# Learning to Sample from Sum-Product Networks

**Lernen von Stichprobenwahlen aus Summen-Produkt Netzwerken**
Master thesis by Felix Divo (Student ID: 2365510)
Date of submission: May 16, 2022

Prüfer*in: Prof. Dr. Kristian Kersting
Betreuer*in: Steven Lang
Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Computer Science Department

Artificial Intelligence and Machine Learning Lab (AIML)

Learning to Sample from Sum-Product Networks
Lernen von Stichprobenwahlen aus Summen-Produkt Netzwerken

Master thesis by Felix Divo (Student ID: 2365510)

Prüfer*in: Prof. Dr. Kristian Kersting
Betreuer*in: Steven Lang

Date of submission: May 16, 2022

Darmstadt

## Erklärung zur Abschlussarbeit gemäß §22 Abs. 7 APB TU Darmstadt

Hiermit versichere ich, Felix Divo, die vorliegende Masterarbeit gemäß §22 Abs. 7 APB der TU Darmstadt ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, May 16, 2022

F. Divo

# Zusammenfassung

Summen-Produkt Netzwerke (SPNs) sind ein Ansatz des Maschinellen Lernens, um die Dichteverteilung eines gegebene Datensatzes unüberwacht zu lernen. Sie sind probabilistische Schaltkreise, welche einfache Wahrscheinlichkeitsverteilungen durch Summen- und Produktknoten zu komplexeren Verteilungen kombinieren, die das effiziente Auswerten beliebiger gemeinsamer, marginaler und bedingter Wahrscheinlichkeiten erlauben. Obwohl sie Datensätze so modellieren können, dass beispielsweise Klassifizierung mit ihnen möglich ist, gibt es bei ihren generativen Fähigkeiten noch viel Raum für Verbesserung.

Um auszumachen, wie die Qualität der erzeugten Daten verbessert werden kann, wird zuerst die übliche Stichprobenwahl theoretisch untersucht. Sie führt eine Tiefensuche durch, bei der in Summenknoten ein einzelnes Kind und in Produktknoten alle Kinder besucht werden. Wir zeigen, dass der entsprechende Algorithmus in linearer Zeit in der Größe des SPNs terminiert, beweisen dass er Stichproben gemäß der vom SPN modellierten Wahrscheinlichkeitsverteilung ohne Verzerrung liefert und dass er numerisch stabil ist.

Wir zeigen dann auf, wie dennoch inkonsistente Stichproben in den gängigen heuristischen Strukturen durch die Unabhängigkeitsannahmen in Produktknoten auftreten können, und leiten daraus eine verbesserte Stichprobenprozedur ab. Die Kernidee ist dabei in dem aktuellen Durchlauf den bereits gegangenen Pfad zu merken, und dadurch im weiteren Verlauf eine informierte Wahl eines Kinds von einem Summenknoten zu treffen. Wir präsentieren eine $Q$-Learning-Formulierung um solch eine Führung zu lernen und diskutieren formale Eigenschaften sowie Limitierungen. Wir zeigen, dass normales und doppeltes $Q$-Learning sich hier empirisch ähnlich gut verhalten und diskutieren typische Konvergenzdauern.

Ein quantitativer Vergleich wird auf drei verschiedenen Graphenstrukturen moderater Größe mit unterschiedlichen Konditionierungsmustern durchgeführt, bei dem fehlende Teile von Bildern aus sechs Datensatzvariationen durch Stichprobenwahlen zu rekonstruieren sind. In $24$ aus $30$ Fällen konnte der mittlere quadratische Fehler reduziert werden, wobei die Verbesserung in drei Fällen bei über $35\%$ lag. In den anderen sechs Fällen wurde keine Verbesserung oder eine kleine Verschlechterung beobachtet. Wenn dieselbe Führung für die Vervollständigung mit der wahrscheinlichsten Erklärung verwendet wird, liegt die Verbesserung bei über $43\%$. Eine darauffolgende qualitative Gegenüberstellung bestätigt einige Verbesserungen, jedoch bleiben die wahrnehmbaren Unterschiede klein.

Im Ergebnis zeigen wir, dass geleitete Stichprobenwahlen eine geeignete Methode zur Verbesserung der Qualität generierter Daten in SPNs sind. Wenn sie auf weitere Traversionsreihenfolgen und die Verwendung eines Funktionsapproximators statt einer $Q$-Tabelle erweitert wird scheint es wahrscheinlich, dass die Technik auch auf größeren SPNs erfolgreich wäre.

# Abstract

Sum-product networks (SPNs) are a machine learning approach used to model the density of a given dataset in an unsupervised fashion. They are probabilistic circuits that combine simple distributions with sum and product nodes to form more complex ones while still allowing for tractable inference of arbitrary joint, marginal, and conditional probabilities. While being able to model many datasets to a degree that facilitates, for example, classification tasks, their generative abilities leave much room for improvement.

To investigate how to improve the quality of generated data, first, the standard sampling procedure is analyzed theoretically. The procedure runs a depth-first search on the graph where in a sum node a single child and in a product node all children are visited. We show that the algorithm runs in linear time in the size of an SPN, proof that it samples from the distribution encoded by the SPN without introducing any bias, and show that it is numerically stable.

We then demonstrate how inconsistent samples can occur nonetheless due to the independence assumptions in product nodes of typical heuristic structures and devise an improved guided sampling procedure. The core idea is to remember which parts of the SPN were traversed in the current sampling run and therefore continue to select subsequent sum node children in an informed way. A $Q$-learning setup to learn such a guide is provided along with a discussion of its formal properties and limitations. Empirically, normal and double $Q$-learning are shown to perform similarly on this task and typical convergence times are discussed.

A quantitative comparison was carried out on three graph structures of moderate size with different conditioning shapes and six image dataset variants, where the task was to reconstruct a missing part of an image by sampling. In $24$ out of $30$ cases, the mean squared error of the reconstruction was reduced, and in three cases by over $35\%$. In the other six cases, no improvement or a small deterioration was observed. If the same guide is used for most probable explanation completions, the improvement even reaches over $43\%$. While a subsequent qualitative comparison confirms some improvements, the perceptual difference is small.

In conclusion, we show that guided sampling is a suitable approach to improving the sample quality of SPNs. If extended to different graph traversal orders and by the use of function approximators instead of a $Q$ table, the method seems likely to be successful on larger SPNs too.

# Contents

# Abbreviations and Notation

## List of Abbreviations

| | |
|---|---|
| **AIML** | *Artificial Intelligence and Machine Learning Lab* at *TU Darmstadt* |
| **AI** | artificial intelligence |
| **AC** | arithmetic circuit |
| **CPU** | central processing unit |
| **DAG** | directed acyclic graph |
| **EF** | exponential family, a type of distribution |
| **EM** | expectation-maximization algorithm |
| **FID** | Fréchet inception distance |
| **GAN** | generative adversarial networks |
| **GPU** | graphics processing unit (here, for general purpose computing) |
| **i.i.d.** | independently and identically distributed |
| **KID** | kernel inception distance |
| **MAP** | maximum a posteriori |
| **MDP** | Markov decision process |
| **MLE** | maximum likelihood estimate |
| **ML** | machine learning |
| **MMD** | maximum mean discrepancy |
| **MNIST** | the modified dataset of grayscale digits of the US *National Institute of Standards and Technology* |
| **MPE** | most probable explanation |
| **MPN** | max-product network |
| **MSE** | mean square error |
| **NN** | (artificial) neural network |
| **PDF** | probability density function |
| **PD** | Poom-Domingos SPN structure |
| **PMF** | probability mass function |
| **RAT-SPN** | a randomized and tensorized SPN structure and implementation |
| **RGB** | image data format: red, green, blue |
| **RL** | reinforcement learning |
| **RV** | random variable |
| **SPN** | sum-product network |
| **SVHN** | a dataset of house number signs from *Google Street View*® |
| **VAE** | variational autoencoders |

## Notation of Symbols and Operators

| | |
|---|---|
| $\mathbb{N}$ | the natural numbers |
| $\mathbb{R}$ | the real numbers |
| $\times$, $\bigtimes_{i=1}^{n} \boldsymbol{B}_i$ | the Cartesian product (over a range of sets $\boldsymbol{B}_i$) |
| $\uplus$, $\biguplus_{i=1}^{n} \boldsymbol{B}_i$ | disjoint union (of sets $\boldsymbol{B}_i$) |
| $\lvert \cdot \rvert$ | the absolute value |
| $\lVert \cdot \rVert$ | the norm of a vector |
| $\boldsymbol{A}^T$ | the transpose of the vector or matrix $\boldsymbol{A}$ |
| $\Omega$ | sample space |
| $X$, $Y$, $X_i$ | a single RV |
| $\boldsymbol{X}$, $\boldsymbol{Y}$ | a set of RVs |
| $x$, $y$, $x_i$ | a single value an RV can take |
| $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{x}_i$ | a tuple of values of an RV set |
| $*$ | missing evidence for an RV $X$ |
| $\mathbf{val}\,(X)$ | set of all values of an RV $X$ |
| $\boldsymbol{x}[Y]$, $\boldsymbol{x}[\boldsymbol{Y}]$ | the projection of $\boldsymbol{x}$ onto $Y/\boldsymbol{Y}$ |
| $\overline{X}$ | the empirical mean of an RV $X$ |
| $\mathrm{Var}(X)$ | the variance of an RV $X$ |
| $\mathbb{E}[X]$ | the expectation of an RV $X$ |
| $\mathbb{I}_{X=x}$ | indicator RV for the case that $X$ takes the value $x$ |
| $\mathcal{U}\,(a \mid \boldsymbol{A})$ | the uniform distribution over the set $\boldsymbol{A}$ |
| $\mathrm{Cat}\,(i \mid \boldsymbol{b})$ | the categorical distribution with parameters $\boldsymbol{b}$ |
| $\mathcal{B}\,(x \mid p)$ | the Bernoulli distribution with parameter $p$ |
| $\mathcal{N}(\mu, \sigma^2)$ | the univariate Gaussian distribution with mean $\mu$ and variance $\sigma^2$ |
| $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ | the multivariate Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ |
| $D_{KL}(P \,\|\, Q)$ | *Kullback-Leibler* divergence of distribution $Q$ to $P$ |
| $\mathcal{S}$, $\mathcal{S}^A$ | an SPN |
| $\mathrm{ch}(N)$ | the set of child nodes of a node $N$ |
| $\mathrm{pa}(N)$ | the set of parent nodes of a node $N$ |
| $\mathrm{sc}(N)$ | the scope of an SPN node $N$ |
| $\mathcal{M}$ | an MDP |
| $\mathcal{R}$ | the set of rewards in an MDP |
| $[\,]$ | the empty list |
| $[a, b, ]$ | the list of two elements $a$ and $b$ |
| $A \parallel B$ | the concatenation of lists $A$ and $B$ |
| $\lvert \cdot \rvert$ | the length of a list |
| $a \leftarrow b$ | assign value $b$ to variable $a$ |
| $a \rightsquigarrow B(a)$ | assign a probabilistic value sampled from distribution $B$ to variable $a$ |
| $\mathcal{O}(\,\cdot\,)$ | big O/*Landau* notation of complexity theory |
| $I_{\text{Algorithm}}$ | the probability induced by a certain probabilistic algorithm |

# Figures, Tables and Algorithms

## List of Figures

## List of Tables



## List of Algorithms

# 1 Introduction

Everybody is talking about artificial intelligence (AI). Or at least in most – if not all – industries, data science and machine learning (ML) are a major beacon of hope for the next big leap in productivity and societal prosperity [51] [59, Preface]. This is backed by the fact that ML methods have found their way into and continue to be increasingly applied to many sectors, including finance, insurance, logistics, entertainment, health care, education, and many more [10]. As of now, many applications employ some members of the large deep neural network family to perform the desired tasks, as the examples later in this section highlight. However, while often defining the so-called "state of the art", and indeed achieving very impressive results as we will see in the next paragraphs, they typically fall short when it comes to providing insights into how results were obtained (explainable AI).

> *I am so clever that sometimes I don't understand a single word of what I am saying.*
> — Oscar Wilde (1854–1900), The Happy Prince and Other Stories

Therefore, there is a large interest in *probabilistic* modeling techniques driven by the fact that many of the models in that wide field try to address these shortcomings [17]. In particular, as their name suggests, they model observations, internal processes, and/or outputs explicitly as probabilities [5, Chap. 8]. They might be one of the most appropriate ways of modeling the various kinds of uncertainty that we have to deal with when handling real-world data.

There are many tasks for which ML methods may be employed, like for example classification, regression, clustering, and sampling – the focus topic of this thesis. Sampling is the process of extracting concrete random instances that are probable according to some probability distribution, which is typically learned from data in this context. It is one of the core operations on distributions and a key building block of probability theory. It, therefore, appears both on its own as well as hidden in many theoretical and applied settings. Viewed through the lens of machine learning, it is part of the broader family of generative methods [6] (see also Section 2.5). They have seen a large revival in recent years, in part due to the great success of Generative Adversarial Networks (GANs) [18, 42]. In practice, sampling procedures can be used for a lot of tasks, including:

- **Completing partially observed data**: Providing multiple completed instances, which are plausible and likely given some partial evidence. This can be used to infer information that is not observed directly, like, for example, the results of a medical test that might be invasive or very expensive to actually perform. Such completed data can then be of immediate use in decision processes or can be used to train downstream systems which cannot inherently handle incomplete data. In addition, it allows to interactively explore conditional (in)dependencies between data attributes by repeated querying.

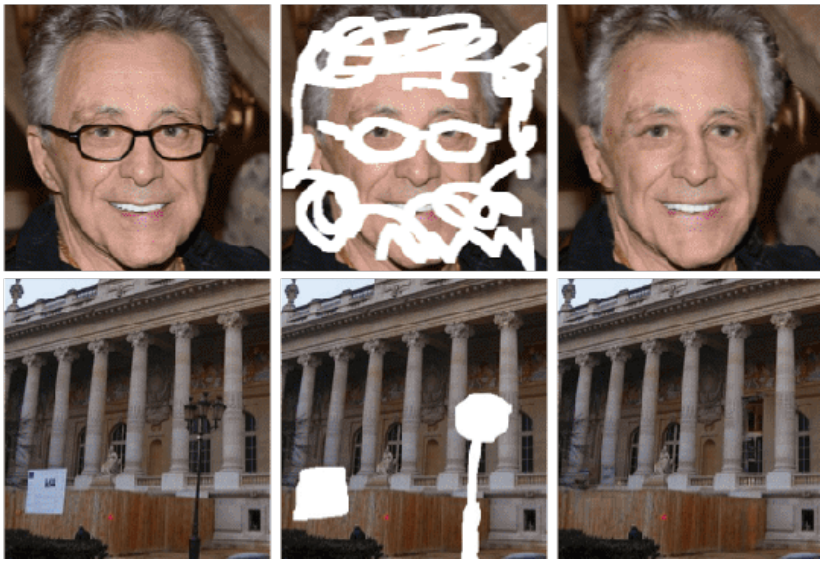


Figure 1.1: Image inpainting results presented by Yu et al. [69, Fig. 1]. Left: Original images, Middle: White area marks the removed parts of the images, Right: Reconstruction of the missing parts combined with the unchanged images.

Another practical application of data completion is image reconstruction or *inpainting*, where the goal is to plausibly reconstruct a rectangular or free-form masked out area. The idea was already explored for the evaluation of the initial sum-product networks [54, Sec. 5]. Some recent and visually impressive results are shown in Figure 1.1. By now, image inpainting is available in commercial tools too.[1]

- **Generating new instances**: Sampling can produce new *synthetic data* from a given data distribution that, for example, contains personally identifiable information. The synthetic data would (ideally) still share the same characteristics as the instances from the original dataset, but at the same time, privacy issues might be resolved by effectively anonymizing the data despite highly personally identifiable traits. On the other hand, it might be difficult to do this with extremely large models since they might have the capacity to memorize some personally identifiable information [8].

  The generation of such synthetic sensitive data has already been explored for health data in multiple studies on images, electrocardiograms and other databases [2, 52, 53, 67]. In addition, there are many applications particularly for the generation of images, with existing applications in marketing, education, art, and others, like for example the portrait generation of *StyleGAN* [26], which has since matured into a commercial service[2]. Some example images are shown in Figure 1.2.

- **Data augmentation**: One can use generative processes to provide more data for training downstream models or to prevent overfitting [56]. So far, this has been mostly applied in various forms to image data using variational autoencoders (VAEs) and GANs [56, pp. 20ff]. In addition, models like GANs have also proven to be able to inter- and extrapolate to unseen data to some degree [7]. This can fill blind spots in datasets with reasonable synthetic instances.

[1] For example the "Content-Aware Fill" in *Adobe Photoshop*® version 23.0: `https://helpx.adobe.com/photoshop/using/content-aware-fill.html`.
[2] `https://generated.photos/`

Figure 1.2: Generated images using *StyleGAN*, taken from [26, Fig. 10]. The images reflect the distribution of the *LSUN bedroom* dataset [68], on which the model was trained on.

- **Model evaluation**: It can help to qualitatively determine whether a learned model is adequate by inspecting generated samples. However, it should be noted that qualitatively assessed good sampling does not imply suitability for other tasks, like classification, since important modes of the data distribution might be missing [20, Sec. 20.14]. Other methods – like verifying test dataset likelihoods – should still be applied.

## 1.1 Motivation

One method for probabilistic modeling are sum-product networks (SPNs), which are a density estimation method and structure that can be estimated from data. They have many desirable properties like enabling linear-time probabilistic inference of arbitrary joint, marginal and conditional distributions as well as (approximate) most probable explanation (MPE) queries, as laid out in the following chapter. While they can also be sampled from directly, it turns out that the quality of such generated instances leaves a lot to be desired. This mainly stems from the use of heuristic graph structures, which have to be used when scaling to larger datasets since learned structures currently do not scale sufficiently to large data dimensions. To improve upon the current state of the art, this thesis will investigate the following questions:

- How precisely does the standard sampling procedure in SPNs work? Does it introduce any bias or numerical issues? What is its time complexity? (Chapter 3)
- What are its shortcomings and how can the introduction of a guide assist? What theoretical framework can this be analysed with? (Chapter 3)
- How does guided sampling compare to standard sampling in a quantitative and qualitative evaluation? How are they related empirically? (Chapter 4)
- What can be done in the future to improve the sample quality using SPNs even further? (Chapter 5)

## 1.2 Overview

The rest of this thesis is structured as follows: We first introduce the terminology and basic concepts used throughout this work in Chapter 2, where much focus is given to SPNs and their theoretical properties. That chapter also presents the related work. We continue with a description and formal study of the standard sampling method in SPNs in Chapter 3. From this, shortcomings and an improved guided sampling procedure are derived. The chapter closes with a reinforcement learning setup to learn such a guide and a formal connection to MPE. Next, various experiments are carried out in Chapter 4 for evaluating the sample quality of the guided sampling procedure compared with standard sampling. We summarize and discuss the insights gained by the previous chapters and close with an outlook on future areas of research in Chapter 5. Finally, the Appendix provides background and further details on select topics.

# 2 Background

This section will provide the basic terminology and covers the foundations of this thesis. Much focus is given to the protagonist data structure of this work: Sum-product networks. Other topics describing methods that are used more as tools, like reinforcement learning in Section 2.4, are described more briefly in later sections. Finally, the related literature is reviewed in the context of generative modeling and sampling from SPNs.

## 2.1 Sum-Product Networks

Sum-product networks (SPNs) are a type of probabilistic circuit, a class of graphs that model probability distributions by explicit computational semantics of the nodes for density estimation [62, 63]. They were originally proposed by Poon and Domingos in 2012 [54] as a method for compactly representing network polynomials. More recently, they have instead been viewed as deep structures that combine probability distributions hierarchically, building complex distributions from simpler ones [43]. These structures as well as their parameterization can also be learned from data, for which multiple algorithms have been proposed (see Section 2.1.4). The following Figure 2.1 shows a simple example of an SPN.

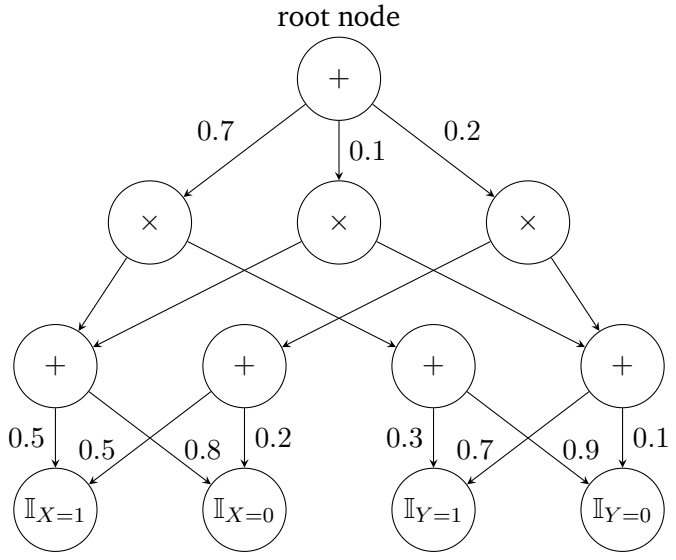



Figure 2.1: A simple, valid SPN which shows sum and product nodes as well as leaves over two random variables $X$ and $Y$ that can take either of the Boolean values $0$ and $1$. The top node is the unique root of the SPN. Note that while this SPN has a very regular structure, this generally does not have to be the case.

SPNs are rooted, directed acyclic graphs (DAGs) with simple uni- or multivariate distributions over random variables (RVs) as leaves and inner nodes consisting of an arbitrary nesting of:

- *Sum Nodes* ($+$): These model a normalized weighted sum (convex combination) over their children, i.e. the edges to their children carry non-negative weights which sum to one for each sum node. We require the child nodes to all range over the same RVs, which gives rise to the interpretation of sum nodes representing a mixture of probability distributions given by the child SPNs.
- *Product Nodes* ($\times$): These model a factorization of the distribution into the ones represented by their child nodes, assuming their mutual independence. The edges to their children have no associated weights.

Note that the only parameters in a fixed SPN structure are the weights of the edges going out of sum nodes and potentially parameters of the leaf distributions, like for example the mean $\mu$ of a Gaussian leaf.

SPNs are interesting due to a variety of properties that allow them to compute answers to many inference queries in time and space that is linear in the size of the DAG. They can efficiently compute joint probabilities for total evidence, i.e. the probability of a complete observation. Moreover, they allow efficient and easy-to-implement computation of any marginals for incomplete evidence. This in turn allows the evaluation of arbitrary conditional probabilities in merely two passes over the data structure using Bayes Theorem. It is still linear in the size of the graph as shown later and includes arbitrary posterior probabilities. In addition to that, with the restriction of *selectivity* on the structure, they enable linear-time computation of *most probable explanations* (MPEs), i.e. the most likely states of all remaining variables given some incomplete evidence. However, marginalization over some variables in MPE queries, called *maximum a-posteriori* (MAP), and MPE in non-selective SPNs remain intractable and in fact even NP-hard [43] [47, Sec. 5.3].

### 2.1.1 Formal Definition

In this section, we introduce the notation necessary for this thesis while assuming a basic familiarity with the topic. For a much more foundational and extensive definition going into the measure-theoretic foundations and probability theory, please refer to the dissertation of Peharz [47, Chap. 2 and 4], where the terminology and notation used here were largely drawn from. In addition to that, the survey of París, Sánchez-Cauce, and Díez [43] gives a well-written, yet brief overview.

Firstly, we need to define some basic terms and expressions:

**Definition 1** (Notation for probability distributions)**.** *Let $X_1, \ldots, X_n$ denote finite-valued discrete RVs with a probability mass function (PMF) $p(X_i)$. We then define the following terms and notation:*

1. *$\boldsymbol{X} := (X_1, \ldots, X_n)$ is a vector of RVs, where we also use set notation like $\boldsymbol{X} \cup \boldsymbol{Y}$ for convenience, being aware that they are technically different objects. $p(\boldsymbol{X})$ denotes the joint PMF over the RVs in $\boldsymbol{X}$.*
2. *The set of all values $X(\omega)$ that a RV $X$ takes under all events $\omega \in \Omega$ in the sample space $\Omega$ is denoted by $\mathbf{val}(X) := \{X(\omega) \,|\, \omega \in \Omega\} = \{x_1, \ldots, x_k\}$.*
3. *Similarly, for a vector of RVs $\boldsymbol{X} = (X_1, \ldots, X_n)$ we define $\mathbf{val}(\boldsymbol{X}) := \bigtimes_{i=1}^{n} \mathbf{val}(X_i)$, i.e. the set of all joint state $n$-tuples of $\boldsymbol{X}$. Such a joint state is called* complete evidence *and denoted by $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbf{val}(\boldsymbol{X})$.*

4. *Another type of evidence is called* incomplete evidence*, where we have complete evidence for some RVs $\boldsymbol{X}$ and none about other variables $\boldsymbol{Y}$.*[3] *In this case, any component of a state tuple belonging to $\boldsymbol{Y}$ takes the placeholder value $*$ to denote the absence of evidence for those particular RVs.*

5. *Let $\{X_{i_1}, \ldots, X_{i_k}\} = \boldsymbol{Y} \subseteq \boldsymbol{X} = \{X_1, \ldots, X_n\}$ be a subset of the RVs of $\boldsymbol{X}$. For a particular complete evidence vector $\boldsymbol{x} = (x_1, \ldots, x_n)$ we define the* projection of $\boldsymbol{x}$ onto $\boldsymbol{Y}$ *as $\boldsymbol{x}[\boldsymbol{Y}] := (x_{i_1}, \ldots, x_{i_k})$. For the sake of a modest notation, for a single RV $X_j$, let $\boldsymbol{x}[X_j] := x_j$. The projection of components with* incomplete *evidence is defined analogously, where any of $x_{i_1}, \ldots, x_{i_k}$ may be placeholders $*$.*

6. *Equivalent definitions shall hold for any continuous real-valued RVs $X_1, \ldots, X_n$, where $p(X_i)$ defines the probability density function (PDF) and $P(X_i)$ the cumulative probability function. Note, that in this case $\mathbf{val}(X_i)$ is uncountably infinite.*

**Definition 2** (Indicator variables)**.** *Given a discrete RV $X$ over a finite number of states and a state $x \in \mathbf{val}(X)$, we define an* indicator variable *(IV) $\mathbb{I}_{X=x}$ as being $1$ if $X$ takes the value $x$ and else $0$. We also allow this notation for joint states $\boldsymbol{y}$ of a set of RVs $\boldsymbol{Y}$ (with both $X \in \boldsymbol{Y}$ or $X \notin \boldsymbol{Y}$) by defining the overloaded $\mathbb{I}_{X=x}(\boldsymbol{y}) : \mathbf{val}(\boldsymbol{Y}) \to \{0, 1\}$:*

$$\mathbb{I}_{X=x}(\boldsymbol{y}) := \begin{cases} 1 & \text{if } X \notin \boldsymbol{Y} \vee \boldsymbol{y}[X] = x, \text{ and} \\ 0 & \text{else} \end{cases}.$$

*$\mathbb{I}_{\boldsymbol{X}}$ describes the vector of all such variables and functions for a given set of RVs $\boldsymbol{X}$.*

With this, we can now define basic SPNs. We will first define SPNs for discrete RVs over finite states and then extend them to continuous RVs over $\mathbb{R}$. SPNs are a rather simple structure, formally defined by:

**Definition 3** (SPNs over finite, discrete distributions)**.** *A* sum-product network *$\mathcal{S}$ over a set of discrete RVs $\boldsymbol{X}$ (with finitely many states) is a rooted DAG with a vector of weights $\boldsymbol{w}$ consisting of three types of nodes $N$, each mapping $\boldsymbol{y}$ ($\boldsymbol{Y} \subseteq \boldsymbol{X}$) to its* value *$N(\boldsymbol{y}) \in \mathbb{R}$:*

- Indicator nodes*:*
$$N(\boldsymbol{y}) := \mathbb{I}_{X=x}(\boldsymbol{y})$$
- Sum nodes *with weights $w_C \geq 0$ on the edges to each child $C$:*
$$N(\boldsymbol{y}) := \sum_{C \in \mathrm{ch}(N)} w_C \cdot C(\boldsymbol{y}), \text{ where } \sum_{C \in \mathrm{ch}(N)} w_C = 1$$
- Product nodes*:*
$$N(\boldsymbol{y}) := \prod_{C \in \mathrm{ch}(N)} C(\boldsymbol{y})$$

*All leaves are indicator nodes and all internal nodes are either sum or product nodes (over one or more children each). The* value *$\mathcal{S}(\boldsymbol{y})$ of the entire graph is given by the value of its root.*

[3] We do not consider *partial evidence* [47, Sec. 4.1], where there is only partial information on the set of values $\boldsymbol{\mathcal{Z}} \subseteq \mathbf{val}(\boldsymbol{X})$ that a variable $Z$ can take. For example, $\boldsymbol{\mathcal{Z}}$ might encode that some of the real-valued RVs are not fully determined but still restricted to be a member of some interval.

A visual example is given in Figure 2.1 and an example for computing values will follow below. The goal of constructing SPNs is to combine primitive distributions – like categorical ones – into deeper and more complex ones. However, without further restriction, the values of general SPNs do not represent true probability distributions, i.e. non-negative functions where for any RV $X$, it holds that

$$\sum_{x \in \mathbf{val}(X)} P(x) = 1.$$

The following example illustrates that:

**Example 2.1.1** (See also [43, Ex. 33 and 34])**.** *Let us consider the two very simple SPNs in Figures 2.2 and 2.3 over two independent binary RVs $Y$ and $Z$, combined in the set $\boldsymbol{\mathcal{S}} = \{Y, Z\}$. Note that* $\mathbf{val}(Y) = \mathbf{val}(Z) = \{0, 1\}$.

