNs, which is where the previously defined constraints of completeness and decomposability are used.

**Theorem** An SPN is valid if it is complete and consistent. (For the proof we refer to Peharz [40]).

Since decomposability implies consistency, our defined structural constraints imply validity, thereby providing a way to verify if an SPN is valid. Continuing with the example, the binomial SPN is valid not only because it represents a known distribution but also because the sum node only has children with the same scope, fulfilling the completeness constraint. In addition to providing a trivial validity verification, we can also use sub-SPNs to provide a simple way of constructing SPNs.

**Theorem** An SPN is complete and consistent if and only if all sub-SPNs are valid. (For the proof we refer to Peharz [40]).

When combined with the previous theorem, we see that an SPN is valid if all its sub-SPNs are valid. This offers a practical method for constructing larger SPNs. By iteratively introducing new sum or product nodes as parents of the root nodes and another SPN, we can construct new SPNs, as long as these new nodes and the new sub-SPN fulfill the constraints of completeness and decomposability. In fact, we can even dissect existing SPNs into their sub-SPNs and recombine these sub-SPNs as needed, thereby creating a wide array of valid SPNs. To illustrate this process of constructing valid SPNs using sub-SPNs, consider the sub-SPN example in Figure 3.5. Assuming that the sub-SPN is valid by having leaves with disjoint scopes, we could construct the entire SPN by reusing the same sub-SPN twice and inserting a sum node above this sub-SPN to ensure that the children of the sum node have the same scope. We can therefore deduce that the SPN in Figure 3.2 can be valid if the leaves 0 and 1 have different scopes, but leaves 0, 1, and 2, 3 have the same scope.

In summary, we have defined the necessary terminology and established what a valid SPN over arbitrary (discrete and continuous) probability spaces is. We have also found ways to construct SPNs with the concept of substructures, recombine them to form new ones, and check the validity of general SPNs using the constraints of completeness and decomposability.

Combining the presented knowledge, we can recursively define valid SPNs using the following three rules:

1. A tractable probability distribution is an SPN.
2. A product of SPNs over disjoint scopes is an SPN.
3. A normalized weighted sum of SPNs over the same scope is an SPN.

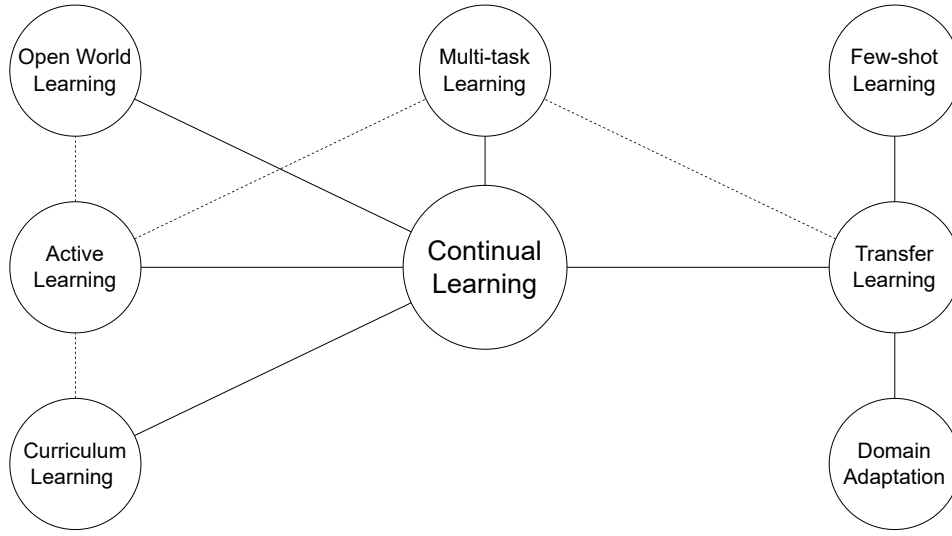


Figure 3.6: This figure is adopted from Mundt, Braun, et al. [32] and shows a reduced form of the relationships between machine learning settings in continual learning. It provides an overview of the different sub-fields and their relationships within the context of continual learning. The dotted lines indicate the influence between these sub-fields.

## 3.2 Continual Learning

In many traditional machine learning applications, a common assumption is that the underlying data distribution remains static after the initial training phase. This assumption implies that the trained model will be applied to the same distribution it was trained on. However, this assumption does not hold in many real-world applications. Environments are dynamic and change over time, leading to shifts in the underlying data distribution [46]. Continual learning (often also referred to as lifelong learning) is a machine learning field that addresses the challenges of training models on data streams, where the underlying data distribution may shift over time [10].

It is a broad field with many sub-fields, seen in Figure 3.6, focusing on different aspects and goals of training models in dynamic environments [32]. Transfer learning is one of these fields, where we reuse a model developed for one task over one domain to train a model on a second task over another domain [36]. By interpreting the dynamic changes of the dataset as a new data distribution over the same feature space, we obtain a new domain on which the model stands to be trained. Using the existing model to improve performance when learning the second task then describes goal of transfer learning. For further specifics on the relations and definitions of the sub-fields, we refer readers to Mundt, Braun, et al. [32].

In the context of our study, we are interested in how SPNs can be adapted to leverage the benefits of transfer learning in continual learning scenarios. This involves exploring how density estimation gained from previous tasks can be effectively reused for future tasks. However, we simultaneously want to avoid forgetting the previous tasks, another major challenge of continual learning models. These two goals of continual learning and transfer learning, in particular, set the two main goals of our modifications.

To later formalize our setting for the exploration, the core concepts and ideas in continual learning are introduced and defined in the following. These subsequent definitions are primarily based on the literature Lesort, Lomonaco, et al. [25], De Lange, Aljundi, et al. [10], and Mundt, Hong, et al. [33], but are also

inspired by the work of Mundt, Braun, et al. [32] and make use of Ven, Tuytelaars, et al. [55]. We encourage the reader to refer to these references for a more in-depth explanation of the concepts presented.

As described in the introduction, addressing the challenges posed by shifts in data distributions is the main problem of continual learning.

### 3.2.0.1 Definition: Distribution Shift

For a given set of random variables  $\mathbf{X}$  as data samples and a random variable  $Y$  as the label, let  $p_{pre}$  be the probability distribution during training, and  $p_{post}$  be the probability distribution after training.

We say a **distribution shift** occurred when:

$$p_{pre}(\mathbf{x}, y) \neq p_{post}(\mathbf{x}, y)$$

These distribution shifts can further be categorized into the following types:

1. **Covariate shift:**  $p_{pre}(y|\mathbf{x}) = p_{post}(y|\mathbf{x})$ , while  $p_{pre}(\mathbf{x}) \neq p_{post}(\mathbf{x})$
2. **Prior probability shift:**  $p_{pre}(\mathbf{x}|y) = p_{post}(\mathbf{x}|y)$ , while  $p_{pre}(y) \neq p_{post}(y)$
3. **Concept shift:** Either  $p_{pre}(y|\mathbf{x}) \neq p_{post}(y|\mathbf{x})$ , while  $p_{pre}(\mathbf{x}) = p_{post}(\mathbf{x})$  or  $p_{pre}(\mathbf{x}|y) \neq p_{post}(\mathbf{x}|y)$ , while  $p_{pre}(y) = p_{post}(y)$

[46] ch. 1, [31]

A distribution shift can be understood as a change in the joint distribution of the input data and labels. By applying Bayes' theorem, we can differentiate between different types of shifts. The most commonly examined forms are the covariate shift, prior probability shift, and concept shift. Covariate shift refers to a change in the distribution of the input data  $\mathbf{x}$ , with the class posterior, i.e., the conditional distribution of the labels given the input data, remaining unchanged. On the other hand, the prior probability shift (also known as label shift) implies a change in the distribution of labels  $y$ , while the likelihood remains unchanged. This type of shift occurs when the number of individual outcomes changes over time. Also known as concept drift, the concept shift indicates changes in the relationship between the input data and its label, which causes changes in the conditional distributions. Although other types of shifts are theoretically possible, they are unnamed and usually not included since they are considered too rare or specific [31].

These shifts occur naturally and can be linked to many causes, such as issues with the dataset, for example, through sample selection bias or imbalanced datasets, or unavoidable environmental changes [46] and can significantly impact the performance of machine learning models [31]. Therefore, it is important to keep these shifts in mind when developing and deploying machine learning models in real-world applications. To create a rigorous framework in which we can model these shifts, we define the concept of tasks.

### 3.2.0.2 Definition: Task

Let  $\mathcal{X} = \{\mathcal{X}_i | \forall i \in Y\}$  be an indexed dataset for a countable set of class labels  $Y$ .

For each index  $t \in \mathcal{T}$  of a countable set of indices  $\mathcal{T}$  starting with 0, we define a task as a tuple  $(\mathcal{X}^t, Y^t, l^t)$  consisting of:

1. A subset of the dataset  $\mathcal{X}^t \subseteq \mathcal{X}$ .
2. A set of labels with  $Y^t = \{0, \dots, |\mathcal{X}^t| - 1\}$ .
3. A labeling function  $l^t : \mathbf{X}^t \rightarrow Y^t$

We call  $t \in \mathcal{T}$  a **task label** and  $|Y^t|$  the **task size**. Furthermore, we call labels  $y \in Y^t$  **within-task labels**.

We say we have **regular** tasks, if we have constant task size  $s = |Y^{t_1}| = |Y^{t_2}| \forall t_1, t_2 \in \mathcal{T}$ . Furthermore, if  $|Y^{t_1} \cap Y^{t_2}| = c \forall t_1, t_2 \in \mathcal{T}$  for a constant  $c \in \mathbb{N}$ , we define  $c$  as the **class intersection size**.

inspired by [25]

These definitions enable the division of the entire dataset into distinct sets of samples. Since, each dataset can be following a mutually independent different distribution, this allows us to model distribution shifts by assigning each task the datasets of the according distributions after the shift. While drawing such borders is often not possible in real-life scenarios, since we often do not know what constitutes single tasks, this modeling can still provide a useful framework when training models and exploring different settings. It allows us to create standardized and comparable scenarios using regular tasks with varying degrees of class intersections.

Additionally, the use of different labels, such as task labels, within-task labels, and general class labels (also known as global labels), provides useful terminology when training. They allow distinguishing between different tasks and even differentiating between discussing classes within a task versus discussing classes in general. Furthermore, by defining these labels, we can model different types of settings as follows.

### 3.2.0.3 Definition: Continual Learning Settings

Given a dataset  $\mathcal{X}$ , with their task labels  $\mathcal{T}$  and within-task labels  $Y^t$ , we distinguish between three settings, depending on which mapping we need to learn:

1. **Task-Incremental Learning:**  $f : \mathcal{X} \times \mathcal{T} \rightarrow Y^t$
2. **Domain-Incremental Learning:**  $f : \mathcal{X} \rightarrow Y^t$
3. **Class-Incremental Learning:**  $f : \mathcal{X} \rightarrow Y^t \times \mathcal{T}$

[55]

These settings are described in ascending order of difficulty, each adding layer of complexity to the labeling. In task-incremental learning, the learning model is provided with the current task context along with the data that it needs to classify. This context helps the model differentiate between the tasks. The domain-incremental setting requires the model to learn without the provided task context. The model must classify the data based on the features only, but only has to provide the within-task label. If the model must additionally infer the task

---

label or, equivalently, provide the global label without any given context, the setting is called class-incremental learning and represents the most challenging scenario. Which setting is ultimately used depends on the specific challenges the continual learning application faces and how the model addresses these challenges, which brings us to the definition of continual learning.

#### 3.2.0.4 Definition: Continual Learning

While there is no universally agreed-upon definition, continual learning can be characterized by several challenges and modeling approaches that provide a clear insight into the scope and objectives that are addressed. [32]

**Challenges** The main challenges faced in continual learning include the following:

1. **Catastrophic Forgetting** describes the degrading performance of a model on a specific task once it has been trained on subsequent tasks.
2. **Knowledge Transfer** refers to the potential ability of a model to apply the knowledge or features learned from one task to other tasks.
3. **Distributional Shift Detection** refers to the need to detect when shifts in the data distribution occur.

[10, 33]

Together, these challenges require accurate detection of distributional shifts, which is essential to employ measures, to hinder the model from essentially forgetting the previously learned information and allow effective knowledge transfer to increase the model efficiency and performance. To address these challenges, models can often be assigned to one of three primary approaches.

#### Approaches

1. **Replay** involves replaying data from previous tasks to preserve learned features. The idea being that by maintaining exposure to older tasks, models can better preserve knowledge across different tasks, mitigating the impact of catastrophic forgetting.
2. **Architectural** methods modify the model structure to mitigate catastrophic forgetting, with the goal of separating the learned features of each task, thereby preventing interference when learning new tasks.
3. **Regularization** strategies aim to restrict the change of model parameters through penalties, in order to avoid overwriting learned features from previous tasks, thus preserving the knowledge encoded in these parameters.

[10, 33]

Each approach targets a different aspect of the entire system, namely the data, model architecture, and parameters, respectively and provide different ideas for designing continual learning models, which also can be combined as needed. For example, one model using the architectural approach is the progressive neural network, on which we base our development.

### 3.3 Progressive Neural Networks

Lastly, we present the general progressive modeling idea of Progressive Neural Networks. Introduced by Rusu, Rabinowitz, et al. [49], they provide a way to tackle the challenges of continual learning through basic architectural modifications of Definition 3.2.0.4. The main idea behind PNNs is to create a new neural network for each new task and connect it to the previous networks via lateral connections. This allows for the preservation of previous knowledge while still enabling the transfer of knowledge via lateral connections to improve the learning of new tasks.

For a neural network with parameters  $\Theta^1$ , let  $l \in \mathbb{N}$  be the number of layers,  $n_i$  be the number of units and  $a_i^{(0)}$  be the activation in the  $i$ -th layer  $i \in \{0, \dots, l-1\}$ . For the  $k$ -th task we create a new neural network called the  **$k$ -th column** with parameters  $\Theta^{(k)}$  and the following  $i$ -th layer activation:

$$a_i^{(k)} = f_i \left( W_i^{(k)} a_{i-1}^{(k)} + \sum_{j=0}^k U_i^{(k:j)} a_{i-1}^{(j)} \right)$$

with  $f_i$  being the non-linearity of the  $i$ -th layer,  $W_i^{(k)}$  being the  $n_i \times n_{i-1}$  weight matrix of the  $i$ -th layer of column  $k$  and  $U_i^{(k:j)}$  being the  $n_i \times n_j$  weight matrix of the **lateral connections** between the  $i$ -th layer of column  $k$  and the  $i-1$ -th layer of column  $j$ .

[49]

When a new task is presented during training, the current parameters are frozen, meaning they are defined as constants in the optimization. To allow the learning of the new task, another column is created with randomly initialized parameters and training proceeds with these new parameters. Furthermore, since the new column is initialized randomly, there is no assumed dependency between tasks, allowing the model to learn completely independent tasks. This feature makes PNNs capable of handling even adversarial tasks in the new column.

By the design of the lateral connections, the temporal order is maintained. The newly learned features can, therefore, not interfere with previous knowledge, preventing catastrophic forgetting. While this design prohibits the backward transfer of knowledge, it allows the forward transfer of previously learned features. Since these lateral connections are not frozen, the model can learn to use the prelearned features as they are or transform them. This can potentially increase convergence speed and improve accuracy. If the previous knowledge is not helpful, the model can learn to ignore it and use the new column to learn entirely new features.

We can now formally develop Progressive Sum-Product Networks to be used in a continual learning setting by using the ideas and concepts of the progressive neural networks to modify SPNs.

---

## 4 Method: Progressive Probabilistic Circuits

---

Using the PNNs, in this chapter, we extend the concept of SPNs to define Progressive Sum-Product Networks (PSPNs), drawing parallels to the extension of neural networks to PNNs. As the application of SPNs in continual learning is a relatively unexplored area, there is a lack of available literature. Therefore, this chapter aims to lay a solid foundation by providing definitions, theories, and a detailed explanation of the methodology of our research.

We divide the explanation of our approach into two main sections describing the established setting and the definition and validation of the PSPN. First, we set the stage for the exploration by establishing the setting in which the experiments are conducted. This includes a detailed specification of the setting itself, as well as a description of why the different choices were made. By identifying the setting upfront, we can design our model and experiments to specifically address it. This involves developing a theoretical framework for PSPNs, based on the principles of SPNs and the specific requirements of our setting. We will begin by outlining their structure and, following the definition, provide proof of the validity of PSPNs. This requires demonstrating that these models fulfill the constraints, ensuring their reliable use as density estimators. To fully describe the entire system, we will then describe the training procedure. This section then discusses which of the continual learning challenges are posed by our setting. Furthermore, we propose a novel design to address a specific limitation we anticipate when using probabilistic models, such as SPNs, instead of traditional deep learning models.

Through the detailed setting and the rigorous theoretical framework for PSPNs, this chapter aims to provide transparency and clarify the scope of our exploration and its limitations.

---

### 4.1 Specification of the Setting

---

Since it is within this setting that we aim to address the challenges and potential advantages of PSPNs, in this section, we want to provide a detailed specification of the setting. For this exploration, we focus on the most basic continual learning task, aiming to avoid hurdles in identifying potential advantages of our approach that might result from increased complexity.

We define our continual learning setting as:

*Transfer learning (3.2) in a task-incremental (3.2.0.3) setting with regular (3.2.0.2) intersecting and non-intersecting (3.2.0.2) classification tasks, using supervised learning of complete data (3.1.1.1) and labels.*

---

Transfer learning forms the basic approach to continual learning, where we reuse models to transfer the previous knowledge to new tasks. We simulate a transfer learning environment by incrementally adding tasks to the learning process. More specifically, we focus on task-incremental learning as it provides the most context of the described incremental learning settings in Definition 3.2.0.3 . For the tasks themselves, we chose regular tasks. This choice allows us to implement a simple data generation without the added complexity of maintaining irregular tasks. Each task is a data classification task, so we can optimize the model as a discriminative one, further simplifying the setting. Finally, we also choose supervised learning with complete data and labels to avoid dealing with incomplete evidence. This choice ensures that our model has all the necessary information to learn effectively.

With the description of the setting concluded, we can now focus on the structure of the PSPN. While our proposed design is tailored to this setting, it can be extended in future research for deployment in other environments.

---

## 4.2 Progressive Sum-Product Network

---

In this section, we explain the design of the PSPN, justifying each modification and proving the validity and tractability of the PSPN, in addition to describing the training procedure of the model. While we define the PSPNs to suit our distinct settings and constraints, we avoid making further assumptions about the dataset or data type. As a result, the implementation details are not specified and may vary across different models. Instead, the outlined structure should be interpreted as a general architecture, which allows us to adapt the final implementation to the different experiments. Moreover, we consciously avoid specifying additional details, such as initialization procedures, in order to keep the focus on the core aspects needed to answer the research question. Overall, our approach aims to specify only the strictly necessary details, to suit the model for our continual learning setting while still allowing for flexible adaptations in order not to limit the exploration of ideas.

