Advances in Understanding and Initializing Einsum Networks

Smits, Julian

Award date:
2021

Department of Mathematics and Computer Science
Uncertainty in Artificial Intelligence Research Group

Advances in Understanding and Initializing Einsum Networks

Master Thesis

Julian Smits

Supervisor:
Dr. habil. Cassio de Campos

Assessment Committee:
Dr. habil. Cassio de Campos
Dr. Sibylle Hess
Dr. Yali Du

Eindhoven, July 26, 2021

Abstract

Exact and tractable inference is the reason why probabilistic circuits are such an interesting field of research. With Einsum Networks, probabilistic circuits achieve much faster training times. However the problem of proper initialization has never been researched before in the context of probabilistic circuits. To perform robust research, this work firstly aims at achieving a better understanding of probabilistic circuits by performing multiple experiments. The second goal of improving on the current state of the art is achieved by introducing new initialization algorithms and online training algorithms.

Constructing a good machine learning setup for a certain task is very important. Therefore different training approaches and network structures are tested on multiple tasks. This gives insight in the best setups for either a generative task, a discriminative task or even a combination of both. One major finding is that the so-called class discriminative structure does greatly improve the discriminative task without harming the ability to learn a generative task.

Initialization has been a topic with very promising results in the field of neural networks. This is the reason why initialization has been further explored in the context of Einsum Networks. The algorithms proposed in this work improve the initial state of the Einsum Networks by a lot. This is achieved by a hierarchical clustering of the training samples to initialize the weights of the Einsum Networks.

Further improving Einsum Networks after which they are trained is called online training. The novel idea of using the initialization of a network to later on process new training samples, with the goal to improve the performance of the network, is presented in this work. The parameters of the Einsum Networks are updated by using the hierarchical clustering, executed during the initialization, to assign the new samples to certain regions of the Einsum Networks and update the weights coherently. This approach excels in scenarios where there is a lack of resources and time to completely retrain the Einsum Network.

Contents

Contents **v**

List of Figures **vii**

1 Problem Statement and Research Context **1**
1.1 Introduction . . . 1
1.2 Problem Statement . . . 1
1.3 Probabilistic Circuits . . . 2
1.4 Parameter Learning . . . 6
1.5 Structure Learning . . . 6
1.6 Einsum Networks . . . 8

2 Literature Review **11**
2.1 General Theory . . . 11
2.2 Structure Learning . . . 12
2.3 Parameter Learning . . . 14
2.4 Extensions . . . 15
2.5 Applications . . . 16
2.6 Context to this Research . . . 17

3 Network Analysis **19**
3.1 Motivation . . . 19
3.2 RAT-SPN Structure . . . 19
3.2.1 Setup . . . 20
3.2.2 Experiments . . . 20
3.2.3 Discussion . . . 22
3.3 Training Approaches . . . 22
3.3.1 Setup . . . 22
3.3.2 Experiments . . . 24
3.3.3 Impressions . . . 25
3.3.4 Observations . . . 26
3.3.5 Discussion . . . 27

4 Initialization **29**
4.1 Motivation . . . 29
4.2 Algorithms . . . 29
4.3 Experiments . . . 32
4.4 Discussion . . . 36

5 Online Adaptation **37**
5.1 Motivation 37
5.2 Algorithms 37
5.3 Experiments 40
5.4 Minimal Results 44
5.5 LearnSPN 45
5.6 Discussion 48

6 Conclusion and Future Work **49**
6.1 Network Analysis 49
6.2 Initialization 50
6.3 Online Adaptation 50
6.4 Possible Extensions 50

Bibliography **53**

List of Figures

1.1 A single Gaussian distribution 3
1.2 A factorization over three different input distributions 3
1.3 A mixture over two different input distributions 4
1.4 Different combinations of building blocks resulting in complex structures 4

3.1 Results of a RAT-SPN structure with variable width 21
3.2 Results of a RAT-SPN structure with variable depth 21
3.3 Results of a RAT-SPN structure with both increasing depth and width 22
3.4 Results of experiments 1 to 8 28

4.1 Results of testing different initialization algorithms directly after initialization . . . 34
4.2 Results of different initialization methods after every epoch 35

5.1 Results of Online Adaptation for Cluster initialization (45k init, 5k online) 41
5.2 Results of Online Adaptation for Einsum Cluster initialization (45k init, 5k online) 42
5.3 Results of Online Adaptation for Einsum Cluster initialization (45k init, 1k online) 43
5.4 Results of Online Adaptation for Einsum Cluster initialization (1k init, 1k online) 43
5.5 Results of Online Adaptation for Einsum Cluster initialization (2k init, 1k online) 44
5.6 Results of Online Adaptation for LearnSPN on Spoken Arabic Digit dataset 47
5.7 Results of Online Adaptation for LearnSPN on Tamilnadu Electricity dataset . . . 47
5.8 Results of Online Adaptation for LearnSPN on Skin Segmentation dataset 48

Chapter 1

Problem Statement and Research Context

1.1 Introduction

Probabilistic circuits are a promising area of research that have received more and more attention over the last years. Since probabilistic circuits provide a high level of expressiveness accompanied with tractable inference, it is a very interesting and promising research topic in the field of uncertainty in artificial intelligence. Probabilistic circuits have different variances, which are specified by their underlying properties. Examples of this are sum-product networks and Einsum Networks. Research has already been done in the learning of the structure of such probabilistic circuits. However, to achieve the full potential from probabilistic circuits, further research has to be conducted in this topic. This work focuses on research in comparing learning approaches for different tasks, initialization methods and online training methods.

In the further sections of this chapter the problem statement will be further motivated and the research context will be explained. A literature review is presented in chapter 2 of this work. The structure of RAT-SPNs is analyzed Chapter 3 accompanied with the experiments regarding different learning approaches and setups are performed in this chapter. Then different initialization methods are introduced and analyzed in chapter 4. Chapter 5 extends the initialization methods to online learning approaches. Finally, chapter 6 elaborates on conclusions that can be drawn and goes into the possible research directions.

1.2 Problem Statement

Neural networks are a widely accepted model for machine learning because of their high level of expressive power. However, neural networks are not able to do exact tractable inference. Tractable inference is important in solving several probabilistic queries. These queries consists of complete evidence queries (EVI), marginal queries (MAR), conditional queries (CON) and Maximum A posteriori queries (MAP).

Probabilistic circuits (PCs) do solve this problem of tractable inference and are therefore a promising research topic. To make sure that the queries are indeed tractable certain properties have to be specified for PCs. In order to specify these properties it is required to understand how a PC is constructed. This will be explained in the following sections of this chapter. A PC can be seen as a special kind of neural network [29].

In order to improve on the state of the art in probabilistic circuits Peharz et al. specify Einsum Networks (EiNets) as a novel implementation design for a PC [29]. Einsum Networks are a probabilistic circuit which adhere to the smoothness and decomposability properties of PCs and are therefore able to solve tractable MAR and tractable CON queries. The EiNets proposed are open for further improvement in different fields, for example learning the structure of the network, improve the initialization and learning the parameters. In this work most research will be done on Einsum Networks. This choice has been made since EiNets are a novel implementation of PCs. EiNets have shown very promising results in improving learning time and memory consumption, up to two orders of magnitude [29].

This work starts with exploring different different training approaches of Einsum Networks. Probabilistic circuits are generative models by nature. To further explore these models, both generative and discriminative training methods will be tested to gain insight into performance differences. These methods will both be tested on generative and discriminative tasks. These experiments should provide insight into possible optimal setups for the Einsum Networks in regard with training them for a certain task.

Improving the initialization of Einsum Networks will be the second focus. Initialization has been found as a very important aspect in speeding up the training of neural networks [46]. In addition to speeding up the training of neural networks, research has also been conducted on using initialization to avoid local minimum [45]. Since initialization provides such promising results in neural networks this should also be applicable to probabilistic circuits. What initialization means in the context of Einsum Networks is the following. The leaf nodes, which are formed by distributions (i.e. Gaussians or Binomials), should be initialized. In addition to the initialization of the leaf distributions, also the weights of the sum nodes of the network should be initialized.

Finally, online training methods will be explored. This will be done in such a manner that the initialization methods are used to determine online update rules. This is a very interesting topic since fully re-training a model takes too much time in specific cases. In these cases a faster method to improve the model in an online fashion will be beneficial to the task, which is executed by the model.

1.3 Probabilistic Circuits

In this section, probabilistic circuits are explained more thoroughly. The images and formulas from this section are taken from the lecture on probabilistic circuits presented during the European conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases [31].

Distributions as Computational Graphs
In probabilistic circuits a graph is constructed, this graph consists of nodes which all do a certain computation, therefore this is called a computational graph. This means that it is possible to provide an input to the computational graph after which it provides an output for that particular input. This simplest possible computational graph consists of a single distribution. This distribution is modeled by an exponential family which expresses the value of a single random variable. This exponential family can, for example, be a Gaussian distribution. A single distribution should be able to answer certain questions efficiently. The queries that a single distribution is able to answer tractably are the complete evidence (EVI) query, the marginal (MAR) query and the Maximum A Posteriori (MAP) query. For the EVI query the distribution takes an input value and gives back the denstiy or the probability value. An example for the EVI query is the following question, what is the probability of a certain car being 2 meters long? For the MAR query the distribution calculates the integral of the distribution, which is always one if the distribution is normalized. The MAP query, or most probable explanation query, is calculated by computing the mean value of the distribution.

A single distribution is only able to model a single variable and is often depicted as shown in figure 1.1. To answer harder questions, it is desired to model more complex distributions over possibly multiple random variables. Therefore these single distributions are used as input nodes in a larger network to model these more complex distributions.

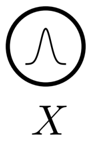

Figure 1.1: A single Gaussian distribution

Factorizations
When modeling a certain distribution over multiple variables, factorizations are needed to manage complexity. A Factorization is able to do this since it can split a distribution over multiple variables into multiple independent distributions of a single variable. More precisely, factorization means that, when a probability over a certain joint probability distribution is known, the assumption can be made that, this probability is computed by multiplying the probabilities of all the single variables in this joint probability distribution. This is mathematically depicted as the following formula, where X_1, X_2 and X_3 are random variables as input distributions:

$$p(X_1, X_2, X_3) = p(X_1) \cdot p(X_2) \cdot p(X_3) \tag{1.1}$$

For this reason the factorizations in a probabilistic circuit are modeled by a product node. The factorization shown in figure 1.2 is a factorization over three different input distributions.

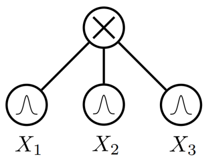

Figure 1.2: A factorization over three different input distributions

Mixtures
Factorizations are able to combine the input distributions and therefore add more variables to our network. Factorizations are not able to make the distributions more complex and fitting. This is where mixtures come in. A mixture combines two distributions over the same random variable into a more complex distribution. This is done by specifying a weight to each of the distributions that are mixed together. Mathematically this is depicted in the following formula, here X is an input distribution over the some random variable, p_1 and p_2 depict two different distributions over this random variable X and w_1 and w_2 are weights corresponding to both of the input distributions.

$$p(X) = w_1 \cdot p_1(X) + w_2 \cdot p_2(X) \tag{1.2}$$

In a probabilistic circuits a mixture is implemented by a sum node. This is shown in figure 1.3. This sum node mixes two input distributions.

Constructing a Circuit and Maintaining Tractability
A single distribution, factorization or mixture itself do not model complex distributions. But when combining multiple of these nodes together into a graph, such a complex distribution can

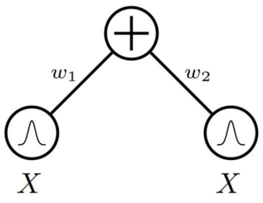

Figure 1.3: A mixture over two different input distributions

be formed. Figure 1.4 shows how the different nodes can be combined into a complex structure.

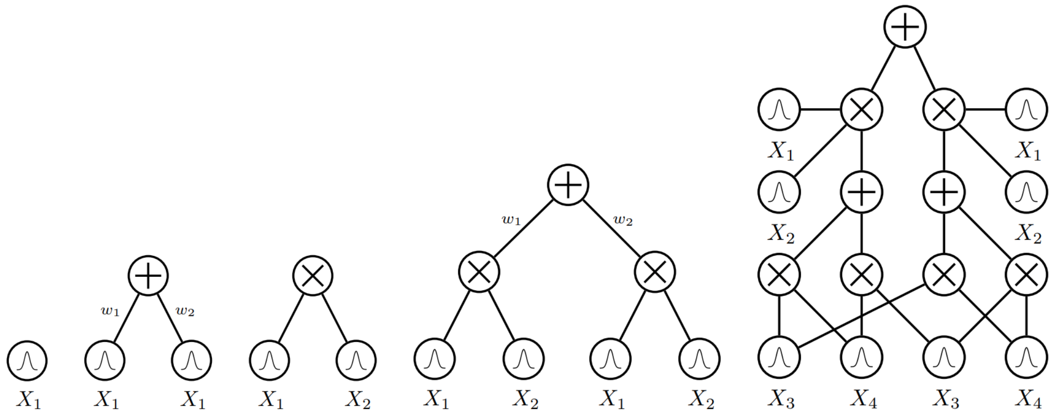

Figure 1.4: Different combinations of building blocks resulting in complex structures

Putting random nodes together in a network results in the model not being tractable anymore. Therefore some structural constraints are necessary to guarantee tractability. These structural constraints are decomposability and smoothness.

The decomposability constraint is a structural constraint over the product nodes. It originates from the factorization mentioned earlier and states that each random variable, which a product node factorizes, should be different. More formally, each child of the product node should have a pairwise disjoint scope. When multiplying the different distributions, over different random variables, a joint probability function arises, which is desired.

The smoothness constraint is a structural constraint over the sum nodes. To generate a mixture from two separate distributions, which a sum node does, both distributions should have identical scopes. Otherwise the distributions cannot be mixed into a more complex distribution. All the networks shown in figure 1.4 adhere to both of these properties, as can be observed.

A MAR query computes the probability of a certain event or multiple events. To calculate a MAR query, it is desired to calculate the integrals over the complete network. This means that both the product nodes and the sum nodes have to be able to calculate the integral in a single step to guarantee tractability.

Due to the decomposability constraint the following formula holds:

$$p(x, y) = p(x)p(y) \tag{1.3}$$

This means, when calculating the integral over this formula, the following steps can be taken:

$$\int\int p(x,y)dxdy = \int\int p(x)p(y)dxdy = \int p(x)dx \int p(y)dy \tag{1.4}$$

What this formula shows is that the integrals are pushed down to the children of the decomposable product node.

For a sum node a similar derivation can be made, only this time using the smoothness structural constraint. Namely due to the smoothess constraint the following formula holds:

$$p(x) = \sum_i w_i p_i(x) \tag{1.5}$$

This means, when calculating the integral over this formula, the following steps can be taken:

$$\int p(x)dx = \int \sum_i w_i p_i(x)dx = \sum_i w_i \int p_i(x)dx \tag{1.6}$$

Again this formula shows that the integrals are pushed down to the children of the smooth sum node.

This pushing down of the integrals can be done in a single step for a single node. The input distributions are also able to calculate the integrals in a single step, as explained earlier. This means that a smooth and decomposable circuit is able to solve MAR queries tractable.

A conditional (CON) query can also be calculated with the same derivations, since a conditional query can be rewritten as two MAR queries. This is done in the following formula:

$$p(x|y) = \frac{p(x,y)}{p(y)} \tag{1.7}$$

Therefore a CON query can also be solved tractably by a smooth and decomposable circuit.

For tractable MAP queries a third constraint is necessary. As explained earlier, the goal of a MAP query is to find out the highest probability, or likelihood of all the random variables for some observation. In mathematics this is expressed by the following formula:

$$\max_x p(x|y) \tag{1.8}$$

This maximum cannot be propagated to the children of a sum node when only adhering to the smoothness and decomposability constraint. This is where the determinism constraint comes in. The determinism constraint is a structural constraint over the sum nodes. It states that only one of the children of the sum nodes can have a non-zero output, for any input. Due to this determinism constraint the following formula holds for a sum node:

$$p(x,y) = \sum_i w_i p_i(x,y) = \max_i w_i p_i(x,y) \tag{1.9}$$

This means, when calculating the MAP query the following steps can be taken:

$$\max_x p(x|y) = \max_x \sum_i w_i p_i(x,y) = \max_x \max_i w_i p_i(x,y) = \max_i \max_x w_i p_i(x,y) \tag{1.10}$$

These steps show that the MAP query can be propagated downwards to the children of a sum node in a single step. The decomposability constraint for product nodes is enough to also propagate the MAP query downwards to the children of this product node. This is done in a similar way as the MAR query. Therefore when a probabilistic circuit also adheres to the determinism structural constraint, it is also able to solve the MAP queries tractably.

1.4 Parameter Learning

Learning the parameters of a probabilistic circuits consists of two parts, namely learning the input distributions and learning the weights in the sum nodes. Both can be learned by a standard back propagation algorithm, for example Stochastic Gradient Descent (SGD). However, there are also other approaches to learn probabilistic circuits. The one which is most popular is Expectation Maximization (EM). This is explained in this paragraph. Expectation Maximization is a more probabilistic circuit tailored approach, which is able to exploit the tractability of PCs due to the structural properties which the PCs adhere to.

