Non-Monotonic Mixture Models: Enhancing the Flexibility of Mixtures

Patrick Tourniaire

4th Year Project Report
Artificial Intelligence and Computer Science
School of Informatics
University of Edinburgh

2024

Abstract

Mixture models are often used to model complex data distributions where single probability distributions are not enough. Traditionally, mixtures cannot subtract away probability mass which limits its expressiveness as complex shapes such as a ring will require a substantial number of components to effectively model. Therefore previous work has been done on enabling non-monotonic mixtures (NMMMs) which enable mixtures to subtract away components by allowing for negative weights. However this work is not generalisable for deep mixtures or mixtures of multiple parametric families. This project will provide such a framework, which we experiment with assuming a shallow mixture of multivariate Gaussians. We will demonstrate that this model achieves comperable results to monotonic mixtures whilst requiring considerably fewer parameters to be optimised. Thus, providing a strong foundation for generalised NMMMs.

Acknowledgements

I would like to thank my supervisor Antonio Vergari for providing this interesting project to work on and the many meetings which he made time for to guide me through this project. Additionally, I would like to thank my parents and the rest of my family for helping me get through these 4 years when times were tough and the money was short.

Table of Contents

Chapter 1

Introduction

Estimating probability distributions of data is a core part of unsupervised learning. Distributions are learned by observing patterns in data, and these patterns can quickly become complex as the amount of data increases. To model such distributions, mixture models are often used [19, 21]. Mixture models are one way to effectively capture such complex data distributions. Weights are attached to each combined distribution, representing the prior probability of a random variable belonging to that density. Therefore, it is assumed that these weights are positive and sum to one, such that the mixture model is a valid probability density function.

However, this limits the ways these mixture models can effectively model complex patterns in data. By allowing negative weights, it is possible to subtract densities, assuming that the resulting mixture model is a valid probability density. Such mixture models are called Non-Monotonic Mixture Models or NMMMs for short. So far, there has been work on practically achieving NMMMs, which have used simple parametric probability distributions such as Gaussians [39, 26] and Weibulls [14]. However, the techniques used to achive these NMMMs are not generalisable for deep mixture models and combinations of parametric families, therefore, this project will create a generalised framework for learning NMMMs.

1.1 Objectives and Contributions

The main objective of this project is to create a foundation for generalised NMMMs, which in the future can easily be extended to support deep mixtures or mixtures of multiple parametric families. Thus, this report will set out to answer the following research questions.

Research question 1: is it feasible to create a generalised framework for NMMMs which can support deep mixtures and multiple parametric families?

Research question 2: can such a generalised framework implemented as a shallow non-monotonic Gaussian mixture (NMGMM) converge to a lower loss than shallow monotonic Gaussian mixtures with the same number of components for shapes with negative spaces?

Research question 3: can the NMGMM provide more generalised fits compared to monotonic GMMs?

Based on these research questions, we have established the following contributions.

- Constructed a theoretical framework for generalised NMMMs.
- Performed a thorough analysis on the new parameters in this theoretical framework, assuming a shallow non-monotonic Gaussian mixture model, which was used to compare against shallow GMMs.
- Built a learning framework which can enable effective learning of NMMMs for shallow and deep mixtures.
- Trained NMGMMs using this generalised framework for identifying positive and negative component and compare against the equivalent GMM.

1.2 Project Outline

Chapter 2: Here the necessary theory and notion will be presented which will serve as a foundation for appreciating the motivation behind NMMMs. This will be followed by an analysis of the past research and a critique for why these methods are not generalisable. To build on that, we present the relevant components of Probabilistic Circuits as a framework which will be used to construct a generalisable NMMM.

Chapter 3: Following relevant components of Probabilistic Circuits, we will present our theoretical framework for NMMMs. Further, we construct a shallow NMGMM using this framework, which will be experimented on.

Chapter 4: Using the theoretical framework for the shallow NMGMM, we will perform preliminary experiments to better understand how the parameters change with respect to a GMM equivalent.

Chapter 5: Based on the findings in Chapter 4, we present a learning framework which will optimise the NMGMM and address any limitations observed in the preliminary experiments.

Chapter 6: Here the experimental setup for the NMGMM will be presented, as well as the dataset which we will be using to compare it against the monotonic Gaussian mixture. Additionally, we will perform hyperparameter optimisation to identify optimal hyperparameter configurations to be used to present the results.

Chapter 7: In this chapter we will present the results of our experiments, and we will analyse the possible limitations of the NMGMM and compare it against a monotonic Gaussian mixture.

Chapter 8: Finally, we will conclude our findings and present our conclusions for the research questions above. Additionally, we will highlight some possible future directions which further research can take.

Chapter 2

Background

In this chapter, we will present the basic theory as a prerequisite of this project, which will be followed by a literature review of the latest work on learning simple NMMMs. This will establish the notations used throughout the project, and solidify the motivations behind NMMMs from a technical perspective. Which will also illuminate the limitations of the previous literature. We will then present the necessary components of Probabilistic Circuits (PCs) which will be used to address these limitations in the following chapters.

2.1 Theory and Notation

Before presenting the relevant theory and notation, it is important to understand which interpretation of probability this project is assuming. There are different views on probability, however this project will be based on the objective Bayesian interpretation as outlined by Jon Williamson [37]. Through the lens of this interpretation, probabilistic models try to model uncertainty in the world through observations. Such observations are seen as evidences for the uncertainty of a model. Another term for such evidences are random variables, and they form the basis of any Bayesian probabilistic model.

A random variable (RV) can be interpreted as possible states of a model. This project denotes RVs with capitalised letters X and some realisation in lowercase x, where $x \in Val(X)$ defines the total state space of X. Further, any set of all possible RVs are denoted in bold $\mathbf{X}$. Lastly, RVs can either be discrete or continuous. The difference between these RV types will affect what type of probability function is produced. Suppose p_m is some probability distribution which is defined over a continuous RV X. Such a distribution is classified as a probability density function (PDF) and is defined by $p_m(X)$. Further, for $p_m(X)$ to be a valid PDF, the following conditions have to be met.

Definition 1 (Probability Density Function) *Assume $p(X)$ is some probability distribution defined over some continuous RV X, then $p(X)$ has to be constrained by the following conditions such that it is a valid PDF.*

$$p(x) \geq 0.\forall x \;\; and \;\; \int_{\infty}^{-\infty} p(x)\, dx = 1 \tag{2.1}$$

On the other side, for some discrete RV Y defined over p_n, the probability distribution p_n will be classified as a probability mass function (PMF). These two classifications of probability functions, form the foundation of most probabilistic modelling.

So far, we have assumed that the probability functions are non-parametric. However, in this project we will only concern ourselves with mixtures of parametric probability distributions, which can either be discrete or continuous. Daniel S. Wilks [36] refers to a parametric distribution as an abstract mathematical form, where its nature or shape is determined by a set of parameters. These parameters are optimised to fit the set of observations which the parametric distribution is modelling. There are many parametric families such as Gaussians, Weibulls and Bernoullis, which are either PMFs or PDFs and their characteristic shape can make them better candidates for certain data distributions. The process of selecting the parametric family for some set of observations to model, is often done by evaluating the closeness to the observed data when their parameters are optimised. On the other hand, real world data can have very complex patterns over multiple dimensions and therefore we need a way to combine multiple parametric distributions to be able to capture such complex patterns.

2.1.1 Finite Mixture Models

This is where Finite mixture models (FMMs) come into the picture, as they provide a way to model a mixture of probability distributions to data where a single distribution is not able to accurately represent the features. This flexibility of FMMs is indeed attractive for many different practical applications such as automatic speech recognition systems [17, 40], machine vision for visual clustering [29, 5], and other fields such as bioinformatics [13]. This section aims to introduce FMMs as described by Geoffrey J. McLachlan, et al. [20]. An FMM is a model which combines 2 or more probability densities. Suppose, we want to create an FMM with k number of parametric distributions, then a d-dimensional RV $X = [x_1, x_2, \dots, x_d]^T$ follows a finite mixture distribution, if its PDF can be written as

$$p(X;\theta) = \sum_{m=1}^{k} \alpha_m p(X;\theta_m) \tag{2.2}$$

Where $\theta = \{\theta_1, \theta_2, \dots, \theta_k\}$ form the parameters for each parametric distribution in the mixture. To support data of multiple dimensions, the parametric distributions are generally multivariate. Meaning that the joint distribution for each dimension in one mixture component generalises to higher dimensions. Further, each α represents the prior probability of an RV realisation belonging to some parametric distribution $p(X;\theta_m)$. As these weights are treated as prior probabilities for a point belonging to one of the components, they are constrained by the following conditions.

$$\forall \alpha \geq 0 \text{ and } \sum_{i=0}^{k} \alpha_i = 1 \tag{2.3}$$

FMMs form the generalised framework for how families of parametric probability distributions can be combined to model more complex shapes. However, the specific details for practically achieving any FMM depends on the parametric family assumed. For this project, we will aim to provide a generalised foundation for NMMMs which will be experimented on using a mixture of Gaussian distributions, also known as a Gaussian mixture model (GMM).

2.1.2 Gaussian Mixture Models

Gaussian mixture models (GMMs) are FMMs, where the model parameters are the parameters of a mixture of Gaussian distributions, which can either be bivariate or multivariate. A bivariate Gaussian mixture is limited to only two dimensions, whereas a multivariate Gaussian is generalised to any number of dimensions [21]. Therefore, this project will be assuming mixtures of multivariate Gaussian, such that our work is generalisable for any number of dimensions.

A multivariate Gaussian distribution is parametrised by a vector μ and a matrix Σ. μ is a D dimensional vector, where each entry μ_i represents the mean of a Gaussian distribution along dimension i. Further, Σ is a covariance matrix, where the diagonal entries represent the squared variance along the different dimensions and the off diagonal entries represent the correlation between the dimensions. Thus, Σ is represented as a square matrix of size $D \times D$. Therefore, given a vector μ and a covariance matrix Σ, the PDF of the multivariate Gaussian is given by.

$$P(X;\Sigma,\mu) = \frac{1}{\sqrt{(2\pi)^D |\Sigma|}} \exp\left(-\frac{1}{2}(X-\mu)^\top \Sigma^{-1}(X-\mu)\right) \tag{2.4}$$

Additionally, the PDF in Equation 2.4 assumes an invertible covariance matrix. In fact, any valid covariance matrix is invertible, therefore it is important to establish what a valid covariance matrix is. A covariance matrix can be any square matrix, as long as the square matrix is symmetric and positive semi-definite. A positive semi-definite Σ is a matrix which follows $\mathbf{x}^\top \Sigma \mathbf{x} \geq 0$ for any D dimensional real vector $\mathbf{x}$ [25]. As long as this condition is upheld, Σ can be considered a valid covariance matrix.

2.1.2.1 Learning Gaussian Mixtures

For learning optimal configurations for the parameters of a Gaussian mixture or any FMM, there are multiple approaches one could take. Traditionally, the expectation maximisation algorithm (EM) is used [21], however, as outlined by Alexander Gepperth, et al. [9] the EM algorithm is a batch-type optimiser. This type of optimiser, takes the entire training data for each optimisation step, which can quickly become inefficient for higher dimensions and large amounts of training data.

To address this in the past, researchers have implemented a stochastic version of the EM algorithm called SEM [6, 15, 30]. The most typical version of the SEM algorithm was presented by Olivier Cappé et al. [2], however, this implementation introduces additional hyperparameters such as step size, mini-batch size and step reduction. However, for certain models, these additional hyperparameters can be undesirable as it can further complicate the optimisation process. This has been addressed in the past by using stochastic gradient descent (SGD) for optimising GMMs using mini-batches [9, 11, 12]. Which only introduces two additional hyperparameters.

Stochastic gradient descent (SGD) is a gradient based optimiser which can support optimisation over mini-batches. Mini-batches are subsets of the training data, where its size is defined by the batch size hyperparameter. Therefore, SGD will segment the training data into multiple batches and for one iteration it will optimise the parameters of some Gaussian mixture for each mini-batch. This is implemented by constructing some loss function which computes the cost over the entire mini-batch. Thus, the objective of SGD is to reduce the output of this loss function by computing the gradients of the loss function with respect to the model parameters, which in the case of a GMM is a set containing parameters μ and Σ for each component. According to Ian Goodfellow et al. [10] the SGD optimisation step can be summarised according to Equation 2.5 and 2.6.

$$\hat{g} := \frac{1}{m} \nabla_\theta \left[\sum_{i=0}^{m} L(f(x_i;\theta), y_i) \right] \tag{2.5}$$

Where $\hat{g}$ represents the estimated gradient given some loss function and some labels y, further the gradient computation is averaged over the size of one mini batch. The new parameters are estimated using this gradient along with a learning rate ε.

$$\theta := \theta - \varepsilon \hat{g} \tag{2.6}$$

For using SGD to optimise GMMs, however, we modify the loss function such that it computes the average negative log likelihood over the minibatch. Therefore, we reframe the computation for $\hat{g}$ in the optimisation step according to Equation 2.7, where $P(X;\theta)$ denotes the PDF of the GMM.

$$\hat{g} := \frac{1}{m} \nabla_\theta \left[\sum_{i=0}^{m} -log\left(p(X_i;\theta)\right) \right] \tag{2.7}$$

Equation 2.7 highlights the versatility of using SGD as an optimiser for training any mixture model. However, depending on the parametric family which is assumed, different constraints on the parameters can apply. As SGD does not enforce any such constraints, Alexander Gepperth et al. [9] address this by performing two reparametrisation techniques, which are presented in Subsection 2.1.2.2.

2.1.2.2 Reparametrisation of GMMs with SGD

To account for the stochasticity introduced by SGD, the parameters of the GMM are reparameterised. As highlighted previously, a valid covariance matrix is a matrix which is positive and semi-definite. To account for this constraint, Alexander Gepperth et al. [9] used Cholesky decomposition to construct a valid covariance matrix. Michael Parker [23] outlines this method in his chapter on matrix inversion, where Cholesky decomposition takes some lower triangular matrix L with all positive entries along its main diagonal, such that a matrix Σ is positive semi-definite as it is defined by $\Sigma = LL^\top$. Thus, SGD will optimise a set of lower triangular matrices for each component, which are then transformed according to the Cholesky decomposition to produce a valid covariance matrix Σ.

Cholesky decomposition ensures that the covariance matrices are valid, however, the priors of the mixture also require some constraints in a monotonic Gaussian mixture. To account for this, Alexander Gepperth et al. [9] perform the softmax over all the priors which SGD optimises, which ensures that the priors are positive and sum to one. The softmax function as formulated by John Bridle, et al. [1] is a normalised exponential, which is applied to the optimised weights from SGD to compute the priors used for the Gaussian mixture.