### 4.2.1 Modeling

As previously motivated, to solve the introduced setting, we use the progressive modeling techniques derived from PNN to design the model. Since PNN are based on the architectural approach of Definition 3.2.0.4, PSPNs are as well.

The idea is for the model to use parameter freezing, such as with the PNN, to prevent catastrophic forgetting by design. The learning of new tasks is likewise enabled by design through the addition of a new column, which can be constructed like the previous column since the task regularity in the setting ensures similar capacity requirements to the previous column. Furthermore, we add lateral connections, similar to PNN, facilitating the forward transfer of knowledge learned in previous columns. While this design is not absolute, since we are exploring possible modifications, the general structure and terminology will stay the same and are formally defined in the following.

#### 4.2.1.1 Definition: Layerwise SPN

Due to the need for a regular structure and way of selecting individual nodes in the definition of lateral connections, we motivate the definition of layers and indexed nodes in SPNs.

Let  $\mathcal{S} = (\mathcal{G}, \Theta)$  with  $\mathcal{G} = (\mathbf{V}, \mathbf{E})$  be an SPN with nodes  $\mathbf{V} = \mathbf{S} \cup \mathbf{P} \cup \mathbf{L}$

We define a **layer**  $\mathcal{L} \subseteq \mathbf{V}$  as one of the following:

1. **Sum Layer**  $\mathcal{L}_S = \{S_j \in \mathbf{S} \mid j \in \{0, \dots, m\}\}$  is defined as an indexed set of sum nodes.
2. **Product Layer**  $\mathcal{L}_P = \{P_j \in \mathbf{P} \mid j \in \{0, \dots, m\}\}$  defined as an indexed set of product nodes.
3. **Leaf Layer**  $\mathcal{L}_L = \{L_j \in \mathbf{L} \mid j \in \{0, \dots, m\}\}$  defined as an indexed set of leaf nodes.

We say an SPN has a **layerwise** structure, if the underlying directed acyclic graph  $V = \bigcup_{j=1}^n \mathcal{L}_i$  can be partitioned into  $n$  layers  $\mathcal{L}_i$ , for  $i \in \{0, \dots, n\}$  where:

$$\forall e \in E : e \in \mathcal{L}_i \times \mathcal{L}_{i-1} \text{ for } i \in \{1, \dots, n\}$$

i.e. edges only exist from one layer to the next layer in the sequence.

If necessary, we use the indices  $j$  to reference individual nodes  $\mathcal{L}_i[j]$  in a given layer  $\mathcal{S}[i] := \mathcal{L}_i$ .

It must be noted that the structure of a layer does, on its own, not provide any information on the children of any given node. Instead, they can be specified by providing the node indices of the children in the preceding layer. For example, any of our previously shown SPNs (see Figures 3.2, 3.3, 3.4, 3.5) are layerwise structured SPNs. To show, for example, that the simplified SPN structure shown in Figure 3.4 is a layerwise structured SPN, we provide the layers  $\mathcal{L}_0 = \{L_0, L_1, L_2, L_3\}$ ,  $\mathcal{L}_1 = \{P_0, P_1\}$ ,  $\mathcal{L}_2 = \{S_0\}$  and define their child nodes  $ch(\mathcal{L}_2[0]) = \{0, 1\}$  for the top sum layer and  $ch(\mathcal{L}_1[0]) = \{0, 1\}$ ,  $ch(\mathcal{L}_1[1]) = \{2, 3\}$  for the product layer. Note that, we only reference the child nodes through their indices. Since, the SPN can be structured as such, this shows that Figure 3.4 is a layerwise structured SPN.

By definition, this layerwise design does not allow the mixing of different node types. This results in a very restrictive use of leaf nodes by only allowing these in the first layer i.e.  $\mathcal{L}_i = \mathcal{L}_L \iff i = 0$ . However, since both sum and product nodes can act as identities if they have a single child node, this structure does not limit the SPN's expressivity. This fact proves that any SPN can bijectively be transformed into a layerwise structure. While not limiting the expressivity, we must accept the negative impact on the network size caused by the layered structure in order to create the regular structure needed for the definition of the PSPN. This structure is further regularized by organizing the layers into columns.

#### 4.2.1.2 Definition: Column

Let  $\mathcal{S}_0 = (\mathcal{G}_0, \Theta_0)$  be a layerwise structured valid SPNs.

We define  $\mathcal{S}_0$  as the **structural template** and for  $c \in \{1, \dots, k-1\}$  define  $\mathcal{S}_c = (\mathcal{G}_c, \Theta_c)$  with  $\mathcal{G}_c = (V_c, E_c)$  using the same structure as  $\mathcal{S}_0$ .

We then define the **(set of) columns**  $\mathcal{C} = \{\mathcal{S}_0, \dots, \mathcal{S}_{k-1}\}$  as an indexed set of these layerwise structured SPNs, and use indices  $c$  to select an individual **column**  $\mathcal{C}[c] := \mathcal{S}_c$ .

Continuing, from here all SPNs in the set of columns are combined into one graph  $\mathcal{G}_C := (\mathbf{V}_C, \mathbf{E}_H)$  with  $\mathbf{V}_C = \bigcup_{c=0}^{k-1} V_c$  and  $\mathbf{E}_H = \bigcup_{c=0}^{k-1} E_c$

These columns are inspired by the columns of the progressive neural network and, as such, are used for a different task each. Consequently, the column indices directly correspond to the tasks  $t = c$  (see Definition 3.2.0.2), allowing us to use the task label provided by our setting to select the appropriate column for inference. Since the tasks, come in a specific temporal order given by the task label, this allows us to talk about current and previous columns, when their indices are smaller than the given current column index. Because of the fixed structure of the structural template, each column has equivalent layers with the same node types in all indexed areas, i.e.  $\mathcal{C}[c][i][j]$  is the same node type in all columns  $c \in \{0, \dots, k-1\}$ . Moreover, they have children of the same node type and with the same scopes.

So far, the different SPNs in our constructed graph  $\mathcal{G}_C$  are disconnected components and only has the preexisting edges within the individual columns  $\mathbf{E}_H$ . We call the preexisting edges horizontal connections and introduce so called lateral connections between them, to connect these independent columns.

#### 4.2.1.3 Definition: Horizontal & Lateral Connections

Let  $\mathcal{C}$  be columns of layerwise structured SPNs.

Given a node  $N = \mathcal{C}[0][i][j]$  in the structural template, and its children  $ch(N)$  given as a set of indices  $J$ , we define the **horizontal connections** of node  $j$  in layer  $i$  in column  $c$  as the following set of edges:

$$E_H^{(\mathcal{C}[c][i][j])} = \{e = (\mathcal{C}[c][i][j], \mathcal{C}[c][i-1][j']) \mid j' \in J\}$$

Furthermore, if the layer  $\mathcal{C}[c][i]$  is a sum layer, we add **lateral connections** to each node defined as:

$$E_L^{(\mathcal{C}[c][i][j])} = \{e = (\mathcal{C}[c][i][j], \mathcal{C}[c'][i-1][j']) \mid j' \in J \ \forall c' < c\}$$

We denote all horizontal connections as  $\mathbf{E}_H$  and all lateral connections as  $\mathbf{E}_L$ .

We redefine our constructed graph as  $\mathcal{G}_C = (\mathbf{V}_C, \mathbf{E}_C)$  with  $\mathbf{E}_C := \mathbf{E}_H \cup \mathbf{E}_L$  to include the lateral connections of all sum nodes.

As seen in the definition, the horizontal connections are defined using the same column index  $c$  and the same set of children as in the structural template, which means they describe the already existing edges in each column SPN. The lateral connections, on the other hand, are defined using all previous columns  $c' < c$ , which therefore describe new edges between different SPNs, more specifically between a given layer  $i$  in column  $c$  and the previous layer  $i-1$  in all previous columns. This maintains the temporal order of the columns, where knowledge can only flow from previous columns to newer columns and only from bottom to top layers. It is also important to note that, these are only defined in sum nodes, which is a requirement for the validity of the PSPN, as we will see in the later proof.

Furthermore, this formal distinction between the edges within the individual columns and the newly introduced edges between the columns, allows us to later specify what is learned within a given column itself, and what knowledge is reused from the previous columns, since these are specifically intended to facilitate forward transfer. When a new task is introduced, a new column needs to be created by **expanding** the network and adding the lateral connections.

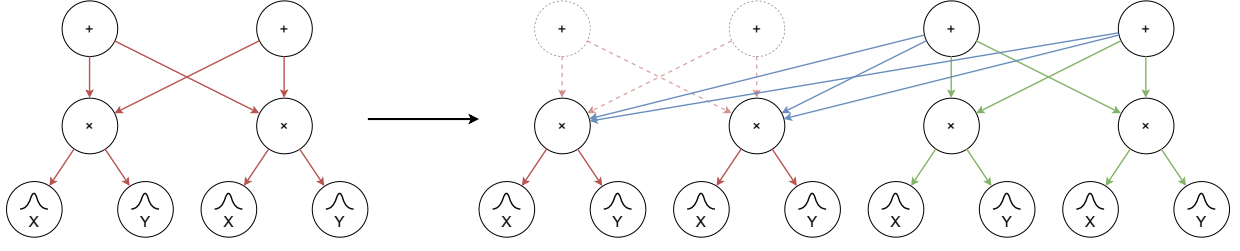


Figure 4.1: This figure shows the expansion process of a simple PSPN seen in Figure 3.2 (without the root node). The colors of the edges indicate the different connections. Red depicts the previous column, green the new column and blue the lateral connections. Note that the structure and scopes of the new column correspond with the structural template.

As seen in the Figure 4.2, a new column is inserted using the structure template of the first column, which is then connected to all the previous columns through the added lateral connections in the sum nodes. Using this structure constructed by the layers, columns, and lateral connections, we can now define the PSPN structure.

#### 4.2.1.4 Definition: Progressive Sum-Product Network

Given the columns  $\mathcal{C}$  of layerwise structured SPNs, and the lateral connections between them, we define the **Progressive Sum-Product Network** as a tuple  $(\mathcal{G}_{\mathcal{C}}, \Theta)$  consisting of the constructed graph  $\mathcal{G}_{\mathcal{C}}$  and parameters  $\Theta$ , which consist of the parameters of all SPNs in  $\mathcal{C}$  as well as the weights for the lateral connections.

As defined, this PSPN is constructed by simply combining all columns and the lateral connections into one directed acyclic graph. Since, this PSPN is not rooted, we define the **induced SPN**  $\mathcal{S}_{R_c}$  of a given column  $\mathcal{C}[c]$ , by extracting the Sub-SPN (see Definition 3.1.1.8) of the root node  $R_c$  of this column. This concept is important, as it describes the inner workings of the PSPN. When classifying with a PSPN, we use the task label  $t$  to index the column and then use the induced SPN  $\mathcal{S}_{R_t}$  for the classification task.

Unlike PNN, we cannot arbitrarily add lateral connections in the PSPN since we have to consider the scopes and the constraints set on SPNs. Instead, in the definition of our model, we add lateral connections only to sum nodes and rely upon the definitions of Sub-SPNs, to prove the validity.

#### 4.2.2 Validity

Given that the PSPN inherits all features from SPNs, both the tractability of probabilistic queries, as well as the guaranteed modeling of correct probability distributions, are subject to the validity of PSPNs. The validity is, thus, a crucial step to ensure that our extension of SPNs maintains the fundamental properties that make SPNs a powerful tool for probabilistic modeling.

Since the addition of lateral connections could potentially disrupt the completeness of sum nodes and, therefore, also interfere with the decomposability of product nodes, the validity of PSPNs is not given by default. However, since we require a valid structural template, and expand only using lateral connections based on the indices of this template, we ensure that the children all have the same scope. The following proof formalizes that

PSPNs are well-defined and correspond to valid SPNs, by building upon the formality derived in the previous sections.

### 4.2.3 Proof of Validity for Progressive Sum-Product Networks

In order to establish the overall validity of an entire PSPN with  $k + 1$  columns, we need to prove all its induced SPNs are valid.

To show:  $\forall c \in \{0, \dots, k - 1\} : \mathcal{S}_{R_c}$  is valid

To do so we use proof by induction on the number of columns, denoted as  $k$ . More specifically, the proof is conducted only on the induced SPN of the most recent column  $\mathcal{C}[k - 1]$ . Therefore, it is important to note that demonstrating the validity of only the most recent induced SPNs is sufficient. Because extensions maintain the temporal sequence of the columns and solely modify the newly added columns, all preceding columns remain unchanged, and their validity holds even after the extension.

**Induction Base**  $k = 1$  We show that all induced SPNs are valid for a PSPN with one column.

$$\mathcal{S}_{R_0} = \mathcal{S}_0 \wedge \mathcal{S}_0 \text{ is valid} \implies \mathcal{S}_{R_0} \text{ is valid}$$

By definition, the induced SPN  $\mathcal{S}_{R_0}$  of the first column is the column  $\mathcal{S}_0$  itself, which is initialized as a valid SPN, implying the validity of the induced SPN. Given that, for  $k = 1$ , this represents all of the induced SPNs, we have successfully proven the base case.

**Induction Step**  $k \rightarrow k + 1$  Assuming the validity of all previous induced SPNs  $\mathcal{S}_{R_c}$ ,  $\forall 0 \leq c \leq k - 1$ , we show that the induced SPN  $\mathcal{S}_{R_k}$  of the latest column  $\mathcal{C}[k]$  is valid after an expansion.

We specifically to prove its validity layerwise, starting from the bottom.

(1) Assuming no lateral connections, we know each SPN  $\mathcal{S}_c \in \mathcal{C}$  in and of itself is valid, since it is based on the structural template  $\mathcal{S}_0$  of a valid SPN. This means that all product nodes are decomposable, and all sum nodes are complete if the lateral connections are omitted.

(2) Hence, considering  $i$  such that  $\mathcal{S}_{R_k}[i] \subseteq \mathbf{S}$  and  $\forall i' < i, \mathcal{S}_{R_k}[i'] \not\subseteq \mathbf{S}$ , we can verify that  $\forall N \in \mathcal{S}_{R_k}[i']$  are decomposable product nodes or leaf nodes using (1), since according to our definition, lateral connections are exclusively introduced to sum nodes. This means when considering the first sum layer, we can verify the decomposability of all preceding layers since these can only consist of product layers with no lateral connections and the leaf layer, which is not subject to any structural constraints. As a result, for any sum node with index  $j$  in the first layer, we can further confirm that all child nodes of the first sum layer  $\forall N, N' \in ch(\mathcal{S}_{R_k}[i][j])$  maintain the same scope  $sc(N) = sc(N')$ .

(3) We show this by distinguish between the horizontal and lateral connections. Given the children of the horizontal connections  $E_H$  of node  $\mathcal{S}_{R_k}[i][j]$  i.e  $\mathbf{CH}_H := ch(\mathcal{S}_k[i][j])$ <sup>1</sup>, we know, that  $\forall N, N' \in \mathbf{CH}_H$  must have the same scopes  $sc(N) = sc(N')$ , due to the completeness of  $\mathcal{S}_k[i][j]$  without lateral connections (see (1)).

Similarly, since the lateral connections  $E_L$  are based on the same indices as the horizontal connections  $E_H$ , we know that the child nodes of all lateral connections  $\mathbf{CH}_L := ch(\mathcal{S}_{R_k}[i][j]) \setminus \mathbf{CH}_H$  must have the same scopes:

$$\forall N \in \mathbf{CH}_L, \forall N' \in \mathbf{CH}_H : sc(N) = sc(N')$$

<sup>1</sup>Note  $\mathcal{S}_k \neq \mathcal{S}_{R_k}$ , they differ since  $\mathcal{S}_k$  has no lateral connections

More specifically, since  $ch(\mathcal{S}_{R_k}[i][j]) = \mathbf{CH}_H \cup \mathbf{CH}_L$  this means that all child nodes have the same scope.

This proves the completeness of the sum node, and since we considered any general sum node  $j$  in the first layer, this proves the completeness  $\forall j$  of the entire layer. Furthermore, since all children of the sum node have the same scope, it follows that their own scope is once again equivalent to the corresponding nodes in the first column. Consequently, we can leverage the validity of the first column to prove the decomposability of all product layers until the next sum layer. By iteratively applying these arguments to each subsequent layer, we prove the decomposability of all newly added product nodes and the completeness of all newly added sum nodes. Moreover, given that all previous columns remain unaltered and the lateral connections maintain the temporal order, all other sum and product nodes also fulfill their respective constraints.

In conclusion, all sum nodes are complete, and all product nodes are decomposable. As a result, the SPN induced by the final column satisfies the completeness and decomposability requirements. We hereby confirm the validity of the entire PSPN, ensuring that our PSPN maintains the desirable properties of SPNs, despite the addition of lateral connections.

#### 4.2.4 Leaf-Node Transformation

Unfortunately, the use of SPNs introduces an inherent limitation when it comes to PSPNs. Unlike neural networks that operate on the feature space, SPNs only define distributions over the features. As a result, once the leaf distributions are set, the subsequent layers can only combine these distributions. The lateral connections, therefore, cannot modify any detected features. This results in limited use of their knowledge in the forward transfer. Because this challenge is unique to our approach, we attempt to overcome this challenge by proposing another novel modification to the PSPN structure. Motivated by these described limitations, the following proposal allows for the transformation of learned distributions.

This modification, which we refer to as the **Leaf-Node Transformation**, aims to overcome this limitation. We introduce a mechanism that allows newer columns to simulate the transformation of prior learned leaf node distributions. For instance, if a leaf node  $L$  models a Gaussian distribution with parameters  $\mu$  and  $\sigma$ , the transformation needs to be of the form

$$f_L(\mu_L, \sigma_L) = (w_\mu \mu_L + b_\mu, w_\sigma \sigma_L)$$

which allows for the scaling and shifting of the mean and the standard deviation scaling.

To maintain the characteristics of an SPNs, this transformation should ideally be simulated using existing SPN structures. Furthermore, the model needs access to randomly initialized leaf nodes to maintain the premise that we do not assume any task relations. This results in an adaptation of the extension procedure.

As seen in [Figure 4.2](#), which shows the modified expansion process with a leaf node transformation, we introduce a modification where each leaf node in the new column is replaced with a sum node mixture of two leaf nodes. One leaf node is pre-initialized with the learned parameters from the previous column, and the other one is randomly initialized. This modification fulfills our requirement of using only SPN components while enabling the adaptation of previously learned leaf node parameters and providing access to the random initialization. Nevertheless, given the change in the architecture, we still need to verify the scopes and ensure the validity of the modified approach. As a result, we provide the following proof of validity.