*A) Let the first SPN be given by:*

$$\begin{aligned}\mathcal{S}^A(\boldsymbol{s}) &:= \mathcal{S}^A((y, z)) \\ &= 0.4 \cdot \mathbb{I}_{Y=0}(y) + 0.6 \cdot \mathbb{I}_{Z=0}(z)\end{aligned}$$

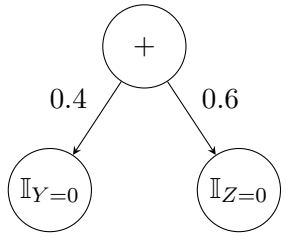


Figure 2.2: Visualization of the SPN $\mathcal{S}^A$

*If we wanted to compute the probability of marginalizing out both $Y$ and $Z$ using $\mathcal{S}^A$, we could compute (marginalizing out $Z$ immediately):*

$$\begin{aligned}P(Y = 0) &\overset{!}{=} \mathcal{S}^A((0, *)) = 0.4 \cdot 1 + 0.6 \cdot 1 = 1, \\ P(Y = 1) &\overset{!}{=} \mathcal{S}^A((1, *)) = 0.4 \cdot 0 + 0.6 \cdot 1 = 0.6.\end{aligned}$$

*Adding both should yield $1$ due to the axioms of probabilities, but instead $P(Y = 0) + P(Y = 1) \overset{!}{=} 1.6 > 1$. So clearly, what we have computed are not real probabilities!*

*B) Now, let the second SPN be given by:*

$$\mathcal{S}^B(\boldsymbol{s}) := \mathbb{I}_{Y=0}(y) \cdot \mathbb{I}_{Y=1}(y)$$

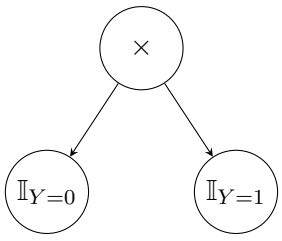


Figure 2.3: Visualization of the SPN $\mathcal{S}^B$

*Like above, we compute*

$$\begin{aligned}P(Y = 0) &\overset{!}{=} \mathcal{S}^B((0, *)) = 1 \cdot 0 = 0, \\ P(Y = 1) &\overset{!}{=} \mathcal{S}^B((1, *)) = 0 \cdot 1 = 0,\end{aligned}$$

*and arrive at a sum unequal to $1$: $P(Y = 0) + P(Y = 1) \overset{!}{=} 0 < 1$. Again, the values of $\mathcal{S}^B$ are no probabilities.*

For the characterization of well-behaved SPNs, we require the notion of *scopes*, which is intuitively just the RVs that a given SPN ranges over. Formally, we define:

**Definition 4** (Scopes). *The* scope $\text{sc}(\mathcal{S})$ *of an SPN* $\mathcal{S}$ *over RVs* $\boldsymbol{X}$ *is given by the scope of its root, which is defined recursively on the vertices* $\boldsymbol{V}$ *of the DAG, where*

$$\text{sc} : \boldsymbol{V} \to 2^{\boldsymbol{X}}$$
$$\text{sc}(N) := \begin{cases} \bigcup_{C \in \text{ch}(N)} \text{sc}(C) & \text{if } N \text{ is an inner node (either sum or product node), and else} \\ \{X\} & \text{if } N \text{ is a leaf node over the RV } X. \end{cases}$$

**Example 2.1.2.** *Consider* $\mathcal{S}^A$ *from Example 2.1.1.A (see Figure 2.2). The left child has scope* $\text{sc}(\mathbb{I}_{Y=0}) = \{Y\}$ *and the right one* $\text{sc}(\mathbb{I}_{Z=0}) = \{Z\}$. *Consequently, for the root node* $N$: $\text{sc}(N) = \{X, Y\} = \boldsymbol{\mathcal{S}} = \text{sc}\left(\mathcal{S}^A\right)$.

**Definition 5** (Sub-SPNs, also called sub-networks). *For a given SPN* $\mathcal{S}$ *with weights* $\boldsymbol{w}$ *and one of its nodes* $N$, *we call the sub-DAG rooted at* $N$ *a* sub-SPN *and denote that SPN as* $\mathcal{S}_N$.

It is easy to see by induction over the structure that we can narrow down the set of RVs when traversing an SPN from the top:

**Proposition 6.** *Let* $\mathcal{S}$ *be an SPN and* $N$ *any of its nodes. For any* $\boldsymbol{x}$, *it holds that* $N(\boldsymbol{x}) = N(\boldsymbol{x}[\text{sc}(N)])$.

### 2.1.2 Validity

Arbitrary SPNs do not compute probabilities, as we have seen in Example 2.1.1 – they are not *valid*:

**Definition 7** (Validity). *An SPN* $\mathcal{S}$ *is valid iff for any (incomplete) evidence* $\boldsymbol{x}$, $\mathcal{S}(\boldsymbol{x}) = \text{P}_{\mathcal{S}}(\boldsymbol{x})$, *i.e. its value is a joint or marginal probability.*

As described in the introduction in Section 2.1, SPNs compose simple leaf distributions into deeper structures, where sums act as mixtures and products as factorizations. We can establish sufficient conditions for the validity of SPNs by dissecting these intuitions [54]. Firstly, as mixtures describe a distribution by a composition of other distributions *over the same variables*, we should ensure that each child of a sum node has the same scope (→ "completeness"). Secondly, a proper factorization of a distribution into a product of non-conditional children should be composed of a product of non-overlapping distributions, i.e. the scope of the children should be disjoint (→ "decomposability"). There is in fact a weaker and also sufficient notion called "consistency", which is, however, of little practical relevance [43, p. 5].

**Definition 8** (Completeness). *An SPN* $\mathcal{S}$ *is* complete *iff in each sum node* $N$, *all children* $C_i, C_j \in \text{ch}(N)$ *have identical scope:* $\text{sc}(C_i) = \text{sc}(C_j)$.

**Definition 9** (Decomposability). *An SPN* $\mathcal{S}$ *is* decomposable *iff in each product node* $N$, *all children* $C_i, C_j \in \text{ch}(N)$ *have (pairwise) disjoint scope:* $\text{sc}(C_i) \cap \text{sc}(C_j) = \emptyset$.

Due to the definition of scopes (Definition 4), completeness is equivalent to all children $C$ of a sum node $N$ having the same scope as the $N$, i.e. $\text{sc}(C) = \text{sc}(N)$. Similarly, decomposability is equal to the fact that for each product node $P$, the scopes of the children partition the scope of the product node:

$$\text{sc}(N) = \biguplus_{C \in \text{ch}(N)} \text{sc}(C).$$

**Definition 10** (Consistency)**.** *An SPN $\mathcal{S}$ is* consistent *iff in each inner product node $N$, no variable $X$ appears with an indicator for value $x_1$ in one child $C_1$ and for a different value $x_2 \neq x_1$ in a different child $C_2 \neq C_1$.*

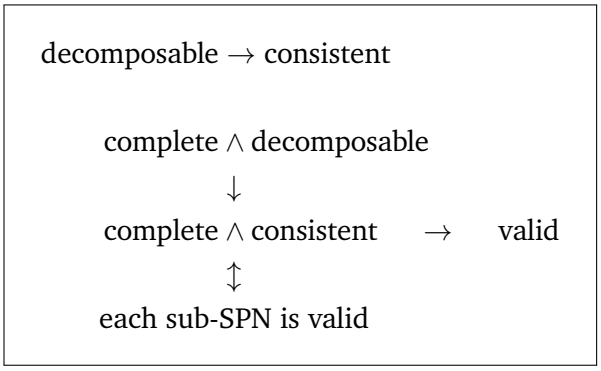



Figure 2.4: Logical connection of the central properties of SPNs related to validity.

It is quite obvious that decomposability implies consistency since no variable $X$ occurs in more than one child at all:

**Proposition 11.** *An SPN $\mathcal{S}$ that is decomposable is also consistent.*

We can finally characterize valid SPNs by purely structural properties:

**Theorem 12** ([54, Thm. 1])**.** *An SPN $\mathcal{S}$ that is complete and consistent is valid.*

**Theorem 13** ([54, p. 3])**.** *An SPN $\mathcal{S}$ is complete and consistent iff each sub-SPN $\mathcal{S}_N$ for any inner node $N$ is valid.*

The relation of the different properties is illustrated in Figure 2.4 and an example is given below. Please also note that invalid SPNs have applications in so-called *discriminative SPNs* too [9, 15], but are out of the scope of this work.

**Example 2.1.3.** *Consider again the SPNs from Example 2.1.1 (see Figures 2.2 and 2.3). We assessed that both are not valid due to the counterexamples that we found. We can see that $\mathcal{S}^A$ is not complete and $\mathcal{S}^B$ is not decomposable (or even just consistent).*

*In contrast, the SPN given in Figure 2.1 is valid since it is complete and decomposable.*

**Example 2.1.4.** *Note that in reverse, validity* does not *imply completeness, consistency, or even just decomposability. This can be seen by the SPN that encodes the following joint probability over two RVs of Boolean domain [54, p. 3]: $\frac{1}{2}\mathbb{I}_{X=0}\mathbb{I}_{Y=0}\mathbb{I}_{Y=1} + \frac{1}{2}\mathbb{I}_{X=0}$. The probability table of all joint and marginal states proofs that this indeed computes true probabilities.*

Since valid SPNs compute probabilities as their values, it makes sense to extend some typical notation of distributions to SPNs:

**Definition 14** (Conditional values)**.** *Let $\mathcal{S}$ be a valid SPN. Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be values of disjoint sets of RVs from the scope of $\mathcal{S}$. Then* conditional values *are defined as $\mathcal{S}(\boldsymbol{x} \mid \boldsymbol{y}) := \mathcal{S}(\boldsymbol{x}, \boldsymbol{y}) / \mathcal{S}(\boldsymbol{y})$.*

### 2.1.3 Generalization to Arbitrary Leaf Distributions

Up until now, we have restricted the basic building blocks of our deep hierarchy of probability distributions to finite-valued discrete distributions. It is straightforward to extend the definition of leaf nodes in SPNs to not only consist of indicator variables for single discrete RVs (compare Definition 3), but instead allow distributions over arbitrary RVs, i.e. both discrete and continuous ones [54, p. 4]. This includes the previous definition of leaf nodes as a special case. It is even possible to extend it further to multivariate distributions [47, Sec. 4.4], which again includes the univariate ones. Essentially, instead of just PMFs we now also allow PDFs in leaves, and the values of SPN nodes are now likelihoods. The previous definitions and theorems hold equivalently for the following wider definition of SPNs:

**Definition 15** (Generalized Sum-Product Network)**.** *A* sum-product network $\mathcal{S}$ *over a set of finite-valued discrete or any continuous, uni- or multivariate RVs* $\boldsymbol{X}$ *is a rooted DAG with a vector of weights* $\boldsymbol{w}$ *consisting of three types of nodes* $N$*, each mapping* $\boldsymbol{y}$ *(*$\boldsymbol{Y} \subseteq \boldsymbol{X}$*) to its* value $N(\boldsymbol{y}) \in \mathbb{R}$*:*

- Leaf *or* distribution nodes $L$ *with an arbitrary distribution over RVs* $\emptyset \neq \boldsymbol{X'} \subseteq \boldsymbol{X}$ *given by a PMF or PDF* $p(\boldsymbol{x'})$*:*

$$L(\boldsymbol{y}) := \begin{cases} 1 & \text{if } \boldsymbol{Y} \cap \boldsymbol{X'} = \emptyset \\ p(\boldsymbol{y}[\boldsymbol{X'}]) & \text{else} \end{cases}.$$

- Sum nodes *as in Definition 3.*
- Product nodes *as in Definition 3.*

*All leaves are distribution nodes and all internal nodes are either sums or products. The* value $\mathcal{S}(\boldsymbol{y})$ *of the entire graph is given by the value of its root.*

**Definition 16** (Generalized Scopes)**.** *The* scope $\text{sc}(\mathcal{S})$ *of a (generalized) SPN* $\mathcal{S}$ *over RVs* $\boldsymbol{X}$ *and its nodes* $\boldsymbol{V}$ *is defined analogously to Definition 4, except for leaf nodes* $L$ *over RVs* $\boldsymbol{X'}$*, where we define* $\text{sc}(L) := \boldsymbol{X'}$*.*

In the following, we will always assume to be working with this more general definition when we talk of SPNs, scopes, and their properties.

### 2.1.4 Learning: Structure and Parameters

Learning in the context of SPNs consists of two problems that can be solved both independently and jointly [43, 62]: *Structure learning* aims to find a tree or DAG structure that can adequately model the density of a given dataset, given proper parameters that are yet to be assigned. Given an SPN as a plain graph, *parameter learning* then finds suitable assignments to the sum node weights $\boldsymbol{w}$ and possibly leaf distribution parameters. This section will not go into the details of parameter learning, as there exist multiple working implementations of it that one can rely on. For example, the libraries used in the experiments described in Chapter 4 implement an online expectation-maximization (EM) algorithm [38, 48]. In the context of sampling, the comparison of different structures is much more fruitful. Note that it is not always necessary to learn the structure from data since one can also hand-craft them and then only perform parameter learning, as was done with the first two structure types below.

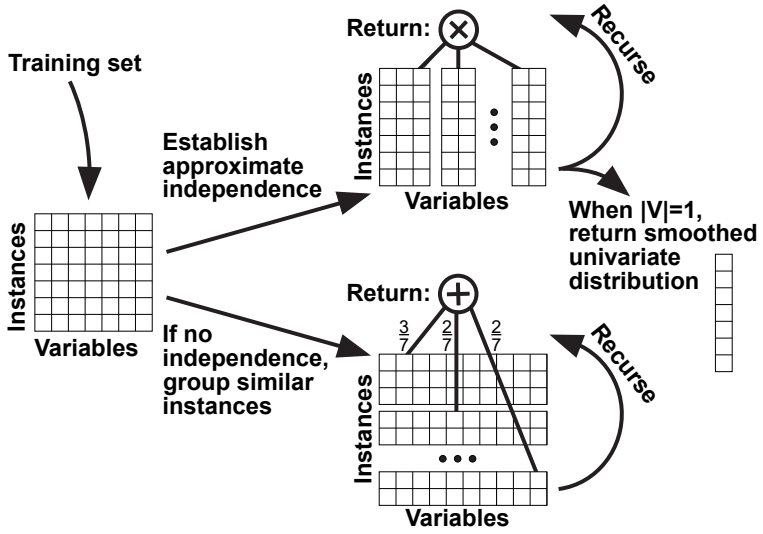



Figure 2.5: Illustration of the LearnSPN procedure which jointly learns structure and parameters of an SPN given some training data. The training set is split recursively and each time either a sum node over data clusters of the same RVs or a product that partitions the set of RVs is grown. Taken from the original publication [16, Fig. 1].

**Poon-Domingos Structure** One of the first proposed structures for SPNs was the one proposed by Poon and Domingos in their first publication [54], where they built a deep hierarchy of features exploiting local structure in images that is reminiscent of convolutional neural networks. To this end, the array-specific structure consists of products over axis-aligned splits and sums over several such splits at multiple levels. In the case of images, the pixel grid is alternatingly split vertically and horizontally in a recursive fashion until a threshold size of the resulting regions is reached. This structure is now called *Poon-Domingos* (PD) although first named `GenerateDenseSPN()` [54, Sec. 5], but the verbal description is arguably too vague to allow for a reliable reproduction of the structure. However, a differently named implementation in the *Java* language is hidden in the accompanying material.[4] Many other libraries like *SPFLow* [38] or *spyn*[5] do not seem to implement it. Nevertheless, an efficient *Python* implementation luckily is available from the *Einsum Networks* publication, where the graph is represented in a special vectorized form to archive higher performance using GPU acceleration [48]. The latter will be used in this thesis after a translation to an explicit graphical representation similar to SPFLow (see Section 4.2.3).

**Binary Tree Structure** This method works on any data domain and not only on arrays by constructing a sum over many balanced trees where each branch covers one half of the scope of the parent. Each branch is then split recursively until a predefined depth is reached. As with the PD structure above, the method does not consider the training data and is therefore a purely heuristic structure. Again, the implementation from Einsum Networks [48] will be used for the experiments.

**LearnSPN** For the sake of simplicity, the popular *LearnSPN* procedure[6] [16, 61] was chosen as the only structure learner that considers the actual training data. There are a multitude of alternative and more

[4] See the file `code/src/spn/SPN.java` on lines 217ff in the method `spn.SPN::init()`.
[5] `https://github.com/arranger1044/spyn`
[6] Not to be confused with the identically named procedure for parameter learning in the original SPN publication [54, Alg. 1].

sophisticated algorithms which are not considered in this work [43, Sec. VI], as will be discussed in Section 4.2.2. The procedure is a framework that allows for different concrete implementations. It hierarchically and alternatingly performs independence tests between sets of RVs to produce product nodes and instance clustering to perform sum nodes. In each step, either the scope of the Sub-SPN or the relevant amount of data gets split up. Eventually, this yields a tree-shaped SPN where the leaves can be uni- or multivariate distributions depending on the cutoff criteria. The original scheme for univariate distributions is shown visually in Figure 2.5.

### 2.1.5 Maximum A-Posteriori and Most Probable Explanation

Let $\mathcal{S}$ be an SPN and $\boldsymbol{E} \subseteq \mathrm{sc}(\mathcal{S})$ a subset of the RVs about which we have complete evidence $\boldsymbol{e}$. We have no evidence on the remaining variables $\mathrm{sc}(\mathcal{S}) \setminus \boldsymbol{E}$. We can now ask what a likely configuration of the missing variables according to the distribution $\mathrm{P}_{\mathcal{S}}$ is, and obtain some concrete ones by conditional sampling as we will explore later in Section 2.1.6. Instead, we can also ask for only the *most* probable values of some or all missing variables. To this end, we can follow the naming and notation of the survey by París, Sánchez-Cauce, and Díez [43] and define two disjoint sets of RVs $\boldsymbol{X}$ and $\boldsymbol{H}$ such that $\boldsymbol{X} \cup \boldsymbol{H} = \mathrm{sc}(\mathcal{S}) \setminus \boldsymbol{E}$. We can then define the so-called *maximum a-posteriori* state as

$$\mathrm{MAP}(\boldsymbol{e}, \boldsymbol{X}) := \operatorname*{arg\,max}_{\boldsymbol{x} \in \mathbf{val}(\boldsymbol{X})} \mathrm{P}_{\mathcal{S}}(\boldsymbol{x} \mid \boldsymbol{e}).$$

Note, that we marginalize over all hidden variables in $\boldsymbol{H}$. In case we are interested in all RVs, that is $\boldsymbol{H} = \emptyset$, we define the special case of the *most probable explanation* as

$$\mathrm{MPE}(\boldsymbol{e}) := \mathrm{MAP}(\boldsymbol{e}, \mathrm{sc}(\mathcal{S}) \setminus \boldsymbol{E}).$$

In this case, we are interested in the state $\boldsymbol{x} \times \boldsymbol{e}$ which has the highest likelihood among all such states for a fixed $\boldsymbol{e}$. In case there are multiple such states, any of them will suffice.

In general, MAP is inherently harder than MPE, and computing any of them is NP-hard in SPNs [49, Sec. 4]. It should be noted that there are several approximations [43, Sec. IV] which are, however, out of the scope of this thesis. There are certain conditions on the structure of SPNs under which MPE computation *is* tractable and even linear in the size of the SPN. This is the case if at each sum node $N$ and for each complete evidence for $\mathrm{sc}(N)$, only one child may propagate a positive probability (and all others are zero). Formally, we define:

**Definition 17** (Selectivity)**.** *A sum node $N$ is said to be* selective *iff for all $\boldsymbol{e} \in \mathbf{val}(\mathrm{sc}(N))$, at most one child $C \in \mathrm{ch}(N)$ has $C(\boldsymbol{e}) > 0$. An SPN is selective iff all its sum nodes are.*

Given this restriction, there is a simple to implement and very efficient algorithm for computing the MPE but not MAP for given incomplete evidence. The necessary computation steps are given in Algorithm 2.1 and it is relevant since it is very similar to the conditional sampling procedures derived later. It performs two passes over the data structure. The first one is an upward pass from the leaves to the root as in any normal inference query, but in this case on each sum node, we propagate the maximum and not the sum over all weighted children's probabilities and remember the (or a) child index $i_{\max}$ where the maximum was reached. We call it UpwardPassMPN($\boldsymbol{e}$) for evidence $\boldsymbol{e}$, where MPN stands for *max-product network*. The second pass is a recursive downward pass originating from the root for assembling the actual MPE. In a product, we recurse into all children since they have scope over disjoint RVs, and in sum nodes, we select the child with the largest

**Algorithm:** MPESelective

**Input** : An SPN $\mathcal{S}$
Evidence $\boldsymbol{e}$ with $\boldsymbol{E} \subseteq \text{sc}(\mathcal{S})$

**Result** : State $\boldsymbol{x}$ of the remaining RVs $\boldsymbol{X} = \text{sc}(\mathcal{S}) \setminus \boldsymbol{E}$

$\mathcal{S}' \leftarrow \text{UpwardPassMPN}(\boldsymbol{e})$ `// Obtain annotated SPN`

$Q \leftarrow$ new queue of nodes

`Put` ($Q$, root of $\mathcal{S}'$) `// Prepare downward pass`

$\boldsymbol{x} \leftarrow \boldsymbol{0}$ `// Initialize the result vector`

**while** *$Q$ is not empty* **do**

  $N \leftarrow$ `Pop` ($Q$)

  **switch** `TypeOf(`$N$`)` **do**

    **case** *leaf* **do**

      $\boldsymbol{Y} \leftarrow \text{sc}(N)$

      $\boldsymbol{x}[\boldsymbol{Y}] \leftarrow \underset{\boldsymbol{y} \in \textbf{val}(\boldsymbol{Y}),\ \boldsymbol{y} \text{ compatible with } \boldsymbol{e}}{\arg\max} \text{P}_N(\boldsymbol{y} \mid \boldsymbol{e})$ `// Compute MPE in leaf distribution`

    **case** *product* **do**

      **foreach** $C \in \text{ch}(N)$ **do**

        `Put` ($Q$, $C$)

    **case** *sum* **do**

      $C_{\max} \leftarrow \underset{C_i \in \text{ch}(N)}{\arg\max}\, w_i C_i(\boldsymbol{e})$ `// Obtained from annotated SPN` $\mathcal{S}'$

      `Put` ($Q$, $C_{\max}$)

**return** $\boldsymbol{x}$

**Algorithm 2.1:** This procedure finds an MPE state $\boldsymbol{x}$ in an SPN given some evidence $\boldsymbol{e}$. It is similar to the pseudocode of Peharz et al. [49, Fig. 7]. It was provided in a textual description as early as in the original paper on SPNs [54, p. 5] – although with a wrong "proof" of its correctness, stating that it would be correct even for non-selective SPNs. It is, however, only guaranteed to be correct for selective SPNs as stated in Theorem 18.

probability (which we have remembered) and only recurse into that one. Once we arrive at a leaf, we only need to compute an MPE state for the single leaf distribution, like the mean $\boldsymbol{\mu}$ for a multivariate Gaussian.

It was shown that this algorithm is indeed correct, and it is easy to see that it is efficient, since each node is visited at most once in the upward and once in the downward pass:

**Theorem 18** (Correctness of MPESelective [49, Sec. 4])**.** *Let $\mathcal{S}$ be a selective SPN and $\boldsymbol{e} \in \mathbf{val}\,(\mathrm{sc}(\mathcal{S}))$ (incomplete) evidence. Then Algorithm 2.1 returns the correct MPE state:* $\mathrm{MPESelective}(\mathcal{S}, \boldsymbol{e}) = \mathrm{MPE}(\boldsymbol{e})$.

**Theorem 19** (Worst-case complexity of MPESelective)**.** *Let the worst-case complexity of estimating any leaf MPE state be $\mathcal{O}(l)$. Then Algorithm 2.1 runs in $\mathcal{O}(s \cdot l)$, where $s$ is the size of $\mathcal{S}$ (i.e. the number of edges).*

In the case of the typically used basic leaf distributions like Poisson, Gaussian, Categorical, Multinomial, etc., MPE states can be found easily in constant time. Thus, finding MPE states in an SPN using these as leaves can be done in linear time in the size of the graph in the worst case. Also, tighter bounds might be possible to prove, since only a sub-*tree* of the SPN and not of the general DAG is actually visited. This is also called the *induced* tree, since the entire probability mass for that query instance $\boldsymbol{e}$ lies on the tree [43, Sec. III-F, Prop. 21].

In a general non-selective SPN, this algorithm is still efficient. However, it can lead to severe deviations from the true MPE [43, p. 8] [49, Sec. 4]. It is possible to efficiently turn any SPN $\mathcal{S}$ into an *augmented* SPN $\mathcal{S}'$ that is selective, but unfortunately an MPE in $\mathcal{S}'$ is not necessarily an MPE in $\mathcal{S}$ or even guaranteed to be close to it. In practice however, it is commonly used as an approximation [55] and also called *Best Tree* algorithm in that context [36]. Other approaches include designing even invalid structures and one-hot queries in a way that ensures correct MPE computation as done by Cheng et al. [9], although they do not provide proof that their approach computes true MPEs.

### 2.1.6 Sampling

SPNs allow for values to be sampled from the probability distribution they represent. The procedure is very similar to the MPE computation in Algorithm 2.1 but does not require selectivity for correctness. When sampling, one also traverses the graph and chooses a single child in sum nodes and visits each child in product nodes. However, the choice in sum nodes is not done deterministically like for MPE, and instead, a child is sampled with probability proportional to its weight and the evidence marginal likelihood in that child. The procedure is stated as pseudocode and thoroughly analyzed in Section 3.1.

## 2.2 Evaluation of Generative Models

The following measures determine through different means how likely a given instance or set thereof is to belong to some training or test dataset. More similar instances should produce lower scores while instances not akin to the reference data should produce high scores. These functions are important to (1) evaluate the performance and improvement of the methods being developed and (2) in order to serve as a reward signal in reinforcement learning settings (see Section 2.4). Most of them are not metrics in strict mathematical terms but intuitively measure distances between the data distribution $p_d$ implicitly given by $N$ i.i.d. sampled data points $\boldsymbol{D} = \{\boldsymbol{d}_1, \ldots, \boldsymbol{d}_N\}$ and a given instance $\boldsymbol{x}$.

The distance of multiple data points to the dataset is typically determined by averaging the individual distances of the instances. This is, for example, used in minibatch stochastic gradient descent (SGD), an optimization routine often employed in learning neural networks (NNs).

### 2.2.1 Nearest Neighbor Distance and Reconstruction Errors

One can assume that, on average, instances in $\boldsymbol{D}$ are "close" to the distribution represented by $\boldsymbol{D}$, since by definition, they were sampled from it. In turn, if an instance $\boldsymbol{d}'$ is largely similar to some $\boldsymbol{d} \in \boldsymbol{D}$, it is likely to be sampled from $p_d$ too. However, since simple Nearest Neighbor searches like $k$-d trees are computationally expensive in high-dimensional spaces like images, determining a meaningful distance of an arbitrary instance $\boldsymbol{d}'$ to the nearest or $k$ nearest neighbors is difficult or even intractable [4, 14]. When we switch the perspective a little bit, we can see that we can compute the distance easily with any vector metric once we know the nearest neighbor. And in fact, we *can* construct data in such a way that we know a probable nearest neighbor by taking an arbitrary point from the dataset $\boldsymbol{d} \in \boldsymbol{D}$, removing some parts of it to obtain $\boldsymbol{d}_{\text{partial}}$, and looking at the complete reconstructed $\boldsymbol{d}'$. We can assume that the nearest neighbor of $\boldsymbol{d}'$ is probably just $\boldsymbol{d}$, and the "distance" of $\boldsymbol{d}'$ to $p_d$ is then approximately $\|\boldsymbol{d} - \boldsymbol{d}'\|$.

In the case of image domains, obtaining $\boldsymbol{d}_{\text{partial}}$ amounts to simple partial occlusion, for example by removing half of the pixels from one side of the image. In addition, it is desirable to punish large deviations in any individual dimension more than little ones in many dimensions, as we expect some noise in the observations anyways. So when using popular Minkowski metrics, it is better to choose Euclidean (L2) over Manhattan (L1) distances for these *reconstruction errors*. In addition, it is common to square the distance and obtain a measure that is proportional to the Euclidean distance, called *Mean Squared Error* (MSE). For $K$-dimensional vectors $\boldsymbol{d}, \boldsymbol{d}' \in \mathbb{R}^K$ it is defined as:

$$\begin{aligned}
\text{MSE}(\boldsymbol{d}', \boldsymbol{d}) :=& \frac{1}{K} \sum_{k=1}^{K} \left(d_k - d'_k\right)^2 \\
\propto& \sum_{k=1}^{K} \left(d_k - d'_k\right)^2 \\
=& \left( \sqrt[2]{\sum_{k=1}^{K} \left(d_k - d'_k\right)^2} \right)^2 \\
=& \left\|\boldsymbol{d} - \boldsymbol{d}'\right\|_2^2.
\end{aligned}$$

Note that the *mean* in MSE is usually the mean over multiple samples, and not like here a normalization over the number of dimensions. This rescaling is, however, useful when directly comparing errors on different datasets as they are all scaled to similar orders of magnitude.

### 2.2.2 Other Metrics

Often, other metrics are used in the evaluation of generative models, namely the *Fréchet inception distance* (FID), *kernel inception distance* (KID) and *maximum mean discrepancy* (MMD) [6, Sec. 7]. However, all of these are only defined and meaningful on more than a single data instance, making them less ideal for a reward

function. In addition, the inception distances have unclear reliability on datasets like MNIST as the neural network that is used for the feature transformation was trained on very different images and transformation into that specific image space is not trivial too [44]. The task of evaluating generative models is difficult [20, Sec. 20.14], and using reconstruction errors as a basis avoids many problems.

## 2.3 Exponential Family Probability Distributions

The exponential families (EFs) are a type of distribution that generalize many common ones like Gaussian, Binomial, Poisson, Geometric, Gamma, and others [29, pp. 261ff]. They are attractive to machine learning applications like SPN leaf distributions since a single framework and learning procedure can encompass a wide range of distributions that are typically of interest. Following the notation of the Einsum Networks paper [48], an exponential family is defined as:

**Definition 20** (Exponential family [29, Def. 8.1])**.** *Let $\boldsymbol{X}$ be a set of RVs. An* exponential family *$p$ over $\boldsymbol{X}$ is defined by*

- *a* sufficient statistics *function $\boldsymbol{T} : \boldsymbol{x} \to \mathbb{R}^N$,*
- natural parameters $\boldsymbol{\theta} \in \mathbb{R}^N$*,*
- *a matching* log-normalizer/log-partition function $A : \boldsymbol{\theta} \to \mathbb{R}$*, and*
- *a* base measure $h : \boldsymbol{x} \to \mathbb{R}$.

*The distribution is given by*

$$p(\boldsymbol{x}) = \frac{h(\boldsymbol{x})}{\exp\left(A(\boldsymbol{\theta})\right)} \exp\left(\boldsymbol{T}(\boldsymbol{x})^T \boldsymbol{\theta}\right), \text{ or equivalently in log-space by}$$
$$\log p(\boldsymbol{x}) = \log h(\boldsymbol{x}) + \boldsymbol{T}(\boldsymbol{x})^T \boldsymbol{\theta} - A(\boldsymbol{\theta}).$$

It is often useful to have a conversion between the typical parameters of a distribution, like for example $\mu$ and $\sigma^2$ of a univariate Gaussian $\mathcal{N}\left(\mu, \sigma^2\right)$, and the natural parameters of the EF denoted as $\boldsymbol{\theta}$. This is also required when converting between implementations using different representations of the same distribution. For the purposes of this thesis, only univariate Gaussian and Binomial distributions are considered. Further information can be found in the book *Probabilistic Graphical Models: Principles and Techniques* of Koller and Friedman (2009) [29, Sec. 8] and the "digest with flash cards" of Nielsen and Garcia (2011) [41].

### Univariate Gaussian Distributions

The PDF of univariate Gaussians $\mathcal{N}\left(\mu, \sigma^2\right)$, which were used in the experiments in this thesis, is given by

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right).$$

We can therefore formulate $\mathcal{N}\left(\mu, \sigma^2\right)$ as an exponential family with parameters $\boldsymbol{\theta}(\mu, \sigma^2)$ using the following translation [29, Ex. 8.3] [41, pp. 3f and 15]:

$$\begin{aligned} \boldsymbol{T}(x)^T &= \begin{pmatrix} x & x^2 \end{pmatrix} \\ \boldsymbol{\theta}(\mu, \sigma^2) &= \begin{pmatrix} \frac{\mu}{\sigma^2} \\ -\frac{1}{2\sigma^2} \end{pmatrix} \\ A(\boldsymbol{\theta}(\mu, \sigma^2)) &= \frac{\mu^2}{2\sigma^2} \\ h(x) &= \frac{1}{\sigma\sqrt{2\pi}} \end{aligned} \tag{2.1}$$

In the leaves of Einsum Networks, the parameters of such an EF are learned. Although they are stored in their so-called "expectation form" $\boldsymbol{\phi}$ [48, Sec. 3.5], a conversion to $\boldsymbol{\theta}$ is implemented alongside. Given $\boldsymbol{\theta}^T = \begin{pmatrix} \theta_1 & \theta_2 \end{pmatrix}$, one can then recover the parameters of the Gaussian distribution for use in SPFlow using equation (2.1) by:

$$\sigma^2 = -\frac{1}{2\theta_2} \qquad \text{and} \qquad \mu = \theta_1 \sigma^2.$$

Note, that this also includes isotropic *multivariate* Gaussians, since in that case all RVs can be converted individually in the same way as done for the case with univariate Gaussians.

