Self-Supervised Learning of Tractable Generative Models

Tom A. Lamb

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2023

Abstract

Einsum Networks (EiNets) are an efficient implementation of a general class of probabilistic models known as probabilistic circuits (PCs). These models have advantages over expressive generative models such as VAEs and GANs because they allow for exact and efficient probabilistic inference of various types. However, as PCs grow in the number of parameters, they become more challenging to train. In particular, they have been shown to be susceptible to ubiquitous problems in deep learning, such as overfitting when trained via maximum likelihood estimation (MLE). Motivated by these problems, we explore an alternative parameter learning method particularly applicable to EiNets known as conditional composite log-likelihood estimation (CCLE). We propose three methods of implementing CCLE for EiNets: uniform random sampling, bisection sampling and grid sampling. In our experiments on MNIST and F-MNIST, we observe that CCLE training shows promise as a valid alternative density and generative training scheme for EiNets to MLE and for providing greater inpainting capabilities. However, a CCLE objective shows mixed results as a form of regularisation during training. Moreover, we note that these findings depend on the CCLE method used, the sizes of the patches chosen for conditional training and the information density of the images within a dataset.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Tom A. Lamb*)

Acknowledgements

I would like to thank my supervisor, Dr Antonio Vergari, for his patience, support and helpful advice throughout this dissertation.

Table of Contents

Chapter 1

Introduction

Probabilistic machine learning provides a flexible framework for rigorously reasoning about machine learning models. Using probabilistic inference, probabilistic models can quantify uncertainty, generate samples, and handle inference tasks like marginalisation or conditioning.

Modern deep generative models such as variational autoencoders (VAEs) [Kingma and Welling, 2014] have proven to be expressive models capable of modelling complex data distributions. The expressiveness of these models often comes at a significant price, however, that of intractable probabilistic inference. This often entails the use of approximate methods, which come with added issues such as optimisation difficulties.

A lack of tractable probabilistic inference for such models, going against the central motivation behind using probabilistic models in general, has led to a growing interest in tractable probabilistic models (TPMs). TPMs are models that allow for exact and efficient probabilistic inference of various kinds. A modern class of TPMs with a deep learning-like structure are probabilistic circuits (PCs) [Choi et al., 2020]. PCs build upon simpler distributions hierarchically using simple operations to create flexible and expressive probabilistic models for discriminate tasks and density estimation [Peharz et al., 2020c].

Despite their promise as effective TPMs, a significant drawback of the basic formulation of PCs is that their computational graph is sparse, which makes their training difficult and particularly limits their scalability to larger datasets Peharz et al. [2020a]. To address this problem, a variant of PCs known as Einsum networks (EiNets) was introduced, a vectorised version allowing many computations to be carried out in parallel [Peharz et al., 2020a]. GPUs, optimised for such computations, can then exploit this parallelisation. This allows EiNets to be trained more efficiently and allows these models to scale to larger and more complex datasets outside the reach of traditional PCs Peharz et al. [2020a].

EiNets, as density estimators or generative models, are often trained using (maximum) likelihood estimation (MLE) using expectation maximisation (EM) or using stochastic gradient descent (SGD) Peharz et al. [2020a], Yu et al. [2022]. These training methods often suffer from problems such as slow and difficult training and susceptibility to overfitting [Liu and Van den Broeck, 2021]. This poses a problem as increasing the capacity and size of a given network would lead to a more expressive model that could better capture complex data distributions. This leads to the primary research direction of this work: *are there alternative ways of training EiNets that avoid some of the aforementioned problems during training or provide additional benefits over standard MLE training via EM or SGD?*

An alternative method of parameter learning of probabilistic models is composite log-likelihood estimation (CLE) [Dillon and Lebanon, 2009], a particular variant of which is conditional composite log-likelihood estimation (CCLE). In CCLE, conditional likelihoods of patches of variables given the remaining variables within a probabilistic model are maximised [Asuncion et al., 2010]. Interestingly, as CCLE involves all variables in a probabilistic model, these conditionals can be decomposed into the difference of full and marginal likelihood terms. This structure resembles a regularised training objective, with the marginal likelihood term resembling a penalisation or regularisation term.

CCLE is often used for training probabilistic models where MLE training can be infeasible due to intractable likelihoods in high-dimensional settings Dillon and Lebanon [2009]. Moreover, CCLE also shares asymptotic properties similar to MLE [Dillon and Lebanon, 2009], making CCLE an attractive alternative to MLE in such scenarios.

However, a major drawback of CCLE is that it is often expensive to compute the conditional likelihoods required as the sizes of the patches of variables increase Asuncion et al. [2010]. However, this is not a problem for EiNets, which allow for efficient and tractable conditioning on arbitrarily sized subsets of random variables [Choi et al., 2020, Peharz et al., 2020a]. This leads us to the following questions and discussion points:

- Is CCLE a viable alternative for training EiNets for density estimation and generative tasks?

- Could incorporating CCLE training for an EiNet model lead to enhanced inpainting performance as the model is specifically trained to maximise the likelihood of regions conditioned on the remaining portions of images?

- Could a CCLE objective act as a form of regularisation during training?

Importantly, addressing these questions would be beneficial not only for training EiNets if CCLE training does show improvements over standard MLE training, but would also give useful insight and motivations for other probabilistic models more generally that allow conditional inference and where full likelihoods may be intractable.

Our contributions in this work are as follows:

- we introduce CCLE training for EiNet models for density estimation;

- we provide three novel methods of conducting CCLE training of EiNets, namely uniform random sampling, bisection sampling and grid sampling;

- we provide adapted metrics for evaluating CCLE-trained models;

- we provide a code base implementing the aforementioned methods and metrics;

- we demonstrate our training techniques on MNIST [LeCun et al., 1998] and Fashion-MNIST [Xiao et al., 2017b]. We find that some of our CCLE-trained models show promise as viable alternative training techniques for EiNets, which further exhibit improved inpainting capabilities. However, our observations indicate that a CCLE objectives show mixed success as regularisers during training.

Structure of this work. This work is divided into distinct chapters, beginning with Chapter 2 that provides detailed information on relevant background material and related work for this project. Specifically, we give detailed information on MLE, CLE, CCLE, TPMs, and EiNets. We conclude the chapter by motivating the research directions for this work.

This is followed by Chapter 3, where we outline our proposed training and evaluation methods for CCLE training. In particular, we introduce uniform random, bisection and grid sampling methods for CCLE training. Moreover, we introduce metrics for evaluating and comparing our models: test set CCLL, degree of overfitting and FID scores.

Finally, chapter 4 details our experimental setup and findings, and is concluded by explaining the limitations of our methodology and details directions for future research.

Chapter 2

Background

In this chapter, we detail the relevant background material and related work that forms the foundation of our research. From here onward, we use bold font capitalised $\mathbf{X} = (X_1, X_2, \cdots, X_k)$ to denote a multivariate random variable with k univariate components denoted X_i, all of which are considered jointly distributed. We use lower case $\boldsymbol{x}$ to denote a vector of values that $\mathbf{X}$ can take. Specifically, $p(\mathbf{X} = \boldsymbol{x})$ denotes the probability that $\mathbf{X}$ takes the value $\boldsymbol{x}$, that is, the joint probability mass/density function, which we often abbreviate to $p(\boldsymbol{x})$.

We adopt the convention that $\mathcal{D} = \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)}\}$ denotes an arbitrary data set with n data points, where, as above, we use bold to denote vectors. We let $x_j^{(i)}$ denote the jth component of the ith data point within $\mathcal{D}$. Finally, we permit using sets of indices within subscripts to select tuples of components from a particular data point. For example, for $\pi = \{1, 3\}$, we let $\boldsymbol{x}_\pi^{(i)} = \left(x_1^{(i)}, x_3^{(i)}\right)$ - note the bold font now - denote a vector consisting of the first and third components of the data point $\boldsymbol{x}^{(i)}$.

2.1 Fundamentals of Probabilistic Modelling

In probabilistic machine learning, we often view a dataset $\mathcal{D}$ as a sample drawn from some underlying distribution of data $p_*(\mathbf{X})$, that is $\mathcal{D} \sim p_*(\mathbf{X})$.

2.1.1 Density Estimation

Density estimation aims to approximate the underlying data generating distribution $p_*(\mathbf{X})$. We consider a statistical model, $p(\mathbf{X}; \boldsymbol{\theta})$, that is a parameterised family of probabilistic models with parameters $\boldsymbol{\theta}$. The goal of density estimation is then to look for parameter settings such that $p(\mathbf{X};\boldsymbol{\theta})$ well approximates the data generating distribution $p_*(\mathbf{X})$. Specifically, we hope and often assume that our statistical model is expressive and flexible enough such that there is some parameter setting $\boldsymbol{\theta}_*$ such that $p(\mathbf{X};\boldsymbol{\theta}_*) = p_*(\mathbf{X})$. [1] In such scenarios, we refer to $\boldsymbol{\theta}_*$ as an *optimal parameter setting*.

In real applications with complex data distributions, it is often not the case that an optimal parameter setting exists, meaning any derived probabilistic model is an approximation of $p_*(\boldsymbol{x})$ whose quality is highly dependent on the statistical model chosen and the complexity of the data that we are working with.

2.1.2 Maximum Likelihood Estimation, MLE

Maximum likelihood estimation (MLE) provides a point estimate, that is, a single choice of parameters $\boldsymbol{\theta}_{MLE}$, that aims for the optimal parameter setting $\boldsymbol{\theta}_*$ such that the resulting probabilistic model, $p(\mathbf{X};\boldsymbol{\theta}_{MLE})$, well approximates the data distribution to be modelled, $p_*(\mathbf{X})$.

A key assumption in MLE is that the data within our dataset $\mathcal{D} = \{\boldsymbol{x}^{(1)},\ldots,\boldsymbol{x}^{(n)}\}$ are independent and sampled from $p_*(\mathbf{X})$, that is the data points are independent and identically distributed (i.i.d). The parameter $\boldsymbol{\theta}_{MLE}$ is chosen in order to maximise the log-likelihood (LL) function of our statistical model for data $\mathcal{D}$, which is defined as

$$\ell_{LL}(\boldsymbol{\theta};\mathcal{D}) = \log p(\mathcal{D};\boldsymbol{\theta}) = \sum_{i=1}^{n} \log p(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \tag{2.1}$$

where the second equality follows from the i.i.d assumption of the data. The process of finding $\boldsymbol{\theta}_{MLE}$ is known as *maximum likelihood estimation* (MLE). It is worth knowing that $\boldsymbol{\theta}_{MLE}$ *is a statistic*, that is, a parameter estimate which is a function of the observed data within our data set.

We often maximise the LL, that is, the logarithm of the *likelihood* $p(\boldsymbol{\theta};\mathcal{D})$, as the LL is often easier to work with and optimise. Indeed, as the logarithm is a strictly increasing function, finding the parameter setting that maximises the LL will also give the parameter setting that maximises the likelihood $p(\boldsymbol{\theta};\mathcal{D})$.

By maximising the LL of our statistical model, we seek a parameter setting that aims to maximise the probability that our model produces data *like* the data contained in $\mathcal{D}$ in the hope that this will ensure that our model approximates the underlying data distribution well.

[1] Essentially, we a making an assumption that $p_*(\mathbf{X})$ belongs to a parametric family of distributions.

MLE has important asymptotic properties, making it a theoretically attractive technique.

Theorem 2.1.1. *Given a statistical model $p(\mathbf{X};\boldsymbol{\theta})$ with data sampled from the data distribution $p_*(\mathbf{X}) = p(\mathbf{X};\boldsymbol{\theta}_*)$, that is $\mathcal{D} = \{\boldsymbol{x}^{(1)},\ldots,\boldsymbol{x}^{(n)}\} \sim p(\mathbf{X};\boldsymbol{\theta}_*)$, then under a set of regularity conditions on the statistical model, $p(\mathbf{X};\boldsymbol{\theta})$, we have that*

$$\sqrt{n}\,(\boldsymbol{\theta}_{MLE} - \boldsymbol{\theta}_*) \xrightarrow{d} \mathcal{N}(\mathbf{0}, I(\boldsymbol{\theta}_*)^{-1}) \tag{2.2}$$

as $n \to \infty$, where $\mathcal{N}(\boldsymbol{\mu},\Sigma)$ denotes the Σ multivariate normal distribution with mean vector $\boldsymbol{\mu}$ and covariance Σ and $I(\boldsymbol{\theta}_*)$ is the Fisher information matrix and $\xrightarrow{d}$ denotes convergence in distribution.

Proof. See Knight [1999] for details on the necessary regularity required of a statistical model and the proof of the theorem. □

In particular, it can be shown that a MLE converges in probability to the optimal parameter setting $\boldsymbol{\theta}_*$ in the limit of infinite data, a property known as *consistency* [Knight, 1999].

There are two standard methods of approaching the computation of MLE in machine learning: stochastic gradient descent (SGD) and expectation maximisation (EM).

2.1.2.0.1 Computing MLE using SGD. In all but the most straightforward cases, MLEs cannot be computed in closed form and necessitate approximations (often in machine learning). One approach for finding approximate MLEs is through first-order gradient-based optimisation algorithms such as stochastic gradient descent (SGD), popular variants of which incorporate momentum Rumelhart et al. [1986] and adaptive scaling of gradients (e.g. the Adam optimiser [Kingma and Ba, 2014]) which can help in speeding up convergence.

2.1.2.0.2 Expectation Maximisation, EM. Sometimes one could be dealing with latent variable statistical models, $p(\mathbf{X},\mathbf{Z};\boldsymbol{\theta})$, where some of the random variables within the model, known as latent variables and denoted by $\mathbf{Z}$, are not observed in practice as data points (they explain some hidden structure beyond the observed data). Many seemingly non-latent models, such as mixture models, can be, and are commonly, written as latent variable models [Bishop and Nasrabadi, 2006].

An alternative method for MLE in latent variable models is expectation maximisation (EM). This is an iterative algorithm that alternatives between computing an expectation (E-step) and then maximising said expectation (M-step). Specifically, given parameters at iteration t, $\boldsymbol{\theta}_t$, the new updated parameters at iteration $t+1$, $\boldsymbol{\theta}_{t+1}$, are given by

$$\boldsymbol{\theta}_{t+1} = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{p(\mathbf{Z}|\boldsymbol{\theta}_t)}[\log p(\mathcal{D}, \mathbf{Z}; \boldsymbol{\theta})] \tag{2.3}$$

This can be thought of as maximising the *expected completed log-likelihood.* One important property of EM is that it yields a non-decreasing sequence of LLs [Dempster et al., 1977].

2.1.2.0.3 Contrasting EM and SGD for MLE. In both of the above cases, MLE suffers from significant issues. Indeed, models trained using MLE often suffer from overfitting, where a model begins to memorise its training set and fails to generalise well to new data [Ziegel, 2003]. Other common problems that MLE-trained models suffer from include difficulty handling rare events, sensitivity to outliers and getting stuck in local optima, which give sub-optimal solutions [Bishop and Nasrabadi, 2006, Jurafsky and Martin, 2019].

In EM, there may not be a closed form solution for the M-step computation, or this can be costly or even intractable to compute depending on the statistical model used, requiring alternative appraoches [Neal and Hinton, 1998]. Moreover, the fact that the algorithm yields a non-decreasing sequence of LLs means that EM is susceptible to getting stuck in local optima and is particularly sensitive to its initial conditions [Spitkovsky et al., 2013]. Moreover, EM often leads to slow training due to working with the entire dataset for each parameter update [Barber, 2012]. These issues can be improved through the use of stochastic EM (s-EM), which replaces the whole dataset in the updates above with mini-batches as in SGD, which can lead to faster training and the ability to escape local-optima [Chen et al., 2018].

SGD suffers from sensitivity to hyperparameters, such as its learning rate, which may require tuning or decaying during training [Goodfellow et al., 2016]. Moreover, the variance in its gradient updates can sometimes be large, leading to slow or unstable training [De and Goldstein, 2016]. Despite this, SGD is used ubiquitously within modern machine learning. Firstly, this is because some of the aforementioned issues can be solved by augmenting SGD with momentum and adaptive scaling, as previously mentioned (a testament to the flexibility of the method). Moreover, SGD is relatively simple as a method and is easily implemented and adaptable to many architectures with modern machine learning libraries implementing automatic differentiation such as PyTorch [Paszke et al., 2017]. In addition, it is relatively computationally cheap and scalable, and the stochasticity coming from batched gradient updates, like with s-EM, aids SGD in escaping local optima. The relative ease of implementation and adaptability makes SGD attractive even over EM methods which often require careful adaption to new models, architectures or problems Peharz et al. [2016].

2.1.3 Composite Likelihood Estimation, CLE

Often for high-dimensional problems, MLE can be costly or practically infeasible. For example, consider a statistical model of the form $p(\mathbf{X};\boldsymbol{\theta}) = \tilde{p}(\mathbf{X};\boldsymbol{\theta})/Z(\boldsymbol{\theta})$, where $Z(\boldsymbol{\theta})$ denotes the normalising partition function and $\tilde{p}(\mathbf{X};\boldsymbol{\theta})$ its associated unnormalised statistical model. In unnormalised statistical models such as Markov random fields (MRF) [Bishop and Nasrabadi, 2006], the computation of the partition function can be computationally costly or even intractable in high-dimensional spaces or in continuous cases due to having to sum or integrate over all possible values of the variables in the models. As a result, the LL defined in Equation (2.1) becomes infeasible to work with [Asuncion et al., 2010].