2.1.3 Limitations of Monotonic Mixtures

As we have now seen, mixture models bring some additional flexibility in the world of probabilistic modelling. However, such monotonic mixtures are limited in that they cannot subtract away probability densities. For example, with a complex shape such as a ring, a monotonic mixture would have to place many small components along the ridge of the ring. However, the most effective way would obviously be to have a circle or component which encloses all the point and have a second component in the middle to ’cut’ out the probability mass in the middle. This is the idea behind non-monotonic mixtures, as they remove the conditions on the priors presented in Equation 2.3, such that it can effectively subtract away the probability mass. However, as mixtures are expected to be valid PDFs or PMFs, the technique for achieving this has to ensure that there are no negative values and that the mixture integrates to one. There has been research looking into practically achieving such models, and this will form the context of this project.

2.2 Subtracting Densities in FMMs

In this section, we will provide a brief outline of the work that has been done on creating NMMMs. Which will allow us to understand the necessary foundation for generalising to deep mixture models and mixtures of multiple parametric families, as this project aims to achieve. The current work has provided learning schemes for shallow NMMMs on a few families of parametric probability distributions, such as Gaussians [39, 26] and Weibulls [14].

Authors have shown how simple mixtures of GMMs and Weibull mixtures can allow for negative weights. All authors have discarded the first constraint of Equation 2.3 to allow for negative weights, however for the NMMM to be a valid PDF they keep the sum constraint in Equation 2.3, as well as some additional constraints based on the techniques they are using. These techniques assume a shallow mixture, and some techniques such as the work by R. Jiang, et al. [14] are only applicable to their mixture families. we intend to present these methods and what can be learned from them, which will serve as the foundation for generalising to deep mixtures.

A crucial step in constructing a learning scheme for NMMMs is to ensure that the resulting mixture is a valid PDF. There are many approaches one could take to achieve this, one way is to initialise the number of negative and positive components and enforce a set of constraints to ensure that the mixture is a valid PDF. This is the approach which researchers Baibo Zhang, et al. [19] took. They proposed a technique where negative components are estimated dependably on positive components, which was achieved by constraining the negative component with respect to a positive component. Therefore, they introduce some covariance constraints on the negative components to ensure that the negative component does not spill over the positive component, thus ensuring that the mixture does not output negative probabilities.

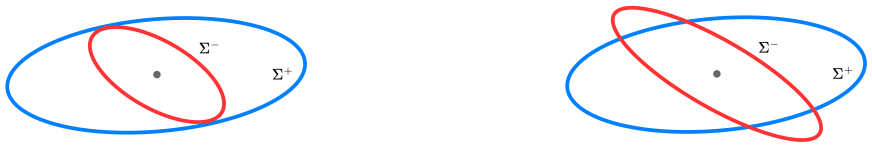

Figure 2.1: Illustration inspired by the Baib Zhang et al. paper [19] showing their covariance constraint on negative components

Further, they propose a learning scheme which is a modified version of the traditional Expectation Maximisation (EM) algorithm used for monotonic mixture models. In their approach, they estimate the positive and negative components (denoted by p^+ and p^- respectively) by first fixing the negative component and performing EM on the positive component. p^- is initialised by performing k-means, and is estimated in the next step by fixing p^+. This process is repeated until a convergence is met. This approach achieved promising results on 2-dimensional data with complex patterns, which monotonic mixtures require a considerable amount of components to achieve.

However, as Baibo Zhang et al. are using a modified version of the EM algorithm it cannot be effectively applied to deep mixture models, in fact this also applies to the approach presented by R. Jiang on non-monotonic Weibulls [14] and Guillaume Rabusseau et al.'s work on non-monotonic GMMs [26]. The reason for this, is that the E-step in the EM algorithm can quickly become intractable for deep mixtures. As discussed by Cinzia Viroli, et al. [33] the computation of the E-step for deep mixtures can become intractable quickly, as the EM algorithm is a batched-optimiser. Thus, to prevent this one could adopt the SEM algorithm [3], use the SGD optimiser [9], or any other technique where the training is performed over mini-batches. However, none of the existing work on learning NMMMs have addressed this limitation.

Further, the authors also assume that the number of negative and positive patterns are known at initialisation, which is an infeasible assumption for high-dimensional data. In their experiments, they have used 2-dimensional data, where it is trivial to estimate the number of positive and negative patterns. Therefore, their work serves as a foundation for achieving non-monotonic mixtures, however to create a generalised learning scheme additional work remains.

2.3 Mixture Models as Probabilistic Circuits

A suitable learning framework to learn generalised NMMMs, has not been explored yet. Probabilistic Circuits (PCs) provide a promising framework for modelling deep mixtures and combinations of parametric families, offering a general framework for constructing hierarchical and recursive structures of probability densities. These structures posses certain properties that make large classes of queries tractable, facilitating exact inference on probability distributions and eliminating the need for estimations. This chapter aims to reformulate shallow mixtures as simple PCs and demonstrate why PCs can effectively handle both shallow and deep mixtures.

To support this claim, we present the relevant theory behind PCs as formulated by Antoni Vergari et al. [32] By doing so, we aim to show that PCs offer a general and effective framework for modelling complex mixtures. This discussion will also highlight the advantages of using PCs over other approaches. Ultimately, this section aims to contribute to the development of more effective learning frameworks for NMMMs that are applicable to a wide range of mixture models, including complex ones.

To reframe mixture models as PCs, it is necessary to first gain a high-level understanding of what a PC is. Essentially, a PC is a computational graph that can be defined by a set of random variables (RVs) $\mathbf{X}$ and a probabilistic circuit C. The circuit is represented as a directed acyclic graph (DAG) with an output distribution $P_C(\mathbf{X})$. The DAG in a PC consists of three types of nodes or units, which include distributional units, product units, and sum units. A computational unit in the circuit, denoted for some node n in the DAG, is represented as C_n and encodes a probability distribution P_{C_n}. The set of child nodes connected to node n is denoted as $in(n)$.

A shallow mixture model can be represented as a set of computational units that are fully connected to a sum unit within a PC. This representation captures the essence of a shallow mixture model, which consists of a single layer of mixture components. By reformulating shallow mixtures as simple PCs, it becomes possible to model more complex mixture models with deep nested mixtures or components of multiple parametric families. An example of a Gaussian Mixture Model (GMM) with two components, represented as a PC, can be seen in Figure 2.2.

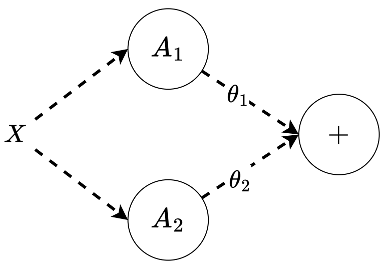

Figure 2.2: GMM as a PC for some RV **X** and a sum unit with weights at the edges.

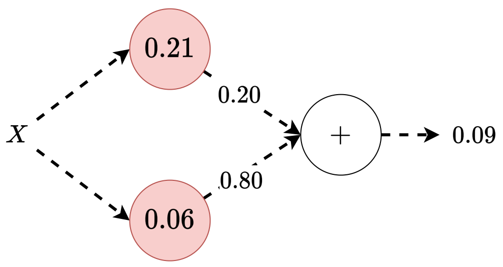

Figure 2.3: Inference on PCs. $A_1 : \mathcal{N}_1(\mu = -2, \sigma = 2)$ and $A_2 : \mathcal{N}_2(\mu = 2, \sigma = 1.5)$.

Efficient training and inference of deep mixture models, characterised by nested mixtures, has traditionally posed a challenge due to the requirement of specialised frameworks distinct from those employed in the case of shallow mixtures. Figures 2.4 and 2.5 visually demonstrate the contrast between shallow GMMs and deep GMMs.

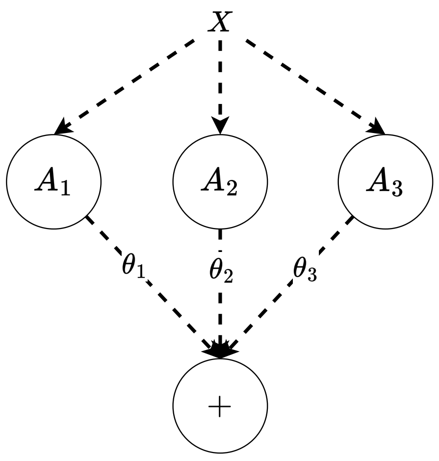

Figure 2.4: Shallow GMM

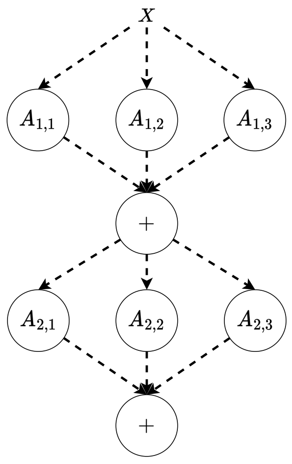

Figure 2.5: Deep GMM

Previous studies on deep mixture models have developed specific learning schemes and structural properties to ensure computationally efficient inference [31, 8, 33]. However, such techniques are often limited to a particular family of parametric distributions, such as GMMs. Hence, the use of Probabilistic Circuits (PCs) as a framework coupled with a generalisable optimisation scheme is proposed to ensure tractable deep mixtures, subject to the enforcement of relevant structural properties.

Section 2.4 formally defines the PC components, properties, and operations relevant to the methodology of this project. By redefining mixtures as PCs and adhering to the structural properties of PCs when constructing a learning scheme for shallow NMMMs, a foundation can be established that can be readily extended to deeper mixtures and support for multiple parametric families.

2.4 Probabilistic Circuits Theory

In this section we will present the necessary components, properties and operations for probabilistic circuits which are directly linked to the theoretical framework of this project.

2.4.1 Distributional Units

Distributional units form the most basic unit in a PC. Such units represent a probability distribution for some input RV. Which in this project will be assumed to be parametric. Thus, when given a full set of evidences, the inference query will output a probability for the given RV realisation. These units are flexible and different probability distributions can be exchanged with one another. Figure 2.6 demonstrates how such computational units are modelled, and Figure 2.7 shows an example of a possible inference query.

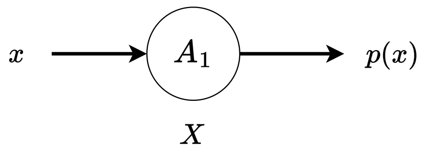

Figure 2.6: Distributional unit

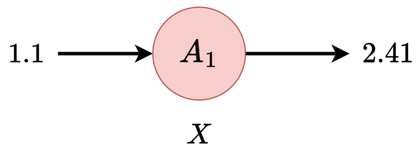

Figure 2.7: $A_1 = \mathcal{N}(X; \mu = 1, \sigma = 0.1)$

2.4.2 Product Units

Product units represent joint distributions which are factorised. Meaning that the outputs of each input node are multiplied together. Suppose we have a factorised Gaussian distribution.

$$
\begin{aligned}
p(X_1) &= p_1(X_1)p_2(X_1)p_3(X_1) \\
&= \mathcal{N}_1(X_1; \mu_1, \sigma_1)\mathcal{N}_2(X_1; \mu_2, \sigma_2)\mathcal{N}_3(X_1; \mu_3, \sigma_3)
\end{aligned} \tag{2.3}
$$

The join probability is simply the output of the Gaussians, multiplied. Therefore, in the world of PCs we can frame this as taking the product of these distributional units. To evaluate some inference query for P_{C_n} where n is the product unit, it is trivial to see that it is the same as evaluating Equation 2.3. Figure 2.8 demonstrates the computational process of product units for a realisation of all random variables belonging to the product node children.

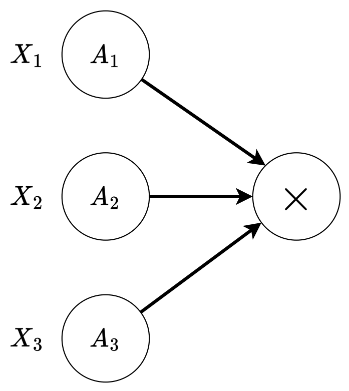

Figure 2.8: Computational unit

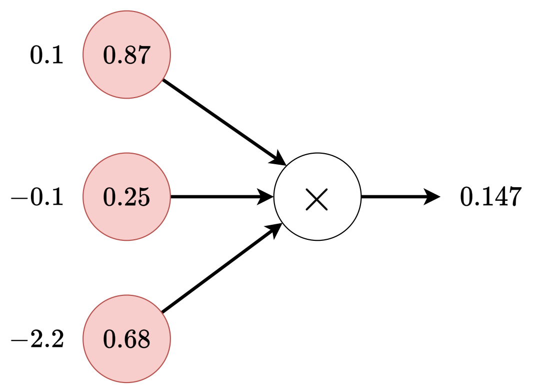

Figure 2.9: $A_1 = \mathcal{N}(X; \mu = 1, \sigma = 0.1)$

2.4.3 Sum Units as Mixture Models

Sum units represent a mixture of probability distributions. Thus, each edge to every child node of the sum node, has a respective weight attached to it. This weight represents the prior probability of a sample belonging to that child node. The formal definition of a sum node, follows the same definition of an FMM, which can be found in Section 2.1.

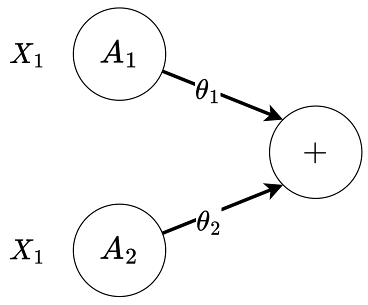

Figure 2.10: GMM as a PC for some RV **X** and a sum unit with weights at the edges.

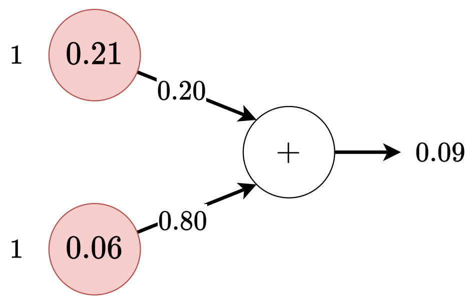

Figure 2.11: $\mathcal{N}_1(\mu = -2, \sigma = 2)$ and $\mathcal{N}_2(\mu = 2, \sigma = 1.5)$.

2.4.4 Structural Properties of PCs

To ensure that different inference queries are tractable for PCs, we have to ensure that the PC follow some properties. These properties depend on the general structure of the PC. These definitions will be based on the work done by Antonio Vergari, et al. [32].

Definition 2 *(Smoothness). A circuit is smooth if, for every sum unit n, its inputs depend on the same variables:* $\forall c_1, c_2 \in in(n), \phi(c_1) = \phi(c_2)$

Definition 3 *(Decomposability). A circuit is decomposable if the inputs of every product unit n depend on disjoint sets of variables:* $in(n) = \{c_1, c_2\}, \phi(c_1) \cap \phi(c_2) = \emptyset$.