Input Distributions
Explaining how probabilistic circuits are learned, starts again at a single distribution. As explained earlier, a single distribution is modeled by an exponential family. The parameters of an exponential family can be learned by performing a maximum likelihood estimation over the data. This consists of simply taking the average, for a parameter of the exponential family, over the complete dataset.

Sum Nodes
As explained earlier, a sum node represents a mixture over multiple input distributions. However, sum nodes can also be interpreted as latent variable models, which is explained by Peharz et al [28]. This means that the sum node models an unknown variable. Now it is desired to apply maximum likelihood learning to the parameters of this single sum node. However, maximum likelihood learning requires to know all the variables in the model and the latent variable in the sum node is not known. This is where Expectation Maximization is introduced, which is essentially maximum likelihood learning under missing data.

Similar to the maximum likelihood learning for exponential families, maximum likelihood learning for sum nodes means counting the number of times a sum node goes in a certain direction and dividing by the number of samples in the dataset. However it is not known exactly how often a sum node goes in a certain direction due to the latent variable interpretation of the sum node. Therefore this hard counting is replaced by a so-called soft counting. The soft counting consists of deriving the latent variable, by calculating the posterior over the data using the current weights of the sum node, and then counting the number of times the sum node goes in a certain direction.

Extension to Probabilistic Circuits
To apply these learning principles to the complete probabilistic circuit, it is also required to know whether or not a certain sum node or input distribution is reached. This is also not know exactly, but this can again be derived by performing a backward pass over the network, using the current weights. In a single backward pass this can be derived for every node in the probabilistic circuit. After which it is known whether or not a sum-node or input distribution is reached, the above explained methods are used to update the weights.

1.5 Structure Learning

Learning the structure of a probabilistic circuit consists of constructing a graph using the sum nodes, product nodes and the input distributions. This can be done in various ways. A lot of research is conducted on this part of the probabilistic circuits. This research is presented in the next chapter of this work.

The structure of a probabilistic circuit is directed acyclic graph (DAG) which consists of sum nodes and product nodes. The leafs of this DAG consist of exponential families which represent the input distributions. These structures can range from very simple to very sophisticated graphs.

The graphs can also be extended to contain cycles, this gives the model more expressiveness while keeping the number of parameters in the model the same. In this work, two different structures are used, namely the structures generated by the RAT-SPN and the LearnSPN algorithms. These are explained in more detail in this section.

RAT-SPN

The RAT-SPN algorithm is introduced by Peharz et al [32]. RAT-SPN stands for random-and-tensorized sum-product network. The RAT-SPN structure performs particularly well when used for generative tasks and shows good results under missing features in the input data.

The algorithm starts with constructing a region graph, the region graph divides the random variables, which the network should model, over the different regions. The region graph consists of regions and partitions. The child of a region is always a partition and vice versa. A region contains all the random variables which that particular region models. A partition is a division of the random variables of the parent region into two balanced groups of random variables. To construct the random region graph the following parameters are required: X: The list of random variables of the complete model, D: the depth of each of the recursions when constructing the Random Region Graph, resulting in an SPN of depth $2D$ and R: the number of recursive splittings in the Random Region Graph. The pseudocode for constructing the Random Region Graph is shown in algorithm 1

Algorithm 1 Random Region Graph

```
1:  procedure RandomRegionGraph($\mathbf{X}$, D, R)
2:      Create an empty region graph $\mathcal{R}$
3:      Insert $\mathbf{X}$ in $\mathcal{R}$
4:      for r = 1...R do
5:          Split($\mathcal{R}$, $\mathbf{X}$, D)
6:      end for

1:  procedure Split($\mathcal{R}$, $\mathbf{R}$, D)
2:      Draw balanced partition $\mathcal{P} = \mathbf{R}_1, \mathbf{R}_2$ of $\mathbf{R}$
3:      Insert $\mathbf{R}_1, \mathbf{R}_2$ in $\mathcal{R}$
4:      Insert $\mathcal{P}$ in $\mathcal{R}$
5:      if D > 1 then
6:          if $|\mathbf{R}_1| > 1$ then
7:              Split($\mathcal{R}$, $\mathbf{R}_1$, D-1)
8:          end if
9:          if $|\mathbf{R}_2| > 1$ then
10:             Split($\mathcal{R}$, $\mathbf{R}_2$, D-1)
11:         end if
12:     end if
```

The second part of constructing the RAT-SPN structure is building the actual SPN from the Random Region Graph. This is done by iterating over the Random Region Graph. For every region in the Random Region Graph, the algorithm either assigns sum nodes or in the case of leaf regions, distribution nodes. The partitions are translated into product nodes. The algorithm requires the following parameters: $\mathcal{R}$: the Random Region Graph, C: the number of root nodes, S: the number of sum nodes per region and I: the number of distributions per leaf region. The algorithm is shown in Algorithm 2

Algorithm 2 Construct SPN from Random Region Graph

```
1: procedure ConstructSPN($\mathcal{R}$, $C$, $S$, $I$)
2:     Construct empty SPN
3:     for $\mathbf{R} \in \mathcal{R}$ do
4:         if $\mathbf{R}$ is a leaf region then
5:             Equip $\mathbf{R}$ with $I$ distribution nodes
6:         else if $\mathbf{R}$ is the root region then
7:             Equip $\mathbf{R}$ with $C$ sum nodes
8:         else
9:             Equip $\mathbf{R}$ with $S$ sum nodes
10:        end if
11:    end for
12:    for $\mathcal{P} = \{\mathbf{R}_1, \mathbf{R}_2\} \in \mathcal{R}$ do
13:        let $\mathbf{N_R}$ be the nodes for region $\mathbf{R}$
14:        for $\mathsf{N}_1 \in \mathbf{N}_{\mathbf{R}_1}$, $\mathsf{N}_2 \in \mathbf{N}_{\mathbf{R}_2}$ do
15:            Introduce product $P = \mathsf{N}_1 \times \mathsf{N}_2$
16:            Let $\mathsf{P}$ be a child for each $\mathsf{N} \in \mathbf{N}_{\mathbf{R}_1 \cup \mathbf{R}_2}$
17:        end for
18:    end for
19:    return SPN
```

LearnSPN
Gens et al. also propose an algorithm for learning the structure of sum-product networks [14]. The algorithm is called LearnSPN. LearnSPN is able to take full advantage of the expressiveness of sum-product networks. The idea behind the algorithm is to partition the variables of the training instances into independant subsets, when this is possible a product node is created. When this is not possible, the training instances are grouped into similar subsets. For these subsets a sum node is created. In both cases the algorithm makes a recursion downwards until only a single variable is left in the training instances, then it creates a leaf distribution. The parameters that the LearnSPN algorithm requires are: T: the set of training instances and V: the set of variables. The pseudocode for LearnSPN is shown in Algorithm 3.

Algorithm 3 LearnSPN(T, V)

```
1: if $|V| = 1$ then
2:     return univariate distribution estimated from the variable's values in $T$
3: else
4:     partition $V$ into approximately independent subsets $V_j$
5:     if success then return $\prod_j$ LearnSPN($T$, $V_j$)
6:     else
7:         partition $T$ into subsets of similar instances $T_i$
8:         return $\sum_i \frac{|T_i|}{|T|} \cdot$ LearnSPN($T_i$, $V$)
9:     end if
10: end if
```

1.6 Einsum Networks

Most experiments in this work are performed in the Einsum Network library [29]. Einsum Networks are a novel implementation of probabilistic circuits proposed by Peharz et al. Einsum Networks aim at improving the learning speed of probabilistic circuits. In Einsum Networks the nodes of the probabilistic circuit are vectorized. This improves the efficiency of the layout of the probabilistic circuit. More concretely, this means that a leaf node is re-defined to be be a vector of

K densities rather than a single density. Here each density still has its own private parameters. A product node is re-defined to be an outer product over its children, hence containing the products of all possible combinations of densities. The sum nodes are re-defined to be a vector of K weighted sums, where each individual sum operation has its private weights. By re-defining the nodes of PCs to their vectorized version in Einsum Networks, the structural properties of smoothness and decomposability are preserved. This means that Einsum Networks are able to solve CON and MAR queries tractably.

This vectorization of the nodes results in a much larger amount of parameters in the network. Normally, when updating each of these parameters, this would take more time. However in Einsum Networks, a smart application of the Einstein summation convention is used to perform a large amount of arithmetic operations in a single operation. This single operation is called the einsum operation, hence the name Einsum Networks. It is possible to use this einsum operation due to the smart representation of the vectorized nodes. The einsum operation is able to take advantage of the GPU very well, since a lot of parallel computations are executed. This results in speeding up the learning time and reducing the memory consumption both up to two orders of magnitude, which enables probabilistic circuits to much larger datasets.

The Einsum Networks itself are not a structure learning algorithm on its own, but only a smart vectorized representation of probabilistic circuits. This means that multiple structure learners, for example the earlier explained RAT-SPN structure, can be applied in combination with Einsum Networks.

Chapter 2

Literature Review

The literature review, presented in this work, will focus on presenting the current literature of the topic of Probabilistic Circuits. More specifically Sum-Product Networks are a major focus. In addition the earlier explained Einsum Networks will be shortly touched on. For each of the sections in this literature review, the papers are presented in the same order as they are published. Finally the relation between the current research and the research conducted in this work will be explained.

2.1 General Theory

In 2011 Poon and Domingos propose a new kind of deep architecture named sum-product networks (SPNs) [33]. SPNs aim at solving hard queries while still maintaining tractable inference. A SPN is a directed acyclic graph of both sum nodes and product nodes. These nodes combine different distributions such that a more complex distribution is formed by the complete network. Backpropagation can be applied to all the nodes of an SPN, which makes it possible to train the sum-product networks. Poon and Domingos show that, when an SPN adheres to the structural constraints of completeness and consistency, the goal of tractable inference is reached for some hard queries.

Delalleau and Bengio investigate the representational power of sum-product networks [7]. In their work, deep architectures are compared to shallow architectures. Delalleau et al. show that certain probabilistic functions can be modeled better by a deep network. More specifically Delalleau et al. show that a shallow network will need to grow exponentially in terms of its parameters, where the deep network only needs to grow linearly, to model the same functions. This work contributes to better understand the theoretical properties of sum-product networks.

Martens et al. did make a deep analysis of properties that provide tractable inference [21]. Namely, the decomposability and the completeness constraints are thoroughly examined. In this work some distributions are highlighted which cannot be modeled when also adhering to the decomposability and completeness constraints. This work provides a deeper understanding of the decomposable and complete sum-product networks and hence added to the theory of probabilistic models.

In addition to better understanding the decomposability and completeness properties, Peharz et al. made an analysis on the theoretical properties of sum-product networks [30]. One of the major findings from Peharz et al. is that a complete and consistent SPN can always be put into a locally normalized form without losing any modeling power of the SPN. This means that weights for a certain sum node can always be normalized. It is also established that consistent SPNs cannot model distributions exponentially more compact than decomposable SPNs. This paper provides a much deeper understand of sum-product networks.

Zhao et al. show the relationship between sum-product networks and Bayesian networks [48]. Zhao et al. prove that every SPN can be converted to a Bayesian network. With the methods proposed their work it is also possible to transform the generated Bayesian network back to a sum-product network. It is also shown that SPNs are not more powerful than Bayesian networks.

2.2 Structure Learning

Dennis and Ventura propose an algorithm to learn the structure of a sum-product network [8]. Earlier only a static structure was used. Dennis et al. show that learning the structure of an SPN from the data using their algorithm significantly improves the performance of the sum-product network. The idea of clustering is used to identify similar groups of data instances in the training data. These groups are then used to construct the structure of a SPN. This is the first algorithm proposed to learn the structure of sum-product networks from data.

After introducing the discriminative learning method of sum-product networks, Gens and Domingos propose an algorithm that learns the structure of the SPN from data [15]. The LearnSPN algorithm that is proposed by Gens and Domingos is already explained in section 1 of this work. The LearnSPN algorithm divides the data into independent subsets at each step. These subsets are splitted again in a recursive manner. The algorithm uses these splits to construct the SPN. This results in models that are superior in inference speed and accuracy.

In earlier proposed structure learning algorithms, every method uses a top-down approach. Peharz et al. propose a method that constructs an SPN from the bottom up [26]. The main idea behind the algorithm is that it merges small probablistic models into more complex ones. Peharz et al show that this algorithm is competing well with the current structure learning algorithms. The algorithm does not rely on properties of images, which makes the algorithm suitable for many applications.

Learning the structure of SPNs from data is a very usefull feature. But in some instances not all data is present when constructing the sum-product network. This gives rise to online structure learning algorithms. Lee et al. propose an online incremental structure learning method for sum-product networks [20]. The proposed method depends on an online clustering of the data. This means that, when new instances come in, the clustering of the data can change, which results in a change of the tree structure of the SPN. Lee et al. show that the algorithm reaches performances of other structure learning algorithms where all the training data is already present.

The structure learning approaches suggested earlier are either focusing on a top-down approach, which captures the similarities of data sample groups and mixes them, or on a bottom-up approach, which captures the interactions between data sample groups. Rooshenas et al. propose an algorithm, ID-SPN, which combines both approaches [37]. The results presented by Rooshenas et al. show that the algorithm performs very well, which suggests that the most effective probabilistic models should combine both the mixing of components as well as the interactions between components to construct an optimal structure for sum-product networks.

Adel et al. provide two structure learning algorithms, one that focuses on the generative cases and another one that focuses on discriminative cases [1]. These algorithms are based on recursively extracting rank-one submatrices from the input data. This means that the algorithm is not dependant on locally splitting the data but instead splits the data globally. These algorithms provide state of the art results on image datasets and digit classifications.

Most structure learning algorithms proposed earlier generally construct tree structures. Dennis et al. propose an algorithm called SearchSPN which learns the structure of an SPN from data based on a greedy search approach [9]. It uses some principles which are already proposed, however the main advantage of SearchSPN is the learning of the structure such that the sum-product networks are less-restrictive and non-tree like. By doing this the network size does not increase dramatically at any point in the algorithm as opposed to other structure learning algorithms.

The earlier proposed LearnSPN structure learning algorithm is further investigated and improved by Vergari et al [43]. Vergari et al. improve both structural quality and the performance of the networks. These improvements are made by better understanding the building process of the network by the LearnSPN algorithm. The first improvement on the algorithm includes limiting the number of node children in the LearnSPN algorithm, which resulted in much simpler and deeper networks. Secondly, bagging is used in the sum nodes to improve the robustness of the models. Vergari et al. proved their methods to be effective, since experiments showed improved results on multiple datasets. These ideas could also be used in other structure learning algorithms.

The earlier proposed LearnSPN algorithm is a structure learning algorithm which assumes that variables are discrete. It also assumes that there is no missing data. Krakovna et al. propose an algorithm MiniSPN which is a practical and simplified version of LearnSPN [19]. MiniSPN runs faster and is able to handle missing data and heterogeneous features which are both very common in real world applications. MiniSPN is an algorithm very suitable for messy real world datasets.

As mentioned before, many structure learning algorithms provide tree structures for sum-product networks. Rahman et al. propose post-processing approaches which induce graph SPNs from tree SPNs by merging similar sub-structures [34]. The benefits of graph SPNs include smaller computational complexity which provides faster online inference and better generalization accuracy. Rahman et al. show that graph SPNs can significantly boost the accuracy and prediction time of tree SPNs. This is the case due to the substantially reduced number of parameters in the sum-product network.

Most structure learning algorithm proposed to date are suitable for inputs of static length. Melibari et al. propose dynamic sum product networks (DSPNs) for sequence data of varying length [23]. A DSPN consists of a template network which is repeated a certain amount of times based on the input sequence. The structure of the template model is learned by making use of a local search technique. Melibari et al. compare the DSPN to a dynamic Bayesian network (DBN). The DSPN fits sequential data better than static SPNs produced by for example LearnSPN. The DSPNs also outperform the DBNs on several datasets.

The earlier proposed SearchSPN algorithm contains properties which are very suitable for making it an online structure learning algorithm. Dennis et al. propose OnlineSearchSPN which adapts SearchSPN such that it is able to use in an online fashion [11]. It is compared to a fast learning, but poor performing model and to a slow learning, but good performing model. Dennis et al. show how OnlineSeachSPN is able to show comparable performance to the slow learning model with a learning time comparable to the fast learning model. This means that OnlineSearchSPN shows very promising results.

Jaini et al. propose an structure learning algorithm called Prometheus [16]. Prometheus is a graph partitioning based algorithm which creates multiple variable decompositions for learning sum-product networks. Prometheus is able to function in low data regimes. The proposed algorithm is tested thoroughly and is able to perform state of the art results on multiple datasets.

Peharz et al. propose the earlier explained RAT-SPNs [32]. RAT-SPNs are random models constructed from the input data. RAT-SPNs are able to perform predictive tasks as well as generative modeling. The models are also highly robust when the input data contains missing features. The RAT-SPNs are able to be trained in a classical deep learning manner, this means by employing automatic differentiation, which results in easy use of GPUs.