A popular alternative parameter estimation technique when working with intractable likelihoods is maximising a *pseudolikelihood* instead of the traditional likelihood function. In particular, the pseudolikelihood function is given by

$$p\ell(\boldsymbol{\theta};\mathcal{D}) = \sum_{i=1}^{n}\sum_{j=1}^{k} \log p\left(x_j^{(i)}|x_{\setminus j}^{(i)};\boldsymbol{\theta}\right), \tag{2.4}$$

where $\mathbf{X}$ is a k-dimensional jointly distributed multivariate random variable and $\setminus j = \{1,2,...,k\} \setminus \{j\}$. Here, j indexes a univariate component of $\mathbf{X}$. The process of finding the maximiser of Equation (2.4) is known as *pseudolikelihood estimation* (PLE). Here, the full conditional likelihood is broken down into maximising the conditional likelihood of each variable given the remaining variables in the model [Joe et al., 2012]. This reduces the cost of computing the partition function to summing or integrating over a single random variable due to the use of conditional terms [Dillon and Lebanon, 2009].

Like MLE, as discussed in Section 2.1.2, PLE is asymptotically consistent and converges in the limit, in probability, to the parameters of the data distribution, assuming that data distribution belongs to the same parametric family of models [Lindsay, 1988]. A downside over, say, MLE is that the asymptotic variance of MPLE is larger than for MLE, making it challenging to work with in practice [Lindsay, 1988, Asuncion et al., 2010].

Subsequently, composite LLs were introduced as a generalisation of the pseudolikelihood and to act as a form of ablation between the two extremes of the pseudolikelihood and the full likelihood [Asuncion et al., 2010].

Consider a finite set of pairs $\pi = \{(\pi_1, \tilde{\pi}_1), (\pi_2, \tilde{\pi}_2), \cdots, (\pi_m, \tilde{\pi}_m)\}$, where each pair, π_i and $\tilde{\pi}_i$, denote disjoint sets of indices that pick out certain components of a k-dimensional random variable $\mathbf{X}$. Specifically, we have $\pi_i, \tilde{\pi}_i \subset \{1, 2, \cdots, k\}$ and $\pi_i \cap \tilde{\pi}_i = \emptyset$. Then, a composite LL (CLL), denoted $c\ell(\boldsymbol{\theta}; \mathcal{D})$, is then given as

$$c\ell(\boldsymbol{\theta}; \mathcal{D}) = \sum_{i=1}^{n} \sum_{j=1}^{m} \log p\left(\boldsymbol{x}_{\pi_j}^{(i)} | \boldsymbol{x}_{\tilde{\pi}_j}^{(i)}; \boldsymbol{\theta}\right). \tag{2.5}$$

A special case of CLL is when for $\backslash\pi_i = \{1, 2, \cdots, k\} \setminus \pi$, we have that $\tilde{\pi}_i = \backslash\pi_i$ for all i, so that each pair in π forms a partition of $\mathbf{X}$. Here, the CLL is referred to as a *conditional composite log-likelihood* (CCLL) [Asuncion et al., 2010]. For CCLL, we can then simplify notation by defining $\pi = \{\pi_1, \pi_2, \cdots, \pi_m\}$, with the $\tilde{\pi}_j$ then being implicitly defined by their corresponding π_j.

A *composite likelihood estimator* (CLE) is then a parameter value that maximises Equation (2.5). We refer to the specific case when working with a CCLL as a *composite conditional likelihood estimator (CCLE)*, which is a particular case of a CLE. In the context of images, we can now think of π_j as patches within an image with CCLE, then maximising the likelihood of patches given the remaining portion of the images.

CLE, like PLE, notably shares similar asymptotic properties to MLE, such as asymptotic normality and consistency, which makes it a theoretically valid and attractive alternative to MLE for parameter estimation in scenarios with intractable likelihoods.

Theorem 2.1.2. *Given a statistical model $p(\mathbf{X}; \boldsymbol{\theta})$ with data sampled from the data distribution $p(\mathbf{X}; \boldsymbol{\theta}_*)$, that is $\mathcal{D} = \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)}\} \sim p(\mathbf{X}; \boldsymbol{\theta}_*)$, then under a set of regularity conditions on the statistical model $p(\mathbf{X}; \boldsymbol{\theta})$, we have that*

$$\sqrt{n}\,(\boldsymbol{\theta}_{CLE} - \boldsymbol{\theta}_*) \xrightarrow{d} \mathcal{N}(\mathbf{0}, G^{-1}(\boldsymbol{\theta}_*)), \tag{2.6}$$

as $n \to \infty$, where $\boldsymbol{\theta}_{CLE}$ denotes the CLE introduced in Section 2.1.3 and $G(\boldsymbol{\theta}_*)$ denotes the Godambe information matrix.

Proof. See Varin et al. [2011] for details on the necessary regularity required of a statistical model and the proof of the theorem. □

Moreover, the asymptotic variance of CLE lies between the two extremes of PLE and MLE, and it is believed, although not rigorously proven as far as we can tell, that increasing the sizes of π_j can reduce the variance of estimators and increase their accuracy [Asuncion et al., 2010].

CLE can be computed similarly to MLE through the use of SGD. However, it is worth noting that computing CLEs via SGD is often costly as the size of π increases, increasing the number of terms in Equation (2.5). Moreover, although one may be able to obtain more accurate estimates using larger patches, π_j, this often comes at the price of increased computational cost for many models since we would have to sum over or integrate over increasingly many variables [Asuncion et al., 2010].

2.1.4 Probabilistic Query Classes and Tractability

The main draw of using probabilistic models is that they allow us to more precisely reason and make decisions in the face of uncertainty and randomness, natural in real-world scenarios. Such decisions often involve performing some form of probabilistic inference.

There are many possible types of probabilistic inference that we may want to perform on a given probabilistic model. Indeed, we may want to: calculate a marginal distribution when some data is missing; calculate probabilities conditioned on/in light of known information; calculate the maximum a posteriori (MAP) estimates, that is, the value of a set of variable that makes an event most likely under the model conditioned on some separate known event; compute moments of a distribution, for example, to compute its expectation or variance; or calculate more complex quantities such as the Kullback-Leibler (KL) divergence between two distributions to quantify the similarity or difference between the two distribution. Choi et al. [2020] divide and classify some of the aforementioned inference types by introducing and concretely defining different important query classes of probabilistic inference.

Consider a k-dimensional multivariate probabilistic model $p(\mathbf{X})$. Firstly, Choi et al. [2020] consider the class of complete evidence queries, denoted by EVI, which refer to the computation of the probability of a complete event using the full joint distribution of a probabilistic model, i.e. the computation of $p(\mathbf{X} = \boldsymbol{x})$ for any $\boldsymbol{x}$.

Another important class introduced by Choi et al. [2020] is that of *marginal queries*, referred to as MAR. The MAR query class consists of all possible marginal evaluations resulting from the summing/integrating out of variables from full joint distributions.

A related query class to MAR is the *conditional query class*, referred to as CON, which is the set of all conditional events that we may want to calculate from our probabilistic model, e.g. the probability of some event given another known or assumed to have occurred event. Note that really CON queries can be considered part of the MAR class, as using the basic definition of conditional probability, we can write a conditional query as the ratio of two MAR queries.

The *maximum a posteriori* (MAP) query class is the final inference class important for our work. This class consists of inference queries that find the values of a set of random variables that maximise the probability of this set of variable values given known values of the remaining variables in a probabilistic model [Choi et al., 2020]. MAP queries are particularly useful for inpainting tasks, where we can fill in missing pixels in images conditioned on the pixel values of the image that we do know.

Having introduced various inference query classes, we come onto the vital notion of *tractability*. An inference query being tractable for a model generally refers to being able to exactly and efficiently compute the query. This is pivotal for practical applications of probabilistic models, such as in finance, where decisions must be made in real-time and with great precision. Choi et al. [2020] formally define the notion of tractability as follows:

Definition 2.1.1 (Tractable Inference for a Class of Inference Queries)**.** *We say that a statistical model, $p(\mathbf{X};\boldsymbol{\theta})$, is tractable for a class of inference queries $\mathcal{Q}$ if and only if any query within $\mathcal{Q}$ can be computed exactly, requiring no approximations, in polynomial time in the size of the statistical model, denoted $|p(\mathbf{X};\boldsymbol{\theta})|$, that is computed in time $O(poly(|p(\mathbf{X};\boldsymbol{\theta})|))$.*

Here, the size, $|p(\mathbf{X};\boldsymbol{\theta})|$, of the statistical model $p(\mathbf{X};\boldsymbol{\theta})$ can refer to different things in different contexts, but in essence is meant to tie in with the overall number of computations required for a given statistical model to compute an inference query. In graphical models, this can refer to the size of their factors.

With a precise definition of tractable inference, we can now detail the challenges many popular and successful deep machine learning models face, that of intractable probabilistic inference.

2.2 Tractable Probabilistic Models, TPMs

Deep generative models such as variational autoencoders (VAEs) [Kingma and Welling, 2014], diffusion models [Yang et al., 2022] and generative adversarial networks (GANs) Goodfellow et al. [2014] have shown great success in modern machine learning tasks, with these models, for example, being capable of modelling complex data distributions and generating high-quality image samples. A lot of the success of these models can be attributed to their incorporation of modern deep learning methods (for example, the use of neural networks as encoder and decoder networks that parameterise the posterior distributions of latent and observed variables within VAEs), which allow these models to scale up to millions if not billions of parameters.

The sheer size and complexity of these models often means that certain types of probabilistic inference remain intractable. Approximate methods such as Monte Carlo sampling (e.g. Markov chain Monte Carlo (MCMC)[Bardenet et al., 2017]) or variational approximations [Kingma and Welling, 2014] are then required to carry out approximate probabilistic inference. These approximate methods come with issues and difficulties. Indeed variational methods are known to be challenging to optimise effectively and can produce biased approximate solutions [Zhang et al., 2018], whilst MCMC suffers from slow convergence, being computationally intensive, can be difficult to optimise and often has difficulties extending to complex higher dimensional distributions [Gilks and Roberts, 1996].

Despite the flexibility and expressiveness of models such as VAEs and GANs, their dependence on approximation methods due to intractable inference is a significant blow. In particular, it goes against the fundamental draw of probabilistic models to allow for exact, accurate and rigorous reasoning for decision-making in uncertain scenarios. This, alongside the difficulties in approximate inference, has led to a growing interest in the literature towards tractable probabilistic models (TPMs) that allow for exact and efficient probabilistic inference, such as marginalisation or conditioning. Simple TPMs such as hidden Markov models (HMMs) Gales et al. [2008] and Kalman filters [Kalman, 1960], which were popular in the past, have fallen out of favour due to the rise, success and scalability of modern deep learning models.

This leads to the idea of a deep probabilistic model that can be both expressive and yet remain tractable for probabilistic inference. A popular class of such models include probabilistic circuits (PCs), which include a particular scalable collection of models known as einsum networks (EiNets).

2.3 Probabilistic Circuits, PCs

PCs are a general group of probabilistic models which includes previously well-studied models such as arithmetic circuits [Lowd and Domingos, 2012] and sum-product networks [Gens and Pedro, 2013]. Under certain conditions, PCs allow efficient and tractable probabilistic inference for various inference query types.

Concretely, a PC is a probabilistic model over a multivariate random variable $\mathbf{X}$ specified by a (DAG) structure containing three types of nodes: input leaf distribution nodes, sum nodes and product nodes. Input leaf distribution nodes, as the name implies, are distributions at the leaves of the DAG of a PC that act as the initial input distributions of the probabilistic model. Each leaf distribution is defined over some subset of the components of $\mathbf{X}$ and can come from a wide variety of probability distributions, such as the exponential family of distributions containing categorical, normal and gamma distributions as members [Bishop and Nasrabadi, 2006].

The second type of node in a PC is the product node. This node multiplies the distributions defined by the outputs of its input nodes. This node takes inspiration from fully factorised distributions which contain independent variables, e.g. $\mathbf{X} \perp\!\!\!\perp \boldsymbol{Y}$ iff $p(\mathbf{X}, \boldsymbol{Y}) = p(\mathbf{X})p(\boldsymbol{Y})$, where $\perp\!\!\!\perp$ denotes the independence of random variables.

The final type of node in a PC is a sum node. This computes a convex sum of the outputs of its input nodes (by convex sum, we refer to a linear combination of the distributions defined by its children with the weights used for the linear combination summing to one - effectively a weighted average). This node effectively functions as a mixture model.

Using these three types of nodes, a PC can be built up hierarchically using a DAG and a mixture of the aforementioned nodes, as shown in Figure 2.1, defining increasingly more complex distributions. A PC is really a statistical model with parameters given by the parameters of the input distributions and the mixture weights specified within sum layers.

Sum-product networks [Peharz et al., 2020c] are a previously well-studied class of probabilistic models that have shown success in both discriminative and generative tasks whilst also, most importantly, allowing tractable inference for certain classes of inference queries. Although sum-product networks precede the introduction of PCs by [Choi et al., 2020], they constitute a particular class of PC satisfying specific structural properties.

Sum-product networks are defined as a PC which is smooth [2] and decomposable, which refer the same concepts introduced in Section 2.3.1. Figure 2.1 gives a concrete example of a sum-product network for clarity. These two properties allow for tractable

[2] It is worth noting that completeness is another name commonly used in the literature to refer to the property of smoothness [Peharz et al., 2020c, 2015].

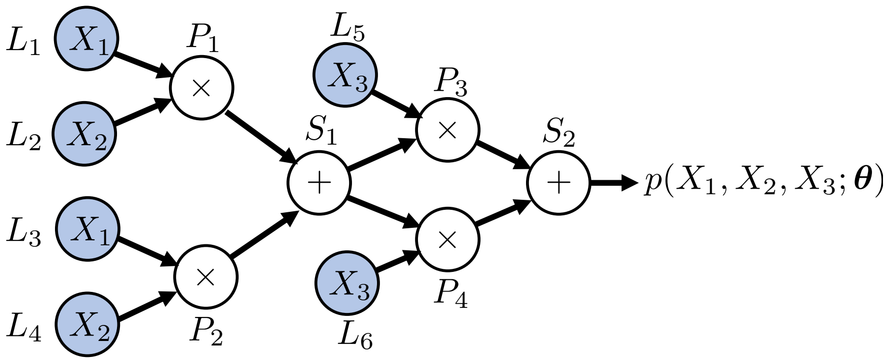

Figure 2.1: *Sum product network example* - an example of a DAG defining a smooth and decomposable PC, a sum-product networks, used for density estimation over three random variables $\mathbf{X} = \{X_1, X_2, X_3\}$ with leaf nodes L_i, product nodes P_i and sum nodes S_i. In this PC, we have, for example, the following scopes: $\phi(P_1) = \phi(P_2) = \phi(S_1) = \{X_1, X_2\}$, and $\phi(L_5) = X_3$.

inference for MAR queries, specifically in linear time in the size of the PC given tractable leaf distributions [Choi et al., 2020]. Moreover, smoothness and decomposability precisely characterise the set of PCs that allow for MAR queries (see proposition 17 and theorem 19 in [Choi et al., 2020] for details). [3] A simple corollary of this fact is that CON queries can also be computed in linear time in the size of the sum-product network as they can be decomposed into the ratios of MAR queries (corollary 18 in [Choi et al., 2020]).

In density estimation tasks, a rooted DAG is used with a single root node with no parents, often assumed to be a sum node, representing the joint distribution over $\mathbf{X}$ (the scope of the root node is $\mathbf{X}$). PCs for density estimation tasks can then be trained using MLE. This can be done using SGD to optimise the trainable weights of a PC. However, it is also common to use EM to perform MLE. Indeed, as highlighted by Poon and Domingos [2011], the sum nodes within PCs are essentially mixture models, where it is common to think of the mixture weights as being associated with a collection of latent variables. Indeed, Peharz et al. [2016] carefully defined the latent variable model associated with smooth and decomposable PCs and derived the EM algorithm to perform MLE for such networks. Another benefit of reformulating PCs as latent variable models is that it allows for sampling via ancestral sampling [Peharz et al., 2020a], allowing PCs also to be generative.

[3] Structural properties required for tractable inference other queries can be found in [Choi et al., 2020].

2.3.1 Probabilistic Circuits and Einsum Networks

Despite their success in density estimation and discriminative tasks whilst allowing for tractable probabilistic inference [Peharz et al., 2020c], PCs struggle to scale to larger models and to more complicated datasets due to their sparse computational graphs [Peharz et al., 2020a]. As a remedy for the sparsity problem, Peharz et al. [2020a] proposed a vectorised variant of PCs named Einsum networks (EiNets) which have a more dense and efficient computational structure that helps these models scale to datasets out of reach of traditional implementations of PCs. Vectorised implementations of PCs have been introduced prior to EiNets. [Trapp et al., 2019, Dennis and Ventura, 2012, Peharz et al., 2013, 2020c]. However, EiNets, by introducing the *einsum operation*, allow for more efficient vectorised models with faster training times and more efficient memory usage [Peharz et al., 2020a].