Definition 4 *(Compatibility). Two circuits p and q over variables X are compatible if (1) they are smooth and decomposable and (2) any pair of product units $n \in p$ and $m \in q$ with the same scope can be rearranged into binary products that are mutually compatible and decompose in the same way:* $(\phi(n) = \phi(m)) \longrightarrow (\phi(n_i) = \phi(m_i)$, *$n_i$ and*

m_i are compatible) for some rearrangement of the inputs of n (resp. m) into n_1, n_2 (resp. m_1, m_2).

Definition 5 *(Structured-decomposable). A circuit is structured-decomposable if it is compatible with itself.*

2.4.5 Powers of Circuits

Additionally, it is possible to perform operations on probabilistic circuits, and they are essentially mathematical operators such as addition, summation, and powers. These operations are applied to the circuit as a whole and thus changes the output of the circuit. For this project, we are only going to concern ourselves with the process of taking powers of a circuit. Which will be done to construct a NMMM which is a valid PDF, the details of this process will be discussed in Chapter 3.

To be able to compute the power of a circuit, it has to be structured-decomposable as outlined in Definition 5. Otherwise, the process of computing the circuit power will be P-hard. Antonio Vergari, et al. [32] gives the following theorem which they prove.

Definition 6 *If p is a structured-decomposable circuit, then for any $\alpha \in N$, its power can be represented as a structured-decomposable circuit in $O(|p|^{\alpha})$ time. Otherwise, if p is only smooth and decomposable, then computing $p^{\alpha}(X)$ as a decomposable circuit is P-hard.*

2.5 Overview

In this chapter, the formal definition of mixture models were established. Further, Gaussian mixture models (GMMs) are commonly used, with the traditional expectation maximization (EM) algorithm for learning optimal configurations, however this can be inefficient for high dimensions and large amounts of training data. Stochastic gradient descent (SGD) has been proposed as an alternative using mini-batches, and the parameters of the GMM are reparameterised with Cholesky decomposition to ensure that the covariance matrices are valid.

However, the priors of the mixture also require some constraints in a monotonic Gaussian mixture, limiting their ability to subtract away probability densities. Non-monotonic mixtures have been proposed as an alternative in the past. However, their work is not generalisable for complex mixtures, therefore this project will now develop practical techniques for achieving this.

Chapter 3

Theoretical Framework

A new generalised learning framework for achieving NMMMs is proposed in the following chapter. This technique involves squaring the circuit representation of the mixture model. By re-framing mixture models through the lens of PCs, it can be ensured that complex mixtures remain tractable as long as the circuit follows the structural properties outlined in Section 2.4.4, and a suitable optimisation procedure is used, which will be outlined in 5. At the end of this chapter, the following contributions will be established.

- A generalised technique involving circuit squaring to remove the possibility of negative values being output from the mixture.
- Assuming that the circuit is squared, a technique for renormalising the squared circuit will pre presented such that it is a valid PDF.

3.1 Squaring the Circuit

As this project no longer assumes that the mixture weights are positive, we have to employ some techniques which will ensure that the resulting NMMM is a valid PDF. Which implies that the mixture model has to be bounded by the constraints in Definition 1.

Ensuring the first constraint is trivial, and can be done by squaring the mixture model. For any mixture of parametric probability distributions, the resulting squared PC will be the Cartesian product of the distributional nodes with respect to the sum node. This is trivial to see, assuming some mixture with N components and a parametric probability distribution denoted by $P(X;\theta)$, where θ represent the distribution parameters. The squared circuit is then the Cartesian product.

$$\left(\sum_{i=0}^{N} \alpha_i P(X;\theta_i)\right)^2 = \sum_{i=0}^{N}\sum_{j=0}^{N} \alpha_i \alpha_j P(X;\theta_i) P(X;\theta_j) \tag{3.1}$$

As a PC for some three component mixture, the DAG for the squared circuit can be seen in Figure 3.2.

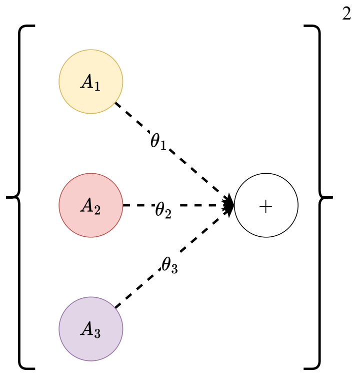

Figure 3.1: A mixture as a sum unit to be squared

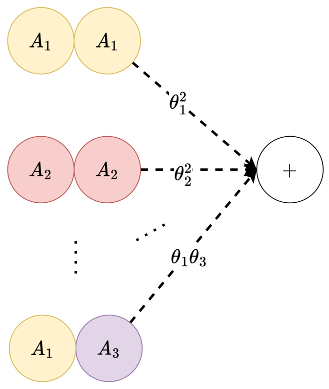

Figure 3.2: Result of squaring a mixture at the sum unit by using Cartesian product.

In Figure 3.2, the distributional nodes that are grouped together represent the product of two parametric distributions. These product components can be treated as new components of the mixture, however, it is not guaranteed that the products between parametric distributions are normalised, which in turn may result in the mixture not being normalised. This would violate the definition of a valid PDF. Therefore, it is essential to compute a normalisation constant for each product component in the squared circuit.

The normalisation scheme depends on the parametric family assumed, however the approach is still applicable as long as a normalisation constant for the product of two densities can be computed effectively. This will be possible for most exponential families, such as Gaussians. As a generalised analysis for computing the normalisation constant, assume two continuous parametric distributions $P(\mathbf{X};\theta_a)$ and $P(\mathbf{X};\theta_b)$ where θ denotes the respective parameters of the distributions. The normalisation constant for the product of the two distributions can be computed by resolving the following integral.

$$\int_{\mathbb{R}^D} P(\mathbf{X};\theta_a)P(\mathbf{X};\theta_b)\, d\mathbf{X} = Z \tag{3.2}$$

The product between the two distribution can therefore be normalised to be a valid PDF.

$$Z^{-1}P(\mathbf{X};\theta_a)P(\mathbf{X};\theta_b) \tag{3.3}$$

Which resolves how it is possible to renormalise the product between parametric distributions, however as will be discussed in Subsection 3.2.2 more work remains to normalise the squared circuit as a whole.

3.2 Assuming Gaussian Distributions

As the computation of the normalisation constant depends on the parametric family assumed, we will assume that the NMMM consists of multivariate Gaussians. Additionally, a shallow mixture is assumed, as this simplifies the theoretical work, however the same principles will apply for deep mixture models. It has been established that the product of parametric distributions is another possibly un-normalised parametric distribution. To create a learning scheme for the approach outlined in Section 3.1, this section will outline how the product of multivariate Gaussians is another possibly un-normalised multivariate Gaussian. Which will be followed by how the normalisation constant can be computed such that it can be propagated to the sum unit in the squared PC mixture to normalise the circuit as a whole.

3.2.1 Product of Gaussians and the Normalisation Constant

The formulation for the product of two multivariate Gaussians and their normalisation constant will be based upon the derivations made by Rasmussen et al. [27]. In their section on Gaussian identities, they present the following formulation for the product of two multivariate Gaussian distributions.

$$\mathcal{N}(\mathbf{X};\Sigma_a,\mu_a)\mathcal{N}(\mathbf{X};\Sigma_b,\mu_b) = Z^{-1}\mathcal{N}(\mathbf{X};\Sigma_c,\mu_c) \tag{3.4}$$

Which shows that the product of two multivariate Gaussians is another multivariate Gaussian. The new parameters of this multivariate Gaussian is expressed in terms of the parameters of the two other Gaussians.

$$\Sigma_c = (\Sigma_a^{-1} + \Sigma_b^{-1})^{-1} \text{ and } \mu_c = \Sigma_c(\Sigma_a^{-1}\mu_a + \Sigma_b^{-1}\mu_b) \tag{3.5}$$

Where Z^{-1} was obtained by performing the following integration

$$\begin{aligned}\int_{\mathbb{R}^D} \mathcal{N}(\mathbf{X};\Sigma_a,\mu_a)\mathcal{N}(\mathbf{X};\Sigma_b,\mu_b)\, d\mathbf{X} &= \int_{\mathbb{R}^D} Z^{-1}\mathcal{N}(\mathbf{X};\Sigma_c,\mu_c)\, d\mathbf{X} && (3.6)\\ &= Z^{-1} && (3.7)\end{aligned}$$

Therefore, Z^{-1} is the normalisation constant for the product of the two Gaussians as seen above. The expression for this constant is given in Equation 3.8.

$$Z^{-1} = \frac{1}{\sqrt{(2\pi)^D|\Sigma_a+\Sigma_b|}}\exp(-\frac{1}{2}(\mu_a-\mu_b)^\top(\Sigma_a+\Sigma_b)^{-1}(\mu_a-\mu_b)) \tag{3.8}$$

Given these definitions and more importantly the normalisation constant Z^{-1}, it is possible to build a non-monotonic mixture which is a valid PDF, by squaring the PC. However, it is important to understand how to apply this normalisation constant correctly such that the mixture as a whole is a valid PDF.

3.2.2 Normalising the Circuit

The goal is not to normalise the product components as we saw in Figure 3.1, the desired outcome is a squared mixture which is normalised. Therefore, this subsection will explore why applying the normalisation constant to the product between Gaussians is not appropriate. Which will be followed by an analysis on how it is possible to propagate the constant to the sum node to normalise the entire circuit.

Firstly, let us analyse why normalising the product components directly might not normalise the entire circuit when we cannot ensure that the weights of the sum node, sum to 1. Here $\mathcal{N}_i$ will represent some Gaussian defined over RV $\mathbf{X}$, and α_i is the weight or prior belonging to that Gaussian. Additionally, we denote Z_{ij}^{-1} to be the normalisation constant produced between component i and component j. Thus, for some squared mixture consisting of these Gaussian, its integral can be expressed as.

$$\int_{\mathbb{R}^D} \sum_{i=0}^{N} \sum_{j=0}^{N} Z_{ij}^{-1} \mathcal{N}_i \mathcal{N}_j \alpha_i \alpha_j \, d\mathbf{X} = \sum_{i=0}^{N} \sum_{j=0}^{N} \int_{\mathbb{R}^D} Z_{ij}^{-1} \mathcal{N}_i \mathcal{N}_j \alpha_i \alpha_j \, d\mathbf{X} \tag{3.9}$$

$$= \sum_{i=0}^{N} \sum_{j=0}^{N} \alpha_i \alpha_j \int_{\mathbb{R}^D} Z_{ij}^{-1} \mathcal{N}_i \mathcal{N}_j \, d\mathbf{X} \tag{3.10}$$

Inserting Equation 3.6 into the above will resolve to the following.

$$= \sum_{i=0}^{N} \sum_{j=0}^{N} \alpha_i \alpha_j \tag{3.11}$$

However, this is a problem, as we cannot assume any more that the priors will sum to one for the non-monotonic Gaussian mixture model (NMGMM). Which is the reason for propagating the normalisation constant to the sum node before normalising the entire circuit. By propagating the normalisation constant to the sum node, the weighted sum of the constants is taken and sent to the sum node. Which then computes the weighted sum for the un-normalised mixture and divides this with the weighted sum of the normalisation constant. The derivation for this process can be seen below.

$$= \int_{\mathbb{R}^D} \left(\sum_{i=0}^{N} \sum_{j=0}^{N} Z_{ij}^{-1} \alpha_i \alpha_j \right)^{-1} \left(\sum_{i=0}^{N} \sum_{j=0}^{N} \mathcal{N}_i \mathcal{N}_j \alpha_i \alpha_j \right) d\mathbf{X} \tag{3.12}$$

We can further simplify by moving the integral to be around the unnormalised mixture term, as the normalisation constants are constant with respect to $\mathbf{X}$.

$$= \left(\sum_{i=0}^{N} \sum_{j=0}^{N} Z_{ij}^{-1} \alpha_i \alpha_j \right)^{-1} \int_{\mathbb{R}^D} \sum_{i=0}^{N} \sum_{j=0}^{N} \mathcal{N}_i \mathcal{N}_j \alpha_i \alpha_j \, d\mathbf{X} \tag{3.13}$$

$$= \left(\sum_{i=0}^{N} \sum_{j=0}^{N} Z_{ij}^{-1} \alpha_i \alpha_j \right)^{-1} \sum_{i=0}^{N} \sum_{j=0}^{N} \alpha_i \alpha_j \int_{\mathbb{R}^D} \mathcal{N}_i \mathcal{N}_j \, d\mathbf{X} \tag{3.14}$$

Inserting Equation 3.6 into the above will resolve to the following.

$$= \left(\sum_{i=0}^{N}\sum_{j=0}^{N} Z_{ij}^{-1}\alpha_i\alpha_j\right)^{-1}\left(\sum_{i=0}^{N}\sum_{j=0}^{N} Z_{ij}^{-1}\alpha_i\alpha_j\right) \tag{3.15}$$

$$= 1 \tag{3.16}$$

This shows that the normalisation scheme can simply propagate the normalisation constants down to the sum node, and that for any combination of weights that might not sum to one, the NMGMM will integrate to 1. Additionally, since the circuit is squared, it is ensured that the output of the mixture will never be negative. In other words, we have constructed the necessary components for an NMGMM which follows the conditions of a PDF as per Definition 1, thus ensuring that the NMGMM is a valid PDF.

However, as the NMGMM is a squared mixture, it effectively increases the number of components. Where the parameters of each new component is computed as a combination of the parameters from the original un-squared PC, which is given by Equation 3.5. Therefore, before we can construct an effective learning framework for this model, we first have to understand how the parameters of these new product components are transformed such that we can account for these transformations in our learning scheme. This analysis will be given in the following chapter.

Chapter 4

Preliminary Analysis

To effectively train any mixture model, the initialisation procedure is a crucial step. However, since the NMGMM model effectively increases the number of components by squaring the circuit, the new product components will not have the same parameters as a monotonic GMM with an equivalent initialisation. Which will influence how good or bad the initialisation is for the squared mixture. In this chapter, we will assume that we have some Gaussian mixture A with possibly negative weights which is squared and normalised according to the procedure outlined in Chapter 3 to produce a NMGMM B. From that, we will analyse how different initialisations for the parameters of mixture A translates for mixture B. Therefore, this chapter will establish the following contributions, which will help construct the learning framework and effective experiments in the following chapters.

- Preliminary experiments investigating the transformation of the means for NMGMM B with respect to initial configurations of GMM A.
- Preliminary experiments demonstrating how the generalised covariance is transformed for NMGMM B with respect to initial configurations of GMM A.
- Thorough algebraic analysis on the transformation of the means and the covariances for NMGMM B, which will justify the results of the preliminary experiments above.

4.1 Transformation of Means

Upon visual inspection of the new parameters of the squared circuit in Equation 3.5 the correlation between the initialised parameters and the parameters of the squared mixture is not obvious. Therefore, this preliminary experiment will investigate four different initialisations for GMM A and observe how the means are transformed for NMGMM B, using the same initialisation.