Since most structure learners of sum-product networks are mostly based on intuition rather than a clear learning approach, Trapp et al. propose an Bayesian framework for SPN structure learning [41]. A very crucial part of the proposed structure learning algorithm is that the learning of the computational graph is separated from the learning of the scope function over this computational graph. The Bayesian SPNs proposed by Trapp et al. often improve test likelihoods over other structure learning algorithms. Due to the Bayesian properties of the framework, the networks are protected against overfitting. This makes them function very well in cases where little data is present, since there is no need for a validation set.

2.3 Parameter Learning

Since sum-product networks are generative models by nature, these have been learned generatively in the beginning. Gens and Domingos are the first to propose a discriminative learning algorithm for SPNs [13]. This is done using an efficient backpropagation-style algorithm. Here, one problem to overcome, is the diffusion of the gradients when using a gradient descent algorithm. This problem is solved using a so-called 'hard' gradent descent. This means that marginal inference is replaced by MPE inference. Gens and Domingos reach the best results to date on the CIFAR-10 dataset. Introducing a discriminative learning method opens up possibilities for more architectures of SPNs.

Peharz, Gens and Domingos introduce a new constraint on sum-product networks [27]. This is the selectivity constraint, which allows each sum node to only have one non-zero output for each possible input. This constraint makes it possible to use new structure learning and optimization approaches. This restricted class of SPNs is competing with the state of the art which makes them viable and therefore expands the research on probabilistic models.

Rashwan et al. propose an algorithm called online Bayesian moment matching (oBMM) [36]. Since the current SPNs do not scale easiliy to large datasets, an online algorithm for parameter learning is proposed by Rashwan et al. The oBMM algorithm can be distributed easily over many machines, which makes it much more applicable to larger datasets. Rashwan et al. also compare the online Bayesian moment matching to online versions of gradient descent. oBMM outperforms other algorithms due to the distribution over multiple machines.

The current learning approaches of SPNs are largely based on the maximum likelihood principle. This makes these approaches subject to overfitting. Zhao et al. propose an algorithm called CVB-SPN which stands for collapsed variational inference for SPNs [47]. CVB-SPN is a deterministic approximate Bayesian inference algorithm which is able to learn the parameters of an SPN without it being subject to overfitting. The proposed algorithm is able to compete with state of the art learning algorithms and is able to learn both in batch mode as in online settings.

Sum nodes in sum-product networks are earlier described as latent variables. However this approach is in conflict with the completeness condition in sum-product networks. Peharz et al. further explore the latent variable interpretation in sum-product networks [28]. Peharz et al. also propose an algorithm, SPN augmentation, which proposes a remedy for this conflict. From the SPN augmentation algorithm Peharz et al. find a sound derivation of the EM algorithm for sum-product networks. The theory provided by peharz et al. is proven to be correct by verifying it on many real-world datasets.

Learning the parameters of sum-product networks is mostly done by improving the internal weights of the SPNs. Desana et al. focus on improving complex leaf distributions [12]. Desana et al. propose an efficient method to learn leaf distributions with Expectation Maximization. This is done by making use of the maximum likelihood function. The models proposed by Desana et al. outperform state of the art parameter learning approaches, while containing a significantly smaller amount of parameters.

Annotated classification data is often expensive to obtain, where on the other hand unlabelled data is much easier to collect. Trapp et al. propose an algorithm which is able to perform semi-supervised learning for sum-product networks [40]. This method has multiple advantages, firstly, it is able to perform generative as well as discriminative semi-supervised learning. In addition it guarantees that a model can only be improved when adding unlabelled data to the training. The algorithm is also computationally efficient and does not enforce structural limitations on the data. Because of these reasons this approach allows for a wide applicability in real-world problems.

Rashwan et al. propose a discriminative learning algorithm for sum-product networks, which is based on the Extended Baum-Welch (EBW) algorithm [35]. A conditional data likelihood function is formulated which is maximized by the EBW algorithm. Rashwan et al. show better results than both the generative Expectation-Maximization algorithm and the discriminative gradient descent.

2.4 Extensions

Sum-product networks are often used to perform classification tasks. However Mauà et al. state that this classification is often prone to overconfidence. Mauà et al. propose credal sum-product networks, which are able to distinguish between reliable and unreliable classifications [22]. This adds a layer of robustness to the SPNs.

Dennis et al. propose an algorithm which combines autoencoders with sum-product networks [10]. The autoencoder-SPN (AESPN) combines two SPNs and an autoencoder into a single model. It shows improved results when sampling from the AESPN compared to just a single SPN. This is the case when sampling all the variables in the input, but also when sampling a subset of variables. This makes the AESPNs very suitable for image reconstruction.

In sum-product networks, a choice has to be made what kind of representation is desired for the random variables, i.e. Gaussian, Poisson or Logit functions. Molina et al. propose Mixed SPNs, which is a deep architecture for hybrid domains [24]. By doing this there is no need for the user to specify what kind of random variable representation should be used. The Mixed SPNs are able to capture complex distributions across hybrid domains.

Sharir et al. propose sum-product quotient networks (SPQNs) which are a extension of sum-product networks [39]. SPQNs are able to boost the expressive power of SPNs significantly. Sharir et al. show how SPQNs are able to model distributions which can only be modelled by exponentially large SPNs. The core concept of SPQNs is to incorporate conditional distributions in the quotient nodes. Sharir et al. also prove how the SPQNs are still tractable models.

As mentioned earlier sum-product networks may be prone to robustness issues, particularly when training them with scarce data. Conaty et al. further investigate this issue [6]. Conaty et al. use a robustness measure to determine the reliability of a classification. When the classification turns out to be unreliable, the task is given to another model. This method can be used to improve the classification accuracy. Here the credal sum-product networks, as described earlier, are used to determine the reliability.

Ko et al. mention the connection between sum-product networks and tensor networks [18].

This connection gives rise to tensor SPNs (tSPNs) which is a highly efficient model in terms of its parameters compared to a normal SPN. The parameter compression does not influence the accuracy of the networks by a high amount.

Sum-product networks struggle with capturing complex spatial relationships in image data. Wolfshaar et al. propose Deep Generalized Convolutional sum-product networks (DGC-SPNs) which are able to encode spatial features similarly to how CNNs do this [42]. This is all done while perserving the validity of the SPN. DGC-SPNs are outperforming the standard SPN structure on both generative and discriminative tasks, which include image inpainting and classification. The DGC-SPNs are implemented in the well known Keras and TensorFlow frameworks, which makes them efficient and straightforward to implement.

The current sum-product networks are still difficult to train on real-world data sets. Peharz et al. propose Einsum Networks (EiNets) which are a novel implementation of sum-product networks [29]. EiNets combine a large number of arithmetic operations in a single einsum-operation. This means that much less memory is required and the networks are able to learn much faster.

2.5 Applications

Cheng et al. are the first ones to explore the field of speach in combination with sum-product networks [5]. Cheng et al. use SPNs to model complex dependencies among words. Due to the tractable inference and learning times, sum-product networks form a suitable framework for language modeling. Cheng et al. show that the SPNs are able to capture rich dependencies among words. The SPNs are outperforming other language models.

Amer et al. show how sum-product networks can be used for activity recognition in videos [2]. The focus of this work is on activities with multiple, alternative structures. The main example that is used is classification of different activities in a volleyball game. Since here distinctions should be made on very specific actions, which is the case in many other applications. The methods used by Amer et al. outperform state of the art algorithms on activity detection.

In addition to the work of Amer et al., Wang et al. propose a spatial sum-product network, which focuses on action recognition [44]. Here the actions are multiple spatial configurations of body parts. This research is done for still images. The spatial SPN is able to model relationships between parts of sub-images. The method proposed by Wang et al. shows to be effective.

Nath et al. propose Tractable Fault Localization Models (TFLMs) which use the newly proposed sum-product networks [25]. This application of SPNs is able to infer locations of bugs in software programs. The models are trained on a corpus of previously seen buggy programs. In addition TFLMs are also able to learn from other sources, for example the output of other fault-localization systems as features of the probabilistic model. TFLMs take advantage of the tractable sum-product networks to make sure that the fault location probabilities are inferred efficiently.

Sguerra et al show an application of SPNs in Micro Aerial Vehicles (MAVs) [38]. Since MAVs are not able to carry heavy sensors such as Lidar or RGD-B, a strategy where image classification is used to guide the MAV is explored. Due to the fast an tractable inference of SPNs, an image classification task is performed while in flight. This image classification can then be used to navigate the MAV. Even though the learning of the SPNs can take longer than other methods, the speed of answering the query out weights this completely.

2.6 Context to this Research

In the current research different learning approaches, including generative and discriminative approaches, have been proposed. However, how both approaches perform on generative and discriminative tasks is not yet examined. This will be done in this work. Firstly the influence of the parameters of the RAT-SPN structure will be tested, since the RAT-SPN structure will be used in comparing the different learning approaches.

The initial parameters of probabilistic circuits are mostly taken from random distributions. The current research does not propose methods to improve the initial state of the networks, before learning them. Therefore several initialization methods will be proposed and tested in this work. In addition, online improvement of probabilistic circuits is also a new topic, since it has not been touched upon in the current research for probabilistic circuits. The Online Adaptation section will focus on this topic, with the goal to extend the current research.

Chapter 3

Network Analysis

In this chapter multiple analysis will be executed. First the RAT-SPN structure, implemented in the Einsum Network library will be investigated. Then multiple different setups for Einsum Networks will be tested.

3.1 Motivation

When using a machine learning model, most of the time the task for the model is already known. This task can for example be a classification job. When this is the case, the most optimal learning approach to reach the best possible performance for this classification job should be used. Sometimes it can also be possible to construct a network for multiple tasks. This means that the network should perform well on multiple tasks.

Tasks can be split up in generative tasks and into discriminative tasks. Also the learning approaches can be split into generative learning approaches and discriminative learning approaches. In this chapter the main focus will be on discovering an optimal setup for performing both a generative and a discriminative task. In the process of finding this out, some research questions that should be answered are the following: Is it possible to improve the performance of a generative task by a discriminative learning approach? Is it possible to improve the performance of a discriminative task by a generative learning approach? What is the most optimal setup and learning approach when performing both a generative and a discriminative task?

Firstly the RAT-SPN structure will be examined, since the tests for finding the most optimal setup and learning approach will be conducted in the Einsum Network library by using the RAT-SPN structure. Examining the RAT-SPN structure will make sure that the conclusions drawn from comparing the different learning approaches will not be influenced by the structure. This makes the research conducted in this chapter much more robust. Here the research questions that arise are the following: How is the performance of the generative task influenced by the parameters of the RAT-SPN structure? How is the performance of the discriminative task influenced by the parameters of the RAT-SPN structure?

3.2 RAT-SPN Structure

The RAT-SPN structure, as explained earlier in chapter 1, is implemented in the Einsum Network library. The depth and the number of repetitions are parameters that have to be provided in order to use the RAT-SPN structure in the Einsum Network library. In this section, the main interest is to find out how both parameters influence different tasks. Tasks that a network should be able to execute can be either generative, for example modeling a distribution, or discriminative, for example performing a classification task. Sometimes it is even desired that a network is able to perform both a generative and a discriminative task. Therefore in this section both tasks will be executed and the influence of the parameters of the RAT-SPN structure on these tasks will be examined.

3.2.1 Setup

The setup that has been chosen to examine the different parameters is a setup that is able to perform both a generative task and a discriminative one. A so-called class discriminative (CD) Einsum Network is used. This means that for every class in the input data a separate sub-Einsum Network is constructed. All these sub-networks are then mixed together by a sum node. These sub-Einsum Networks for every class adhere to the RAT-SPN structure. Each of the sub-Einsum Networks are trained using Stochastic Gradient Descent with the objective to minimize the negative log-likelihood. As for the initialization of the network, random values are used. These values are sampled from a uniform distribution between zero and one.

For these experiments very large, including very deep and wide networks are trained. This means that a high amount of GPU memory is required. Because of this reason only the MNIST and Fashion MNIST datasets are tested. The SVHN dataset, which will be used in later experiments, did simply not fit into the GPU memory.

Some general settings which are required for Einsum Networks include the following. All the leaf distributions are Binomials. Since every node in a Einsum Network is vectorized, a so-called K parameter has to be specified. Here the vector length is set to 10. The batch size is set to 100 and every network is trained for 10 epochs.

3.2.2 Experiments

The goal of identifying the influence of both the depth of the network and the width of the network can be reached by testing both the generative task and the discriminative task when only adapting either the depth or the width.

Variable Width
To test the influence of the width of a network, a setup will be constructed such that the depth will stay the same and only the number of repetitions will be increased. For each sub-Einsum Network in the class discriminative structure the depth will be set to 3. The number of repetitions will range from 2 to 64. This means that the width of the network will greatly increase, this also increases the number of parameters of the network.

The results from this experiment are shown in figure 3.1. As can be observed both the log-likelihood and the classification accuracy do not improve when increasing the network size by adding more width to the model. This would suggest that increasing the width of the network does not have any influence of the performance.

Variable Depth
Since the number of repetitions did not influence the performance of both the generative and the discriminative task, the depth of the network will be tested now. Generally the depth of a network makes it possible to learn more complicated distributions, it is expected that the depth will increase the log-likelihood. For each sub-Einsum Network in the class discriminative structure the depth will be tested in a range from 1 to 5. The number of repetitions will stay the same for every network size, this will be set to 20.

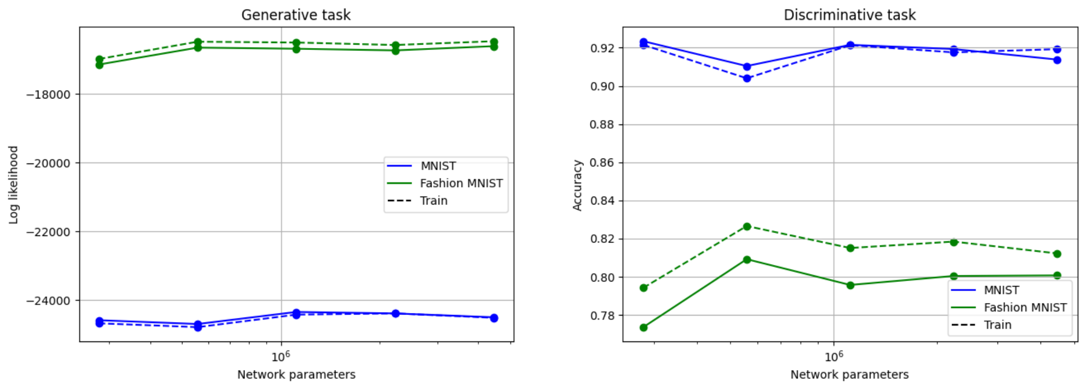

Figure 3.1: Results of a RAT-SPN structure with variable width

The results from this experiment are shown in figure 3.2. As can be observed the log-likelihood grows with the increase in depth. This is exactly as expected, as the depth of the network increases the capability of the network to model more complicated distributions. However when inspecting the results of the classification task, for larger networks, and thus larger depths, the classification accuracy makes a huge drop which is of course not desired.

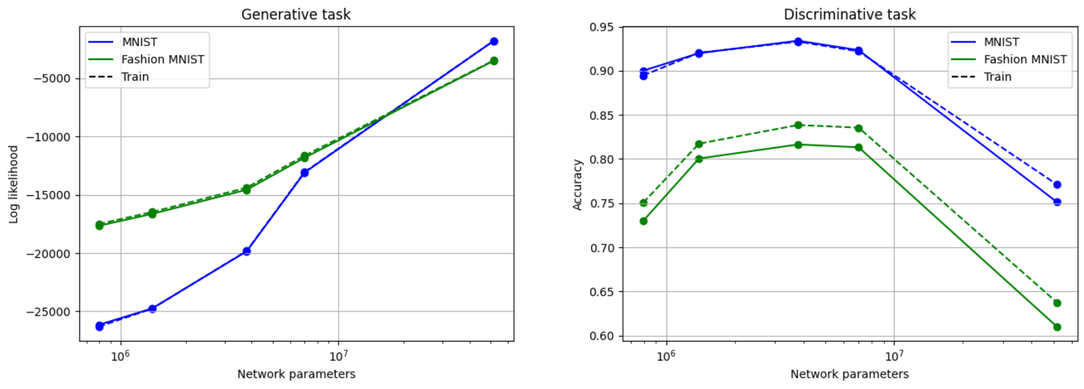

Figure 3.2: Results of a RAT-SPN structure with variable depth

The Optimal Combination between Depth and Width

Only increasing the depth of the network did greatly increase the ability to improve the log-likelihood. However a for larger network sizes a decline in classification performance is observed. This was not the case when only increasing the number of repetitions. Because of this a combination of increasing both the depth and the number of repetitions will be tested. The depth of the networks will range from 1 to 4 and the number of repetitions will range from 2 to 32.

The results from this experiment are shown in figure 3.3. As can be observed the drop in classification performance is not so clear anymore for the MNIST dataset. For the Fashion MNIST dataset there exists still quite a drop in performance. This means that increasing the number of repetitions made the classification accuracy more stable. It also means that for the larger network sizes even more repetitions are needed to keep the results completely stable. Due to GPU memory limitations, it was not possible to fully verify this hypothesis. And hence this could be tested more thoroughly in some future work. However since the drop in performance for the MNIST dataset is almost gone when increasing the number of repetitions, it is almost certain that adding more width to the network will stabilize the classification accuracy.