#### Binomial Distributions

Similarly as for Gaussians, the PMF of the Binomial distribution

$$p(x) = \binom{n}{x} p^x (1-p)^{n-x}, \quad x \in \{0, \ldots, n\}$$

can also be written as an EF. For the sake of brevity, only the two conversions between the natural parameters $\boldsymbol{\theta} \in \mathbb{R}^1$ and the *success probability* $p$ are provided here [41, p. 22, called $\boldsymbol{\Theta}$ and $\boldsymbol{\Lambda}$]:

$$\begin{aligned} \boldsymbol{\theta}(p) &= \log\left(\frac{p}{1-p}\right), \quad \text{and} \\ p(\theta) &= \frac{\exp(\theta)}{1+\exp(\theta)}. \end{aligned}$$

## 2.4 Reinforcement Learning

Reinforcement learning (RL) is one of the broad main groups of methods in machine learning, alongside the well-known supervised (including classification and regression) and unsupervised (including clustering and density estimation) settings [57, Sec. 1.1]. Note, that this taxonomy does not attempt to be complete, as there are more approaches and hybrids, like for example self-supervised learning. In RL, the abstract setting is for an *agent* to learn good *actions* in a given *environment*. In contrast to supervised learning, however, the agent is not explicitly told which actions are good or bad. Instead, a *reward* is provided from which the agent has to deduce from exploration of different possible actions which of them produce high rewards. The problem is

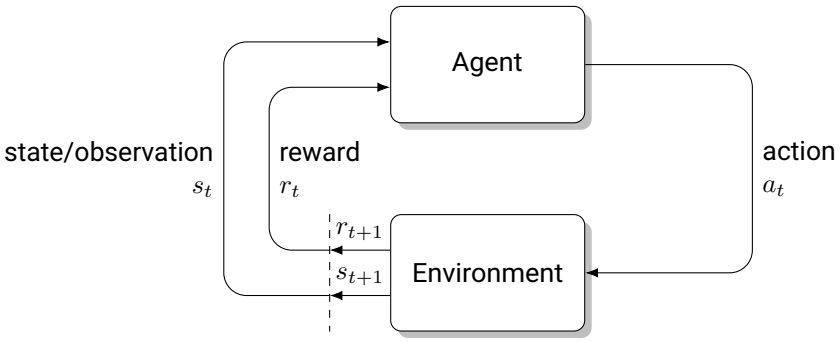



Figure 2.6: In episodic reinforcement learning, the agent repeatedly chooses actions and receives a new observation of the environment along with a reward, from which it has to learn how to act best. Note that at the dashed line the notion of $t+1$ changes to $t$ as the next iteration begins.[7]

commonly depicted as in Figure 2.6. Typically, the environment is modeled in *episodes*, where stating from an *initial state*, multiple actions are to be chosen until a *terminal state* is reached in a finite number of steps. This setting is called *episodic* (as opposed to *continuing*) and is the only one that we will require in this thesis. A reward may be provided at each step as shown in the figure or only at the end. We will also constrain ourselves to the class of model-free algorithms, that do not attempt to learn the environment dynamics. Instead, they directly focus on learning which actions yield which rewards.

This section is based on *Reinforcement learning: An introduction* from Sutton and Barto (2018) [57], where more in-depth information can be found.

Formally, we introduce the notion of *Markov Decision Processes* (MDPs), which serve as a possibly idealized framework of the aforementioned RL setting:

**Definition 21** (MDP)**.** *An MDP $\mathcal{M}$ is defined by a tuple $(S, S_{init}, \{A_s\}_{s\in S}, \mathcal{R}, p)$, where:*

- *$S$ is the set of possible states $s$.*
- *$S_{init}$ is a distribution over initial states $s_0 \in S$ that we can sample from: $s_0 \sim S_{init}$.*
- *$\{A_s\}_{s\in S}$ is a family of possible actions $A_s$ that are allowed in some particular state $s \in S$. If the set of actions is the same for all states, we simply use $A := A_s$ for any state $s$.*
- *$\mathcal{R} \subseteq \mathbb{R}$ is the set of possible rewards. It is often omitted if $\mathcal{R} = \mathbb{R}$.*
- *$p : S \times \mathcal{R} \times S \times A \to [0, 1]$ is the dynamics function, defined as the probability of a certain next state $s'$ and reward $r$ being reached when choosing action $a \in A_s$ in state $s$ at a certain time step $t \in \{0, 1, 2, \dots\}$:*

$$p\left(s', r \,\middle|\, s, a\right) := \mathrm{P}\left(S_{t+1} = s', R_{t+1} = r \,\middle|\, S_t = s, A_t = a\right).$$

The central *Markov property* that is encoded in the dynamics function states that the distribution over the next state $s'$ and reward $r$ is fully determined by the current state $s$ and action $a$. In particular, it cannot depend on past states and must be *fully observable*. Even if those assumptions are not always met in practice, they often form a useful approximate model.

[7] Figure adapted from `https://tex.stackexchange.com/a/461318/140355`, accessed 19 April 2022. It originally stems from Sutton and Barto [57, Fig. 3.1 on p. 54].

The objective in RL is to find a good *policy* $\pi(a \mid s)$, which is a distribution over actions $a$ that should be taken in state $s$. Due to different reasons, one typically also defines a *discount factor* $\gamma \in [0, 1]$, which weights any future rewards less or equal than the the immediate reward (cf. [57, pp. 54f]). Its effect becomes apparent once we properly define the optimization goal, which is to find a policy $\pi$ that maximizes the expected *discounted future return* when following it for entire episodes starting in the state $s_t$:

$$\begin{aligned} G_t &:= r_{t+1} + \gamma\left(r_{t+2} + \gamma\left(r_{t+3} + \dots\right)\right) \\ &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots \\ &= \sum_{k=0}^{\infty} \gamma^k r_{t+1+k}. \end{aligned}$$

Based on the definitions above, we can derive some helpful quantities and define what a best policy would have to fulfill:

**Definition 22** (Auxiliary definitions and Optimality)**.** *Let an MDP $\mathcal{M}$ be defined as in Definition 21 and $\gamma \in [0, 1]$ be the discount factor. Let $\pi$ be any policy. $\mathbb{E}^{\pi}[\,\cdot\,]$ denotes the expected value, where probabilities of actions are as defined by $\pi$ and probabilities of states and rewards as in $\mathcal{M}$. Then:*

- *The* value *of a state $s$ (when continuing according to $\pi$) is defined by the* value function

$$v_\pi(s) := \mathbb{E}^{\pi}\left[G_t \mid s_t = s\right].$$

- *The* action-value function *is defined as*

$$q_\pi(s, a) := \mathbb{E}^{\pi}\left[G_t \mid s_t = s, a_t = a\right].$$

- *An* optimal policy *is defined as*

$$\pi_* := \underset{\text{policy } \pi}{\arg\max} \underset{s \sim S_{init}}{\mathbb{E}}\left[v_\pi(s)\right] = \underset{\text{policy } \pi}{\arg\max} \underset{s \sim S_{init}}{\mathbb{E}^{\pi}}\left[G_0 \mid s_0 = s\right].$$

- *The value and action-value function of any optimal policy $\pi_*$ is denoted by $v_*$ and $q_*$, respectively.*

In fact, it can be shown that even if different optimal policies $\pi_*^1, \pi_*^2, \dots$ exist, they all share the same value and action-value functions [57, pp. 62f], that is

$$\begin{aligned} v_{\pi_*^1} = v_{\pi_*^2} = \dots = v_*, \text{ and} \\ q_{\pi_*^1} = q_{\pi_*^2} = \dots = q_*. \end{aligned}$$

This justifies the notation suggesting unique functions and shows that it is rather a property of the process $\mathcal{M}$ than that of an (optimal) policy. This idea is built upon in the following section.

**Algorithm:** $Q$-learning

**Input** : An MDP with discount factor $\gamma \in [0, 1]$
**Hyperparameters** : A small greedy-factor $\varepsilon \in (0, 1]$ (or a suitable schedule)
A step size $\alpha \in (0, 1]$ (or a suitable schedule)
The number of episodes $N$ to learn from
**Result** : The approximated action-value function $Q(s, a)$

Initialize $Q(s, a)$ as a table filled with zeros
**repeat** $N$ **times**
  Sample $s \leftsquigarrow S_{\text{init}}$
  **while** $s$ *is not terminal* **do**
    `// Choose action` $a$ `from` $Q$ `in` $\varepsilon$`-greedy fashion`
    Sample $x \leftsquigarrow \mathcal{B}(x \mid \varepsilon)$ `// Bernoulli distribution with` $p = \varepsilon$
    **if** $x == 0$ **then**
      $a \leftarrow \arg\max_{a \in A_s} Q(s, a)$
    **else**
      Sample $a \leftsquigarrow \mathcal{U}(a \mid A_s)$ `// Uniform distribution over` $A_s$
    Observe $s'$, $r \leftsquigarrow p(\cdot, \cdot \mid s, a)$ `// Take action` $a$ `in state` $s$
    $Q(s, a) \leftarrow Q(s, a) + \alpha \left[ r + \gamma \max_{a \in A_s} Q(s', a) - Q(s, a) \right]$ `// Perform` $Q$ `update`
    $s \leftarrow s'$
**return** *table* $Q$

**Algorithm 2.2:** The $Q$-learning algorithm for learning an approximate action-value function $Q \approx q_*$. This follows the notation of Sutton and Barto [57, pp. 131f].

### 2.4.1 Q-Learning

$Q$-learning is a popular method in RL due to its effectiveness despite its simplicity. It was also chosen since it had proven to be effective in computationally similar settings, where also a tree of choices was traversed [3, 39]. It is an off-policy algorithm, meaning that it learns a policy from observations of another policy. The motivation is rather straightforward: Suppose the MDP is in state $s \in S$ and there are only a finite number of possible actions $A_s$. If we knew the optimal state-action function $q_*$, we could simply compute the optimal next action by $a = \max_{a \in A_s} q_*(s, a)$. Such a function is also particularly simple to represent if the number of states is finite and small too since in that case, $q_*$ is just a lookup table with $|S|$ rows and $\max_{s \in S} |A_s|$ columns. Otherwise, a function approximator like a neural network can be used to represent and learn the state-action function [37]. Algorithm 2.2 shows the algorithm which iteratively approximates $q_*$ by a tabular $Q$. It was shown back in 1992 that this algorithm indeed converges to the true optimal state-action function $q_*$:

**Theorem 23** (Convergence of $Q$-learning [65])**.** *Let $\mathcal{M}$ be an MDP with discount factor $\gamma \in (0, 1)$ and bounded rewards, i.e. $\exists R_{max} \in \mathbb{R} : \forall r \in \mathcal{R} : |r| \leq R_{max}$. Let $\varepsilon \in (0, 1]$. Further assume that the learning rate $\alpha_n \in [0, 1)$ changes with the index of the current episode $n$ in a way that the infinite series $\sum_{n=0}^{\infty} \alpha_n$ diverges to infinity and*

$\sum_{n=0}^{\infty} \alpha_n^2$ *converges. Let* $Q_n$ *denote the result of running Q-learning with these* $\varepsilon$ *and* $\alpha$ *for* $n$ *episodes. Then*

$$\mathrm{P}\left(\lim_{n\to\infty} Q_n = q_*\right) = 1.$$

In practice, however, the learning rate schedule for $\alpha$ is usually kept constant as in Algorithm 2.2 or is decayed over a finite amount of steps for simplicity.

### 2.4.2 Double Q-Learning

There are numerous extensions to $Q$-learning, of which many fall in the realm of "deep" $Q$-learning, where the $Q$ table is approximated by a neural network [25]. A very simple extension is Double $Q$-learning, where two tables are learned and training is interlocked to avoid overestimation biases sometimes observed in normal $Q$-learning, especially with high discount factors $\gamma$ [24] [57, Sec. 6.7]. To this end, Algorithm 2.2 is changed as follows:

- In the beginning (line 2), two tables $Q_1$ and $Q_2$ are initialized.
- If greedily following the current $Q$ table (line 8), the arg max is computed over $Q_1(s,a) + Q_2(s,a)$ instead of just $Q(s,a)$.
- The update in line 12 is replaced. With probability $0.5$ the following update is performed, and with equal probability the same one where both tables $Q_1$ and $Q_2$ are swapped in every occurrence:

$$Q_1(s,a) \leftarrow Q_1(s,a) + \alpha\left[r + \gamma Q_2\left(s', \operatorname*{arg\,max}_{a' \in A_{s'}} Q_1(s',a')\right) - Q_1(s,a)\right]$$

This algorithm empirically works better in some environments. It also provably converges to $q_*$ as the number of training episodes grows under conditions similar to Theorem 23 [24, Thm. 1].

## 2.5 Related Work

This section provides an overview of work that relates to this thesis by pursuing similar goals. It is threefold: Firstly, alternatives to SPNs as deep *probabilistic* models are presented. Afterward, other means of generating samples are presented and discussed in relation to the goals of this thesis. Finally, prior work on sampling procedures in SPNs is reviewed.

### 2.5.1 Probabilistic Graphical Models

SPNs are structurally closely related to arithmetic circuits (ACs) [12], which also describe a computational graph. But instead of carrying probabilistic interpretations like SPNs do (see Section 2.1), they only compactly model how to compute a polynomial of the leaf values. This stems from the fact that they were initially geared towards allowing for fast approximate inference in Bayesian networks that they were compiled from [34] [43, App. A]. SPNs, however, model a distribution by composing simpler ones into a deep hierarchy while also not requiring the graph to be deterministic/selective, like ACs do. SPNs are therefore strictly more expressive

than ACs [54, Sec. 3] [43, App. A], which comes with certain limitations too. Here, the central one is the lack of tractable exact MPE as discussed in Section 2.1.5, which is closely related to the task of sampling discussed in Section 2.1.6. Lifting the difficulty of that task is one of the main goals of this thesis, in effect allowing the combination of the comparatively strong expressiveness of SPNs [43, 50] with tractable good approximations to formally intractable operations. Expressiveness and traceability of certain queries of these probabilistic graphical models can be seen as two often opposing targets of a spectrum, and exploring this space is an active area of research [62, 63].

Another perspective is the comparison to artificial neural networks: This actively explored and exploited class of models is the main workhorse of current deep learning and machine learning in general [20, Chap. 1]. In their simplest form, feed-forward NNs (also called multilayer perceptrons) describe a computational graph, much like SPNs do. However, they (usually) do not carry any probabilistic semantics, and can therefore incorporate much richer structure and computation nodes. Instead of mere convex sums and products with additional strong limitations on the scope of incoming edges, NN nodes usually compute a non-linear transformation $f$ like traditionally the hyperbolic tangent of an arbitrarily weighted sum with bias $b$, e.g. $f\left(\boldsymbol{w}^T x + b\right)$. Structure learning is not typically done (except for neural architecture search), and instead, gradient-based optimization of the parameters is done. Their outstanding expressiveness stems from their deep and in general often massive structure, which are made practical in both learning and inference by very fast implementations like *Pytorch* [45] being widely available. This idea has since also been applied to SPNs in the form of random sum-product networks (*RAT-SPN*s) [50] and Einsum networks [48].

### 2.5.2 Deep Generative Modeling

There is a multitude of deep generative modeling techniques, of which many achieve impressive results as was shown in Chapter 1. This section shall briefly provide an overview of two of the basic major methods, namely Variational Autoencoders and Generative Adversarial Networks. There are more methods and in particular hybrids and refinements in the literature, like normalizing flows [28], energy-based models, and auto-regressive models, for which the reader is referred to the excellent survey of Bond-Taylor et al. [6] and to *Deep Learning* by Goodfellow, Bengio, and Courville (2016) [20, Chap. 20].

The first and perhaps one of the more widely-known approaches are Variational Autoencoders (VAEs), which are composed of an encoder-decoder architecture like normal autoencoders [27]. The encoder compresses a data instance into a latent lower-dimensional encoding, and the decoder projects such an embedding back to the reconstructed data space. Typically, the encoder and decoder are realized as NNs, but due to their deterministic nature, they are not suitable for generative sampling. VAEs improve on this by introducing special semantics to the encoding: The latent representation is modeled to parameterize a distribution, like an isotropic multivariate Normal distribution or more elaborate techniques [6]. Samples can be generated by sampling from a prior of the latent distribution and decoding it. Often, the output of the decoder is again taken to be a parameterization of a distribution. The samples generated by VAEs are often a little blurry and typically inferior to those generated by other methods like GANs [6], but are conceptually simple and much more stable in training.

Generative Adversarial Networks (GANs), originally presented in 2014 [22], are a game-theoretic and non-probabilistic take at sampling where two functions in the form of NNs are learned [21]: Firstly, a generator $G$ that projects a random latent vector $z$ sampled from a simple distribution to a sample that should mimic the one of the dataset. Secondly, a discriminator $D$ that learns to differentiate between data from the true training data distribution or the output of $G$. They are adversaries, in the sense that $G$ tries to fool $D$ into thinking that fake images are real, to which end $G$ has to learn to produce convincing examples. In the sought-after

equilibrium, the discriminator would have to randomly guess whether it is fake with probability $1/2$. At that point, only the generator is retained. GANs produce very convincing samples and have since been extended in various ways [6, 11]. However, they lack the probabilistic semantics of probabilistic graphical models and there is no straightforward way of recovering that [59].

Finally, it should be noted that samples from SPNs are certainly not expected to be on par with those of VAEs, GANs, or other similar deep generative modeling methods – even if that would of course be a very exciting development. The latter specialize in generative modeling and can only do that (albeit very successfully), while SPNs additionally explicitly model an entire distribution and allow for real probabilistic inference, tractable marginalization, and forms of MPE estimation.

### 2.5.3 Sampling in SPNs

In the first paper introducing SPNs [54] "sample face completions" were provided in Fig. 5 and discussed in the experiments section, but it is not obvious from the paper how these were obtained. Inspection of the source code,[8] however, shows that they were obtained not by sampling in the narrower sense but instead by MAP and MPE (which was thought to be correct for the graphs at hand, see Section 2.1.5). It generally was not a particularly central topic in SPNs: For example, it was not mentioned once in the recent survey by París, Sánchez-Cauce, and Díez [43]. For other types of graphical models, and in particular for Bayesian networks [29, Chap. 12], the matter is studied much better [5, Chap. 11].[9]

Recently, however, there are efforts to use differentiable sampling procedures in SPNs to be able to employ loss-based optimization with back-propagation as is typical in deep learning with NNs. To this end, the selection of a child index when sampling from sum nodes is changed from the discrete and therefore non-differentiable categorical distribution to a differentiable operation using the "Gumbel-Softmax trick" [30, 55]. The first approach uses this formulation to train SPNs with new objectives and does not aim to improve the actual sampling routine, making it an orthogonal direction of research to this thesis. The second approach uses the differentiability to combine neural networks and SPNs into a fully differentiable architecture, permitting training of the entire system jointly. Again, this does not focus on the actual sampling routine, beyond the differentiable formulation. One disadvantage of that approach is that the natural sparsity in the normal sampling procedure as described in the upcoming Section 3.2 is abandoned, and instead the entire SPN is traversed. This is then compensated for by very fast vectorized operations which can even be accelerated by GPUs.

Some of the best samples can currently be obtained from *Einsum Networks*, a vectorized implementation of layered SPN structures mentioned in Section 2.5.1. Figure 2.7 shows images that are sampled conditionally (as reconstruction sampling) and unconditioned on two datasets using that implementation. It is visible that the SPNs that are learned did converge to a somewhat useful representation of the data, but sampling is still perceptibly "blocky" due to the underlying PD structure.

Our experimental setup to generate Figure 2.7 was as follows:

- MNIST (top row): The dataset was the full grayscale $28 \times 28$ MNIST dataset [31] with the usual train and test splits obtained from *Huggingface datasets* [32]. The number of leaves and sums was set to $K = 40$, the leaf distributions were chosen to be factorized Binomial distributions over $n = 16$ possible

[8] In particular, see `code/src/spn/SPN.java` line 137, method `void completeLeftImg(Instance inst)`.

[9] See also the instructive course material *Lecture notes on CS188 Fall 2013. Artificial Intelligence: Lecture 16, Bayes Nets IV, Sampling* [1] and *Lecture notes on CS228 Winter 2021–22. Probabilistic Graphical Models: Sampling-based inference* [13].

(a) MNIST dataset: Instances from the test data split

(b) MNIST dataset: Reconstruction samples of the upper half using the SPN conditioned on the lower half of (a)

(c) MNIST dataset: Unconditional samples from the SPN

(d) SVHN dataset: Test instances as above

(e) SVHN dataset: Reconstructions of the left given the right half of (d)

(f) SVHN dataset: Samples as above

Figure 2.7: Visualization of the generative capabilities of SPNs using the Einsum Networks implementation [48]. The results clearly show the blocky nature stemming from the PD structure which assumes independencies between rectangular regions of the images. This results in separate sampling processes in the children of products and introduces clearly visible inconsistencies that are the major shortcoming that shall be addressed in this thesis.

states (to give sharper images than with $255$ states). $4$ vertical and horizontal splits were performed in the PD structure, corresponding to $\Delta = 8$ in the Einsum Networks paper. The training was performed with online EM for $5$ epochs with batch size $500$ and step size $0.05$.

- SVHN (bottom row): This closely followed the original procedure of Einsum Networks [48, Sec. 4.2] for the $32 \times 32 \times 3$ RGB color SVHN dataset in the cropped digits variant [40]. In summary, leaves were chosen to be isotropic Gaussian distributions with $K = 40$. For the structure, *scikit-learn* $k$-means [46] was performed to find $100$ clusters, for which separate PD structures with $4$ vertical splits were trained each with online EM for $3$ epochs with batch size $10$ and step size $0.5$.

# 3 Methods

As motivated in the introduction in Chapter 1, sampling – the process of drawing instances according to a given probability distribution – is relevant in many different contexts. As lined out in Section 2.5.3, this topic is not particularly well-explored for SPNs. This chapter describes different procedures to perform it on the distribution modeled by an SPN: Firstly, the *standard sampling* procedure from the literature is presented and discussed. However, when applied to typical graph structures like binary trees or PD, that method has weaknesses that are explored subsequently. This leads to the second part, which describes the *guided sampling* approach contributed by this thesis and a discussion of possible alternatives.

## 3.1 Standard Sampling

To the best of our knowledge, the standard sampling procedure for SPNs has not been thoroughly analyzed before, for which reason the full algorithm, as well as a discussion of its numerical stability, the computational complexity, and a proof of its unbiasedness (consistency), are provided. This is important to show, since it sheds light on the reason why sampling appears to produce subpar images as shown, for example, in Figure 2.7. As it will turn out, is not because of the sampling procedure introducing distortions, but instead a problem of the structures that are typically used.

### 3.1.1 The Algorithm

The general sampling routine is very similar to the simple approximate MPE computation given in Algorithm 2.1. While that procedure is given in iterative form using a waitlist, the sampling routine in Algorithm 3.1 is provided recursively to simplify proofs but is trivial to rewrite into an iterative form. The key difference is that sampling occurs on the SPN and not the MPN, where sums have been replaced by maximums. In particular in the typical sampling routine – here coined standard sampling – a child of a sum (corresponding to a mixture component) is sampled according to the weights of the sum node at hand and the probabilities of the evidence $\boldsymbol{e}$, which are combined into new weights $\widetilde{\boldsymbol{w}}$. This procedure again induces a tree, much like when computing MPE on a selective SPN.

Note that due to performance reasons, a single upward pass should be performed before the sampling procedure in order to store the evidence marginal probability $N(\boldsymbol{e})$ for all nodes $N$ of the SPN $\mathcal{S}$. This is similar to the UpwardPassMPN call in the MPE computation with Algorithm 2.1. With this trick, the reweighted responsibilities $\widetilde{\boldsymbol{w}}$ in line 20 can be computed efficiently.

1 **Algorithm:** ConditionallySample: A general framework for conditional sampling in SPNs
**Input** : Some partial (possibly empty) evidence $\boldsymbol{e}$ for RVs $\boldsymbol{E}$
The root $node$ of the SPN to sample from, with $node(\boldsymbol{e}) > 0$
A Guide distribution to choose child indices in unconditioned sum nodes
**Result** : Complete evidence $\boldsymbol{e}'$ for all RVs in $\text{sc}(node)$ obtained from sampling

2 **function** *ConditionallySample(**e**, node,* Guide*)*:
3  **switch** `TypeOf(`*node*`)` **do**
4  **case** *leaf* **do**
5  $\boldsymbol{Y} \leftarrow \text{sc}(node) \setminus \boldsymbol{E}$
6  **if** $\boldsymbol{Y} = \emptyset$ **then**
7  **return** $\boldsymbol{e}$
8  **else**
9  Let $\text{P}_{node}$ be the distribution underlying $node$
10  Sample $\boldsymbol{y} \rightsquigarrow \text{P}_{node}(\boldsymbol{y} \mid \boldsymbol{e})$
11  **return** $\boldsymbol{e} \cup \boldsymbol{y}$
12  **case** *product* **do**
13  **foreach** $C_i \in \text{ch}(node)$ **do**
14  $\boldsymbol{e} \leftarrow \text{ConditionallySample}(\boldsymbol{e}, C_i, \text{Guide})$
15  **return** $\boldsymbol{e}$
16  **case** *sum* **do**
17  **if** $\text{sc}(node) \cap \boldsymbol{E} \neq \emptyset$ **then**
18  Let $\boldsymbol{w}$ be the weights of $node$
19  **foreach** $C_i \in \text{ch}(node)$ **do**
20  $\widetilde{w_i} \leftarrow \frac{w_i C_i(\boldsymbol{e})}{node(\boldsymbol{e})}$
21  Sample $i \rightsquigarrow \text{Cat}(i \mid \widetilde{\boldsymbol{w}})$
22  **else**
23  $i \rightsquigarrow \text{Guide}(node)$ `// Sample index from guidance distribution`
24  **return** $\text{ConditionallySample}(\boldsymbol{e}, \text{ch}(node)[i], \text{Guide})$ `// Select` $i^{\text{th}}$ `child`

25 **function** *StandardGuide(node)*:
26  Let $\boldsymbol{w}$ be the weights of $node$
27  **return** $i \rightsquigarrow \text{Cat}(i \mid \boldsymbol{w})$

**Algorithm 3.1:** Algorithm for sampling from SPNs while optionally using guidance when sampling from sum nodes. The standard guide is provided which selects children of sum nodes proportional to the weight of the edge to them, effectively implementing the typical sampling routine used in the literature (see Section 2.5.3). Instead of sampling such an index $i$ only according to the weights $\boldsymbol{w}$ by $i \sim \text{Cat}(i \mid \boldsymbol{w})$ as in StandardGuide, one could also imagine different guides for other objectives. For this very reason, the case distinction in line 17 is required, as we could else always sample from $i \sim \text{Cat}(i \mid \widetilde{\boldsymbol{w}})$ no matter the evidence marginal probabilities.

Typically, probabilities are passed in log-space because else they might exceed the range of the usual floating point numbers. In that case, the computation of the new weights can also be done numerically stable if the log-likelihood of the children $\log(C_i(\boldsymbol{e}))$ and of the sum node $\log(node(\boldsymbol{e}))$ are given, by

$$\widetilde{w_i} = \frac{w_i C_i(\boldsymbol{e})}{node(\boldsymbol{e})} = \exp\left(\log\left(\frac{w_i C_i(\boldsymbol{e})}{node(\boldsymbol{e})}\right)\right) = \underbrace{\exp\left(\underbrace{\log(w_i)}_{\in(-\infty,0]} + \underbrace{\log(C_i(\boldsymbol{e}))}_{\text{given}} - \underbrace{\log(node(\boldsymbol{e}))}_{\text{given}}\right)}_{\in[0,1]}.$$

While some entries might overflow the floating point range *in log-space* to the symbolic value $-\infty$, they will merely result in $0$ after the final exponentiation. No entries will overflow since the sum of the $\widetilde{\boldsymbol{w}_i}$ is known to be $1$. That suffices to allow sampling $i \sim \mathrm{Cat}\,(i \mid \widetilde{\boldsymbol{w}})$.

The algorithm also demands input evidence $\boldsymbol{e}$ that has positive probability in the SPN, i.e. $root(\boldsymbol{e}) > 0$. Usually, this poses little practical difficulty. Often, at least some leaf nodes have infinite support like Gaussians and therefore always contribute strictly positive probability mass (given a suitable structure). In other cases, the evidence has to be of the right form to fulfill this condition. One should also keep in mind that *conditionally* sampling some value $\boldsymbol{x}$ from a distribution P where the evidence $\boldsymbol{e}$ has probability of measure zero is an ill-posed query in the first place, much like trying to compute $\mathrm{P}\,(\boldsymbol{x} \mid \boldsymbol{e})$ where $\mathrm{P}\,(\boldsymbol{e}) = 0$. Also, if $root(\boldsymbol{e}) > 0$ then in any recursive call of the algorithm the condition is fulfilled too, as will be shown in the proof of Theorem 26. This is required for the algorithm to be well-defined in lines 10 and 20.

One can see that the algorithm is efficient (or tractable), since much like in Algorithm 2.1 each node is visited at most once, in addition to the also linear marginalization upward pass (see Theorem 19):

**Theorem 24** (Time complexity of Algorithm 3.1)**.** *Let an SPN $\mathcal{S}$, partial evidence $\boldsymbol{e}$, and* Guide *of constant runtime be suitable inputs to* ConditionallySample*. Let the worst-case complexity of inference and conditionally sampling form any leaf be $\mathcal{O}(l)$. Let the number of children in a sum node be $m$ and the runtime of* Guide *in that node be in $\mathcal{O}(m)$. Then Algorithm 3.1 runs in $\mathcal{O}(s \cdot l)$, where $s$ is the size of $\mathcal{S}$ (i.e. the number of edges).*

In particular, if $l$ is a constant, Algorithm 3.1 is linear in $s$. The theorem assumes that Guide has runtime in $\mathcal{O}(m)$, where $m$ is effectively the length of $\widetilde{\boldsymbol{w}}$. The StandardGuide fulfills this condition (compare line 27). However, the statement is naturally extendable to different Guide runtimes.

