UNIVERSITY OF CALIFORNIA

Los Angeles

Scaling Up Probabilistic Circuits for Inference Demanding Applications

A thesis submitted in partial satisfaction
of the requirements for the degree
Master of Science in Computer Science

by

Meihua Dang

2023

ABSTRACT OF THE THESIS

Scaling Up Probabilistic Circuits for Inference Demanding Applications

by

Meihua Dang
Master of Science in Computer Science
University of California, Los Angeles, 2023
Professor Guy Van den Broeck, Chair

There is a trade-off between expressiveness and tractability in generative modeling. On the one hand, while neural-based deep generative models are extremely expressive, the ways we can query them are limited; on the other hand, while tractable probabilistic models support efficient computation of various probabilistic queries, scaling them up is a major challenge. Probabilistic circuits are a tractable representation of probability distributions allowing for exact and efficient computation of likelihoods and marginals. We study the task of scaling up the learning of probabilistic circuits and then applying them to various applications. On the learning front, we propose a new algorithm for learning the sparse structures of probabilistic circuits that can significantly improve their capacity. On the application front, we further demonstrate the expressiveness and tractability of probabilistic circuits in two downstream applications: genetic sequence modeling and controllable language generation.

To my family

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I am fortunate to be advised by Professor Guy Van den Broeck, who introduced me to the field of Artificial Intelligence and has been mentoring and inspiring me ever since. Thank you for always being considerate, valuing my thoughts no matter how naive they are, and encouraging me to explore the unknown. I hope that one day I can be a researcher like you.

I would also like to thank every other professor who has mentored me, including my committee members: Professor Aditya Grover, Professor Nanyun Peng, and Professor Sriram Sankararaman. You always bring unique insights to my research and the projects would not be successful without your guidance.

I would also like to thank all the incredible collaborators and friends in LA. Life won't be such colorful without you.

Lastly, I am really grateful for all the unconditional love and support from my family.

VITA

2016 – 2020 B.S. (Computer Science), Nanjing University. China.

PUBLICATIONS

Honghua Zhang*, **Meihua Dang***, Nanyun Peng, and Guy Van den Broeck. Tractable Control for Auto-regressive Language Generation. *In Proceedings of the 40th International Conference on Machine Learning (ICML)*, 2023.

Baiting Zhu, **Meihua Dang**, and Aditya Grover. Scaling Pareto-Efficient Decision Making via Offline Multi-Objective RL. *In Proceedings of the 11th International Conference on Learning Representations (ICLR)*, 2023.

Meihua Dang, Anji Liu, and Guy Van den Broeck. Sparse Probabilistic Circuits via Pruning and Growing. *In Advances in Neural Information Processing Systems 35 (NeurIPS)*, 2022.

Meihua Dang, Anji Liu, Xinzhu Wei, Sriram Sankararaman*, and Guy Van den Broeck*. Tractable and Expressive Generative Models of Genetic Variation Data. *In Proceedings of the International Conference on Research in Computational Molecular Biology (RECOMB)*, 2022.

CHAPTER 1

Introduction

Probabilistic circuits (PCs) are a unifying framework to abstract from a multitude of tractable probabilistic models [VCP20, CVB20]. The key property that separates probabilistic circuits from other deep generative models such as flow-based models [PNR21] and VAEs [KW13] is their *tractability*. It enables them to answer various probabilistic queries, including marginal probabilities, exactly and efficiently [VCL21]. Therefore, probabilistic circuits are increasingly used in inference-demanding applications such as enforcing algorithmic fairness [CFB20, CDB21], making predictions under missing data [CPC20, KCL19, LZV21], data compression [LMB22], and anomaly detection [DMT22].

Despite the tractability of probabilistic circuits, scaling them up for generative modeling on large real-world datasets has been a major challenge. Unlike deep neural networks with pre-defined model architectures, probabilistic circuits rely on *structure learning* to learn architectures before *parameter learning*. Prior structure learning approaches try to incrementally improve existing structures by iteratively adding nodes and edges [LBB17, DVB20]. Such techniques cannot be easily parallelized and often suffer from poor local optimums.

Recently, another line of research also explores pre-defined probabilistic circuit architectures and focuses on scaling up the model by e.g., increasing the size of latent space [SSE21, LB21]. Although there has been significant progress in improving the scale and expressiveness of probabilistic circuits, however, the performance of probabilistic circuits plateaus as the model size increases. We discover that most capacity in existing large model structures is wasted: fully-connected parameter layers are only sparsely used.