Experiment	Covariance type	Mean range	Covariances range
Alpha - 1	Diagonal	$\mathcal{N}(\mu = 0, \sigma = 0.5)$	$\mathcal{N}(\mu = 2, \sigma = 0.5)$
Alpha - 2	Diagonal	$\mathcal{N}(\mu = 0, \sigma = 2)$	$\mathcal{N}(\mu = 2, \sigma = 0.5)$
Alpha - 3	Full	$\mathcal{N}(\mu = 0, \sigma = 0.5)$	$\mathcal{N}(\mu = 2, \sigma = 0.5)$
Alpha - 4	Full	$\mathcal{N}(\mu = 0, \sigma = 2)$	$\mathcal{N}(\mu = 2, \sigma = 0.5)$

Table 4.1: Configurations for the preliminary experiments on the transformation of means for NMGMM B over different initialisations for the parameters in GMM A.

4.1.1 Experiment Setup

To examine the impact of mean initialisation, we will conduct four experiments with both full and diagonal covariances. Additionally, we will compare two initial mean distributions for component A, one with tightly initialised means using a normal distribution having a low standard deviation and the other with a large standard deviation. The objective is to gain a better understanding of how means for NMGMM B can deviate from the initial distribution in mixture A. The parameters used for this investigation are detailed in Table 4.1.

4.1.2 Results

The preliminary experiments suggest that the distribution of means for the squared circuit significantly relies on the shape of the covariance matrices. The experiments Alpha-1 and Alpha-2 in Figure 4.1 demonstrate that the means of mixture B follow a similar distribution as mixture A, as evidenced by the similarity in the convex hull shapes of both distributions. Nevertheless, the convex hull for mixture B appears to be slightly scaled compared to the original distribution, indicating that the initialisation of mixture A might result in some components of mixture B falling beyond the desired area of initialisation.

Remarkably, a significant change in the distribution of the means for mixture B occurs when the covariance matrices are full. In reality, the distribution of means for mixture B appears to be influenced by the shape of the covariance matrices, which aligns with Equation 3.5. Therefore, experiment Alpha-3 and Alpha-4 reveal that increasing the range of the mean initialisation leads to the expansion of the gap between the two convex hulls.

These experiments demonstrate the significance of creating an effective initialisation scheme for the NMGMM, as conventional initialisation procedures for GMMs cannot ensure similar initialisation for the NMGMM. This is especially critical when using full covariance matrices, where we must be cautious when initialising, as the means can quickly go beyond the initialisation space. However, before creating better initialisations for the NMGMM, we must first analyse how the covariances are transformed.

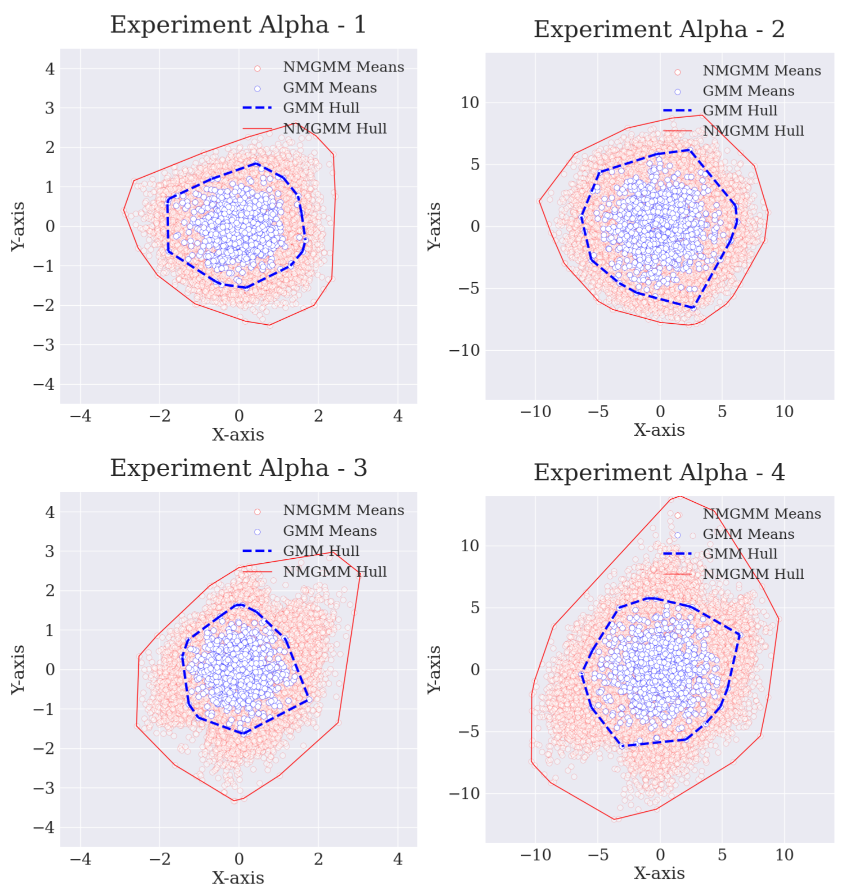

Figure 4.1: The transformation of means for NMGMM B marked in red.

4.2 Transformation of Covariances

It is clear that any initialisation procedure for the NMGMM needs a regularisation of the means, however more work remains to understand how the covariances are initialised. As the covariance matrices for the NMGMM are computed using Equation 3.5 it is not immediately obvious how the covariance matrices for NMGMM B will respond to some initialisation of the covariance matrices for mixture A.

4.2.1 Experiment Setup

Therefore, two preliminary experiments will be setup according to Table 4.2, which will investigate the effect of diagonal vs full covariance matrices by looking at the generalised covariances. The results of these experiments can be seen in Figure 4.2.

Experiment	Covariance type	Mean range	Covariances range
Bravo - 1	Diagonal	$\mathcal{N}(\mu = 0, \sigma = 2)$	$\mathcal{N}(\mu = 2, \sigma = 0.5)$
Bravo - 2	Full	$\mathcal{N}(\mu = 0, \sigma = 2)$	$\mathcal{N}(\mu = 2, \sigma = 0.5)$

Table 4.2: Configurations for the preliminary experiments on covariance transformations.

4.2.2 Results

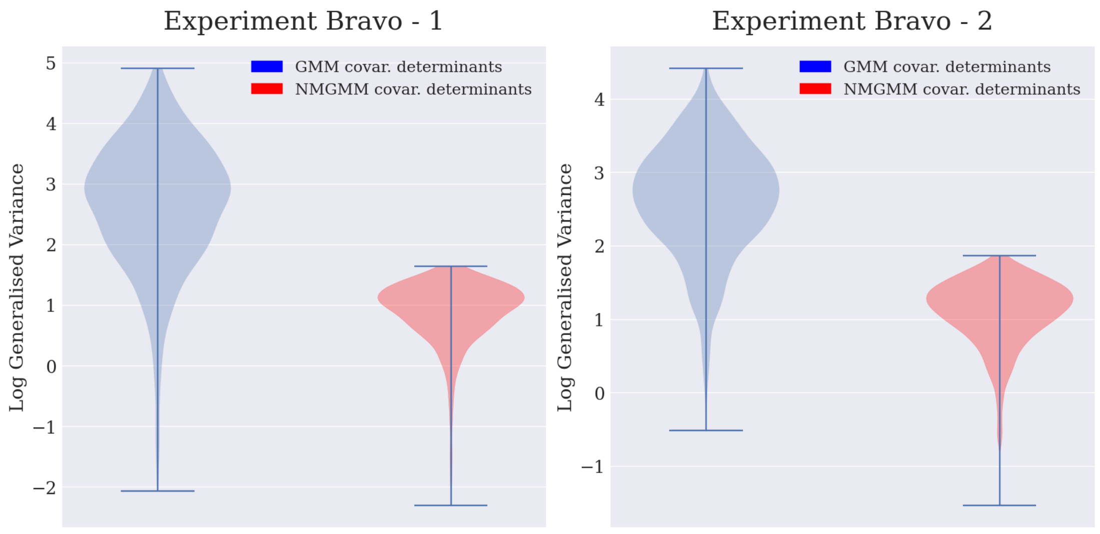

Figure 4.2: Log generalised covariance for a GMM and a NMGMM, log is performed to be able to compare the generalised covariance for both models.

Figure 4.2 illustrates that the covariance matrices are considerably scaled down in comparison to the original covariance matrices. Moreover, there seems to be no substantial variation between using diagonal or full covariance matrices. This implies that the initialisation of the covariance matrices should make use of large values to avoid scaling down the covariances. Additionally, this suggests that the optimiser will have to expand the covariance matrix considerably more for the NMGMM than for the GMM. To better understand why these transformations occur for the means and the covariances, we will provide an algebraic analysis on the effects observed in the above experiments.

4.3 Algebraic Analysis

When performing the preliminary experiments, some interesting patters were discovered with respect to the placement of the means and the transformation of covariances after the mixture is squared. Therefore, this section will explore these patterns for diagonal and full covariance matrices. Which will result in a general expression for the placement of means and the shape of the new covariance matrix, which will be used to highlight some interesting properties. It is worth noting that this section does not intend to provide any rigorous proofs of these patterns, but rather serve as an explanation for the observations in Figure 4.1 and Figure 4.2.

4.3.1 Diagonal Covariance Matrices

In this subsection, we will explore what happens when we multiply two components with diagonal covariance matrices. These components are assumed to be a part of a mixture we are squaring to produce the NMGMM as outlined in Chapter 3. This analysis will form a deeper understanding of the initialisation of means and the transformation of covariance matrices for the NMGMM. Which will be necessary for creating an effective learning framework and performing experiments. Throughout this subsection we will be assuming that we have two mixture components A and B with diagonal covariance matrices which will contribute to produce some new component C in the NMGMM where C is defined by $C = A \times B$.

Given the above context for component A and component B, we define the parameters of these components according to the parameters in Equation 4.1. Where the covariance matrix for component B is defined by scaling the covariance matrix for component A using a scaling vector δ, which will help provide some insight into the relationship between the two components.

$$\Sigma_A = \begin{bmatrix} a_{11} & 0 \\ 0 & a_{22} \end{bmatrix} \quad \mu_A = \begin{bmatrix} x_A \\ y_A \end{bmatrix} \qquad \Sigma_B = \Sigma_A \begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix} \quad \mu_B = \begin{bmatrix} x_B \\ y_B \end{bmatrix} \tag{4.1}$$

Using these parameters and Equation 3.5 for the parameters of the product between Gaussians, we derive the following expression for the means and the covariance matrix for the new component. The details of this derivation can be found in the Appendix A.1.

$$\Sigma_C = \begin{bmatrix} a_{11}(1+\delta_1)^{-1} & 0 \\ 0 & a_{22}(1+\delta_2)^{-1} \end{bmatrix} \quad \mu_C = \begin{bmatrix} (x_A + \delta_1 x_B)(1+\delta_1)^{-1} \\ (y_A + \delta_2 y_B)(1+\delta_2)^{-1} \end{bmatrix} \tag{4.2}$$

This expression of the placement of the means gives us some interesting properties, where each property will be highlighted below for different configurations of the scaling vector δ.

4.3.1.1 Similar Covariance Matrix

If component A and B have the same variance in their respective covariance matrices, then the scaling vector δ becomes a unit vector. Thus, reducing the expression for the mean and covariance matrix of component C to become.

$$\Sigma_C = \begin{bmatrix} a_{11}2^{-1} & 0 \\ 0 & a_{22}2^{-1} \end{bmatrix} \quad \mu_C = \begin{bmatrix} (x_A + \delta_1 x_B)2^{-1} \\ (y_A + \delta_2 y_B)2^{-1} \end{bmatrix} \tag{4.3}$$

Which will place the new component exactly between the component A and component B. Additionally, the covariance matrix of component C will be scaled down with half the variance of component A and B, which is consistent with the results in Figure 4.2.

4.3.1.2 Scaled Covariance Matrix

Further, if component B becomes infinitely larger compared to component A then we observe that the mean of the component C is bounded by the mean of component B.

$$\lim_{\delta_1,\delta_2\to\infty}\begin{bmatrix}(x_A+\delta_1 x_B)(1+\delta_1)^{-1}\\(y_A+\delta_2 y_B)(1+\delta_2)^{-1}\end{bmatrix}=\begin{bmatrix}x_B\\y_B\end{bmatrix} \tag{4.4}$$

Therefore, if component B becomes sufficiently larger than component A then component C will gravitate towards component B. Further, it is trivial to see that the inverse will happen when δ approaches zero. For the covariance matrix of the new component, we see observe in Figure 4.2 that the covariances for component C is scaled down considerably compared to component A and B. In Equation 4.5, we see what happens when the covariance of a component B is scaled up with respect to component A.

$$\lim_{\delta_1,\delta_2\to\infty}\begin{bmatrix}a_{11}(1+\delta_1)^{-1} & 0\\0 & a_{22}(1+\delta_2)^{-1}\end{bmatrix}=\begin{bmatrix}0&0\\0&0\end{bmatrix} \tag{4.5}$$

This indicates that there is an inverse relationship between the scale of component B and the scale of the resulting component C. Which is again consistent with our findings in Figure 4.2.

4.3.1.3 Dissimilar Covariance Matrix

In Figure 4.1 we observe that the means of the new components in the squared circuit are transformed beyond the original distribution of the means. This, happens when some δ_i is scaled down sufficiently, and some other δ_j is scaled up, which is expressed by the following.

$$\lim_{\delta_1\to 0}\lim_{\delta_2\to\infty}\begin{bmatrix}(x_A+\delta_1 x_B)(1+\delta_1)^{-1}\\(y_A+\delta_2 y_B)(1+\delta_2)^{-1}\end{bmatrix}=\begin{bmatrix}x_A\\y_B\end{bmatrix} \tag{4.6}$$

Which shows that the means of the newly created component can indeed go beyond the original distribution of the means. Further, as the scaling factor δ is bounded by $\delta\in(0,\infty)$, this will limit how large the convex hull of the components of the new means are, which is consistent with our observation in Figure 4.1. As for the covariance matrix, we again observe an inverse relationship, where component C will be scaled along the axis which we reduce in component B.

$$\lim_{\delta_1\to 0,\delta_2\to\infty}\begin{bmatrix}a_{11}(1+\delta_1)^{-1} & 0\\0 & a_{22}(1+\delta_2)^{-1}\end{bmatrix}=\begin{bmatrix}a_{11}&0\\0&0\end{bmatrix} \tag{4.7}$$

This demonstrates that for mixtures which are initialised to be diagonal, it is possible to estimate the location of the means as they are bounded. Additionally, we observe that the covariance matrix for the new component does not have the same bounds which the means have. As our experiments will be conducted on data which also requires full covariance matrices, we will extend this analysis for full covariance matrices to investigate whether this relationship is also consistent for full covariances.