We consider an EiNet to be a probabilistic model over a random variable $\mathbf{X}$ that is defined by a DAG consisting of three types of nodes: leaf, sum and product nodes, which we denote by L, S and P respectively. The inputs and outputs of each node within an EiNet's DAG encode multiple probability distributions as vectors rather than as single distributions as in traditional PCs. We, therefore, refer to the outputs of leaf, sum, and product nodes by $\boldsymbol{L}$, $\boldsymbol{S}$ and $\boldsymbol{P}$.

Peharz et al. [2020a] impose three assumptions on EiNets: that each node in the DAG of an EiNet contains the same number of distributions and denote this quantity by K; that in the DAG of an EiNet, we have an alternating structure of product and sum nodes [4]; and that for density estimation, an EiNet has a single output that is a sum node [5].

An EiNet's leaf nodes' outputs consist of vectors of distributions defined over a subset of $\mathbf{X}$, which act as the input distributions of the EiNet. Leaf node distributions are often assumed to come from the flexible and broad family of exponential distributions whose members include the gamma, categorical and normal distributions [Peharz et al., 2020a].

A product node in an EiNet takes vectors of input distributions and computes the outer product of these vectors, that is, the product of each combination of distributions with a single distribution selected from the outputs of each input node. Specifically, for a product node P, we have that $\boldsymbol{P} = \bigotimes_{C \in in(P)} \boldsymbol{C}$. In particular, denoting $in(P) =$

[4] Technically this condition is broken through the introduction of mixture layers.

[5] These assumptions are not strictly necessary but, say, the assumption on K begin constant for each node simplifies the description of EiNets.

$\{C_1, C_2, \cdots, C_{|in(P)|}\}$ as the input nodes to P, the components of tensor $\boldsymbol{P}$ are given by $P_{i_1,\cdots,i_{in(P)}} = (C_1)_{i_1} \cdot (C_2)_{i_2} \cdots (C_{|in(P)|})_{i_{|in(P)|}}$, for $i_j \in \{1, \cdots K\}$. As a result, product nodes can be considered to represent fully factored distributions of independent random variables. Note that the number of components in the output of a product node is given by $K^{|in(P)|}$, which grows exponentially in the number of input nodes of $\boldsymbol{P}$. Hence, Peharz et al. [2020a] limit the number of input nodes for each product node to two to limit the computational cost of product nodes within EiNets. Under this assumption, we see that the output of each product node is a $K \times K$ matrix of probability distributions.

A sum node, S, assuming alternating sum and product nodes, takes the $K \times K$ matrix output of its input product node P and computes convex sums of the elements of $\boldsymbol{P}$. Specifically, this can be written as $\boldsymbol{S} = W vec(\boldsymbol{P})$, where $vec(\boldsymbol{P})$ unrolls the $K \times K$ input to $\boldsymbol{S}$ (coming from $\boldsymbol{P}$) into a K^2-dimensional vector which is then multiplied by a $K \times K^2$ weight matrix W, giving a K-dimensional output. Note that in order to compute convex sums of the elements of $\boldsymbol{P}$ using W, we have that $\sum_j W_{i,j} = 1$ for all i, i.e. the rows of W must sum to one.

Peharz et al. [2020a] then combine the computations performed by the sum and product nodes into a single operation they term the *einsum operation*, which we denote $\boldsymbol{E}$. This takes two vectors $\boldsymbol{C}$ and $\boldsymbol{C}'$ as inputs (i.e. really the outputs of $in(P)$ of a product node P as above) and computes an output whose components are given via the following inner products: $E_i = C_j W_{ijk} C'_k$, where $i, j, k \in \{1, \ldots K\}$ and where we have implicit sums over repeated indices following the Einstein summation convention [Arfken and Weber, 1972]. Einsum operations can then be computed in parallel within an *einsum layer* which is computed as $E_{li} = C_{lj} W_{lijk} C'_{lk}$, for $i, j, k \in \{1, \ldots K\}$ and $l \in \{1, \ldots L\}$, where L denotes the number of einsum operations in the einsum layer.

The final layer introduced by the authors is a *mixing layer*. As we have seen in the einsum layer, each einsum operation is effectively represented as a product of the outputs of two input nodes, whose output is then fed into a single sum node. This, therefore, assumes that each sum node only has a single input node. For EiNets to be flexible to working with sum nodes with multiple input nodes whilst maintaining computational efficiency, the authors introduce what they term a mixing layer. Here, consider the output of a sum node $\boldsymbol{S}$ with several product nodes as inputs $in(S) = \{P_1, \ldots, P_{|in(S)|}\}$. Then for each input node P_i, we introduce a sum node S_i for which P_i is the single input (creating an einsum layer). To create a single output sum node, $\tilde{S}$, a weighted combination of $\boldsymbol{S}_i$ is taken as $\tilde{\boldsymbol{S}} = \sum_{i=1}^{in(S)} w_i \boldsymbol{S}_i$, for some trainable weights w_i. This is effectively a mixture of the outputs of the introduced sum nodes. Figure 2.2 gives a visual description of the process of introducing a mixing layer.

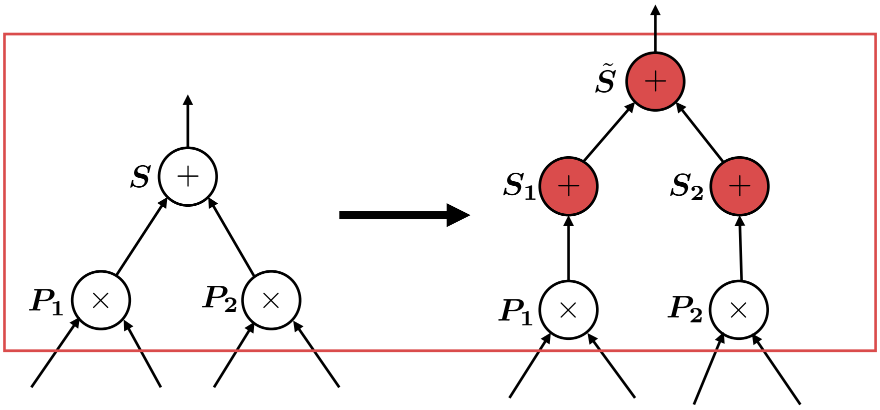

Figure 2.2: *Einsum operation and mixing layer* - figure showing an einsum operation and mixing layer within the DAG of an EiNet. In particular, the red box highlights the addition of a mixing layer, transforming a sum node with multiple product node children to multiple sum nodes with single products as children which are subsequently mixed by a mixing layer sum node. Added nodes are highlighted in red.

Figure 2.2 in the appendix gives a visual description of the process of introducing a mixing layer.

EiNets are then built up using einsum and mixing layers in a hierarchical fashion, layer by layer, creating increasingly more expressive distributions. Examples of DAGs for simple EiNets can be found in Figure 2.3 and in Figure 2.1 in the appendix. Finally, we note that EiNets are statistical models with learnable parameters coming from the parameters of the input leaf distributions and the mixture weights in the sum nodes and mixing layers.

An essential property of EiNets is that *like PCs and under the same structural constraints on their DAGs allow for exact and efficient probabilistic inference of various inference queries* assuming tractable leaf distributions. To discuss specific properties relevant to this work, we first introduce the concept of the scope of a node within an EiNet's DAG.

Definition 2.3.1 (Scope)**.** *Let M be a EiNet defined over a k-dimensional random variable $\mathbf{X}$ with an associated DAG, $\mathcal{G}$. The scope of $N \in V(\mathcal{G})$, denoted $\phi(N)$, is then defined as*

$$\phi(N) = \bigcup_{P \in in(N)} \phi(P), \tag{2.7}$$

where $V(\mathcal{G})$ denotes the vertices of $\mathcal{G}$ and $in(N)$ the input nodes of N. If N is an input *leaf node, $\phi(N)$ is defined as the set of random variables, a subset of* $\mathbf{X}$*, that the leaf node's distribution is defined over.*

Using the definition of the scope of a node, two important structural proprieties for the efficient computation of MAR and thus CON queries are *smoothness* and *decomposability* [Choi et al., 2020]. An EiNet is smooth if the inputs of each sum node share identical scope - this effectively means that each distribution in a sum node represents a valid mixture of distributions. Decomposability refers to the inputs of each product node having disjoint scopes - effectively defining factorised distributions.

An EiNet that is smooth and decomposable allows for the efficient computation of MAR and CON queries [Peharz et al., 2020a, Choi et al., 2020]. Traditional PCs that are smooth and decomposable are known as sum-product networks (SPNs) [Peharz et al., 2020c]. SPNs can compute MAR and CON queries in linear time in the number of edges in an SPNs DAG. This isn't quite true in EiNets due to having to carry out vectorised computations, so here, one must also account for K.

Nevertheless, exact MAR and CON queries can be computed efficiently, which is impressive given the size that EiNet models can grow to. Indeed all of the introduced operations within an EiNet, specifically the einsum layer, can be computed in parallel, allowing EiNet training and computations to take advantage of GPUs, making training quick and efficient. Indeed Peharz et al. [2020a] showed that training speeds and storage requirements for EiNets can be up to one or even two orders of magnitude better than previous PC implementations such as LibSPN [Pronobis et al., 2017].

Finally, Peharz et al. [2020a] show that EiNets can be trained and show good performance on larger and more complex datasets than are usually discussed in the prior PC literature such MNIST [LeCun et al., 1998], CelebA [Liu et al., 2018] and SVHN [Netzer et al., 2011]. The authors show that EiNets scale well to these complex high-dimensional image datasets while maintaining exact, efficient and tractable probabilistic inference even for large models with millions of parameters.

2.3.2 PC and EiNet Structure

For PCs and EiNets, there is great freedom in formulating their DAG structure beyond satisfying the basic requirements for tractable inference for certain query classes. Indeed this freedom has led to research into methods of best learning or formulating the DAG structures for PCs. These include random binary trees which recursively divide the scopes of nodes randomly into equal halves [Peharz et al., 2020c] and clustering

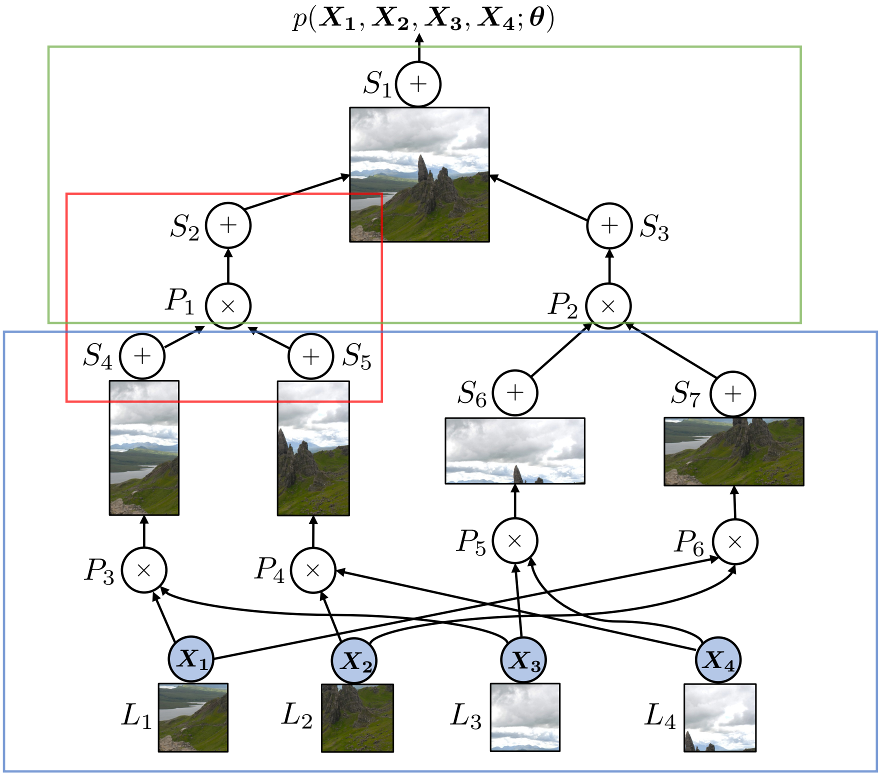

Figure 2.3: *PD-structure* - an example of a PD-structure used to define the feed-forward DAG structure of an EiNet. The image is divided into halves horizontally and vertically and then into quarters. The subsequent partitions (cuts) and regions (sub-images) are then populated with product P_i, leaf L_i and sum nodes S_i. The red box shows an einsum operation, the blue box shows an einsum layer, and the green box shows the introduction of a mixing layer to deal with the two product nodes introduced from two separate cuts of the root image.

processes including LearnSPN [Gens and Pedro, 2013] and its various augmentations and successors [Vergari et al., 2015, Rahman and Gogate, 2016, Di Mauro et al., 2018].

One particular approach for creating DAGs for PCs, and by extension EiNets, specifically when working with image data, was introduced by Poon and Domingos [2011], which we refer to as PD-structure. PD-structures are suitable for image data as they encode relative locations of regions within images, are very flexible, can work with images of any size, and provide a scalable way of producing EiNets. Furthermore, the ability to encode relative positions within a PC's DAG structure helps to provide a useful inductive bias for these models when dealing with images.

Following the notation used by Peharz et al. [2020a], a PD structure is formed from an image by applying axis-aligned cuts which are displaced by a specific step size, Δ, recursively dividing an image into smaller and smaller rectangles. Note that Δ can also be a collection of step sizes such as $\Delta = \{7,28\}$. For collections of step sizes, each rectangle is divided using the largest step size within Δ that fits within the largest edge of the rectangle. For example, for a 2×8 rectangle with $\Delta = \{5,15\}$, a step size of 5 would be used to cut the image along the edge of length 8, creating 2×5 and 2×3 rectangles. The recursive splitting stops when no step size in Δ can be used to split any rectangle.

The cutting process can be used to define a bipartite graph (really a region graph [Peharz et al., 2020c]) with alternating nodes corresponding to cuts (partitions) and nodes corresponding to the subsequent sub-images created (regions). We can then construct an EiNet by populating each partition with a product node and all but the final leaf regions, which become leaf nodes, with sum nodes. This then creates an alternating graph of sum and product nodes. We can then group the sum and product nodes into einsum operations and subsequently einsum layers using a topological ordering of the nodes within the DAG (see algorithm 1 in Peharz et al. [2020a]).

Care must be taken for sub-images that can be divided by multiple cuts. This would result in the sum node in such a sub-image's corresponding region having more than one product node as children, which is against the assumptions introduced in Section 2.3.1. To get around this, for such scenarios, we introduce a mixing layer, as discussed previously. Figure 2.3 depicts how a PD-structure is formed and used to create a DAG for an EiNet, including how mixing layers are introduced.

Importantly, we note that *using PD-structures creates smooth and decomposable PCs and EiNets.* Moreover, we see that dividing an image with a single horizontal cut and then a vertical cut or by a single vertical cut and then a horizontal cut would lead to the same resulting sub-image region. Importantly, we reuse the nodes in these repeated sub-regions within an EiNet's DAG, which leads to parameter sharing and provides a further good inductive bias to the subsequent EiNet model and can be thought of as reminiscent of convolutions within CNNs [Vergari et al., 2022].

2.3.3 Training PCs and EiNets

Both standard PCs and EiNets can be trained using SGD or by thinking of these models as latent variables models in order to use EM, specifically s-EM. Indeed, Peharz et al. [2020a] show that for EiNets, the use of modern automatic differentiation methods can lead to a straightforward and efficient implementation of the s-EM algorithm, adapted from previous EM updates derived for PCs Peharz et al. [2016]. Moreover, formulating EiNets and PCs as latent variable models allows for ancestral sampling [Peharz et al., 2020a], allowing PCs to become generative models capable of generating new samples.

On more straightforward binarised datasets commonly used in the PC literature [Peharz et al., 2016, 2020c, Gens and Pedro, 2013], Peharz et al. [2020a] show that s-EM generally leads to better generalisation performance than SGD or its popular variant the Adam optimiser. However, the authors do not complete the comparison between training methods on larger and more complex datasets.

One major downside of MLE training is that it often leads to overfitting or difficulty in training large and complex models (c.f. Section 2.1.2). PCs and EiNets are no different, with the problem of overfitting and difficulties with MLE training, alongside proposed solutions such as entropy regularisation or EM with bagging, having been discussed within the PC literature [Liang et al., 2017, Liu and Van den Broeck, 2021, Vergari et al., 2015]. The problem with model complexity and overfitting becomes a more significant worry for EiNets which are readily scalable to models with millions, if not billions, of parameters.

2.4 Motivation and Research Problems

As noted in Section 2.1.2 and Section 2.3.3, EiNets and large probabilistic models, in general, suffer from problems such as overfitting when trained with MLE. This leads us to question if there are viable alternative parameter learning schemes for these models that circumvent some of these issues whilst still producing good density estimators and generative models.

Conditional training schemes, including CLE, as introduced in section 3.2, are particularly suitable for smooth and decomposable EiNets, which allow for the efficient computation of CON queries over arbitrarily sized subsets of variables. EiNets, therefore, bypass the increased computational cost of using larger patch sizes π_j for CLE training, as discussed in Section 2.1.3. In addition, EiNets are well adapted for training on more complex and high-dimensional datasets containing image data whilst maintaining the required tractable inference routines for CLE training (c.f. Section 2.3.1). These points, alongside the similar theoretical proprieties to MLE, make CLE a theoretically justifiable alternative parameter learning scheme to consider for training EiNets.