In this thesis, we propose a *pruning* and *growing* algorithm to exploit the sparsity of probabilistic circuit structures, significantly improving their expressiveness [DLB22]. As suggested by its name, the pruning and growing algorithm consists of two operations: the pruning operation removes unimportant sub-networks of probabilistic circuits for model compression; the growing operation increases model capacity by increasing the size of the latent space. By alternatively applying pruning and growing operations, we increase the capacity that is meaningfully used, allowing us to significantly scale up probabilistic circuit learning.

For the software support of scaling up probabilistic circuits, we also develop the JUICE library [DKL21]. JUICE is an open-source Julia package providing tools for logic and probabilistic reasoning and learning based on circuit representations. It provides a range of efficient algorithms for probabilistic inference queries, such as computing marginal probabilities and maximum a posterior state, as well as many other advanced queries. Additionally, it supports several parameter and structure learning algorithms proposed in the recent literature. By leveraging parallelism on GPU, JUICE provides a fast implementation of circuit-based algorithms, which makes it suitable for tackling large-scale datasets and models.

Empirically, our learner achieves state-of-the-art likelihoods on MNIST-family image datasets and on Penn Tree Bank language data compared to the other probabilistic circuit learners and the less tractable deep generative models such as flow-based models and variational autoencoders.

Based on the effort for scaling up probabilistic circuits, we further demonstrate their *expressiveness* for generative modeling and the power of their *tractability* in terms of challenging probabilistic inferences on two challenging downstream applications: *genetic sequence modeling* and *controllable language generation* [DLW22, ZDP23].

For genetic sequence modeling, we leverage probabilistic circuits for modeling genetic sequence data and simulating artificial genomes (AGs). We present a new simulator for artificial genomes based on probabilistic circuits, which has comparable or better performance than the current state-of-the-art AG simulators in capturing key population genetic statistics such as allele frequencies, linkage disequilibrium, pairwise haplotype distances, and population structure, while being tractable and expressive. By capturing long-range correlations and offering probabilistic inference, probabilistic circuits are particularly suitable for modeling genetic data.

Furthermore, we demonstrate the tractability of probabilistic circuits in controllable language generation tasks. Despite the success of autoregressive large language models in text generation, it remains a significant challenge to generate text that satisfies complex constraints: sampling from the conditional distribution $\Pr(\text{text}|\alpha)$ is intractable for even the simplest lexical constraints α. In this part, we use tractable probabilistic models, viewed as probabilistic circuits, to impose complex lexical constraints (denoted α) in autoregressive language generation from large language models. Specifically, we provide token-level guidance to autoregressive generation by computing $\Pr_{\text{TPM}}(x_{t+1}|x_{1:t}, \alpha)$. With the hidden Markov model as a running example, we (1) present an efficient dynamic programming algorithm for conditioning HMMs on complex lexical constraints and (2) demonstrate the effectiveness of our method on various constrained generation benchmarks; it achieves state-of-the-art generation quality (i.e. BLEU-4 scores) while guaranteeing 100% constraint satisfaction. This work opens up new avenues for constrained language generation and motivates the development of more expressive tractable probabilistic models.

This thesis is organized as follows. In Chapter 2, we introduce the basics for probabilistic circuits and the Juice library for their efficient implementation. In Chapter 3, we describe the pruning and growing algorithm to learn scalable and efficient probabilistic circuits. In Chapter 4, we demonstrate the expressiveness of probabilistic circuits in the genetic sequence modeling task, and in Chapter 5, we show the tractability of probabilistic circuits in the controllable language generation task. Finally, we conclude in Chapter 6.

CHAPTER 2

Probabilistic Circuits

In this chapter, we review probabilistic circuits, a unifying framework for tractable probabilistic models. Specifically, we first go through its syntax, e.g. representations; and then semantics, namely, how tractability is guaranteed by its structural properties. Additionally, we also introduce parameter estimation and several structure learning algorithms of probabilistic circuits and finally their efficient implementations.