3.2.3 Discussion

To summarize this section, an optimal setup when using the RAT-SPN structure is a combination of both a deep and a wide network. The research questions that were stated earlier can be answered by the following two statements. The depth of the network provides the ability for the network to improve the generative task. The width of the network provides the ability for the network keep the classification performance stable.

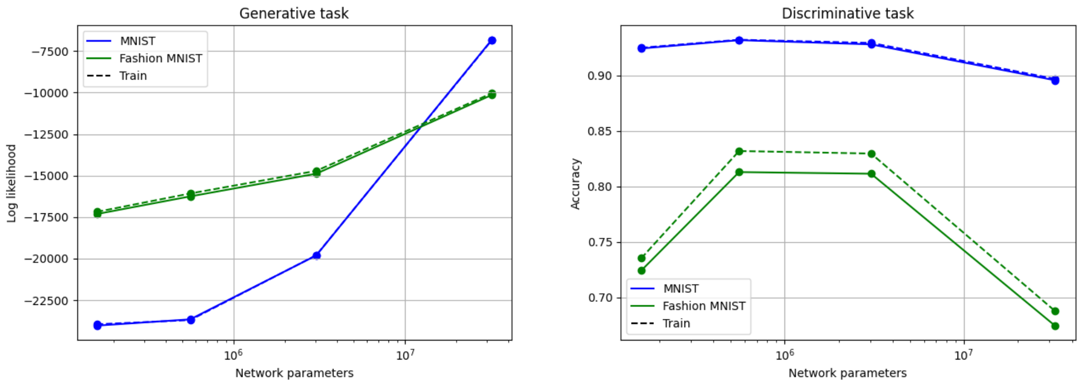

Figure 3.3: Results of a RAT-SPN structure with both increasing depth and width

3.3 Training Approaches

In this section, different setups for Einsum Networks are compared. The experiments conducted will be focusing on comparing different learning approaches, different network structures, different network sizes and different tasks. From all these options eight different experiments have been set up. The main goal is to compare generative and discriminative learning methods and generative and discriminative tasks to each other. Some research questions that these experiments should answer are: How do the different Einsum Networks perform when conducting a generative or discriminative task? Is it possible to improve generative tasks with discriminative learning approaches? Is it possible to improve discriminative tasks with generative learning approaches?

3.3.1 Setup

Initialization
Initialization is the main interest in this work. However the focus of this section is on comparing generative and discriminative learning approaches. Therefore a simple initialization will be used. This means that the parameters of the Einsum Networks are a initialized randomly. The random samples are taken from a uniform distribution between zero and one.

Learning Approaches
Training the Einsum networks will be done by a generative and a discriminative learning approach. For both the generative approach and the discriminative approach, the same optimizer is used. Namely Stochastic Gradient Descent (SGD). This makes the experiments comparable since only the objective function is different in the generative and the discriminative learning approach. SGD also requires a learning rate which is set to 0.2. The objective function that is optimized for

the generative learning approach is the log-likelihood. Since SGD minimizes the loss, a negative log-likelihood (nLL) loss is optimized. For the discriminative task the objective function that has been chosen is Cross Entropy (CE) loss. This choice has been made since CE is a very well performing objective function for a classification task. In both approaches the learning rate will not be experimented with in these experiments. Therefore the learning rate is set to a generally well performing rate.

Network Structures and Sizes
An Einsum Network, just like a neural network, can be constructed in different ways. Different structures could be performing better on different tasks. This is the reason why the choice has been made to suggest two different network structures. The first structure is a RAT-SPN as introduced in Peharz et al. [32]. The RAT-SPN structure has been chosen due to its easy scalability of network size. This makes it possible to compare different sizes of networks in a structured manner. This structure will be referred to as a standard Einsum Network. The second structure is the so-called class discriminative (CD) Einsum Network, as already explained earlier. This means that for every class in the input data a separate sub-Einsum Network is constructed. All these sub-networks are then mixed together by a sum node. These sub-Einsum Networks for every class also adhere to the RAT-SPN structure.

When the same size is used for the standard Einsum Network as for every sub-Einsum Network in the class discriminative structure, the logical remark can be made that the different structures are very different in size. To make sure that the experiments are balanced, it is desired that networks of the same size are compared. Hence this is why the sub-networks are scaled down in size, based on the amount of classes in the input data. To give an example, when there are four classes in the input data, there exist four sub-Einsum Networks in the class discriminative structure, which are all four times as small as the standard Einsum Network. This means that, when the sub-networks are mixed together the sizes of both structures compare to each other. By constructing a class discriminative network the width of the network will be much larger than of the standard Einsum network. Therefore it is most logical to make both the class discriminative structure and the standard Einsum Network the same size by tweaking number of repetitions in the RAT-SPN structure for every sub-Einsum Network in the class discriminative structure, since the number of repetitions influences the width of the network. Hence when comparing the two structures not only the number of parameters is comparable but also the width and the depth of the networks are comparable.

In addition to just a single network size, it is still interesting to see how different sizes of networks perform. Because of this, every experiment will be tested with five different network sizes, ranging from very small to relatively large. This is realized by tweaking the depth and the number of repetitions of the RAT-SPN structure. The depth ranges from 1, in smallest case to 5, in the largest case for both the standard Einsum Network and the class discriminative Einsum Network. The number of repetitions ranges from 2 to 10 for each of the sub-Einsum networks in the class discriminative structure. In the standard Einsum Network the number of repetitions ranges from 20 to 100.

Tasks
The networks will be performing both a generative and a discriminative task. For the generative task, the log-likelihood will be measured. This is a measure which expresses how well the network models the input distribution. A classification task is performed as the discriminative task. The accuracy of the classification task is measured. It is interesting to see how both tasks perform when changing other factors in the network.

Datasets
In order to get more robust results, all the experiments will be tested on three different datasets. These datasets are MNIST, Fashion MNIST and the Street View House Numbers (SVHN) data-

set. Both the MNIST and Fashion MNIST are black and white pictures of size 28 by 28 pixels. The SVHN dataset contains colored images of 32 by 32 pixels. This means that the datasets slightly differ from each other in terms of size and dimensions. Testing the experiments on different datasets creates the opportunity to make a wider and more robust analysis of the experiments.

General Settings
Finally the experiments require some more parameters which will not be experimented with here. Hence in all experiments these will be set to the same value. Namely the leaf distributions of all the networks are chosen to be Binomial Distributions. The K, specifying the vector length in Einsum Networks, is set to 10 in all experiments. The batch size for training and evaluation is set to 100. This speeds up the training and evaluation by a lot, additionally it also stabilizes the learning of the networks. Every network is trained for 10 epochs, which seems to be enough for convergence.

3.3.2 Experiments

From the above described learning approaches and structures, four different setups have been constructed. Every setup will be tested on five different network sizes and on all the datasets. Every setup will be tested on both the generative task and the discriminative task, which results in a total of eight experiments. In order to get a clear view what is tested, table 3.1 shows an overview of the different experiments. Every experiment is explained shortly down below.

	nLL on EiNet	**nLL on CD EiNet**	**CE on EiNet**	**CE on CD EiNet**
log-likelihood	Experiment 1	Experiment 3	Experiment 5	Experiment 7
Classification	Experiment 2	Experiment 4	Experiment 6	Experiment 8

Table 3.1: Overview of the experiments conducted in this section

Experiment 1
A standard Einsum Network is trained by optimizing the log-likelihood by Stochastic Gradient Descent. The log-likelihood is measured.

This is the first experiment the goal is to optimize the log-likelihood and hence model the input data as well as possible. Here a generative learning method is combined with testing a generative task. This network should be good at generative tasks like completing a sample image or generating new images.

Experiment 2
A standard Einsum Network is trained by optimizing the log-likelihood by Stochastic Gradient Descent. A classification task is executed.

In this experiment the Einsum Network is also trained by the generative approach, namely by optimizing the negative log-likelihood loss. In the learning phase this network is optimizing the log-likelihood. However after learning the network, a classification task will be conducted, which is a discriminative task. Therefore it will be interesting to see the results of this experiment since this gives insight into whether or not it is possible to train a network generatively when conducting a discriminative task.

Experiment 3
A class discriminative Einsum Network is trained by optimizing the log-likelihood of each sub network independently. The log-likelihood of the full network is measured.

In experiment 3 a class discriminative Einsum Network is constructed. The generative learning approach is executed by optimizing the log-likelihood of each sub-network for a particular class by means of Stochastic Gradient Descent. Every sub-network is trained by the samples in the training data that correspond to the same class of the sub-network. Then the log-likelihood of the complete network is measured.

Experiment 4
A class discriminative Einsum Network is trained by optimizing the log-likelihood of each sub network independently. A classification task is executed.

In this experiment the same structure is used as in experiment 3. Also the learning approach is the same. However instead of measuring the log-likelihood, a classification task will be conducted. When comparing this experiment to experiment 2, it is very interesting to see the change in performance due to the structure of this network.

Experiment 5
A standard Einsum Network is trained by executing a SGD algorithm with a Cross Entropy objective function. The log-likelihood is measured.

This is the first experiment in which the discriminative learning approach will be tested. This is realized by executing a standard Stochastic Gradient Descent algorithm with the objective to minimize the Cross Entropy loss. The log-likelihood is measured. It is interesting to see how the discriminative learning approach affects the generative task which is performed in this experiment.

Experiment 6
A standard Einsum Network is trained by executing a SGD algorithm with a Cross Entropy objective function. A classification task is executed.

In experiment 6, the discriminative learning approach from the last experiment will be combined with a discriminative classification task.

Experiment 7
A class discriminative Einsum Network is trained by executing a SGD algorithm with a Cross Entropy objective function. The log-likelihood is measured.

Experiment 7 will be measuring the log-likelihood of a class discriminative Einsum Network. The structure is the same as earlier described in experiment 3. In this experiment this structure is combined with a discriminative learning approach, namely a standard Stochastic Gradient Descent algorithm with the objective to minimize Cross Entropy loss. As opposed to experiment 3 the full class discriminative network is trained using all the training samples. In this experiment, the class discriminative structure is combined with a discriminative learning approach. A generative task is executed, namely measuring the log-likelihood. It is interesting to see how the generative task is affected by the discriminative learning approach in this experiment.

Experiment 8
A class discriminative Einsum Network is trained by executing a SGD algorithm with a Cross Entropy objective function. A classification task is executed.

Finally in experiment 8, a class discriminative Einsum Network is trained by applying SGD with the objective to minimize the Cross Entropy loss. Here a classification task is executed. This experiment will be testing the synergy between the discriminative learning approach and the discriminative task.

3.3.3 Impressions

Generative task
The results of the experiments will show the differences in the effectiveness of the learning approaches in combination with the structure. When analyzing the results of the generative task, namely measuring the log-likelihood, it is expected that the generative learning approach outperforms the discriminative learning approach, when the structure of the networks is the same.

Therefore it is expected that experiment 1 outperforms experiment 5 and experiment 3 outperforms experiment 7.

Then the comparison between the structure from the standard Einsum Network and a class discriminative Einsum Network has to be made. Here it is expected that for the generative task both structures are able to perform very equally. Since both structures have the same amount of parameters and since both structures are equally deep and wide models, the networks should be able to model the input distribution very similarly. Therefore it is expected that experiment 3 and experiment 1 perform very similar and also experiment 7 and experiment 5 should perform very similarly.

Putting it all together, experiment 1 and experiment 3 should perform very similar. Also experiment 5 and 7 should perform very similar. Experiment 1 and experiment 3 are expected to outperform experiment 5 and experiment 7.

Discriminative Task
For the discriminative task the results are expected to differ. Namely the class discriminative structure should make it much easier for the Einsum Networks to classify a sample. Also, the discriminative learning approach is expected to outperform the generative learning approach, when the network structure is the same. Hence experiment 6 should outperform experiment 2 and experiment 8 should outperform experiment 4.

Since the class discriminative structure should make the classification task so much easier, it is expected that experiment 4 outperforms experiment 2 and experiment 8 should perform better than experiment 6. In addition it is anticipated that the change in structure has a larger impact than the change in learning method for the discriminative task. This means that experiment 4 is expected to perform better than experiment 6.

This will result in the following order. Experiment 8 should be the best performing on the discriminative task, followed by experiment 4. Then experiment 6 comes in at the third best performing experiment. Which leaves experiment 2 to be worst performing for the discriminative task.

3.3.4 Observations

Generative Task
The results of the generative task are very robust, since all datasets show very similar trends in their results for every experiment. In addition to that, all experiments show a positive trend between the log-likelihood and the network size. This means that the differences in performance that occur are a result of the change in the structure of the network and the different learning approaches.

The first hypothesis that was made stated that the generative learning method should outperform the discriminative learning method. This hypothesis is completely accepted when analyzing the results. Experiment 1 and 3 largely outperform experiment 5 and 7, due to the difference in learning approach.

The second hypothesis stated that the class discriminative structure should not have a large impact on the generative task. When inspecting experiments 1 and 3 in figure 3.4, it can be observed that the class discriminative structure does slightly outperform the standard structure. However in experiments 5 and 7 the class discriminative structure does not show a difference. This is very much in line with the expectations, namely that the class discriminative structure does not have a large impact on the generative task.

To make sure that the generative task is performed best, it is desired to have a generative learning method. The structure of the network does have less of an influence. This means that the setup of either experiment 1 or experiment 3 is optimal for the generative task.

Discriminative Task
The classification task was greatly impacted by the structure. The standard Einsum Network was not able to learn how to perform the classification task when the network was trained by the generative learning approach. The discriminative learning approach did improve the results for the discriminative task already. However when the class discriminative structure is applied, the networks were able to perform the classification task even better.

When comparing the different learning approaches, it was expected that the discriminative learning approach would outperform the generative learning approach. This hypothesis is correct for the standard Einsum network. However when using the class discriminative structure the results of both training methods perform quite similar. For larger network sizes it can be observed that experiment 8 performs slightly better than experiment 4, this shows the synergy between the discriminative learning method and the discriminative task. However the results of experiment 4 are much more stable and generally better performing. The second hypothesis, that states that the class discriminative structure outperforms the standard structure is completely accepted since experiment 4 outperforms experiment 2 and experiment 8 outperforms experiment 6. With that the last hypothesis, stating that the network structure has more impact on the results than the learning approach, is also accepted.

This means that for the discriminative task, a class discriminative structure is desired. The learning approach is of less importance for the discriminative task.

3.3.5 Discussion

The experiments described in this section make for a very good analysis of the different learning approaches for different tasks. General trends that are discovered are: The classification accuracy is improved a lot by the class discriminative structure. The class discriminative structure does not harm the ability to improve the log-likelihood .

When constructing a model which should perform both a generative task as well as a discriminative task, the most optimal setup would be a class discriminative structure, combined with generative learning for each of the sub-Einsum Networks in the class discriminative structure. This is the setup that was used in experiments 3 and 4. This setup will therefore be used during further experiments in this work.