### 3.1.2 Unbiasedness

We now continue by proving that the standard sampling routine is indeed *consistent* with the SPN, i.e. that it samples from the distribution encoded by the SPN without introducing any bias. To this end, we adopt the consistency property commonly used for sampling from Bayesian networks [1] by using the notion of an *induced probability* of a randomized algorithm [19, Sec. 5]:

**Definition 25** (Consistency/Unbiasedness of a sampling routine)**.** *Let $\boldsymbol{X}$ and $\boldsymbol{E}$ be disjoint sets of RVs, $\boldsymbol{e} \in \mathbf{val}\,(\boldsymbol{E})$, and $\mathcal{B}(\boldsymbol{x}, \boldsymbol{e})$ any distribution over $\boldsymbol{X} \uplus \boldsymbol{E}$, where either one of $\boldsymbol{X}$ and $\boldsymbol{E}$ may be empty. Let $\mathcal{A}(\boldsymbol{e})$ be a probabilistic algorithm that computes the remaining evidence $\boldsymbol{x} \in \mathbf{val}\,(\boldsymbol{X})$. We call $\mathcal{A}$* consistent *with $\mathcal{B}$ or* unbiased *iff for the distribution $I_{\mathcal{A}(\boldsymbol{e})}(\boldsymbol{x})$ induced by the randomness of $\mathcal{A}$, it holds that for all $\boldsymbol{x} \in \mathbf{val}\,(\boldsymbol{X})$, $I_{\mathcal{A}(\boldsymbol{e})}(\boldsymbol{x}) = \mathcal{B}\,(\boldsymbol{x} \mid \boldsymbol{e})$, if $\mathcal{B}(\boldsymbol{e}) > 0$.*

The requirement $\mathcal{B}(\boldsymbol{e}) > 0$ reflects the analog requirement on the inputs $\boldsymbol{e}$ and $node$ to Algorithm 3.1.

Now we can show that the algorithm is correct in a probabilistic sense. The condition $\boldsymbol{X} \uplus \boldsymbol{E} \supseteq \text{sc}(\mathcal{S})$ was chosen over $\boldsymbol{X} \uplus \boldsymbol{E} = \text{sc}(\mathcal{S})$ to conveniently allow the theorem to be applied in the inductive proof without requiring any auxiliary theorems. We also acknowledge the fact that Algorithm 3.1 returns $\boldsymbol{e} \cup \boldsymbol{x}$, but for the sake of the proof we only consider $\boldsymbol{x}$ (as required by Definition 25) since the original $\boldsymbol{e}$ remains untouched in each step. Note, that the SPN is assumed to be complete and decomposable, not just valid and also not just complete and consistent, which are both weaker properties (see also Figure 2.4). In practice, however, completeness and decomposability are the typical means of showing validity anyway, so the practical applicability of the theorem is not significantly constrained. Therefore, no attempt at lifting the restriction was undertaken.

**Theorem 26** (Consistency of standard sampling)**.** *Let $\mathcal{S}$ be a complete and decomposable SPN. Let $\boldsymbol{X}$ and $\boldsymbol{E}$ be disjoint sets of RVs and $\boldsymbol{e} \in \textbf{val}\,(\boldsymbol{E})$ (incomplete) evidence with $\mathcal{S}(\boldsymbol{e}) > 0$. Either one of $\boldsymbol{X}$ and $\boldsymbol{E}$ may be empty, but $\boldsymbol{X} \subseteq \text{sc}(\mathcal{S})$ and $\boldsymbol{X} \uplus \boldsymbol{E} \supseteq \text{sc}(\mathcal{S})$. Then* $\text{ConditionallySample}\,(\boldsymbol{e}, \mathcal{S}, \text{StandardGuide})$ *is consistent with $\mathcal{S}$.*

*Proof.* Let $\mathcal{S}$, $\boldsymbol{X}$, $\boldsymbol{E}$, and $\boldsymbol{e}$ be defined as above.

To ensure that the algorithm is well-defined in the recursive calls, we first note that if $\mathcal{S}(\boldsymbol{e}) > 0$, then in any occurring recursive call the evidence $\boldsymbol{e}$ has positive probability too. This can be shown by a simple case distinction as follows: If $\mathcal{S}$ is a leaf, no recursive call will occur and there is nothing to show. If we are in a product node, then $\mathcal{S}(\boldsymbol{e}) = \prod_{C \in \text{ch}(\mathcal{S})} C(\boldsymbol{e})$. In that case, $\mathcal{S}(\boldsymbol{e}) > 0$ can only happen if for all children $C$ it holds that $C(\boldsymbol{e}) > 0$, since we know that $C(\boldsymbol{e}) \geq 0$ and $C(\boldsymbol{e}) = 0$ would imply $\mathcal{S}(\boldsymbol{e}) = 0$. Similarly, if $\mathcal{S}$ is a sum node, then at least one of the children with positive weights must have propagated positive probability mass upward. In turn, one such child will be selected in line 21. We can therefore assume without loss of generality that $N(\boldsymbol{e}) > 0$ for any root node $N$ that the algorithm is recursively called with.

Let $I_{\boldsymbol{e}}(\boldsymbol{x}) := I_{\text{ConditionallySample}(\boldsymbol{e}, \mathcal{S}, \text{StandardGuide})}(\boldsymbol{x})$ be the induced distribution and $\boldsymbol{x} \in \textbf{val}\,(\boldsymbol{X})$ arbitrary but fixed. We show $I_{\boldsymbol{e}}(\boldsymbol{x}) = \mathcal{S}\,(\boldsymbol{x} \mid \boldsymbol{e})$ by structural induction on the depth of $\mathcal{S}$. The depth $d(\mathcal{S})$ of the DAG $\mathcal{S}$ is zero iff it is a leaf (see Definition 15), yielding that case as the only base case that needs to be considered:

**Base case** ($d(\mathcal{S}) = 0$): In this case the algorithm will jump to line 5. Let $\boldsymbol{Y} = \boldsymbol{X} \setminus \boldsymbol{E}$, i.e. the set of variables without evidence. If $\boldsymbol{Y} = \emptyset$, the algorithm will just deterministically return $\boldsymbol{e}$ in line 7, i.e. with probability

$$\mathcal{S}\,(\boldsymbol{y} \mid \boldsymbol{e}) = \frac{\mathcal{S}(\boldsymbol{y}, \boldsymbol{e})}{\mathcal{S}(\boldsymbol{e})} \overset{(\boldsymbol{Y} = \emptyset)}{=} \frac{\mathcal{S}(\boldsymbol{e})}{\mathcal{S}(\boldsymbol{e})} = 1.$$

Else, i.e. if $\boldsymbol{Y} \neq \emptyset$, let P be the distribution underlying $\mathcal{S}$. Then the result is $\boldsymbol{e}$ complemented with a sample $\boldsymbol{y} \sim \text{P}\,(\boldsymbol{y} \mid \boldsymbol{e})$, which also follows the distribution defined by $\mathcal{S}$: $\text{P}\,(\boldsymbol{y} \mid \boldsymbol{e}) = \mathcal{S}\,(\boldsymbol{y} \mid \boldsymbol{e})$.

**Induction step** ($d(\mathcal{S}) \rightsquigarrow d(\mathcal{S}) + 1$): Assume $I_{\boldsymbol{e}}(\boldsymbol{x}) = \mathcal{S}'\,(\boldsymbol{x} \mid \boldsymbol{e})$ holds for any SPN $\mathcal{S}'$ of depth $d(\mathcal{S}') \leq n$ (induction hypothesis, "IH"). Let $\mathcal{S}$ be an SPN of depth $d(\mathcal{S}) = n + 1$. We need to show $I_{\boldsymbol{e}}(\boldsymbol{x}) = \mathcal{S}\,(\boldsymbol{x} \mid \boldsymbol{e})$ for the two possible types of nodes $\mathcal{S}$:

1. $\mathcal{S}$ **is a product node**: Then the algorithm jumps to line 13. Since $\mathcal{S}$ is decomposable, all $K$ children $C_i$ for $i \in \{1, \ldots, K\}$ have disjoint scopes and we can define pairwise disjoint $\boldsymbol{Y}_i := (\text{sc}(C_i) \setminus \boldsymbol{E}) \subseteq \boldsymbol{X}$. Intuitively, these are the RVs that need to be sampled in that child node and it is simple to see that for any child $C_i$ with $i \in \{1, \ldots, K\}$:

a) $\boldsymbol{Y}_i \subseteq \text{sc}(C_i)$,

b) $\boldsymbol{Y}_i \uplus \boldsymbol{E} \supseteq \text{sc}(C_i)$,

c) not both $\boldsymbol{Y}_i$ and $\boldsymbol{E}$ can be empty since else $\text{sc}(C_i) = \emptyset$,

d) for any "surplus" evidence $\boldsymbol{Y}' \subseteq \biguplus_{j=1, j\neq i}^{K} \boldsymbol{Y}_j$ it holds that $C_i\left(\boldsymbol{y}_i \mid \boldsymbol{e}, \boldsymbol{y}'\right) = C_i\left(\boldsymbol{y}_i \mid \boldsymbol{e}\right)$, since it is simply ignored by the evaluation of inner nodes and leaves (see Proposition 6) as it is by ConditionallySample, and

e) due to the definition of scopes (Definition 4), the $\boldsymbol{Y}_j$ are a partition of $\boldsymbol{X}$, i.e. $\boldsymbol{X} = \biguplus_{j=1}^{K} \boldsymbol{Y}_j$.

Properties (a)–(c) allow us to apply the IH. The iterative completion in lines 13 to 15 effectively results in a product of child queries, where each call to ConditionallySample is conditioned on the sampling result of the previous calls. However, by (d) and effectively by the decomposability of the product, these calls are actually sampling from independent distributions and by (e) return a part $\boldsymbol{y}_i \subseteq \boldsymbol{x}$, so we can show that

$$
\begin{aligned}
I_{\boldsymbol{e}}(\boldsymbol{x}) &\overset{\text{(lines 13–15, (e))}}{=} \prod_{i=1}^{K} I_{\text{ConditionallySample}\left(\boldsymbol{e} \uplus \biguplus_{j=1}^{i-1} \boldsymbol{y}_j, C_i, \text{StandardGuide}\right)}(\boldsymbol{y}_i) \\
&\overset{\text{(IH)}}{=} \prod_{i=1}^{K} C_i\left(\boldsymbol{y}_i \,\middle|\, \boldsymbol{e} \uplus \biguplus_{j=1}^{i-1} \boldsymbol{y}_j\right) = \prod_{i=1}^{K} C_i\left(\boldsymbol{y}_i \,\middle|\, \boldsymbol{e}, \biguplus_{j=1}^{i-1} \boldsymbol{y}_j\right) \\
&\overset{\text{(d)}}{=} \prod_{i=1}^{K} C_i\left(\boldsymbol{y}_i \mid \boldsymbol{e}\right) \\
&= \prod_{i=1}^{K} \frac{C_i\left(\boldsymbol{y}_i, \boldsymbol{e}\right)}{C_i(\boldsymbol{e})} = \frac{\prod_{i=1}^{K} C_i\left(\boldsymbol{y}_i, \boldsymbol{e}\right)}{\prod_{i=1}^{K} C_i(\boldsymbol{e})} \\
&\overset{\text{(Definition 15 \& Proposition 6)}}{=} \frac{\mathcal{S}\left(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_K, \boldsymbol{e}\right)}{\mathcal{S}(\boldsymbol{e})} \\
&\overset{\text{(e)}}{=} \frac{\mathcal{S}\left(\boldsymbol{x}, \boldsymbol{e}\right)}{\mathcal{S}(\boldsymbol{e})} = \mathcal{S}\left(\boldsymbol{x} \mid \boldsymbol{e}\right).
\end{aligned}
$$

2. $\mathcal{S}$ **is a sum node**: Then the algorithm jumps to line 17. Let $\boldsymbol{w}$ be the weights of the sum node $\mathcal{S}$ which has $K$ children. For any such sum node, it holds due to the definition of SPNs that

$$
\begin{aligned}
\mathcal{S}\left(\boldsymbol{x} \mid \boldsymbol{e}\right) &= \frac{\mathcal{S}(\boldsymbol{x}, \boldsymbol{e})}{\mathcal{S}(\boldsymbol{e})} = \frac{1}{\mathcal{S}(\boldsymbol{e})} \sum_{i=1}^{K} w_i C_i(\boldsymbol{x}, \boldsymbol{e}) = \frac{1}{\mathcal{S}(\boldsymbol{e})} \sum_{i=1}^{K} w_i C_i(\boldsymbol{e}) C_i\left(\boldsymbol{x} \mid \boldsymbol{e}\right) \\
&= \sum_{i=1}^{K} \frac{w_i C_i(\boldsymbol{e})}{\mathcal{S}(\boldsymbol{e})} C_i\left(\boldsymbol{x} \mid \boldsymbol{e}\right) = \sum_{i=1}^{K} \widetilde{w_i} C_i\left(\boldsymbol{x} \mid \boldsymbol{e}\right), \quad \text{where} \qquad (3.2) \\
\widetilde{w_i} &:= \frac{w_i C_i(\boldsymbol{e})}{\mathcal{S}(\boldsymbol{e})}. \qquad (3.3)
\end{aligned}
$$

Note that since $\mathcal{S}(\boldsymbol{e}) = \sum_{i=1}^{K} w_i C_i(\boldsymbol{e})$, we have $\sum_{i=1}^{K} \widetilde{w_i} = 1$ and the vector $\widetilde{\boldsymbol{w}}$ therefore forms a valid parameter for a categorical distribution. Considering the case where $\text{sc}(\mathcal{S}) \cap \boldsymbol{E} \neq \emptyset$ we can show the correctness of lines 21 and 24 by

$$I_{\boldsymbol{e}}(\boldsymbol{x}) = \sum_{i=1}^{K} \widetilde{w_i} I_{\text{ConditionallySample}(\boldsymbol{e}, C_i, \text{StandardGuide})}(\boldsymbol{x}) \overset{\text{(IH)}}{=} \sum_{i=1}^{K} \widetilde{w_i} C_i\left(\boldsymbol{x} \mid \boldsymbol{e}\right) \overset{(3.2)}{=} \mathcal{S}\left(\boldsymbol{x} \mid \boldsymbol{e}\right). \tag{3.4}$$

Since $\mathcal{S}$ is complete all children $C_i \in \text{ch}(\mathcal{S})$ have identical scope $\text{sc}(C_i) = \text{sc}(\mathcal{S})$, so we were allowed to use the IH in the derivation above.

If on the other hand $\text{sc}(\mathcal{S}) \cap \boldsymbol{E} = \emptyset$, we land in lines 23 and 24. However, in that case for any $C_i$ with $i \in \{1, \ldots, K\}$ we know that $C_i(\boldsymbol{e}) = 1$ and therefore also $\mathcal{S}(\boldsymbol{e}) = \sum_{i=1}^{K} w_i C_i(\boldsymbol{e}) = 1$, since all variables in the identical scope of the nodes are marginalized out. Thus, StandardGuide samples an index from exactly the same distribution as the line 21 and is thereby correct too, since for every of the $i \in \{1, \ldots, K\}$ outcomes

$$\widetilde{w_i} \overset{(3.3)}{=} \frac{w_i C_i(\boldsymbol{e})}{\mathcal{S}(\boldsymbol{e})} = \frac{w_i \cdot 1}{1} = w_i. \tag{3.5}$$

This concludes the proof by structural induction. □

## 3.2 Guided Sampling

In some sense, one might think that Algorithm 3.1 is the perfect algorithm: We have a relatively simple-to-implement sampling routine that is both tractable and consistent/unbiased. However, when we look at the samples in Section 2.5.3 (see Figure 2.7), we can see that it still leaves a lot to be desired. In particular, one can clearly see that the independence assumptions of product nodes lead to chunky images with inconsistent patches. This leads to the realization that we actually want to find a Guide such that $\text{ConditionallySample}\left(\boldsymbol{e}, \mathcal{S}, \text{Guide}\right)$ is consistent with the data distribution $p_d$ used to estimate the SPN $\mathcal{S}$, instead of with $\mathcal{S}$ itself, since in practice there is some modeling error and therefore $p_d \neq \mathcal{S}$. A large contributor to this discrepancy is the heuristic nature of the structure generation that we require for scaling to large datasets, like the PD and binary tree structures. In more formal terms, a possible problem statement would be that we seek a Guide such that for the induced sampling distribution $I$

$$D_{KL}(p_d \mid\mid I) \leq D_{KL}(p_d \mid\mid \mathcal{S}). \tag{3.6}$$

Unfortunately, it is very difficult to derive a concrete improved algorithm from this expression. Instead, it helps to clarify the root problem of standard sampling on the discussed structures and derive an improvement from there.

In product nodes, one needs to consider every single child since each one governs a separate part of the instance, and we do not want patches of – say images – to be missing from the samples. In sum nodes, however, we can make a decision, and there are good and bad choices for children to continue with. The outcome of that decision decides whether samples are self-consistent or not, and can lead to the bad examples shown in Figure 2.7 when done poorly by simply following Algorithm 3.1. The issue and afterward the proposed solution shall be illustrated by the following Figure 3.1, showing a simple SPN as a distribution over two types of digit images.

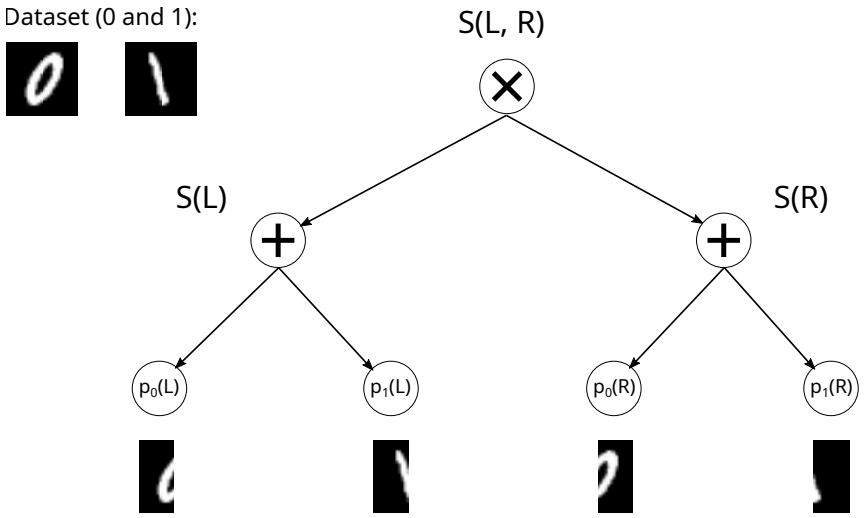
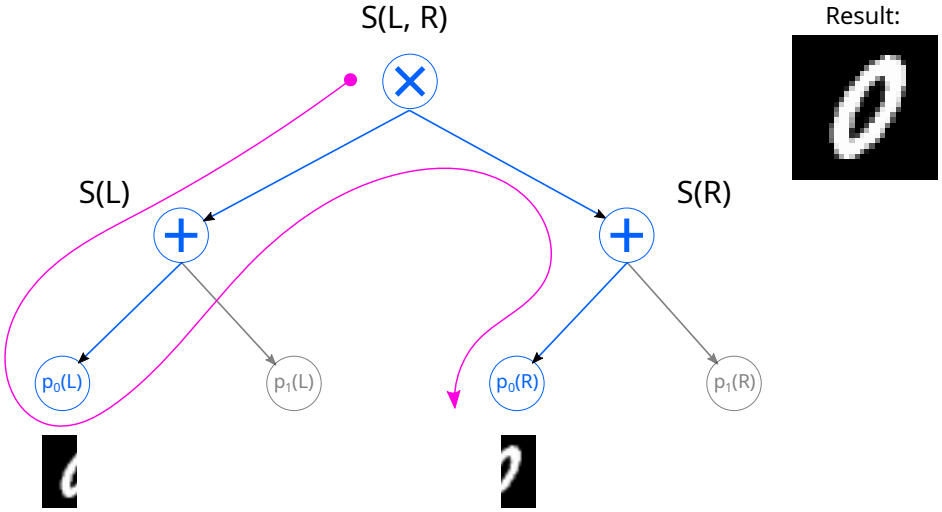


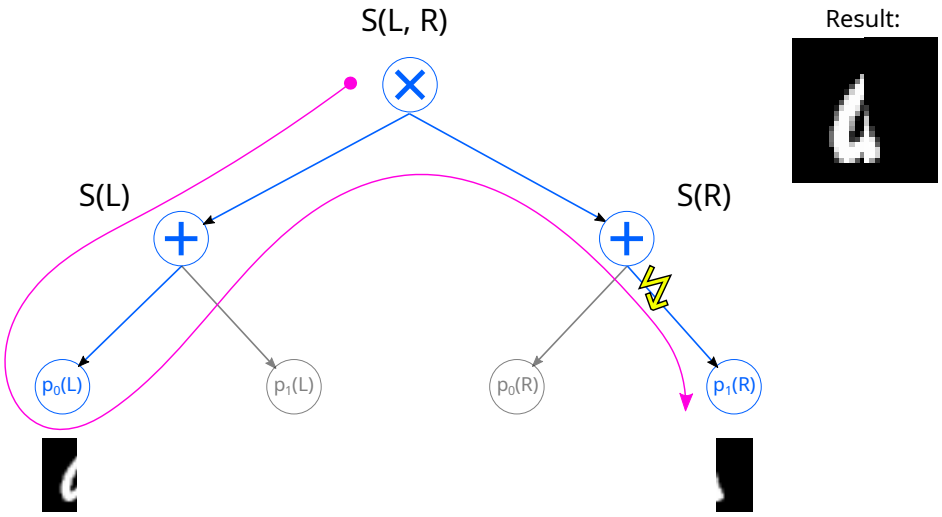

Figure 3.1: An illustrative SPN over a dataset consisting of digits $0$ and $1$ as shown in the upper left corner (copied from the MNIST dataset [31]). The model boldly assumes that the left- and right-hand sides of an image are independent: $S(L, R) = S(L) \times S(R)$. In each of the two children, a sum over two leaves each model image halves showing a $0$ and a $1$, respectively. Samples of those partial images are shown below the leaf nodes. We ignore the weights in this visualization, let us just assume that they are all equally set to $w = 1/2$.

When we now sample from this SPN using the StandardGuide in Algorithm 3.1, we perform a depth-first search of the DAG (with special handling of sum nodes). If we perform a left-to-right traversal in inner nodes with multiple children, we first descend into $S(L)$. It should be noted that the order of the traversal is arbitrary when using the StandardGuide and that we just fix it to left-to-right for illustrational purposes. If there is evidence about the left hand side, we sample $i \sim \text{Cat}\,(i \mid \widetilde{\boldsymbol{w}})$ (line 20), or else we choose $p_0(L)$ or $p_1(L)$ with probability $w = 1/2$ each as defined by StandardGuide. Next, we need to sample the right-hand side of the image using $S(R)$. Assume that this time, no evidence is given about that variable and we need to sample. That image patch is now chosen independently of the fixed and known left-hand side, and there is a $w = 1/2$ chance that we sample from the cluster matching the one from $S(L)$, and a $w = 1/2$ chance to choose one that results in the two halves showing parts of different digits. Those two possible outcomes are shown in Figure 3.2.

If the child that was chosen in $S(L)$ was known, one could have made a more informed decision in $S(R)$ on the right and produced a more consistent image. It, therefore, seems promising to consider different Guides that given the choices so far choose upcoming sum node children representing clusters that are consistent with the partial instance so far. To this end, we simply have to remember the current path through the SPN by remembering the indices of children chosen so far, either by conditioning on evidence or by sampling a child using the Guide. This is a simple extension to Algorithm 3.1, where in addition to tracking the current path the Guide is also given it as an argument to incorporate that information. Note, however, that this also means that only information on parts of the SPN that were already sampled can be incorporated. Lifting this restriction is deferred to future work, but its effects are illustrated and discussed in Section 4.1 and Appendix A.1.




(a) A reasonable sample of a $0$ digit sampled from $p_0(L) \times p_0(R)$.




(b) An inconsistent sample showing parts of both a $0$ and a $1$ sampled from $p_0(L) \times p_1(R)$.

Figure 3.2: Visualization of how sampling in the SPN of Figure 3.1 can produce (a) consistent and (b) inconsistent samples. The part of the SPN that is visited is shown in blue while the rest is grayed out. The pink path shows the depth-first left-to-right graph traversal of the sampling algorithm. The resulting partial images are shown at the bottom. It is apparent that the inconsistency in the second traversal arises from choosing the wrong child in $S(R)$, which is marked with a $\lightning$.

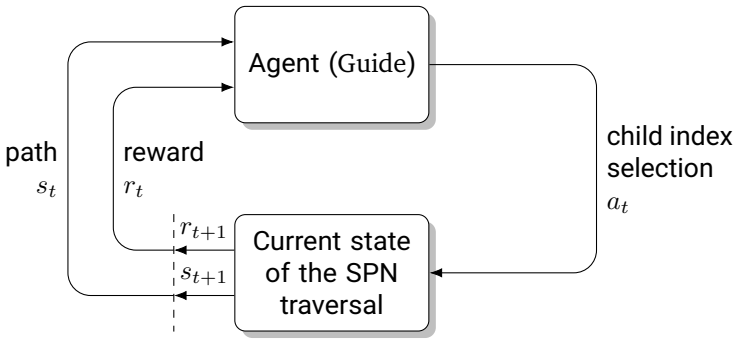



Figure 3.3: This figure shows how the guided sampling algorithm can be viewed as an RL problem. The diagram is analogous to Figure 2.6 from the foundations chapter. The environment is the graph traversal process, which receives a sum child index $a_t$ and then continues traversal to the next sum node over possibly multiple leaves, products, and conditioned sum nodes. Eventually, it provides the old and newly visited sum node indices path as a state $s_{t+1}$ and a reward $r_{t+1}$ to continue the episode at the next unconditioned sum node.

## 3.3 Reinforcement Learning Setting

A natural formulation of the problem of predicting a new sum node child based on the current partial path is reinforcement learning as introduced in Section 2.4. Here, the states are partial paths and actions are sum node children to be chosen. The children are identified by their index, and we, therefore, choose an arbitrary but fixed ordering of the children of all inner nodes including products. The environment is provided by Algorithm 3.1 and the agent is the Guide. Possible reward signals were discussed in Section 2.2, and the one we will use is the reconstruction error MSE. The construction is visualized and explained by Figure 3.3.

More formally, we can define this setting for a specific SPN $\mathcal{S}$ as an MDP $\mathcal{M} = (S, S_{\text{init}}, \{A_s\}_{s \in S}, \mathcal{R}, p)$ following Definition 21:

- The set of states $S$ is the set of all partial and complete paths, i.e. all possible sequences of sum node indices that can be visited by a traversal in the style of Algorithm 3.1. A possible sequence of states when traversing the exemplary SPN from the previous section (Figure 3.1) is shown in Figure 3.4.

- There is a single initial state $s_0 = [\,]$ (the empty list).

- In a non-terminal state $s$, there is a sum node $N$ where the next child index needs to be determined by the actor. Let $K$ be the number of children of $N$. Then the set of actions in $s$ is just the set of all possible indices $A_s = \{0, \ldots, K-1\}$.

- The set of possible rewards is defined by the concrete reward that is used. In the case of negative reconstruction errors, $\mathcal{R} \subseteq (-\infty, 0]$.

- The dynamics function $p : S \times \mathcal{R} \times S \times A \to \{0, 1\}$ is deterministic in the sense that any child index that is chosen will certainly be visited next (see the discussion of the Markov property below). Let $s$ be a non-terminal state and $N$ be the next sum node in the traversal order. Let $K$ be the number of children of $N$ and let $\text{ch}(N)[i]$ denote the $i^{\text{th}}$ zero-indexed child of $N$. Let $R(r)$ be the distribution of

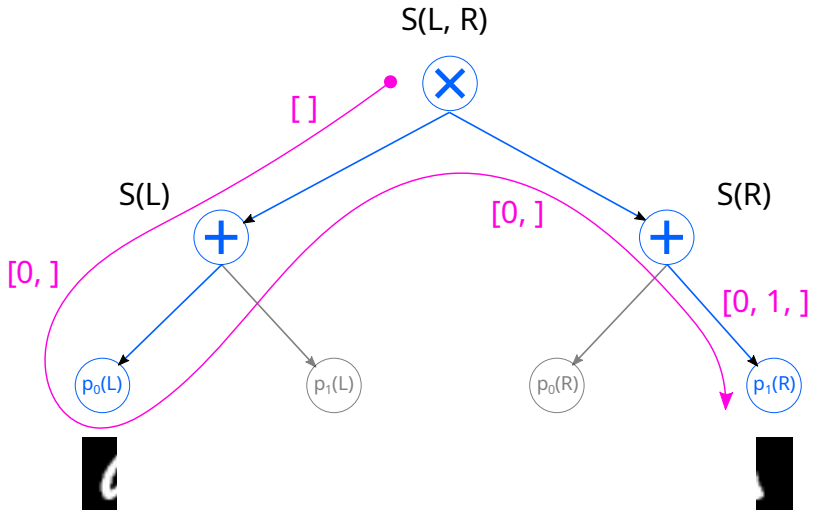



Figure 3.4: Illustration of the (partial) paths that occur when sampling as in Figure 3.2b. Note that the path starts out as an empty list $[\,]$ and each time a sum node is traversed, the index of the chosen child node is appended (giving $[0,\,]$ and ultimately $[0, 1,\,]$). This means that in product nodes, the starting path of a child is the ending path of the preceding product node's child (here $[0,\,]$).

rewards $r$ in that node. Then the dynamics function is given by:

$$p\left(s', r \,\middle|\, s, a\right) := \begin{cases} R(r) & \text{if } a < K \wedge s' = \mathrm{ch}(N)[a] \\ 0 & \text{else} \end{cases}.$$

This setting is loosely inspired by "Neural Architecture Search of Deep Priors: Towards Continual Learning Without Catastrophic Interference" (2021) by Mundt, Pliushch, and Ramesh [39], where similar path states were used to learn good NN architectures for a given dataset using tabular $Q$-learning. Similarly, we only care about the reward at the end of the episode, i.e. the negative of the reconstruction error when the entire image is sampled. We therefore want the reward $r_t$ in any terminal state $s_t$ to equal the discounted future return of all previous nodes, i.e. $G_{t'} = r_t$ for all $t' \in \{0, \ldots, t\}$. In other words, we do not want to discount rewards in any way, since we do not want the actor to implicitly favor shallower traversals of the SPN above deeper ones. It therefore makes sense to set the discount factor to $\gamma = 1$.[10] This decision is still reasonable if we want to split the reward across multiple actions, where the negative reconstruction loss is provided only for newly sampled parts of the instance and $G_t$ eventually adds up to the full reconstruction error or similar metric.

It is also worth questioning whether the Markov property holds for $\mathcal{M}$, i.e. whether the current state and action fully determine the next state and the reward. Viewing the dynamics function as a proper probability (see also [57, pp. 48f]), the marginal *state-transition probability* $p\left(s' \,\middle|\, s, a\right)$ is 1 if $s'$ is the child at index $a$ and else 0. So the only question is whether the distribution over rewards is also fully defined by the current path. This is however clearly the case since the path simply selects the leaf nodes that the RV slices are sampled from, and the set of those leaves then implicitly defines the reward, independent of the partial intermediate paths by which the leaves have been selected. The same argument holds for rewards in non-terminal states.

[10] As was done by Mundt, Pliushch, and Ramesh [39], see `https://github.com/ccc-frankfurt/DP-NAS/blob/main/lib/cmdparser.py#L72` at commit `b8a2fd9463d670da1090b631fcfaf0a16b96ceda`.

**Algorithm:** Guided sampling using $Q$-Learning

**Input** : The sum $node$ to choose a child index in
The current state $s$
A suitable $Q$ table

**Result** : The index of the child of the sum node to continue in
The new state $s'$

`// The current state` $s$ `is represented as the list` $[i_1, \ldots, i_{|s|}]$ `of sum child indices visited so far`

**function** *QLearningGuide*($node$, $s$, $Q$)**:**
  Let $K$ be the number of children in $node$
  $i_{|s|+1} \leftarrow \arg\max_{a \in \{1,\ldots,K\}} Q(s, a)$ `// Find the action` $a$ `with maximum expected reward`
  $s' \leftarrow s \parallel i_{|s|+1}$ `// Concatenate to form the new state`
  **return** $i_{|s|+1}$, $s'$

**Algorithm 3.2:** The algorithm for providing a sampling guide for Algorithm 3.1 based on a learned $Q$ table for the MDP $\mathcal{M}$ for a specific SPN.