Let us focus on CCLE, a specific form of CLE, where each conditional in the CCLL involves all of the components of a random variable $\mathbf{X}$ (c.f. Section 2.1.3). Consider a conditional log probability of a patch of pixels π within an image, forming a term in the CCLL. We can write the resulting log-conditional using the definition of conditional probability as $\log p(\boldsymbol{x}_\pi|\boldsymbol{x}_{\setminus\pi}) = \log p(\boldsymbol{x}) - \log p(\boldsymbol{x}_{\setminus\pi})$.[6] We can see that by splitting a random variable $\mathbf{X}$ into disjoint sets consisting of a patch π and its complement, we recover the full joint distribution and a marginal distribution involving the variables in the complement of the patch. The marginal term is interestingly reminiscent of penalisation or regularisation terms common within machine learning [Bishop and Nasrabadi, 2006, Liu and Van den Broeck, 2021].

Moreover, if we consider using CCLE during training, our model would be trained to maximise the likelihood of patches of images conditioned on the remaining portions of the images, essentially learning to model local image features conditionally. Although we would not expect such models to produce better whole image samples or overall densities, as unlike MLE, they are not directly trained for this, training models to model local regions in images should allow them to better perform tasks such as inpainting for dealing with missing patches of data, an important task in computer vision [Peharz et al., 2020a, Yasuda et al., 2005, Lugmayr et al., 2022].

These motivating points lead us to consider three specific research questions which we aim to investigate within this work:

Q 1: *Is CCLE a viable alternative method to MLE for training EiNet density models in terms of generalisation performance and generative quality?*

Q 2: *Do CCLE objectives act as a form of regularisation during training, helping EiNet models to be less prone to overfitting?*

Q 3: *Does CCLE training lead to greater inpainting quality over MLE training?*

As far as we can tell within the PC literature, looking solely into CCLE training for PCs has yet to be investigated, placing this as novel research into new methods for training PCs, particularly EiNets. Moreover, addressing the above research questions will provide information and hint at the utility of more broadly incorporating CCLE training for other probabilistic models that allow for tractable CCLL, where full LL estimation may be difficult or infeasible.

[6] Note that if we considered CLE, then we would not recover the full likelihood here, but would recover two marginals distributions instead (c.f. eq. (2.5)). Hence, to connect with MLE, we focus on CCLE training in this work.).

Chapter 3

Methodology

This chapter outlines the methodology for our proposed investigations into using CCLE as an alternative method to MLE for training EiNets, as motivated in Section 2.4. We focus on image datasets following the scalability of EiNets demonstrated by Peharz et al. [2020a]. As we will deal with image data, we also construct EiNet models using PD-structures which provide a suitable inductive bias for images as discussed in Section 2.3.2.

Following the notation conventions introduced in Chapter 2, we consider a dataset $\mathcal{D} = \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)}\}$ and further consider mini-batches of size n_{mini}, randomly sampled from $\mathcal{D}$, which at training step t we denote as $\mathcal{D}_t^{\text{mini}} = \{\boldsymbol{x}^{(i_1)}, \boldsymbol{x}^{(i_2)}, \cdots, \boldsymbol{x}^{(i_{n_{\text{mini}}})}\} \subset \mathcal{D}$, for a random subset of indices $\{i_1, \cdots i_{n_{\text{mini}}}\} \subset \{1, 2, \ldots n\}$. Finally, for coordinates in images, we consider $(x, y) \in \mathbb{N}_0^2$ to be the pixel in the xth row and yth column of the image with the origin, $(0, 0)$ (making use of zero indexing), located in the top left corner.

3.1 MLE Baselines

As discussed in section 2.3.3 there are two main methods for training EiNets via MLE, SGD and EM. We present details of both methods below that we will use within our experiments, introducing MLE for SGD within the context of empirical risk minimisation.

3.1.0.0.1 MLE via SGD. In the domain of machine learning, particularly for Maximum Likelihood Estimation (MLE) training via Stochastic Gradient Descent (SGD), we often adopt the perspective of Empirical Risk Minimisation (ERM) [Vapnik, 1991].

This paradigm focuses on minimising the expected risk over the true data generating distribution. Specifically, for a statistical model $p(\boldsymbol{X};\boldsymbol{\theta})$, the risk we aim to minimise is expressed as the expectation of the negative log-likelihood over the underlying data distribution:

$$\mathcal{R}(\boldsymbol{\theta}) = \mathbb{E}_{p*(\boldsymbol{X})}[-\log p(\boldsymbol{X};\boldsymbol{\theta})], \tag{3.1}$$

where $\mathcal{R}(\theta)$ denotes the expected risk. In practice, this expectation is intractable, as we generally do not know the underlying data distribution, as discussed in Section 2.1.1.

To get around this issue, we draw unbiased MC samples from a training set which we assume is drawn i.i.d from the underlying distribution of data. At each training step t, we draw a mini-batch $\mathcal{D}_t^{\text{mini}}$ and update the model parameters $\boldsymbol{\theta}$ by minimising the empirical risk loss computed over this mini-batch:

$$\mathcal{L}_{ERM}(\mathcal{D}_t^{\text{mini}};\boldsymbol{\theta}) = -\frac{1}{N_{\text{mini}}}\sum_{i=1}^{N_{\text{mini}}} \log p\left(\boldsymbol{x}^{(i)};\boldsymbol{\theta}\right). \tag{3.2}$$

This mini-batch loss serves as an unbiased estimator of the expected risk $\mathcal{R}(\boldsymbol{\theta})$, ensuring that each gradient update step is guided towards minimising the true expected risk across the entire data distribution. The use of mini-batches in SGD provides a balance between computational efficiency and the accuracy of the risk estimation.

This minimisation of empirical loss over the mini-batch aligns directly with maximising the likelihood of the model parameters $\boldsymbol{\theta}$ given the observed data in our training set. Therefore, in the context of ERM, the act of minimising risk is equivalent to maximising the likelihood training data under our model. Hence, we usually use the notation $\mathcal{L}_{MLE}$ to denote the loss in eq. (3.2).

3.1.0.0.2 MLE via EM. For EiNets specifically, the class of density models we focus on in this work, the EM updates for MLE training using leaf distributions from the exponential family of distributions were derived by Peharz et al. [2020a]. Due to the connection with mixture models, they, too, have parameter updates reminiscent of weighted averages Barber [2012]. Specific details can be found in [Peharz et al., 2020a], with a corresponding implementation found at [Peharz et al., 2020b], which we utilise and adapt for our experiments.

3.2 CCLE Training

We have introduced the general form of a CCLE in Section 2.1.3, which we would like to train using SGD. In a similar way to the method used to train diffusion models across multiply denoising time steps [Ho et al., 2020], we formulate the CCLE loss at training time t as

$$\mathcal{L}_{CCLL}(\mathcal{D}_t^{\text{mini}};\boldsymbol{\theta}) = \mathbb{E}_{t\sim U(|\pi|),\,\boldsymbol{x}\sim p_*(\boldsymbol{X})}\left[-\log p\left(\boldsymbol{x}_{\pi_t}|\boldsymbol{x}_{\backslash\pi_t};\boldsymbol{\theta}\right)\right], \tag{3.3}$$

where $U(|\pi|)$ denotes the discrete uniform distribution over the possible patches that could be selected (c.f. Section 2.1.3). Here, we have modelled the loss introduced in section Equation (2.5) by writing the sum over patches as an expectation over a uniform distribution over the patches π and ignoring the necessary scaling over the size of π which will not effect minimisation.

Again this is intractable as we do not know the true underlying data distribution and so we once again compute an an empirical loss:

$$\mathcal{L}_{CCLL}(\mathcal{D}_t^{\text{mini}};\boldsymbol{\theta}) = -\frac{1}{N_{mini}}\sum_{j=1}^{N_{\text{mini}}}\log p\left(\boldsymbol{x}_{\pi_t}^{(i_j)}|\boldsymbol{x}_{\backslash\pi_t}^{(i_j)};\boldsymbol{\theta}\right), \tag{3.4}$$

where $\backslash\pi_t = \{1,2,\cdots,k\}\setminus\pi_t$ for k-dimensional data and π_t is a uniformly randomly chosen patch from a predefined collection of patches π.

The proposed method affords a particular degree of freedom in that the collection of possible patches π and the patches sampled at each time step $\pi_t \in \pi$ can be constructed and chosen in many different ways. This then reduces our investigation of CCLE methods to image patch sampling design. We list several approaches that we investigate in the sections to follow.

3.2.1 Uniform Random Sampling

We term the first sampling method that we investigate *uniform random sampling*. Using this method, we consider rectangular patches of size $p \times q$ (p pixels in height and q pixels in width) and consider the set of all possible patches of this size within an image, defining π. At training step t, we randomly sample a patch uniformly from this collection which then defines the patch π_t to be used within the loss defined in Equation (3.4).

Specifically, we uniformly select the top left-hand corner of the patch, making sure only to consider top left-hand corner locations so that the resulting patch would remain

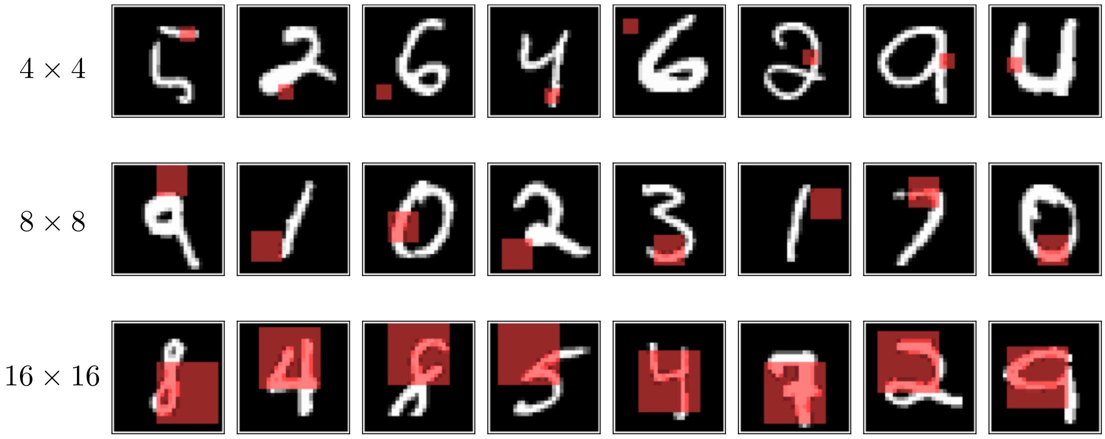

Figure 3.1: *Uniform random sampling patch examples* - patch samples, π_t, coloured in red, sampled from MNIST images using random sampling introduced in Section 3.2.1 for patches of size 4×4, 8×8 and 16×16.

.

within the boundaries of the image. We subsequently select the rest of the pixels from the image to form a size $p \times q$ patch relative to this chosen location. For clarity, we include examples of sampled patches of various sizes using uniform random sampling in Figure 3.1.

This is a very simple method to begin investigating how well CCLE performs without imposing any inductive bias in choosing patches. Moreover, this simplistic sampling method will act as a good baseline for comparison against alternative and more complicated sampling methods.

3.2.2 Grid Sampling

We note that the samples chosen are single rectangles of contiguous pixels within images for random uniform sampling. Considering that we are focusing on EiNets formed using a PD-structure, pixels or patches within an image that are close to each other will also be 'close' to each other within the DAG of the EiNet - that is, PD-structures already take into account the locality of image regions. This is illustrated in Figure 2.3 where we see that L_1 and L_2 could be considered close to each other as they can be produced from a single region split at S_6. Whereas, L_1 ad L_4 aren't as close as a single split of a larger region cannot produce them both. Alternatively, there is a shorter path between L_1 and L_2 compared to L_1 and L_4 within the DAG. Therefore, using contiguous patches may not add any benefits beyond the inductive bias already provided by the PD-structure of an EiNets. This leads us to consider investigating non-contiguous patches, which allows

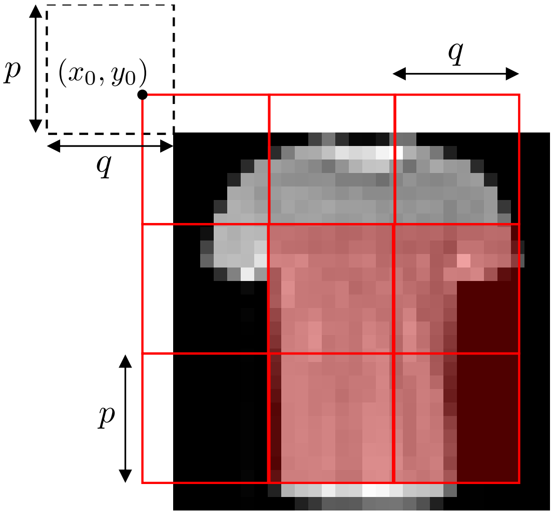

Figure 3.2: *Grid sampling formation* - example showing how a grid of patches to be sampled from is created when using grid sampling, here for image from the F-MNIST data set. The patches shaded in red show the valid patches for this iteration that each can be sampled with grid probability γ.

for patches that are made up of pieces of the image that are not local to one another within the image or the EiNet's associated PD-structure.

Along these lines, we investigate what we term as *grid sampling*, which combines elements of the uniform random sampling method proposed above whilst also allowing for sampling non-contiguous patches.

Precisely, we form a grid of potential patches of size $p \times q$ to sample from for each batch at training step t. To do this, we first randomly sample a point with coordinates $(X_0 = x_0, Y_0 = y_0)$ as $x_0 \sim U(-p+1, -p+2, \cdots, 0)$ and $y_0 \sim U(-q+1, -q+2, \cdots 0)$ which will become the top left-hand corner of the grid. By sampling the top left-hand corner like this, we can form a grid that is randomly translated for each batch.

From here, we build a grid of patches equidistant in the vertical and horizontal direction every p and q pixels, respectively, starting at the point (X_0, Y_0). Note that we discard any patches that are not entirely within the image. From this formed grid of valid patches, we loop through and select each patch within the grid with probability γ, which we refer to as the *grid probability*. We then form π_t as the concatenation of the randomly selected patches from the shifted grid. In this method, we can think of π as the collection of patches sampled from each possible shifted grid we could form over an image using grid patch sizes of $p \times q$. For clarity, we include a visual demonstration of how the grid of patches is generated at each training iteration in Figure 3.2 and include

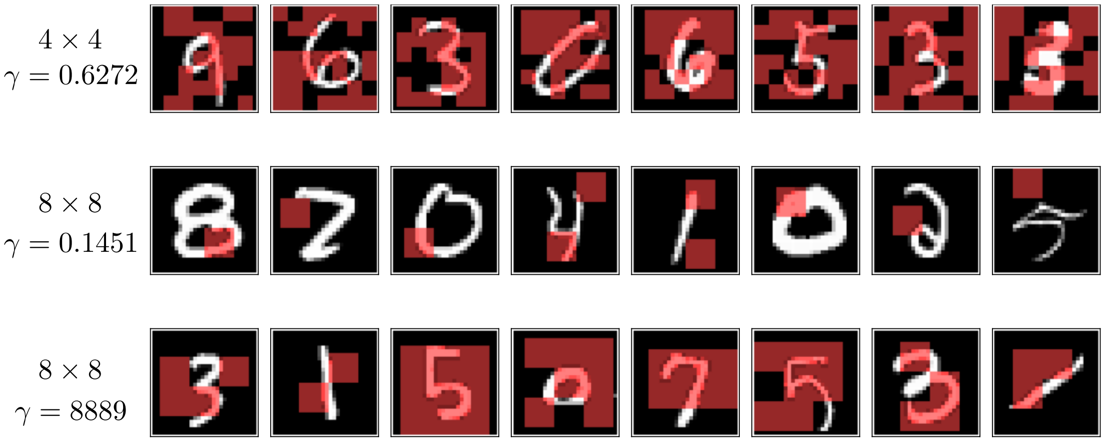

Figure 3.3: *Grid sampling patch examples* - patch samples, π_t, coloured in red, sampled from MNIST images using grid patch sampling introduced in Section 3.2.2. Here, we provide samples for patches of sizes 4×4 with $\gamma = 0.6272$ and 8×8 with $\gamma = 0.8889$ which sample on average half of the pixels within the images, and 8×8 with $\gamma = 0.1451$ which samples on average one 8×8 patch.

samples generated using grid sampling using different patch sizes and γs in Figure 3.3.