2.1 A Unifying Framework

Probabilistic circuits (PCs) [VCP20, CVB20] model probability distributions with a structured computation graph. It is a unified framework for a large family of tractable probabilistic models (TPMs) including (1) tractable graphical models such as hidden Markov models [RJ86], bounded tree-width graphical models [MJ00] and (2) probabilistic models in circuit representations such as arithmetic circuits [Dar02, Dar03], sum-product networks (SPNs) [PD11], cutset networks [RKG14], and-or search spaces [MD05], probabilistic decision graphs [JNS06] and probabilistic sentential decision diagrams [KVC14].

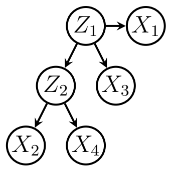
2.2 Representations

Notation. We use upper-case letters for random variables, e.g., X, Y, and lower-case ones for their assignments e.g., x, y. Analogously, sets of random variables are denoted by upper-case bold letters, e.g., $\mathbf{X}$, $\mathbf{Y}$, and their joint values by the corresponding lower-case ones, e.g., $\boldsymbol{x}$, $\boldsymbol{y}$. Specifically, we use random variables $\mathbf{Z}$ to denote hidden variables or latent states.

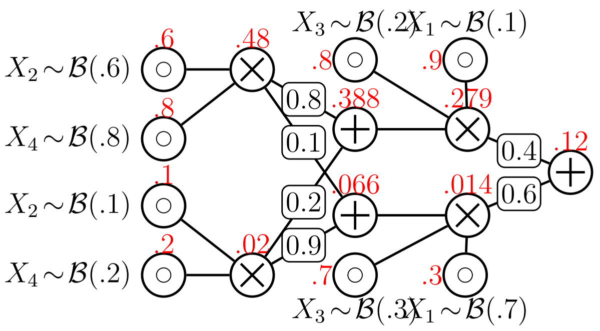
Definition 1 (Probabilistic Circuit)**.** A probabilistic circuit $\mathcal{C} := (\mathcal{G}, \boldsymbol{\theta})$ represents a joint probability distribution $\Pr(\mathbf{X})$ over random variables $\mathbf{X}$, where $\mathcal{G}$ is a directed acyclic graph (DAG) representing a computational graph, also called the *circuit structure*; and $\boldsymbol{\theta}$ are the *circuit parameters*. Similar to neural networks, each node in the DAG defines a computational unit. Specifically, the DAG $\mathcal{G}$ consists of three types of units — *input*, *sum*, and *product*. Every leaf node in $\mathcal{G}$ is an input unit; every inner unit n (i.e., sum or product) receives *inputs* from its children $\mathsf{in}(n)$, and computes *output*, which encodes a probability distribution $\Pr_n$ defined recursively as follows:

$$\Pr_n(\boldsymbol{x}) := \begin{cases} f_n(\boldsymbol{x}) & \text{if } n \text{ is an input unit,} \\ \prod_{c \in \mathsf{in}(n)} \Pr_c(\boldsymbol{x}) & \text{if } n \text{ is a product unit,} \\ \sum_{c \in \mathsf{in}(n)} \theta_{c|n} \cdot \Pr_c(\boldsymbol{x}) & \text{if } n \text{ is a sum unit,} \end{cases} \tag{2.1}$$

where $f_n(\boldsymbol{x})$ is a univariate input distribution (e.g, Gaussian, Categorical), and $\theta_{c|n}$ denotes the parameter that corresponds to edge (n, c) in the DAG. For every sum unit n, its input parameters sum up to one, i.e., $\sum_{c \in \mathsf{in}(n)} \theta_{c|n} = 1$. Intuitively, a product unit defines a factorized distribution over its inputs, and a sum unit represents a mixture over its input distributions with weights $\{\theta_{c|n} : c \in \mathsf{in}(n)\}$. Finally, the probability distribution of a probabilistic circuit (i.e., $\Pr_{\mathcal{C}}$) is defined as the distribution represented by its root unit r (i.e., $\Pr_r(\boldsymbol{x})$), that is, its output neuron. The size of a probabilistic circuit denoted $|\mathcal{C}| = |\boldsymbol{\theta}|$, is the number of parameters in $\mathcal{C}$. We assume without loss of generality that a probabilistic circuit alternates between layers of sum and product units before reaching its inputs. Figure 2.1 shows an example of a probabilistic circuit.



(a) An equivalent Bayesian network over 4 variables $\mathbf{X}$ and 2 hidden variables $\mathbf{Z}$.