Finally, we are able to learn an optimal agent for $\mathcal{M}$ using either $Q$-learning or its double variant described in Sections 2.4.1 and 2.4.2, respectively. To this end, we simply couple the traversal of Algorithm 3.1 and the relevant learning algorithm like Algorithm 2.2 using a suitable Guide. That QLearningGuide is given in Algorithm 3.2. It simply selects the most promising node index. In practice, however, one might add a random tie-break for states of equal $Q$ value to not introduce an exploration bias. Also note that the choice of setting $\gamma = 1$ formally makes the convergence guarantees of both normal and double $Q$-learning (e.g. Theorem 23) inapplicable, since $\gamma \notin (0, 1)$. Nonetheless, in practice and in particular for these episodic environments it still works well.

## 3.4 Relation of Standard and Guided Sampling

Is is also noteworthy that the approach of obtaining a $Q$ table for decision making in sum nodes does not subsume standard sampling, it formally is something different. While on one hand the standard guide from Algorithm 3.1 *samples* $i \sim \text{Cat}(i \mid \boldsymbol{w})$, the guided learning approach from Algorithm 3.2 *deterministically* selects $i = \arg\max_{a \in \{1,\ldots,K\}} Q(s, a)$. If we changed line 4 to instead sample $i \sim \text{Cat}(i \mid Q(s, 1), \ldots, Q(s, K))$, we obtain the modified QLearningGuide$^*$ and can show:

**Theorem 27.** *For any SPN $\mathcal{S}$ there exists a (tabular) function $q_{\mathcal{S}}$ such that for any $\boldsymbol{e}$ and $\boldsymbol{x}$ (with the restrictions as in Theorem 26)*

$$I_{\text{ConditionallySample}(\boldsymbol{e},\, \mathcal{S},\, \text{StandardGuide})}(\boldsymbol{x}) = I_{\text{ConditionallySample}(\boldsymbol{e},\, \mathcal{S},\, \text{QLearningGuide}^*(\cdot,\cdot,\, q_{\mathcal{S}}))}(\boldsymbol{x}). \tag{3.7}$$

*Proof.* We first state $q_{\mathcal{S}}$ explicitly and then show that the above statements holds. Each state $s$ is either terminal or encodes a path to exactly one sum node $N$ as a surjection (i.e. there can be more paths $s'$ resulting in

the same sum node $N$). If it is terminal, we arbitrarily choose $q_{\mathcal{S}}(s, \cdot) = 0$ as it is not used by Algorithm 3.1 anyway. If it is a sum node, let $q_{\mathcal{S}}(s, i) := w_i$ for all $i \in \{1, \ldots, K\}$, where $K = |A_s|$ is the number of children of $N$ or equally the number of possible actions in $s$.

The proof is then formally again by induction on the depth of $\mathcal{S}$ (like in the proof of Theorem 26). However, it suffices to only inspect what happens when the Guide is used in line 23 of Algorithm 3.1. This is because the algorithms are equal in all other regards except for the bookkeeping of the states $s$, which however only has an influence when the Guide is used. In the case of using the Guide, by construction, both StandardGuide and QLearningGuide$^*$ sample from exactly the same distribution. Thereby the above statement (3.7) holds. □

While such a $q_{\mathcal{S}}$ is trivial to compute from an SPN $\mathcal{S}$, it is not a real state-action value function since a $Q$-learning agent follows the maximum action deterministically and does not sample from it. Therefore, one cannot just view standard sampling and the tabular $Q$ guide as parameters of the same induced distribution – they instead do encode different but closely related ones. Also, note that actually sampling from $q_{\mathcal{S}}$ would be pointless since it encodes the same sampling distribution as when applying the simpler StandardGuide.

A possible future direction of a formal study of these procedures might try to formulate them as parameters of the same distribution. This might then be used to reason about how $Q$-learning performs maximum likelihood estimation (MLE) of some sort. The MLE solution of a problem is in turn closely related to the KL divergence[11] and might be useful for showing whether equation (3.6) provably holds. This line of reasoning appears to be much more promising than the pursuit of bounding the error of the related task of approximate MPE solvers, where Mei, Jiang, and Tu conclude: “We also showed that it is almost impossible to find a practical bound for approximate MAP solvers” [36].

[11] Loosely speaking, the MLE using $n$ data points finds parameters $\theta_{\text{MLE}}$ of a model distribution $m$ that minimize the distance to the data distribution $p_d$ in the limit of infinite amounts of i.i.d. data: $\theta_{\text{MLE}} = \lim_{n\to\infty} \arg\min_\theta D_{KL}(p_d(x) \,||\, m(x \,|\, \theta))$. This result is well presented in the lecture material at `https://web.stanford.edu/class/stats200/Lecture16.pdf`, which in turn references the relevant publications.

# 4 Experiments

In this chapter, we investigate how the methods of the previous chapter perform empirically. The main question that shall be answered is: **In which cases can guided sampling using $Q$-learning improve the sample quality beyond standard sampling and by how much?** To this end, we first construct a simple task from synthetic bivariate data and inspect how the methods perform. For a subsequent evaluation on larger datasets, we first describe how the underlying SPNs were trained, the data and preprocessing steps that were used, as well as the actual exploration of the different sampling procedures. The chapter closes with a qualitative analysis of a few select settings and an exploration of different variations of sampling and MPE.

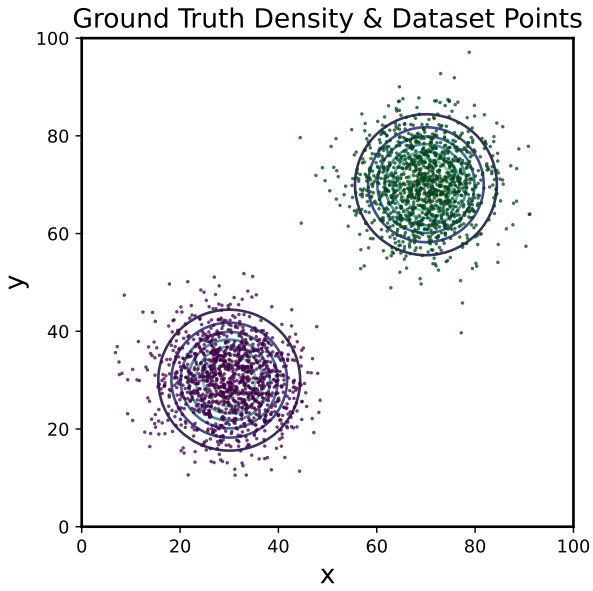
The implementation was mostly developed in Python [60] (targeting versions 3.8–3.10) and makes heavy use of the general scientific libraries *NumPy* [23], *SciPy* [64], *pandas* [35] and *Pytorch* [45]. *SPFlow* (pre-release version) [38] was used as the base SPN data model and structure generation and associated parameter learning was mainly performed using *EinsumNetworks* [48]. Due to numerical issues, the RAT-SPN implementation of SPFlow and its `optimize_tf()` method were not used successfully. The datasets were obtained from and processed using *Huggingface datasets* [32].

## 4.1 Proof of Concept on Synthetic Data

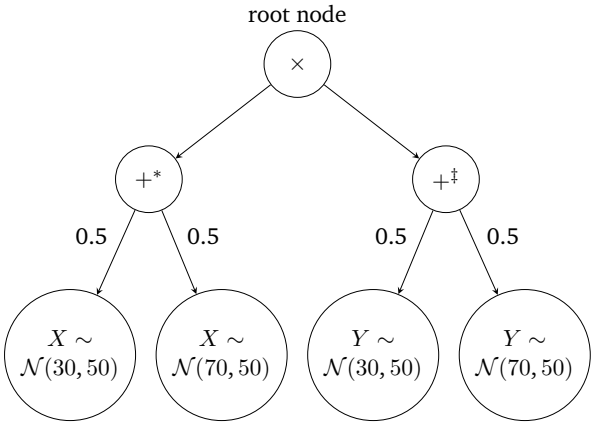
As a first experiment, it is useful to inspect a very simple scenario where the full data distribution is properly visualizable in 2D. To this end, let us consider a simple data distribution $p_d$ over two variables $X$ and $Y$. The data is defined as a mixture of two Gaussians $\mathcal{N}_0$ and $\mathcal{N}_1$ with mixture coefficients $0.5$ each:

$$\mathcal{N}_0\left(\begin{pmatrix}30\\30\end{pmatrix}, \mathbf{\Sigma}\right) \quad \text{and} \quad \mathcal{N}_1\left(\begin{pmatrix}70\\70\end{pmatrix}, \mathbf{\Sigma}\right),$$
$$\text{where} \quad \mathbf{\Sigma} := \begin{pmatrix}50 & 0\\0 & 50\end{pmatrix}.$$

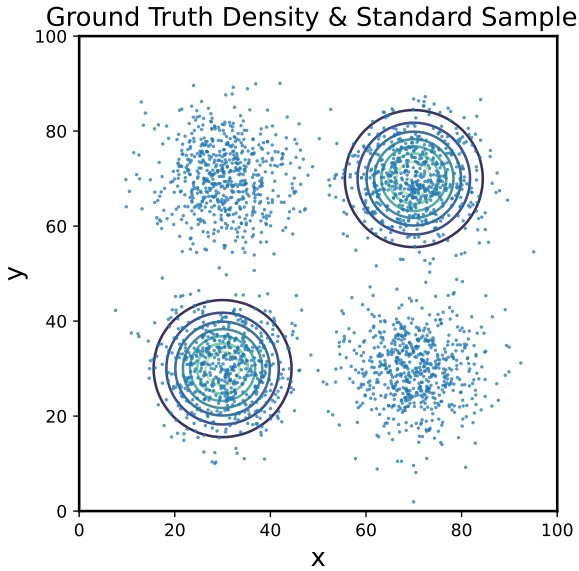
The distribution and a set of samples from it are plotted in Figure 4.1a. Presumably, an appropriate SPN structure for this dataset would be a sum node over the two clusters with weight $\boldsymbol{w} = \begin{pmatrix}0.5 & 0.5\end{pmatrix}$. Each of the clusters could then be modeled by a product over one univariate Gaussian for each of the RVs $X$ and $Y$. However, since we want to show what can be done when the structure makes independence assumptions where it is not appropriate, the root node will be a product node. This corresponds to the assumption that $X$ and $Y$ are independent, which they are clearly not since that is only the case within a single one of the clusters. Each child of the product is then a sum node over two Gaussians, again with weight $\boldsymbol{w} = \begin{pmatrix}0.5 & 0.5\end{pmatrix}$. Each of the leaves models the univariate Gaussian marginal of one of the RVs for a single cluster like $\mathcal{N}_{0,X}(\mu, \sigma^2)$, with mean $\mu = 30$ and variance $\sigma^2 = 50$. The structure of the SPN is shown in Figure 4.1b. As expected, the standard samples drawn from the SPN are inconsistent, as is apparent by the new lumps of samples in the



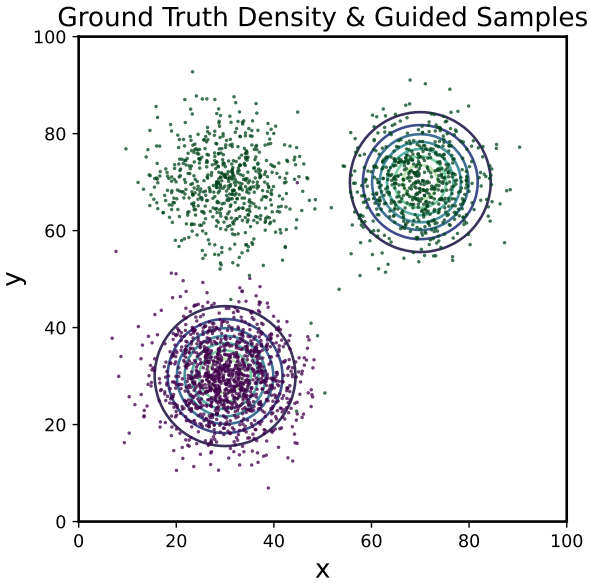

(a) The data distribution as isolines of the density of the two clusters and some samples. The color indicates the membership of the samples to the clusters.




(b) The SPN used for the initial evaluation. The symbols ($*$) and ($\ddagger$) on the sum nodes help reference them later.




(c) Similarly to (b), this shows the data distribution in the background and standard samples from the SPN. They clearly show the inconsistencies that we try to mitigate by learning guides.




(d) Similarly to (b), this shows the data distribution in the background and guided samples from the SPN using $Q$-Learning. The color indicates membership of the conditioning evidence to the clusters. Only about $^3/_4$ of the samples are consistent.

Figure 4.1: This figure visualizes the synthetic dataset task and the results of learning a guide. In particular, the SPN used in the evaluation as well as standard and guided sampling results are shown. The Appendix A.1 also visualizes guided sampling but shows separate plots for different conditioning modes to illustrate when guiding is most effective.

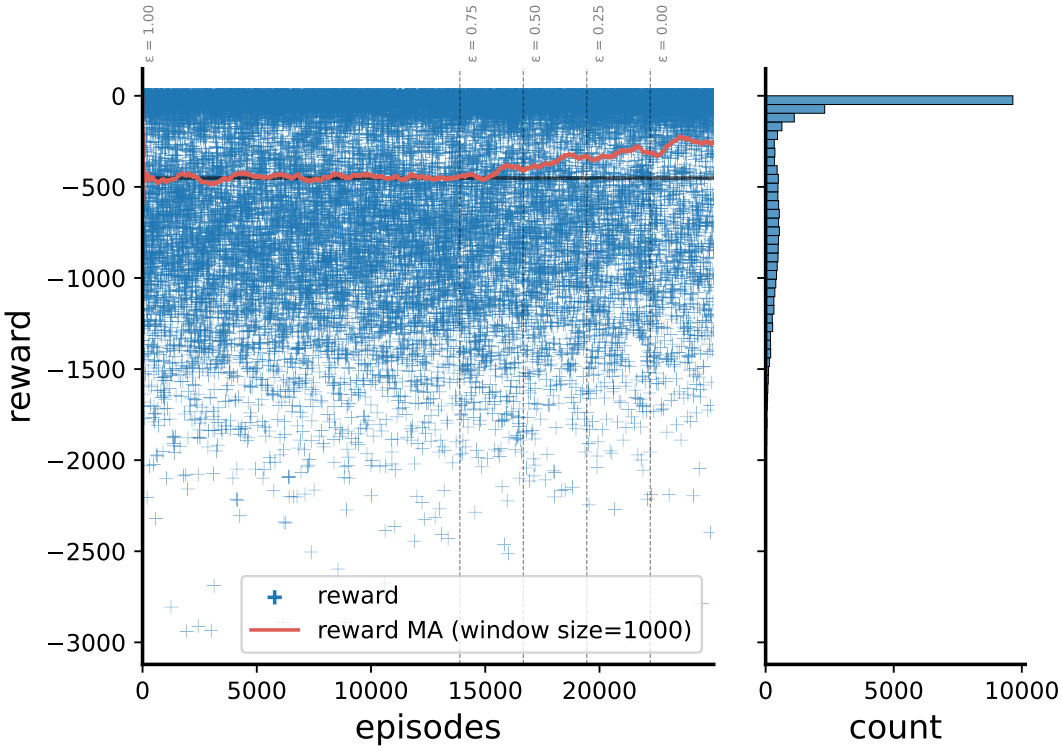



Figure 4.2: This illustrates how training progressed on the synthetic dataset. The blue scattered markers on the main graph are the rewards for each episode, and the moving average over the most recent $1000$ ones is shown in red on top. The black horizontal line is the reward obtained when using the baseline of standard sampling (averaged over $2500$ trails). The top is annotated with the decaying $\varepsilon$ values that were used in that phase (see Algorithm 2.2). The right shows a histogram of all rewards obtained from training, showing that while the variance is large, most rewards are rather high for this dataset and SPN. Overall, the diagram shows that the training does progress to a regime of increased reward/decreased error. The visualization on the left-hand side is inspired by Mundt, Pliushch, and Ramesh [39, Fig. 2].

top-left and bottom-right corners of Figure 4.1c. The likelihood of the real distribution $p_d$ is very low in those regions and we would not expect samples there if we used a better model.

We then ran $Q$-Learning with the same parameters as later in the qualitative comparison Section 4.6.1 where applicable. The dataset contained $25\,000$ samples (i.e. a single one for each episode of training) and the evaluation of the final reward was the mean over $2500$ trials. The progress of improving the reward during training is shown in Figure 4.2, where we see a clear improvement each time we decrease the $\varepsilon$ a step from $1.0$ (purely random choices for exploration) to eventually $0.0$ (always following the learned $Q$ function). Recall that the reward is just the inverse of the reconstruction error. However, a final MSE of about $242.57$ is still considerable, and it helps to look at Figure 4.1d to see that the samples from $\mathcal{N}_1$ are reconstructed convincingly but the ones of $\mathcal{N}_0$ only about half of the time. This is expected and shows a limitation of the path encoding.

There are two possible cases, where either $X$ or $Y$ is provided and the other one is to be conditionally sampled. Suppose for now that $X$ is given as evidence and $Y$ is to be reconstructed. We note that the sub-SPN over $X$ – the sum node marked with $(*)$ – is visited first. Depending on the index of the child with the larger likelihood of the value $X$, we either land in the state $[0,]$ or $[1,]$. We know the most likely cluster that the data instance

| State ID | Path | $Q$ Values (rounded) | Next Sum Node |
|---|---|---|---|
| 0 | $[\,]$ (empty) | -523.86, -591.93 | left (*) |
| 1 | $[0,]$ | -55.24, -860.28 | right (‡) |
| 2 | $[0,0,]$ | terminal | — |
| 3 | $[0,1,]$ | terminal | — |
| 4 | $[1,]$ | -843.01, -43.16 | right (‡) |
| 5 | $[1,0,]$ | terminal | — |
| 6 | $[1,1,]$ | terminal | — |

Table 4.1: This table lists all possible states and the path they correspond to. The table also shows the $Q$ values that were learned and the node that the two actions (choose left, choose right) can be taken in. Note that the $Q$ table entries for terminal states are always zero per definition and therefore omitted.

probably came from and select a child of the sum node over $Y$ marked with (‡) in an informed way. This is made apparent by the concrete $Q$ table that was learned and is shown in Table 4.1. We can, for example, clearly see that in state $[0,]$, in (‡) also the first node has to be chosen for a consistent sample. Indeed, this is faithful to the training data. If, on the other hand, $Y$ was provided as evidence and $X$ was to be reconstructed instead, we would again start in state $[\,]$. This time, however, we would not immediately move to the next state by conditioning on the value of $X$, but would need to sample $X$ *first*, and then consider $Y$. This decision is therefore not based on the evidence about $\boldsymbol{E} = \{Y\}$. That lack of distinctness is also reflected by the $Q$ table, which assigns both states a similar expected reward.

The Appendix A.1 goes into more detail on why the reward near $-250$ is to be expected based on an analytical solution and how samples when conditioned on $X$ and $Y$ each look. This sheds some further light on the strengths and weaknesses of the guided sampling method.

In summary, it is clear to see that the guide is able to learn what it can given the framework, i.e. choose sum node children about which we do not have any given or sampled evidence yet in an informed way. However, due to the limitation of the path encoding, certain inconsistencies cannot be avoided, resulting in a spurious point cloud of instances in Figure 4.1d where none is present in the training data. While that is a central point of further research, our work now continues with an evaluation of the presented guided sampling approach. The question to be answered is whether the observations on this simplistic synthetic dataset also apply to real-world datasets with two to three orders of magnitude more RVs.

## 4.2 Parameter and Structure Learning

The first step in the evaluation of the guided sampling method was obtaining some SPNs on which the experiments could be carried out. To this end, the three different types of structures introduced in Section 2.1.4 were generated and parameters were learned each. The specific datasets that would be used were not of particular interest at this point since the evaluation was carried out on image data of similar dimensions, for which similar structures were expected. Note that while most datasets contain label information for supervised learning, these were never used for training the SPNs since the analysis should apply to the wide range of (unsupervised) density estimation tasks which subsume supervised learning.

### 4.2.1 Choice of Leaf Distributions

In theory, any type of probability distribution of fitting domain can be used as a leaf for SPNs as defined in Section 2.1.3. In practice, however, certain well-known ones are typically used, which include Gaussian and Binomial distributions for grayscale and color image data [48, 50, 54]. Also note that both can be expressed as an exponential family as described in Section 2.3. For the sake of numerical stability and computational efficiency, the choice is typically further reduced to univariate distributions like Binomials or equivalent ones like isotropic multivariate Gaussians, which are effectively a product of univariate Gaussians.

Since Gaussians are defined on continuous RVs and image data is typically available as discrete $8$-bit pixels or color values in $\{0, \dots, 255\} \subset \mathbb{N}$, problems can occur when learning from dataset slices with no variation in pixel values due to infinitesimal variances $\sigma^2$ or when evaluating the model. To combat this it is beneficial to just add uniform noise from $[0, 1)$ to the data to perform *dequantization* [58, Sec. 3.1] [59, Sec. 3.1.3.3]. Also note that for full mathematical correctness, truncated Gaussians on the interval $[0, 256)$ should be used instead of normal Gaussians which have infinite support. As a typically used practical hack, samples can also just be clipped to the respective data range and usual Gaussian distributions can then be used as a simple approximation.

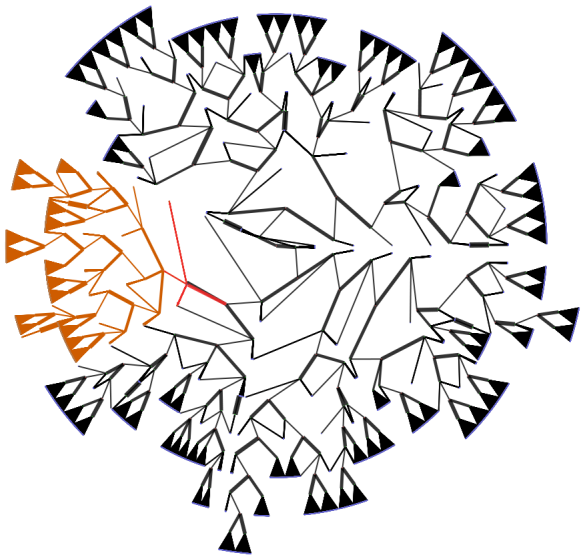
A benefit of Binomials over Gaussians is the generation of much sharper images since the continuity of Gaussians introduces smeary values in between the usual discrete pixel values. An additional trick to produce images that are easier to evaluate qualitatively as a human is the reduction of the number of possible discrete states of the Binomial distribution from, say $256$ (corresponding to $8$ bit integers) to $16$ ($4$ bits). With proper rescaling at the end, this still produces images of sufficient quality but at the same time makes the borders of structures in the image much clearer. By that, they allow for easier qualitative evaluation of the consistency or lack thereof in generated images. Due to this, they were used for most of the evaluations where not noted differently.

### 4.2.2 LearnSPN

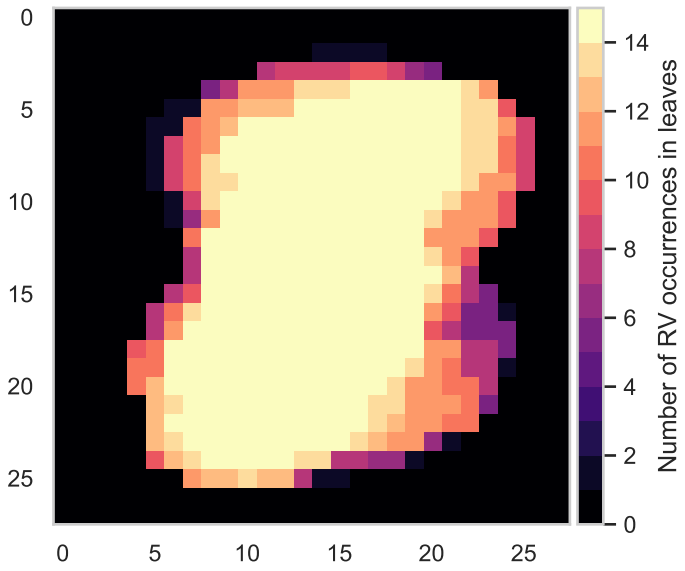
The specific implementation of LearnSPN used in these experiments was `learn_parametric()` from SPFlow [38], with only minor changes due to portability and parallelization performance issues. For the clustering step in sum nodes, classic $k$-means was performed using *scikit-learn* [46]. For finding independent features, the randomized dependence coefficient [33] was calculated and results were grouped by thresholding at $0.3$. If only a single RV was left, no more clustering was performed and a single leaf was created. Naive factorization was performed once a cluster contained less than $1000$ instances, i.e. a product node with a leaf for each remaining RV in the sub-tree was created. All leaves were set to be univariate Gaussians $\mathcal{N}(\mu, \sigma^2)$ where both mean $\mu$ and variance $\sigma^2$ were estimated from the data.

Unfortunately, despite the parallelization to $12$ CPU cores on a capable desktop machine, training still took about $2$ hours for the $28 \times 28$ pixels MNIST dataset. This is due to the discovery of independent feature groups to form product nodes, which scales very unfavorably in the dimension of the input as it computes pairwise correlation statistics. It would therefore not scale to larger datasets without modifications.

However, a different problem with these types of structures was yet more limiting. To use reconstruction losses, one has to be able to choose which parts of the image are to be reconstructed and which serve as conditioning. Due to the very heterogeneous structure of the scopes, this is not very practical. It proved to be unfeasible to find a slice of the image that did not (almost) always cause all visited sum nodes to be conditioned. This is because different sum node children may choose different factorizations, which is especially problematic



(a) The SPN arranged in a radial tree layout[12], where the root is at the center and the leaves are at the perimeter. The orange highlight on the left shows the section of the tree that is analyzed in (b). Note that this layout is only possible for tree-shaped SPNs like the one at hand.




(b) This shows how many leaves of the sub-SPN highlighted in (a) are responsible for each RV of the images. Black regions indicate that these pixels are outside of the scope of the entire sub-SPN. Lighter regions correspond to RVs that occur in many leaves.

Figure 4.3: Visualization of the structure of an SPN learned with LearnSPN over the MNIST image dataset and a heatmap of the number of leaves that range over a selected sub-SPN.

in the upper parts of the SPN. Scope statistics of an exemplary sub-SPN and the greater SPN structure are illustrated in Figure 4.3.

In conclusion, we have learned that these types of structures are not well suited for a first evaluation of the sampling methods to be analyzed. Therefore, other structures based on Einsum Networks were used instead as documented in the next section.

### 4.2.3 Poon-Domingos and Binary Tree Structure

The two main types of structures that were used are the PD and binary tree structure learned using EinsumNetworks [48]. A major consideration for this was the training speed, which took from less than a minute to a few minutes using GPU acceleration depending on the specific structure and dataset. After training, the vectorized structure was disassembled and transferred to a data model that explicitly models the nodes and edges of each individual node. That node-based structure is less efficient but facilitates traversal of the SPN graph that would else be rather cumbersome. The translation was verified by making sure that the log-likelihood of the test dataset split did not change beyond tiny numerical differences.

[12]Visualized using an interactive visual tool that makes use of *graph-tool* (`https://graph-tool.skewed.de/`).

| | | Nodes | | | | | | Paths | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Structure | Layers | $+$ | $\times$ | Leaf | Total | Edges | Param. | Term. | Non-term. | Total |
| PD-4-1 | 8 | 40 | 512 | 3136 | 3688 | 4643 | 6787 | 20 480 | 1396 | 21 876 |
| PD-2-2 | 7 | 19 | 304 | 3136 | 3459 | 4002 | 6562 | 8192 | 547 | 8739 |
| Binary tree | 6 | 9 | 160 | 3136 | 3305 | 3568 | 6416 | 4096 | 273 | 4369 |

Table 4.2: This table provides statistics on the selected SPN structures and the numbers of path states in it. The number of layers is the depth of the SPN from root to the deepest leaf. *Param.* is the number of parameters of the entire SPN, consisting of sum node weights $\boldsymbol{w}$ and leaf distribution parameters like $n$ and $p$ for Binomials. The number of paths corresponds to the number of states in the MDP $\mathcal{M}$ for $Q$-learning.

A major issue with the proposed $Q$-learning approach is that all states have to be enumerated and also visited sufficiently often in training. While space to store a large $Q$ function as a table is less of a concern at first, the fact that training time grows certainly is. Therefore it is very important to limit the size of the SPN in the sense that the total number of possible paths stays relatively low, e.g. at a few thousand states for learning to converge within less than an hour. Note that the number of leaves is not an issue and SPNs of such sizes are still able to reasonably model small image datasets, at least when reducing the number of classes.

The SPNs that were used in later evaluations feature Binomial leaves with $n = 16$ instead of $256$ states as discussed previously. The parameters `num_sums` and `num_input_distributions` (called $K$ in the Einsum Networks paper [48]) were set to $4$ for all structures. The training was performed using online EM with updates after each batch, a batch size of $500$, a step size of $0.05$, and GPU acceleration. It was run until convergence of the training loss, e.g. $5$ epochs for the full MNIST dataset or $25$ for the same one reduced to the first two labels. The $5$-fold increase in the number of epochs stems from the much shorter epochs and therefore reduced number of EM updates when reducing the datasets to about a fifth. In total, three structures were investigated, each generated to fit the concrete datasets at hand:

- **PD-4-1**: This PD structure was generated by splitting the image into $4$ vertical stripes and performing no splits on the vertical axis ($1$ slice). This is similar to what was done for Einsum Networks [48] as shown in Figure 2.7.

- **PD-2-2**: This PD structure was generated by splitting the image both horizontally and vertically in half ($2$ slices each). It was chosen as an alternative to PD-4-1.

- **Binary tree**: For this, no repetitions were performed and only a single splitting tree was generated to obtain a single RV set cut and not run into issues similar to LearnSPN structures, where reconstruction tasks are hard to design in a useful way. The splitting depth was set to $2$.

The hyperparameters for the structure generators were chosen in a way that resulted in similar SPN sizes and manageable numbers of paths. Some key statistics for the generated graphs on the MNIST dataset are provided in Table 4.2, like the number of nodes of certain kinds. Note that they only significantly differ by their number of sum nodes, where PD-4-1 has the most. Since those and their arrangement are crucial in determining the number of possible sampling paths, PD-4-1 has about five times as many non-terminal states as the binary tree structure. Note that we do not care that much about terminal states since they are effectively ignored in $Q$-learning anyways. For datasets of different sizes, like SVHN, the statistics only changed by small amounts.

| Name | Source | Instance Shape | Channels | #RVs | #Training Ex. | #Test Ex. |
|---|---|---|---|---|---|---|
| MNIST | [31] | $28 \times 28$ | 1 (grayscale) | 784 | 60 000 | 10 000 |
| Fashion MNIST | [66] | $28 \times 28$ | 1 (grayscale) | 784 | 60 000 | 10 000 |
| SVHN (cropped digits) | [40] | $32 \times 32$ | 3 (RGB) | 3072 | 73 257 | 26 032 |

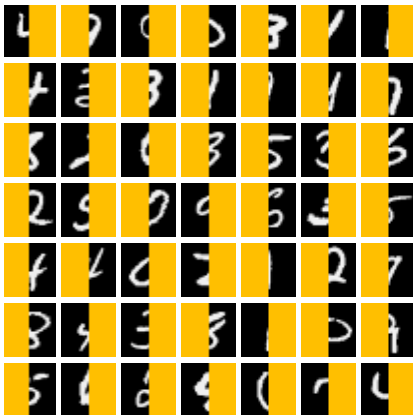
Table 4.3: This table lists and compares the datasets that were used in the evaluation of guided sampling. *#RVs* is the dimensionality of each instance, i.e. width $\times$ height $\times$ channels. *#Training Ex.* and *#Test Ex.* are the numbers of examples in the training and test splits as described in the relevant sources, respectively.