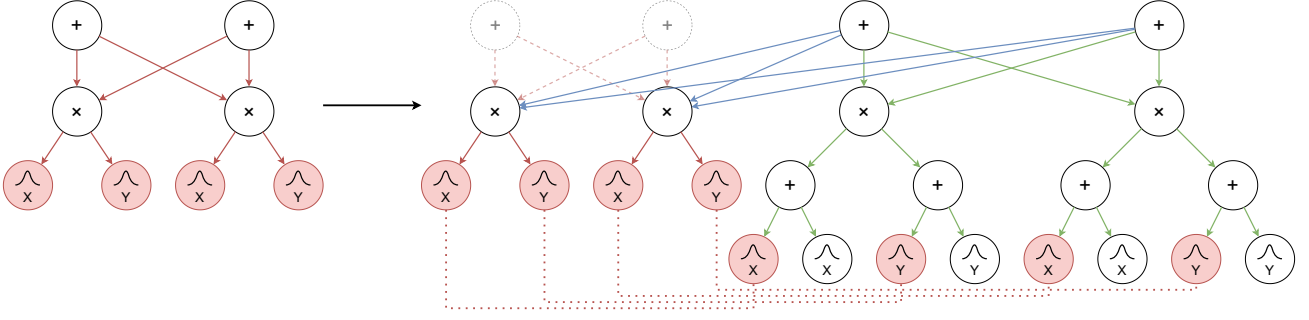


Figure 4.2: This figure shows the modified expansion process of a simple PSPN seen in [Figure 3.2](#) (without the root node). The edge colors indicate the different connections. Red depicts the previous column, green depicts the new column, and blue depicts the lateral connections. Note that the structure of the structural template is adapted to include sum nodes over the same scope, instead of the leaves. Its children are a copied leaf node, depicted in red, and a randomly initialized leaf node, depicted in white. Which leaf node is copied is indicated by the dotted red lines.

#### 4.2.4.1 Proof of Validity

To prove the validity of this new architecture, we can rely on the proof provided for the base version of PSPNs and demonstrate that our adaptation still satisfies the constraints. As outlined in the background chapter in [Definition 3.1.2.4](#), we can use the interchangeability of any valid Sub-SPN within the network with another valid Sub-SPN over the same scope. In our case, we need to demonstrate that the modification can be described by a valid sub-SPN and does not alter the scope.

We note that according to the definition of the PSPN structure, the corresponding nodes in corresponding layers have the scope in all columns. Since our construction uses two leaf nodes with the same index as the corresponding leaf node they replace, it follows that they are defined over the same scope. As a result, the introduced sum node fulfills the completeness constraint, which results in a valid sub-SPN structure. Additionally, this means its scope remains identical to that of the replaced leaf node, which proves the validity of our modification.

#### 4.2.5 Training

Since both the base PSPN and the Leaf-Node Transformation reuse the predefined structure of the first column for all expansions in the PSPN, this mitigates the need for any structure learning algorithms. Instead, we analyze what distributions the tasks might need and manually construct the needed structure for each experiment using the methods described in the background chapter in [Definition 3.1.2.4](#).

For parameter learning, we acknowledge that, following our proof for the validity of PSPNs, all standard parameter learning algorithms for SPNs can be employed. For a given structure, the parameters are learned through basic optimization methods. These methods involve using the gradient to minimize a predefined loss or probabilistic methods such as expectation maximization to maximize the likelihood of the given data [50]. While the probabilistic nature of PSPNs suggests the use of probabilistic algorithms such as the Expectation-Maximization (EM) algorithm, our goal is to explore and compare different approaches. Therefore, motivated by our attempt to avoid the introduction of any unnecessary complexity and gain a clearer insight into the potential benefits, we instead use a gradient descent algorithm to estimate both the weights and the

---

parameters of the distributions. Gradient descent is popular optimization algorithm in machine learning that iteratively adjusts the parameters to minimize a given cost function. It is used as it is readily available in widely used libraries, such as PyTorch, thus simplifying the implementation process.

By combining a detailed setting, a rigorous theoretical framework for PSPNs, and a general training methodology, this chapter provides the solid foundation required for our exploration in the following chapter.

---

## 5 Experiments: Exploration of Progressive Probabilistic Circuits

---

In this chapter, our objective is to determine the strengths and weaknesses of our approach and thereby assess whether the characteristics of SPNs facilitate or hinder the integration of SPNs within continual learning settings. We aim to answer our primary research question: What is the impact of using PSPNs on accuracy, convergence speed, and avoidance of forgetting in continual learning tasks? Our primary hypothesis is that PSPNs, through forward transfer, will improve accuracy and convergence speed and that forgetting will be avoided by design. To test this hypothesis, we perform a comprehensive evaluation of PSPNs through a series of experiments using the defined setting.

To enable this evaluation, this chapter serves as a detailed report of all experiments conducted within this exploration. The experiments are presented sequentially, explaining the knowledge gathered from each experiment and how it informed the modifications for subsequent model iterations or experiments. We have categorized our experiments and findings into six main explorations. The first three are conducted on synthetic datasets, providing a proof of concept and allowing us to look into the intricacies of our approach in a well-understood setting. The remaining three explorations are performed on more complex datasets, namely MNIST [23], SVHN [34], and CIFAR10 [22], to test the robustness and adaptability of our approach in more challenging scenarios. Given that the implementation for synthetic data and image data differs, we will clarify these differences in each corresponding section. We also outline the setup of the experiment, by detailing the experiment and evaluation methods, the dataset, the model, and any hyperparameters used for training procedures. This level of detail is intended to ensure reproducibility and provide a clear understanding of the environment in which our model operates and the specific challenges it needs to overcome. Following these descriptions, we will go into more detail for each experiment, outlining its objectives as well as objectively presenting our results and provide a brief analysis and evaluation of these findings. These will explain our motivations for the subsequent experiment and highlight any general advantages or limitations discovered. In the final section, we then conduct a thorough analysis of the best-performing model and using the CLEVA Compass [32], increase the comparability to any future work in this new research area. We summarize all findings from our experiments in terms of their implications for the final model. Using all theoretical concepts derived thus far, we will also evaluate any further measures relevant to continual learning and potential real-world deployments, providing a holistic view of its performance and applicability.

We evaluate the performance of this new architecture by analyzing the performed experiments with the methods described in the following sections.

---

## 5.1 General Evaluation Methods

---

This section aims to introduce the general evaluation methods used in this chapter. Note that the individual experiments are not all evaluated with every method described in this section. Instead, which of the specific methods is used is described in the setup section of the experiments.

The results of these experiments will be analyzed using various evaluation methods, and specific metrics to also provide a mathematically rigorous evaluation. As suggested in Mundt, Braun, et al. [32], we resort to standardized metrics to increase comparability. Generally speaking, we aim to analyze two main components in each experiment, namely the performance and knowledge transfer.

### 5.1.1 Performance Evaluation

We evaluate the performance of the model based on its final classification capabilities and training speed, mainly using accuracy metrics and the accuracy learning curve.

Given the classification accuracy  $a_{i,j}$  of a model on task  $j$  after being trained on task  $i$ , we construct the classification accuracy per task  $a = \{a_{i,j}\}_{0 \leq i,j \leq T} \in \mathbb{R}^{T \times T}$ , where  $T$  represents the total number of tasks [32]. While the entries below the diagonal of a given classification accuracy matrix could potentially provide insights about forgetting, we do not focus on this aspect as our design inherently prevents forgetting. Similarly, the entries above the diagonal could theoretically be used to calculate forward transfer. However, in the context of PSPNs, these entries cannot be computed since the relevant columns are only created when a new task is introduced. Instead, we will use the per-task accuracy matrix to demonstrate performance on each task as the training progresses, as well as to calculate an average accuracy for the entire experiment if necessary.

The plotted accuracy over training steps (accuracy learning curve) will provide additional visual insight into the speed of classification while also confirming the final accuracy.

### 5.1.2 Transfer Evaluation

Our evaluation of knowledge transfer primarily uses the loss curve, the transfer score and a so-called perturbation analysis. While the accuracy learning curve also provides a visual insight, it also allows us to calculate the transfer score metric derived in Rusu, Rabinowitz, et al. [49], which provides a definitive metric for the effectiveness of forward transfer in the model. For a given column, the transfer score is calculated as the relative area under the learning curve, which corresponds to the accuracy curve in our setting. Hereby, the relative area means the area under the curve of a column in the PSPN, compared to a single-column SPN baseline trained on the corresponding task only. While this metric measures the amount of knowledge transfer that has occurred, it does not provide a way of locating where this forward transfer has taken place within the model.

In line with the approach used in progressive neural networks, we will employ perturbation analysis to research these details. This analysis involves introducing Gaussian noise into layers in previous columns and measuring the consequent performance decrease to assess how much a given column contributes to the final output [49]. However, for the PSPN, this analysis requires modification. The probabilistic interpretation of each node does not allow perturbation of any layer activations, as this disrupts the interpretation of the outputs as probabilities. Instead, we can only add noise to the input of the leaf layers, which limits us to the location of only the column providing knowledge transfer.

---

## 5.2 Exploration on Synthetic Data

---

Our exploration begins with experiments on synthetic data, which provides a controlled environment to understand the practicality and inner workings of our method. We perform three experiments, beginning with a Proof-of-Concept where we identify any performance differences, and show the general viability of using PSPN for our continual learning setting. Furthermore, we provide indicative results signifying that our concerns regarding the limitations of SPNs for forward transfer may be valid. To this end, we evaluate the proposed Leaf-Transformation method in [Section 4.2.4](#) and in our last experiment, build upon the idea by simplifying it into a initialization technique.

### 5.2.1 Setup

**Implementation** For our synthetic data, we choose 2D data points due to their simplicity when visualizing. We generate these data points by creating  $x$  and  $y$  coordinates for each sample using general scientific packages such as scikit-learn [39] and NumPy [15]. To then find a model of our PSPN architecture that is compatible with this data type, we pick the needed structure manually by using the methods described in [Section 3.1.2.4](#). The system is then implemented using Python and NumPy [15], along with other scientific packages such as Matplotlib [16] for visualizations and PyTorch [37] for parameter optimizations. This implementation is used for all subsequent experiments involving synthetic data.

**Evaluation** For the metrics used, we generally used the element-wise median of 50 repetitions for all metrics and curves as the basis for each evaluation. The median was chosen over the mean to avoid the influence of outliers, resulting from our implementation. The mean was only used when explicitly stated. Since the synthetic experiments require only small model sizes, and the data complexity is manageable, we can gain deep insights into the model without the need for complex analyses such as the perturbation analysis.

**Dataset** In these experiments, we use synthetically generated data to allow for complete control over the complexity and composition of the tasks, providing increased flexibility in the experiments. We use data sampled from a Gaussian distribution and a slightly more complex toy dataset called moons. The use of two datasets allows for a more meaningful evaluation using the more complex dataset, while still allowing individual tests with the well understood Gaussian data, to help understand specific intricacies.

Given the dataset for the initial task, we perform four selected transformations on the different classes to generate the next tasks for the PSPN, thereby creating four unique test cases for the experiment. These test cases are:

1. **Independent Tasks (IND)** We generate the same number of classes randomly.
2. **Fixed-Cluster Tasks (FC)**: We fix one of the clusters and generate the other classes randomly.
3. **Mean-Shifted Tasks (MS)** We generate new classes by randomly shifting the means of the data clusters.
4. **Variance-Scaled Tasks (VS)**: We generate new classes by randomly scaling the variance of the clusters.

For the Gaussian data, each class  $c$  is generated by selecting two random means  $\mu_x, \mu_y \sim U(0, 1)$  and two random standard deviations  $\sigma_x, \sigma_y \sim U(0, 0.1)$ , and sampling from the following distribution:

$$\mathcal{N}_c\left(\begin{pmatrix} \mu_x \\ \mu_y \end{pmatrix}, \begin{pmatrix} \sigma_x & 0 \\ 0 & \sigma_y \end{pmatrix}\right)$$

We repeat this for all classes in the first task and then apply each of the four transformations as illustrated by the example of the four test cases generated from the same initial task in [Figure 5.1](#). One entire dataset in this example consists of two tasks with two classes each.

For the moon datasets, we generate the required number of classes using scikit-learn. During the generation, Gaussian noise is added, using a random standard deviation  $\sigma \sim U(0, 0.2)$ . The generated data is then normalized to coordinates in the range of  $[0, 1]$  and further transformed by shifting the center of the data to randomly selected coordinates  $m_x, m_y \sim U(0, 1)$  and applying two independent scalings in the x and y direction, with randomly selected scaling factors  $s_x, s_y \sim U(0, 1)$ . This is done to generate a wide variety of data while providing similar general settings to the Gaussian data and ensuring interoperability. [Figure 5.2](#) shows another example of the four test cases, this time using the moon dataset to generate these cases.

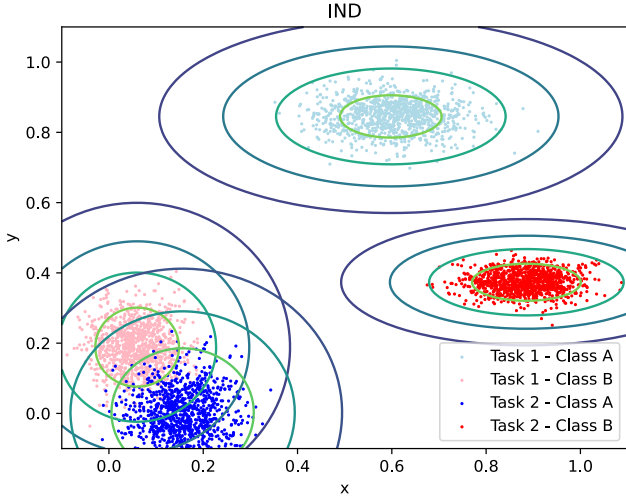
In the following experiments, we generate three tasks consisting of three classes each. For each of these four classes, we generate 10,000 samples.

**Model** Since these datasets are simple enough, we derive an appropriate structure manually by using the methods described in [Section 3.1.2.4](#). For the Gaussian datasets, each cluster is generated by two independent univariate Gaussian distributions. Therefore, we can model each class as a product node with two univariate leaf nodes over the two dimensions, respectively. To enable the use of lateral connections, we also add a sum layer as a classification head of the network as shown in [Figure 5.3](#). This figure shows the base structure used as initialization for the PSPN, with 12 parameters. After two expansions this results in a total of 36 parameters for the induced SPN. For the expansion, we refer to [Figure 4.2](#). This structure is further used for an SPN trained solely on the last task, which acts as a baseline for any comparisons.

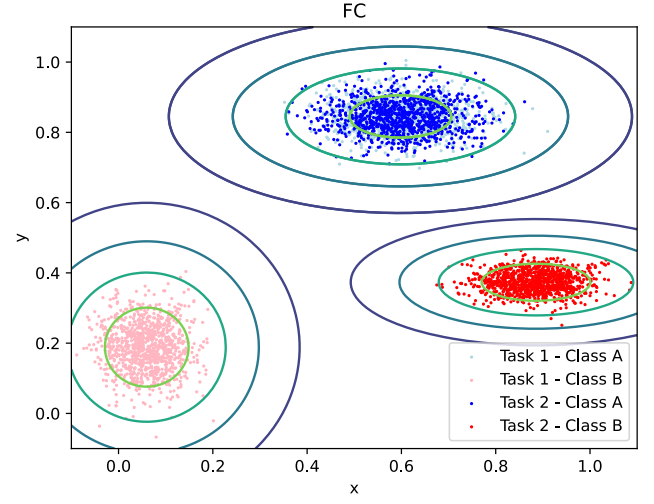
For the moon dataset, we adapted the previous model to have the capacity to model mixtures of six Gaussian distributions, resulting in 36 parameters for the structural template and 108 parameters in total for the induced SPN of the last column. This is done with the motivation of enabling the model to approximate the half-circle using three distributions each. [Figure 5.4](#) shows the SPN used for the baseline as well as the structural template of the PSPN for the moon data.

All model parameters are initialized randomly, with the leaf distribution being initialized using the same uniform distributions as used in the data generation.

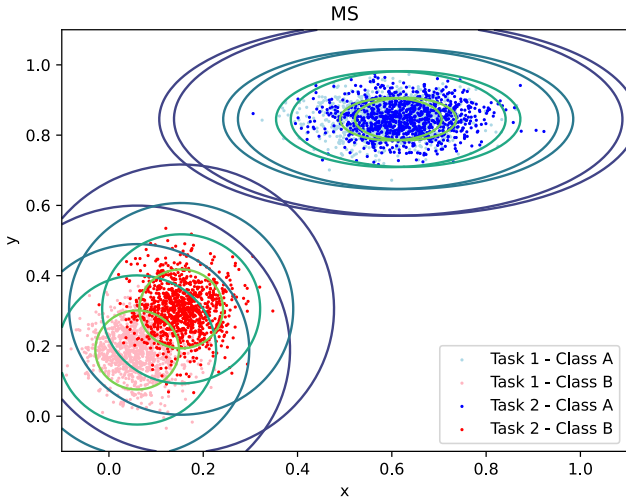
**Training** Each task was trained using 200 mini-batches of batch size 100 with a learning rate 0.02 (0.03 for moon-model) using the Adam optimizer [17]. The optimization objective was the Negative Log-Likelihood Loss, which we used to optimize the class-conditional likelihoods.



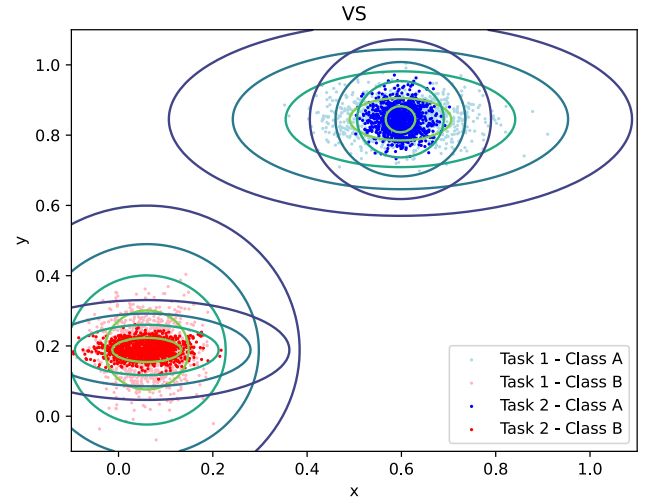
(a) This subfigure shows the Independent test case, where the new clusters for each class are chosen independently from the previous task.