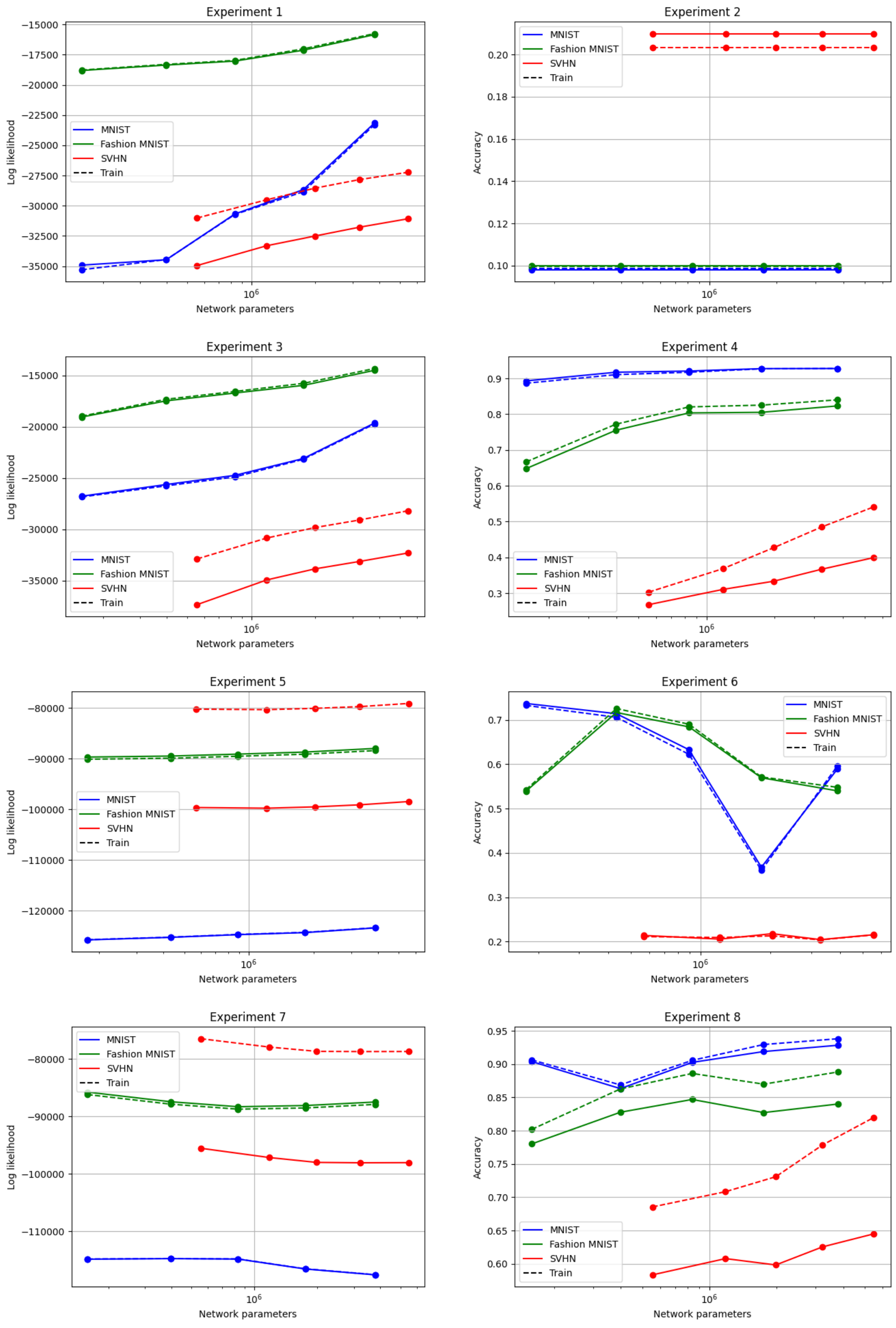

Figure 3.4: Results of experiments 1 to 8

Chapter 4

Initialization

This chapter explains three different initialization algorithms, namely Leaf initialization, Cluster initialization and Einsum Cluster initialization. The main idea of the algorithms is to provide the Einsum Network with a suitable initialization for further training and an improved starting point.

4.1 Motivation

Now that it is known how the RAT-SPN structure influences the ability to improve both the generative and the discriminative task and which possible setup to use for performing both a generative and a discriminative task, it is time to move on to the next topic.

Initialization seems like a very small part of training a neural network. However the initial values of a network can make the difference between convergence and no convergence at all. In neural networks, initialization is a very well researched topic, however, in probabilistic circuits there has not yet been done any direct research to the topic of initialization at all. In the current state of the art, all the probabilistic circuits are initialized with random weights. These random weights seem to converge, but it is not yet known how much performance improvement can be reached with a new and improved initialization method. Also the initialization of a network can possibly speed up the training process. This chapter aims at proposing several initialization algorithms to improve the performance of Einsum Networks. These initialization algorithms will be tested thoroughly.

4.2 Algorithms

Leaf Initialization
The first initialization method proposed is a more basic one. The Leaf initialization namely consists of only initializing the leaf distributions of the Einsum Network. This means that the internal weights are initialized by random noise as is standard in Einsum Networks.

The leaf nodes consist of Binomial distribution vectors which each have to be initialized with a success probability. To improve the initialization of these leaf distributions, the training data is used to determine how these distributions should be initialized. This is realized in the following way. Each pixel in the input data is defined by a value between 0 and 255. For each pixel the average value of all the input samples is computed, then this average is normalized to get a value between 0 and 1.

By using this method, a probability, for each pixel in the input images, can be calculated. This probability is then used to initialize the success probability of the Binomial distribution vector which corresponds that particular pixel.

Cluster Initialization
The second initialization method proposed is the Cluster initialization algorithm. An Einsum Network is built by combining multiple layers consisting of nodes into a full network. To improve the starting point for such a network, a clustering algorithm will be tested. The main idea behind this clustering algorithm is to use the samples of the training data to determine weights for an Einsum Network. This is realized by executing a hierarchical clustering method on the training data.

The algorithm can be explained best when applying it to a normal SPN. A SPN consists of sum nodes, product nodes and leaf nodes. The start of the algorithm will put all data samples in the top node of the network, then the network will be traversed from top to bottom.

A sum node will take the samples that it is given. This consists of the union of all the samples that are passed to this sum node from its parents. When the sum node is a root node it is simply given all the samples. The sum node then performs a KMeans clustering algorithm which clusters the data samples in as many clusters as the sum node has children. The weights for the sum node are determined by the number of data samples in a cluster. Then the data samples in a cluster are passed to the correct child, namely corresponding to the cluster the samples are assigned to.

A product node will take the union of all the samples that are passed to this product node from its parents. Then the product node will distribute all the samples to all of its children. Here no clustering is executed, since there are no weights that have to be initialized in a product node.

A leaf node does also not perform clustering, however it should be initialized. This is done by taking the union from all the samples that are passed to this leaf node from its parents. Then it will take the average value of these samples to initialize the leaf node in the same way as explained in the Leaf initialization algorithm.

Summarizing, the training samples are assigned to the root node of a SPN, when reaching a sum node the samples are divided over its children, when reaching a product node all the samples are distributed to all its children and a leaf node uses the assigned samples. In this way the samples are divided over the network in a structured way and the initial values are based on this division of samples. This should improve the networks ability to learn.

Since this is how the algorithm would work in a SPN the step to a Einsum Network has to be made. An Einsum Network is a vectorized version of a SPN. This means that a node in a Einsum Network has a full vector of values instead of one. However the algorithm can still be implemented in the same way. The initial values determined by the algorithm as described above can simply be duplicated for every value in the vector. This is a rather simple approach to use the algorithm in Einsum Networks. A more Einsum Network tailored approached is described as the Einsum Cluster initialization algorithm in the next section. Algorithm 4 shows the pseudo code for the Cluster initialization.

Algorithm 4 Cluster initialization

```
1: Assign all training samples to root node
2:
3: for node in EinsumNetwork.nodes do
4:     if node == sum_node then
5:         KMeans(#node.children, node.samples)
6:         node.weights = #samples in each cluster
7:         for child in node.children do
8:             for sample in node.samples do
9:                 if sample is in same cluster as child then
10:                    child.samples = child.samples + sample
11:                end if
12:            end for
13:        end for
14:    end if
15:
16:    if node == product_node then
17:        for sample in node.samples do
18:            for child in node.children do
19:                child.samples = child.samples + sample
20:            end for
21:        end for
22:    end if
23:
24:    if node == leaf_node then
25:        node.weights = average(node.samples)
26:    end if
27: end for
```

Einsum Cluster Initialization
The Einsum Cluster Algorithm is a more advanced version of the Cluster initialization algorithm described above. It makes use of the vectorized nodes in an Einsum Network. In order to determine the weights, the Einsum Network is traversed top down, in layer-wise fashion. An Einsum Network consists of Einsum Mixing layers, Einsum layers and a Leaf layer. Each layer functions differently and therefore has its own implementation of the Einsum Cluster initialization algorithm.

The Einsum Mixing Layer consists of M sum nodes, each having a maximum number of children D. Every sum node is vectorized by a vector of length K. This means that the weight tensor of a Einsum Mixing Layer has a dimension of $D \times M \times K$. For every sum node in the Einsum Mixing Layer, the algorithm executes a KMeans clustering of D clusters, for every K. This means that in total $M \times K$ clusterings are executed to determine the complete weight tensor of size $D \times M \times K$.

These clustering algorithms are executed based on the samples that are saved in the layer. Namely every sum node in the Einsum Mixing Layer has, for every $k \in K$, a list of sample indices of the training data. These samples are used in the clustering algorithm for that particular node and k and are clustered into D clusters. From such a clustering the weights are determined by counting the number of samples in each cluster. Then the samples are propagated to the successor of the sum node in the next layer.

The Einsum Layer consists of L sum nodes, each having one product node as a child. These product nodes again have exactly two children. All these nodes are vectorized by vectors of length K. Here every combination of children and parent have its own weight. Which means that an Einsum Layer has a weight tensor of the dimension $L \times K \times K \times K$. For every sum node in the

Einsum Layer, the algorithm executes a clustering K times, each clustering consists of $K \times K$ clusters. This means that in total $L \times K$ clusterings are executed to determine the complete weight tensor of size $L \times K \times K \times K$.

Again these clustering algorithms are executed based on the samples that are saved in the layer. Namely, for every sum node in the Einsum Layer has, for every $k \in K$, a list of sample indices of the training data is saved. These training samples are clustered into $K \times K$ clusters. The weights are again determined by counting the number of samples in a cluster. Again for every sum node the correct samples are propagated to the successors of the sum node.

The Factorized Leaf Layer does not perform clustering. The weight matrix of the Factorized Leaf layer has the dimensionality $D \times K \times R$. Here D is the number of Random Variables, K is again the length of the vectors and R is the number of the so-called replica as described in Peharz et al.[29]. The replica index provides a connection between the nodes in the Factorized Leaf Layer and the weight matrix. Hence from the samples that are assigned to a particular node, an average value can be calculated for the correct spot in the weight matrix. This results in the initialization of the complete Einsum Network. Algorithm 5 shows the pseudo code for the Einsum Cluster initialization

Algorithm 5 Einsum Cluster initialization

```
1: Assign all training samples to every spot in the root node vector
2:
3: for layer in EinsumNetwork.layers do
4:     if layer == EinsumMixingLayer or layer == EinsumLayer then
5:         for node in layer.nodes do
6:             for v in node.vector do
7:                 KMeans(#v.children, v.samples)
8:                 v.weights = #samples in each cluster
9:                 for child in v.children do
10:                    for sample in v.samples do
11:                        if sample is in same cluster as child then
12:                            child.samples = child.samples + sample
13:                        end if
14:                    end for
15:                end for
16:            end for
17:        end for
18:    end if
19:
20:    if layer == FactorizedLeafLayer then
21:        for node in layer.nodes do
22:            for v in node.vector do
23:                v.weights = average(v.samples)
24:            end for
25:        end for
26:    end if
27: end for
```

4.3 Experiments

The goal of an initialization algorithm is to improve the starting point of a network. In this section the Leaf initialization, Cluster initialization and Einsum Cluster initialization algorithms will be tested in various ways. The setup that has been chosen is the same as in experiment 3 and experiment 4 in described in section 3.3. This means that a class discriminative structure is trained by a standard SGD algorithm which optimizes the log-likelihood of each sub-Einsum Network in the class discriminative structure. The initialization algorithms are tested on all three datasets, namely the MNIST, Fashion MNIST and SVHN dataset. The general settings that are used are also the same, namely the K, specifying the vector length, is set to 10 and the batch size is set to 100.

In addition to this the KMeans algorithm from the SKLearn library also requires two parameters. These are the maximum iterations that the KMeans algorithm can run and the number of different random starts of the KMeans algorithm. These settings are set to 100 and 3 respectively.

Absolute Improvement
The first experiment that will be executed is testing the absolute improvement of the state of the network after only the initialization. All three algorithms, namely the Leaf initialization, Cluster initialization and Einsum Cluster initialization will be compared to a completely random initialization. This will be done by testing the log-likelihood and classification accuracy directly after initialization. Here it is expected that all three algorithms will outperform the random initialization. Since the Einsum Cluster initialization will execute the most amount of clusterings and has therefore the most amount of variation in weights, it is expected that the Einsum Cluster initialization will perform best.

Figure 4.1 shows the results of these experiments. Every initialization algorithm is shown by a different color. As can be seen, the results of the Einsum Cluster initialization algorithm and Leaf initialization algorithm are very comparable. They both outperform a random initialization in almost all cases. The Cluster initialization algorithm does show the best performance in both experiments for all the datasets. This is a surprising result since this algorithm is more general version of the Einsum Cluster initialization.

Performance During Training
To further investigate the impact of the initialization algorithms, a test will be executed where the performance is measured after every epoch. The goal of this experiment is to find out whether or not the initializations impact the results when training the network. It is expected that the Leaf initialization, Cluster initialization and Einsum Cluster initialization will perform better in earlier epochs.

Figure 4.2 shows the results of the different initialization algorithms after every epoch. When testing the log-likelihood it can be observed that only directly after the initialization there is a significant difference between the algorithms. This means that the initialization does not influence the ability of the network to improve the generative task. When inspecting the results of the classification task there are some differences between the initialization algorithms. This is mainly present in the SVHN dataset, which is actually the most complicated one. The Cluster initialization algorithm performs best in this case.

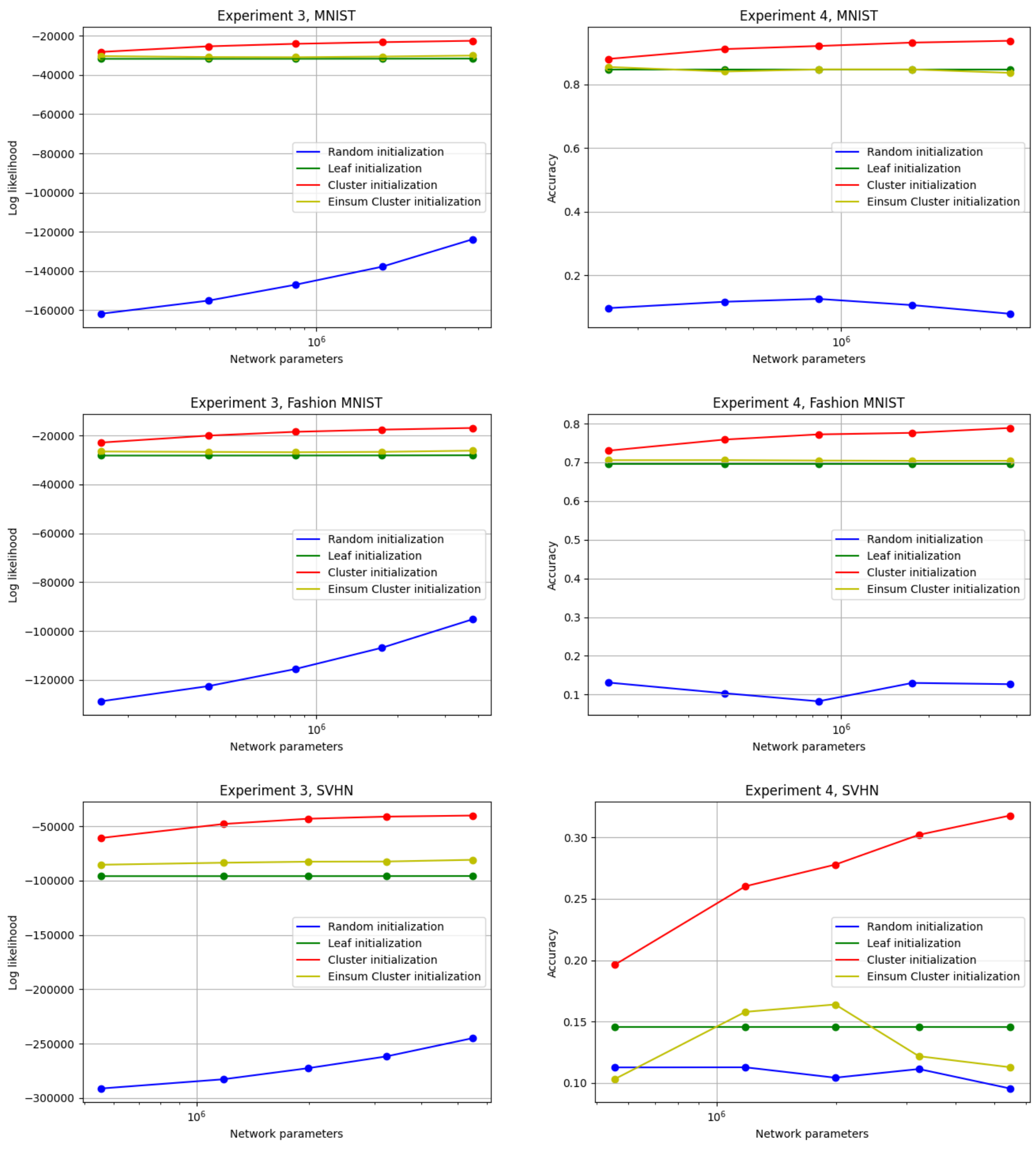

Figure 4.1: Results of testing different initialization algorithms directly after initialization