## 4.3 Datasets

To be able to quickly assess the quality of generated samples, image data was selected as the domain of application of the sampling procedures. Furthermore, the mature and active field of machine learning for computer vision has given rise to (a) a large number of datasets, (b) a variety of research on many different aspects of image processing, and (c) high-quality implementations of many common techniques including visualizations. Images can also be a challenging domain due to the high-dimensional nature of images of even low resolution, as already tiny $32 \times 32$ RGB images consist of $3072$ RVs each.

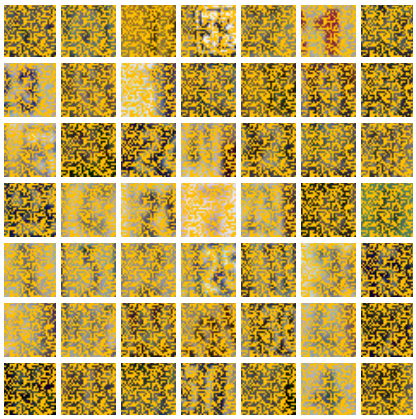
In total, three image datasets were chosen due to their popularity and size. *MNIST* and *Fashion-MNIST* were chosen as they are commonly used for prototype and proof of concept works due to their limited size and only being grayscale. SVHN stands for *Google Street View* house number signs and is a larger dataset, in both its spatial size and the added color depth [40]. Unfortunately, even larger datasets could not be used since the SPNs were of very limited capacity and it is unclear what the meaning of the guided sampling evaluation would be if the SPN cannot even remotely model the data distribution. Note that the SVHN "extra" split was not used for training the SPN or the sampling guide. These datasets are also suitable because they were used in the somewhat related publications on RAT-SPNs [50] (MNIST and Fashion MNIST) and Einsum Networks [48] (SVHN). In both of the papers, additional datasets were used too, but to simplify the implementation and qualitative visual evaluation only image datasets were used (and thus not, for example, *20 Newsgroups*) and the selection was cut down further. The lesser-known *SEMEION* dataset was not used and *CelebA* was deemed too large for this initial evaluation. Examples of some dataset instances with their occlusion applied are shown in the next section (Figure 4.4).

As mentioned above, the improvement gained by learning a guide (or the lack thereof) is only clear to interpret if the SPNs reasonably model the data in the first hand. As the SPNs were very limited in size, an additional variation of each of the datasets in Table 4.3 was considered. To reduce the complexity and have for each class/label in a dataset at least as many leaves in the SPN, each dataset was reduced to its first two labels. In MNIST and SVHN, this means that only data with the labels "0" and "1" was retained. In Fashion MNIST only images tagged as "T-shirt/top" and "Trouser" were used in the reduced variant. Note that this label information was again only used for splitting the data, and not for the actual training of the SPNs or the sampling Guide.



(a) Examples for the 2 slices variant on full MNIST.

(b) Examples for the 4 slices variant on full Fashion MNIST.



(c) Examples for the tree variant on full SVHN.

Figure 4.4: This figure shows what area of the images was to be reconstructed in the different tasks in orange. It also shows the three datasets used later in the evaluation, where MNIST and Fashion MNIST are grayscale and SVHN are RGB color images.

## 4.4 Reconstruction Task

We train the guides for sampling using the error of reconstruction of partial images. That is, given some partial image as evidence $\boldsymbol{e}$, the reward is the negative of the MSE of the reconstruction with respect to the ground truth. This means that we have to delete parts of the images in order to obtain the occluded image task to learn on. Three methods were used in this thesis, of which all are visualized in Figure 4.4. They are:

- **2 slices**: In this setting, the image was vertically sliced into two halves, of which either the left or the right-hand side had to be reconstructed from the other one. This was used for evaluating the PD-4-1 and PD-2-2 structures, as in both cases the conditioning did not fix all sum nodes and therefore was something to learn. The choice of which side was to be reconstructed was sampled uniformly in each episode.

- **4 slices**: Similarly to the previous setting, two or three slices out of four were provided as evidence and it was used for the same structures. If conditioning fixed all sum nodes, the task was re-generated until an SPN with an unconditioned sum node was found. That can happen on PD-2-2 if two slices from the two different halves were provided.

- **Tree**: The previous two tasks were for the PD structure. The binary tree structure was trained with a different task, which is specific to the SPN at hand. The structure has a sum node at the root over several products that equally partition the scope into disjoint sets. In the case of binary trees, two such sets are created and shared by all root node children. On partition was therefore randomly sampled and evidence was provided for only that set of RVs.

In total, three datasets in both the full and reduced label variant and several structures and occlusions were tested. The six dataset variants and five SPN and occlusion pairs resulted in a total of $6 \times 5 = 30$ combinations.

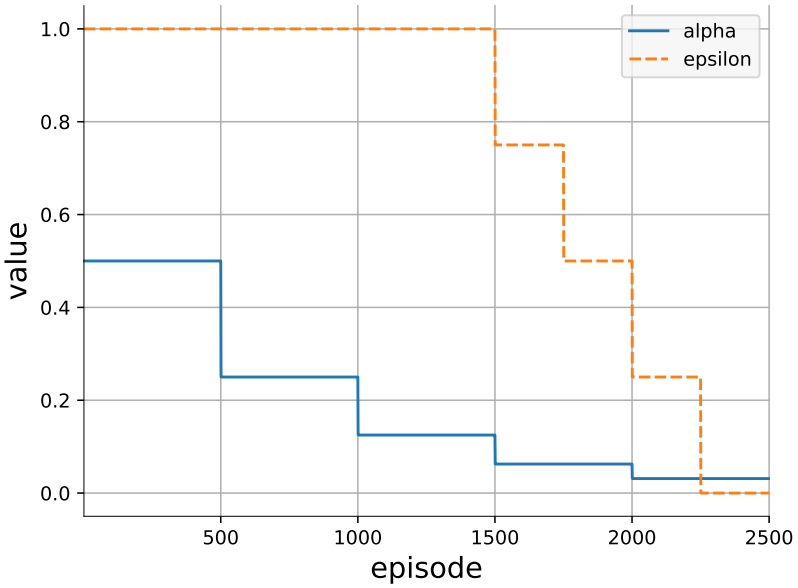



Figure 4.5: This figure visualizes the precise schedule used for all $Q$-learning runs over an example number of episodes of $25\,000$ as was used later. If the number of episodes differed, the schedules contracted or stretched accordingly, i.e. the first reduction of $\varepsilon$ always occurred after $60\%$ of the training progress.

## 4.5 Q-Learning Configuration

The $Q$-learning Algorithm 2.2 has the following hyperparameters: the number of episodes to train, the exploration factor $\varepsilon$, and the learning rate $\alpha$. Similarly to the work on neural architecture search by Mundt, Pliushch, and Ramesh [39], $\varepsilon$ was decayed from a purely random exploration phase in the beginning to ever more deterministic greedy behavior. In particular, the initial value of $\varepsilon = 1$ is decayed at $60\%$, $70\%$, $80\%$, and $90\%$ of the training progress by $0.25$ each to eventually reach $\varepsilon = 0$. Similarly, the learning rate $\alpha$ was exponentially decayed from $2^{-1} = 0.5$ to eventually $2^{-5} = 0.031\,25$ in four equally spaced steps. Both the $\alpha$ and the $\varepsilon$ schedules are visualized in Figure 4.5.

Another point of consideration is the number of episodes to run. An appropriate number was determined by running a grid search over episode counts of $200$, $400$, $600$, $800$, $1000$, $2000$, $4000$, $6000$, $8000$, $10\,000$, $20\,000$ and $50\,000$. As shown in Figure 4.6, training for $25\,000$ episodes was certainly enough to let the $Q$ table converge to sufficient accuracy and thus was chosen for subsequent experiments. It is also noteworthy that all structures converged after a similar number of episodes of training, despite having $Q$ tables of varying sizes as shown in Table 4.2.

The same figure also compares how normal and double $Q$-learning perform relative to each other. We can see that the difference is negligible once several thousand episodes of training are done, which is required anyway for convergence. Note that the evaluation was only performed on Fashion MNIST since it is expensive, but the conclusions should hold for other datasets too since the time to convergence is mainly influenced by the number of states, which is in turn only determined by the SPN structure and not the concrete dataset in use. The following experiments all use double learning, but simple $Q$-learning should perform similarly.

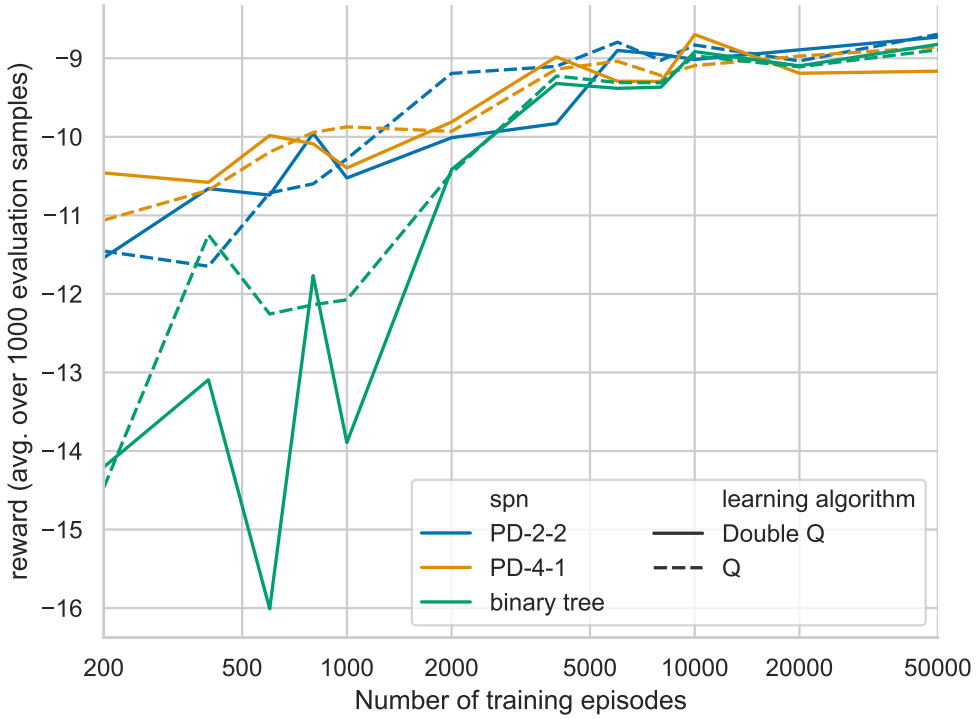



Figure 4.6: Comparison of the mean reward (over $1000$ samples) after training guided sampling for a certain number of episodes on the full Fashion MNIST dataset. Separate results are shown for the three types of structures and normal as well as double $Q$-learning. Note that the horizontal scale is logarithmic and fresh runs were started from scratch for each data point to prevent “lucky” runs to distort conclusions. Progress toward higher rewards is already observed at a few hundred episodes of training, but convergence occurs at around $10\,000$ to $20\,000$ episodes – curiously regardless of the concrete SPN structure. Also, double $Q$-learning does not yield a significant improvement or deterioration with respect to normal $Q$-learning.

## 4.6 Comparing Standard and Guided Sampling

We have now chosen SPN structures, defined hyperparameters, and successfully trained them on several datasets and their reduced variant. We also determined how to train a Guide using $Q$-learning. It is now time to compare how guided sampling based on $Q$-learning performs when compared with standard (unguided) sampling. To investigate this, several experiments were carried out which shall be presented in this section. We first show and discuss the main comparison and insights and then investigate some select configurations in more detail.

### 4.6.1 Quantitative Comparison

The first thorough comparison is shown in Figure 4.7. It is clear to see that while generally speaking guided sampling does lower the error, the improvement is sometimes small. This effect is thus better viewed in relative terms. We should first recall that the MSE is defined as the negative of the reward $r := -\,\text{MSE}$, meaning that we want to *maximize* the reward to *reduce* the error by switching from standard to guided sampling. Therefore in Figure 4.8a, the improvement $\left(\overline{r}_{\text{guided}} - \overline{r}_{\text{standard}}\right) / \overline{r}_{\text{standard}}$ or equivalently $\left(\overline{\text{MSE}}_{\text{standard}} - \overline{\text{MSE}}_{\text{guided}}\right) / \overline{\text{MSE}}_{\text{standard}}$ is shown in percent. In most cases, the error could be reduced by at least a few percent. The effect is generally more pronounced with the reduced label variants of the datasets. This supports the hypothesis that there is more to be learned when the SPN better models the data, since groupings of similar instances can be identified better by the $Q$ table. Also, the improvement is largest with the binary tree structure and with the Fashion-MNIST dataset. However, there are a few cases where a slight deterioration can be observed on full MNIST, namely with the PD-2-2 structure ($2$ slices) as well as PD-4-1 ($4$ slices). Additionally, in four of the ten cases on SVHN, no improvement or even a slight degradation of error can be seen. In the other six cases, however, improvements of up to $10\%$ are observable. Looking at the entire set of experiments, we can conclude that in $24$ out of all $30$ cases the error could be reduced, and in $3$ cases even by over $25\%$.

As touched upon in Section 3.4, the $Q$-learning guide is some combination of sampling with aspects of MPE: While child indices in conditioned sum nodes and values in leaves are sampled, the guide is deterministic – much like the arg max in Algorithm 2.1. While a future direction of research might consider using MPE reconstruction errors as a reward too, one can also just use the $Q$ table learned with sampling and use it directly for MPE. The analogous visualization of the absolute scores as in Figure 4.7 for sampling is provided in the appendix in Figure A.2. In short, it can be seen that the overall error is lower than for sampling as one would expect it to be. Jumping directly to the relative comparison as provided in Figure 4.8b, we can see that the improvement that we can observe by introducing a learned guide is significantly higher than for sampling – even though we did not train the guide on that specific task and instead only on the closely related one of sampling. In particular, on MNIST and with binary trees improvements of over $43\%$ can be observed, while the number of instances in which performance decreased by introducing the guide went back. For any individual combination of the dataset, reconstruction task, and structure, improvements when sampling and when performing MPE is roughly similar, suggesting that either of the evaluations is indicative of the other.

### 4.6.2 Qualitative Comparison

Next, it is interesting to inspect a few selected ones of the many experiments that were presented in the previous section to gain some further insight into the guiding method. The first obvious question is whether

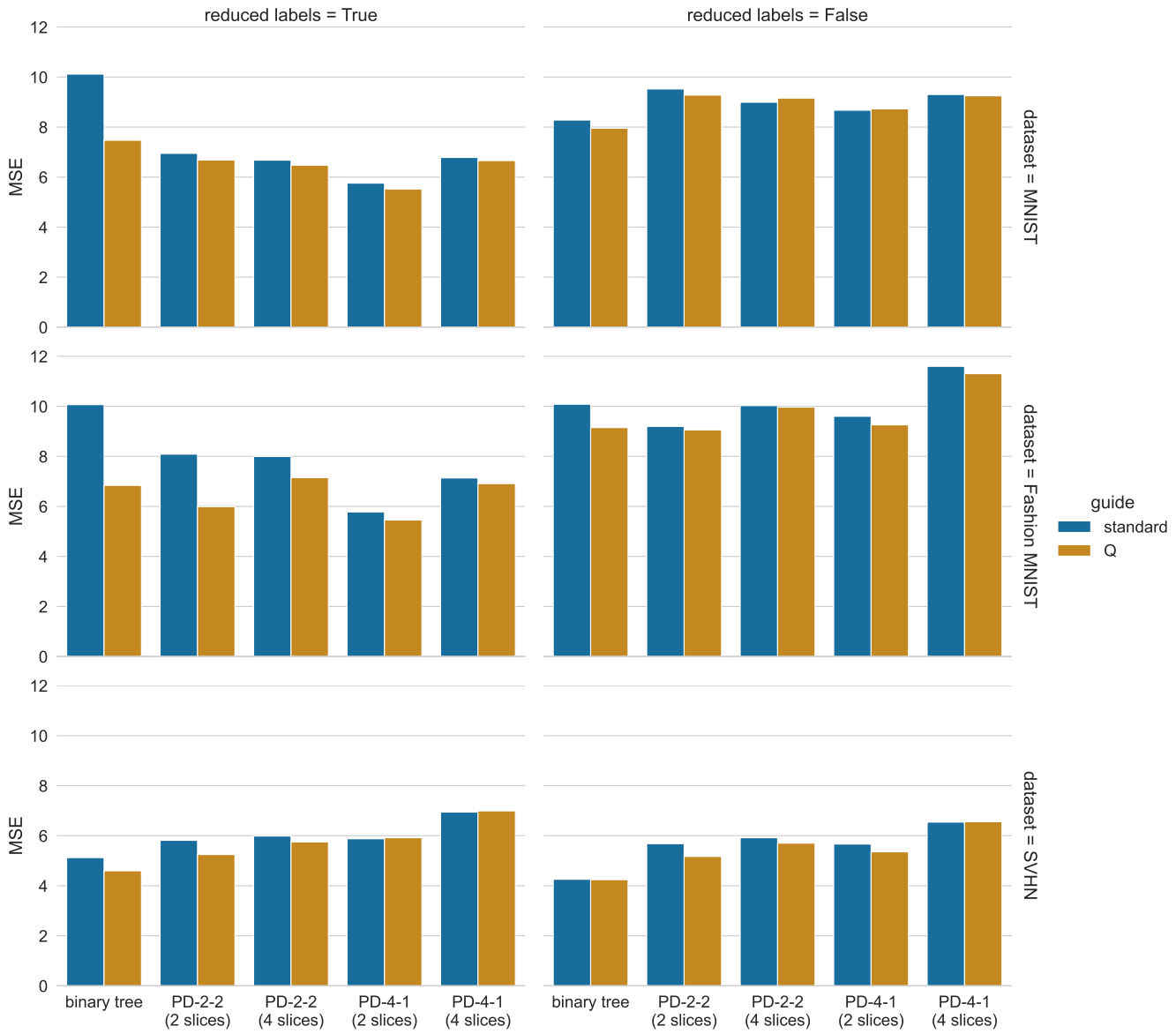
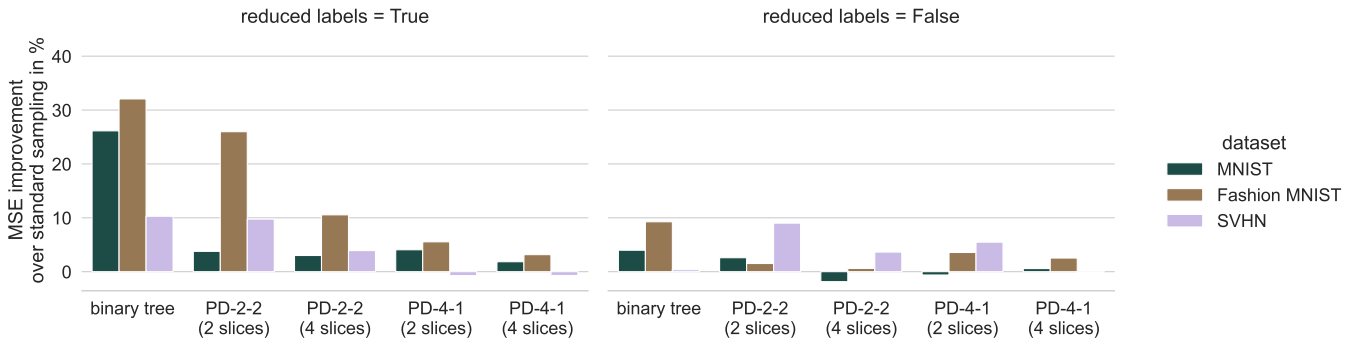
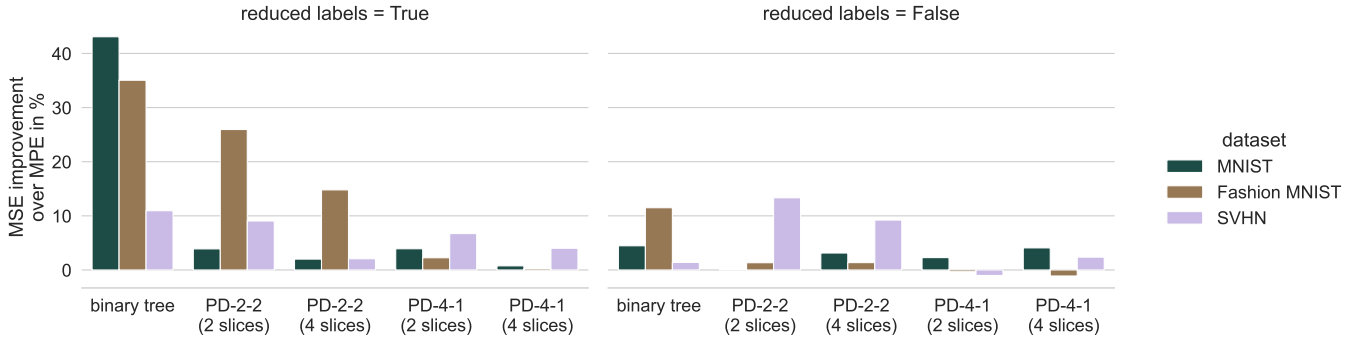



Figure 4.7: This figure compares the mean squared error ($\mathrm{MSE}$) when using standard and guided sampling over several datasets and two reduction variants. The values were obtained by performing $2000$ sampling operations. Lower values are better since they correspond to more accurate reconstructions. The corresponding visualization for the case of MPE is provided in the appendix in Figure A.2.




(a) Qualitative comparison of the improvement gained by learning a guide for sampling.




(b) Qualitative comparison of the improvement gained by learning a guide for sampling and using it for MPE.

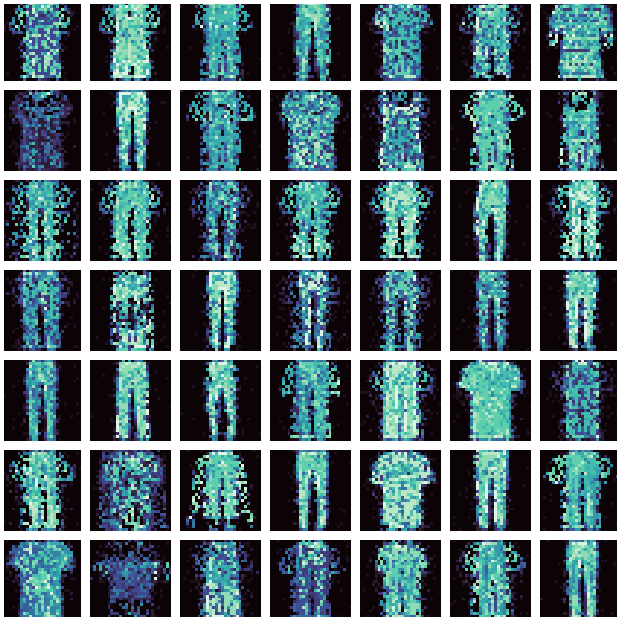
Figure 4.8: These bar charts show the relative improvement of the $\mathrm{MSE}$ when using guided over the baseline of standard sampling ($100\%$) over several datasets and two reduction variants in (a). In addition, (b) shows the same comparison when performing MPE in both the conditioned sum nodes and leaves. Note that the multiple rows of the absolute value comparison in Figure 4.7 have been collapsed to multiple bars per SPN structure and reconstruction task since the distinction between the guides is now expressed as a single value in relative terms. The data points were again averaged over $2000$ samples. Note that larger values are better since we want to increase the reduction of the error caused by introducing guiding. Positive values indicate that guided and negative values that standard sampling is better. $100\%$ improvement would reduce the error to zero and would correspond to a perfect reconstruction of the query evidence $e$ to the full underlying image.

the larger improvements are already discernible. To this end, two structures that are very different in nature shall be inspected on the Fashion MNIST dataset with reduced labels for better visual understanding. If we expect to see a difference somewhere, it will probably be most pronounced on this dataset where the largest improvements were observed as shown in Figure 4.8. Firstly, the binary tree structure and, since it showed the largest improvement of the PD structures, the PD-2-2 ($2$ slices) structure shall be inspected. The comparison of the standard and guided sampling is shown in Figure 4.9. Note that the quantization of the dataset as discussed in Section 4.2.1 is retained in this image and therefore might look different than in other dataset illustrations (*quantization noise*). The shown images were generated at random without cherry-picking. In the first row, standard and guided samples on the binary tree structure with corresponding tree occlusions are compared (4.9a and 4.9b). Inconsistency in this randomly generated SPN architecture would correspond to the occurrence of more pronounced noise, similar to salt-and-pepper noise. The guide appears to reduce such extremely noisy images somewhat, but there is still much that could be improved. The second row shows the same comparison on the PD-2-2 ($2$ slices) structure. We can see some inconsistencies on the left (4.9c), where both the two horizontal halves often do not match, but also the vertical two parts within each half are inconsistent. The right (4.9d) contains about half as many images that are strongly inconsistent.

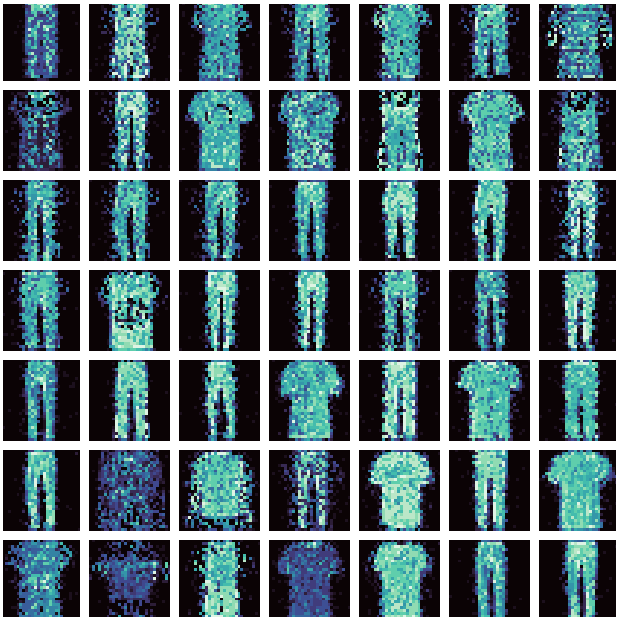
Again, it is interesting to look at MPE reconstructions as provided in Figure 4.10 for another view. Here, the improvement by guiding on binary trees is arguably better perceptible then for sampling, and the results on the right (4.10b) look more consistent than on the left (4.10a). It is, however, difficult to come to a final judgment. In the case of the PD-2-2 structure, the difference between MPE and sampling is immediately obvious by the more smooth but blurry images. Usually, we would expect sharp instances here, but since the data is modeled by Binomials, the MPE query returns the mean of many such individual leaves and therefore produces a smeary image. As is the case for sampling, guided MPE reconstructions (4.10d) show almost half as many strong inconsistencies as unguided ones. The left- and right-hand sides of the guided MPE reconstructed images appear to match the same type of clothing much more often as was the case for sampling.

It is also insightful to look at a visualization of the number of updates of the $Q$ table cells that occurred during learning. Figure 4.11 shows that the value $q(s, a)$ for some $s, a$ pairs is updated much more frequently than others. The same could already be seen for the few states of the synthetic dataset scenario in Section 4.1 too. This is, however, not avoidable since it mainly stems from the fact that non-terminal states containing indices of nodes that are visited early in the traversal are visited much more often than later ones, and $Q$-learning is designed to work under these conditions as long as training is performed sufficiently long. In this RL setting, there is indeed an abundance of data. Additionally, due to the way conditioning is performed, some states are only visited by the conditioning pass and not the agent and therefore not all actions in that state get "explored", coincidentally leading to intriguing staircase patterns. The $Q$ table will still learn to act optimally for the tasks that it had seen sufficiently often during training, meaning that validation and subsequent applications must be performed on the same types of tasks. A possible point for optimization is to remove all terminal states from the table since they can only ever contain $0$ per definition as discussed above. This would reduce the $Q$ table size by about $94\%$ on all structures as can be seen in Table 4.2. While requiring only trivial algorithm changes, it would not change the update rule and by extent the overall speed of convergence.

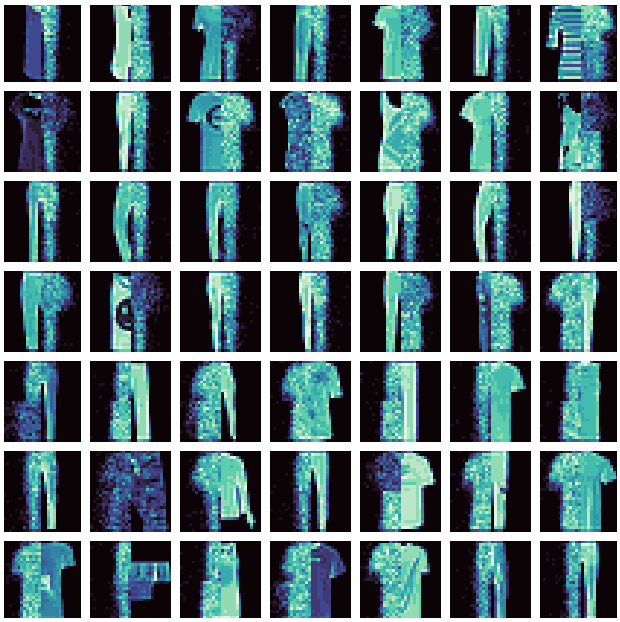
A different way of viewing the results is by inspecting how the distribution of rewards of reconstruction sampling changes during the course of training. To this end, Figure 4.12 compares the two extremes that were observed in the quantitative sampling comparison in Figure 4.8a. On the left (4.12a), we see almost no change in the distribution, neither in terms of the mean nor in terms of the percentiles or overall shape. On the right (4.12b), we see that as soon as the greedy factor $\varepsilon$ is decayed from *part 6* onward, the distribution shifts to the right in the direction of better reconstructions. In particular, in the last parts, the distribution becomes increasingly skewed and the median improves more strongly than the mean. Also note that while the $95^{\text{th}}$ percentile does not significantly change for the last four parts, the $5^{\text{th}}$ percentile does so substantially. In



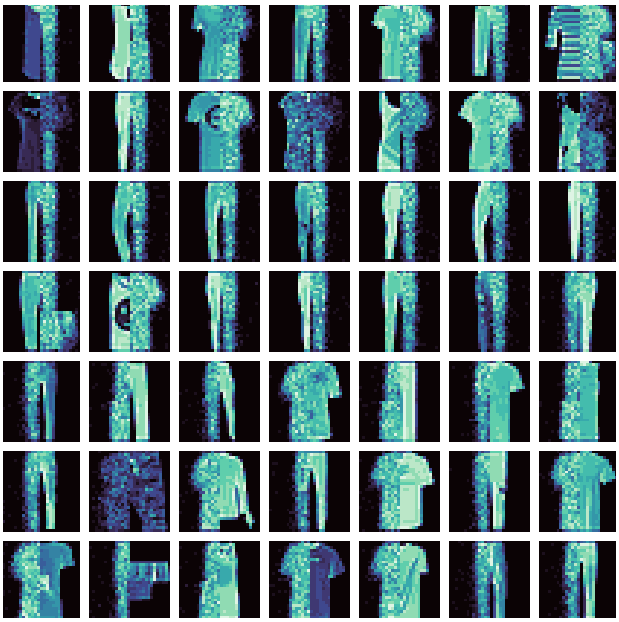
(a) Standard reconstruction samples from binary trees.