4.3.2 Full Covariance Matrices

For full covariance matrices, the derivation for the means and the covariance matrix of the new component is not as straight forward. Therefore, in this analysis, we will simply assume that we have some mixture component A and another component B which is scaled with respect to A by some constant δ. The product component will be expressed as C and will be one of the new components defined in some NMGMM. This investigation will give light to whether or not the means of component C are bounded, as observed for diagonal covariance matrices. Using this context for component A and component B we define their parameters by the following.

$$\Sigma_A = \begin{bmatrix} a_{11} & 0 \\ 0 & a_{22} \end{bmatrix} \quad \mu_A = \begin{bmatrix} x_A \\ y_A \end{bmatrix} \qquad \Sigma_B = \delta\, \Sigma_A \quad \mu_B = \begin{bmatrix} x_B \\ y_B \end{bmatrix} \tag{4.8}$$

Using these parameters and Equation 3.5 for the parameters of the product between Gaussians, we derive the following terms for the expression of the means and the covariance matrix for component C. The details of this derivation can be found in the Appendix A.2.

$$\Sigma_C = \begin{bmatrix} \delta^2 a_{11}(\delta+1)^{-1} & \delta^2 a_{12}(\delta+1)^{-1} \\ \delta^2 a_{21}(\delta+1)^{-1} & \delta^2 a_{22}(\delta+1)^{-1} \end{bmatrix} \tag{4.9}$$

$$\Sigma_A^{-1} \times \mu_A + \Sigma_B^{-1} \times \mu_B = \begin{bmatrix} (a_{22}(\delta x_A + x_B) - a_{12}(\delta y_A + y_B))(\delta|\Sigma_A|)^{-1} \\ (-a_{21}(\delta x_A + x_B) + a_{11}(\delta y_A + y_B))(\delta|\Sigma_A|)^{-1} \end{bmatrix} \tag{4.10}$$

Using the expression for Σ_C and $\Sigma_A^{-1} \times \mu_A + \Sigma_B^{-1} \times \mu_B$ we will provide insight into the transformation of means for component C when component B is scaled with respect to component A.

4.3.2.1 Scaled Up Covariance Matrix

In this analysis, we will explore what happens when we scale up component B such that it is infinitely larger than component A. Which will indicate whether the means of component C are bounded as for the diagonal covariances.

$$\lim_{\delta\to\infty} [\Sigma_C(\Sigma_A^{-1} \times \mu_A + \Sigma_B^{-1} \times \mu_B)] = \lim_{\delta\to\infty} \Sigma_C \lim_{\delta\to\infty} [\Sigma_A^{-1} \times \mu_A + \Sigma_B^{-1} \times \mu_B] \tag{4.11}$$

By first preforming the limit over Σ_C we observe the following

$$\lim_{\delta\to\infty} \Sigma_C = \begin{bmatrix} \infty & \infty \\ \infty & \infty \end{bmatrix} \tag{4.12}$$

$$\lim_{\delta\to\infty} [\Sigma_A^{-1} \times \mu_A + \Sigma_B^{-1} \times \mu_B] = \begin{bmatrix} (a_{22}x_A - a_{12}y_A)(|\Sigma_A|)^{-1} \\ (-a_{21}x_A - a_{11}y_A)(|\Sigma_A|)^{-1} \end{bmatrix} \tag{4.13}$$

Which shows that, when component B is sufficiently larger than component A, then the means for component C will move without the bounds we observe for diagonal covariance matrices, which is consistent with Figure 4.1. Additionally, it seems that the means for component C will be distributed dependably on the variance and correlation of component A, which is also consistent with our observations in Figure 4.1.

4.3.2.2 Scaled Down Covariance Matrix

When we scaled down component B with respect to component A sufficiently, we observe that the placement of means for component C will indeed go towards the origin. We first compute the limit over, Σ_C as previously.

$$\lim_{\delta \to 0} \Sigma_C = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{4.14}$$

$$\lim_{\delta \to 0} [\Sigma_A^{-1} \times \mu_A + \Sigma_B^{-1} \times \mu_B] = \begin{bmatrix} \infty \\ \infty \end{bmatrix} \tag{4.15}$$

Therefore, there are no bounds for the placement of the means for component C when we have two full covariance matrices for component A and component B. Further, we notice that there is a mutual relationship between the scale of component B and the scale of component C.

4.4 Discussion

Throughout this chapter, we have seen that the transformation from a traditional GMM to a squared NMGMM can result in additional components which can have significantly smaller covariances than what was initialised. Additionally, the means can be spread out beyond the original distribution of means which we have initialised for. For diagonal covariance matrices, it is easier to account for this, as both the covariance matrix and the means for the new component in the NMGMM are bounded by the components which we use to construct it.

However, for full covariance matrices we have to be more careful as both the means and the covariances can explode or collapse depending on the relationship between the two components we use to construct the new product component in the NMGMM. This should not be a significant problem for our experiments, however, this will be discussed in Chapter 6. Before, we can start defining effective experiments, however, we will first establish the complete learning framework for NMGMM using the findings presented in this chapter as well as the theoretical framework in Chapter 3.

Chapter 5

Learning Framework

In Chapter 3 the method for constructing a NMMM as a valid PDF was established, additionally we discovered some interesting patterns about the transformation of components in Chapter 4. However, some work still remains to create a complete learning framework for the NMGMM. Specifically, this chapter will introduce how we are using the SGD momentum optimiser for learning NMGMMMs, similarly to the background material presented in Subsection 2.1.2.1. Additionally, a new hyperparameter will be introduced called the sparsity prior which will aid the optimisation procedure as there are many components which are dependent on each other as presented in Chapter 4. Therefore, at the end of this chapter the following outcomes will be established.

- An overview of how SGD is used to learn NMGMM, as well as the necessary reparametrisations which will be done to learn valid parameters for the NMGMM.
- The sparsity prior for weight regularisation will be introduced, which will help the NMGMM achieve better fits as the weights for each new component are better balanced.
- Finally, a complete learning framework for the NMGMM will be presented as an algorithm which will combine the theoretical framework with the newly established optimiser and weight regularisation.

5.1 Learning NMGMMs with SGD Momentum

From the background research, methods for training GMMs using numerical optimisers such as SGD were explored. To enable such optimisation algorithms, it is essential that some restrictions on the covariance matrix Σ are enforced. Specifically, to compute the PDF of a multivariate Gaussian, it is assumed that Σ is positive and semi-definite. Since optimisers such as SGD has some stochasticity it is not guaranteed that Σ is positive and semi-definite when it is optimised directly. To enforce this, we use Cholesky decomposition according to the past research. For each component in the mixture, we first perform the Cholesky decomposition such that we can ensure that all the covariance matrices in the mixture are valid. Which will be followed by the process of squaring the circuit as outlined in Chapter 3.

To further speed up the learning process, we opted for using SGD with momentum. SGD with momentum has the same base components as SGD which was presented in Chapter 2, however it adds a parameter called the momentum. This variant of SGD has also been applied to numerous optimisation problems, such as neural networks [18, 7]. The SGD with momentum optimiser simply adds the momentum parameter to the computation of the gradient, as described by Ian Goodfellow et al. [10].

$$\hat{g}_t := \beta \hat{g}_{t-1} - \varepsilon \nabla_\theta \left[\frac{1}{m} \sum_{i=0}^{m} L(f(x_i;\theta), y_i) \right] \tag{5.1}$$

Where $\hat{g}_t$ represents the estimated gradient given some loss function at time step t, further the parameter β represents the momentum. The computation for the new parameters are the same as for SGD, however, since the learning rate is already applied in the gradient computation we get.

$$\theta := \theta + \hat{g}_t \tag{5.2}$$

By using SGD with momentum we speed up the learning process, as the gradient of the loss function is only estimated it can have a lot of noise. Therefore, by applying momentum, we effectively smooth out such noise and thus avoid local minima more effectively.

5.2 Weight Regularisation

Since the NMGMM does not enforce any constraints on the weights, we add a custom regularisation term to our model. In the past, there has been some research on regularising mixtures [22, 4]. The rationale behind using regularisation choice, is that we want SGD to avoid local minima where the components capture a small subset of some shape and then explode the weights to become infinitely large. This can indeed happen without regularisation, as we no longer have any bounds on the weights, which a monotonic GMM does.

L1 and L2 regularisation are techniques which are popular in deep neural networks, to prevent exploding gradients and overfitting [38, 16, 24]. Similarly, in the case of the NMGMM, we can reframe the problem above as a problem of overfitting. However, as L1 and L2 regularisation will squeeze the weights close to zero, we use a modified version of these regularisation techniques, which can be seen below.

$$\mathcal{L}(B) = NLL(P(B;\theta)) + \lambda \left(\frac{1}{k} \sum_{i=0}^{k} \sqrt{\frac{e^{w_i}}{\sum_{j=0}^{k} e^{w_j}}} \right) \tag{5.3}$$

Here λ represents an additional hyperparameter called the sparsity prior, additionally we perform the softmax over the weights such that the square root can be performed as the weights can be negative. The reason behind this is to ensure that the weights do not get scaled down significantly towards zero, which is the case for L1 and L2 regularisation. Further, we use the mean over the square root softmax values such that we ensure a soft penalisation of weights which are too large. By introducing an additional hyperparameter we further complicate the hyperparameter optimisation, however this is balanced out by the importance of creating generalisable fits from the NMGMM.

5.3 Learning Scheme

Given all the components of the learning scheme above, this section will tie these concepts together to a complete learning scheme. As outlined in Subsection 5.1, the training procedure will be using SGD with momentum to learn optimal configurations for the NMGMM. Which requires some loss function $\mathcal{L}$ to be minimised, for the NMGMM implementation this will be the mean negative log-likelihood over a batch, additionally, we add the sparsity prior to this loss function during training. The loss function is given in Equation 5.4, where B_m denotes a mini-batch of points during training, and $p(B_i;\theta)$ represents the PDF of the NMGMM over some mini-batch instance i.

$$\mathcal{L}(B_i) = \frac{-log_e\left(P(B_i;\theta)\right)}{|B_i|} + \lambda\left(\frac{1}{k}\sum_{i=0}^{k}\sqrt{\frac{e^{w_i}}{\sum_{j=0}^{k} e^{w_j}}}\right) \tag{5.4}$$

Given the loss function above and the SGD with momentum optimiser, the learning procedure will look like the following.

1. Compute the new Σ covariance matrices for each component using the Cholesky decomposition method outlined in Subsection 5.1.
2. Perform the Cartesian product over the mixture components.
3. Over each pair of components i and j from the Cartesian product, compute the normalisation constant and the product between the components.
4. Perform the weighted sum using the non-monotonic weights for the normalisation constants and the products of components.
5. Normalise the circuit by dividing the weighted sum of components with the weighted sum of normalisation constants, which produces the PDF for the NMMM.
6. Compute $\mathcal{L}$ and perform an optimisation step to update the parameters.
7. Repeat the above steps until the maximum number of iterations is reached.

This concludes the work on constructing a learning scheme for NMMMs as valid PDFs. Additionally, a more thorough overview of the learning scheme is given in Algorithm 1.

5.3.1 Learning Algorithm

Algorithm 1: NMMM Learning Scheme

```
Input: Dataset batch $\mathcal{D}$, validation batch $\mathcal{V}$; Number of dimensions $N$, number of
       components $C$, sparsity prior $\lambda$
$\mu, L, w \leftarrow InitParams(C, N, \mathcal{D})$;
$cart\_ids \leftarrow CartesianProduct([0, \ldots, C-1])$;
while not converged do
    $Z_t, G_t \leftarrow 0, 0$;
    $Z_v, G_v \leftarrow 0, 0$;
    for $i, j \in cart\_ids$ do
        $\Sigma_i, \Sigma_j \leftarrow CholComposition(tril(L_i)), CholComposition(tril(L_j))$;
        $Z_t \leftarrow Z_t + w_i w_j ComputeNormalisation(\mathcal{D}, (\Sigma_i, \mu_i), (\Sigma_j, \mu_j))$;
        $G_t \leftarrow G_t + w_i w_j GaussianPDF(\mathcal{D}, \Sigma_i, \mu_i) GaussianPDF(\mathcal{D}, \Sigma_j, \mu_j)$;
        With zero grad do
            $Z_v \leftarrow Z_v + w_i w_j ComputeNormalisation(\mathcal{V}, (\Sigma_i, \mu_i), (\Sigma_j, \mu_j))$;
            $G_v \leftarrow G_v + w_i w_j GaussianPDF(\mathcal{V}, \Sigma_i, \mu_i) GaussianPDF(\mathcal{V}, \Sigma_j, \mu_j)$;
        end
    end
    $\mathcal{L}_t, \mathcal{L}_v \leftarrow \mathcal{L}(G_t, Z_t, w, \lambda), \mathcal{L}(G_v, Z_v, w, \lambda)$;
    Optimise for $L, \mu, w$ using $\mathcal{L}_t$;
end
```

The functions used in Algorithm 1 can be described by the following.

- *initParams:* method which optimally initialises the parameters, uses predefined optimal positions for the means, covariances and weights given a shape to fit. Can also initialise using methods such as K-means, however that is not an optimal initialisation procedure for the NMGMM as it will not effectively place the negative components.
- *CartesianProduct:* computes a list of integer pairs, representing the IDs of each component in the mixture. Therefore, we iterate over each pair as the product is performed between each paired ID in the Cartesian product.
- *CholComposition:* uses Cholesky decomposition as outlined in Section 5.1 given a lower triangular version of the matrix L which the optimiser has computed.
- *ComputeNormalisation:* computes the normalisation constant for the product between two Gaussian components as outlined in Subsection 3.2.2.
- *GaussianPDF:* computes the PDF of the multivariate Gaussian component as outlined in Subsection 2.1.2.
- *$\mathcal{L}$:* computes the negative log likelihood using the weighted sum of Gaussian PDFs and normalisation constants, as well as the weights and sparsity prior to compute the regularisation term as outlined in Section 5.2.

Chapter 6

Experimental Setup

As an effective learning framework for non-monotonic Gaussian mixtures was established in the previous chapter, the next stage is to lay a foundation for how the experiments for this model will be setup. This chapter will therefore contribute with the following.

- Introduce a collection of artificial shapes with negative space which will be used throughout the experiments on the NMGMM.
- Introduce the baseline model to compare against the NMGMM.
- Establish the initialisation procedure which will be used for both the NMGMM and the GMM baseline model.
- The results of performing Bayesian hyperparameter optimisation, and the identified optimal hyperparameter for each artificial shape in the dataset.

6.1 Dataset

To evaluate the performance of the NMGMM, we will be using a dataset of artificial shapes created by Wenliang, Li et al. [35]. This dataset presents multiple artificial shapes which are typically difficult for monotonic mixtures to fit to, as they will require considerably more components to get a good fit. One example is the ”ring” shape, which simply distributes the points along a ring. A monotonic GMM will have to fit multiple components along the ridge to get a good fit, whereas the NMGMM should in theory only need 2 components. Where one is positive and overlaps the points, and one negative to reduce the probability mass in the middle close to zero. The shapes, as presented by Wenliang, Li et al. [35] can be observed in Figure 6.1.