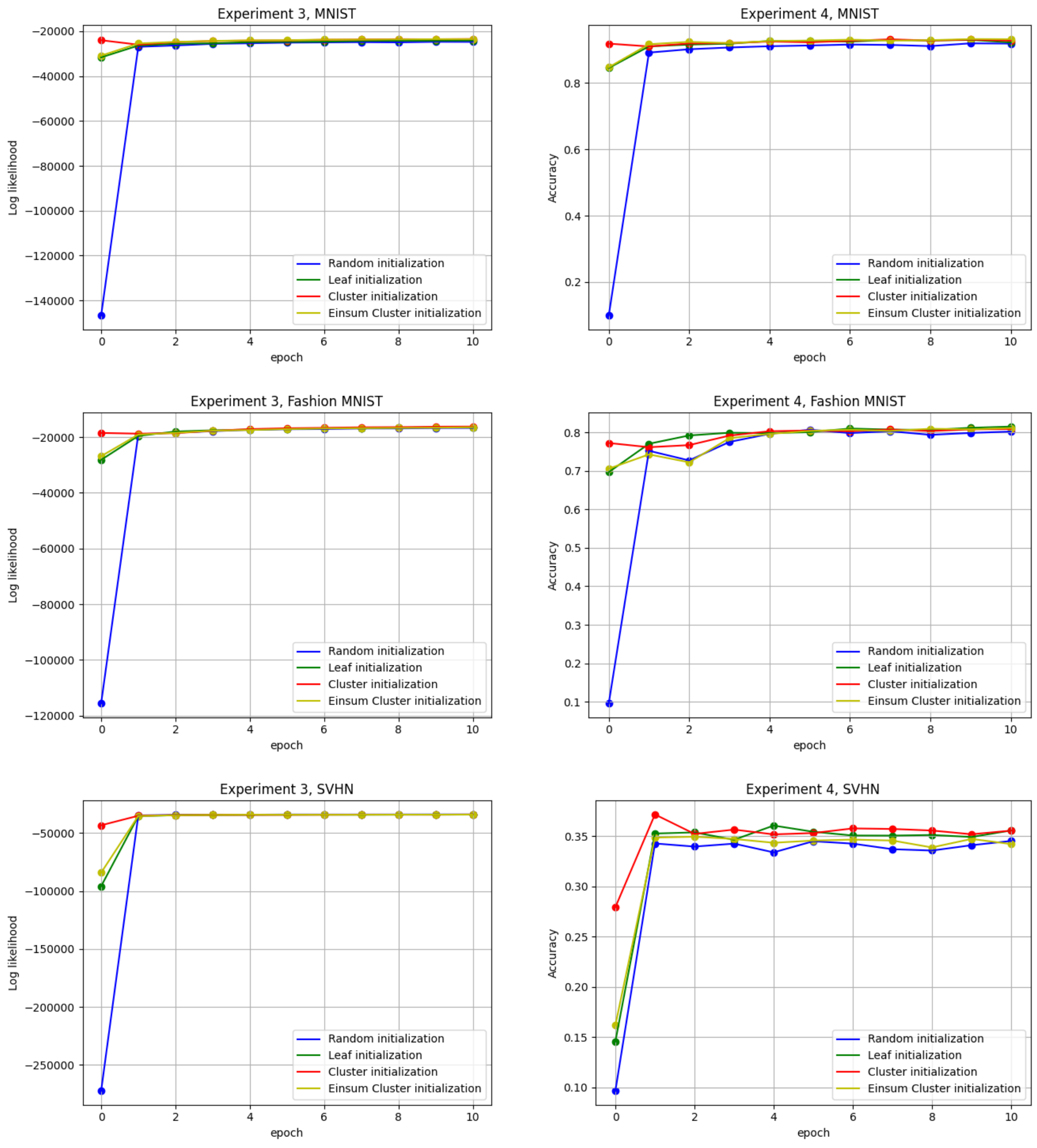

Figure 4.2: Results of different initialization methods after every epoch

4.4 Discussion

The initialization algorithms proposed in this section do improve the initial state of the Einsum Network. After training there is no difference anymore between the random initialization and the proposed initialization algorithms. This means that the initialization algorithms proposed in this chapter can be used in scenarios where the initial state of the Einsum Network is good enough to perform a certain task. In addition, this also includes scenarios where there is a lack of resources to properly train an Einsum Network.

Chapter 5

Online Adaptation

This chapter introduces Online Adaptation, which aims to improve a network at a very low cost. Online Adaptation means updating the network with new samples without completely reconstructing the network again. More precisely this means updating the weights and distributions of the Einsum Network in an online fashion.

5.1 Motivation

In the previous chapter, the main takeaway was that the initialization algorithms did only improve the state of the Einsum Network right after the initialization. When training the network the proposed initialization algorithms did not show improved results over random initialization. Because of this, the novel idea came up to use the initialization to improve the networks after training. This resulted in Online Adaptation algorithms, which improve the state of the network when new samples flow in. This has never been done in the context of probabilistic circuits. The Online Adaptation does not completely retrain the Einsum Networks, but use the initialization to slightly improve the weights and leaf distributions of the networks.

The Online Adaptation could be very useful in scenarios where initially only very few samples are available. Then when new samples present itself, they can immediately be used to improve the state of the network, without having the high cost of completely retraining the network. This also means that a certain network, which is trained on more general samples, is able to adapt to a very specific scenario. This can be best explained by an example. Right now smart lawn mowers and smart vacuum cleaners are used more and more. When they are trained on very general samples to perform its job it does not yet know the environment in which it will operate. When it is already set in use, the new environment does provide new samples to the network. It is also the case that both the lawn mower and the vacuum cleaner do not have the hardware to fully retrain a network. This is where Online Adaptation can be very useful, since it is able to improve and tailor the network to the new environment in which the smart devices operate. This can be done at a very low cost.

5.2 Algorithms

Both the Cluster initialization and the Einsum Cluster initialization use the training samples to generate clusters and then divide the samples over the network to construct a good initialization. When a new sample arrives after the initialization this sample can still be considered. This can be done in the following way.

Online Adaptation for Cluster Initialization
A new sample flows in and is assigned to the root node of the Einsum Network. This sample is

considered and propagated downwards in the network.

In a sum node, previously, when initializing the network, a clustering is performed. For this sum node the cluster centers are saved and the new sample can be assigned to one of the clusters by a distance calculation. Then the weights of this sum node are updated by using the new amount of samples in a cluster. Afterwards the sample is propagated downward to the child node which corresponds to the cluster to which the sample is assigned to.

In a product node, no clustering was executed during the initialization. Similar to the initialization algorithm, the Online Adaptation algorithm also passes the new sample to all of the children of this product node.

In a leaf node, during initialization an average value was calculated based on the samples that were assigned to that particular leaf node. This average value can be updated with the new sample and this updated average is used in the leaf node.

Summarizing, the already initialized network is adapted by feeding it a new sample. This gives rise to applications where no training is or can be used. It is expected that after adding a substantial amount of samples the network is not configured optimally and would need a complete new initialization, since the clusters would not represent the data well enough anymore. Algorithm 6 shows how this is implemented. This algorithm can be executed multiple times after the initialization.

Algorithm 6 Online Adaptation for Cluster initialization

```
Assign online samples to root node

for node in EinsumNetwork.nodes do
    if node == sum_node then
        Assign node.online_samples to appropriate cluster
        node.weights = #samples in each cluster
        for child in node.children do
            for sample in node.online_samples do
                if sample is in same cluster as child then
                    child.online_samples = child.online_samples + sample
                end if
            end for
        end for
    end if

    if node == product_node then
        for sample in node.online_samples do
            for child in node.children do
                child.online_samples = child.online_samples + sample
            end for
        end for
    end if

    if node == leaf_node then
        node.weights = update_average(node.weights, node.online_samples)
    end if
end for
```

Online Adaptation for Einsum Cluster Initialization
The Einsum Cluster initialization algorithm did take more advantage of the structure of Einsum Networks to cluster the samples of the training data into valid weights for the network. This is done by enabling different values for every spot in the vector of length K. Online Adaptation for the Einsum Cluster initialization algorithm follows the same principles as before. Namely saving the cluster centers, assigning new samples to the clusters that are already present and updating weights accordingly.

The new samples, which are assigned to a sum node, are assigned to the clusters, that were previously calculated during initialization, by a distance calculation. Afterwards the samples are propagated downwards to the correct children of the sum node, corresponding to the cluster to which the sample is assigned. In the Einsum Cluster initialization algorithm, different clusterings have been performed for every sport in the vector of length k. In the Online Adaptation algorithm this is also taken into account, such that when a sample is assigned to i.e. the first sport in the vector, then when propagating the sample downwards, it is only considered for the first spot of the child vectors. By doing this the algorithm takes full advantage of the vectorized format of the Einsum Networks.

During Einsum Cluster initialization, a clustering is also executed in a product node. When running the Online Adaptation algorithm, the same principle is used as in a sum node. Namely cluster centers are saved during initialization and with a distance calculation the samples are assigned to clusters. Here again the vectorized form of Einsum Networks is considered as explained before. After the weights are updated to the new number of samples in a cluster, the samples are propagated to the correct children corresponding to the cluster, to which the sample is assigned to.

In a leaf node, no clustering is executed but an average value is calculated during initialization. This average value is different for every place in the vector of length K. Hence, every average value is updated with the new sample assigned to that particular node and to the particular spot in the vector.

This is implemented in the Einsum Networks in the following way, depicted by Algorithm 7. Again this algorithm can be executed multiple times after the initialization.

Algorithm 7 Online Adaptation for Einsum Cluster initialization

```
Assign all online samples to every spot in the root node vector

for layer in EinsumNetwork.layers do
    if layer == EinsumMixingLayer or layer == EinsumLayer then
        for node in layer.nodes do
            for v in node.vector do
                Assign node.online_samples to appropriate cluster
                v.weights = #samples in each cluster
                for child in v.children do
                    for sample in v.samples do
                        if sample is in same cluster as child then
                            child.online_samples = child.online_samples + sample
                        end if
                    end for
                end for
            end for
        end for
    end if

    if layer == FactorizedLeafLayer then
        for node in layer.nodes do
            for v in node.vector do
                v.weights = update_average(v.weights, v.online_samples)
            end for
        end for
    end if
end for
```

5.3 Experiments

Testing the Online Adaptation algorithm can be done in various ways. Namely it is interesting to see how performance is adapting when adding samples over time. This can be done by one sample at a time but this can also be done by adding multiple samples at a time, in a batched manner. In addition to that it will be interesting to see how the algorithm will perform when adding only a small amount of samples, but also when adding a large amount of samples. Perhaps there exists a clear point where performance degrades such that a new initialization has to be executed. This gives rise to comparing a complete fresh initialization on a certain amount of samples to an initialization combined with online adapted samples.

The setup for the tests of the Online Adaptation will be as follows. A fixed amount of data samples will be used for the initialization of the network. This can be a large amount or a smaller amount. The network will be initialized and trained with these samples. This is the starting point of the graph that will be shown. Then another part of the data will be used for Online Adaptation. After each batch of Online Adaptation the performance of the network will be tested. To get good insight into the influence of the Online Adaptation, the performance without adaptation will also be tested when adding samples. This will be shown in the graphs with a black color.

Again for the Einsum Network the same setup is used as in experiment 3 and 4 of section 3.3. This means that a class discriminative structure is used. Each of the sub-Einsum Networks of the class discriminative structure is trained using SGD with the objective to optimize the log-likelihood. All the experiments in this chapter are executed on the MNIST dataset and sometimes also the Fashion MNIST is test, this is mentioned when this is the case. The standard parameters for the Einsum Network are also the same as in the previous chapters, namely the K is set to 10, the batch size is set to 100, the number of different random starts of the clustering algorithm is set to 3 and the maximum number of iterations of the clustering algorithm is set to 100.

Online Adaptation for Cluster Initialization with 45k Samples Initialization and 5k Samples Online Adaptation

In this section the results of running the Online Adaptation for Cluster initialization will be presented. 45k samples are used to initialize and train the network and 5k samples are used for the Online Adaptation. The network is adapted by adding 100 samples each time.

Figure 5.1 shows the results of this first test. The two images show how the performance adapts when running the Online Adaptation algorithm. It can be observed that, due to the Online Adaptation, the network performance makes a huge drop for both the log-likelihood as well as the classification accuracy. This means that the first batch of the Online Adaptation makes the Einsum Network worse. After this first drop, the performance is increasing slightly due to the Online Adaptation. However the performance never reaches the black reference line, which shows the performance of not doing anything after training the network. This means that it is never worth it to do the Online Adaptation for this setup. Here only the results for the MNIST dataset are shown. The Fashion MNIST dataset provides exactly the same results as are shown here. A logical next step in testing will be the Online Adaptation for the Einsum Cluster initialization. Since the Einsum Cluster initialization takes more advantage of the structure of the Einsum Networks it is expected that the Online Adaptation for the Einsum Cluster initialization is able to perform better.

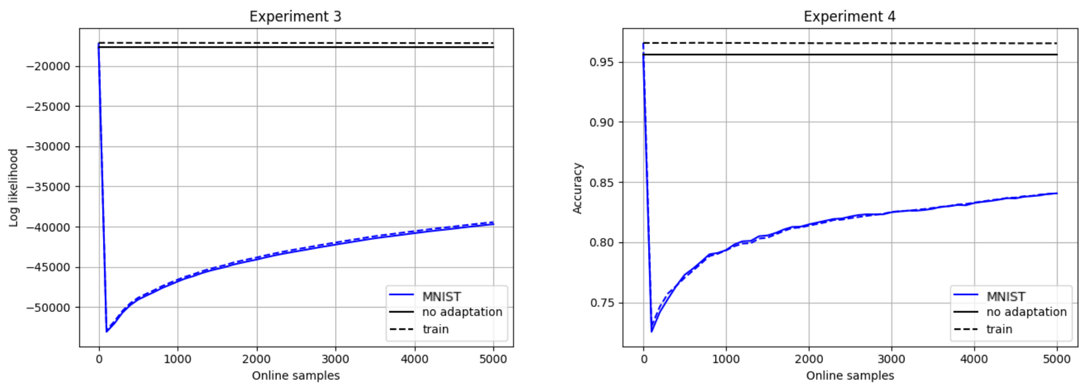

Figure 5.1: Results of Online Adaptation for Cluster initialization, 45k samples initialization, 5k samples Online Adaptation

Online Adaptation for Einsum Cluster Initialization with 45k Samples Initialization and 5k Samples Online Adaptation

Since the Online Adaptation for Cluster initialization did show a drop in performance due to the Online Adaptation, the question arises whether or not the Online Adaptation for Einsum Cluster initialization also show this drop in performance. This is tested using 45k samples for the initialization of the network and for 5k samples for the Online Adaptation.

Figure 5.2 shows the result of this testing setup. Here only the results for the MNIST dataset are shown, since the results of the Fashion MNIST dataset are very similar. As can be observed the drop in performance does not exist here, which is a positive sign. However after adaptation the performance slowly goes down by a little bit after every iteration. This again means that the Online Adaptation never outperforms the reference of not doing anything after training.

When looking very closely, experiment 3 shows very small positive results when adapting with only 500 samples. It could be that the number of samples added by the Online Adaptation should be much smaller. Therefore a next logical step in testing the Online Adaption will consist of a setup by using a smaller amount of samples for the Online Adaptation.

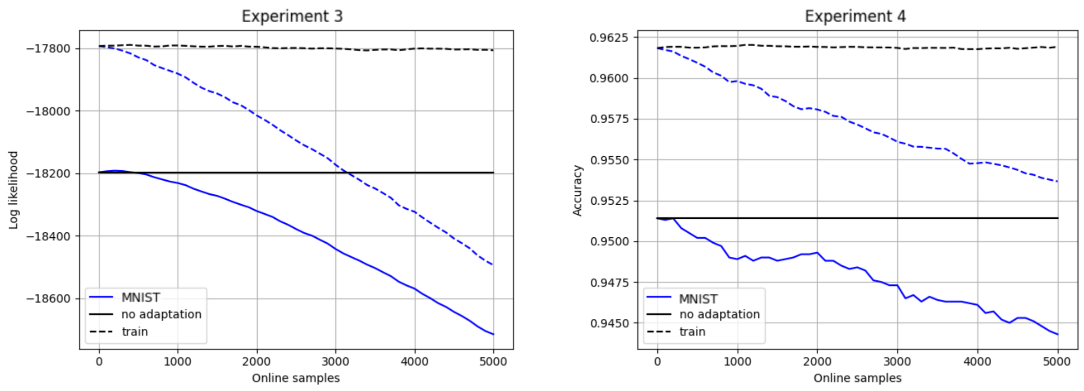

Figure 5.2: Results of Online Adaptation for Einsum Cluster initialization, 45k samples initialization, 5k samples Online Adaptation

Online Adaptation for Einsum Cluster Initialization with 45k Samples Initialization and 1k Samples Online Adaptation

In the previous setup, a very small positive result when only adapting the network with a small amount of samples is observed. Therefore a closer look to this scenario is taken in this setup. Namely the same 45k samples is used for initialization as well as training the network. Then 1000 samples are used to adapt the network.

Figure 5.3 shows the results of this experiment. The results of testing the log-likelihood go up for the first 600 samples of Online Adaptation. Here the Online Adaptation outperforms the reference line of doing nothing after training the network. This is a positive result and shows that even after training the meaning of the clustering, performed during initialization, does still hold up. When inspecting the classification accuracy, the Online Adaptation performs very similar to the reference line. This means that for a classification task the Online Adaptation does not improve the results. When adding more than 600 samples, the performance of the log-likelihood goes slowly down. This means that the network would need a fresh initialization combined with a fresh training phase.

Online Adaptation for Einsum Cluster Initialization with 1k Samples Initialization and 1k Samples Online Adaptation

Since initialization and training with a large amount of samples shows very minimal results, this setup has quite the opposite, namely it is initialized and trained with a very small amount of samples. The setup is initialized and trained with only 1k samples. Then the Online Adaptation is executed for another 1k samples. By only using 1k samples for initialization and training, it is expected that that the training did not reach its full potential. This should make room for the Online Adaptation to improve the results.