$X_2 \sim \mathcal{B}(.6)$
$X_4 \sim \mathcal{B}(.8)$
$X_2 \sim \mathcal{B}(.1)$
$X_4 \sim \mathcal{B}(.2)$
$X_3 \sim \mathcal{B}(.2)$
$X_1 \sim \mathcal{B}(.1)$
$X_3 \sim \mathcal{B}(.3)$
$X_1 \sim \mathcal{B}(.7)$
.6
.8
.1
.2
.48
.02
.8
.9
.7
.3
.388
.066
.279
.014
.12
0.8
0.1
0.2
0.9
0.4
0.6

(b) A smooth and decomposable probabilistic circuit. The feedforward computation order is from left to right; $\odot$ are input Bernoulli distributions, $\otimes$ are product units, and $\oplus$ are sum units; parameter values are annotated in the box. The probability of each unit given input assignment $\{X_1 = 0, X_2 = 1, X_3 = 0, X_4 = 1\}$ is labeled red.

Figure 2.1: A smooth and decomposable probabilistic circuit and an equivalent Bayesian network over 4 variables $\mathbf{X}$ and 2 hidden variables $\mathbf{Z}$ with $h = 2$ hidden states.

2.3 Tractable Inferences

Computing the log-likelihood of a probabilistic circuit $\mathcal{C}$ given a sample $\boldsymbol{x}$ is equivalent to evaluating its computation units in $\mathcal{G}$ in a feedforward manner following Equation 2.1. The key property that separates probabilistic circuits from other deep probabilistic models such as flows [DKB14] and VAEs [KW13] is their *tractability*, which is the ability to exactly and efficiently answer various probabilistic queries. This thesis focuses on probabilistic circuits that support linear time (w.r.t. model size) marginal probability computation, as they are increasingly used in downstream applications such as data compression [LMB22] and making predictions under missing data [KCL19], and also achieve on-par expressiveness [LMB22, LB21, LBB17]. To support efficient marginal inference, probabilistic circuits need to be *smooth* and *decomposable*. We first define *scope* $\phi(n)$ of a PC unit n as the set of input variables that it depends on, and then define smoothness and decomposability as follows.

Definition 2 (Smoothness)**.** A PC $(\mathcal{G}, \boldsymbol{\theta})$ is smooth if for any sum node $n \in \mathcal{G}$, its children have identical scope: $\forall c_1, c_2 \in \mathsf{in}(n) : \phi(c_1) = \phi(c_2)$.

Definition 3 (Decomposability [DM02])**.** A PC $(\mathcal{G}, \boldsymbol{\theta})$ is decomposable if for any produce node $n \in \mathcal{G}$, its children have disjoint scopes: $\forall c_1, c_2 \in \mathsf{in}(n), c_1 \neq c_2 : \phi(c_1) \cap \phi(c_2) = \varnothing$.

Decomposability ensures that every product unit encodes a well-defined factorized distribution over disjoint sets of variables; smoothness ensures that the mixture components of every sum unit are well-defined over the same set of variables. Given a smooth and decomposable PC, querying an arbitrary marginal probability boils down to a feedforward evaluation of its DAG as in Equation 2.1 and except that the output of every input unit taking latent variable $\boldsymbol{z}$ is 1, e.g., $f_n(\boldsymbol{z}) = 1$. Thus the computation time of marginal probability is linear with respect to the size of the PC.

To compute maximum a posterior (MAP) queries in linear time, a PC should additionally satisfy a structural constraint termed *determinism*, which is a property of the PC nodes' support: for any PC node n, its support $\mathsf{supp}(n)$ is the set of complete assignments for which the output of n is non-zero: $\mathsf{supp}(n) := \{\boldsymbol{x} : \Pr_n(\boldsymbol{x}) > 0\}$. Intuitively, the support of a node n is the set of assignments $\boldsymbol{x}$ that activate it.

Definition 4 (Determinism [CD17])**.** A PC $(\mathcal{G}, \boldsymbol{\theta})$ is deterministic if for any sum node $n \in \mathcal{G}$, its children have disjoint support: $\forall c_1, c_2 \in \mathsf{in}(n)(c_1 \neq c_2), \mathsf{supp}(c_1) \cap \mathsf{supp}(c_2) = \varnothing$.

Smoothness and decomposability structure properties will be the key to guaranteeing the effectiveness of the learning and inference algorithms introduced in the following chapters. In this thesis, we will not utilize deterministic probabilistic circuits, but the concept of determinism will be useful to introduce the parameter estimation algorithms in Section 2.4.