All shapes in this collection have shapes which in theory should enable the NMGMM to learn negative components to subtract away probability mass, and therefore achieve a better fit than monotonic GMMs. It is worth noting, however, that these shapes are all in two dimensions. This project will not perform any experiments in higher dimensions, as that will further complicate the initialisation procedure of the NMGMM. Further,

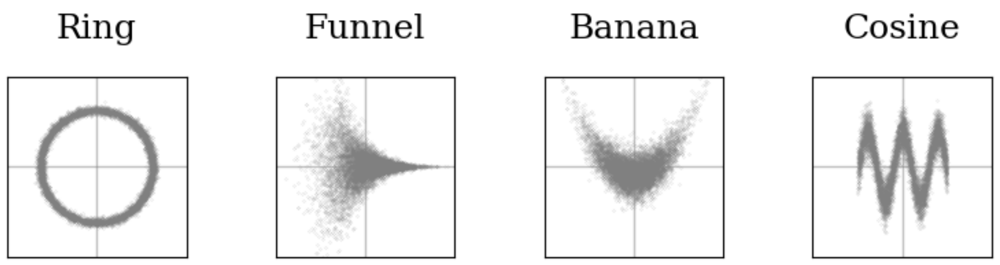

Figure 6.1: A collection of complex artificial shapes presented by Wenliang, Li et al.

work is necessary to better understand effective initialisations for the NMGMM to be able to perform meaningful experiments in higher dimensions.

Additionally, we construct a train, validation and test subset for each dataset shape. Based on the code provided by Wenliang, Li et al. [35] we sample k number of points at random from the distribution of the shape. The number of points for each split can be seen in Table 6.1.

Train size	Validation size	Test size
1000	200	200

Table 6.1: Train, validation and test split denoted by the number of points sampled from the distribution of each shape.

6.2 Baseline

To effectively compare the benefits of the NMGMM, we will be using a monotonic GMM which is trained using expectation maximisation (EM). For the GMM baseline implementation and the EM optimiser, we will be using the implementation provided by Scikit-Learn [28]. The EM optimiser is traditionally used for monotonic GMMs as it provides some convergence guarantees which numerical optimisers are not able to provide to the same extent. Therefore, using EM for the baseline experiments is the most appropriate strategy for comparing it against the NMGMM. Further, the baseline will be using its own initialisations which Scikit-Learn provides, as the initialisations for the NMGMM are crafted specifically to support negative components. Therefore, we will be using K-means initialisations for the baseline experiments.

Additionally, to give a fair comparison with some NMGMM initialised for, N we will be using $(N(N+1))2^{-1}$ components for the baseline GMM. As the NMGMM squares the circuit it will have N^2 components, however as it is produced by doing the Cartesian product we will have some components such as $A \times B$ and $B \times A$ which will be identical. Thus, we ignore these for the baseline by treating such components as the same.

6.3 NMGMM Initialisation Procedure

For initialising the NMGMM, the mean and the covariances are initialised according to the positive and negative spaces for each dataset shape. the components of the NMGMM places the components which should be positive over the points in the dataset, and the negative components are placed in the negative spaces such that the NMGMM can learn to subtract this probability mass. Further, the weights were initialised at random between 0 and 1, such that we can observe that the NMGMM correctly identifies which components are supposed to be negative, thus subtracting away probability mass in those areas. To measure the NMGMM against the baseline, we initialise the baseline to have the same number of components as the squared circuit representing the NMGMM, the initialisation for each shape can be seen in Figure 7.1.

6.4 Evaluation Metrics

To evaluate the NMGMM against the monotonic GMM, we compare the negative log likelihood, which is computed according to Equation 5.4. Further, we will present the heatmaps of the resulting fits for the NMGMM as well as the confidence plots, which will highlight the positive and negative components according to the ellipses which are formed by the covariance matrices. For each confidence ellipse, we will be using a standard deviation of 2.3 to highlight the area where most of the probability mass is located for each component. Other evaluation metrics such as silhouette score were considered, however since the NMGMM will have negative components this score would be misleading as each negative component would contribute to a worse silhouette score when that might not actually be the case.

6.5 Hyperparameter Optimisation

Since the NMGMM introduces additional hyperparameters for both the optimiser and the model, we chose to perform a Bayesian hyperparameter search. The goal of this search was to observe the optimal hyperparameter configuration for each dataset shape, where each configuration was measured against the validation set, using the negative log likelihood loss. Therefore, through this experiment, we aim to answer the following research questions.

1. How many components are necessary in the NMGMM to have enough degrees of freedom to create generalisable fits for the shapes presented?
2. What is the effect of adding the sparsity prior, and which shapes have a higher gain when using higher sparsity priors?
3. What hyperparameters have the most significant impact, for the different shapes in the dataset?

Further, to ensure that the configuration indeed resulted in a good fit, we also plotted the resulting fits for each configuration, such that we could discard configurations where the fit resulted in small components with exploded weights. This often happened when the sparsity prior was set too low, thus we selected the hyperparameters which had low validation loss and a good fit according to the data distribution. The resulting hyperparameter configurations for each shape can be observed in Table 6.2 and Table 6.3.

Dataset	Components	Covariance Shape
Ring	2	Diagonal
Funnel	4	Full
Banana	3	Full
Cosine	6	Diagonal

Table 6.2: Optimal number of components, obtained through Bayesian hyperparameter optimisation.

To get a better understanding of the importance of the different hyperparameters for the different shapes, we also denote a prior weight for each hyperparameter. These priors are obtained through training a random forest supplied by Weights and Biases [34], which takes the hyperparameters as input and the validation loss as an output to calculate the feature importance of the random forest. These importance priors can be observed next to the hyperparameter values in Table 6.3.

Dataset	Iterations	Batch size	Learning rate	Momentum	Sparsity prior
Ring	100	64 (0.24)	0.0004 (0.52)	0.57 (0.08)	0.37 (0.16)
Funnel	500	128 (0.57)	0.001 (0.23)	0.94 (0.13)	0.89 (0.07)
Banana	500	128 (0.15)	0.0006 (0.34)	0.79 (0.26)	0.29 (0.25)
Cosine	100	32 (0.20)	0.0006 (0.24)	0.82 (0.41)	0.24 (0.15)

Table 6.3: Optimal hyperparameter configuration for each dataset for the NMGMM model, obtained by performing Bayesian hyperparameter optimisation.

From the results of the hyperparameter optimisation in Table 6.3 we observe some interesting results. For batch size, we see that cosine requires the lowest batch size, which is due to the fact that there are more components and the shape itself is more complex to fit. Additionally, we see that the funnel and banana shape requires more iterations to achieve convergence, which is in large part caused by the optimiser having to scale down the positive components sufficiently, as seen in Figure 7.1. Moreover, the learning rate and the momentum seems to be quite stable for all the different shapes except for the funnel which handles a higher learning rate.

Lastly, for the sparsity prior, we observe that some shapes have a higher importance on this parameter. Such as the funnel, which has an importance prior of 0.89. This, makes sense as we observed configurations which had collapsed covariances with exploded weights for this shape when the sparsity was set low. Thus, we see that the sparsity prior can indeed aid the NMGMM to create better fits.

Chapter 7

Results

Given the learning scheme presented in Chapter 5 and the experimental setup in the previous chapter, we will now present the results of the experiments with optimal hyperparameters for each dataset shape. These results will highlight the expressiveness of the NMGMM as well as highlight its limitations, which will provide a motivation for any future improvements of our proposed framework. The initialisations and final fits for the NMGMM can be seen in Figure 7.1, additionally the fits for the monotonic GMM equivalents can be seen on the right.

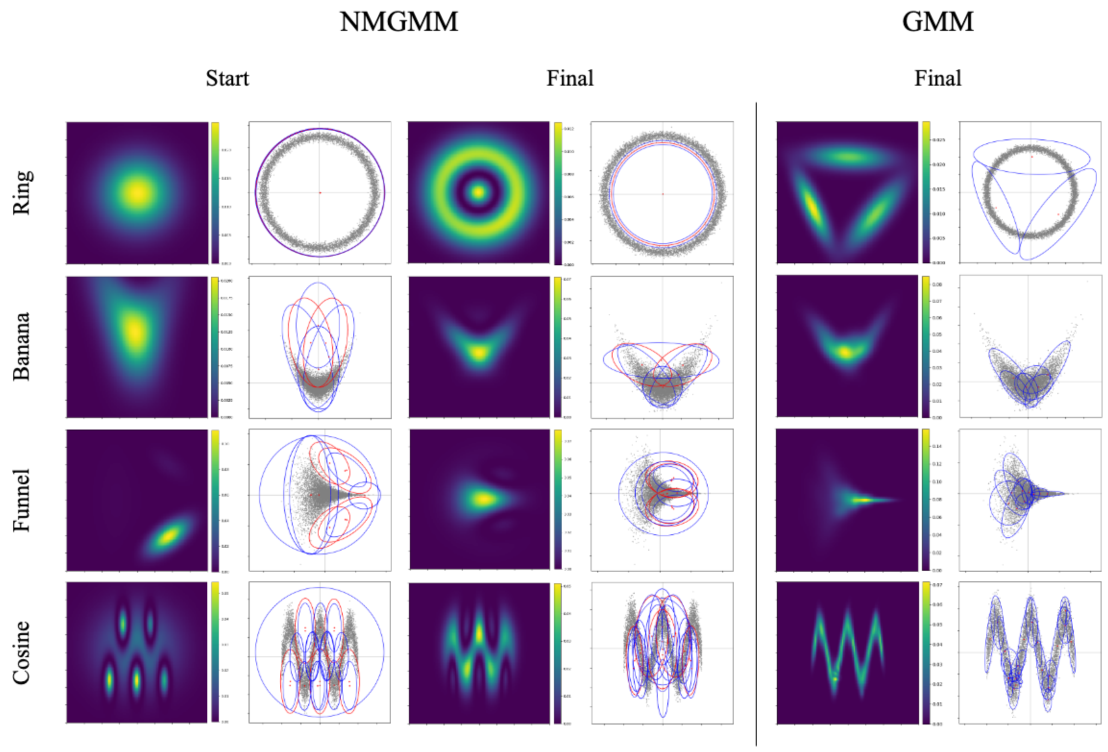

Figure 7.1: Fits achieved for the NMGMM with the initialisations on the left with the equivalent monotonic GMM on the right.

In the following sections, we will provide a more detailed breakdown of these results, as well as the learning curves and final loss values when compared against the monotonic GMM.

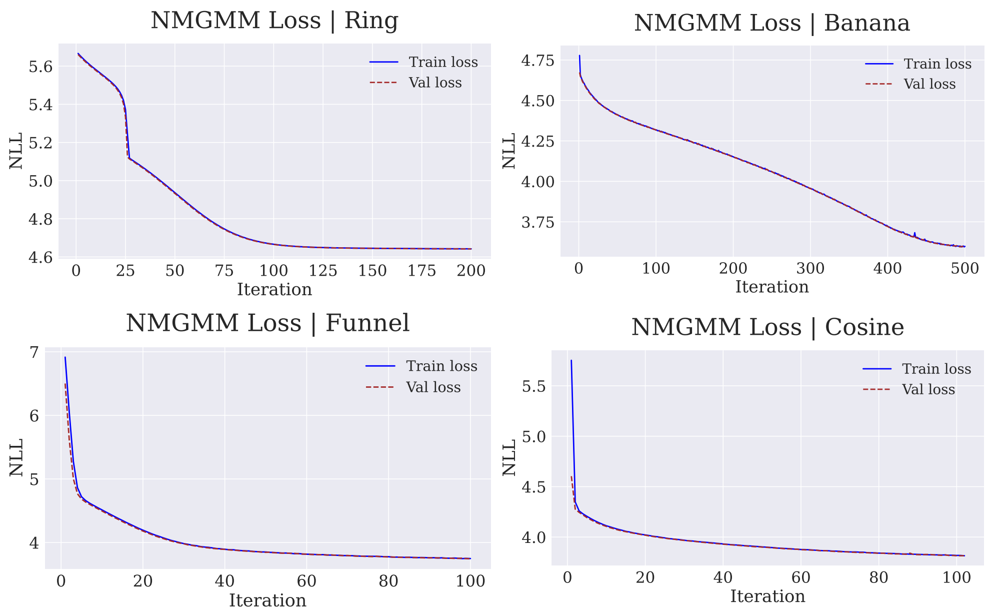

Figure 7.2: Loss curves for each shape fitted with the NMGMM.

In Figure 7.2 we observe that the NMGMM does converge for all shapes, however the final loss value for each shape are either slightly higher or comparable to the monotonic GMMs at convergence. These results can be observed in Table 7.1.

Dataset	Model	Components	Train loss	Validation loss	Test loss
Ring	NMGMM	2	4.6421	4.6417	4.6417
	GMM	3	4.6401	4.6526	4.6332
Funnel	NMGMM	4	3.7093	3.7078	3.7078
	GMM	10	3.4856	3.4626	3.4924
Banana	NMGMM	3	3.5948	3.5907	3.5907
	GMM	6	3.5423	3.5372	3.5699
Cosine	NMGMM	6	3.8198	3.8423	3.8423
	GMM	21	3.4953	3.5089	3.5068

Table 7.1: Final loss values for each dataset shape, comparing the results for the NMGMM with the GMM at convergence. Additional results for varying number of components can be seen in Table B.1 of the Appendix.

In Table 7.1 it is clear that the NMGMM achieves a comparable result to the monotonic GMM, however the NMGMM is able to create more uniform fits which generalise better than the GMM. In fact, the average generalisation gap between the train and validation loss for the NMGMM is 0.007, whereas the GMM has an average generalisation gap of 0.014. This demonstrates that the NMGMM is able to create more generalised fits for these complex shapes as it can subtract probability masses, additionally, the NMGMM requires considerably fewer parameters to be optimised to achieve this.

These results seem promising, however there are some interesting patterns in Figure 7.1 which are yet to be discussed. Therefore, in the following section, we will break down the results we observe for each dataset shape to better understand how the NMGMM is achieving these convergences and thus highlight some of its limitations.

7.1 Probability Mass in Negative Components

For the ring shape, we observe that there exists a small blob in the middle of the ring at convergence. In fact, in Figure 7.3 we observe that for iteration 39 it has a perfect ring, but this ring does not stretch fully out to the points. This seems somewhat counterintuitive as the probability mass should be distributed along the border of the ring to achieve a perfect fit and thus outperform the GMM considerably. However, this is an inherent limitation of the NMGMM, and we hinted at this in Chapter 4 on the preliminary analysis.