(b) This subfigure shows the Fixed-Cluster test case, where the new clusters for class A is fixed and the new cluster for class B is chosen independently from the previous cluster.

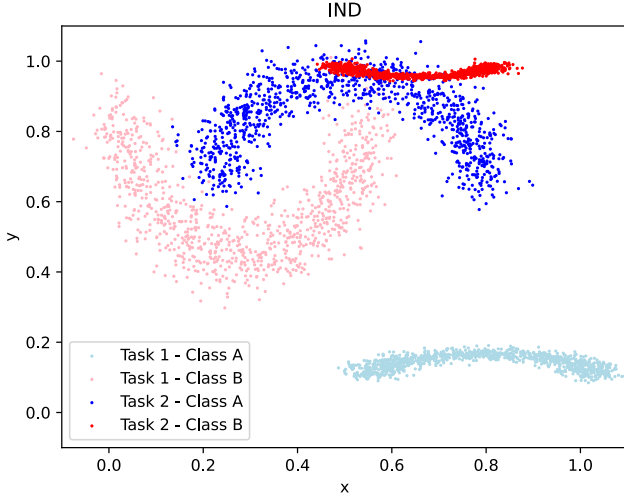


(c) This subfigure shows the Mean-Shift test case, where the new clusters are generated by shifting the mean of the previous clusters, but keeping the variance constant as seen in the contour lines.

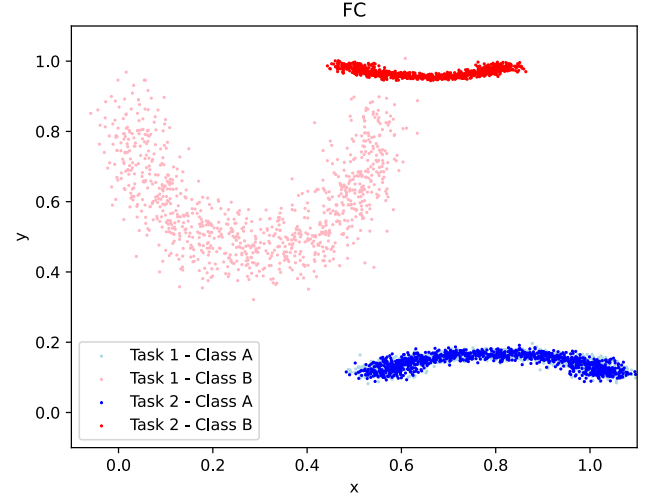


(d) This subfigure shows the Variance-Scaled test case, where the new clusters are generated by keeping the mean of the previous clusters constant, but scaling the variance as seen in the contour lines.

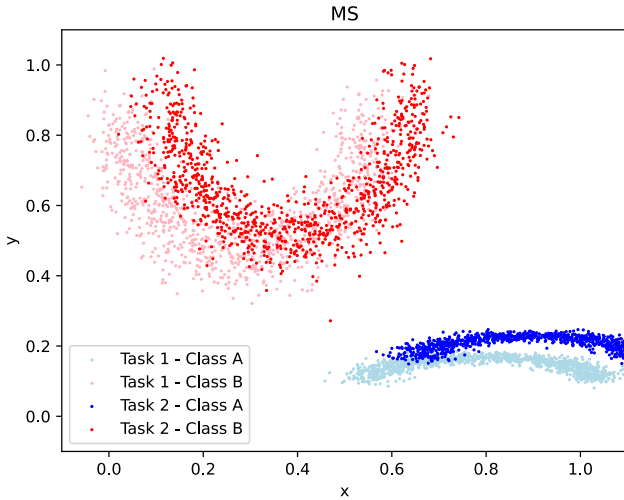
Figure 5.1: This figure visualizes the four test cases for the synthetically generated Gaussian datasets. The contour lines show the multivariate Gaussian used to generate each data cluster. The color hue indicates the membership to the within-task classes, while the saturation indicates membership to the tasks.



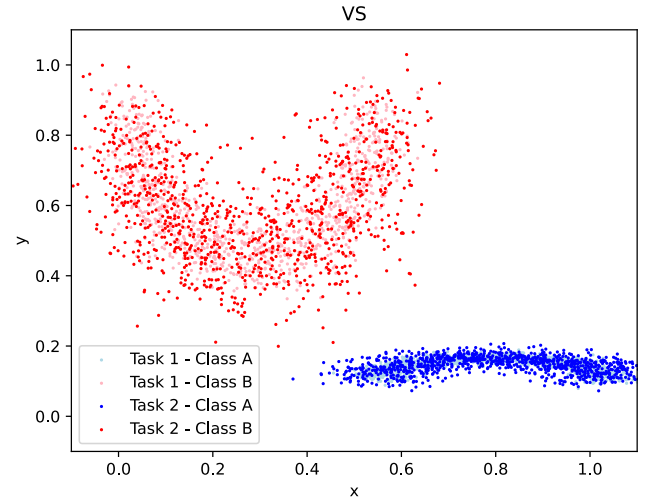
(a) This subfigure shows the Independent test case, where the new moons for each class are chosen independently from the previous task.



(b) This subfigure shows the Fixed-Cluster test case, where the new moon for class A is fixed and the new moon for class B is chosen independently from the previous moon.



(c) This subfigure shows the Mean-Shift test case, where the new moons are generated by shifting the center of the previous moon, but keeping the variance of the added Gaussian noise constant.



(d) This subfigure shows the Variance-Scaled test case, where the new moons are generated by keeping the center of the previous moon constant, but scaling the variance of the added Gaussian noise.

Figure 5.2: This figure visualizes the four test cases for the synthetically generated moon datasets. The color hue indicates the membership to the within-task classes, while the saturation indicates membership to the tasks.

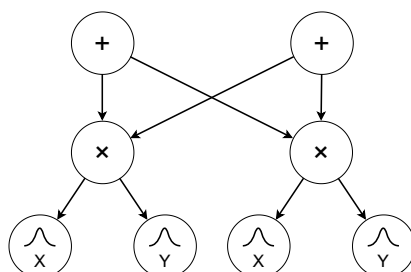


Figure 5.3: This figure shows the structural template used for experiments with Gaussian data. The model contains two product node with two univariate leaf nodes over the two dimensions, each. To this product layer a sum layer with two nodes for the two classes is added as a classification head, which allows for lateral connections during the expansion.

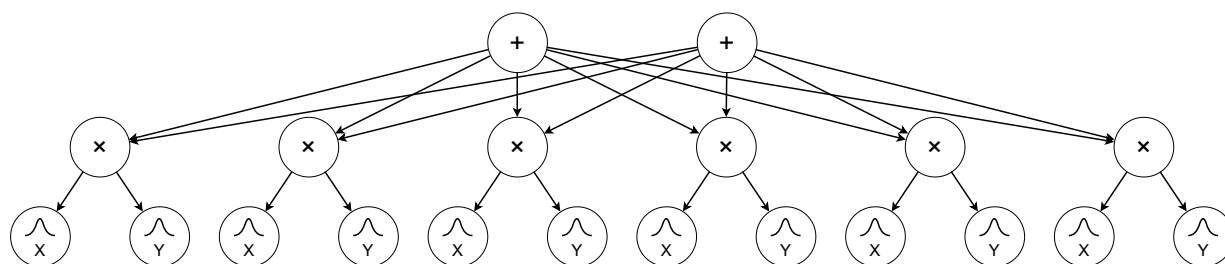


Figure 5.4: This figure shows the structural template used for experiments with moon data. The model contains six product node with two univariate leaf nodes over the two dimensions, each. To this product layer a sum layer with two nodes for the two classes is added as a classification head, which creates a mixture of the product nodes and allows for lateral connections during the expansion.

| Median Accuracy |     |   | Task Learned |      |      |              |     |     |
|-----------------|-----|---|--------------|------|------|--------------|-----|-----|
|                 |     |   | PSPN         |      |      | SPN Baseline |     |     |
|                 |     |   | 0            | 1    | 2    | 0            | 1   | 2   |
| Task Evaluated  | IND | 0 | 1.0          | 0.52 | -    | 1.0          | -   | -   |
|                 |     | 1 | 1.0          | 1.0  | 0.54 | -            | 1.0 | -   |
|                 |     | 2 | 1.0          | 1.0  | 1.0  | -            | -   | 1.0 |
|                 | FC  | 0 | 1.0          | 0.51 | -    | 1.0          | -   | -   |
|                 |     | 1 | 1.0          | 1.0  | 0.49 | -            | 1.0 | -   |
|                 |     | 2 | 1.0          | 1.0  | 1.0  | -            | -   | 1.0 |
|                 | MS  | 0 | 1.0          | 0.48 | -    | 1.0          | -   | -   |
|                 |     | 1 | 1.0          | 1.0  | 0.5  | -            | 1.0 | -   |
|                 |     | 2 | 1.0          | 1.0  | 1.0  | -            | -   | 1.0 |
|                 | VS  | 0 | 1.0          | 0.5  | -    | 1.0          | -   | -   |
|                 |     | 1 | 1.0          | 1.0  | 0.53 | -            | 1.0 | -   |
|                 |     | 2 | 1.0          | 1.0  | 1.0  | -            | -   | 1.0 |

Table 5.1: This table shows the median per-task metrics of PSPN and SPN on moon data for both the PSPN and the SPN Baseline on each test case. Note that all diagonal entries are 1.0, indicating a perfect classification on each task. The entries below the diagonals are also equal to 1.0 for the PSPN, indicating that forgetting is avoided. For the SPNs, these values are not given as we only consider a one-column SPN Baseline. If these were reused, we would see identical performance. Further note that the entries above the diagonal are approximately equal to 50%, which results from the equal mixture of the random initialization in the classification head.

## 5.2.2 Proof-of-Concept on Synthetic Data

This initial experiment serves as a Proof-of-Concept. We aim to test the performance of our PSPN model. To do so, we use well-understood datasets, which allow us to observe the behavior of our model in a controlled setting and identify potential issues or limitations early in the exploration. The results from this experiment inform the design and implementation of subsequent experiments, helping us refine our approach as we progress toward more complex datasets.

### 5.2.2.1 Results

We begin with a basic evaluation of the final performance of the PSPN. This performance evaluation compares the per-task metrics of the PSPN to the SPN baseline on the moon dataset. In [Table 5.1](#) we observe that both models reach an identical final accuracy of 100% in the per-task metrics. Furthermore, we can confirm that forgetting is avoided. This is indicated by the 100% classification accuracy displayed in all entries below the diagonal. We also note that no forward transfer is detected by the per-task metric, as entries above the diagonal reveal a value close to 50%. This can be explained by the lateral connections, which due to random initialization, equally mix all leaf distributions for both classes.

To gain an understanding of the transfer, we instead use the transfer score, which encloses information about convergence speed by measuring the area under the accuracy curve. Using this measure in [Table 5.2](#), we observe that the PSPN, in the independent case, yields a transfer score close to 0%, signifying a negligible

| Task Evaluated        | IND  | FC   | MS   | VS   |
|-----------------------|------|------|------|------|
| Transfer Score (in %) | 0.52 | 1.14 | 1.34 | 4.58 |

Table 5.2: This table shows the median transfer score of the PSPN on moon data. Note that the Independent test case, shows a value close to 0%, indicating that practically no transfer took place. Further note that the Fixed-Cluster and Mean-Shifted test cases only show minimal transfer, and only the Variance Scaled test case shows a significant transfer.

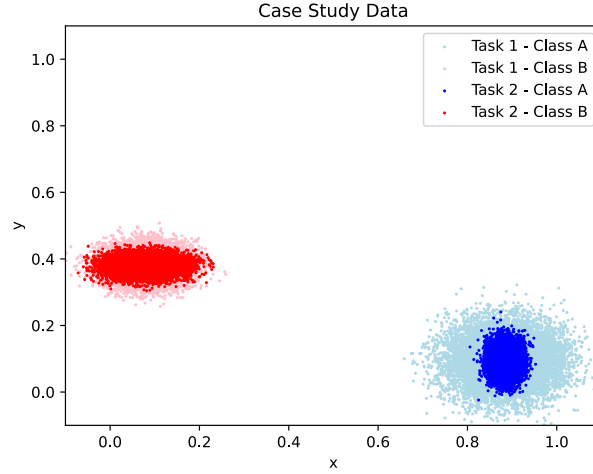


Figure 5.5: This figure shows the Variance-Scaled Gaussian dataset used for the case study. It consists of two tasks with two classes each. The classes are generated by the parameters  $\mu_x = 0.07, \mu_y = 0.38$  and  $\sigma_x = 0.07, \sigma_y = 0.03$  for the first class and  $\mu_x = 0.88, \mu_y = 0.1, \sigma_x = 0.06, \sigma_y = 0.08$  for the second class.

amount of transfer. In comparison, both the mean-shifted and fixed-cluster cases demonstrate a slightly higher transfer score. Only in the variance-scaled case can we identify a relatively high transfer score.

In order to further analyze this, we calculate the optimally reachable accuracy through two single multivariate Gaussians to be 94%. This optimal accuracy is determined empirically using the original data distributions to calculate the posterior for each data sample for the classification. Due to the possibility of overlapping clusters, this optimal accuracy is lower than 100%, when using the two single Gaussians. However, the mean accuracy that the PSPN reaches on the Gaussian data in the variance-scaled case is 96%. Notably, the PSPN performs better than the original distribution, which is only possible if the model is able to estimate the single Gaussian data samples with multiple Gaussians. This allows for the modeling of randomly occurring clusters within the samples.

A further exploration of this increased performance, is conducted using a case study with a Gaussian dataset generated from the Variance-Scaled test case as shown in Figure 5.5. The test case uses classes generated by the Gaussian parameters  $\mu_x = 0.07, \mu_y = 0.38$  and  $\sigma_x = 0.07, \sigma_y = 0.03$  for the first class and  $\mu_x = 0.88, \mu_y = 0.1, \sigma_x = 0.06, \sigma_y = 0.08$  for the second class. The PSPN model is then trained by optimizing the likelihood on 20,000 data points, using the training parameters established in the setup.

To analyze the transfer in a rudimentary way, we inspect the learned weights of the sum node to see a difference in the weight distribution and looking at the learned leaf distributions to see if the correct distributions are learned. The first column of the leaf parameters in Figure 5.6 reveals that the correct distributions are indeed

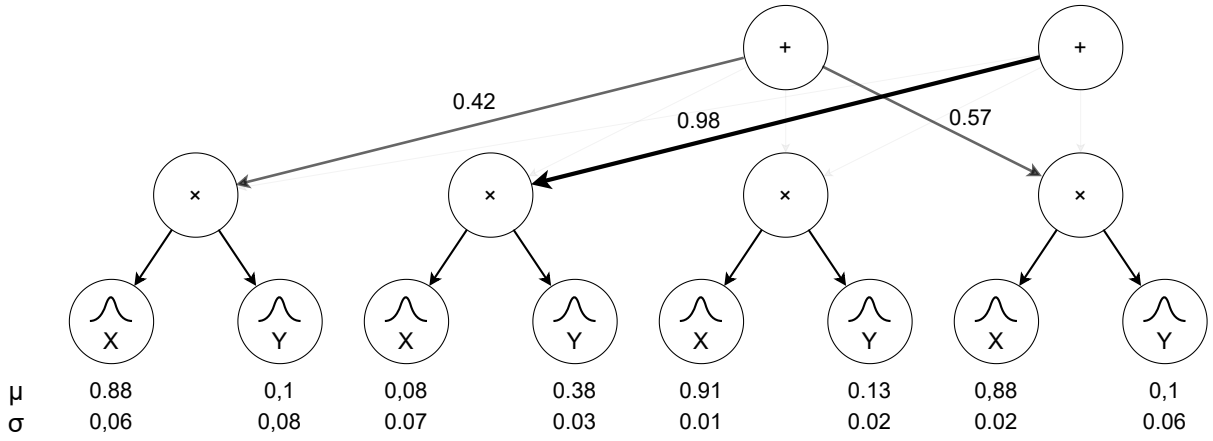


Figure 5.6: This figure shows the induced SPN of the second column after training on the case study data for 200 batches. The learned weights are represented by the opacity of the edges as well as given numerically for the significant edges. Note that the leaf nodes in the four leftmost leaves learn the correct parameters of the dataset. Further, note that the left sum node in the classification head mixes two multivariate Gaussian equally to represent a single Gaussian class.

learned for the first task. Furthermore, viewing the lateral connections, we verify that we have high forward transfer, as the PSPN reuses the distributions learned in the first column as well. Lastly, we can see that the first classification root node describes a mixture, which means the data cluster generated by a single Gaussian is modeled by two of the product-node sub-SPNs, each capable of modeling a 2D cluster.

Plotting the decision boundaries of the first and second columns in Figure 5.7, we can see that, while the model still correctly classifies the data, in the second column, it learns a decision boundary skewed towards the decision boundary of the first column, instead of the true decision boundary showed in the left figure.

We finally continue optimizing the likelihood by training the same model on 20,000 additional data points to see if the correct distributions are learned when more data points are given. To this end, Figure 5.8 shows that the second node does learn to focus on one leaf node and learn the correct parameters, while the other node still uses the previous column for the data distribution.

### 5.2.2.2 Analysis & Evaluation

While we acknowledge that the case study is only an indication of the limited usability of forward transfer, it provides valuable insights into the behavior of the PSPN model.

Since the PSPN performed better than the original distribution, we hypothesized that the model overfits, which we could confirm in the case study, where the model used a mixture to simulate a single Gaussian. This capacity is not available in the first column and the baseline SPN, which is why these learn the correct distributions. Only when trained long enough the model learns the true distributions. However, the other node was still stuck on the incorrect previous distribution. We presume that this was the case because of the relatively low variance scaling for the generation of the second task, as seen in the data in Figure 5.5. This might have resulted in a high enough overlap between the classes, for the distribution to be effectively reused.