2.4 Circuit Flows and Parameter Estimation

In this section, we briefly introduce *circuit flows* – a computational tool that allows us to perform parameter estimation efficiently. Later in Chapter 3, we will introduce circuit flows again in the perspective of model sampling, here we first formally define circuit flows via the definition of *context*.

Definition 5 (Context)**.** Let $\mathcal{C}$ be a probabilistic circuit over random variables $\mathbf{X}$ and n be one of its nodes. The *context* γ_n of node n denotes all joint assignments that return a nonzero value for all nodes in a path between the root of $\mathcal{C}$ and n.

$$\gamma_n := \bigcup_{p \in \mathsf{out}(n)} \gamma_p \cap \mathsf{supp}(n)$$

where $\mathsf{out}(n)$ refers to the parent node of n and $\mathsf{supp}(n) := \{\boldsymbol{x} : \Pr_n(\boldsymbol{x}) > 0\}$ is the support of unit n.

Note that the context of a node is different from its support. Even if the node returns a non-zero value for some input, its output may be multiplied by 0 at its ancestor nodes; i.e., such node does not contribute to the circuit output of that assignment.

We can now express circuit flows in terms of contexts. Intuitively, the context of a circuit node is the set of all complete inputs that "activate" the node. Hence, an edge is "activated" by an input if it is in the contexts of both nodes for that edge.

Definition 6 (Circuit Flow)**.** Let $\mathcal{C}$ be a probabilistic circuit over random variables $\mathbf{X}$, (n, c) its edge, and $\boldsymbol{x}$ a joint assignment to $\mathbf{X}$. The circuit flow of (n, c) given $\boldsymbol{x}$ is

$$\mathrm{F}_{n,c}(\boldsymbol{x}) = [\boldsymbol{x} \in \gamma_n \cap \gamma_c]. \tag{2.2}$$

Given a deterministic probabilistic circuit $\mathcal{C}$, for any sum node n and its child c, the associated maximization likelihood estimation (MLE) parameter $\theta_{n,c}$ on a dataset $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^N$ is [KVC14]:

$$\theta_{n,c} = \mathrm{F}_{n,c}(\mathcal{D}) / \mathrm{F}_n(\mathcal{D}), \text{ where } \mathrm{F}_{n,c}(\mathcal{D}) := \sum_{\boldsymbol{x} \in \mathcal{D}} \mathrm{F}_{n,c}(\boldsymbol{x}) \text{ and } \mathrm{F}_n(\mathcal{D}) = \sum_{c \in \mathsf{in}(n)} \mathrm{F}_{n,c}(\mathcal{D}). \tag{2.3}$$

The quantity $\mathrm{F}_{n,c}(\mathcal{D})$ is called the *aggregate circuit flow* of edge (n, c) over dataset $\mathcal{D}$. Intuitively, circuit flows count the number of samples in $\mathcal{D}$ that "activate" an edge.

If determinism is not satisfied, the MLE solution will not have a closed-form expression. Instead, we utilize Expectation-Maximization (EM) algorithm. Specifically, we first randomly initialize the parameters $\boldsymbol{\theta}^0$, and then, at each iteration t, we compute aggregated circuit flows $\mathrm{F}_{n,c}(\mathcal{D})$ and $\mathrm{F}_n(\mathcal{D})$ over dataset $\mathcal{D}$ based on $\boldsymbol{\theta}^t$ in the E step, and then estimate new parameters via the closed-form MLE formulation $\theta_{c|n}^{t+1} = \mathrm{F}_{n,c}(\mathcal{D}) / \mathrm{F}_n(\mathcal{D})$ in the M step [KF09, CDB21].

2.5 Probabilistic Circuit Structures

In this section, we discuss several model architectures of probabilistic circuits.

Chow-Liu Trees. A Chow-Liu tree (CLT) [CL68] is a tree-shaped Bayesian network. The classic Chow-Liu algorithm learns a CLT from data by running a maximum spanning tree algorithm over a complete graph induced by the pairwise mutual information matrix over variables as estimated from data. These MI estimates are used to compute parameters and can be smoothed by adding a Laplace correction factor. CLTs guarantee to encode the best tree model in terms of KL divergence with the data distribution.