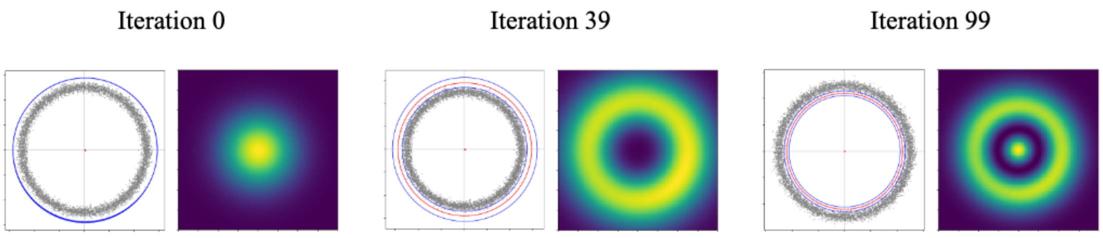

Figure 7.3: Fits achieved for NMGMM over iteration 0, 39 and the final iteration at convergence. Demonstrating how the blob in the middle occurs.

We can explain this pattern by looking at the distribution of the multivariate Gaussians over the x-axis, which has one negative component which we denote as A and one positive denoted by B. After squaring the circuit, we will have two positive components $B \times B$ and $A \times A$, and two negative $A \times B$ and $B \times A$. We can then investigate why this happens when we plot the distributions of the NMGMM along the x-axis, as seen in Figure 7.4.

In Figure 7.4 we observe that for iteration 39, we still have a slight blob in the middle. However, it is not visible since the gap between the peaks of the distributions for component *AxB* and *BxB* is very small, thus the probability mass is negligible. For iteration 99 however, the covariances are squished slightly which further exaggerates this gap, thus producing a very visible blob in the middle. This is an inevitable fact about our NMGMM model, because as we saw in the Chapter 4 on the preliminary analysis, the negative components will always produce a positive one with the same means and with variance values which are effectively halved. Which will produce a slight gap between the peaks of the negative component and the positive component as observed in Figure 7.4.

This observation is also consistent across the other shapes we have experimented with. Particularly, for the cosine and funnel shape, we observe a very small probability mass

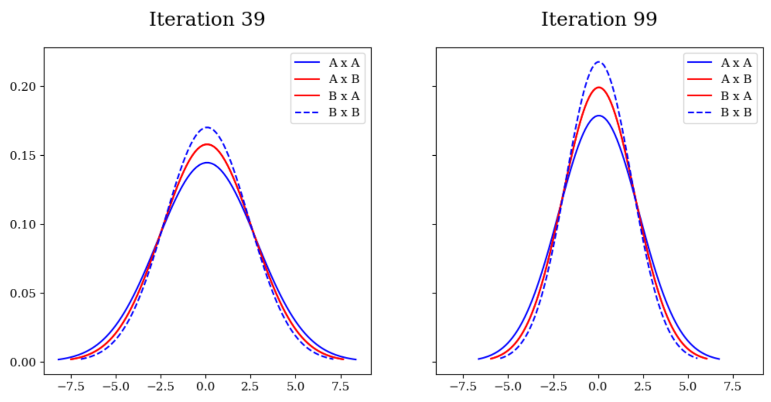

Figure 7.4: Distribution of the components in the NMGMM for the ring shape along the x-axis.

in the middle of where the negative components are placed. However, for the banana shape, we observe a very interesting pattern. Due to the way that the NMGMM is initialised for the banana shape, it learns to place the new negative components in such a way that the positive component in the middle is almost fully faded away, whilst also subtracting away the necessary probability mass. It manages to do this as the negative components have full covariance matrices and this is the placement of the positive components, with respect to the negative, does not have the same bounds as diagonal covariance matrices have. Thus, enabling the NMGMM to get a good uniform fit for the banana shape without much probability mass remaining from the negative components.

7.2 Additional Components

Furthermore, we experimented with using additional components for the NMGMM. During our sweeps for the Bayesian hyperparameter optimisation, we observed that adding more components for the NMGMM had very little effect on the validation loss. The idea behind using additional components was to effectively remove the probability mass we observe in the middle of negative ones. However, to do this we have to strategically place new positive components such that the product between these components and the negative components can produce new negative components which are able to subtract away this probability mass.

The results of these experiments can be observed in Table B.1 of the appendix, along with the initialisations in Figure B.1. For the funnel we are able to remove some of the mass produced by the negative components which we can observe when looking at the 3 component vs the 4 component initialisation. This did in fact reduce the loss by 0.1378, however, increasing the number of components beyond 4 simply increased the loss, which highlights the difficulty of effectively fitting the NMGMM due to the number of components which are dependent on each other. Thus, these results highlight that the NMGMM can produce decent fits for complex shapes with negative spaces. It does not reduce the overall loss when compared to the equivalent GMMs, however it produces fits which are more generalisable and uniform with considerably fewer parameters than comparable GMM instances.

Chapter 8

Conclusion

In this project, we have explored how we can build a generalisable framework for achieving NMMMs. Additionally, we explored how this technique comes with some additional complexity in terms of the squared circuit, namely introducing new components which are dependent on each other. Based on that analysis, we addressed these issues by introducing weight regularisation to promote empty clusters, which was combined with SGD with momentum to achieve effective learning of both shallow and deep mixtures. Finally, we presented the results of our experiments, which demonstrated how the NMGMM can achieve more generalised fits than monotonic GMMs with considerably fewer parameters.

Based on this work, we can provide the following conclusions to the research questions we set out to answer.

Research question 1: is it feasible to create a generalised framework for NMMMs which can support deep mixtures and multiple parametric families?

It is indeed feasible to create a generalised framework for NMMMs, and our approach can support mixtures of multiple parametric families as long as the computation of the normalisation constant is tractable. Additionally, our method can support deep mixtures, as our learning framework is based on techniques often used to learn deep mixtures.

Research question 2: can such a generalised framework implemented as a shallow non-monotonic Gaussian mixture (NMGMM) converge to a lower loss than shallow monotonic Gaussian mixtures with the same number of components for shapes with negative spaces?

As discussed in the previous chapter, the NMGMM is not able to converge to a lower loss than monotonic GMMs for the shapes we have experimented with. However, it can converge very close to the same loss value of an equivalent GMM whilst requiring considerably fewer parameters. Which demonstrates the flexibility of the NMGMM as a model for learning complex shapes with negative spaces.

Research question 3: can the NMGMM provide more generalised fits compared to monotonic GMMs?

The NMGMM does indeed create more uniform and generalisable fits than the monotonic GMM. As we saw in the previous chapter, the GMM will have to place multiple small components along the distribution of points. Thus resulting in fits which are not uniformly distributed, whereas the NMGMM can simply place a large positive component where the bulk of the points are, and then clip away the excess probability mass. However, we do observe that for certain initialisations, some lingering probability mass exists where we place the negative components. Therefore, any future work will have to investigate efficient algorithms for initialising the NMGMM.

8.1 Limitations and Future Directions

So far we have seen a few limitations of the NMGMM, specifically we know that it has a lot of components which are dependent on each other. This can limit the model in that it makes it harder to optimise, for example, negative components in the NMGMM are product components which consist of a negative and positive component. Thus, any negative component in our model will be dependent on a positive one. However, there are ways of tackling this. One way is to tackle this is to construct specially crafted initialisations such that these dependencies are not an issue. Which highlights a second limitation for our NMGMM, namely its sensitivity to initialisations. Therefore, any future work should investigate initialisation procedures which can enable effective learning of NMGMMs.

Furthermore, we observed in our results that the NMGMM can produce probability mass inside negative components. However, this is obviously not desirable. This can be addressed by creating better initialisations, such as the result we observed for the banana shape. However, for any practical application this should be automated, and a future direction would be to explore additional techniques for minimising the likelihood produced inside negative components.

In our experiments, we have only tested our model using simple mixtures, which are shallow and in two dimensions. However, to better evaluate the generalisability of our model, it is necessary to perform additional experiments using deep mixtures and mixtures of multiple parametric families. This would also highlight which families that are tractable for computing the normalisation constant and which combinations that are possible to make.

In conclusion, we have seen that our approach provides a good foundation for achieving generalised NMMMs. Where a shallow NMGMM can produce more generalisable fits than monotonic GMMs whilst using considerably fewer parameters. However, work still remains to investigate its performance on complex mixtures which are deep or with multiple parametric families. Additionally, we would like to see some new proposals for initialisation algorithms of our model to capture negative components. Overall, we can conclude that our NMMM implementation improves the overall flexibility of mixture models.

Bibliography

[1] John Bridle. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989.

[2] Olivier Cappé and Eric Moulines. On-Line Expectation–Maximization Algorithm for latent Data Models. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 71(3):593–613, 02 2009.

[3] G. Celeux and J. Diebolt. The SEM algorithm: a probabilistic teacher algorithm derived from the EM algorithm for the mixture problem. *Computational Statistics Quarterly*, 2:73–82, 1985.

[4] Faicel Chamroukhi and Bao Tuyen Huynh. Regularized maximum-likelihood estimation of mixture-of-experts for regression and clustering. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2018.

[5] Vedang Chauhan and Brian Surgenor. A comparative study of machine vision based methods for fault detection in an automated assembly machine. *Procedia Manufacturing*, 1:416–428, 2015. 43rd North American Manufacturing Research Conference, NAMRC 43, 8-12 June 2015, UNC Charlotte, North Carolina, United States.

[6] Jianfei Chen, Jun Zhu, Yee Whye Teh, and Tong Zhang. Stochastic expectation maximization with variance reduction. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[7] Xiaohan Ding, guiguang ding, Xiangxin Zhou, Yuchen Guo, Jungong Han, and Ji Liu. Global sparse momentum sgd for pruning very deep neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[8] Laurent Dinh, Jascha Sohl-Dickstein, Razvan Pascanu, and Hugo Larochelle. A RAD approach to deep mixture models. *CoRR*, 2019.

[9] Alexander Gepperth and Benedikt Pfülb. Gradient-based training of gaussian mixture models in high-dimensional spaces. *CoRR*, abs/1912.09379, 2019.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[11] Reshad Hosseini and Suvrit Sra. Matrix manifold optimization for gaussian mixtures. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[12] Reshad Hosseini and Suvrit Sra. An alternative to em for gaussian mixture models: Batch and stochastic riemannian optimization. *Mathematical Programming*, 181, 06 2017.

[13] Yuan Ji, Chunlei Wu, Ping Liu, Jing Wang, and Kevin R. Coombes. Applications of beta-mixture models in bioinformatics. *Bioinformatics*, 21(9):2118–2122, 02 2005.

[14] R. Jiang, M.J. Zuo, and H.-X. Li. Weibull and inverse weibull mixture models allowing negative weights. *Reliability Engineering & System Safety*, pages 227–234, 1999.

[15] Kenneth Lange. A gradient algorithm locally equivalent to the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 57(2):425–437, 1995.

[16] Mei Liu, Liangming Chen, Xiaohao Du, Long Jin, and Mingsheng Shang. Activated gradients for deep neural networks. *CoRR*, abs/2107.04228, 2021.

[17] Shilin Liu and Khe Chai Sim. On combining dnn and gmm with unsupervised speaker adaptation for robust automatic speech recognition. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 195–199, 2014.

[18] Yanli Liu, Yuan Gao, and Wotao Yin. An improved analysis of stochastic gradient descent with momentum. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 18261–18271. Curran Associates, Inc., 2020.

[19] Geoffrey J. McLachlan. *Finite mixture models / Geoffrey McLachlan, David Peel.* Wiley, first edition, 2000.

[20] Geoffrey J. McLachlan, Sharon X. Lee, and Suren I. Rathnayake. Finite mixture models. *Annual Review of Statistics and Its Application*, 6(1):355–378, 2019.

[21] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, first edition, 2012.

[22] Samet Oymak and Talha Cihad Gulcu. A theoretical characterization of semi-supervised learning with self-training for gaussian mixture models. In Arindam Banerjee and Kenji Fukumizu, editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 3601–3609. PMLR, 13–15 Apr 2021.

[23] Michael Parker. Chapter 13 - matrix inversion. In Michael Parker, editor, *Digital Signal Processing 101 (Second Edition)*, pages 149–162. Newnes, second edition edition, 2017.

[24] Razvan Pascanu, Tomás Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR*, abs/1211.5063, 2012.

[25] K. B. Petersen and M. S. Pedersen. The matrix cookbook, nov 2012. Version 20121115.

[26] Guillaume Rabusseau and François Denis. Learning negative mixture models by tensor decompositions. *CoRR*, 2014.

[27] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[28] Scikit-Learn. Gaussian mixture, 2023.

[29] Jun Sun, Chao Li, Xiao-Jun Wu, Vasile Palade, and Wei Fang. An effective method of weld defect detection and classification based on machine vision. *IEEE Transactions on Industrial Informatics*, 15(12):6322–6333, 2019.

[30] D. M. Titterington, Adrian F. M. Smith, and U. E. Makov. *Statistical analysis of finite mixture distributions*. Wiley series in probability and mathematical statistics. Wiley, Chichester, 1985.

[31] Aaron van den Oord and Benjamin Schrauwen. Factoring variations in natural images with deep gaussian mixture models. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2014.

[32] Antonio Vergari, YooJung Choi, Anji Liu, Stefano Teso, and Guy Van den Broeck. A compositional atlas of tractable circuit operations for probabilistic inference. In *NeurIPS, December 2021*, pages 13189–13201, 2021.

[33] Cinzia Viroli and Geoffrey J. McLachlan. Deep gaussian mixture models, 2017.

[34] Weights and Biases. Parameter importance, 2023.

[35] Li Wenliang, Danica J. Sutherland, Heiko Strathmann, and Arthur Gretton. Learning deep kernels for exponential family densities, 2021.

[36] D.S. Wilks. Chapter 4 - parametric probability distributions. In Daniel S. Wilks, editor, *Statistical Methods in the Atmospheric Sciences*, volume 100 of *International Geophysics*, pages 71–131. Academic Press, 2011.

[37] Jon Williamson. Philosophies of probability. In *Philosophy of Mathematics*, Handbook of the Philosophy of Science, pages 493–533. North-Holland, 2009.

[38] Xue Ying. An overview of overfitting and its solutions. *Journal of Physics: Conference Series*, 1168(2):022022, feb 2019.

[39] Baibo Zhang and Changshui Zhang. Finite mixture models with negative components. In *Machine Learning and Data Mining in Pattern Recognition*, pages 31–41. Springer Berlin Heidelberg, 2005.

[40] Marina Zimmermann, Mostafa Mehdipour Ghazi, Hazım Kemal Ekenel, and Jean-Philippe Thiran. Visual speech recognition using pca networks and lstms in a tandem gmm-hmm system. In Chu-Song Chen, Jiwen Lu, and Kai-Kuang Ma, editors, *Computer Vision – ACCV 2016 Workshops*, pages 264–276, Cham, 2017. Springer International Publishing.