Figure 5.4 shows the result of this testing setup. As can be observed, the blue lines showing the performance of the Online Adaptation, lie above the black reference lines. This means that the Online Adaptation is doing a better job than doing nothing after training. This means that the hypothesis made did fully hold up, namely the Online Adaptation is able to improve the results

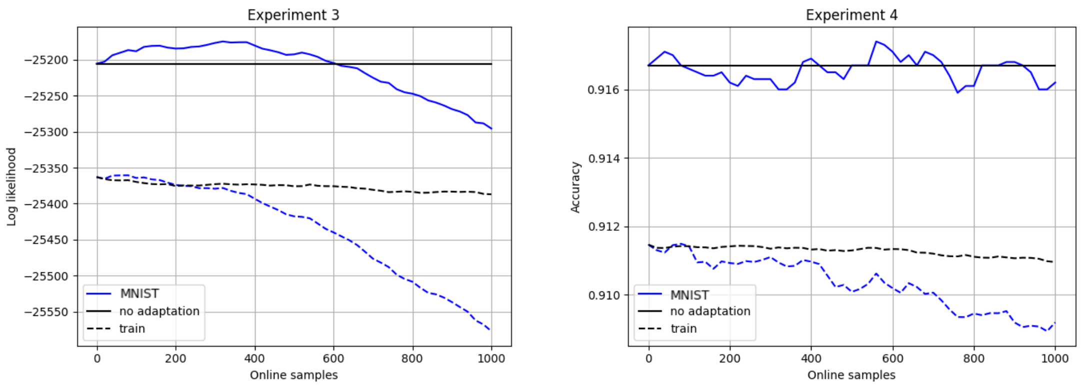

Figure 5.3: Results of Online Adaptation for Einsum Cluster initialization, 45k samples initialization, 1k samples Online Adaptation

for both the log-likelihood and the classification accuracy. However these results are only in a very specific scenario, where only a very small amount of samples is used for training and initialization.

To further explore this specific scenario, it is interesting to know how large the initialization and training set can be, such that the algorithm still shows positive results. This will be tested in further experiments

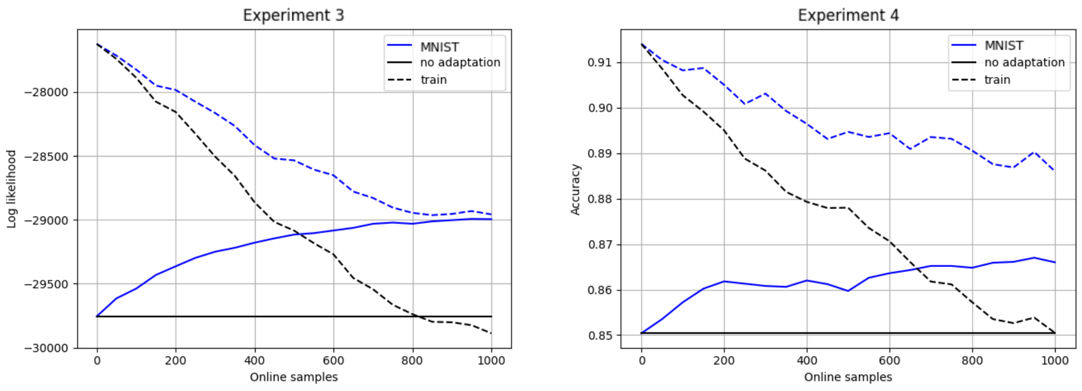

Figure 5.4: Results of Online Adaptation for Einsum Cluster initialization, 1k samples initialization, 1k samples Online Adaptation

Online Adaptation for Einsum Cluster Initialization with 2k Samples Initialization and 1k Samples Online Adaptation

After seeing some positive results in the previous testing setup, it will be tested when these positive results stop. Because of this, a setup has been created using 2k samples for initialization and training. The network is adapted using 1k samples. This means that the training is able to improve the network by a larger margin and hence it is harder for the Online Adaptation to improve on these results.

Figure 5.5 shows the result of this testing setup. The results of this setup immediately show negative results of the Online Adaptation. The Online Adaptation never outperforms the reference of doing nothing shown by the black line, for the generative task, namely improving the log-likelihood. The classification accuracy improves slightly when adapting the network. These results show that only for very small initialization and training sets the Online Adaptation actually makes sense. When using 2k samples to initialize and train the network, the network is already too good for the Online Adaptation to show results.

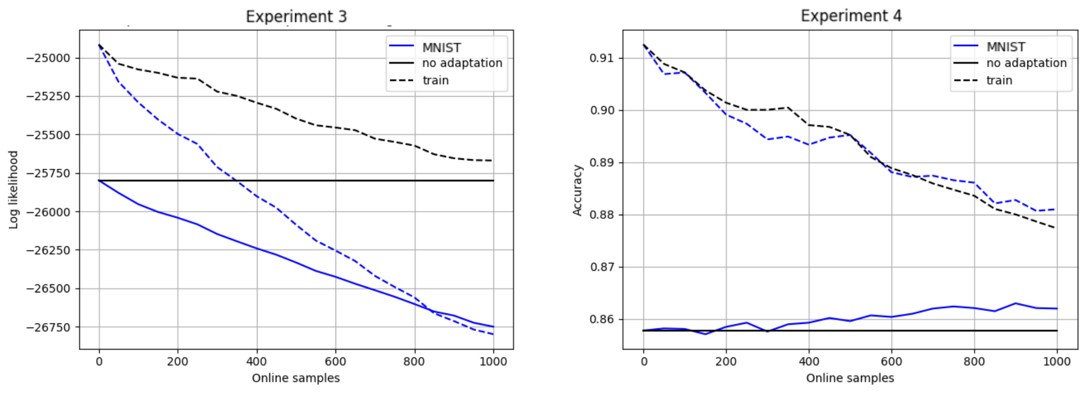

Figure 5.5: Results of Online Adaptation for Einsum Cluster initialization, 2k samples initialization, 1k samples Online Adaptation

5.4 Minimal Results

The experiments of the Online Adaptation algorithms show very minimal results. The cases where the Online Adaptation shows positive results have a very small set of samples which the network is trained with. The initializations itself did improve the results by a lot. On the contrary the Online Adaptation that adheres to this initialization does not show such good results, therefore it is interesting to find out why this is the case. This is analyzed in this section.

Clustering
A large part of both the Cluster initialization and the Einsum Cluster initialization algorithms is the clustering algorithm. The clustering algorithm that is used is KMeans algorithm from sklearn library. When the clustering is not stable, it is logical that the results for the Online Adaptation are not positive. Therefore the clustering algorithm will be analyzed more closely. To get a good clustering the algorithm requires parameters which define the number of starts and the maximum number of iterations the algorithm can make when it does not converge. In all earlier experiments these are set to 3 random starts and a maximum of 100 iterations.

A bad clustering of the samples would cause the algorithm to not perform. Of course the more starts and iterations the clustering algorithm can make, the more stable the results are. This would also take a lot more time to perform the initialization algorithms. Therefore, testing how much difference it makes when using more random starts and more iterations in the KMeans algorithm will give insight to set these parameters to appropriate values, to perform a good clustering of the samples.

To test how much of a difference these parameters make in the clustering of the data samples, the number of samples that are assigned to a different cluster are counted. Since the Einsum Cluster Algorithm performs multiple clusterings of samples in a hierarchical manner. Only the clustering in the root node is analysed here. To make sure the algorithm makes more than enough starts, this setting is changed from 3 to 3000 starts. The maximum number of iterations the algorithm can make is changed from 100 to 10000. When clustering the full training set of 50000 samples with these new settings, only 36 samples are assigned to a different cluster. This is only a very minor part of the full set and will therefore not have a large impact in the performance of the initialization and Online Adaptation algorithms.

When clustering only 1000 of the 50000 training samples, the number of differently assigned samples is a much larger part of the complete set, namely 54 out of 1000. This still is only 5.4% of the training set, but this could possibly lead to differences in performance of the initialization and the Online Adaptation algorithms. However, since the Online Adaptation algorithm did not show results when using larger training sets and the algorithm did show results when using small training sets like 1000 samples, the clustering cannot be the main cause for this.

Stable Networks
In the Online Adaptation algorithms proposed in this section, the assumption is made that the Einsum Networks are stable models that adhere to a certain clustering. However when initializing an Einsum Network with either the Cluster initialization algorithm or the Einsum Cluster initialization algorithm, the training phase could completely change the weights and the distributions such that the clustering does not have any meaning anymore. To test this, the amount of change, due to the training phase, in the parameters of the Einsum Network will be measured. The amount of change is measured per layer in the Einsum Network. For this test the smallest possible RAT-SPN structure is used, which consists of only three layers, including the leaf layer containing the distributions. The amount of change will probably increase in the deeper layers of the network since these layers depend on previous layers, this should be taken into account when analyzing the results.

The first setup that is tested is initialized using the full training set by the Einsum Cluster initialization algorithm. After the initialization the weights are saved and then the training is executed. The normalized weights of the root layer of the Einsum Network have been changed 92% on average due to the training phase. The second layer shows even more change, namely 174% on average. The leaf layer shows on average a 43% change in the distributions. This shows that the weights are completely altered and hence the meaning from the clustering is completely lost and therefore these results are in line with the results of the Online Adaptation, namely the Online Adaptation does not show positive results.

The second setup is initialized and trained using only 1k training samples since this setup did show some positive results for the Online Adaptation. Here the weights of the root layer are changed by 82% on average, the second layer is changed by 145% and the leaf distributions have been changed by 55%. Here the first two layers are a bit more stable than in the previous setup and the leaf layer did change a slightly more. However the amount of change is still very large, which means that the meaning of the clustering, performed in the initialization algorithms, is lost. This makes it very hard for the Online Adaptation algorithms to improve the networks.

5.5 LearnSPN

The LearnSPN algorithm described earlier in chapter 1 of this work makes use of clustering to learn the structure and parameters of a sum-product network. Therefore it would be interesting to see how the Online Adaptation algorithm performs when using the LearnSPN structure. The main argument for this could be that after learning a sum-product network , using the LearnSPN algorithm, the meaning of the clustering, which is used during this learning, will still have more meaning.

The SPFlow library implemented the LearnSPN algorithm and therefore the SPFlow library will be used to test the Online Adaptation algorithm. Since SPFlow uses regular sum-product networks instead of Einsum Networks the Online Adaptation algorithm will be much easier to implement compared to the Einsum Network variant. SPFlow also uses Gaussian distributions as their random variables in the leaf nodes. The Online Adaptation algorithm will be used as described in the beginning of this chapter.

The Online Adaptation for the LearnSPN structure will be tested on three different datasets, namely the spoken arabic digit dataset, the tamilnadu electricity dataset and the skin segmentation dataset [4][17][3].

Online Adaptation on Spoken Arabic Digit Dataset
The setup that is used for this experiment includes the spoken arabic digit dataset. From this dataset three different setups are tested. The differences between the setups are the number of training samples which are used to construct and train the network using the LearnSPN algorithm. The three values that are used are 1k samples, 30k samples and 120k samples. As observed in earlier experiments on Online Adaptation, the setup with the least amount of samples did perform best, due to the lack of training ability on the small amount of samples. For all three setups the Online Adaptation is tested using 1k samples. This means that 1k samples are used to adapt the network after that it is learned using the LearnSPN algorithm. The task that is executed is a classification task. The spoken arabic digit dataset contains ten classes, which are sampled in a balanced way. This means that the training samples contain a similar amount of samples from every class. This is also the case for the Online Adaptation set of samples.

Figure 5.6 shows the results of this experiment. The setup that is constructed and trained using 1k samples shows very similar results for the Online Adaptation as for no adaptation. The Online Adaptation is depicted by the blue line. These results are similar to earlier experiments. However when the networks are constructed and training using more samples, the performance of the Online Adaptation lies below the reference line of doing nothing after construction and training. This means that Online Adaptation only makes the network worse. This is also exactly the same as the results of the Online Adaptation in the Einsum Networks.

Online Adaptation on Tamilnadu Electictricity Dataset
The second dataset that is used to test the Online Adaptation is the Tamilnadu Electricity dataset. This dataset contains 20 classes, and the number of samples in the dataset are divided unbalanced over these classes. However this problem is solved by taking the same amount of samples from every class to test the Online Adaptation. Every sample in this dataset contains only four features, which is less compared to the Spoken Arabic Digit dataset, which contained 15 features per sample. Since this dataset contained a lower amount of samples, there are only two different setups tested. The first one is constructed and trained using 1k samples, the second one is constructed and trained using 10k samples. The decision has been made to test 2k samples for the Online Adaptation, which can also be observed in the results.

Figure 5.7 shows the results of these two tests. This dataset does not show similar performance between the Online Adaptation and the no adaptation for the setup which is constructed and trained using only 1k samples. Here a massive drop in performance is present for both the construction with 1k and 10k samples. The 1k samples setup shows an even larger drop in performance than the 10k samples setup. This again means that the Online Adaptation does not show positive results for the Tamilnadu Electricity dataset.

Online Adaptation on Skin Segmentation Dataset
The last dataset which is tested is the Skin Segmentation dataset. This dataset only contains two classes. This means that is an much easier dataset. However this can also mean that the clustering which is performed in the LearnSPN algorithm has a much harder time to make distinctions between samples. The Skin Segmentation dataset contains four features per sample. The two setups that are tested include again a structure which is constructed and trained using 1k samples, the second setup is constructed and trained using 30k samples. Again the samples for

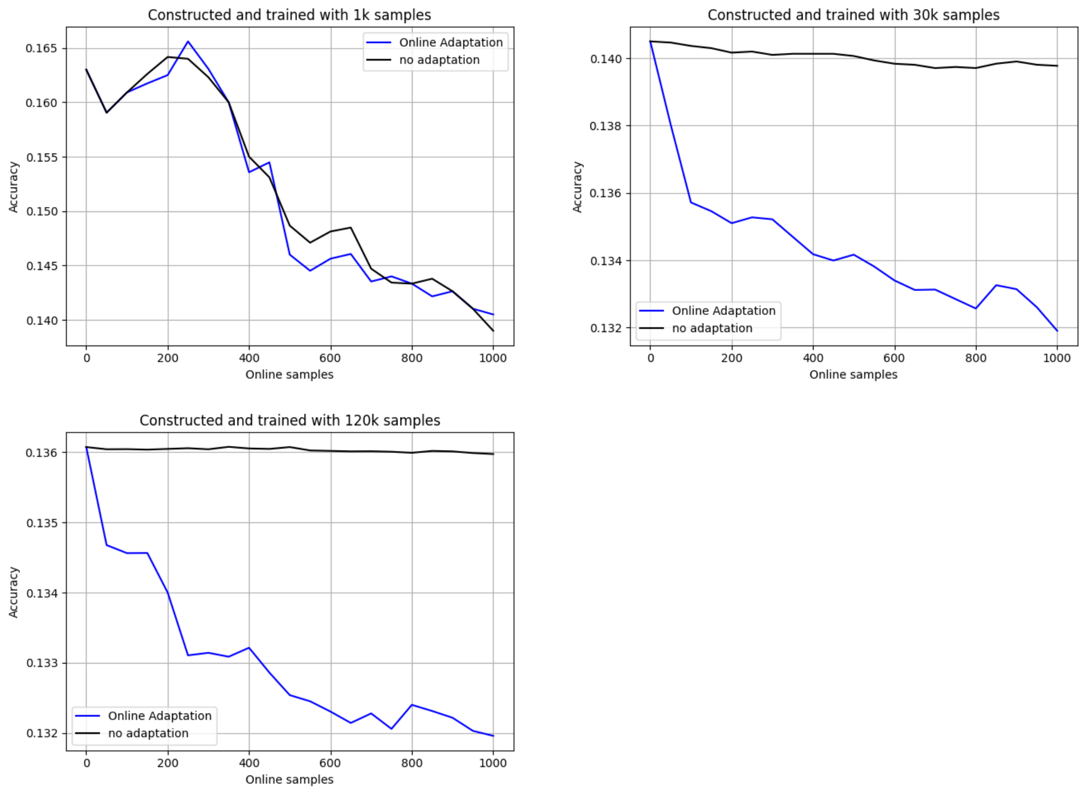

Figure 5.6: Results of Online Adaptation for the LearnSPN structure on the Spoken Arabic Digit dataset

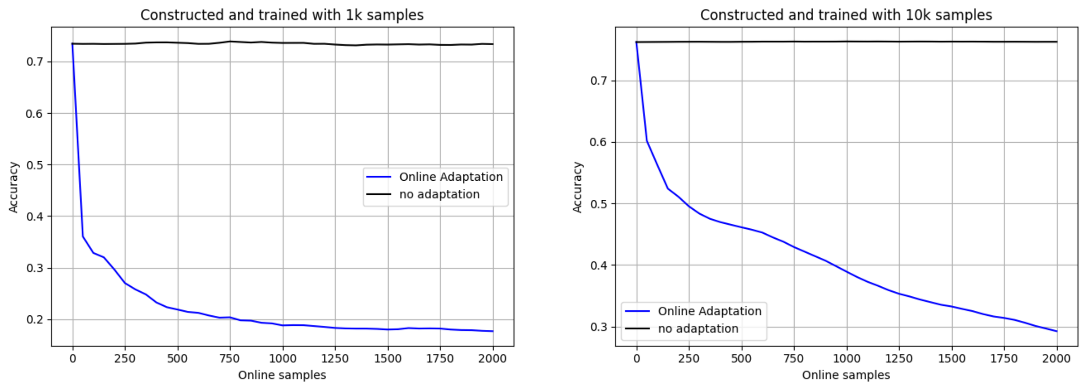

Figure 5.7: Results of Online Adaptation for the LearnSPN structure on the Tamilnadu Electricity dataset

both the training and the Online Adaptation set are sampled in a balanced way, hence for both classes an equal amount of samples. Also for this dataset a classification task is executed to test the Online Adaptation. The number of samples used for the Online Adaptation is 200 for this dataset. This is a smaller amount of samples than earlier.