Strudel. STRUDEL [DVB20, DVB22] learns a deterministic probabilistic circuit by first transforming a CLT into a PC and then performing a heuristic search guided by log-likelihoods to edit model structures.

Hidden Chow-Liu Trees. Hidden Chow-Liu trees (HCLTs) [LB21] are constructed by adding hidden variables in CLTs. We first learn a CLT over random variable $\mathbf{X} = \{X_i\}_{i=1}^N$, and then modify it through the following steps. Specifically, we first introduce a set of N latent variables $\mathbf{Z} = \{Z_i\}_{i=1}^N$; next we replace all observed variables in the CLT with its corresponding latent variable (i.e., $\forall i$, X_i is replaced by Z_i); finally, we add an edge from every latent variable to its corresponding observed variable (i.e., $\forall i$, add an edge $Z_i \rightarrow X_i$).

Models	Logic Circuits	Probabilistic Circuits	Pairs of Circuits
Algorithms	forward & backward traversal	likelihoods, marginals, MAP	multiply
	smooth, condition, split, merge	(conditional) sampling	KL-divergence
	(weighted) model counting	MLE/ EM parameter learning	expectations
	compilation, SAT	hill climbing structure learning	moments

Table 2.1: The list of major functionalities that Juice supports. Many routines benefit from SIMD/GPU parallelization.

The HCLT structure is then compiled into a smooth and decomposable PC that encodes the same probability distribution.

2.6 Efficient Implementations

We also implement the representations, inference, and learning algorithms of probabilistic circuits as an open-source Julia package, which we name as Juice. Table 2.1 summarizes the main compilation, reasoning, and learning functionality implemented.

Design We model probabilistic circuits as linked node structures. Inference routines iterate over the circuit forward or backward, passing results from node to node. Arbitrary inference algorithms can be implemented by providing different lambda functions, corresponding to different computations, to a general-purpose, optimized circuit traversal and propagation infrastructure.

Parallel computing on CPU and GPU A linked node representation is an intuitive data structure for circuits. However, it has the drawback that it makes computations sparse, making it harder to leverage parallelism to speed up computation. To optimize performance during inference and learning, we translate the circuit's DAG into a layered computational graph, starting with the input layer, and each layer only depending on the previous layers.

Since the computations on the nodes in the same layer can be cached in one large vector, we can simultaneously parallelize our computation over the nodes in the layer on the one hand, and training examples or inference task data on the other hand. Additionally, we leverage Julia's multiple dispatch to provide customized kernels to accelerate computation on both CPUs and GPUs (using SIMD and CUDA kernels respectively). Experiments show that CPU parallelism gives significant speed-ups, which even become an order of magnitude faster with GPU parallelism, all using the same underlying data structures.

CHAPTER 3

Learning Sparse Probabilistic Circuits

This chapter introduces an algorithm to scale up the learning of probabilistic circuits, specifically, we exploit the sparsity properties of tractable probabilistic models, which is also well-characterized by their probability semantics.

We begin by reviewing the phenomenon that probabilistic models are inherently sparse and then introduce a model pruning technique that leverages the probabilistic semantics of parameters to learn the sparse structures. Furthermore, by incorporating a growing operation, we are able to effectively learn large yet sparse tractable probabilistic models that achieve state-of-the-art likelihoods on image datasets as well as language modeling datasets.

3.1 Motivation: Probabilistic Circuits are Effectively Sparse

Recent advancements in PC learning and regularization [SSE21, LB21], and efficient implementations [PLV20, MVS19, DKL21] have been pushing the limits of PC's expressiveness and scalability such that they can even match the performance of less tractable deep generative models, including flow-based models and VAEs. However, the performance of PCs plateaus as model size increases. This suggests that to further boost the performance of PCs, simply scaling up the model size does not suffice and we need to better utilize the available capacity. We discover that this might be caused by the fact that the capacity of large PCs is wasted. As shown in Figure 3.1, most parameters in a PC with 2.18M parameters have close-to-zero values, which have little effect on the PC distribution. Since existing PC struc-