Based on this process, we can calculate the expected number of patches sampled for a given batch. Denote the number of patches sampled at a given training iteration t by N_t. Then given a sampled grid starting position, $(X_0, Y_0) = (x_0, y_0)$ for an images of dimensions $h \times w$, we can calculate the expected number of sampled patches given this grid starting position and a grid with patch sizes of size $p \times q$ as:

$$\mathbb{E}[N_t | X_0 = x_0, Y_0 = y_0] = \begin{cases} \gamma \lfloor (h - x_0 - p)/p \rfloor \lfloor (w - y_0 - q)/q \rfloor & \text{if } x_0 < 0, y_0 < 0, \\ \gamma \lfloor (h - x_0 - p)/p \rfloor \lfloor w/q \rfloor & \text{if } x_0 < 0, y_0 = 0, \\ \gamma \lfloor h/p \rfloor \lfloor (w - y_0 - q)/q \rfloor & \text{if } x_0 = 0, y_0 < 0, \\ \gamma \lfloor h/p \rfloor \lfloor w/q \rfloor & \text{else.} \end{cases} \tag{3.5}$$

where $\lfloor \cdot \rfloor$ denotes the floor function. Using the total law of expectation and that $p(X_0 = x_0, Y_0 = y_0) = pq$, we can then calculate the expected number of patches sampled at a given iteration t using Equation (3.5) as

$$\mathbb{E}[N_t] = \frac{1}{pq} \sum_{x_0=-p+1}^{0} \sum_{y_0=-q+1}^{0} \mathbb{E}[N_t | X_0 = x_0, Y_0 = y_0]. \tag{3.6}$$

The calculation $\mathbb{E}[N_t]$ allows a more fair comparison against other methods we investigate, such as uniform random sampling, as we can ensure that, on average, we are sampling the same number of pixels as other sampling schemes to account for varying patch sizes as a confounding factor in CCLE performance. For example, we can use

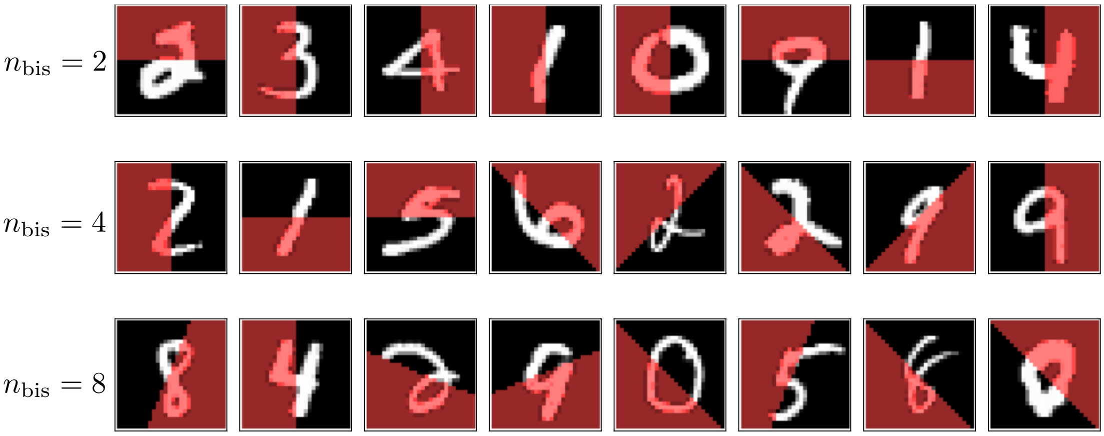

Figure 3.4: *Bisection sampling patch examples* - patch samples, π_t, coloured in red, sampled from MNIST images using the bisection sampling introduced in Section 3.2.3 for $n_{\text{bis}} \in \{2,4,8\}$.

Equation (3.6) for a say 8×8 grid to find out what γ we should choose to, on average, sample a single 8×8 patch to compare against uniform random sampling with 8×8 patches. This turns out to be $\gamma = 0.1451$ in this case.

3.2.3 Bisection Sampling

So far, the sampling methods that we have proposed are highly random. This could provide benefits in modelling images, such as better inpainting capabilities due to high stochasticity, but could also make training difficult for our model. One factor contributing to this difficulty is that the information conditioned on during training may vary in information quality. For example, a sampled patch whose complement lies in a more central region of an MNIST image will contain more useful information than a patch whose complement lies close to the border, containing homogeneous regions of uninformative black pixels. We would therefore like to investigate a method that is more likely to provide patches whose complements contain more useful information when conditioned upon so that the conditional nature of CCLE training is better utilised.

As a solution, we propose to investigate what we term *bisection sampling*. We consider several bisections of images. Specifically, consider a horizontal line extending through the centre of an image. We then then divide the π radians along the upper half of an line into n_{bis} equal angles $\{\pi/1, \pi/2, \ldots, \pi/n_{\text{bis}}\}$. We then bisect an image along lines through the central pixel at these angles to the horizontal. For example, using two bisection lines making angles $\pi/2$ and π to the horizontal leads to four potential half images, two horizontal and two vertical.

Once generated, we can sample from these half-image patches and condition on the remaining half of the image. There is an ambiguity when dividing an image on whether to include the diagonal and, if so, which half to include it in. To solve these issues, we generate the set of half images for each batch and randomly assign the diagonal to one of the two generated halves for each bisection. We include examples of samples drawn using bisection sampling using different values of n_{bis} in Figure 3.4.

3.3 Evaluation Metrics

In order to quantify the quality of the models we train, we consider several evaluation metrics. These include standard metrics commonly used to evaluate generative image models in the literature alongside additional methods that we adapt specifically for our investigations.

3.3.1 Evaluating a Model's Fit, Overfitting and Generalisation

We record a model's negative LL (NLL) in bits-per-dimension, as is common in image modelling, on training and test sets [Kingma and Dhariwal, 2018, Salimans et al., 2017].[1] This allows us to compare how well each model fits the data during training and how well it generalises to new data, as measured by the LL on the test set. In particular, this will allow us to investigate how our CCLE-trained models compare as density estimators to MLE baseline models trained using EM and SGD. This will help us address our first research question to see if a CCLE objective as framed in Section 3.2 can be a viable training alternative to MLE for training density estimators.

Further, as posed in our second research question, we would like to investigate whether CCLE training has a regulatory effect during training, making a model less prone to overfitting. To investigate this, we first give the learning curves of CCLE and MLE-trained models - curves showing the training and validation LL during training. Here, we can use the validation LL of models to observe when a model begins to overfit the training data. This is indicated when the training LL continues decreasing during training, but the validation LL begins to increase. This highlights that our model is

[1] The unit bpd is simply the LL or CCLL in base 2 normalised by the number of dimensions (often pixels, as in our case), that is the number of variables to be predicted, in a probabilistic model. One can think of bpd as roughly capturing the average number of bits needed to encode the information within each dataset dimension.

beginning to generalise worse to new data, becoming too focused and specialised on the data it sees during training.

In addition to monitoring learning curves, we also give plots comparing the generalisation performance and degree of overfitting seen during training. This is demonstrated by Liu and Van den Broeck [2021] who use a measure of the *degree of overfitting* for a model M with parameters $\boldsymbol{\theta}_M$, which we denote by $O(\boldsymbol{\theta}_M)$, given by $O(\boldsymbol{\theta}_M) = (\ell_{LL}(\boldsymbol{\theta}_M; \mathcal{D}_{\text{valid}}) - \ell_{LL}(\boldsymbol{\theta}_M; \mathcal{D}_{\text{train}}))/\ell_{LL}(\boldsymbol{\theta}_M; \mathcal{D}_{\text{valid}})$. This is simply the percentage difference of the training LL with respect to the validation LL. Plotting this at the end of training indicates the difference between how well the model has learned to fit the data and how well it generalises to new data, therefore acting as a rough indicator of the degree of overfitting. Liu and Van den Broeck [2021] plot $O(\boldsymbol{\theta}_M)$ against the test LL improvement of a model over the baselines they were comparing against. We do something similar by plotting the degree of overfitting against the test LL for each model. This allows us to compare the trade-off and relationship between the degree of overfitting during training and how well a model can generalise, which can help us, alongside the analysis of training curves, to investigate whether CCLE objectives provide a regulatory effect during training, addressing our second research question discussed in Section 2.4.

3.3.2 Measuring CCLL on the Test Set

During CCLE training, we can record the training loss, which estimates the average CCLL over the training set. We want to extend this to measure the CCLL on test sets. This metric would then indicate the *conditional generalisation* capabilities of our models. In particular, it quantifies how well our networks can model missing patches of data given the remaining portions of images. In some sense, this can then be thought of as a *quantitative measure of the inpainting capabilities of a model.*

After training, we consider a model M, with parameters $\boldsymbol{\theta}_M$. To measure the CCLL on the test set, we follow a procedure similar to that introduced by Gens and Pedro [2013]. Specifically, for each image $\boldsymbol{x} \in \mathcal{D}_{\text{test}}$, we randomly sample n_{windows} patch windows of size $p \times q$. We then average the conditional likelihood of these patches over each image's sampled patches of a given size and over the dataset:

$$\ell_{CCLL,(p,q)}(\boldsymbol{\theta}_M; \mathcal{D}_{\text{test}}) = \frac{1}{n_{\text{windows}} pq \log(2) |\mathcal{D}_{\text{test}}|} \sum_{j=1}^{|\mathcal{D}_{\text{test}}|} \sum_{i=1}^{n_{\text{windows}}} \log p\left(\boldsymbol{x}^{(j)}_{\pi_i(p,q)} | \boldsymbol{x}^{(j)}_{\setminus \pi_i(p,q)}; \boldsymbol{\theta}_M\right), \quad (3.7)$$

where here we have given the CCLL in bits per dimension (bpd). This gives a measure of the CCLL of our model on the test set for particular patch size $p \times q$.[2]

Given a set of patch sizes $\mathcal{S} \subset \mathbb{N}^2$, we can compute a measure of the CCLL of our model on the test set averaged over multiple patch windows as

$$\ell_{CCLL}\left(\boldsymbol{\theta}_M; \mathcal{D}_{\text{test}}\right) = \frac{1}{|\mathcal{S}|} \sum_{(p,q)\in\mathcal{S}} \ell_{CCLL,(p,q)}\left(\boldsymbol{\theta}_M; \mathcal{D}_{\text{test}}\right) \tag{3.8}$$

This gives a more general measure of the CCLL of our model on the test set that evaluates the CCLL performance of a model across a whole collection of patch sizes rather than for specific patch sizes.

It is worth highlighting that we record both the overall average CCLL in Equation (3.8) and the average CCLL for particular patch sizes in Equation (3.7) as reporting both will help to give us a more nuanced picture when evaluating our models. For example, this will help us see if specific methods perform better on certain patch sizes.

3.3.3 Image Quality

As EiNets are generative models as discussed in Section 2.3.1, we would also like to evaluate the quality of our models by looking at their generative capabilities, specifically the quality of images sampled from these models. This provides an alternative measure of the viability of CCLE as an alternative to MLE for training EiNets.

The Fréchet inception distance (FID)[Heusel et al., 2017] is a standard metric used within the generative model literature. It considers two sets of data, the test set and a set of images generated by a model. Firstly, feature representations of the sets of images are extracted from a deep layer of a pre-trained convolutional neural network, specifically the Inception-v3 model [Szegedy et al., 2016]. Using these extracted feature representations, a measure of similarity between the two datasets is then computed using the Fréchet metric [Fréchet, 1957] between two Gaussian distributions fitted to the extracted feature representations of both datasets [Heusel et al., 2017]. The pre-trained Inception-v3 model's deeper layers have learned to pick out key features within images, such as edges and objects, which can then be used to evaluate the similarity of pairs of images.

In particular, we can compute the FID scores on whole image samples (FID_{full}) as a measure of the overall quality of generated samples from our models. Moreover, we can report FID scores on partial image samples (FID_{inp}), that is, for images with missing

[2] Note that here we average over the number of pixels in the patch used to compute the conditional likelihood for an image **not** the total number of pixels in each image as one would do when converting LLs to bpd.

patches which are subsequently filled, conditioned on the rest of the image. Therefore, FID_{inp} scores will help us to investigate the inpainting capabilities of our models.

Inception scores (IS), in general, measures like FID scores based on feature extraction from inception models, seem to correlate with human judgment on evaluating image quality [Salimans et al., 2016]. However, a limitation of IS is that they are black-box methods involving neural networks. Similarly to other pre-trained metrics like COMET [Rei et al., 2020], a pre-trained model used for evaluating machine translation models, IS metrics lack concrete interpretability. Therefore, to supplement FID scores, we include generated samples from our models to give a more concrete and more easily interpretable measure of the quality of sampled images.

Chapter 4

Experiments

In this chapter, we give specific details on the setup and findings of our experiments investigating the research questions detailed in Section 2.4.

4.1 Experimental Setup

We investigate EiNet models trained on 28 × 28 greyscale images from MNIST [LeCun et al., 1998] and Fashion-MNIST (F-MNIST) [Xiao et al., 2017a]. Treating each pixel as a discrete random variable therefore results in models defined over a 784-dimensional multivariate random variable. We use EiNet models generated with a PD-structure of $\Delta = \{4,1\}$, cutting images recursively into rectangles down to 4×4 patches and then to 1×1 pixels, in line with the description in Section 2.3.2.

Our models use categorical leaf distributions with 256 categories, representing pixel values, and $K = 32$ distributions within each leaf node. In particular, our hyperparameters, including the PD structure, are comparable to those used by Peharz et al. [2020a]. [1]

We train baseline models using MLE with SGD and s-EM, with batch sizes of 100 and learning rates of 0.01 and s-EM step-sizes of 0.05 on MNIST and F-MNIST. [2] The models are denoted by SGD and EM.

For grid sampling, we investigate square grid patch sizes of 4 and 8. We choose values of γ so that, on average, we sample single square patches of sizes 4 and 8 to

[1] Peharz et al. [2020a] mention using $\Delta = \{7,28\}$. Looking at their code [Peharz et al., 2020b], we see that they are referring to dividing the MNIST images into 7 and 28 pieces along each axis, which results in step sizes of 4 and 1 - c.f. their SVHN training for comparison. Care must be taken with the order of the Δ values in their code.

[2] These values were chosen through a hyperparameter search based on the best generalising model.

compare against uniform random sampling and patches that, on average, sample half of an image worth of pixels to compare against bisection sampling. This allows for fairer comparisons between methods and accounts for variations in patch sizes and, specifically, the number of pixels sampled. For size 4 grid patches, this means using γ values of 0.0256, 0.1024 and 0.6272 for comparison against RAND_4, RAND_8 and bisection sampling respectively. For size 8 grid patches, this means using γ values of 0.1451 and 0.8889 for comparison against RAND_8 and bisection sampling, respectively.[3].

The datasets are divided into training (50,000 images), validation (10,000 images), and test sets (10,000 images). Training is done for up to 64 epochs with early stopping using a patience of 8 epochs based on validation LL. LLs are reported in bpd, and the training is performed on NVIDIA RTX A6000 GPUs, typically completing in under 10 hours.

For test evaluation of our models, we record the test LL and the test negative CCLL (NCCLL) in bpd as defined in Equation (3.7) and Equation (3.8). For NCCLL test scores, we use a variety of patch sizes for a fine-grained view of model performance on various patch shapes and sizes. In particular, we used patch sizes of $4 \times 4, 4 \times 12, 8 \times 8$ and 12×12, where rectangular patches' dimensions are swapped randomly during evaluation. For each image in a data set's test set, we sample three independent patches for computational efficiency, using $n_{\text{sample}} = 3$ in Equation (3.7) and Equation (3.8).

We assess the generative quality of our models both visually and by using FID scores. For FID scores, we first sample the same number of images as in each test set and compute the similarity between the sampled images and the test set using FID scores which we denote by FID_{full}. Furthermore, we also include FID scores to measure a model's capability in inpainting images with missing data. Specifically, we report FID scores, denoted by FID_{inp}, comparing each data set's test set to a collection of inpainted images from each model. To align the FID_{inp} to our test NCCLL scores, we use the same patch sizes as for the test NCCLL evaluation. In particular, we randomly sample a patch size for a given test set image and use our model to fill in the missing patch using MAP inference (c.f. Section 2.1.4), which is feasible and efficient for EiNets [Peharz et al., 2020a, Choi et al., 2020]. To allow fair comparisons, we use the same patch locations and sizes for each test set image when computing both test NCCLL and FID_{inp} scores.

The code we have written to conduct these experiments can be found at this GitHub repository. Our code adapts and builds upon [Peharz et al., 2020b].

[3] These γ values were calculated to 4 dp. using a script that computes Equation (3.6).

4.2 Results

In this section, we present the results of our experiments following the experimental design given above in Section 4.1. We divide this section into three parts corresponding to each research question introduced in Section 2.4.

4.2.1 CCLE as a viable alternative to MLE training (Q1)

NLL comparison Table 4.1 shows training and test NLL of our EiNet models. On MNIST, our EM model achieves comparable test NLL performance to the model Peharz et al. [2020a] use in their MNIST experiments, highlighting that we can replicate their results. Moreover, the SGD models achieve reasonable test NLLs compared to other PC models trained on these datasets [Liu et al., 2021].

The test NLL of the MNIST and F-MNIST SGD models are lower than for the EM models, although the gap between EM and SGD seems tighter on F-MNIST, with the EM algorithm seemingly able to better fit F-MNIST data. This suggests that SGD training could allow for a greater generalisation performance than s-EM training of larger models trained on larger datasets. It would be interesting for future work to focus on the comparison of s-EM and SGD for training larger EiNet models on more complex data sets. [Peharz et al., 2020a] did this for smaller binariased datasets but failed to compare s-EM to SGD on the larger datasets of SVHN [Netzer et al., 2011], MNIST and CelebA [Liu et al., 2015], as we noted in Section 2.3.1.

We note that the test NLLs of our SGD models are higher than other probabilistic models, such as normalising flows [Keller et al., 2021]. However, unlike these models, we emphasise that EiNets allow for exact and tractable inference of various types. This is particularly impressive, given the size and complexity of the models and datasets we work with.