(b) Guided reconstruction samples from binary trees.

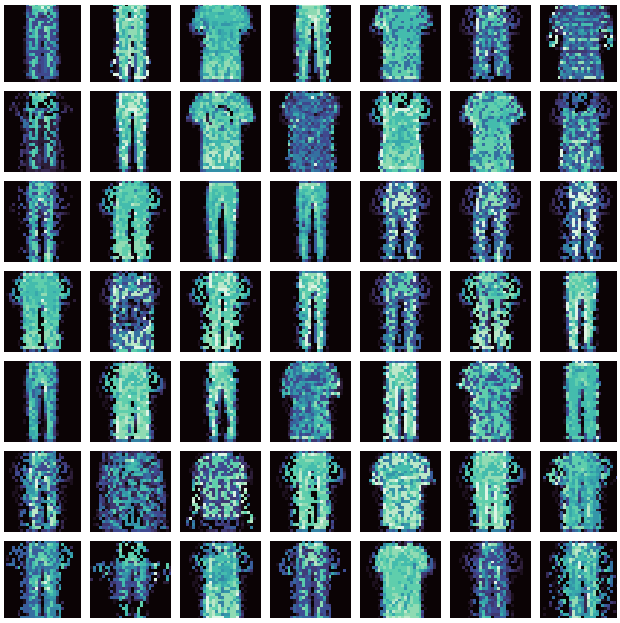


(c) Standard reconstruction samples from the PD-2-2 ($2$ slices) structure.

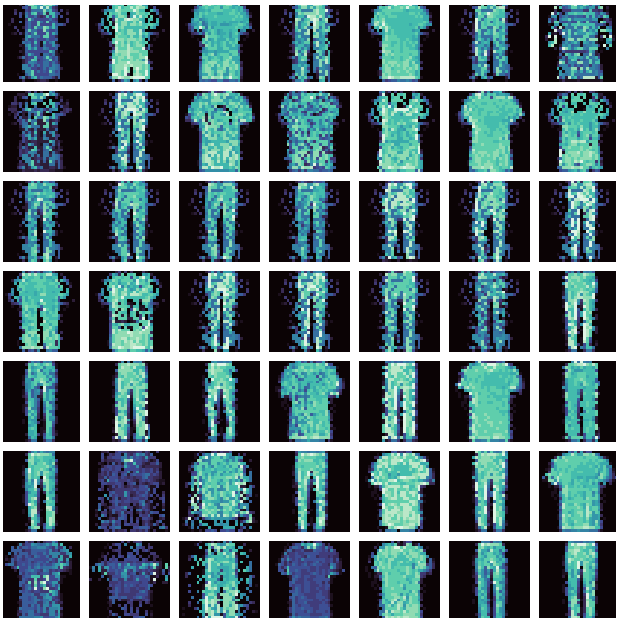


(d) Guided reconstruction samples from the PD-2-2 ($2$ slices) structure.

Figure 4.9: This figure shows randomly chosen standard and guided samples on two select structures where a large improvement was observed by the introduction of guidance. The dataset is Fashion MNIST reduced to the first two labels. All $7 \times 7$ images are the same in all four plots, and the first four and last three rows are each conditioned on the same pixels, respectively. In addition to the luminance of the grayscale dataset, the turquoise hue was modified for a better distinction of the values. While improvements of the right- over the left-hand side can be seen, the difference is rather subtle.



(a) Unguided MPE reconstruction from binary trees.



(b) Guided MPE reconstruction from binary trees.

(c) Unguided MPE reconstruction the PD-2-2 ($2$ slices) structure.

(d) Guided MPE reconstruction from the PD-2-2 ($2$ slices) structure.

Figure 4.10: This figure is analogous to Figure 4.9 and shows example reconstructions using standard and guided MPE instead of sampling. The images and conditioning patterns are also the same to aid comparisons.

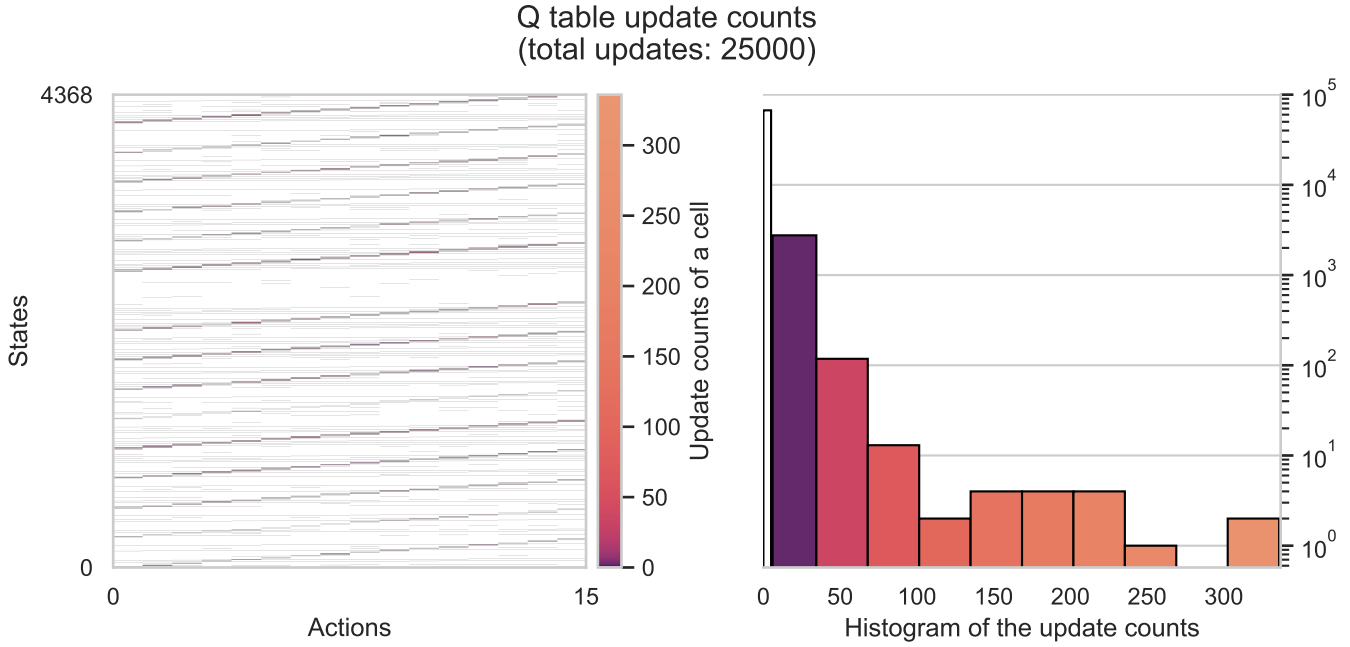
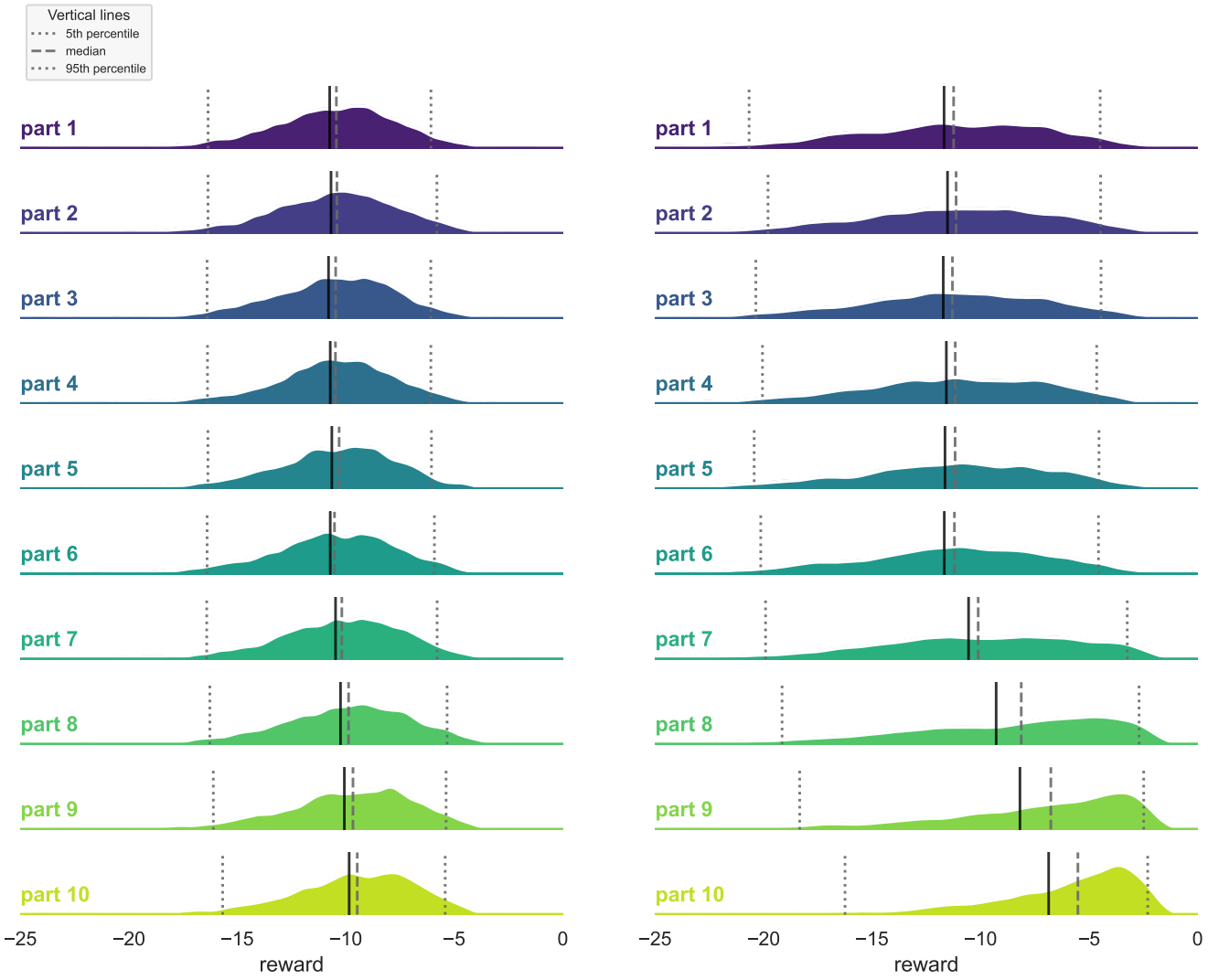



Figure 4.11: The left shows a visual representation of how often the entries in the $Q$ table are visited. The vertical extent shows all states in the SPN and the horizontal direction all possible actions each. Note that each state has the same number of actions since this is about the binary tree structure (on reduced MNIST). In general, there might be different numbers of actions $|A_s|$ in different states $s$. The number of states is much larger than the number of actions, making each cell appear wider than tall. White areas indicate states without updates, which are likely terminal ones that never receive updates in $Q$-learning per design. The right shows a histogram over the update counts in the cells. A total of $25\,000$ updates were performed, i.e. one for every episode.




(a) Training on MNIST with all labels and on PD-2-2 with $4$ slices.

(b) Training on Fashion MNIST with reduced labels and on the binary tree structure with corresponding occlusions.

Figure 4.12: This figure shows how the distribution of rewards shifts as the training of the guide progresses. To this end, all $25\,000$ reward values were collected, divided into $10$ equal parts, and the distribution was plotted in a slightly different color each. Additionally for each graph, the mean is given in thick black. Percentiles at $5\%$, $50\%$ (median), and $95\%$ are provided as gray dashed lines. The shared vertical scale is omitted for visual clarity. More rewards further to the right are better.

turn, this means that while the best reconstructions do not improve any further, particularly bad ones become more scarce. This is likely just a by-product of decaying $\varepsilon$ step by step. Overall, the distribution shift on the left provides little insight into why training could not improve sampling in that specific case, but the shift observed on the binary tree structure on the right confirms the successful improvement of sampling in that scenario.

## 4.7 Exploring Variations of Conditioning in Sampling and MPE

There are multiple ways in which conditioning of a sum node $N$ on some partial evidence $\boldsymbol{e}$ can be performed. The one that is provably correct in the sense of not introducing any bias is the one where in a sum node with conditioning, a child index $i$ is sampled from the categorical distribution induced by the updated weights $\widetilde{\boldsymbol{w}}$. It is defined in lines 18 to 21 of Algorithm 3.1 and its unbiasedness/consistency was proven as Theorem 26. This method is called SampleStandard in the context of this comparison. The goal of this comparison is to provide an understanding of which components of $\widetilde{\boldsymbol{w}}$ are most relevant in the style of an ablation study. In total, these types of conditioning were investigated and compared with each other and with the $Q$-learning approach:

- **SampleStandard**: Sample $i \sim \mathrm{Cat}\,(i \,|\, \widetilde{\boldsymbol{w}})$, where $\widetilde{w}_j = \frac{w_j C_j(\boldsymbol{e})}{node(\boldsymbol{e})}$, $\boldsymbol{w}$ are the weights of the sum node $N$, and $C_j$ the $j^{\text{th}}$ child.
- **SampleWeights**: Sample $i \sim \mathrm{Cat}\,(i \,|\, \boldsymbol{w})$, where the conditioning by evidence $\boldsymbol{e}$ is effectively ignored when choosing sum node children.
- **SampleLikelihood**: Sample $i \sim \mathrm{Cat}\,(i \,|\, \boldsymbol{l})$, where $l_j = \frac{C_j(\boldsymbol{e})}{node(\boldsymbol{e})}$. In this variant, the mixture weights $\boldsymbol{w}$ are ignored and effectively assumed to all be equally set to $1/K$, where $K$ is the number of leaves of $N$.
- **Off**: Sample $i$ uniformly from $\{0, \ldots, K-1\}$. This should perform the worst as it ignores both the SPN weights as well as the evidence $\boldsymbol{e}$. It is equal to SampleWeights, where all $K$ weights are changed to $w'_j = 1/K$.

Note that the above choices are only considering the *conditioning*, and not the operation performed in sum nodes without evidence, where the Guide is still used. In the leaves (conditional) sampling is still performed. The $Q$-learning approach should ideally outperform all of them.

Similarly, one can imagine the same methods to be applied in MPE computation, where we are taking the arg max over the varying responsibilities $\widetilde{w}_j$ instead of sampling from them. In addition to the different conditioning modes, the operation in unconditioned leaves changes from sampling to MPE as well. The following methods were considered:

- **MPE**: Choose $i = \underset{j \in \{0,\ldots,K-1\}}{\arg\max}\ \widetilde{w}_j$ deterministically.
- **MaxWeights**: Choose $i = \underset{j \in \{0,\ldots,K-1\}}{\arg\max}\ w_j$ deterministically.
- **MaxLikelihood**: Choose $i = \underset{j \in \{0,\ldots,K-1\}}{\arg\max}\ l_j$ deterministically, where $\boldsymbol{l}$ is defined as for SampleLikelihood.
- **Off**: Sample $i$ uniformly as above, i.e. choose a random child index.

Visualizations of the absolute performance of these variants are given as background information in the Appendix A.3. Here, the focus lies directly on the relative comparison, where all of the three methods in the sampling and MPE variant each are compared to the corresponding Off variant.

The resulting relative improvement over that baseline is given in Figure 4.13 for sampling and in Figure 4.14 for MPE. Note that the evaluations were performed using the very same $Q$ table as in the comparison above in Section 4.6.1. We can see that first of all, the results on the full Fashion MNIST datasets are very different from the rest, in that sampling only by the weights strongly increases the error over the respective Off baseline. The likelihood of the evidence $\boldsymbol{e}$ appears to be of particular importance in that case. In general, it is clear to see that at least on MNIST and Fashion MNIST, the inclusion of the evidence is crucial for both good sampling and MPE, and much more important than the weights or a learned $Q$ guide. Interestingly, this does not always seem to be the case for MPE on SVHN. Aside from that case, combining both weights and evidence in SampleStandard/MPE usually improves over the SampleLikelihood/MaxLikelihood variant, even if SampleWeights/MaxWeights alone performs even worse than Off. The theoretical analysis of standard sampling in Section 3.1 and the good performance of MPE from Algorithm 2.1 are confirmed in that they are the best of all unguided methods, where both weights and evidence likelihood are crucial for good sampling performance. They are only topped by the learned guide, although relatively speaking, weights and evidence likelihood are more important than that.


Figure 4.13: Improvement of the various sampling variants SampleWeights, SampleLikelihood, SampleStandard, and guided sampling with $Q$-learning compare relative to the Off baseline ($100\%$). More is better. The average rewards were estimated over $2000$ samples and the absolute error is shown in the Appendix A.3.

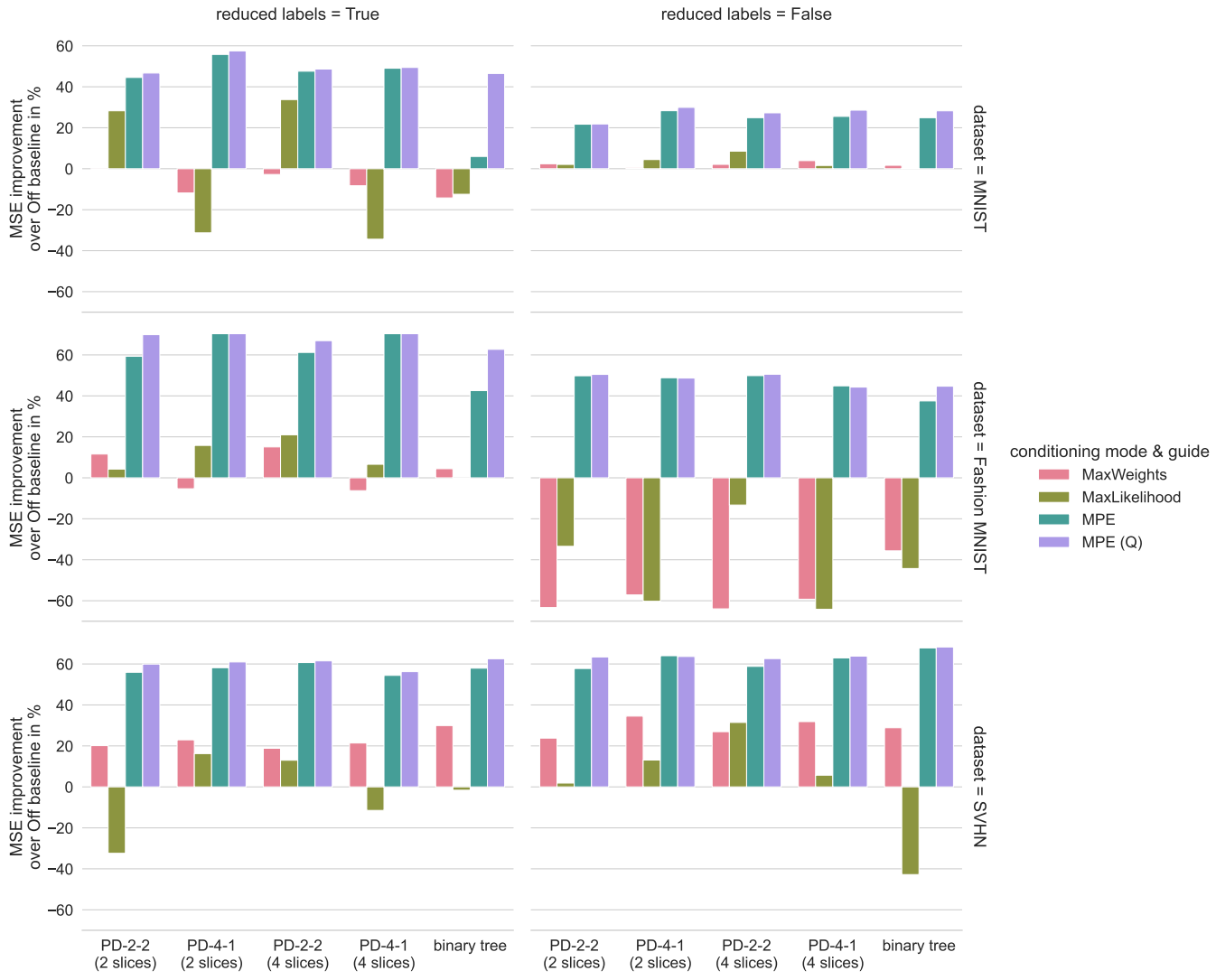



Figure 4.14: Improvement of the various MPE computation variants MaxWeights, MaxLikelihood, MPE, and guided MPE with $Q$-learning over the Off baseline, analogously to Figure 4.13. The absolute error is shown in the Appendix A.4.

# 5 Conclusion and Future Work

As we have motivated in the introduction, probabilistic modeling offers many advantages over the prevalent deep neural networks, and SPNs are one of the more practical approaches to that. While they have since been applied to more and more challenging density estimation, classification, and out-of-data detection tasks, sampling remains one of their weaknesses. The conditionally and unconditionally sampled instances of, for example, images often show high-quality subregions but the overall appearance shows inconsistent patches. To investigate this, a formal analysis of the standard sampling method showed several properties: The algorithm is (1) tractable, i.e. it runs in linear time in the size of an SPN, (2) correct, in the sense that it samples from the distribution encoded by the SPN without any bias, and finally (3) that it is numerically stable.

Nonetheless, there is still much room for improvement, since the heuristic structures like PD and binary tress necessary for scaling to large data domains make wrong independence assumptions which in turn produce inconsistent samples. For this reason, the guided sampling procedure was presented and analyzed. It keeps book on the path that the current sampling run has since taken through the SPN and learns how to sample later parts accordingly. It does so in a reinforcement learning setting with tabular normal and double $Q$-learning, where the task is to reconstruct occluded images with little error. Subsequent evaluation on synthetic data showed that the approach is indeed feasible in principle but also confirmed that there are further opportunities to improve the framework. A qualitative and quantitative comparison of standard and guided sampling on several SPNs, occlusion types, and three typical image datasets showed that the method indeed mostly improves the sample quality significantly, although in a few cases it did decrease performance. The improvement was mostly clear to observe in quantitative error comparisons and less visually apparent on individual images. In the course of the evaluation, typical convergence times were discovered empirically and it was shown that normal and double $Q$-learning perform similarly. Finally, some experiments were carried out to discover the relevance of different parts of the standard sampling procedure and MPE computation. It was shown that both the weights of a sum node and the likelihood are both important, and most effective in the provably unbiased combination.

In conclusion, it is apparent that the learned guide provides an improvement over standard sampling at the cost of relatively cheap tabular $Q$-learning with good convergence properties. However, there is much work to be done for improving the reinforcement learning setting with better path encodings, scaling the guide learning to larger SPNs, and more.

---

One limitation that was discussed is that the state that the guide can base decisions on does not incorporate information that is modeled in parts of the SPNs that come later in the top-down left-to-right traversal. An obvious remedy would be to traverse the states in a different order, where for example sorting them by the fraction of RVs in the scope having evidence would perhaps be a good first choice. In that case, however, the state would need to carry that information too, e.g. by interspersing child indices of the product nodes that

are currently traversed, and not only child indices of sum nodes anymore. While it is a promising idea, it would, however, lead to a further explosion of the already large state-space. This further adds to the very central problem of increasing the number of states, which currently limits the size of the SPN structures to complexities often insufficient for real-world applications. As $Q$ tables scale linearly in the number of states and that number can increase exponentially in certain SPN structures, it seems promising to employ function approximators like neural networks. They might perform better due to the discovery of hidden symmetries and redundancies in the path information, e.g. by implicitly finding groups of sub-paths that result in similar predictions later in the SPN. Recurrent neural networks might also be a very natural choice for the sequential nature of the path traversals, which were initially investigated, but the approach was scaled back to first obtain a working system using tabular $Q$-learning. This is the main avenue for improvement. The usually data-hungry deep neural network learning methods are particularly applicable here since there is an almost unlimited supply of training data as long as there are sufficient instances from the data domain, although each data instance can produce many paths. One might also consider using experience replay to stabilize learning of the function approximators. Another approach would be to learn an ensemble of guides, for example by stacking, each with different path encodings corresponding to different sum child orders.

Typically in deep generative modeling of images using connectionist models, mostly unconditional sampling (as in plain GANs) and also different types of target metrics like FID and KID scores are used. They could then also be employed as a reward for learning a guide. They were not used in our evaluation since they were deemed less specific reward signals due to requiring batching. Furthermore, the appropriateness of the measure in particular on the small and grayscale MNIST and Fashion-MNIST datasets is doubtful. Alternatively, a GAN-style discriminator model could be learned to provide a reward signal, possibly providing a joint framework for both conditional and unconditional sampling.

Using unconditional sampling with suitable different losses/rewards would also enable the consideration of arbitrary structures, and not only very regular ones as was described when exploring to use SPNs generated using the LearnSPN structure and parameter learner. This might also be interesting in discovering how much the human bias in the assembly of the structure of SPNs might influence the difficulty of sampling. Of course, structure learning algorithms more or less subtly induce bias towards certain structures too but that one is deemed to be much less severe due to the diverse structure generated by many learners. Validating this claim could be an interesting future research topic.

Another consideration is the applicability of the method and empirical conclusions to other domains that are not images. We reckoned that the main properties depend more on the SPN structure than the actual data domain since the guidance learning never has direct access to the images used in this thesis. This claim should, however, still be backed by further experiments.

Finally, further theoretic analysis of the sampling routine might provide more justification for the approach. For this, one could continue where Section 3.4 stopped and attempt to show that the guided sampling procedure is indeed more similar to the data distribution than the SPN itself. As part of that, the delicate interplay of aspects of both sampling and MPE needs to be investigated further. Maybe, when using MPE reconstructions as the base for training rewards, a guide can also produce a good sampling guide, as the reverse seems to be the case.

# Appendix

This chapter provides supplementary material for select topics as referenced in the main part.

## A.1 Analytical Solution for Sampling from the Synthetic Dataset

We want to show that in the synthetic data scenario of Section 4.1 in expectation we will see a mean reward of just below $-250$ when training and evaluating on datasets of sufficient size. Recall that the expected error is $\mathbb{E}\left[\text{MSE}(x, x_{\text{true}})\right] = \mathbb{E}\left[\|x - x_{\text{true}}\|_2^2\right] = \mathbb{E}\left[(x - x_{\text{true}})^2\right]$, where the expectation is over the reconstruction $x$ being sampled from an SPN child and $x_{\text{true}}$ being sampled from the data distribution $p_d$. We also note that we always only have to reconstruct a single variable as the other one is given as evidence, allowing us to simplify the norm as done above. For now, let us ignore that we also usually normalize by the number of RVs, where in this case we would divide by 2. We also know that both $x$ and $x_{\text{true}}$ are Gaussian RVs, so we have:

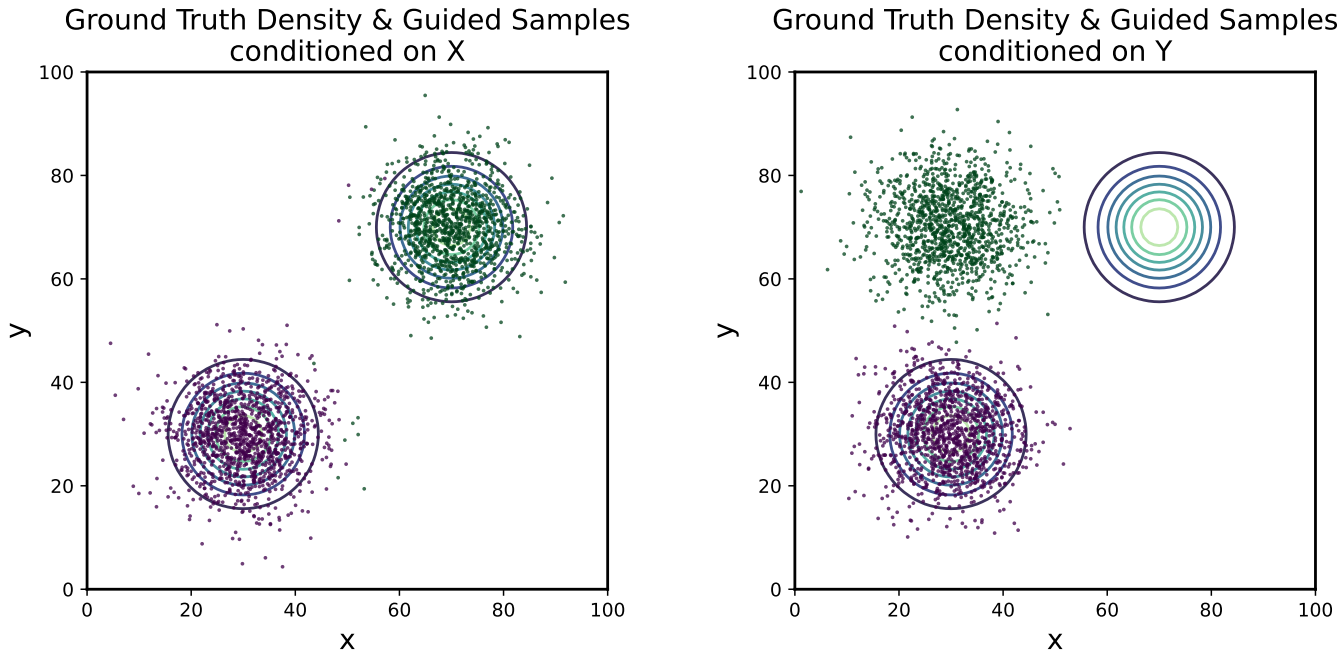
$$
\begin{aligned}
\mathbb{E}\left[\text{MSE}\right] &= \mathbb{E}_{x \sim \mathcal{N}(\mu_a, \sigma_a^2),\, x_{\text{true}} \sim \mathcal{N}(\mu_b, \sigma_b^2)} \left[(x - x_{\text{true}})^2\right] \\
&\overset{\text{(Gaussians are symmetric)}}{=} \mathbb{E}_{x \sim \mathcal{N}(\mu_a, \sigma_a^2),\, x' \sim \mathcal{N}(-\mu_b, \sigma_b^2)} \left[(x + x')^2\right] \\
&\overset{\substack{\text{(Sums of Gaussian RVs} \\ \text{are again Gaussian)}}}{=} \mathbb{E}_{z \sim \mathcal{N}(\mu_a - \mu_b, \sigma_a^2 + \sigma_b^2)} \left[z^2\right] \\
&\overset{\left(\text{Var}(V) = \mathbb{E}[V^2] - \mathbb{E}[V]^2\right)}{=} \mathbb{E}_{z \sim \mathcal{N}(\mu_a - \mu_b, \sigma_a^2 + \sigma_b^2)} \left[z\right]^2 + \text{Var}(Z) \\
&= (\mu_a - \mu_b)^2 + \sigma_a^2 + \sigma_b^2
\end{aligned}
$$

This also holds for $y$ and $y_{\text{true}}$, respectively.