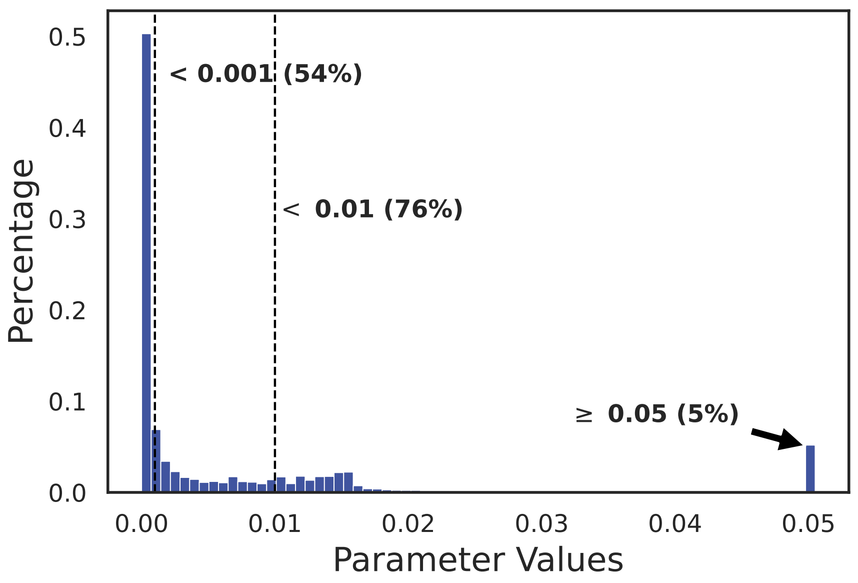


Figure 3.1: Histogram of parameter values for a state-of-the-art PC with 2.18M parameters on MNIST. 95% of the parameters have close-to-zero values.

tures usually have fully-connected parameter layers [LB21, RKG14], this indicates that the parameter values are only sparsely used.

In this chapter, we propose to better exploit the sparsity of large PC models by two structure learning primitives — *pruning* and *growing*. Specifically, the goal of the pruning operation is to identify and remove unimportant sub-networks of a PC. This is done by quantifying the importance of PC parameters w.r.t. a dataset using *circuit flows*, a theoretically-grounded metric that upper bounds the drop of log-likelihood caused by pruning. Compared to L1 regularization, the proposed pruning operator is more informed by the PC semantics, and hence quantifies the global effects of pruning much more effectively. Empirically, the proposed pruning method achieves a compression rate of 80-98% with at most 1% drop in likelihood on various PCs.

The proposed growing operation increases the model size by copying its existing components and injecting noise. In particular, when applied to PCs compressed by the pruning operation, growing produces larger PCs that can be optimized to achieve better performance.

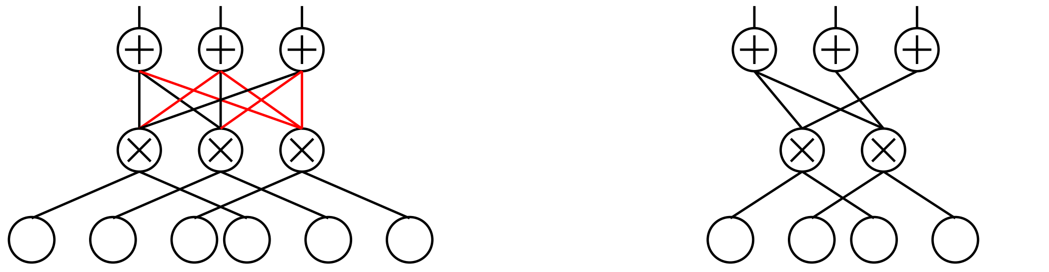
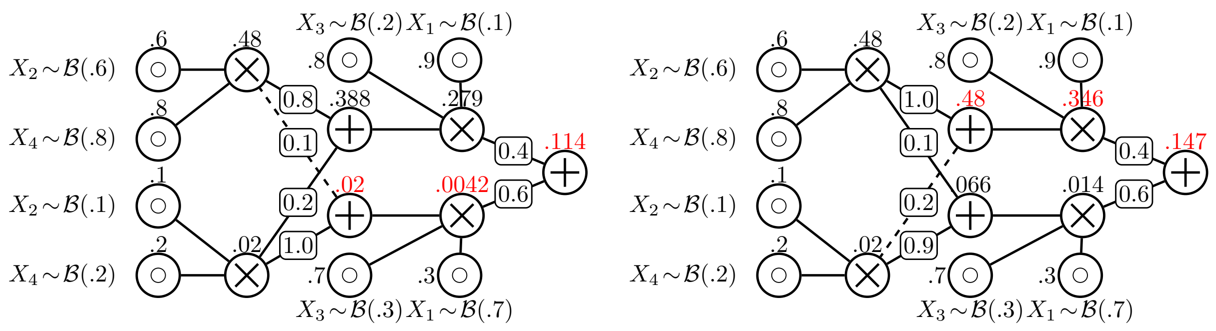

(a) PC with fully connected layers (b) PC after pruning operation

Figure 3.2: A demonstration of the pruning operation where the red edges are pruned.