As seen in the decision boundaries, reusing previous distributions can impact the ability to generalize to unseen data. This result highlights the importance of the need for distribution transformations. The current

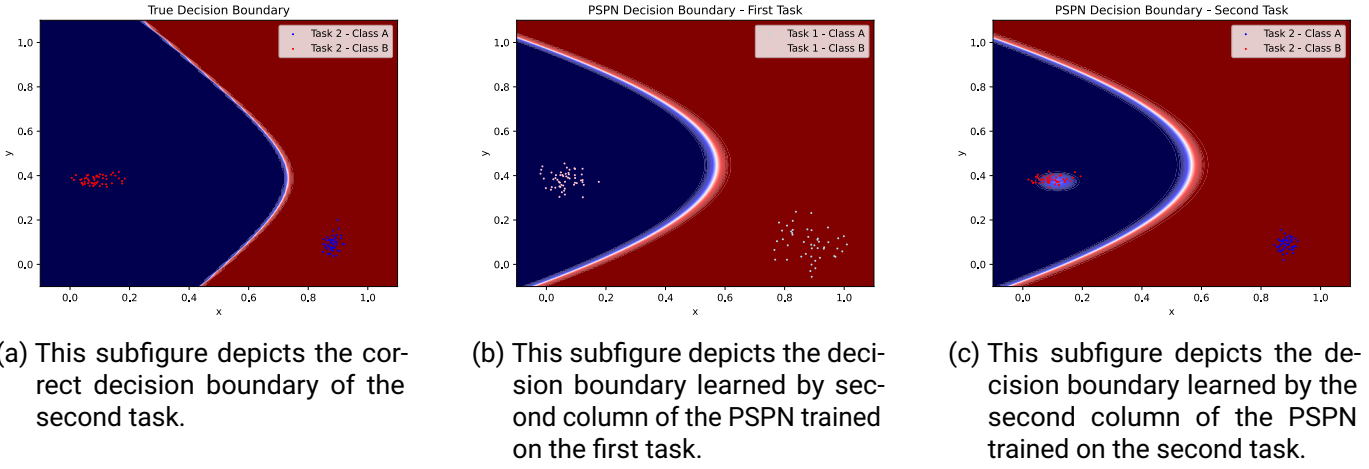


Figure 5.7: This figure shows the decision boundaries of the case study data, as well as some data points of the given task. Note that the decision boundary learned by the second column is skewed towards the decision boundary learned by the first column and is not equal to the true boundary.

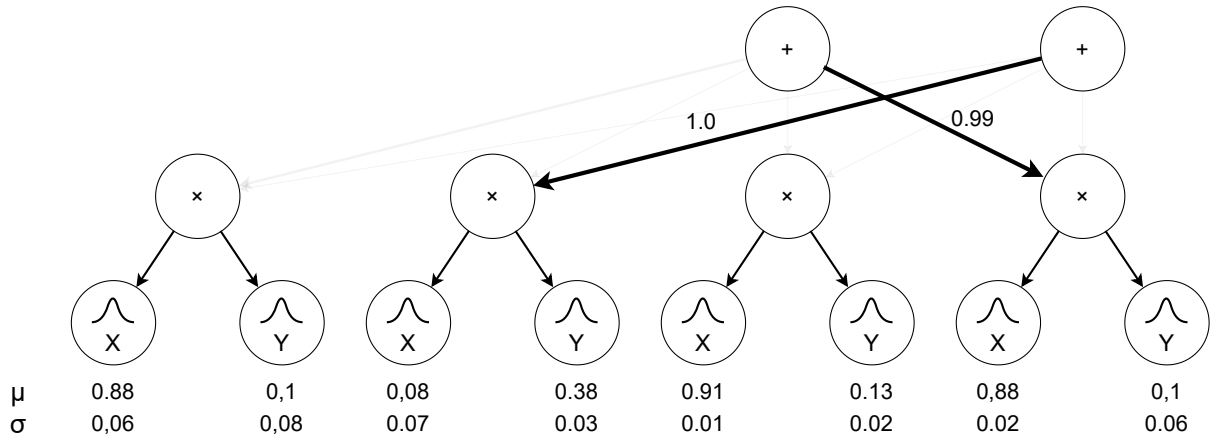


Figure 5.8: This figure shows the induced SPN of the second column after training on the case study data for 400 batches. The learned weights are represented by the opacity of the edges as well as given numerically for the significant edges. Note that the leaf nodes in the two rightmost leaves have learned the correct parameters of the dataset. Further, note that the left sum node in the classification head now uses a single product node to represent the single Gaussian class.

| Median Accuracy |     |   | Task Learned |      |      |              |     |     |
|-----------------|-----|---|--------------|------|------|--------------|-----|-----|
|                 |     |   | PSPN-LT      |      |      | SPN Baseline |     |     |
|                 |     |   | 0            | 1    | 2    | 0            | 1   | 2   |
| Task Evaluated  | IND | 0 | 1.0          | 0.52 | -    | 1.0          | -   | -   |
|                 |     | 1 | 1.0          | 0.89 | 0.52 | -            | 1.0 | -   |
|                 |     | 2 | 1.0          | 0.89 | 0.9  | -            | -   | 1.0 |
|                 | FC  | 0 | 1.0          | 0.53 | -    | 1.0          | -   | -   |
|                 |     | 1 | 1.0          | 0.93 | 0.48 | -            | 1.0 | -   |
|                 |     | 2 | 1.0          | 0.93 | 0.92 | -            | -   | 1.0 |
|                 | MS  | 0 | 1.0          | 0.51 | -    | 1.0          | -   | -   |
|                 |     | 1 | 1.0          | 0.91 | 0.47 | -            | 1.0 | -   |
|                 |     | 2 | 1.0          | 0.91 | 0.92 | -            | -   | 1.0 |
|                 | VS  | 0 | 1.0          | 0.52 | -    | 1.0          | -   | -   |
|                 |     | 1 | 1.0          | 0.94 | 0.51 | -            | 1.0 | -   |
|                 |     | 2 | 1.0          | 0.94 | 0.9  | -            | -   | 1.0 |

Table 5.3: This table shows the median per-task metrics of PSPN and SPN on moon data for both the modified PSPN and the SPN Baseline on each test case. Note that the diagonal entries for the PSPN drop up to 10% compared to the SPN Baseline, indicating a problem with the model. Further, note that again the entries below the diagonals are equal to the diagonal entries, confirming that forgetting is avoided.

PSPN model struggles with this, as it can only scale the entire distribution but cannot transform distributions into other distributions. To this end, we again refer to the limitation that we model probability distributions over the feature space and cannot transform distributions into other distributions by accessing only the output. This means we cannot change the estimated probability of detected features except for scaling the entire distribution by a factor in the mixture.

To address this limitation, we allow the model to change and access the parameterization of the distributions by using the Leaf-Transformation, as discussed in [Section 4.2.4](#).

### 5.2.3 Transforming Leaf-Nodes

In this experiment we evaluate the modified expansion method for transforming leaf nodes, as described in [Section 4.2.4](#), with the goal of determining whether the Leaf-Node Transformation can alleviate the limitation of transforming distributions.

For the experiment, we use the same base models as in the previous experiment. However, we expand the PSPN using the Leaf-Transformation modification, which raises the number of parameters to 384 for the induced SPN of the final column. To balance the added parameters in the training procedure, we use a learning rate of 0.05 (refer to [Section 5.2.1](#)).

#### 5.2.3.1 Results

The per-task metrics comparison of the modified PSPN and the standard SPN model in [Table 5.3](#) were calculated on the moon dataset. They reveal a significant drop of at least 6% in accuracy across in the first column and

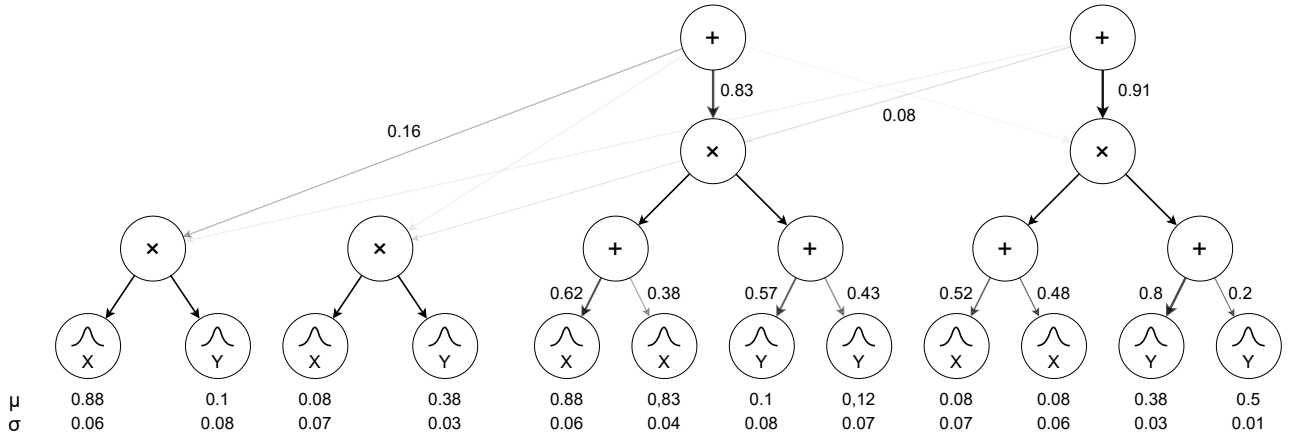


Figure 5.9: This figure shows the induced SPN of the modified second column trained on the case study data. The learned weights are represented by the opacity of the edges as well as given numerically for the significant edges. Note that all sum nodes, added to allow the Leaf-Node Transformation in Section 4.2.4, perform a mixture of both of its children instead of selecting a Leaf-Node, as intended.

8% in the second column, across all four test cases. We identify the cause by looking into another case study with the same Gaussian dataset as in the previous study seen in Figure 5.5.

Plotting the sum weights again in Figure 5.9 reveals that the sum nodes used to integrate the copied leaf node do not select a single leaf node. Instead, they form mixtures of both respective leaf nodes, which, with the help of the parent product nodes, form a cross-product with all four leaves, resulting in a mixture of four 2D Gaussian distributions.

### 5.2.3.2 Analysis & Evaluation

This experiment shows that the modified expansion method for transforming leaf nodes, as described in the method chapter, does not work due to structural errors, hindering us from determining whether the leaf-node transformation can alleviate the limitation of transforming distributions.

In the performance metrics we can see that the Leaf-Transformation modification did not work as intended. As discussed in Section 4.2.4, we have exponential growth in the number of leaf nodes using this expansion. However, while this could theoretically impair training, it is unlikely to be the cause at this scale. To instead identify the reason behind this failure, we looked into another case study, which again provided useful insights into the operational behavior of our model. In this case, we demonstrated that the sum node used to incorporate the second leaf node does not act as a binary selector, as assumed. Instead, the sum node forms a mixture of two univariate distributions, which, using the product node in the next layer, creates four interdependent Gaussian clusters. We, therefore, conclude that this unwanted structural modeling error is likely the cause of the reduced performance.

Nevertheless, it is worth noting that the mixture weights seem to indicate a preference towards the pre-trained leaf node. Motivated by this preference and the need to adjust the previously learned leaf parameters, while also avoiding structural errors, we revert back to the base PSPN model and copy the leaves instead of choosing between randomly initialized and pre-trained leaves.

---

## 5.2.4 Copying Leaf-Nodes

The last experiment on synthetic data is performed on the same setup as before to investigate another approach, which was motivated by the results of the first two experiments. In this approach, we copy the leaf parameters of the first column, as an initialization for the new column.

We once again use the same base model as in the preceding experiments but construct the PSPN by initializing the leaf parameters with those of the previous column during expansion. We also reduce the learning rate back to 0.03.

### 5.2.4.1 Results

Since the synthetic data case is not complex enough for significant differences in the final results with the per-task metrics and transfer scores, we instead provide visual data using the median loss curve and median accuracy learning curve.

First, we report our results in the Mean-Shift and Variance-Scaled test cases in [Figure 5.10](#). While both curves reach higher accuracies earlier than the baseline SPN, we see an additional increase when using the Leaf Search. Additionally, we can confirm the better initialization of the PSPN-LC by its lower starting loss and the fact that its loss curve consistently stays below that of the PSPN.

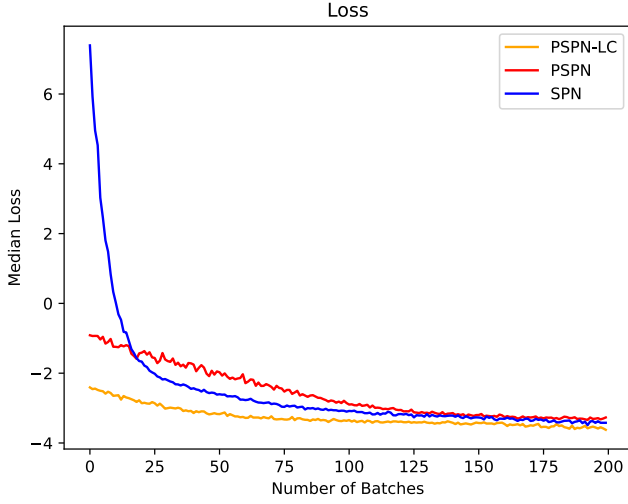
To similarly evaluate the performance for the Independent and Fixed-Cluster cases, we again plot the median accuracy curves in [Figure 5.11](#), where we see that both PSPN and PSPN-LC, perform similarly, with no visible differences.

### 5.2.4.2 Analysis & Evaluation

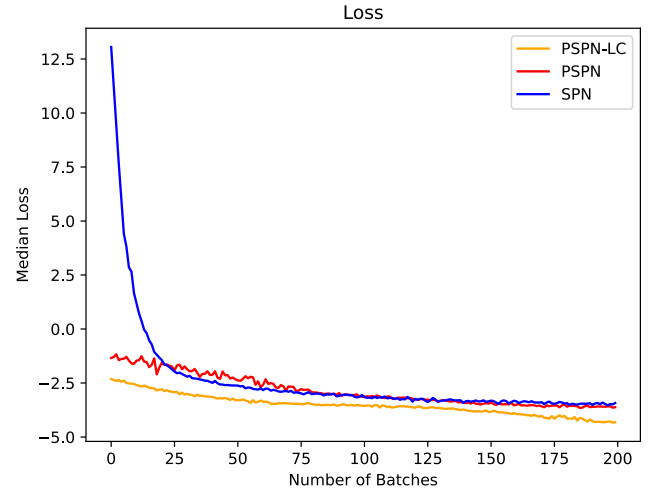
In this experiment, we show that the Leaf Copying technique provides the needed transforming capabilities under specific assumptions, by showing that leaf copying provides a better initialization in cases of related tasks while maintaining the performance of random initializations in the case of independent tasks.

For the evaluation of these results, we began with the Mean-Shift and Variance-Scaled test cases. Here we confirmed the hypothesized improvement when using the pre-trained initialization due to the lower initial loss and the improvement seen in the accuracy curve. Moreover, since the area under the accuracy curve showed an improvement, we can conclude another improvement in the transfer score. In the Fixed-Cluster case, there was no significant difference in the performance, despite the relation of the tasks. We think this might be due to the distribution of the fixed cluster in the base PSPN being reusable by the lateral connections, which mitigates the need for the copying of leaf nodes. However, further exploration of this phenomenon is outside the scope of this exploration. Lastly, since no significant difference in the performances of the independent test case was observed, this experiment confirms that the copied parameters offer no disadvantage compared to random initializations while also enabling the potential for parameter transformation in the leaves.

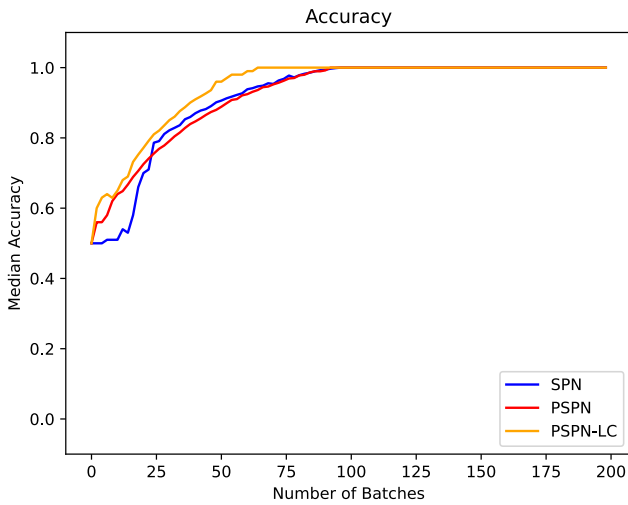
We recognize that this approach has the potential to introduce unwanted assumptions of task relations. However, for this simple 2D data case, we argue that this is not the case, assuming independent tasks. This argument is as follows, when initializing the leaf parameters of a column, we implicitly initialize the corresponding distributions as random Gaussians in the feature space, and since we use the same uniform distributions for data generation and parameter initialization, the random initialization and the data distributions are virtually indistinguishable. Now suppose the first column converges to the correct data distribution during



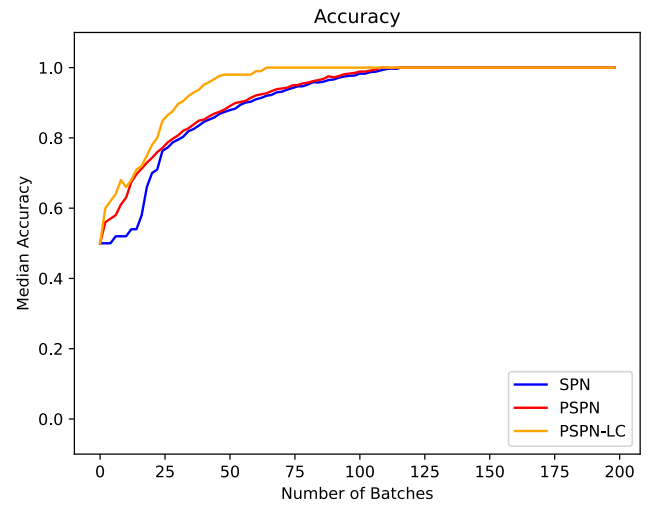
(a) This subfigure shows the median NLL loss after a given number of batches for the Mean-Shift test case.



(b) This subfigure shows the median NLL loss after a given number of batches for the Variance-Scaled test case.

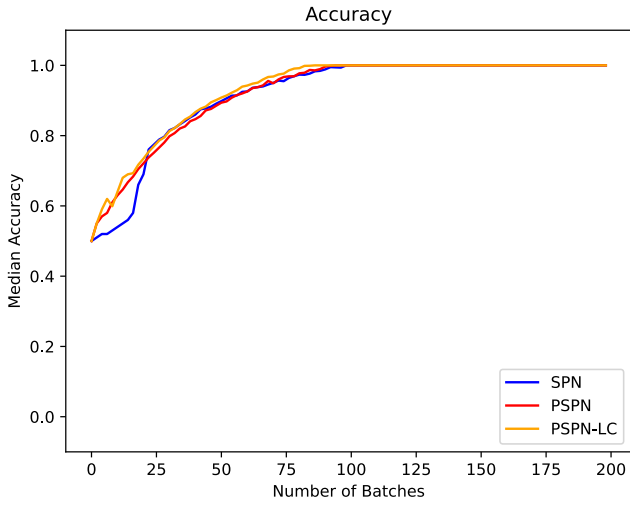


(c) This subfigure shows the median accuracy after a given number of batches for the Mean-Shift test case.