Appendix A

Derivations

A.1 Diagonal NMGMM Transformation of Means

Given some GMM with possibly negative weights, we produce the NMGMM by squaring the circuit of the GMM. Thus, we will have some component A and component B with diagonal covariances from the GMM, which will be multiplied to produce a new component in the NMGMM. Further, the covariance matrix for component B is defined by scaling the covariance matrix for component A using a scaling vector δ, which will help provide some insight into the relationship between the two components.

$$\Sigma_A = \begin{bmatrix} a_{11} & 0 \\ 0 & a_{22} \end{bmatrix} \quad \mu_A = \begin{bmatrix} x_A \\ y_A \end{bmatrix} \qquad \Sigma_B = \Sigma_A \begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix} \quad \mu_B = \begin{bmatrix} x_B \\ y_B \end{bmatrix} \tag{A.1}$$

Given Equation 3.5 for the definition of the parameters for the squared mixture, we first have to compute the covariance matrix of the new component to get the expression for its mean. By inserting the above into Equation 3.5 we get the following.

$$\Sigma_C = \left(\begin{bmatrix} a_{11} & 0 \\ 0 & a_{22} \end{bmatrix}^{-1} + \begin{bmatrix} \delta_1 a_{11} & 0 \\ 0 & \delta_2 a_{22} \end{bmatrix}^{-1} \right)^{-1} \tag{A.2}$$

$$= \left(\begin{bmatrix} \dfrac{1+\delta_1}{a_{11}} & 0 \\ 0 & \dfrac{1+\delta_2}{a_{22}} \end{bmatrix} \right)^{-1} = \begin{bmatrix} \dfrac{a_{11}}{1+\delta_1} & 0 \\ 0 & \dfrac{a_{22}}{1+\delta_2} \end{bmatrix} \tag{A.3}$$

Given the above derivation for, Σ_C we can now derive the expression for the means of the new component in the squared mixture, expressed as μ_C.

$$\mu_C = \Sigma_C \left(\begin{bmatrix} a_{11} & 0 \\ 0 & a_{22} \end{bmatrix}^{-1} \begin{bmatrix} x_A \\ y_A \end{bmatrix} + \begin{bmatrix} \delta_1 a_{11} & 0 \\ 0 & \delta_2 a_{22} \end{bmatrix}^{-1} \begin{bmatrix} x_B \\ y_B \end{bmatrix} \right) \tag{A.4}$$

$$= \Sigma_C \left(\begin{bmatrix} \frac{x_A}{a_{11}} \\ \frac{y_A}{a_{22}} \end{bmatrix} + \begin{bmatrix} \frac{\delta_1 x_B}{a_{11}} \\ \frac{\delta_2 y_B}{a_{22}} \end{bmatrix} \right) = \Sigma_C \begin{bmatrix} \frac{x_A + \delta_1 x_B}{a_{11}} \\ \frac{y_A + \delta_2 y_B}{a_{22}} \end{bmatrix} \tag{A.5}$$

$$= \begin{bmatrix} \frac{x_A + \delta_1 x_B}{1+\delta_1} \\ \frac{y_A + \delta_2 y_B}{1+\delta_2} \end{bmatrix} \tag{A.6}$$

Thus, we have computed the general expression for the placement of component C, which is given by μ_C.

A.2 Full NMGMM Transformation of Means

Given some GMM with possibly negative weights, we produce the NMGMM by squaring the circuit of the GMM. Thus, we will have some component A and component B with full covariances from the GMM, which will be multiplied to produce a new component in the NMGMM. Further, the covariance matrix for component B is defined by scaling the covariance matrix for component A using a scaling vector δ, which will help provide some insight into the relationship between the two components.

$$\Sigma_A = \begin{bmatrix} a_{11} & 0 \\ 0 & a_{22} \end{bmatrix} \quad \mu_A = \begin{bmatrix} x_A \\ y_A \end{bmatrix} \qquad \Sigma_B = \delta\, \Sigma_A \quad \mu_B = \begin{bmatrix} x_B \\ y_B \end{bmatrix} \tag{A.7}$$

As for the derivation with diagonal covariance matrices, we will insert the above parameters into Equation 3.5 to first compute the covariance matrix of component C, given by Σ_C.

$$\Sigma_C = \left(\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}^{-1} + \begin{bmatrix} \delta a_{11} & \delta a_{12} \\ \delta a_{21} & \delta a_{22} \end{bmatrix}^{-1} \right)^{-1} \tag{A.8}$$

$$= \left(\begin{bmatrix} a_{22}|\Sigma_A|^{-1} & -a_{12}|\Sigma_A|^{-1} \\ -a_{21}|\Sigma_A|^{-1} & a_{11}|\Sigma_A|^{-1} \end{bmatrix} + \begin{bmatrix} \delta a_{22}|\Sigma_B|^{-1} & -\delta a_{12}|\Sigma_B|^{-1} \\ -\delta a_{21}|\Sigma_B|^{-1} & \delta a_{11}|\Sigma_B|^{-1} \end{bmatrix} \right)^{-1} \tag{A.9}$$

For now we will ignore computing $|\Sigma_A|$, however we will compute $|\Sigma_B|$ as it can be expressed in terms of $|\Sigma_A|$ which will simplify the computation further.

$$|\Sigma_B|^{-1} = (\delta^2 a_{22}a_{11} - \delta^2 a_{12}a_{21})^{-1} \tag{A.10}$$

$$= (\delta^2(a_{22}a_{11} - a_{12}a_{21}))^{-1} \tag{A.11}$$

Notice that the term $a_{11}a_{22} - a_{12}a_{21}$ is the same as the determinant of component A, thus we can simplify further.

$$= \delta^{-2}|\Sigma_A|^{-1} \tag{A.12}$$

Given this experession for the inverse determinant of component B we can derive the expression for the Σ_C.

$$\Sigma_C = |\Sigma_C|^{-1} \begin{bmatrix} a_{11}(\delta+1)|\Sigma_A|^{-1}\delta^{-1} & a_{12}(\delta+1)|\Sigma_A|^{-1}\delta^{-1} \\ a_{21}(\delta+1)|\Sigma_A|^{-1}\delta^{-1} & a_{22}(\delta+1)|\Sigma_A|^{-1}\delta^{-1} \end{bmatrix} \tag{A.13}$$

Interestingly, we can compute the determinant for Σ_C as an expression of $|\Sigma_A|$.

$$\begin{aligned} |\Sigma_C|^{-1} &= ((a_{11}a_{22}(\delta+1)^2|\Sigma_A|^{-2}\delta^{-2}) - (a_{12}a_{21}(\delta+1)^2|\Sigma_A|^{-2}\delta^{-2}))^{-1} && \text{(A.14)} \\ &= ((\delta+1)^2\delta^{-2}|\Sigma_A|^{-2}(a_{11}a_{22} - a_{12}a_{21}))^{-1} && \text{(A.15)} \\ &= ((\delta+1)^2\delta^{-2}|\Sigma_A|^{-2}|\Sigma_A|)^{-1} && \text{(A.16)} \\ &= (\delta+1)^{-2}\delta^2|\Sigma_A| && \text{(A.17)} \end{aligned}$$

Finally, using this expression for $|\Sigma_C|$ we can compute Σ_C.

$$\Sigma_C = \begin{bmatrix} \delta^2 a_{11}(\delta+1)^{-1} & \delta^2 a_{12}(\delta+1)^{-1} \\ \delta^2 a_{21}(\delta+1)^{-1} & \delta^2 a_{22}(\delta+1)^{-1} \end{bmatrix} \tag{A.18}$$

Again using Equation 3.5 we can compute the expression for the means of component C with respect to component A and component B. Firstly, we will compute $\Sigma_A^{-1} \times \mu_A$ and $\Sigma_B^{-1} \times \mu_B$ as a part of deriving the expression of the means.

$$\begin{aligned} \Sigma_A^{-1} \times \mu_A &= \begin{bmatrix} a_{22}|\Sigma_A|^{-1} & -a_{12}|\Sigma_A|^{-1} \\ -a_{21}|\Sigma_A|^{-1} & a_{11}|\Sigma_A|^{-1} \end{bmatrix} \begin{bmatrix} x_A \\ y_A \end{bmatrix} && \text{(A.19)} \\ &= \begin{bmatrix} (a_{22}x_A - a_{12}y_A)|\Sigma_A|^{-1} \\ (-a_{21}x_A + a_{11}y_A)|\Sigma_A|^{-1} \end{bmatrix} && \text{(A.20)} \\ \Sigma_B^{-1} \times \mu_B &= \begin{bmatrix} a_{22}(\delta|\Sigma_A|)^{-1} & -a_{12}(\delta|\Sigma_A|)^{-1} \\ -a_{21}(\delta|\Sigma_A|)^{-1} & a_{11}(\delta|\Sigma_A|)^{-1} \end{bmatrix} \begin{bmatrix} x_B \\ y_B \end{bmatrix} && \text{(A.21)} \\ &= \begin{bmatrix} (a_{22}x_B - a_{12}y_B)(\delta|\Sigma_A|)^{-1} \\ (-a_{21}x_B + a_{11}y_B)(\delta|\Sigma_A|)^{-1} \end{bmatrix} && \text{(A.22)} \end{aligned}$$

To get the second part of the expression of the means for component C we have to add these terms together, thus producing.

$$\Sigma_A^{-1} \times \mu_A + \Sigma_B^{-1} \times \mu_B = \begin{bmatrix} (a_{22}(\delta x_A + x_B) - a_{12}(\delta y_A + y_B))(\delta|\Sigma_A|)^{-1} \\ (-a_{21}(\delta x_A + x_B) + a_{11}(\delta y_A + y_B))(\delta|\Sigma_A|)^{-1} \end{bmatrix} \tag{A.23}$$

Appendix B

Additional Experiments

Dataset	Model	Components	Train loss	Validation loss	Test loss
Ring	NMGMM	2	4.6421	4.6417	4.6417
	GMM	3	4.6401	4.6526	4.6332
	NMGMM	3	4.6545	4.6545	4.6545
	GMM	6	3.5582	3.566	3.5640
	NMGMM	4	4.5761	4.5746	4.5746
	GMM	10	3.3029	3.3029	3.3029
Funnel	NMGMM	3	3.8471	3.8517	3.8517
	GMM	6	3.4983	3.4645	3.4974
	NMGMM	4	3.7093	3.7078	3.7078
	GMM	10	3.4821	3.4551	3.4868
	NMGMM	5	3.7227	3.7241	3.7241
	GMM	15	3.4693	3.4464	3.4783
Banana	NMGMM	3	3.5948	3.5907	3.5907
	GMM	6	3.5423	3.5372	3.5699
	NMGMM	4	3.6211	3.6219	3.6219
	GMM	10	3.5376	3.5419	3.5678
	NMGMM	5	3.7847	3.7885	3.7885
	GMM	15	3.5256	3.5248	3.5502
Cosine	NMGMM	6	3.8198	3.8423	3.8423
	GMM	21	3.4922	3.5149	3.5062
	NMGMM	8	3.7222	3.7429	3.7429
	GMM	36	3.4912	3.5087	3.5052
	NMGMM	10	4.1004	4.1185	4.1185
	GMM	55	3.4839	3.5000	3.4952

Table B.1: Result of varying number of components for the different shapes introduced in Section 6.1. The initialisations for each number of components can be seen in Section B.1.

B.1 Initialisations

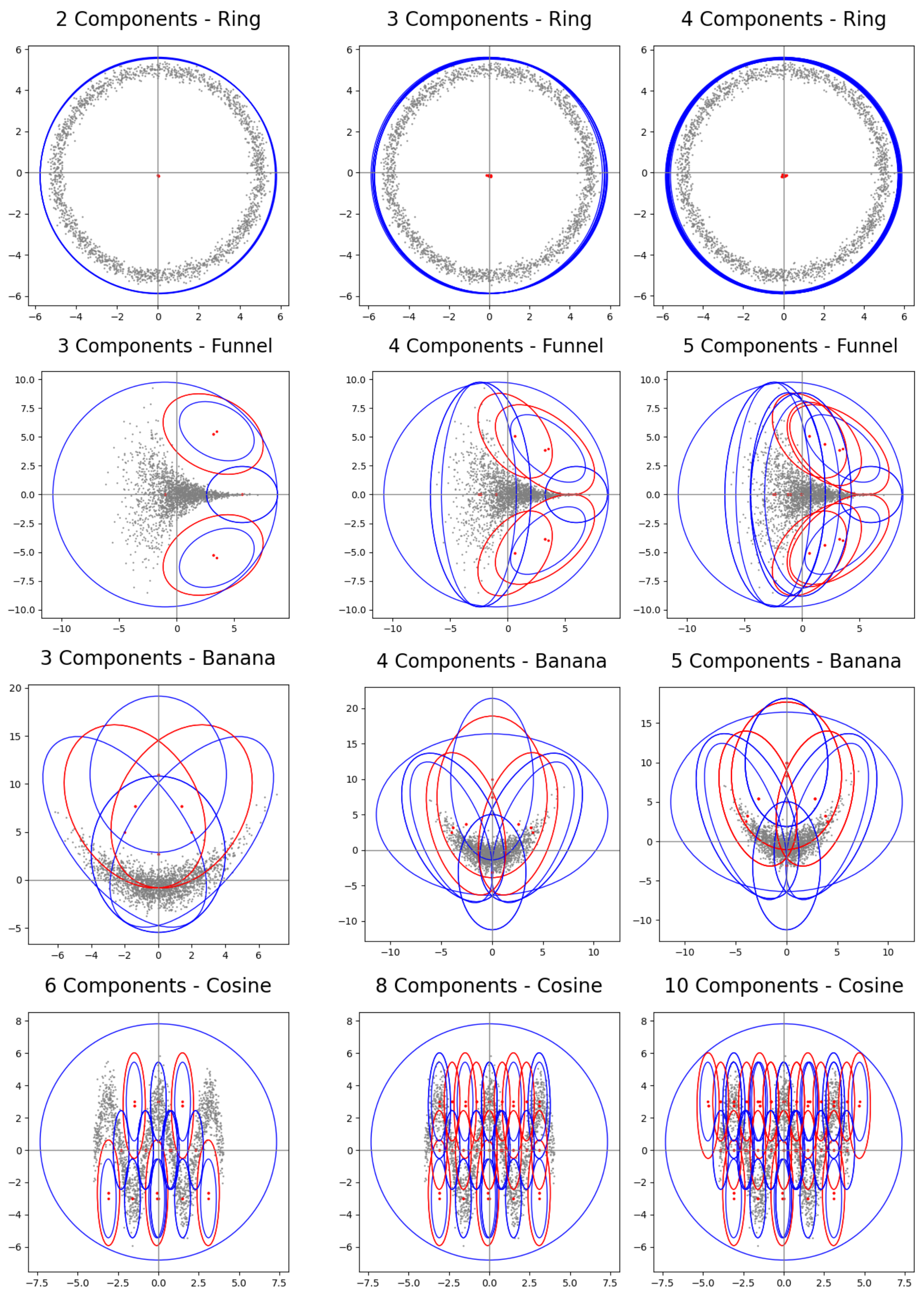

Figure B.1: Initialisations used for different number of components for the different shapes presented in Section 6.1. Note that the number of components in the titles are before the circuit is squared.