Suppose that we knew the ideal guide based on the optimal $q_*$, which we indeed successfully approximated as can be seen by the fact that learning in this simplistic setting has converged. Then we would reconstruct one of the variables in the query $\boldsymbol{Q}$ and the other one is therefore given as part of the evidence $\boldsymbol{E}$. Four cases (i)–(iv) can occur:

- $\boldsymbol{E} = \{X\}$ **and** $\boldsymbol{Q} = \{Y\}$ (see Figure A.1a):

  (i) $x \sim \mathcal{N}_0$: Then we would condition on $X$ and presumably land in the state $[0, ]$. We ignore the small possibility of choosing the wrong child since for this to happen, $x - \mu_0$ would have to be unlikely large. According to the first line of Table 4.1 we would choose the first child since it has maximum expected reward, and according to $q_*$ we would probably choose the same since that choice will produce a consistent sample. Then $\mathbb{E}\left[\text{MSE}\right] = (\mu_0 - \mu_1)^2 + \sigma_0^2 + \sigma_1^2 = (30 - 30)^2 + 50 + 50 = 100$.




(a) Reconstructions of $Y$ given $X$.

(b) Reconstructions of $X$ given $Y$.

Figure A.1: This contrasts how different occurrences of the conditioning evidence $\boldsymbol{E}$ in the path encoding may result in vastly different samples. In this concrete SPN from Figure 4.1b, $X$ is determined before $Y$. Therefore, when reconstructing $Y$ given $X$ in (a) the results are consistent, but the other way around in (b) they are only sometimes, as discussed in the analytical inspection above.

(ii) $x \sim \mathcal{N}_1$: This is analogous to the previous case, where we consistently sample from the second cluster and have $\mathbb{E}\left[\text{MSE}\right] = (70-70)^2 + 50 + 50 = 100$.

- $\boldsymbol{E} = \{Y\}$ **and** $\boldsymbol{Q} = \{X\}$ (see Figure A.1b):

(iii) $y \sim \mathcal{N}_0$: Then according to the first line of Table 4.1 we would choose the second child, or according to $q_*$ we would probably choose each child with equal probability since we have no way of knowing better in state $[\,]$. In any case, half of the time this is consistent by accident, and half of the time it is not. This results in $\mathbb{E}\left[\text{MSE}\right] = (30-70)^2 + 50 + 50 = 1700$ and $\mathbb{E}\left[\text{MSE}\right] = (30-30)^2 + 50 + 50 = 100$, respectively. In expectation over both scenarios, we have an MSE of $(1700 + 100)/2 = 900$.

(iv) $y \sim \mathcal{N}_1$: This is analogous to the previous case, that is $\mathbb{E}\left[\text{MSE}\right] = 900$.

In summary, since each of the four cases occurs with equal probability, we have an MSE of $(100 + 100 + 900 + 900)/4 = 500$. Since we normalize by 2 this results in the expected reward of $r = -500/2 = -250$. Note that in reality, the expected reward will be ever so slightly lower by tendency, as there is still the small possibility of the dataset containing a sample that lies near the data points of the other cluster than it was drawn from. This can occur since multivariate Gaussians, be they isotropic or not, will give even areas of such unlikely events non-zero density.

## A.2 Additional Results of the Quantitative Comparison

In the quantitative comparison of standard and guided sampling in Section 4.6.1, only relative improvements from introducing guidance for MPE computations are provided. This appendix provides the equivalent of the absolute values comparison in Figure 4.7 but for MPE. Both figures share the same vertical scale and data point layout for easy comparison.

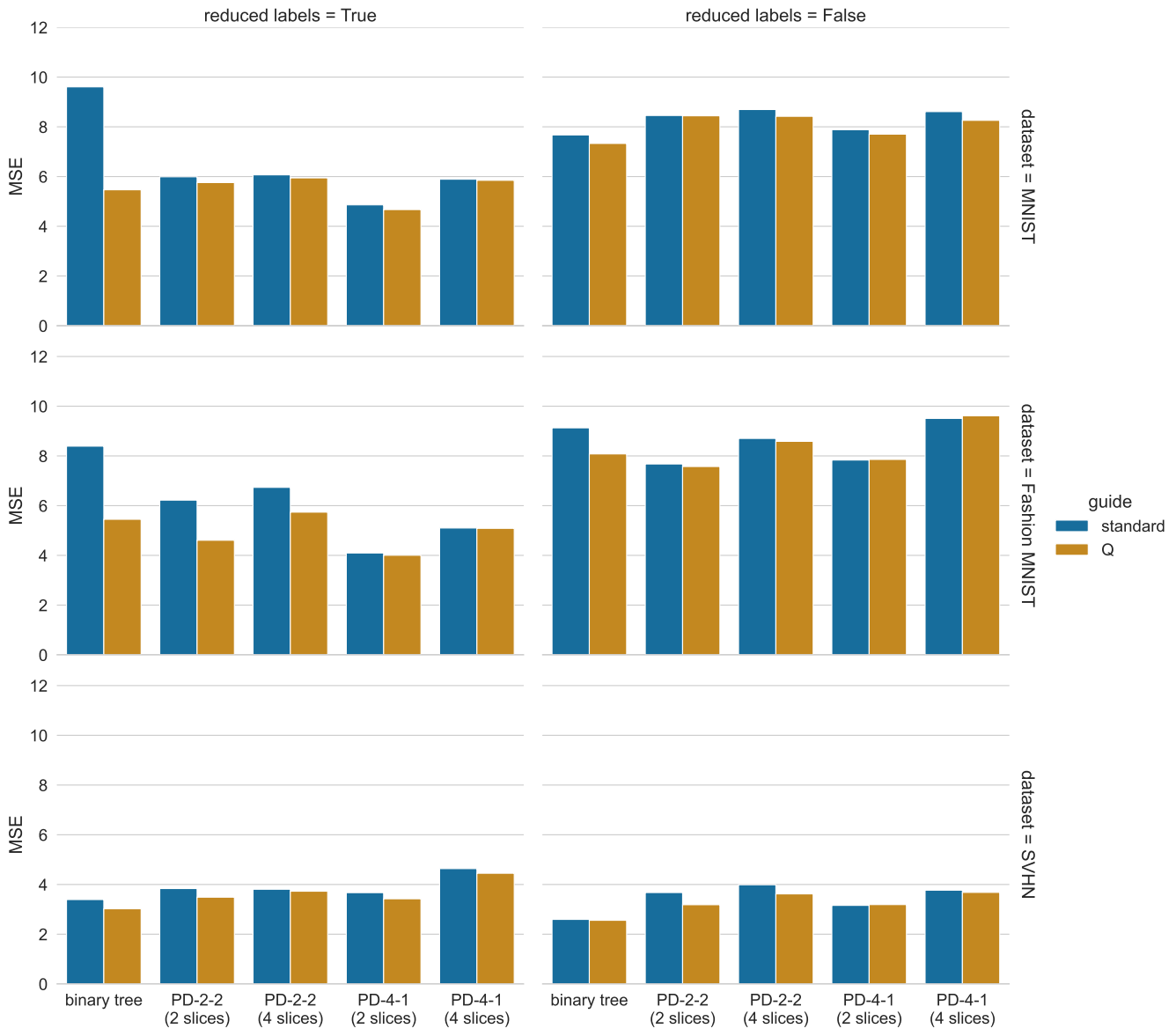



Figure A.2: This figure compares the mean squared error (MSE) when using standard and guided ("Q") MPE computation over several datasets and two reduction variants. The values were obtained by performing 2000 MPE operations. Lower values are better since they correspond to more accurate reconstructions.

## A.3 Additional Results of Exploring Variations of Conditioning in Sampling and MPE

The different conditioning modes are explained in Section 4.7, where relative comparisons are provided in Figures 4.13 and 4.14, respectively. This section provides the absolute values of the mean rewards for each of the combinations. The two visualizations shown below both share the same vertical MSE scale and data point layout for easy comparison.

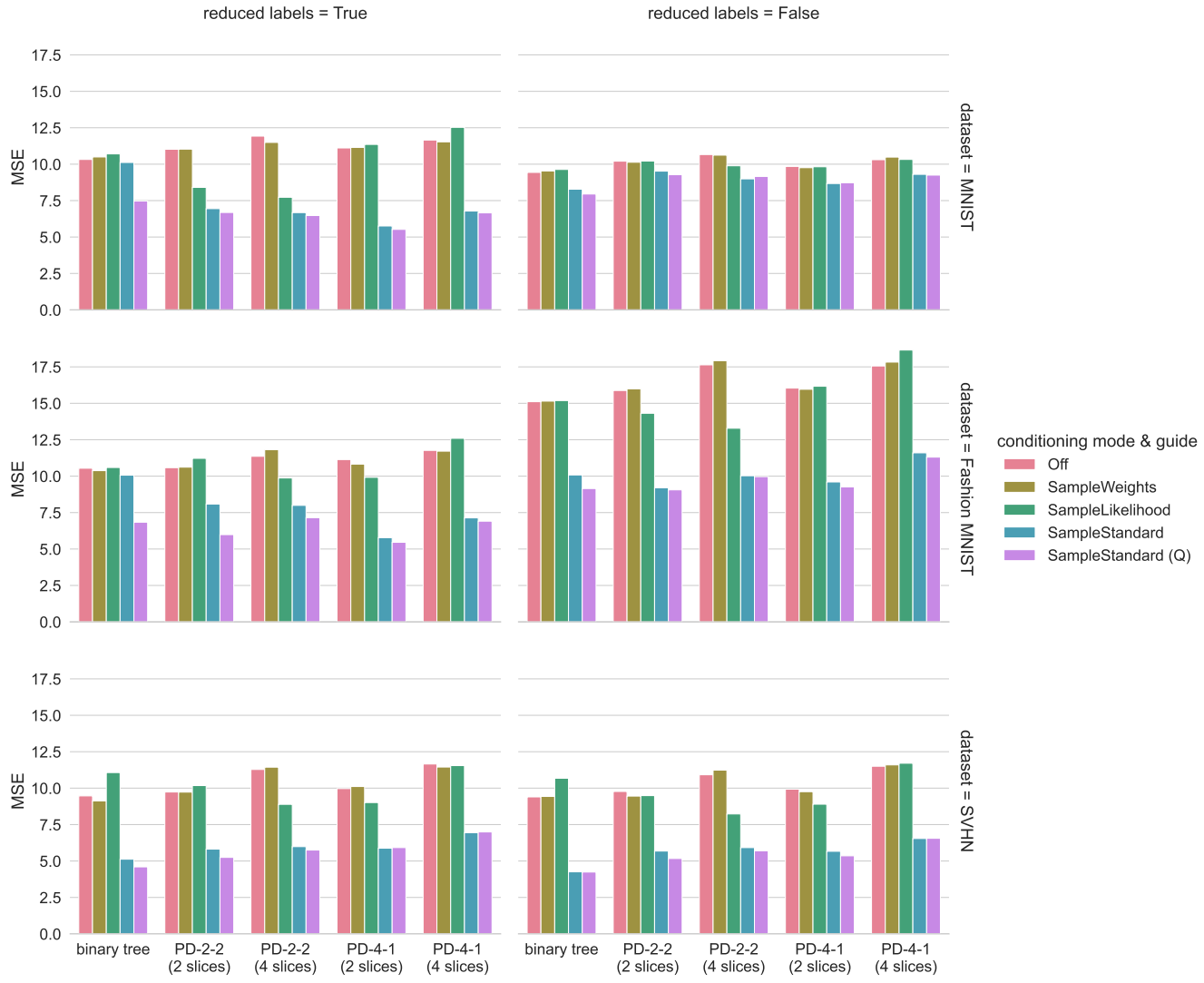



Figure A.3: This comparison shows how different variations of the conditioning in the sampling affect the reward on various datasets, SPN structures, and occlusions. The mean reward is taken over 2000 random reconstructions. Lower is better.

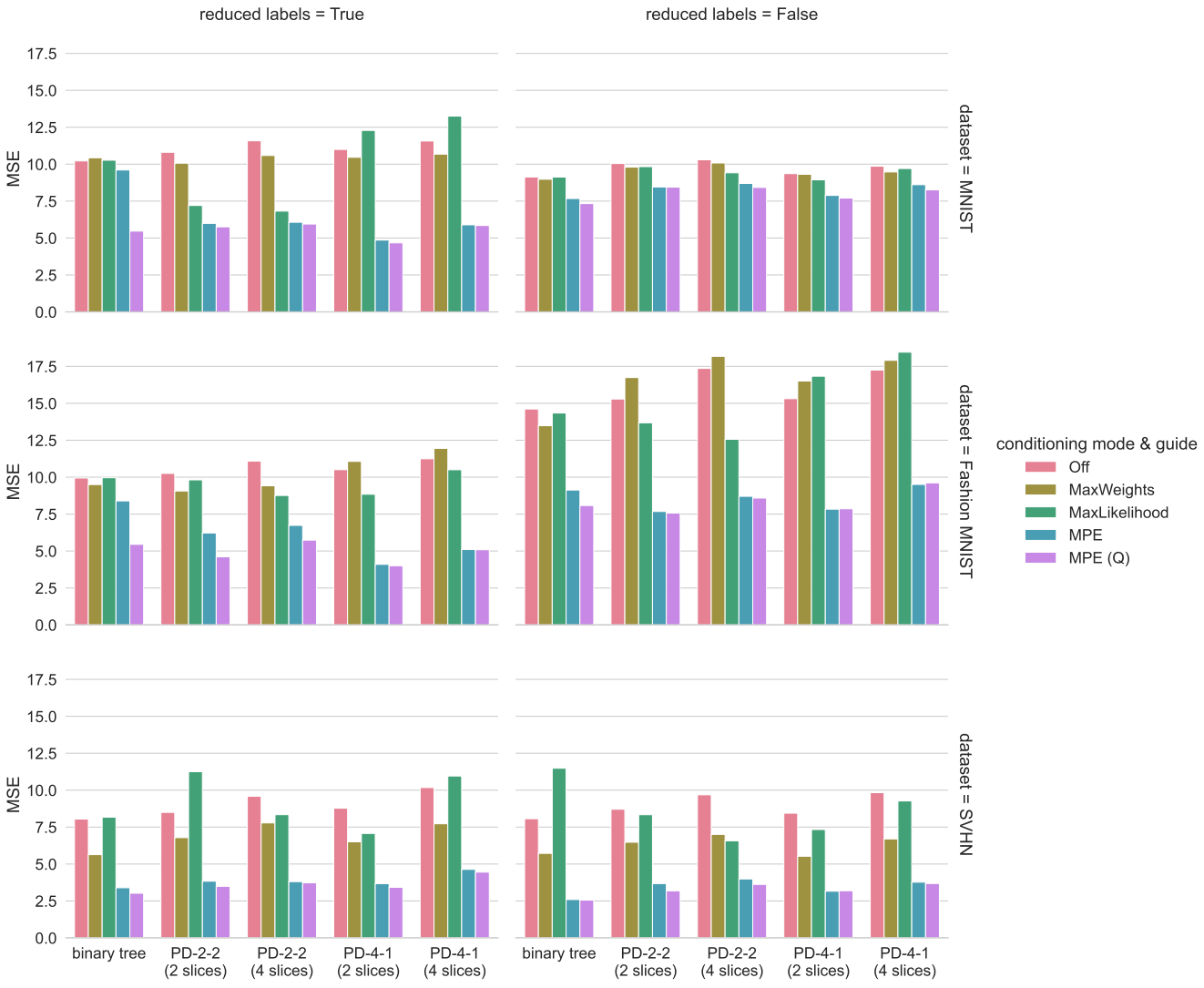



Figure A.4: This comparison shows how different variations of the conditioning in the MPE procedure affect the reward on various datasets, SPN structures, and occlusions. The mean reward is taken over $2000$ random samples. Lower is better.

# Literature


[1] Pieter Abbeel and Dan Klein. *Lecture notes on CS188 Fall 2013. Artificial Intelligence: Lecture 16, Bayes Nets IV, Sampling*. Recordings: https://people.eecs.berkeley.edu/~russell/classes/cs188/f14/lecture_videos.html#Fall2013, accessed 18th March 2022. Berkeley, CA, USA, Oct. 2013.

[2] Ho Bae et al. "AnomiGAN: Generative adversarial networks for anonymizing private medical data". In: *Pacific Symposium on Biocomputing*. World Scientific. 2019, pp. 563–574. DOI: 10.1142/9789811215636_0050. arXiv: 1901.11313 [cs.CR].

[3] Bowen Baker et al. "Designing Neural Network Architectures using Reinforcement Learning". In: *International Conference on Learning Representations (ICLR)* (Nov. 2017).

[4] Kevin Beyer et al. "When Is "Nearest Neighbor" Meaningful?" In: *Database Theory – ICDT'99*. Ed. by Catriel Beeri and Peter Buneman. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 217–235. ISBN: 978-3-540-49257-3. DOI: 10.1007/3-540-49257-7_15.

[5] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006.

[6] Sam Bond-Taylor et al. "Deep Generative Modelling: A Comparative Review of VAEs, GANs, Normalizing Flows, Energy-Based and Autoregressive Models". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021). ISSN: 1939-3539. DOI: 10.1109/tpami.2021.3116668. arXiv: 2103.04922 [cs.LG].

[7] Christopher Bowles et al. *GAN Augmentation: Augmenting Training Data using Generative Adversarial Networks*. 2018. arXiv: 1810.10863 [cs.CV].

[8] Nicholas Carlini et al. "Extracting Training Data from Large Language Models". In: *USENIX Security Symposium*. 2021. arXiv: 2012.07805 [cs.CR].

[9] Wei-Chen Cheng et al. "Language modeling with sum-product networks". In: *Fifteenth Annual Conference of the International Speech Communication Association, INTERSPEECH*. Jan. 2014, pp. 2098–2102. DOI: 10.21437/interspeech.2014-476.

[10] Michael Chui et al. "Notes from the AI frontier: Insights from hundreds of use cases". In: *McKinsey Global Institute* (2018), p. 28.

[11] Antonia Creswell et al. "Generative Adversarial Networks: An Overview". In: *IEEE Signal Processing Magazine* 35.1 (Jan. 2018), pp. 53–65. ISSN: 1053-5888. DOI: 10.1109/msp.2017.2765202. arXiv: 1710.07035 [cs.CV].

[12] Adnan Darwiche. "A Differential Approach to Inference in Bayesian Networks". In: (June 2000). Ed. by Craig Boutilier and Moisés Goldszmidt, pp. 123–132. DOI: 10.5555/2073946.2073962.

[13] Stefano Ermon. *Lecture notes on CS228 Winter 2021–22. Probabilistic Graphical Models: Sampling-based inference*. Online resource: https://ermongroup.github.io/cs228-notes/inference/sampling/, accessed 18th March 2022. Stanford, CA, USA, 2021.

[14] Damien Francois, Vincent Wertz, and Michel Verleysen. “The Concentration of Fractional Distances”. In: *IEEE Transactions on Knowledge and Data Engineering* 19.7 (July 2007), pp. 873–886. ISSN: 1558-2191. DOI: 10.1109/TKDE.2007.1037.

[15] Robert Gens and Pedro Domingos. “Discriminative Learning of Sum-Product Networks”. In: *Advances in Neural Information Processing Systems (NIPS)*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper/2012/file/573f7f25b7b1eb79a4ec6ba896debefd-Paper.pdf.

[16] Robert Gens and Domingos Pedro. “Learning the Structure of Sum-Product Networks”. In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, June 2013, pp. 873–880. URL: https://proceedings.mlr.press/v28/gens13.html.

[17] Zoubin Ghahramani. “Probabilistic machine learning and artificial intelligence”. In: *Nature* 521.7553 (2015), pp. 452–459. DOI: 10.1038/nature14541.

[18] Harshvardhan GM et al. “A comprehensive survey and analysis of generative models in machine learning”. In: *Computer Science Review* 38 (2020), p. 100285. ISSN: 1574-0137. DOI: 10.1016/j.cosrev.2020.100285. URL: https://www.sciencedirect.com/science/article/pii/S1574013720303853.

[19] Vibhav Gogate and Rina Dechter. “SampleSearch: A Scheme that Searches for Consistent Samples”. In: *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*. Ed. by Marina Meila and Xiaotong Shen. Vol. 2. Proceedings of Machine Learning Research. San Juan, Puerto Rico: PMLR, Mar. 2007, pp. 147–154. URL: https://proceedings.mlr.press/v2/gogate07a.html.

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. DOI: 10.1038/nature14539. URL: http://www.deeplearningbook.org.

[21] Ian Goodfellow et al. “Generative Adversarial Networks”. In: *Communications of the ACM* 63.11 (Oct. 2020), pp. 139–144. ISSN: 0001-0782. DOI: 10.1145/3422622.

[22] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. DOI: 10.48550/arXiv.1406.2661. arXiv: 1406.2661 [stat.ML].

[23] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2.

[24] Hado Hasselt. “Double Q-learning”. In: *Advances in Neural Information Processing Systems (NIPS)*. Ed. by J. Lafferty et al. Vol. 23. Curran Associates, Inc., 2010. URL: https://proceedings.neurips.cc/paper/2010/file/091d584fced301b442654dd8c23b3fc9-Paper.pdf.

[25] Matteo Hessel et al. “Rainbow: Combining Improvements in Deep Reinforcement Learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 32.1 (Apr. 2018). URL: https://ojs.aaai.org/index.php/AAAI/article/view/11796.

[26] Tero Karras, Samuli Laine, and Timo Aila. *A Style-Based Generator Architecture for Generative Adversarial Networks*. 2019. DOI: 10.1109/cvpr.2019.00453. arXiv: 1812.04948 [cs.NE].

[27] Diederik P Kingma and Max Welling. “Auto-encoding variational bayes”. In: (2013). arXiv: 1312.6114 [stat.ML].

[28] Ivan Kobyzev, Simon J.D. Prince, and Marcus A. Brubaker. “Normalizing Flows: An Introduction and Review of Current Methods”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.11 (Nov. 2021), pp. 3964–3979. ISSN: 1939-3539. DOI: 10.1109/tpami.2020.2992934. arXiv: 1908.09257 [stat.ML].

[29] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. Cambridge, Massachusetts, USA/London, England, UK: MIT press, 2009. ISBN: 978-0-262-01319-2.

[30] Steven Lang et al. “Elevating Perceptual Sample Quality in Probabilistic Circuits through Differentiable Sampling”. 2021.

[31] Yann LeCun, Corinna Cortes, and CJ Burges. “MNIST handwritten digit database”. In: *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist* 2 (2010).

[32] Quentin Lhoest et al. “Datasets: A Community Library for Natural Language Processing”. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 175–184. DOI: 10.18653/v1/2021.emnlp-demo.21. arXiv: 2109.02846 [cs.CL]. URL: https://aclanthology.org/2021.emnlp-demo.21.

[33] David Lopez-Paz, Philipp Hennig, and Bernhard Schölkopf. “The Randomized Dependence Coefficient”. In: *Proceedings of the 26th International Conference on Neural Information Processing Systems-Volume 1*. 2013, pp. 1–9.

[34] Daniel Lowd and Pedro Domingos. “Approximate Inference by Compilation to Arithmetic Circuits”. In: *Advances in Neural Information Processing Systems*. Ed. by J. Lafferty et al. Vol. 23. Curran Associates, Inc., Nov. 2010, pp. 1477–1485. URL: https://proceedings.neurips.cc/paper/2010/file/addfa9b7e234254d26e9c7f2af1005cb-Paper.pdf.

[35] Wes McKinney. “Data Structures for Statistical Computing in Python”. In: *Proceedings of the 9th Python in Science Conference. SciPy 2010*. Python in Science Conference (Austin, Texas). Ed. by Stéfan van der Walt and Jarrod Millman. Proceedings of the Python in Science Conference. SciPy, June 2010, pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

[36] Jun Mei, Yong Jiang, and Kewei Tu. “Maximum A Posteriori Inference in Sum-Product Networks”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 32.1 (Apr. 2018). URL: https://ojs.aaai.org/index.php/AAAI/article/view/11550.

[37] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602 [cs.LG].

[38] Alejandro Molina et al. *SPFlow: An Easy and Extensible Library for Deep Probabilistic Learning using Sum-Product Networks*. 2019. eprint: arXiv:1901.03704.

[39] Martin Mundt, Iuliia Pliushch, and Visvanathan Ramesh. “Neural Architecture Search of Deep Priors: Towards Continual Learning Without Catastrophic Interference”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. June 2021, pp. 3523–3532. DOI: 10.1109/CVPRW53098.2021.00391.

[40] Yuval Netzer et al. “Reading Digits in Natural Images with Unsupervised Feature Learning”. In: *Advances in Neural Information Processing Systems (NIPS)* (2011). URL: https://ai.stanford.edu/~twangcat/papers/nips2011_housenumbers.pdf.

[41] Frank Nielsen and Vincent Garcia. *Statistical exponential families: A digest with flash cards*. 2011. arXiv: 0911.4863v2 [cs.LG].

[42] Achraf Oussidi and Azeddine Elhassouny. “Deep generative models: Survey”. In: *2018 International Conference on Intelligent Systems and Computer Vision (ISCV)*. Apr. 2018, pp. 1–8. DOI: 10.1109/ISACV.2018.8354080.

[43] Iago París, Raquel Sánchez-Cauce, and Francisco Javier Díez. *Sum-product networks: A survey*. 2020. DOI: 10.1109/tpami.2021.3061898. arXiv: 2004.01167 [cs.LG].

[44] Gaurav Parmar, Richard Zhang, and Jun-Yan Zhu. *On Aliased Resizing and Surprising Subtleties in GAN Evaluation*. 2021. DOI: 10.48550/ARXIV.2104.11222v3. arXiv: 2104.11222v3 [cs.CV]. URL: https://arxiv.org/abs/2104.11222v3.

[45] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems (NIPS)*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: https://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[46] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[47] Robert Peharz. “Foundations of Sum-Product Networks for Probabilistic Modeling”. PhD thesis. Graz: Graz University of Technology, Austria, Feb. 2015. URL: https://diglib.tugraz.at/download.php?id=576a7b8cc939f&location=browse.

[48] Robert Peharz et al. “Einsum Networks: Fast and Scalable Learning of Tractable Probabilistic Circuits”. In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, July 2020, pp. 7563–7574. URL: https://proceedings.mlr.press/v119/peharz20a.html.

[49] Robert Peharz et al. “On the Latent Variable Interpretation in Sum-Product Networks”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.10 (Jan. 2016), pp. 2030–2044. DOI: 10.1109/TPAMI.2016.2618381. arXiv: 1601.06180.

[50] Robert Peharz et al. “Random Sum-Product Networks: A Simple and Effective Approach to Probabilistic Deep Learning”. In: *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*. Ed. by Ryan P. Adams and Vibhav Gogate. Vol. 115. Proceedings of Machine Learning Research. PMLR, July 2020, pp. 334–344. URL: https://proceedings.mlr.press/v115/peharz20a.html.

[51] Bryan Edward Penprase. “The fourth industrial revolution and higher education”. In: *Higher education in the era of the fourth industrial revolution* 10 (2018), pp. 978–981. DOI: 10.1007/978-981-13-0194-0_9.

[52] Esteban Piacentino and Cecilio Angulo. “Anonymizing personal images using generative adversarial networks”. In: *International Work-Conference on Bioinformatics and Biomedical Engineering*. Springer. 2020, pp. 395–405. DOI: 10.1007/978-3-030-45385-5_35.

[53] Esteban Piacentino, Alvaro Guarner, and Cecilio Angulo. “Generating Synthetic ECGs Using GANs for Anonymizing Healthcare Data”. In: *Electronics* 10.4 (2021), p. 389. DOI: 10.3390/electronics10040389.

[54] Hoifung Poon and Pedro Domingos. *Sum-Product Networks: A New Deep Architecture*. 2012. DOI: 10.1109/iccvw.2011.6130310. arXiv: 1202.3732 [cs.LG].

[55] Xiaoting Shao et al. “Conditional sum-product networks: Modular probabilistic circuits via gate functions”. In: *International Journal of Approximate Reasoning* 140 (2022), pp. 298–313. ISSN: 0888-613X. DOI: 10.1016/j.ijar.2021.10.011. URL: https://www.sciencedirect.com/science/article/pii/S0888613X21001766.

[56] Connor Shorten and Taghi M. Khoshgoftaar. “A survey on image data augmentation for deep learning”. In: *Journal of Big Data* 6.1 (2019), pp. 1–48. DOI: 10.1186/s40537-019-0197-0.

[57] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. Second. Cambridge, Massachusetts, USA; London, England, UK: MIT press, 2018. ISBN: 9780262039246. DOI: 10.1109/tnn.1998.712192. URL: http://incompleteideas.net/book/RLbook2020.pdf.

[58] Lucas Theis, Aäron van den Oord, and Matthias Bethge. “A note on the evaluation of generative models”. In: *International Conference on Learning Representations (ICLR)*. Apr. 2016. arXiv: 1511.01844 [stat.ML].

[59] Jakub M. Tomczak. *Deep Generative Modeling*. Springer International Publishing, Feb. 2022. ISBN: 9783030931575. DOI: 10.1007/978-3-030-93158-2.

[60] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.

[61] Antonio Vergari, Nicola Di Mauro, and Floriana Esposito. “Simplifying, Regularizing and Strengthening Sum-Product Network Structure Learning”. In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by Annalisa Appice et al. Cham: Springer International Publishing, 2015, pp. 343–358. ISBN: 978-3-319-23525-7. DOI: 10.1007/978-3-319-23525-7_21.

[62] Antonio Vergari, Nicola Di Mauro, and Guy Van den Broeck. “Tutorial on Tractable Probabilistic Models: Representations, Algorithms, Learning, and Applications”. In: *Conference on Uncertainty in Artificial Intelligence (UAI)*. Accessed on Dec. 06, 2020. Recording available (LINK). Tel Aviv, July 2019. URL: https://web.archive.org/web/20191210113543/http://auai.org/uai2019/tutorials.php#tutorial1.

[63] Antonio Vergari et al. “Tutorial on Probabilistic Circuits: Representations, Inference, Learning and Applications”. In: *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD)*. Accessed on Dec. 06, 2020. Recording available (LINK). July 2020. URL: https://web.cs.ucla.edu/~guyvdb/talks/ECML-PKDD20-tutorial/.

[64] Pauli Virtanen et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[65] Christopher J. C. H. Watkins and Peter Dayan. “Q-learning”. In: *Machine learning* 8.3 (1992), pp. 279–292. DOI: 10.1007/bf00992698.

[66] Han Xiao, Kashif Rasul, and Roland Vollgraf. “Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms”. In: *Computing Research Repository (CoRR)* (2017). arXiv: 1708.07747. URL: http://arxiv.org/abs/1708.07747.

[67] Jinsung Yoon, Lydia N. Drumright, and Mihaela van der Schaar. “Anonymization Through Data Synthesis Using Generative Adversarial Networks (ADS-GAN)”. In: *IEEE Journal of Biomedical and Health Informatics* 24.8 (Aug. 2020), pp. 2378–2388. ISSN: 2168-2208. DOI: 10.1109/JBHI.2020.2980262.

[68] Fisher Yu et al. *LSUN: Construction of a Large-scale Image Dataset using Deep Learning with Humans in the Loop*. 2016. arXiv: 1506.03365 [cs.CV].

[69] Jiahui Yu et al. “Free-Form Image Inpainting With Gated Convolution”. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 4470–4479. DOI: 10.1109/ICCV.2019.00457. arXiv: 1806.03589 [cs.CV].