Comparing the CCLE models, we notice that on both datasets the *models utilising uniform random sampling generalise worse than the other CCLE methods* and, looking at their training NLLs, seem less able to fit the training data than the other CCLE models. In particular, this indicates that these models are underfitting. However, $RAND_{16}$ on MNIST seems to be overfitting the data, as shown by its low training NLL and its high test NLL. The general poor fitting and generalisation could be due too to the fact that when using random sampling, at each training step, we have to model the conditional likelihood of ever-changing, most likely consecutively non-overlapping patch locations for which there are many. After optimising for one patch, a model must suddenly

Table 4.1: NLL and FID_{full} scores - average negative training and test LLs (bpd) of EiNet models trained on MNIST and F-MNIST alongside their FID_{full} score. *Lower is better.*

Model		MNIST			F-MNIST	
(M)	$-\ell_{LL}(\boldsymbol{\theta}_M;\mathcal{D}_{\text{train}})$	$-\ell_{LL}(\boldsymbol{\theta}_M;\mathcal{D}_{\text{test}})$	FID_{full}	$-\ell_{LL}(\boldsymbol{\theta}_M;\mathcal{D}_{\text{train}})$	$-\ell_{LL}(\boldsymbol{\theta}_M;\mathcal{D}_{\text{test}})$	FID_{full}
EM	1.238	1.249	173.644	3.295	3.348	403.604
SGD	**1.135**	**1.195**	**159.119**	**3.190**	**3.329**	**119.252**
RAND_4	1.315	1.346	255.204	3.488	3.559	228.446
RAND_8	1.206	1.273	238.499	3.372	3.500	192.118
RAND_{16}	1.160	1.263	238.235	3.370	3.521	186.351
$\text{BIS}_{n_{\text{bis}}=2}$	1.152	1.211	180.320	**3.197**	3.339	123.707
$\text{BIS}_{n_{\text{bis}}=8}$	1.148	1.207	176.031	3.204	**3.330**	132.872
$\text{BIS}_{n_{\text{bis}}=32}$	1.144	1.203	172.471	3.200	3.337	**122.769**
$\text{GRID}_{4,\gamma=0.0256}$	1.212	1.263	208.416	3.309	3.400	175.426
$\text{GRID}_{4,\gamma=0.1024}$	1.197	1.256	204.004	3.260	3.375	148.063
$\text{GRID}_{4,\gamma=0.6272}$	1.152	1.215	179.734	3.216	3.344	140.870
$\text{GRID}_{8,\gamma=0.1451}$	1.162	1.222	186.534	3.216	3.352	149.865
$\text{GRID}_{8,\gamma=0.8889}$	**1.130**	**1.197**	**162.081**	3.202	3.335	136.320

switch to optimising for another patch that could be far away in the image, making such training difficult and could introduce high variability in training updates.

We observe that bisection sampling models fit the training data sets reasonably well and obtain good generalisation performance, surpassing the baseline EM models and coming reasonably close to the baseline SGD models. We observe that all of the bisection sampling models generalise comparably well, indicating that increasing the number of bisections has a negligible effect on the model's overall fit to a data distribution.

For grid sampling, we see that using large values of γ generally leads to better performance for CCLE training. Larger values of γ indicate that more pixels are being sampled at every training iteration on average. In particular, we note comparing $\text{GRID}_{4,\gamma=0.6272}$, $\text{GRID}_{8,\gamma=0.8889}$ and the bisection models, which all sample on average comparable patch sizes, that all models show similar performance for the most part. These two observations, alongside the performance of bisection models and the trend in uniform random sampling models, hint that *using larger patch sizes, with MLE as the limit of this process, seems to allow for better generalisation performance*. This aligns with the observation presented in Section 2.1.3 where it is suggested that increasing patch sizes reduces the variability in CLE, leading to better estimators on average [Asuncion et al., 2010]. In addition, the similarity between grid and bisection performance suggests that *patch size may be the more important factor as compared to how the patches are sampled, contiguously or not, for generalisation performance*.

We note that bisection and grid sampling methods with larger γs are also more likely to sample consecutive patches containing overlapping regions during training. For example, consider bisection sampling, where we sample vertical and horizontal slices of an image which will overlap by a quarter of the image. This could also help to stabilise training and explain the differences between, say, the performance of the grid and bisection models and the poor performances of uniform random sampling models, which are less likely to sample consecutive overlapping patches during training.

Comparing $\text{GRID}_{4,\gamma=0.0256}$ and $\text{GRID}_{8,\gamma=0.1024}$ to RAND_4 and RAND_8, respectively, we see that the grid methods do seem to outperform their counterparts despite on average sampling the same number of pixels per iteration. This could be because, on average, the grid sampling models will occasionally see more than one patch, which may help to stabilise the difficulty in training that we have discussed above when using random sampling.

Generative comparison Finally, we compare the generative performance of our models by looking at their FID_{full} scores as given in Table 4.1 and generated image samples in Figure 4.1 and Figure 4.2 in the appendix. We firstly note that the *EM models seem to have an extremely high FID_{full} scores* indicating that the samples they have generated are very much unlike the samples in the MNIST and F-MNIST test sets. This is interesting as the densities of the EM models are not unreasonable on both datasets, surpassing models such as RAND_{16}. Indeed looking at samples drawn from our EM models in Figure 4.1 and Figure 4.2, we see that the full images sampled from our EM models essentially amount to noise. Therefore, the *EM models seem to be able to model the data distribution to some extent but cannot use this to generate whole image samples*.

Interestingly, Peharz et al. [2020a] show that a comparably performing model still allows for good anomaly detection, confirming that these models are still learning some information about the distributions they are modelling, which is further shown in our case by the inpainted images they produce in Figure 4.5. The poor performance of EM could be due to the random nature of training, indicating that we have fitted noise here. Future work should look further into this to see if this is a systemic issue when using EM for training large EiNet models using discrete pixel input variables.

Looking at our CCLE models, all produce FID_{full} scores larger than the SGD baseline models indicating poorer whole image samples are generated by these distributions. Looking at Figure 4.1 and Figure 4.2, we visually confirm this for a selection of our models, with the uniform random sampling models producing the worst quality im-

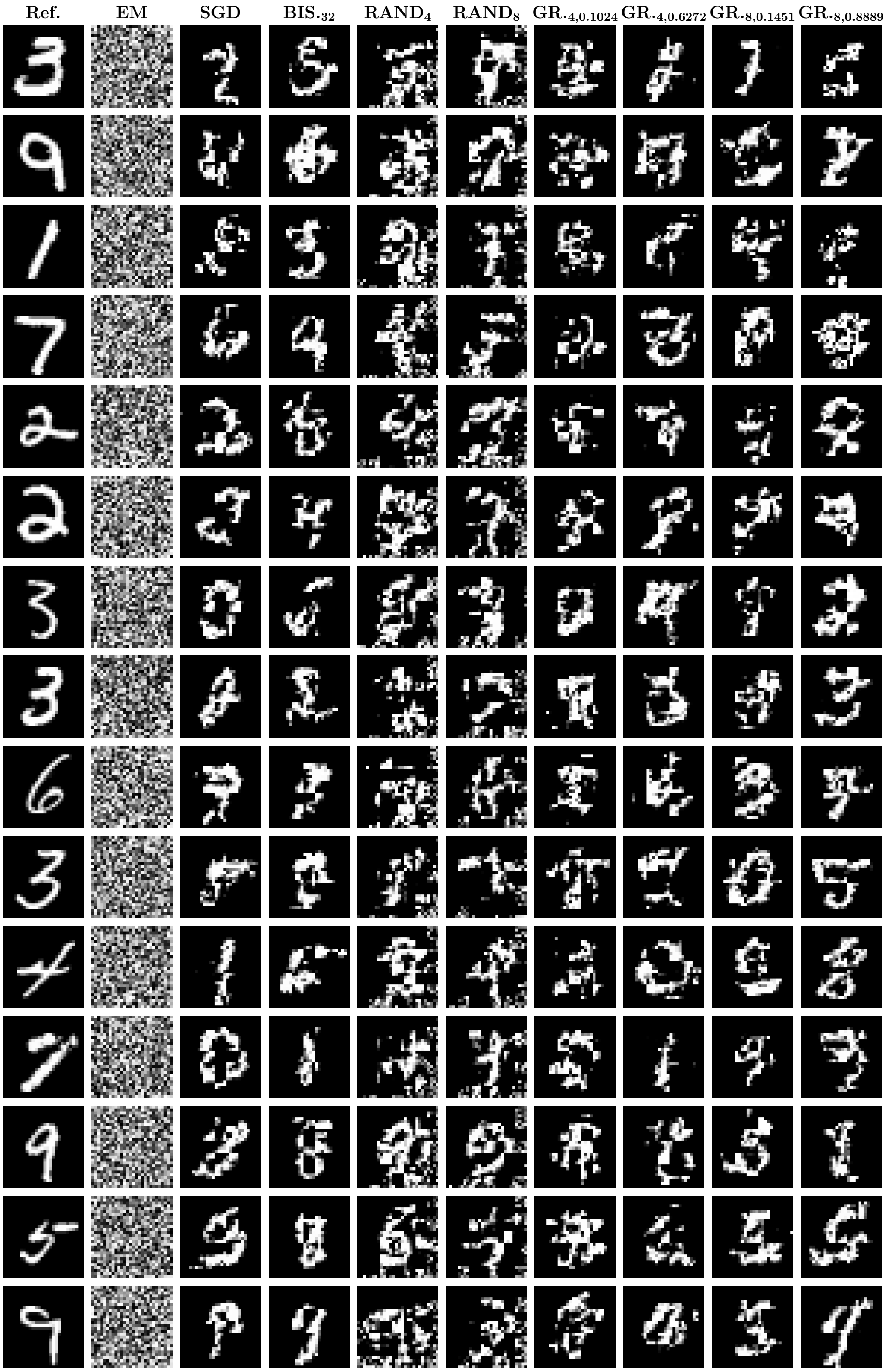

Figure 4.1: *MNIST Samples* - whole images sampled from a collection of EiNets trained on MNIST.

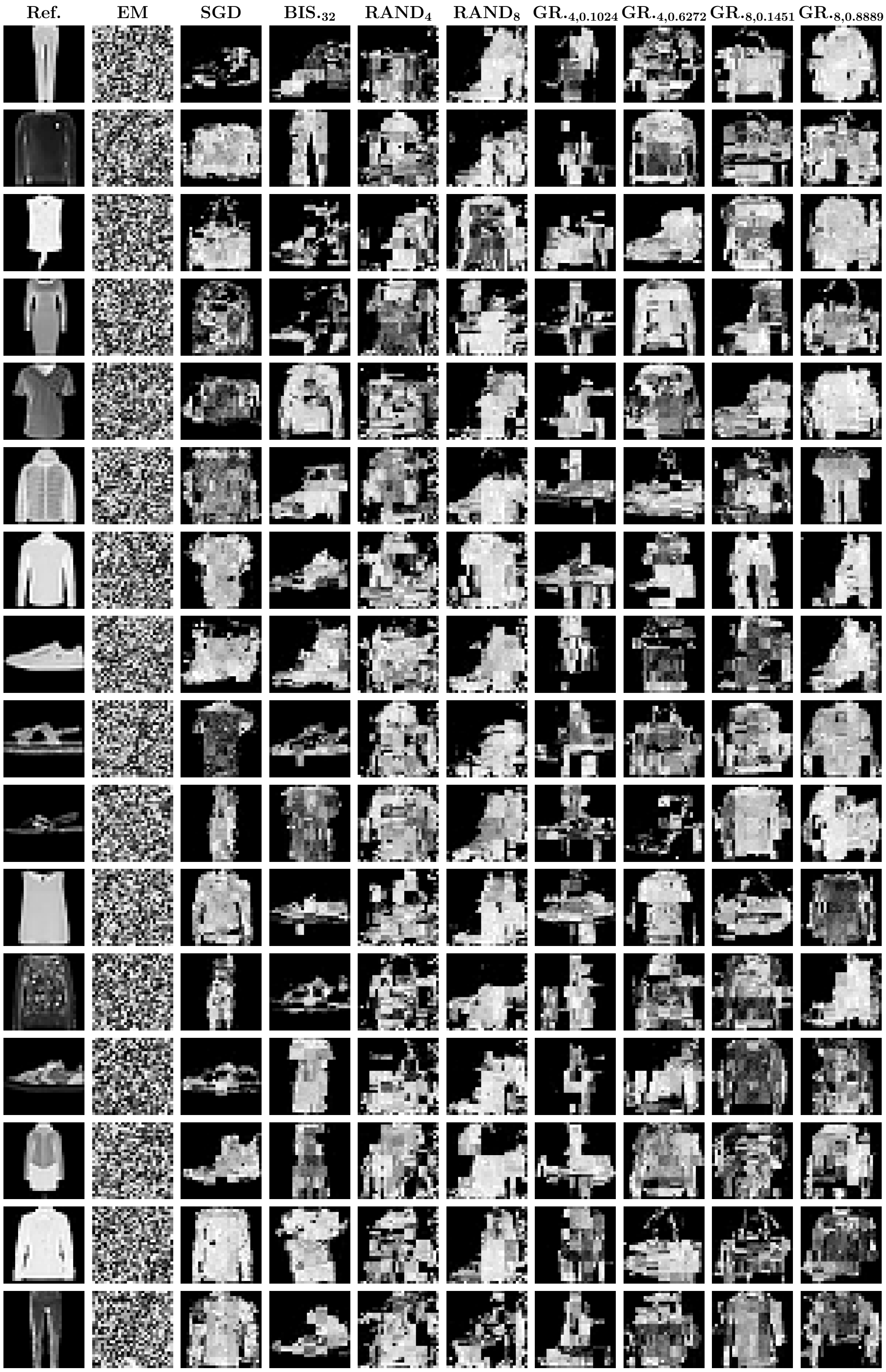

Figure 4.2: *F-MNIST Samples* - whole images sampled from a collection of EiNets trained on F-MNIST.

ages on both MNIST and F-MNIST. However, despite their high FID_{full} scores, the remaining models in Figure 4.1 and Figure 4.2 generate reasonable samples with a semblance of structure and form on both datasets. Moreover, *models such as $BIS_{n_{bis}=32}$ and $GRID_{8,\gamma=0.8889}$ have comparable FID_{full} to the baseline SGD models on both datasets (closer on MNIST) and we see visually that the samples generated from these models alongside $GRID_{4,\gamma=0.6272}$ are of similar quality also*. This is impressive given that these models are trained to model local regions, highlighting that *local modelling can also allow for reasonable global generation*. Again we notice that the larger patch-size models seem to perform better. The visual difference between sample quality of the larger patch size CCLL models is negligible, *again suggesting that it is patch size more so than the contiguous nature of the patches that makes a performative difference*.

In conclusion, we find that **CCLE training is a viable alternative to training using MLE**. Indeed CCLE training, using bisection and grid sampling, **can achieve comparable generalisation and generative performance as MLE SGD-trained models and that can outperform MLE models trained via EM, with these EM models showing poor generative quality**. Moreover, we find that generally, the **larger the patch sizes used for CCLE training, the better the generalisation performance and generative performance.**

4.2.2 CCLE objectives as regularisation (Q2)

Now we look into whether CCLE objectives have a regularisation effect during training as discussed in Section 2.4. Looking at Table 4.1, we note that the SGD baseline models achieve the lowest training NLL (apart from one model on MNSIT). Moreover, if we look at Figure 4.3, we see that the *SGD model begins to overfit the earliest during training* on MNIST as shown by the earliest increase in validation NLL whilst its training NLL decreases. Adding to this, in Figure 4.4, we see that the SGD model shows one of the highest degrees of overfitting on both MNIST and F-MNIST. All of this goes to show the *tendency of EiNets trained using MLE via SGD to be prone to overfit early during training*, which fits with our discussions in Section 2.4 and observations made in the literature [Liu and Van den Broeck, 2021, Shih and Ermon, 2020].

Turning to CCLE-trained models, we can see in Figure 4.3 that the *MNIST CCLE models as a whole do not suffer as greatly from early overfitting as the SGD models*. This is shown as the uptick in validation NLL occurs later in training for most mod-

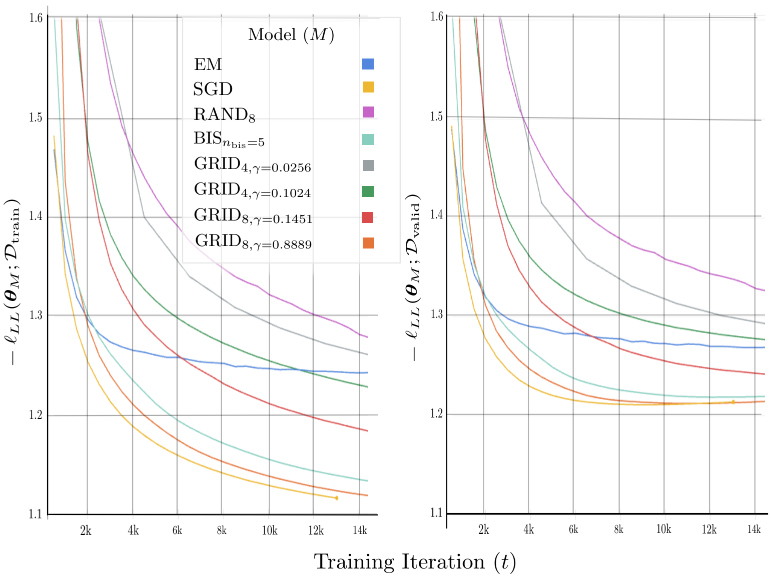

Figure 4.3: *Degree of overfitting vs generalisation* - plot depicting the degree of overfitting of each model, $O(\boldsymbol{\theta}_M)$, against its test set NLL on MNIST.

els.[4] Furthermore, looking at Figure 4.4, we note that *on MNIST, most of the CCLE models show a smaller degree of overfitting during training with often comparable generalisation performance for grid and bisection models*, aligning with our remarks on Figure 4.3.