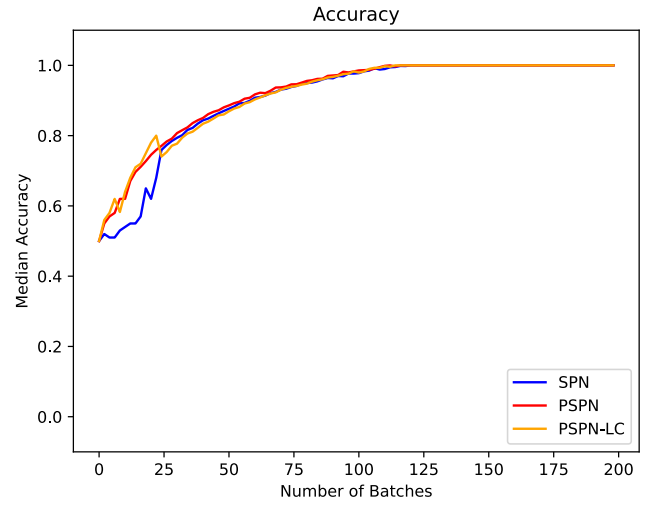


(d) This subfigure shows the median accuracy after a given number of batches for the Variance-Scaled test case.

Figure 5.10: This figure shows a comparison of the median loss and accuracy learning curves in Mean-Shift and Variance-Scaled test cases using moon data, for the baseline SPN, the base PSPN and PSPN with Leaf Copy. Note that the PSPN with Leaf Copy outperforms both base PSPN and SPN in terms of the lower loss as well as the area under the accuracy curve.



(a) This subfigure shows the median accuracy after a given number of batches for the Independent test case.



(b) This subfigure shows the median accuracy after a given number of batches for the Fixed-Cluster test case.

Figure 5.11: This figure shows a comparison of the accuracy learning curves in Independent and Fixed-Cluster test cases using moon data, for the baseline SPN, the base PSPN and PSPN with Leaf Copy. Note that the PSPN with Leaf Copy performs similar to the base PSPN.

training. In that case, the learned distributions resemble the data distribution and, by the same logic, cannot be differentiated from a random initialization. Therefore, when the second task is independent of the first, the initialization through copying is equivalent to a random initialization. If the tasks are related, the copying exactly enables the transformable forward transfer we seek. It is, however, important to note that this is only true for our limited synthetic test case and should not be adapted in general without considering the unknown assumptions.

### 5.2.5 Summary

The experiments conducted on synthetic data provide a valuable Proof-of-Concept for the theoretical definition of PSPNs. They demonstrate that the design can effectively prevent catastrophic forgetting, a significant challenge in continual learning scenarios. However, the experiments also reveal limitations in the use of lateral connections for forward transfer. These limitations can be traced back to the differences in modeling between SPNs and neural networks. In the simple synthetic cases where the data distribution can be exactly modeled with a model that requires only one sum layer in the classification layer, the model has enough capacity to learn the entire distributions before lateral connections can be used to perform forward transfer. In these cases, the density estimation finds the true distributions and any column can only reuse the probability distribution learned by the previous columns identically. As such, if trained long enough, the lateral connections only serve as binary selectors in these cases. To address this limitation, we propose the approach of copying the learned parameters during the expansion of the PSPN to enable an adaptation of the parameters in the new columns. Our experiments suggest that this approach can provide advantages in these simple cases.

---

Looking forward, we anticipate that the use of more complex data that cannot be modeled as easily, such as images, will allow the first few layers of the model to learn distributions of very low-level features. Furthermore, pixels in given images with similar content are expected to exhibit a degree of similarity since they often represent the same physical objects and therefore tend to cluster in the feature space [28]. Therefore, we expect these features might be reused more efficiently in new tasks, even if the different columns do not model the identical distributions at a higher level, which could enhance the effectiveness of lateral connections for forward transfer.

---

## 5.3 Exploration on Image Data

---

The exploration of real data in this thesis is conducted through three major experiments, each designed to test the PSPNs as well as a variation of our Leaf Copying approach on more advanced scenarios. The first experiment aims to evaluate the ability of the base PSPN to reuse previous knowledge effectively when dealing with complex data. We then revisit the idea of non-random initializations in the second experiment, proposing a more systematic approach to incorporate Leaf Copying into the PSPN framework, specifically through a parameter searching method. This method involves copying the initialization parameters from the best performing model. This experiment aims to find whether, this approach provides a more effective starting point for the training. In the final experiment, we examine the necessity of lateral connections when parameters are copied. We remove all lateral connections and apply the columns to each task, providing a comparison with the previous experiment.

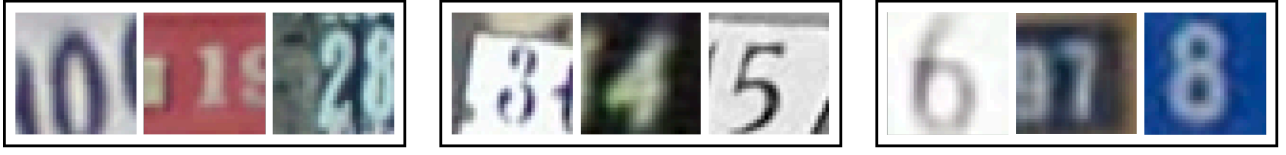
### 5.3.1 Setup

**Implementation** Thus far, the implementation has been limited to 2D data, as it only involves distributions over two random variables, which allowed us to estimate or even know the desired probability distribution and infer the needed structure. However, manually constructing the model becomes infeasible when dealing with more complex data, such as image data, where the desired distribution is too complex.

To efficiently implement PSPNs for image data, we instead use a more efficient method for constructing SPNs, specifically by using RAT-SPNs [43], which use random divisions of the RVs and combine them with layered structures based on predefined parameters to avoid the need for structure learning and learn the parameters in a deep learning manner. The efficiency of RAT-SPNs is further enhanced by EiNets [41], which combine different sum and product operations into a single einsum operation and structure the network into layers. This increases efficiency by allowing for parallel computations of these single einsum operations, enabling the creation of larger models. The implementation of PSPN for the following experiments builds upon the simple-einet project, a PyTorch implementation of EiNets by Braun [3]. This project was chosen over the official EiNets implementation due to its simplified and modular design, which facilitates the needed modifications, allowing for experimentation with the architecture and training setting.

The datasets used in these experiments, namely MNIST [23], SVHN [34], and CIFAR10 [22], were taken from the PyTorch-Vision package and manually modified to fit our setting.

**Evaluation** Given the larger model structures involved in these experiments, we cannot look into the models, as done in the case studies. Instead, we base our analysis mainly on the metrics and employ the perturbation analysis where necessary, which allows us to gain insights into the model.



(a) This subfigure shows the test case without task intersections. The tasks consists of classes  $\{0, 1, 2\}$ ,  $\{3, 4, 5\}$ , and  $\{6, 7, 8\}$  respectively.



(b) This subfigure shows the test case with task intersections. The intersection between the first and the second task is depicted by the red dotted line, while the blue dotted line depicts the intersection between the second and third task. The tasks consists of classes  $\{0, 1, 2\}$ ,  $\{1, 2, 3\}$ , and  $\{2, 3, 4\}$  respectively.

Figure 5.12: This figure shows the two test cases for the SVHN [34] datasets. The tasks are depicted as black boxes and the three classes in each task are given by a sample data instance.

**Dataset** In order to, allow for the comparability and reproducibility of our experiments, we also using common datasets, which we need to modify to generate appropriate datasets for the setting. By ensuring that we use datasets with complete data and labels, we allow their use in our supervised classification. We then split each dataset into all individual classes and recombine them regularly into the specified tasks to obtain the needed regular tasks. Hereby, we create the task label, as well as the within-task labels. This enables the use of the dataset for task-incremental learning. Furthermore, the recombination of the classes also allows us to intersect the tasks as much as desired to allow for a greater variety in the construction of our experiments. For this experiment, we designed two test cases consisting of three tasks with three classes each. The first test case was generated using no intersection (NI) between the tasks, while the second one has a class intersection size of two between each task (2TI).

Since the classification of the MNIST [23] dataset proved not to be complex enough to reveal significant differences in performance, we chose to present the results from the Street View House Numbers (SVHN) [34] dataset for the following experiments and evaluate all three experiments using this dataset in order to allow for comparisons between them. Figure 5.12 shows both test cases using a data instance for each of the three classes for each task.

**Model** Because our implementation is based on the structure of RAT-SPNs, we provide the construction parameters as introduced in Peharz, Vergari, et al. [43] as a reference for the model used. For all following experiments, we used the following parameters  $S = 15$ ,  $I = 15$ ,  $R = 3$  and  $D = 5$ , which result in a base model with 296,879 parameters. For the final induced SPN this results in 951,792 parameters.

This model structure is used as the initialization for the first column. The PSPN is then generated by applying the base expansion depicted in Figure 4.2 for each task. Furthermore, when used as a single-column SPN, this model also serves as a baseline for our evaluation.

| Mean Accuracy  |     |   | Task Evaluated |      |      |              |      |      |
|----------------|-----|---|----------------|------|------|--------------|------|------|
|                |     |   | PSPN           |      |      | SPN Baseline |      |      |
|                |     |   | 0              | 1    | 2    | 0            | 1    | 2    |
| Task Evaluated | NI  | 0 | 0.84           | 0.32 | -    | 0.84         | -    | -    |
|                |     | 1 | 0.84           | 0.73 | 0.34 | -            | 0.85 | -    |
|                |     | 2 | 0.84           | 0.73 | 0.66 | -            | -    | 0.85 |
|                | 2TI | 0 | 0.85           | 0.35 | -    | 0.85         | -    | -    |
|                |     | 1 | 0.85           | 0.85 | 0.32 | -            | 0.85 | -    |
|                |     | 2 | 0.85           | 0.85 | 0.84 | -            | -    | 0.85 |

Table 5.4: This table shows the mean per-task accuracy matrices of the SPN baseline and PSPN on the two test cases. Note that the PSPN has similar diagonal entries to the baseline in the test case with intersection, indicating similar performance. Further, note that the PSPN shows significant performance drops in the test case without task intersection.

**Training** The training process involved running five repetitions of the model using 100 epochs with a batch size of 256 and a learning rate of 0.001. For the optimization, we used the Adam optimizer and the Negative Log-Likelihood Loss function, to optimize the posterior. We use the posterior instead of the likelihood due to stability reasons during training. Furthermore, because of the increased training duration, the randomly initialized baseline SPNs were trained on all tasks, which allows us to compare the performance of the PSPN across different tasks, not only the latest task.

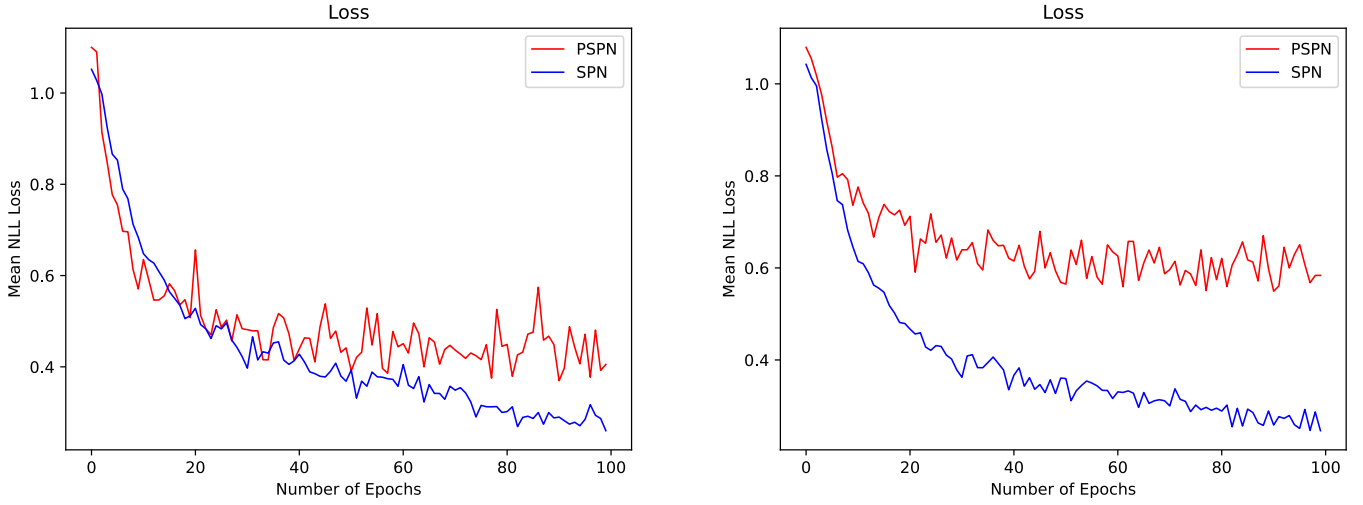
### 5.3.2 Evaluation on Image Data

This experiment aims to test our hypothesis that the probability distributions of the first layers can be reused effectively when dealing with more complex data, thereby leading to improved performance.

#### 5.3.2.1 Results

To evaluate the general performance of the model, the experiment results are reported using the per-task metric, shown in Table 5.4. We can again confirm that catastrophic forgetting is avoided in the PSPN model, as seen by the consistent performance across the task progression. We also reassure that the first column and the SPN baseline show an equal performance in both test cases, since all these cases are equivalent in the first column. In the performance of the test case with class intersections (2TI), we see a similar performance, across all three tasks, with only a negligible drop in the final accuracy. However, in the test case with no intersection between tasks (NI), we can observe a significant decrease of in performance in the PSPN, compared to the baseline SPN. This decrease of over 10% in the final accuracy in task 1, further drops by nearly another 10% in task 2.

Plotting the losses in Figure 5.13 reveals that the PSPN columns converge to higher losses. We analyze this result by performing a perturbation analysis by adding Gaussian noise to the input of the leaf layers of each column individually. In both test cases, this analysis needed significantly lower standard deviation in the noise, when applied to the first column and high standard deviations when applied to the other two.



(a) This subfigure shows the mean loss curves for the second column of the PSPN and an SPN baseline trained on the same task. Note that the PSPN converges to a higher loss.

(b) This subfigure shows the mean loss curves for the third column of the PSPN and an SPN baseline trained on the same task. Note that the PSPN converges to a much higher loss.

Figure 5.13: This figure shows a comparison of the mean loss curves for the baseline SPN and the base PSPN in the non intersecting test case. Note that the PSPN converges to a higher loss in the third column compared to the second column.

We further note that examining the gradients during the first training steps of the new columns reveals much bigger gradients for the lateral connections to the first column than the horizontal connections in both test cases.

### 5.3.2.2 Analysis & Evaluation

Contrary to our previous hypothesis, the results of this experiment indicate that the base PSPN struggles to learn optimally in tasks with higher complexity due to high gradients in the lateral connections. These gradients stem from initialization problems, emphasizing the need for further modifications to enhance the model performance.

Since higher gradients and a high use of lateral connections can be expected in the test case with class intersection, we would like to focus on the test case without intersection for the evaluation. We hypothesize that the leaf distributions of the pre-trained column have been learned to detect general features of each pixel as hypothesized in [Section 5.2.5](#). However, the new leaf nodes cannot detect these as the according distributions are randomly initialized. This difference leads to higher activations in the previous leaf layers, causing high gradients during the backward pass. We further suggest that these lateral connections are therefore enforced during training, even though these distributions may not apply as well to the new task. Moreover, since these distributions cannot be adapted due to the frozen parameters, the higher gradients enforce the reuse of the non-optimal distributions learned in previous columns, thus causing the higher loss since they are stuck in local optima.

Attempting to avoid these initialization problems, we propose returning to the strategy of copying parameters for optimized initializations in the next experiment.

---

### 5.3.3 Parameter Search on PSPNs

With this experiment, we explore different types of initializations aiming to alleviate the gradient problems identified in the previous experiment.

To provide a more formal approach compared to previous leaf copying, where we arbitrarily copied the parameters of the previous column, we introduce the following method. Generally speaking, we initialize the new column randomly and copy the parameters of the previous columns one by one. For each copy, we then perform a forward pass on a fixed-size mini-batch of the new task and calculate the loss on this batch. We lastly use the parameters with the smallest loss as the final initialization. We call this stochastic approach **parameter search**, as it aims to provide an estimate of how good the initialization is for the given new task and find the best one.

Depending on the specific parameters copied before searching, we define the following specific procedures:

1. **No Search (NS)**: We do not search any columns. This is equivalent to the base PSPN model.
2. **Leaf Search (LS)**: We copy only the leaf distributions and randomly initialize all sum weights.
3. **Isolated Column Search (ICS)**: We copy all leaf distributions as well as horizontal connections. The lateral connections are randomly initialized.
4. **Column Search (CS)**: We copy all leaf distributions as well as horizontal and lateral connections. The missing lateral connections (since previous columns do not have as many) are randomly initialized.
5. **Trained Column Search (TCS/20TCS)**: We perform the same copying as in Column Search but train each column before evaluating. (In our experiments, we specifically train each column for 20% of the normal training duration, i.e. 20 epochs)
6. **Forced Column Copy (FCC)**: We do not perform any search and instead always copy the first column (leaf distributions and sum weights).

We acknowledge that copying a previous column assumes correlated tasks and try to avoid this assumption by adding a randomly initialized column to each search. We refer to these procedures as **search types**.

Using the same base model as in [Section 5.3.1](#), we construct the PSPN by initializing the parameters before training through parameter search during the expansion. The specific parameters and the column from which the parameters are copied depend on the search type. We therefore run each search type as an individual test case and reuse the base PSPN performance from the previous experiment as well as the randomly initialized SPN as baselines by using the same setup as previously, to allow these comparisons.

#### 5.3.3.1 Results

To get a first overview over the performance and make an initial comparison, we provide the final reached accuracies for all search types.