Figure 5.8 shows the results for the Online Adapation on the Skin Segmentation dataset. The figure show very similar results as the Online Adaptation on the Spoken Arabic Digit dataset.

Namely the setup which is constructed and trained using 1k samples shows similar results for the Online Adapation as for the reference of no adaptation. The setup which is constructed and trained using using 30k samples shows again negative results for the Online Adaptation, since the blue line lies below the black reference line. This means that for all three datasets tested on the LearnSPN structure, the Online Adaptation never shows positive results, it is always best to not change the model after construction and training.

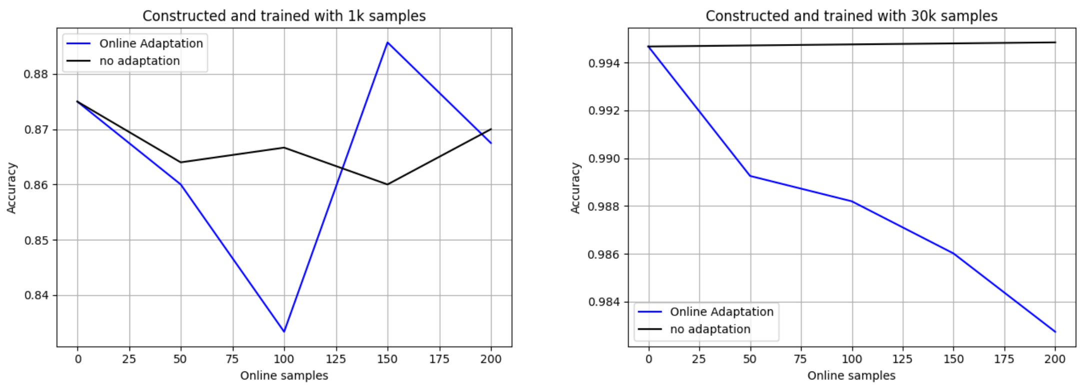

Figure 5.8: Results of Online Adaptation for the LearnSPN structure on the Skin Segmentation dataset

5.6 Discussion

The novel idea of using the initialization to later on improve the state of the network, after which the network is trained already, is presented in this chapter. The results show that only in very specific scenarios the Online Adaptation actually improves the networks. These scenarios include training setups in which very few samples are used for the actual training phase of the network.

It was also found out that the meaning of the initialization, on which the Online Adaptation algorithms are based, is completely lost after training. This would suggest that the ideas proposed in this chapter can be used later on, if there exists a better initialization which preserves its meaning after training.

Chapter 6

Conclusion and Future Work

This chapter sums up all the conclusions that can be derived from the previous chapters. This is divided in three major sections which are all chapters of this work, namely Network Analysis, Initialization and Online Adaptation. In addition, some extensions to this work are provided in a separate section.

6.1 Network Analysis

In this work several setups for Einsum Networks are tested, with the goal to find out which setup performs well on a discriminative task and which performs well on a generative task. Also the combination of performing both a generative and a discriminative task is a major interest of the chapter Network Analysis. Therefore, these setups are tested on both generative and discriminative tasks. Here an optimal setup can be derived from the test results.

When optimizing an Einsum Network for both a generative and a discriminative task, a class discriminative Einsum Network should be used. Hence a network that mixes multiple sub-Einsum Networks, one for every class in the input data. Each of the sub-Einsum Networks in the class discriminative Einsum Network should be trained generatively to perform optimally on both a generative and a discriminative tasks. This conclusion can be made since Experiment 3 and 4 of section 3.3 are the overall best performing setups.

When optimizing an Einsum Network for only a discriminative task, the conclusion can be made that the synergy between a class discriminative Einsum Network structure and a discriminative learning approach provides the best performance. This can be observed in Experiment 8 of section 3.3. In addition, section 3.2, where the RAT-SPN structure is analyzed, showed that it is important to keep the width of the network high enough compared to the depth. Namely when the width of the network is too low, the discriminative task loses performance.

When optimizing an Einsum Network for a generative task only, the most optimal setup is a standard Einsum Network. This standard Einsum Network should be trained generatively. In section 3.2, the finding was made that the performance of the generative task is highly dependant on the depth of the network. Therefore when optimizing for a generative task, it is important to create a deep network.

6.2 Initialization

Initialization is not yet been researched in the context of probabilistic circuits and is proven to be highly important in training deep models. Initialization could lead to faster training and better overall performance of the models. Therefore the algorithms proposed chapter 4 aim at improving the Einsum Networks.

The three different initialization algorithms that are proposed are Leaf initialization, Cluster initialization and Einsum Cluster initialization. All these approaches are compared to a random initialization. All three approaches consistently outperform the random initialization after only initializing the Einsum Network. Here the Cluster initialization shows the best performance of the three. However the costs of Leaf initialization are much lower than the Cluster initialization algorithm, which can be considered when using one of the algorithms.

After training the Einsum Networks, there is no clear performance difference between the random initialization and one of the proposed algorithms. The only dataset that shows a difference is the more complicated SVHN dataset when performing a classification task. Here the Cluster initialization slightly outperforms the other methods.

In situations where there is very little time and resources available to train a complete network, these initialization can be used to perform a certain task, without even training the network. This can be desirable in certain use cases.

6.3 Online Adaptation

The novel idea of using the initialization methods proposed in chapter 4 to improve the network after training is presented in chapter 5. The Online Adaptation algorithm, aims at improving the Einsum Networks after it is already trained. This would be useful in scenarios where there are no resources or no time to completely retrain the Einsum Network.

When new training samples present itself, the Online Adaptation should be able to improve networks in an online fashion. The Online Adaptation algorithm is based on the Cluster or Einsum Cluster initialization which are presented in chapter 4. Re-using the clustering, performed in the initialization algorithms, to later on improve the networks, after training, is a novel idea, which is implemented in the Online Adaptation algorithm.

However the results of the Online Adaptation did not show remarkable results, they mostly made the networks worse. Only in a few cases, where the networks were initially trained with very few samples, the Online Adaptation algorithm shows positive results. To verify the algorithm, it is also tested on the LearnSPN structure, which makes use of clustering while building the sum-product networks. Also here the Online Adaptation did not show positive results. Section 5.4 shows that the meaning of the clustering, performed in the initialization algorithms, is lost after training, which results in the Online Adaptation algorithm not performing as expected.

6.4 Possible Extensions

Using initialization algorithms to not only improve the state of the networks right after the initialization, but also increase performance after training still remains to be interesting topic to perform further research in. In addition improving probabilistic circuits in an online fashion is still a very new topic. Little research has been conducted in this field. Therefore further research into this topic would be a very logical next step.

In this work, the following statement has been made. After training a probabilistic circuit the meaning of the clustering, performed during initialization, is completely lost. This results in a negative performance of the Online Adaptation algorithm. When a clustering could be made after which the network is already trained, which fits the parameters and leaf distributions of the probabilistic circuit, it would mean that the clustering still has all its meaning. When using this clustering for the Online Adaptation algorithm, the networks should improve. It would be very interesting to see the results of this particular experiment.

In addition, it will be very interesting to see how the probabilistic circuits, including Einsum Networks, evolve in the next couple of years.

Bibliography

[1] Tameem Adel, David Balduzzi, and Ali Ghodsi. Learning the structure of sum-product networks via an SVD-based algorithm. In *UAI*, pages 32–41, 2015. 12

[2] Mohamed R Amer and Sinisa Todorovic. Sum product networks for activity recognition. *IEEE transactions on pattern analysis and machine intelligence*, 38(4):800–813, 2015. 16

[3] Rajen Bhatt and Abhinav Dhall. Skin Segmentation dataset. 2015. https://www.openml.org/d/1502. 46

[4] Data Collected by the Laboratory of Automatic and Algeria Signals, University of Badji-Mokhtar Annaba. Spoken Arabic Digit dataset. 2008. https://www.openml.org/d/1503s. 46

[5] Wei-Chen Cheng, Stanley Kok, Hoai Vu Pham, Hai Leong Chieu, and Kian Ming A Chai. Language modeling with sum-product networks. In *Fifteenth Annual Conference of the International Speech Communication Association*, 2014. 16

[6] Diarmaid Conaty, Jesús Martínez Del Rincon, and Cassio P De Campos. Cascading sum-product networks using robustness. In *International Conference on Probabilistic Graphical Models*, pages 73–84. PMLR, 2018. 15

[7] Olivier Delalleau and Yoshua Bengio. Shallow vs. deep sum-product networks. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011. 11

[8] Aaron Dennis and Dan Ventura. Learning the architecture of sum-product networks using clustering on variables. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. 12

[9] Aaron Dennis and Dan Ventura. Greedy structure search for sum-product networks. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015. 13

[10] Aaron Dennis and Dan Ventura. Autoencoder-enhanced sum-product networks. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1041–1044. IEEE, 2017. 15

[11] Aaron Dennis and Dan Ventura. Online structure-search for sum-product networks. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 155–160. IEEE, 2017. 13

[12] Mattia Desana and Christoph Schnörr. Learning arbitrary sum-product network leaves with expectation-maximization. *arXiv preprint arXiv:1604.07243*, 2016. 15

[13] Robert Gens and Pedro Domingos. Discriminative learning of sum-product networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. 14

[14] Robert Gens and Domingos Pedro. Learning the structure of sum-product networks. In *International conference on machine learning*, pages 873–880, 2013. 8

[15] Robert Gens and Domingos Pedro. Learning the structure of sum-product networks. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 873–880, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. 12

[16] Priyank Jaini, Amur Ghose, and Pascal Poupart. Prometheus: Directly learning acyclic directed graph structures for sum-product networks. In *International Conference on Probabilistic Graphical Models*, pages 181–192. PMLR, 2018. 13

[17] k Kalyani. Tamilnadu Electricity Board Hourly Readings dataset. 2013. https://www.openml.org/d/40985. 46

[18] Ching-Yun Ko, Cong Chen, Yuke Zhang, Kim Batselier, and Ngai Wong. Deep compression of sum-product networks on tensor networks. *arXiv preprint arXiv:1811.03963*, 2018. 15

[19] Viktoriya Krakovna and Moshe Looks. A minimalistic approach to sum-product network learning for real applications. *arXiv preprint arXiv:1602.04259*, 2016. 13

[20] Sang-Woo Lee, Min-Oh Heo, and Byoung-Tak Zhang. Online incremental structure learning of sum–product networks. In *International Conference on Neural Information Processing*, pages 220–227. Springer, 2013. 12

[21] James Martens and Venkatesh Medabalimi. On the expressive efficiency of sum product networks. *arXiv preprint arXiv:1411.7717*, 2014. 11

[22] Denis D Mauá, Fabio G Cozman, Diarmaid Conaty, and Cassio P Campos. Credal sum-product networks. In *Proceedings of the Tenth International Symposium on Imprecise Probability: Theories and Applications*, pages 205–216. PMLR, 2017. 15

[23] Mazen Melibari, Pascal Poupart, Prashant Doshi, and George Trimponias. Dynamic sum product networks for tractable inference on sequence data. In *Conference on Probabilistic Graphical Models*, pages 345–355. PMLR, 2016. 13

[24] Alejandro Molina, Antonio Vergari, Nicola Di Mauro, Sriraam Natarajan, Floriana Esposito, and Kristian Kersting. Mixed sum-product networks: A deep architecture for hybrid domains. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. 15

[25] Aniruddh Nath and Pedro Domingos. Learning tractable probabilistic models for fault localization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016. 16

[26] Robert Peharz, Bernhard C Geiger, and Franz Pernkopf. Greedy part-wise learning of sum-product networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 612–627. Springer, 2013. 12

[27] Robert Peharz, Robert Gens, and Pedro Domingos. Learning selective sum-product networks. In *LTPM workshop*, 2014. 14

[28] Robert Peharz, Robert Gens, Franz Pernkopf, and Pedro Domingos. On the latent variable interpretation in sum-product networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(10):2030–2044, 2016. 6, 14

[29] Robert Peharz, Steven Lang, Antonio Vergari, Karl Stelzner, Alejandro Molina, Martin Trapp, Guy Van den Broeck, Kristian Kersting, and Zoubin Ghahramani. Einsum Networks: Fast and scalable learning of tractable probabilistic circuits. *arXiv preprint arXiv:2004.06231*, 2020. 1, 2, 8, 16, 32

[30] Robert Peharz, Sebastian Tschiatschek, Franz Pernkopf, and Pedro Domingos. On theoretical properties of sum-product networks. In *Artificial Intelligence and Statistics*, pages 744–752, 2015. 11

[31] Robert Peharz, Antonio Vergari, YooJung Choi, and Guy Van den Broeck. Probabilistic circuits: Representations, inference, learning and theory. 2020. https://www.youtube.com/watch?v=2RAG5-L9R70&ab_channel=UCLA-StarAI. 2

[32] Robert Peharz, Antonio Vergari, Karl Stelzner, Alejandro Molina, Martin Trapp, Kristian Kersting, and Zoubin Ghahramani. Probabilistic deep learning using random sum-product networks. *arXiv preprint arXiv:1806.01910*, 2018. 7, 13, 23

[33] Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 689–690. IEEE, 2011. 11

[34] Tahrima Rahman and Vibhav Gogate. Merging strategies for sum-product networks: From trees to graphs. In *UAI*, 2016. 13

[35] Abdullah Rashwan, Pascal Poupart, and Chen Zhitang. Discriminative training of sum-product networks by extended baum-welch. In *International Conference on Probabilistic Graphical Models*, pages 356–367. PMLR, 2018. 15

[36] Abdullah Rashwan, Han Zhao, and Pascal Poupart. Online and distributed Bayesian Moment Matching for parameter learning in sum-product networks. In Arthur Gretton and Christian C. Robert, editors, *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 1469–1477, Cadiz, Spain, 09–11 May 2016. PMLR. 14

[37] Amirmohammad Rooshenas and Daniel Lowd. Learning sum-product networks with direct and indirect variable interactions. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, page I–710–I–718. JMLR.org, 2014. 12

[38] Bruno Massoni Sguerra and Fabio G Cozman. Image classification using sum-product networks for autonomous flight of micro aerial vehicles. In *2016 5th Brazilian Conference on Intelligent Systems (BRACIS)*, pages 139–144. IEEE, 2016. 16

[39] Or Sharir and Amnon Shashua. Sum-product-quotient networks. In *International Conference on Artificial Intelligence and Statistics*, pages 529–537. PMLR, 2018. 15

[40] Martin Trapp, Tamas Madl, Robert Peharz, Franz Pernkopf, and Robert Trappl. Safe semi-supervised learning of sum-product networks. *arXiv preprint arXiv:1710.03444*, 2017. 15

[41] Martin Trapp, Robert Peharz, Hong Ge, Franz Pernkopf, and Zoubin Ghahramani. Bayesian learning of sum-product networks. *arXiv preprint arXiv:1905.10884*, 2019. 14

[42] Jos van de Wolfshaar and Andrzej Pronobis. Deep generalized convolutional sum-product networks for probabilistic image representations. *arXiv preprint arXiv:1902.06155*, 2019. 16

[43] Antonio Vergari, Nicola Di Mauro, and Floriana Esposito. Simplifying, regularizing and strengthening sum-product network structure learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 343–358. Springer, 2015. 13

[44] Jinghua Wang and Gang Wang. Hierarchical spatial sum–product networks for action recognition in still images. *IEEE Transactions on Circuits and Systems for Video Technology*, 28(1):90–100, 2016. 16

[45] Lodewyk FA Wessels, Etienne Barnard, et al. Avoiding false local minima by proper initialization of connections. *IEEE transactions on neural networks*, 3(6):899–905, 1992. 2

[46] Jim YF Yam and Tommy WS Chow. A weight initialization method for improving training speed in feedforward neural network. *Neurocomputing*, 30(1-4):219–232, 2000. 2

[47] Han Zhao, Tameem Adel, Geoff Gordon, and Brandon Amos. Collapsed variational inference for sum-product networks. In *International Conference on Machine Learning*, pages 1310–1318. PMLR, 2016. 14

[48] Han Zhao, Mazen Melibari, and Pascal Poupart. On the relationship between sum-product networks and Bayesian networks. In *International Conference on Machine Learning*, pages 116–124. PMLR, 2015. 12