On F-MNIST, the story is slightly more complicated as in Figure 4.4, we *observe slightly lower but comparable levels of overfitting between our best performing CCLE models to the SGD model, but also with comparable generalisation performances*. Significance testing with repeats is required in future work to confirm any overfitting differences.

On both datasets, we observe in Figure 4.4 and Figure 4.3 that *models using larger patches sizes, even on average as with grid sampling models, show higher degrees of overfitting*. This could be because, for models using smaller patch sizes, as mentioned already, there are many more potential locations for smaller patches to be placed, which could make training more difficult and therefore have a greater regularisation effect. This is further supported by their generally higher training NLL in Table 4.1. This suggests that *a balance is required when training a CCLE model between using smaller patch sizes for regularisation and using larger patch sizes for the generally greater generalisation capabilities*. This indicates a promising avenue for future research to

[4] In Figure 4.3, we only plot the first 25 epochs of training, at which point the training of the MLE SGD-trained model stops due to early stopping.

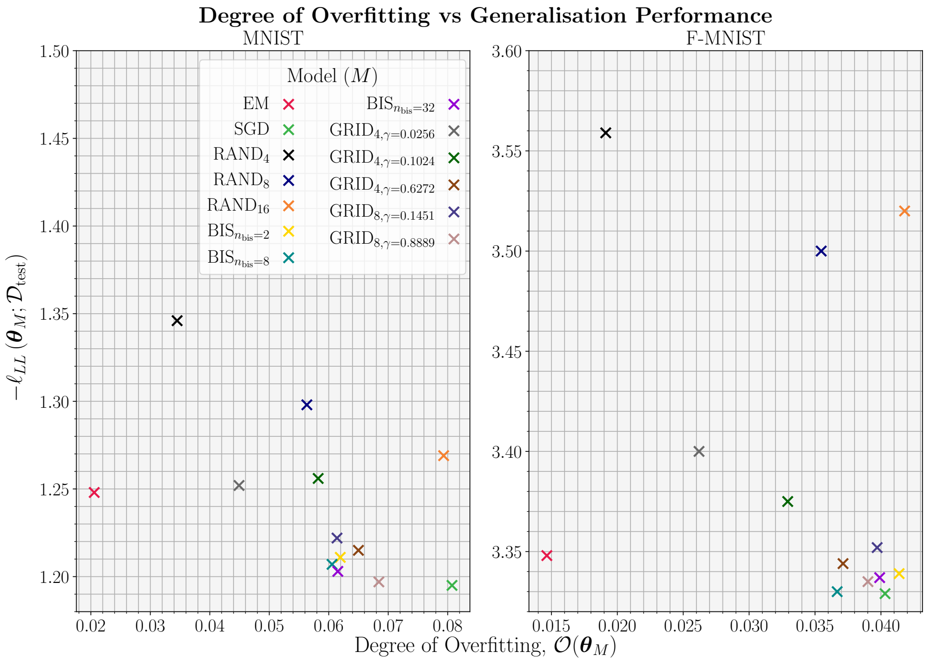

Figure 4.4: *MNIST learning curves* - learning curves of a selection of EiNet models trained on MNIST. The left figure shows the training NLLs, and the right figure shows the validation NLLs.

perform a more fine-grained analysis on the relation between patch size, overfitting and generalisation performance.

Interestingly, on both datasets, we observe that *EM achieves the lowest degree of overfitting compared to MLE trained via SGD or over the CCLE trained models also trained via SGD*. This is less meaningful on MNIST, where EM is not fitting data as well as the SGD or some of the CCLE models. However, this is rather significant on F-MNIST, where EM shows very competitive generalisation performance. Overall, alongside our previous comments on LLs, this suggests that *EM could be less likely to suffer from overfitting* in general compared to SGD-based training, which *could come at the cost of being more prone to underfitting depending on the dataset*. Again repeated experiments and additional datasets would be required to make a more definitive statement.

In conclusion, we find that **CCLE objectives show mixed results as a regularisation method**. In particular, we find that the degree of regularisation **could depend on the dataset and the patch size chosen with smaller patch sizes exhibiting a more significant regularisation effect**. Moreover, we find that **SGD-based methods, in general, show more significant degrees of overfitting as compared to EM-trained**

models, which is countered by the fact that the EM models seem to show less consistent and worse generalisation performance..

4.2.3 Inpainting capabilities of CCLE models (Q3)

NCCLL Comparison We note a significant difference between the NCCLL results on MNIST and F-MNIST shown in Table 4.2. Indeed, we observe that *on MNIST, the CCLE trained models*, although some achieve comparable scores, *show higher and therefore worse NCCLLs than the baseline SGD-trained MLE model, yet lower than the EM-trained MLE model.* However, *on F-MNIST, all CCLE models attain lower NCCLL scores across all patch sizes compared to the MLE baselines, showing better conditional modelling of local regions.* This suggests that CCLE training has been most effective across all sampling methods on F-MNIST, *with all models able to conditionally generalise better to new local patches.*

We observe that the *EM-trained MLE baseline models show dramatically higher NCCLL compared to the rest of the models across all patch sizes*, indicating this training method has learned little on the local structure of the images it is trying to model despite producing a reasonable density specifically on F-MNIST (c.f. Table 4.1). This is less expected on F-MNIST, where the EM model shows significantly worse NCCLLs than models with worse density estimates such as RAND$_4$.

Now focusing on the CCLE-trained models, we note that on MNIST, there is a more significant variation in NCCLLs between models than on F-MNIST. For example, this is shown in the larger differences in CCLL scores between the uniform random and grid models such as GRID$_{8,\gamma=0.1451}$ and GRID$_{4,\gamma=0.8889}$. However, on F-MNIST, we observe very comparable NCCLL scores across the board for all models, with only GRID$_{8,\gamma=0.8889}$ GRID$_{n_{\text{bis}}=32}$ showing more significant lower NCCLL scores. This indicates that on *F-MNIST, our CCLE models generally show more comparable local conditional modelling.* It is worth highlighting that the uniform sampling methods do not perform significantly better on test patch sizes of the same size as those they were trained on. This could link to the difficulty in training models using uniform random sampling as discussed in Section 4.2.1.

We observe that the NCCLLs of RAND$_8$ models are similar to the comparable models GRID$_{4,\gamma=0.1024}$ on F-MNIST, which sample around four 4×4 patch samples during training (i.e. the same number of pixels as sampling a single 8×8 but non-contiguously). Furthermore, on both datasets, we observe comparable NCCLL across

Table 4.2: Test NCCLL and FID$_{\text{inp}}$ scores - average test negative CCLLs and their associated standard deviations over a collection of patch windows of different sizes as well as the total average over all patch size windows of EiNet models on MNIST and F-MNIST, alongside their associated FID$_{\text{inp}}$ values. *Lower is better.*

	Model	$-\ell_{CCLL,(p,q)}(\boldsymbol{\theta}_{\boldsymbol{M}};\mathcal{D}_{\text{test}})$				$-\ell_{CCLL}(\boldsymbol{\theta}_{\boldsymbol{M}};\mathcal{D}_{\text{test}})$	FID$_{\text{inp}}$
	(M)	$(4,4)$	$(4,12)$	$(8,8)$	$(12,12)$		
MNIST	EM	$7.029_{\pm 0.472}$	$7.118_{\pm 0.375}$	$7.153_{\pm 0.346}$	$7.216_{\pm 0.228}$	$7.129_{\pm 0.0369}$	21.568
	SGD	$\mathbf{1.520}_{\pm 1.860}$	$\mathbf{1.833}_{\pm 1.435}$	$\mathbf{1.971}_{\pm 1.337}$	$\mathbf{2.199}_{\pm 0.910}$	$\mathbf{1.881}_{\pm 1.433}$	**4.961**
	RAND$_4$	$1.584_{\pm 1.916}$	$1.917_{\pm 1.483}$	$2.070_{\pm 1.377}$	$2.311_{\pm 0.932}$	$1.970_{\pm 1.493}$	9.132
	RAND$_8$	$1.557_{\pm 1.900}$	$1.879_{\pm 1.467}$	$2.022_{\pm 1.367}$	$2.257_{\pm 0.933}$	$1.929_{\pm 1.465}$	6.713
	RAND$_{16}$	$1.565_{\pm 1.921}$	$1.876_{\pm 1.473}$	$2.015_{\pm 1.368}$	$2.237_{\pm 0.934}$	$1.922_{\pm 1.489}$	5.095
	BIS$_{n_{\text{bis}}=2}$	$1.539_{\pm 1.879}$	$1.857_{\pm 1.451}$	$1.997_{\pm 1.352}$	$2.230_{\pm 0.920}$	$1.906_{\pm 1.445}$	6.522
	BIS$_{n_{\text{bis}}=8}$	$1.531_{\pm 1.869}$	$1.847_{\pm 1.445}$	$1.990_{\pm 1.345}$	$2.220_{\pm 0.917}$	$1.897_{\pm 1.442}$	5.740
	BIS$_{n_{\text{bis}}=32}$	$1.529_{\pm 1.868}$	$1.843_{\pm 1.442}$	$1.984_{\pm 1.342}$	$2.212_{\pm 0.917}$	$1.892_{\pm 1.440}$	5.652
	GRID$_{4,\gamma=0.0256}$	$1.584_{\pm 1.916}$	$1.917_{\pm 1.483}$	$2.070_{\pm 1.377}$	$2.304_{\pm 0.932}$	$1.969_{\pm 1.491}$	8.263
	GRID$_{4,\gamma=0.1024}$	$1.584_{\pm 1.923}$	$1.914_{\pm 1.483}$	$2.064_{\pm 1.379}$	$2.302_{\pm 0.940}$	$1.966_{\pm 1.480}$	9.256
	GRID$_{4,\gamma=0.6272}$	$1.539_{\pm 1.879}$	$1.858_{\pm 1.452}$	$2.001_{\pm 1.350}$	$2.233_{\pm 0.920}$	$1.910_{\pm 1.430}$	6.116
	GRID$_{8,\gamma=0.1451}$	$1.547_{\pm 1.887}$	$1.868_{\pm 1.458}$	$2.012_{\pm 1.357}$	$2.244_{\pm 0.927}$	$1.920_{\pm 1.147}$	6.860
	GRID$_{8,\gamma=0.8889}$	$\mathbf{1.528}_{\pm 1.868}$	$\mathbf{1.836}_{\pm 1.438}$	$\mathbf{1.976}_{\pm 1.339}$	$\mathbf{2.200}_{\pm 0.915}$	$\mathbf{1.885}_{\pm 1.133}$	**4.837**
F-MNIST	EM	$7.568_{\pm 0.485}$	$7.650_{\pm 0.381}$	$7.680_{\pm 0.349}$	$7.732_{\pm 0.245}$	$7.658_{\pm 0.380}$	13.420
	SGD	$\mathbf{3.949}_{\pm 2.662}$	$\mathbf{4.318}_{\pm 2.164}$	$\mathbf{4.462}_{\pm 2.035}$	$\mathbf{4.700}_{\pm 1.532}$	$\mathbf{4.357}_{\pm 2.154}$	**3.442**
	RAND$_4$	$3.848_{\pm 2.613}$	$4.221_{\pm 2.124}$	$4.371_{\pm 2.000}$	$4.615_{\pm 1.506}$	$4.264_{\pm 2.117}$	3.963
	RAND$_8$	$3.822_{\pm 2.621}$	$4.189_{\pm 2.131}$	$4.333_{\pm 2.010}$	$4.571_{\pm 1.513}$	$4.229_{\pm 2.124}$	3.054
	RAND$_{16}$	$3.827_{\pm 2.628}$	$4.190_{\pm 2.134}$	$4.332_{\pm 2.012}$	$4.565_{\pm 1.515}$	$4.229_{\pm 2.127}$	2.765
	BIS$_{n_{\text{bis}}=2}$	$3.811_{\pm 2.612}$	$4.178_{\pm 2.127}$	$4.322_{\pm 2.008}$	$4.560_{\pm 1.512}$	$4.218_{\pm 2.119}$	2.910
	BIS$_{n_{\text{bis}}=8}$	$\mathbf{3.798}_{\pm 2.604}$	$\mathbf{4.166}_{\pm 2.120}$	$\mathbf{4.309}_{\pm 2.000}$	$4.546_{\pm 1.506}$	$\mathbf{4.205}_{\pm 2.112}$	2.935
	BIS$_{n_{\text{bis}}=32}$	$3.810_{\pm 2.613}$	$4.178_{\pm 2.125}$	$4.321_{\pm 2.004}$	$4.560_{\pm 1.508}$	$4.217_{\pm 2.117}$	3.082
	GRID$_{4,\gamma=0.0256}$	$3.825_{\pm 2.608}$	$4.198_{\pm 2.120}$	$4.352_{\pm 1.997}$	$4.594_{\pm 1.504}$	$4.242_{\pm 2.113}$	4.013
	GRID$_{4,\gamma=0.1024}$	$3.816_{\pm 2.611}$	$4.185_{\pm 2.121}$	$4.335_{\pm 2.000}$	$4.574_{\pm 1.506}$	$4.228_{\pm 2.115}$	3.517
	GRID$_{4,\gamma=0.6272}$	$3.806_{\pm 2.606}$	$4.172_{\pm 2.119}$	$4.317_{\pm 1.999}$	$4.556_{\pm 1.505}$	$4.212_{\pm 2.111}$	2.826
	GRID$_{8,\gamma=0.1451}$	$3.813_{\pm 2.622}$	$4.182_{\pm 2.132}$	$4.325_{\pm 2.014}$	$4.562_{\pm 1.516}$	$4.221_{\pm 2.125}$	3.039
	GRID$_{8,\gamma=0.8889}$	$3.802_{\pm 2.613}$	$4.167_{\pm 2.125}$	$\mathbf{4.309}_{\pm 2.007}$	$\mathbf{4.543}_{\pm 1.511}$	$\mathbf{4.205}_{\pm 2.118}$	**2.534**

bisection and grid models that, on average, all sample around half an image worth of pixels, specifically GRID$_{4,\gamma=0.6272}$ and GRID$_{8,\gamma=0.8889}$. Moreover, we note negligible differences between bisection NCCLLs, further adding to our comments in Section 4.2.1. In particular, these grid and bisection models show the lowest NCCLLs over all CCLE models. These observations indicate that *non-contiguous patching does not seem to aid in local conditional modelling performance significantly*, and that generally, *larger patch sampling methods seem to be able to model both small and large local regions better*, echoing what we noted in our discussions in Section 4.2.1.

Finally, we note the large standard deviations of the computed NCCLLs, which indicates great variability in every model's NCCLL performance. This variability most likely occurs between images and patches selected since some images and patches will likely be more straightforward to model than others. For example, patches closer to the border of MNIST images tend to be formed mostly of black pixels. However, this could also be down to our use of $n_{\text{sample}} = 3$ for test NCCLLs calculations.

Visual Inpainting and FID$_{\text{inp}}$ Comparisons We see, in Table 4.2, lower FID$_{\text{inp}}$ scores of CCLE models to the MLE baseline models on F-MNIST, indicating better inpainting performance and aligning with the NCCLL scores discussed above. On MNIST, we observe all models showing higher and, therefore, seemingly worse inpainting quality through higher FID$_{\text{inp}}$ scores than the SGD baseline. However, GRID$_{8,\gamma=0.8889}$ performs better than the baseline on MNIST, although not significantly. Moreover, despite showing comparable NCCLL performance to GRID$_{8,\gamma=0.6272}$ and the bisection models, GRID$_{8,\gamma=0.8889}$ shows a lower FID$_{\text{inp}}$ on F-MNIST. This highlights that GRID$_{8,\gamma=0.8889}$ shows better generative inpainting performance despite having comparable local modelling performance.

Further looking at Figure 4.5, we observe that *GRID$_{4,\gamma=0.6272}$, GRID$_{8,\gamma=0.1451}$, GRID$_{8,\gamma=0.8889}$ and BIS$_{n_{bis}=32}$ show the highest quality inpainted images on both datasets of all CCLE-trained models*. Indeed they generally show comparable quality inpainted images to the SGD baseline on MNIST, albeit the overall quality of all inpainting on MNIST is relatively poor. *On F-MNIST, these models show very good inpainting quality, generally surpassing the quality of the SGD baseline model*; this is shown by the trouser and boot inpainted images where we generally observe a smoother transition of pixel intensities and shape compared to the SGD baseline model. This aligns with the lower NCCLL and FID$_{\text{inp}}$ scores we observe in Table 4.2 on F-MNIST.

Moreover, we note that the uniform random sampling models, although, as discussed in Section 4.2.1, provide poor overall densities, seem to perform much better locally on F-MNIST. However, on MNIST, these models show poor inpainting in comparison. Again we note that the *models sampling larger patches during training show the greatest inpainting performance on MNIST and F-MNIST*, with the GRID$_{8,\gamma=0.8889}$ model on F-MNIST showing excellent inpainting for all images.