As shown in [Table 5.5](#), the final accuracy in the tasks with no intersection show a similar decline in performance as with the base PSPNs (NS) for both Column Search and Isolated Column Search. However, when incorporating the Leaf Search, Trained Column Search or the Forced Column Copy we observe similar final accuracies to the baseline SPN, despite having no intersection between the tasks. When evaluating the performance in the test case with class intersection, we observe that the base PSPN with No Search, the Isolated Column Search, and the Column Search all perform similar to the baseline. In the case of Trained Column Search, and Forced

| Model        | Task Evaluated | Mean Accuracy (in %) |      |
|--------------|----------------|----------------------|------|
|              |                | NI                   | 2TI  |
| PSPN-NS      | 0              | 0.84                 | 0.85 |
|              | 1              | 0.73                 | 0.85 |
|              | 2              | 0.66                 | 0.84 |
| PSPN-LS      | 0              | 0.83                 | 0.85 |
|              | 1              | 0.83                 | 0.86 |
|              | 2              | 0.82                 | 0.88 |
| PSPN-ICS     | 0              | 0.84                 | 0.84 |
|              | 1              | 0.74                 | 0.86 |
|              | 2              | 0.67                 | 0.83 |
| PSPN-CS      | 0              | 0.84                 | 0.85 |
|              | 1              | 0.74                 | 0.85 |
|              | 2              | 0.67                 | 0.85 |
| PSPN-TCS     | 0              | 0.84                 | 0.84 |
|              | 1              | 0.84                 | 0.85 |
|              | 2              | 0.82                 | 0.85 |
| PSPN-FCC     | 0              | 0.83                 | 0.85 |
|              | 1              | 0.85                 | 0.86 |
|              | 2              | 0.85                 | 0.86 |
| SPN Baseline | 0              | 0.84                 | 0.85 |
|              | 1              | 0.85                 | 0.85 |
|              | 2              | 0.85                 | 0.85 |

Table 5.5: This table shows the final reached accuracy of all columns for the different search types in both test cases. Note that in the case of the non intersecting tasks, the base PSPN (PSPN-NS), the Column Search, and Isolated Column Search all exhibit the same accuracy drops on the second and third tasks, while Leaf Search, Trained Column Search, and Forced Column Copy all perform similar to the SPN Baseline. Further, note that in the intersecting task case, all searches perform similarly, with the exception of the Leaf Search, which for task 2 shows an increase of 3% in accuracy.

| Search Type |     | LS    |       | ICS   |       | CS    |       | TCS   |       |
|-------------|-----|-------|-------|-------|-------|-------|-------|-------|-------|
| Expansion   |     | 1     | 2     | 1     | 2     | 1     | 2     | 1     | 2     |
| Copied      | IND | 0 (4) | 0 (3) | R (5) | R (5) | R (5) | R (4) | 0 (5) | 0 (4) |
| Column      | 2TI | 0 (5) | 1 (5) | R (5) | R (5) | R (5) | R (3) | 0 (5) | 1 (5) |

Table 5.6: This table shows the index of the most copied column (where R denotes the random column) and number of times it was copied across the 5 repetitions in parentheses, for selected search types (NS and FCC were omitted since always copy the same column). The figure is given for both expansions in each test case. Note that the search types that performed poorly in Table 5.5 used the random initialization most of the time, while the other two reused previous columns.

| Transfer     |     | Search Type |       |      |      |
|--------------|-----|-------------|-------|------|------|
| Score (in %) |     | NS          | LS    | TCS  | FCC  |
| Task         | NI  | -19.70      | -1.77 | 0.75 | 0.79 |
| Evaluated    | 2TI | 0.78        | 7.17  | 5.93 | 4.18 |

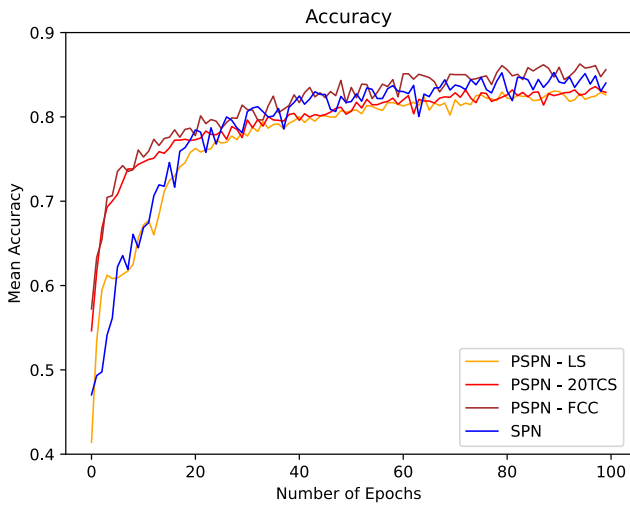
Table 5.7: This table shows the mean transfer score of the third column for select search types in both test cases. Note that the base PSPN (NS) exhibits very poor performance in the non intersecting tasks and requires the Trained Column Search or the Forced Column Copy to mitigate the performance decrease fully. Further, note that Leaf Search shows the highest transfer in the intersecting test cases.

Column Copy, we also see similar classification accuracy. Moreover, when using Leaf Search we can even see a slight improvement of 3% in the total accuracy reached.

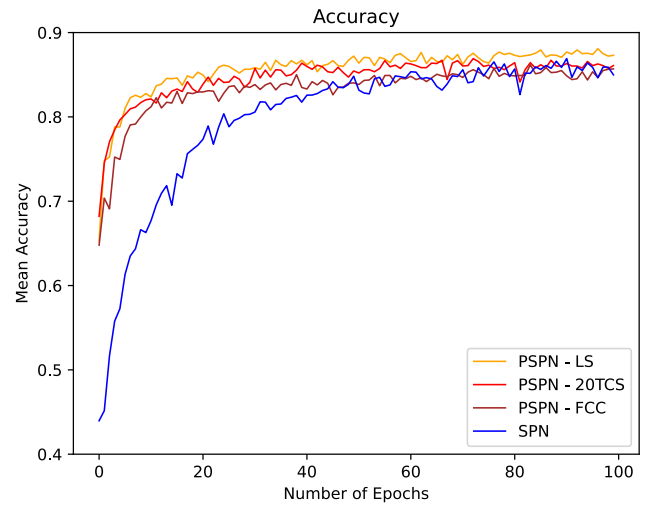
To find out the reason behind the performance decline in the two column searches, we consider what column was copied most often in each search. In Table 5.6 we can see that Leaf, Trained Column Search, and Forced Copy all reused the previous parameters, while the Isolated Column Search, as well as the Column Search, preferred the randomly initialized column. Since the two column searches always use the random columns, they are essentially equivalent to performing no search. Going forward, we therefore omit their result presentation.

For the remaining four search types, we present their transfer scores for the third column. In these transfer scores provided in Table 5.7, we again see that not conducting any search yields very poor performance. While Leaf Search drastically remedies this problem, only the column copying in the Trained Column Search and the Forced Column Copy can fully mitigate the performance decrease. Only when the tasks share classes do we observe a significant performance improvement by using parameter search. In such cases, Leaf Search emerges as the most effective approach, yielding the highest performance.

In order to additionally provide a visual aid combining both the final accuracy and the transfer score, we can plot the average accuracy for each training epoch for the two test cases in the second columns. These results, shown in Figure 5.14, suggest that in non-intersecting tasks, Leaf Search performs on par with the SPN baseline while demonstrating a considerable performance increase in intersecting tasks. On the other hand, the column copying methods demonstrate performance improvements in both cases, as seen by the faster increase in accuracies. However, it is important to note that the performance increase in the case of the intersecting task is slightly lower compared to Leaf Search.



(a) This subfigure shows the mean accuracy curves for the Leaf Search, Trained Column Search, Forced Column Copy and SPN baseline trained on the non intersecting tasks. Note that both column search types perform better than the SPN baseline and the Leaf Search performs similarly.



(b) This subfigure shows the mean accuracy curves for the Leaf Search, Trained Column Search, Forced Column Copy and SPN baseline trained on the intersecting tasks. Note that all search types show higher areas under the curve compared to the SPN baseline. Further, note that the Leaf Search reaches a slightly higher accuracy.

Figure 5.14: This figure shows a comparison of the mean accuracy curves for the third column of selected search types and the baseline SPN in both test cases. Note that all search types perform better in the intersecting tasks case.

---

### 5.3.3.2 Analysis & Evaluation

This experiment, empirically shows that copying previous parameters grants a better initialization. While the results show that when tasks are correlated, this improved initialization can provide slight performance boosts, the main advantage we validate is the effectiveness of this approach in resolving the previous issues.

Since the tasks are correlated even in the NI test case, the copying provides a better initialization than the random initialization, as seen using the forced copy. However, we still note that both Isolated Column Search and Column Search methods preferred the random column. We hypothesize that this preference of the random column results from a higher initial loss when reusing the entire column. Because, when doing so, we also copy the sum weights, which force the use of the exact mixtures and therefore also the same final learned distribution in the classification head. With Leaf Search, where we copy only the leaf distributions, we get the benefit of initializing with the distributions over the general pixel features but still make no assumption of how the pixel distributions must be combined to describe any concrete data, which avoids the high initial loss of the column copy. This is circumvented with the Trained Column Search since we can capture the quick initial improvement in the loss, such that the Trained Column Search does in fact prefer previous columns. Based on these results, we suspect that the improvement in classification speed results mainly from the improved initialization.

Nonetheless, using the Leaf Search, we could see an increase in the final reached accuracy compared to the baseline SPN, which suggests a slight general improvement in the model performance. We attribute this to the increased capacity using the lateral connections, and suspect the reuse of some distribution from previous columns frees up capacity in the second column, which can be used to increase the accuracy. However, the increase is too small to conclude a general improvement confidently. To test this hypothesis and answer whether we still need lateral connections if we copy the parameters, we perform the following final experiment.

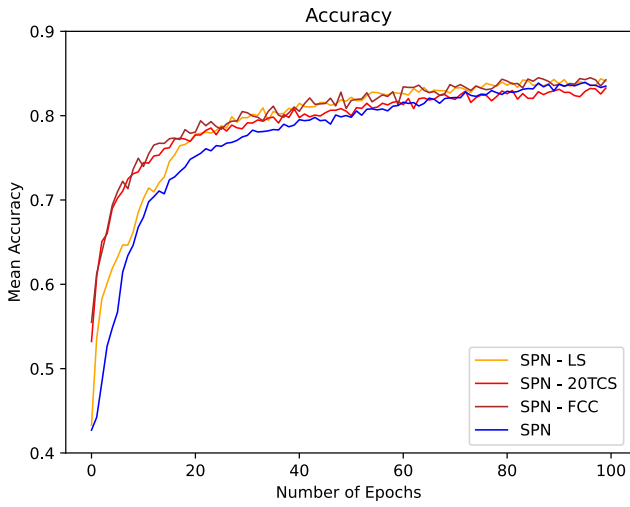
### 5.3.4 Parameter Search on SPNs

For the final evaluation, we aim to find whether the improvement in transfer score results mainly from parameter searching and whether we need lateral connections when the parameters are copied. To perform the experiment, the model used is the same as in [Section 5.3.1](#), but we omit the lateral connections.

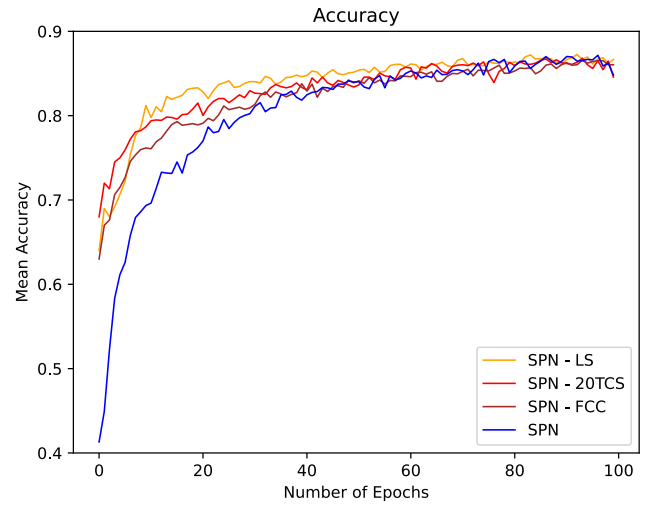
#### 5.3.4.1 Results

To provide an initial overview of the performance of the different search types over time, we present the average accuracy curve for each of the two test cases across epochs, illustrated in [Figure 5.15](#). Here, we observe similar curves to the PSPN results. In the non intersecting test case, Leaf Search closely follows the baseline, although it consistently remains slightly above it. In the intersecting test case, it provides a high initial increase in accuracy. On the other hand, the column copying methods display comparable curves in both test cases, demonstrating a faster rate of accuracy improvement.

To get a measure of the relative knowledge transfer, we again calculate the transfer scores and compare them with the PSPN counterparts. In [Table 5.8](#), we see that when the tasks do not intersect, the SPN outperforms the PSPN in terms of knowledge transfer, regardless of the search type employed. In contrast, the transfer scores in [Table 5.9](#) show that when the tasks intersect, all search types show higher transfer when using the PSPN.



(a) This subfigure shows the mean accuracy curves for the Leaf Search, Trained Column Search, Forced Column Copy applied on individual SPNs and the No Search SPN baseline trained on non intersecting tasks. Note that all search types perform show higher areas under the curve than the SPN Baseline.



(b) This subfigure shows the mean accuracy curves for the Leaf Search, Trained Column Search, Forced Column Copy applied on individual SPNs and the No Search SPN baseline trained on intersecting tasks. Note that all search types perform show higher areas under the curve than the SPN Baseline. Further, note that the Leaf Search does not show an increased final accuracy this time.

Figure 5.15: This figure shows a comparison of the mean accuracy curves for the third column of selected search types applied on individual SPNs and the No Search SPN Baseline in both test cases. Note that all search types perform better in the intersecting tasks case.

| Transfer Score (in %) | Search Type |       |      |      |
|-----------------------|-------------|-------|------|------|
|                       | NS          | LS    | TCS  | FCC  |
| PSPN                  | -19.70      | -1.77 | 0.75 | 0.79 |
| SPN                   | 0.00        | 2.27  | 2.67 | 2.74 |

Table 5.8: This table shows the mean transfer score of the third column for select search types applied on both PSPNs and SPNs in the non intersecting test case. Note that the SPN outperform the PSPN with every search type.

| Transfer Score (in %) | Search Type |      |      |      |
|-----------------------|-------------|------|------|------|
|                       | NS          | LS   | TCS  | FCC  |
| PSPN                  | 0.78        | 7.17 | 5.93 | 4.18 |
| SPN                   | 0.00        | 3.83 | 3.24 | 1.95 |

Table 5.9: This table shows the mean transfer score of the third column for select search types applied on both PSPNs and SPNs in the intersecting test case. Note that this time the PSPN outperforms the SPN with every search type.

---

#### 5.3.4.2 Analysis & Evaluation

Our last experiment shows that, while minute, the lateral connections do indeed provide a slight performance improvement in the case of intersecting tasks.

While, we can conclude that while parameter search does, in fact, contribute to knowledge transfer on its own, the results also demonstrate that the lateral connections do provide benefits to the knowledge transfer when the tasks intersect. However, when the tasks do not intersect, these lateral connections provide disadvantages in a similar order, despite the high correlation between the tasks provided by using the same dataset for each task.

As a side note, we would like to mention that the lateral connections appear to slightly push the maximally achievable final accuracy. This is seen as the independent columns do not exhibit the same indications of increased performance as observed with Leaf Search in the previous experiment. We do, however, acknowledge that these minute differences could also have different causes and should only be regarded as indications of increased performance and by no means as concrete evidence. Further exploration of this phenomenon is left for future research.

#### 5.3.5 Summary

In our experiments on image data, we have demonstrated empirically that the general pixel features learned in the leaf layers can indeed be reused in subsequent tasks. However, in contrast to our initial hypothesis, this reuse does not provide the desired increase in performance but rather serves as a hindrance through high gradients in the lateral connections. To enable the PSPN to efficiently use lateral connections and overcome the training problems, we employed a parameter searching method to copy previously learned distributions in the leaf nodes. This approach helps alleviate the problems when tasks do not overlap and provide benefits when they do.

However, our experiments also revealed that using parameter searching alone, without the PSPN architecture, also leads to performance improvements. Although these are not as significant in cases where tasks do not intersect, we do not have the associated disadvantages. These findings highlight the complexity of leveraging lateral connections for knowledge transfer and suggest that further research is necessary to fully understand and optimize them.

In terms of runtime and memory requirements, Leaf Search is the most efficient type of search, as batches can be evaluated on all column leaves in parallel and then concatenated in the batch dimension to be evaluated in a single forward pass. The Trained Column Search is the least efficient since each search requires the training of all columns for multiple epochs, which are not going to be used in all but one case. In theory, Trained Column Search is the best search type since, since with a higher number of epochs, we approach the true training. However, these computational requirements are way too high to even be considered in practical applications.

Therefore, the most promising outcome of our exploration was, in our assessment, the Progressive Sum-Product Network with Leaf Search (PSPN-LS). This model demonstrated comparable performance to traditional SPNs in tasks with no class intersection and better performance in tasks with class intersection, with the least computational overhead, balancing performance and efficiency.

---

## 5.4 Overall Analysis

---

To provide an all-encompassing view of the PSPN, we conduct a comprehensive evaluation of the PSPN-LS model by summarizing our experimental observations and addressing remaining questions regarding the model by analyzing the theoretical concepts outlined in [Chapter 4](#).

**Significance** Our experiments have incorporated results from multiple runs, ensuring the empirical significance of our findings, and address the scalability by evaluating the model on multiple tasks. We have conducted our experiments using publicly available datasets to enhance reproducibility and delivered detailed descriptions of the data and their division into tasks used were also provided, aiming to further improve reproducibility and enhance comparability with other models. Furthermore, we evaluated these experiments using standard metrics like transfer score, and per-task metrics.

**Summary of Experiments** Based on the fewer steps required to meet higher classification thresholds and to reach convergence, we conclude that the model does offer a slight advantage over traditional SPNs. The computed transfer score suggests that the model is able to utilize lateral connections to facilitate the transfer of previous knowledge, to some extent as long as the model is properly initialized. Furthermore, our design inherently avoids catastrophic forgetting, primarily through the design of the model, which circumvents the need to retrain parameters, which can therefore be frozen. It also prevents backward transfer to circumvent interference with previously learned tasks. Hence, we did not compute the backward transfer.

**Further Context** To further contextualize our thesis and address remaining questions regarding the model, we conduct an analysis based on other common measures used to evaluate continual learning systems. Inspired by the questions posed in Lesort, Lomonaco, et al. [25], we described the data stream, algorithm, assumptions, and requirements regarding supervision, memory, and computation. We also assessed the extent to which our research fulfilled the recommendations provided in that work. As illustrated in the CLEVA-Compass in [figure Figure 5.16](#), we decided to limit our study to basic configurations. This intentional decision was made to avoid adding additional complexity, to identify performance differences more easily, and to be able to associate these to a single modification, thus facilitating our evaluation process.

**Supervision Requirements and Limitations of Setting** Consequently, our selected setting required a high degree of supervision, including the availability of the task label during both training and evaluation, needed for the supervised model selection among multiple models, which disallows task-agnostic learning. Furthermore, we decided to give the model complete data with complete labels, thereby inherently limiting the open-world capabilities. Despite the model not requiring iterative revisiting of old data, the necessity for multiple episodes prohibits the deployment in online settings.