Looking at Figure 4.5, we observe that rather than producing noise as in Figure 4.1 and Figure 4.2 for full images, the EM models on both datasets can paint in some portion of the sampled images, although at a much worse quality than any other model. This indicates that the EM method has been able to learn enough information for some level of generative capabilities, but not to the extent that it can generate full image samples or even very good local samples.

The *differences in NCCLL results and inpainting quality between the MNIST and*

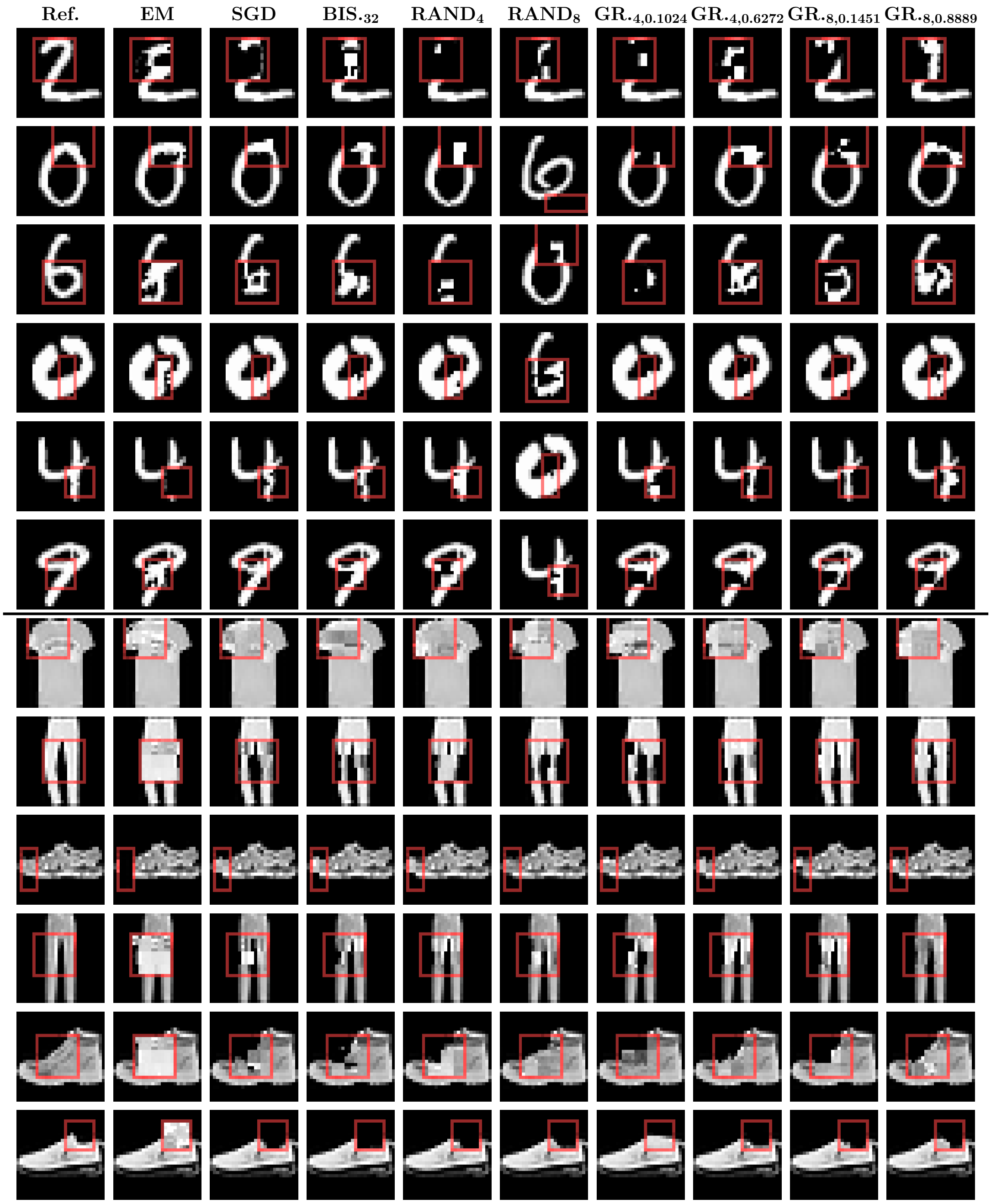

Figure 4.5: *Inpainted images* - inpainted patches (inside red boxes) of MNIST (top 6 rows) and F-MNIST (bottom 6 rows) images. Ref. denotes unaltered samples.

F-MNIST datasets may be due to the different levels of information sparsity within their images. As seen in Figure 4.1 and Figure 4.2, MNIST images contain more black pixels with mean pixel values of around 33, compared to 128 in F-MNIST. These black regions, containing little information, could affect the CCLE models during training. This sparsity may reduce the local modelling benefits that conditional training offers. Such sparsity could be particularly affecting uniform random sampling performance.

In conclusion, **we find that CCLE training shows promise in allowing for improved inpainting capabilities over MLE-trained models**. However, we note that this **could depend on the dataset trained on and the sparsity of information contained within data set images**.

4.3 Limitations and Future Work

One of the main limitations in our work presented above is the need for statistical significance testing of any improvements or differences observed during our experiments. We could not explore this due to the time and computational constraints of training the large models we investigated in this work. This is something that future work should look into to more rigorously validate or disprove any claims or observations that we have made in this project.

Another limitation of our work is highlighted by the large standard deviations of the CCLLs scores shown in Table 4.2. This could be an artifact of the difficulty in local modelling, which can be variable as discussed in Section 4.2.2. However, these standard deviations likely can be reduced by sampling more patches per image for test CCLL calculations - we were only able to sample three for computational efficiency. This would allow for a more informed view of the variability in the learned local modelling capabilities of networks and better allow for comparisons of models.

Additional areas for future work include performing a more fine-grained analysis on the relation of patch sizes to generalisation performance, regularisation and inpainting capabilities of CCLE models and between EM and SGD-trained models more generally. Moreover, related to patch sizes, future work could investigate the selection of consecutive overlapping regions in helping to stabilise the training of CCLE models as suggested in Section 4.2.1.

During our experiments, we observed that the inpainting capabilities of CCLE models depend on the dataset they were trained on, which could specifically be because of the density of information within the dataset images. Future work should explore this important connection further on datasets such as SVHN or CelebA, which alongside also investigating different architectures, would provide insights into scenarios where CCLE training could be successful or limited.

Chapter 5

Conclusion

Probabilistic inference is the primary draw of probabilistic machine learning, allowing for precise reasoning when dealing with uncertainty. EiNets are an efficient and scalable class of probabilistic models. With their ability to provide efficient and exact probabilistic inference, EiNets are an attractive alternative to expressive yet intractable deep generative models like GANs.

However, large EiNet models, necessary for modelling complex distributions of data such as images, often face problems when trained via maximum likelihood estimation (MLE), such as a susceptibility to overfitting. We address this by investigating conditional composite likelihood estimation (CCLE) as an alternative method of training EiNets. We propose three novel implementations of CCLE training: uniform random, bisection, and grid sampling. Our experiments on MNIST and Fashion-MNIST show CCLE as a promising alternative training method for density estimation and generation with EiNets. However, we find that CCLE objectives show mixed results as a form of regularisation, where we further find a trade-off between better generalisation with larger patch sizes and more significant regularisation with smaller sizes.

Moreover, we observe that CCLE-trained models often show improved inpainting capabilities over MLE-trained models, especially when using larger patch sizes. However, the success of inpainting seemingly depends on the information density of a dataset. Our work highlights the capabilities, benefits, and drawbacks of CCLE training for EiNets, providing insights more generally for probabilistic models that allow tractable conditional inference with intractable likelihoods. Future work should: confirm the findings in this work through statistical significance testing, explore the relationship between patch size and regularisation and generalisation, and explore the effect of information sparsity in training images for the success of CCLE training.

Bibliography

George B Arfken and Hans-Jurgen Weber. Mathematical methods for physicists. 1972. 16

Arthur Asuncion, Qiang Liu, Alexander Ihler, and Padhraic Smyth. Learning with blocks: Composite likelihood and contrastive divergence. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 33–40. JMLR Workshop and Conference Proceedings, 2010. 2, 8, 9, 10, 37

D. Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012. 7, 24

Rémi Bardenet, Arnaud Doucet, and Chris Holmes. On markov chain monte carlo methods for tall data. *Journal of Machine Learning Research*, 18(47), 2017. 12

Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006. 6, 7, 8, 13, 22

Jianfei Chen, Jun Zhu, Yee Whye Teh, and Tong Zhang. Stochastic expectation maximization with variance reduction. *Advances in Neural Information Processing Systems*, 31, 2018. 7

YooJung Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic modeling. 2020. 1, 2, 10, 11, 13, 14, 18, 35

Soham De and Tom Goldstein. Efficient distributed sgd with variance reduction. In *2016 IEEE 16th international conference on data mining (ICDM)*, pages 111–120. IEEE, 2016. 7

Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society: series B (methodological)*, 39(1):1–22, 1977. 7

Aaron Dennis and Dan Ventura. Learning the architecture of sum-product networks using clustering on variables. *Advances in Neural Information Processing Systems*, 25, 2012. 15

Nicola Di Mauro, Floriana Esposito, Fabrizio Giuseppe Ventola, and Antonio Vergari. Sum-product network structure learning by efficient product nodes discovery. *Intelligenza Artificiale*, 12(2):143–159, 2018. 19

Joshua Dillon and Guy Lebanon. Statistical and computational tradeoffs in stochastic composite likelihood. In *Artificial Intelligence and Statistics*, pages 129–136. PMLR, 2009. 2, 8

Maurice Fréchet. Sur la distance de deux lois de probabilité. In *Annales de l'ISUP*, volume 6, pages 183–198, 1957. 32

Mark Gales, Steve Young, et al. The application of hidden markov models in speech recognition. *Foundations and Trends® in Signal Processing*, 1(3):195–304, 2008. 12

Robert Gens and Domingos Pedro. Learning the structure of sum-product networks. In *International conference on machine learning*, pages 873–880. PMLR, 2013. 13, 19, 21, 31

Walter R Gilks and Gareth O Roberts. Strategies for improving mcmc. *Markov chain Monte Carlo in practice*, 6:89–114, 1996. 12

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014. 11

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016. 7

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017. 32

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020. 25

Harry Joe, Nancy Reid, PX Somg, David Firth, and Cristiano Varin. Composite likelihood methods. In *Report on the Workshop on Composite Likelihood*, 2012. 8

Dan Jurafsky and James H Martin. *Speech and Language Processing (3rd (draft) ed.)*. Stanford Univ, 2019. 7

Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. 1960. 12

Thomas A Keller, Jorn WT Peters, Priyank Jaini, Emiel Hoogeboom, Patrick Forré, and Max Welling. Self normalizing flows. In *International Conference on Machine Learning*, pages 5378–5387. PMLR, 2021. 36

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 6

Diederik P Kingma and Max Welling. Stochastic gradient vb and the variational auto-encoder. In *Second international conference on learning representations, ICLR*, volume 19, page 121, 2014. 1, 11, 12

Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *Advances in neural information processing systems*, 31, 2018. 30

Keith Knight. *Mathematical statistics*. CRC Press, 1999. 6

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 3, 18, 34

Yitao Liang, Jessa Bekker, and Guy Van den Broeck. Learning the structure of probabilistic sentential decision diagrams. In *Proceedings of the 33rd Conference on Uncertainty in Artificial Intelligence (UAI)*, 2017. 21

Bruce G Lindsay. Composite likelihood methods. *Comtemporary Mathematics*, 80(1): 221–239, 1988. 8

Anji Liu and Guy Van den Broeck. Tractable regularization of probabilistic circuits. *Advances in Neural Information Processing Systems*, 34:3558–3570, 2021. 2, 21, 22, 31, 41

Anji Liu, Stephan Mandt, and Guy Van den Broeck. Lossless compression with probabilistic circuits. *arXiv preprint arXiv:2111.11632*, 2021. 36

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015. 36

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Large-scale celebfaces attributes (celeba) dataset. *Retrieved August*, 15(2018):11, 2018. 18

Daniel Lowd and Pedro Domingos. Learning arithmetic circuits. *arXiv preprint arXiv:1206.3271*, 2012. 12

Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. Repaint: Inpainting using denoising diffusion probabilistic models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11461–11471, June 2022. 22

Radford M Neal and Geoffrey E Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pages 355–368. Springer, 1998. 7

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011. 18, 36

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017. 7

Robert Peharz, Bernhard C Geiger, and Franz Pernkopf. Greedy part-wise learning of sum-product networks. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part II 13*, pages 612–627. Springer, 2013. 15

Robert Peharz, Sebastian Tschiatschek, Franz Pernkopf, and Pedro Domingos. On theoretical properties of sum-product networks. In *Artificial Intelligence and Statistics*, pages 744–752. PMLR, 2015. 13

Robert Peharz, Robert Gens, Franz Pernkopf, and Pedro Domingos. On the latent variable interpretation in sum-product networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(10):2030–2044, 2016. 8, 14, 21

Robert Peharz, Steven Lang, Antonio Vergari, Karl Stelzner, Alejandro Molina, Martin Trapp, Guy Van den Broeck, Kristian Kersting, and Zoubin Ghahramani. Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In *International Conference on Machine Learning*, pages 7563–7574. PMLR, 2020a. 1, 2, 14, 15, 16, 18, 19, 20, 21, 22, 23, 24, 34, 35, 36, 38

Robert Peharz, Antonio Vergari, Karl Stelzner, Alejandro Molina, Xiaoting Shao, Martin Trapp, Zoubin Ghahramani, and Kristian Kersting. Einsumnetworks: Fast and scalable learning of tractable probabilistic circuits. `https://github.com/cambridge-mlg/EinsumNetworks`, 2020b. GitHub repository. 24, 34, 35

Robert Peharz, Antonio Vergari, Karl Stelzner, Alejandro Molina, Xiaoting Shao, Martin Trapp, Kristian Kersting, and Zoubin Ghahramani. Random sum-product networks: A simple and effective approach to probabilistic deep learning. In *Uncertainty in Artificial Intelligence*, pages 334–344. PMLR, 2020c. 1, 13, 15, 18, 20, 21

Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 689–690. IEEE, 2011. 14, 19

Andrzej Pronobis, Avinash Ranganath, and Rajesh PN Rao. Libspn: A library for learning and inference with sum-product networks and tensorflow. In *Principled Approaches to Deep Learning Workshop*, 2017. 18

Tahrima Rahman and Vibhav Gogate. Merging strategies for sum-product networks: From trees to graphs. In *UAI*, 2016. 19

Ricardo Rei, Craig Stewart, Ana C Farinha, and Alon Lavie. Comet: A neural framework for mt evaluation. *arXiv preprint arXiv:2009.09025*, 2020. 33

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986. 6

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *Advances in neural information processing systems*, 29, 2016. 33

Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*, 2017. 30

Andy Shih and Stefano Ermon. Probabilistic circuits for variational inference in discrete graphical models. *arXiv preprint arXiv:2010.11446*, 2020. 41

Valentin I Spitkovsky, Hiyan Alshawi, and Dan Jurafsky. Breaking out of local optima with count transforms and model recombination: A study in grammar induction. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1983–1995, 2013. 7

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016. 32

Martin Trapp, Robert Peharz, Hong Ge, Franz Pernkopf, and Zoubin Ghahramani. Bayesian learning of sum-product networks. *Advances in neural information processing systems*, 32, 2019. 15

Vladimir Vapnik. Principles of risk minimization for learning theory. *Advances in neural information processing systems*, 4, 1991. 23

Cristiano Varin, Nancy Reid, and David Firth. An overview of composite likelihood methods. *Statistica Sinica*, pages 5–42, 2011. 9

Antonio Vergari, Nicola Di Mauro, and Floriana Esposito. Simplifying, regularizing and strengthening sum-product network structure learning. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part II 15*, pages 343–358. Springer, 2015. 19, 21

Antonio Vergari, YooJung Choi, and Robert Peharz. Probabilistic circuits: Representations, inference, learning and applications. Online tutorial presented at the NeurIPS Workshop, 2022. URL `https://nips.cc/virtual/2022/tutorial/55809`. Moderators: Guy Van den Broeck, Jessica Schrouff. 20

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017a. 34

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017b. 3

Ling Yang, Zhilong Zhang, Yang Song, Shenda Hong, Runsheng Xu, Yue Zhao, Yingxia Shao, Wentao Zhang, Bin Cui, and Ming-Hsuan Yang. Diffusion models: A comprehensive survey of methods and applications. *arXiv preprint arXiv:2209.00796*, 2022. 11

M. Yasuda, J. Ohkubo, and K. Tanaka. Digital image inpainting based on markov random field. In *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*, volume 2, pages 747–752, 2005. doi: 10.1109/CIMCA.2005.1631558. 22

Zhongjie Yu, Devendra Singh Dhami, and Kristian Kersting. Sum-product-attention networks: Leveraging self-attention in energy-based probabilistic circuits. In *The 5th Workshop on Tractable Probabilistic Modeling*, 2022. 2

Cheng Zhang, Judith Bütepage, Hedvig Kjellström, and Stephan Mandt. Advances in variational inference. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):2008–2026, 2018. 12

Eric R Ziegel. The elements of statistical learning, 2003. 7