**Data Requirements and Assumptions of Prior Knowledge** While our exploration used Gaussian leaves, the general implementation can employ any distribution family. Thus the model does not presuppose a specific distribution. It does, however, assume that data within tasks follow the same distributions and does not detect distribution shifts, relying instead on the knowledge of the task label. We are, therefore, not agnostic to the data stream structure. While this exploration was based on correlated tasks, the model itself

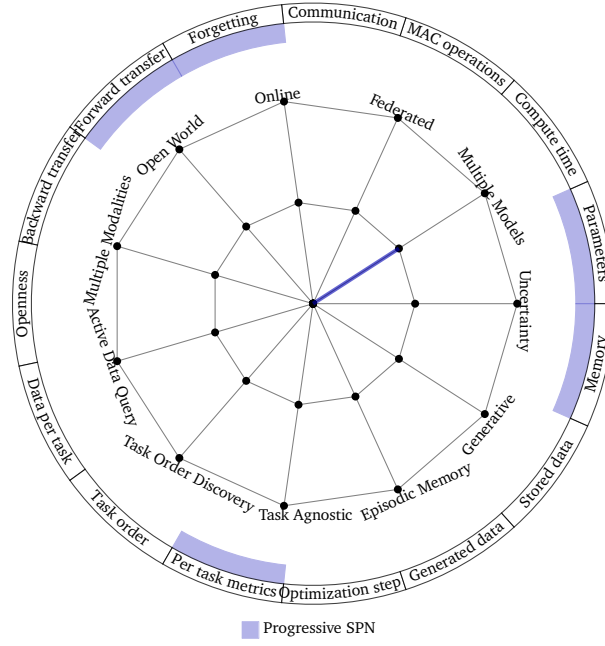


Figure 5.16: This figure shows the CLEVA-Compass for the developed Progressive SPN.

does not presume any task correlation when including randomly initialized columns in the search. However, through the initialization of the columns using the same structure, an implicit assumption of similar capacity requirements for each task is made. Nevertheless, the model can cater to any new task given similar capacity needs, including independent and even adverse tasks, as long as the implementation is designed to handle the corresponding data type. Our implementation is single-modal but can be adapted since the concept of PSPNs does not assume any specific data types. Likewise, we do not presume any number of tasks as long as the model fits into memory. However, this also imposes a practical limit on the number of tasks.

**Memory and Computational Requirements** The PSPN does not require data storage, as the need for revisiting previous data is avoided by freezing the parameters. However, it requires high storage for the model itself. The number of columns  $c(t)$  exhibits linear growth  $c(t) = \mathcal{O}(t)$  in the number of tasks  $t$ . The parametric growth  $p(t)$  is even quadratic  $p(t) = \mathcal{O}(t^2)$  in the number of parameters due to the addition of lateral connections. Each new task also increases both computational runtime and the number of operations due to the increase in parameter count and searches performed as the number of columns increases.

---

## 6 Conclusion

---

In this research, we investigated the potential of Sum-Product Networks (SPNs) for continual learning tasks, focusing on the challenges of enabling forward transfer and avoiding catastrophic forgetting.

Our findings suggest that, although the network’s architecture and training prevent catastrophic forgetting and enable forward transfer, the effectiveness of forward transfer is significantly limited compared to progressive neural networks. This limitation stems from fundamental differences between neural networks and SPNs. While neural networks directly transform features with each layer by combining various features to construct new ones, SPNs operate on the probability distributions of features. They combine these distributions to construct distributions over all possible features, modeling the distribution of entire data samples. Since distributions cannot be transformed as needed by weighted sums used in SPNs, any deviations from the distribution cannot be corrected through the lateral connections, severely restricting their effectiveness. Consequently, SPNs can only leverage the forward transfer of probabilities when the distribution of a specific feature is identical across different tasks. For tasks with a simple feature space, where different classes are scattered across the feature space, the knowledge provided by the lateral connections proves to be unusable. Only for tasks with complex feature spaces, where data lies in similar subspaces of the feature space, PSPNs can reuse the distributions and prove beneficial.

Moreover, lateral connections can inadvertently have adverse effects. When a new column is randomly initialized, lateral connections provide access to pre-trained columns, which can correlate to the data, causing high gradients during the backward pass. This hinders the effective training of the new column while simultaneously not providing the correct distributions. Therefore, unless the true distributions overlap between tasks, lateral connections can be more of a hindrance.

To mitigate these drawbacks of ineffective training, we propose initializing the new column similar to previous columns by reusing leaf distributions or entire previous columns. This provides the needed baseline for similar high gradients so as not to hinder the training. When it comes to tasks with high correlations, the reuse of previous columns via lateral connections can allow the model to free additional capacity to improve performance. This is further facilitated by the initialization methods that can act as a baseline for the model to build upon. Together, it provides a slight edge in continual learning scenarios involving highly correlated tasks and can compete with regular SPNs in uncorrelated tasks when combined with Leaf Search or Trained Column Search. However, it should be noted that Trained Column Search could prove inefficient in the long run because of the lacking scalability.

All in all, considering 1. the limitations we have set for the setting, 2. the overall results of only minute advantages in highly limited cases, 3. the increased model complexity stemming from the additional lateral connections and needed search, as well as 4. the poor scalability resulting from the linear growth in columns, the use of PSPNs cannot be universally recommended. Because of this, we conclude limited general progressive growing capabilities of SPNs.

---

## 7 Limitations & Future Work

---

In the interest of transparency and to motivate further research, we would like to address the shortcomings and limitations of our approach in this final chapter, beginning with the limitations posed by our setting and continuing with the limits of the model.

While the definition of the setting as the most basic approach has advantages in the exploration, the limitations of the setting are too strict to allow for real-life applications. It does not allow for the modeling of all challenges presented in Definition 3.2.0.4. As defined, the current task-incremental setting, together with the accordingly generated datasets, already provide the task labels. The model, therefore, does not use any shift-detecting measures to detect distributional shifts but relies on the task label during interference, which might not be given in real-world applications. Instead, our focus in developing the PSPN is to avoid catastrophic forgetting and to learn new tasks more effectively by transferring knowledge.

A future focus should be to conduct a more comprehensive evaluation by applying the model to more representative settings, ensuring its real-world applicability. Furthermore, as pointed out in Mundt, Hong, et al. [33], many existing models are often evaluated in artificially constructed settings, which may not reflect real-world data and scenarios. Evaluating the model in a broader range of settings, including tasks involving incomplete data, would provide a more comprehensive understanding of its performance.

As far as limitations of the model go, there are a few restrictions that result from the progressive modeling approach used. One of the primary limitations stems from the strict goal of preventing catastrophic forgetting. In our pursuit to avoid this, we must also avoid backward transfer, as any perturbations could potentially interfere with previously learned features. This leads to frozen progress on all previous tasks, as any newly gathered knowledge cannot be used to improve these tasks. Furthermore, the progressive expansion of the model we employ results in a linear increase in network size with the addition of each new task, which presents scalability challenges. This is again required to avoid forgetting since we need to add capacity when freezing all previous parameters. Additionally, copying the initial network structure sets a predefined limit on network capacity for all future tasks. This requires knowledge of the capacity requirements of all future tasks beforehand, which may not always be feasible. Another limitation arises from the structure of the lateral connections in the network. These connections extend from a given layer of the considered column to the previous layer of the previous column. Therefore, the number of columns reached by lateral connections in the induced SPN is limited. More specifically, the number of tasks that can even be used for forward transfer is constrained by the number of sum layers in the structure template, as only sum layers have lateral connections. Given that these limitations are inherent to the progressive modeling technique adapted from the progressive neural network, we again refer to future research to address these constraints.

For example, the scalability can be addressed by further developing the model as follows. Considering that lateral connections have shown potential for reusing previously learned distributions, it could be possible to take advantage of these distributions without needing an entire new column. This could be achieved by selectively pruning the column. While this might result in a mismatch in the modeled distribution, it could lead to better scalability and allow for a wider range of solvable tasks. Additionally, following the concepts

---

of Dynamically Expandable Networks [24], which also build upon progressive neural networks, could be a promising direction. The idea is to add capacity only as needed, reuse structures that model features shared across different tasks, and selectively retrain parts of the model.

However, the modeling limitations do not arise from the progressive modeling approach only. The use of SPNs introduces an inherent limitation when it comes to PSPNs as well. Unlike neural networks that operate on the feature space, SPNs only define distributions over the features. As a result, once the leaf distributions are set, the subsequent layers can only combine these distributions. The lateral connections, therefore, cannot modify any detected features. This limited transformability of previously learned distributions, was the main limitation of our approach. To further analyze this phenomenon and find the reasons for the indication of higher model accuracy observed when using Leaf Search, employing more localized perturbation methods, such as pixel-level perturbations, would be beneficial. Future research should strive to provide clearer insights into what aspects are being reused, which could then motivate new modifications to mitigate the problem.

Looking forward, there are several avenues for further exploration of potential improvements to our model. To this end, future research could consider introducing non-linear transformations before mixtures to allow for the necessary transformations. However, formalizing a robust probabilistic model to ensure tractability and efficient density estimation would be crucial in this case. Another suggestion is to place a neural network before the leaf layer for each new task. This neural network could perform basic feature transformations to enable the shifting of the estimated distributions across the feature space to follow the distribution shift and facilitate a more effective reuse of learned distributions. The PSPN would then be trained as a probability density estimator on these transformed features.

---

## Bibliography

---

- [1] Shams Forruque Ahmed, Md Sakib Bin Alam, Maruf Hassan, Mahtabin Rodela Rozbu, Taoseef Ishtiak, Nazifa Rafa, M Mofijur, ABM Shawkat Ali, and Amir H Gandomi. “Deep learning modelling techniques: current progress, applications, advantages, and challenges”. In: *Artificial Intelligence Review* (2023).
- [2] Hongjoon Ahn, Sungmin Cha, Donggyu Lee, and Taesup Moon. “Uncertainty-based continual learning with adaptive regularization”. In: *Advances in Neural Information Processing Systems* (2019).
- [3] Steven Braun. *Simple-einet: An EinsumNetworks Implementation*. URL: <https://github.com/braun-steven/simple-einet>.
- [4] Danton S Char, Nigam H Shah, and David Magnus. “Implementing machine learning in health care – addressing ethical challenges”. In: *The New England Journal of Medicine* (2018).
- [5] Y Choi, Antonio Vergari, and Guy Van den Broeck. “Probabilistic circuits: A unifying framework for tractable probabilistic models”. In: *UCLA* (2020).
- [6] Michael Chui, James Manyika, Mehdi Miremadi, Nicolaus Henke, Rita Chung, Pieter Nel, and Sankalp Malhotra. “Notes from the AI frontier: Insights from hundreds of use cases”. In: *McKinsey Global Institute* (2018).
- [7] Alvaro HC Correia, Gennaro Gala, Erik Quaeghebeur, Cassio de Campos, and Robert Peharz. “Continuous mixtures of tractable probabilistic models”. In: *CoRR* (2022).
- [8] Meihua Dang, Anji Liu, and Guy Van den Broeck. “Sparse probabilistic circuits via pruning and growing”. In: *Advances in Neural Information Processing Systems* (2022).
- [9] Adnan Darwiche and Auguste Hirth. “On the (complete) reasons behind decisions”. In: *Journal of Logic, Language and Information* (2023).
- [10] Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. “A continual learning survey: Defying forgetting in classification tasks”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).
- [11] Nicola Di Mauro, Gennaro Gala, Marco Iannotta, and Teresa MA Basile. “Random probabilistic circuits”. In: *Uncertainty in Artificial Intelligence* (2021).
- [12] Sayna Ebrahimi, Mohamed Elhoseiny, Trevor Darrell, and Marcus Rohrbach. “Uncertainty-guided continual learning with bayesian neural networks”. In: *CoRR* (2019).
- [13] Robert Gens and Domingos Pedro. “Learning the structure of sum-product networks”. In: *International Conference on Machine Learning* (2013).
- [14] Zoubin Ghahramani. “Probabilistic machine learning and artificial intelligence”. In: *Nature* (2015).
- [15] Charles R Harris, K Jarrod Millman, Stéfan J Van Der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. “Array programming with NumPy”. In: *Nature* (2020).
- [16] J. D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing in Science & Engineering* (2007).

- 
- [17] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *CoRR* (2014).
- [18] Polina Kirichenko, Mehrdad Farajtabar, Dushyant Rao, Balaji Lakshminarayanan, Nir Levine, Ang Li, Huiyi Hu, Andrew Gordon Wilson, and Razvan Pascanu. “Task-agnostic Continual Learning with Hybrid Probabilistic Models”. In: *CoRR* (2021).
- [19] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. “Overcoming catastrophic forgetting in neural networks”. In: *Proceedings of the National Academy of Sciences* (2017).
- [20] Mykel J Kochenderfer. *Decision making under uncertainty: theory and application*. The MIT Press, 2015.
- [21] Akhil Alfons Kodiyan. “An overview of ethical issues in using AI systems in hiring with a case study of Amazon’s AI based hiring tool”. In: *Researchgate Preprint* (2019).
- [22] Alex Krizhevsky, Geoffrey Hinton, et al. “Learning multiple layers of features from tiny images”. In: (2009).
- [23] Yann LeCun and Corinna Cortes. “MNIST handwritten digit database”. In: (2010). URL: <http://yann.lecun.com/exdb/mnist/>.
- [24] Jeongtae Lee, Jaehong Yoon, Eunho Yang, and Sung Ju Hwang. “Lifelong Learning with Dynamically Expandable Networks”. In: *CoRR* (2017).
- [25] Timothée Lesort, Vincenzo Lomonaco, Andrei Stoian, Davide Maltoni, David Filliat, and Natalia Díaz-Rodríguez. “Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges”. In: *Information Fusion* (2020).
- [26] Zhizhong Li and Derek Hoiem. “Learning without forgetting”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2017).
- [27] Anji Liu, Honghua Zhang, and Guy Van den Broeck. “Scaling Up Probabilistic Circuits by Latent Variable Distillation”. In: *CoRR* (2022).
- [28] Jiri Matas and Josef Kittler. “Spatial and feature space clustering: Applications in image analysis”. In: *Computer Analysis of Images and Patterns: 6th International Conference* (1995).
- [29] Martial Mermillod, Aurélia Bugaïska, and Patrick Bonin. “The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects”. In: *Frontiers in Psychology* (2013).
- [30] Alejandro Molina, Antonio Vergari, Nicola Di Mauro, Sriraam Natarajan, Floriana Esposito, and Kristian Kersting. “Mixed sum-product networks: A deep architecture for hybrid domains”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* (2018).
- [31] Jose G Moreno-Torres, Troy Raeder, Rocío Alaiz-Rodríguez, Nitesh V Chawla, and Francisco Herrera. “A unifying view on dataset shift in classification”. In: *Pattern Recognition* (2012).
- [32] Martin Mundt, Steven Braun, Quentin Delfosse, and Kristian Kersting. “CLEVA-compass: A continual learning evaluation assessment compass to promote research transparency and comparability”. In: *CoRR* (2021).
- [33] Martin Mundt, Yongwon Hong, Iuliia Pliushch, and Visvanathan Ramesh. “A wholistic view of continual learning with deep neural networks: Forgotten lessons and the bridge to active and open world learning”. In: *Neural Networks* (2023).
- [34] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. “Reading digits in natural images with unsupervised feature learning”. In: (2011).

- 
- [35] Cuong V Nguyen, Yingzhen Li, Thang D Bui, and Richard E Turner. “Variational continual learning”. In: *CoRR* (2017).
- [36] Sinno Jialin Pan and Qiang Yang. “A survey on transfer learning”. In: *IEEE Transactions on Knowledge and Data Engineering* (2010).
- [37] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. “Pytorch: An imperative style, high-performance deep learning library”. In: *Advances in Neural Information Processing Systems* (2019).
- [38] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. 1988.
- [39] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* (2011).
- [40] Robert Peharz. “Foundations of sum-product networks for probabilistic modeling”. PhD thesis. 2015.
- [41] Robert Peharz, Steven Lang, Antonio Vergari, Karl Stelzner, Alejandro Molina, Martin Trapp, Guy Van den Broeck, Kristian Kersting, and Zoubin Ghahramani. “Einsum networks: Fast and scalable learning of tractable probabilistic circuits”. In: *CoRR* (2020).
- [42] Robert Peharz, Sebastian Tschiatschek, Franz Pernkopf, and Pedro Domingos. “On theoretical properties of sum-product networks”. In: *Artificial Intelligence and Statistics* (2015).
- [43] Robert Peharz, Antonio Vergari, Karl Stelzner, Alejandro Molina, Martin Trapp, Kristian Kersting, and Zoubin Ghahramani. “Probabilistic Deep Learning using Random Sum-Product Networks”. In: *CoRR* (2018).
- [44] Benedikt Pföhl, Alexander Gepperth, and Benedikt Bagus. “Continual Learning with Fully Probabilistic Models”. In: *CoRR* (2021).
- [45] Hoifung Poon and Pedro Domingos. “Sum-product networks: A new deep architecture”. In: *IEEE International Conference on Computer Vision Workshops* (2011).
- [46] Joaquin Quinonero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D Lawrence. “Dataset shift in machine learning”. In: *The MIT Press* (2008).
- [47] Anthony Robins. “Catastrophic forgetting, rehearsal and pseudorehearsal”. In: *Connection Science* (1995).
- [48] Cynthia Rudin. “Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead”. In: *Nature Machine Intelligence* (2019).
- [49] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. “Progressive Neural Networks”. In: *CoRR* (2016).
- [50] Raquel Sanchez-Cauce, Iago París, and Francisco Javier Díez. “Sum-product networks: A survey”. In: *CoRR* (2021).
- [51] Andy Shih, Dorsa Sadigh, and Stefano Ermon. “HyperSPNs: compact and expressive probabilistic circuits”. In: *Advances in Neural Information Processing Systems* (2021).
- [52] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. “Continual learning with deep generative replay”. In: *Advances in Neural Information Processing Systems* (2017).
- [53] Katharina Simbeck. “They shall be fair, transparent, and robust: auditing learning analytics systems”. In: *AI and Ethics* (2023).
- [54] Mihaly Varadi and Sameer Velankar. “The impact of AlphaFold Protein Structure Database on the fields of life sciences”. In: *Proteomics* (2022).

- 
- [55] Gido M van de Ven, Tinne Tuytelaars, and Andreas S Tolias. “Three types of incremental learning”. In: *Nature Machine Intelligence* (2022).
  - [56] Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. “A comprehensive survey of continual learning: Theory, method and application”. In: *CoRR* (2023).
  - [57] Ju Xu and Zhanxing Zhu. “Reinforced continual learning”. In: *Advances in Neural Information Processing Systems* (2018).
  - [58] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. “Lifelong learning with dynamically expandable networks”. In: *CoRR* (2017).