Applying pruning and growing iteratively can greatly refine the structure and parameters of a PC. Empirically, the log-likelihoods metric can improve by 2% to 10% after a few iterations. Compared to existing PC learners as well as less tractable deep generative models such as VAEs and flow-based models, our proposed method achieves state-of-the-art density estimation results on image datasets including MNIST, EMNIST, FashionMNIST, and the Penn Tree Bank language modeling task.

3.2 Model Compression via Pruning

Figure 3.1 shows that most parameters in a large PC are very close to zero. Given that these parameters are weights associated with mixture (sum unit) components, the corresponding edges and sub-circuits have little impact on the sum unit output. Hence, by pruning away these unimportant components, it is possible to significantly reduce model size while retaining model expressiveness. Figure 3.2b illustrates the result of pruning five (red) edges from the PC in Figure 3.2a. Given a PC and a dataset, our goal is to efficiently identify a set of edges to prune, such that the log-likelihood gap between the pruned PC and the original PC on the given dataset is minimized.


$X_2 \sim \mathcal{B}(.6)$
$X_4 \sim \mathcal{B}(.8)$
$X_2 \sim \mathcal{B}(.1)$
$X_4 \sim \mathcal{B}(.2)$
$X_3 \sim \mathcal{B}(.2)$
$X_1 \sim \mathcal{B}(.1)$
$X_3 \sim \mathcal{B}(.3)$
$X_1 \sim \mathcal{B}(.7)$
.6
.8
.1
.2
.48
.02
.8
.9
.7
.3
0.8
0.1
0.2
1.0
.388
.02
.279
.0042
0.4
0.6
.114
1.0
0.1
0.2
0.9
.48
.066
.346
.014
.147

(a) ePARAM removes the edge with $\theta = 0.1$ (b) eFLOW removes the edge with $\theta = 0.2$

Figure 3.3: A case study comparing pruning heuristics (ePARAM and eFLOW) given instance $\{X_1 = 0, X_2 = 1, X_3 = 0, X_4 = 1\}$. The pruned edges are dashed and parameters are re-normalized. Compared to the likelihood computed in Figure 2.1, the changed likelihoods are in red, showing that pruning by flows results in a less likelihood decrease.

Pruning by parameters. The parameter value statistics in Figure 3.1 suggest that a natural criterion is to prune edges by the magnitude of their corresponding parameter. This leads to the ePARAM (edge parameters) heuristic, which selects the set of edges with the smallest parameters. However, edge parameters themselves are insufficient to quantify the importance of inputs to a sum unit in the entire PC's distribution. The parameters of a sum unit are normalized to be 1 so they only contain local information about the mixture components. Specifically, $\theta_{c|n}$ merely defines the relative importance of edge (n, c) in the conditional distribution represented by its corresponding sum unit n, not the joint distribution of the entire PC. Figure 3.3a illustrates what happens when the edge with the smallest parameter is pruned from the PC in Figure 2.1.

However, as shown in Figure 3.3b, pruning another edge delivers better likelihoods as it accounts more for the "global influence" of edges on the PC's output. This global influence is highly related to the probabilistic "circuit flow" semantics of PCs. We will introduce circuit flows later in this section, along with their corresponding heuristics eFLOW. Before that, we first introduce an intermediate concept based on the notion of the generative significance of

Algorithm 1: PC sampling

```
Input  : a PC representing joint probability $\Pr_{\mathcal{C}}(\mathbf{X})$
Output : an instance $\boldsymbol{x}$ sampled from $\Pr_{\mathcal{C}}$
1 Function SAMPLE(n)
2     if n is a an input unit then
3         $f_n(X) \leftarrow$ univariate distribution of $n$; return sample $x \sim f_n(X)$
4     else if n is a product unit then
5         $\boldsymbol{x}_c \leftarrow$ SAMPLE($c$) foreach $c \in \mathsf{in}(n)$; return Concatenate($\{\boldsymbol{x}_c\}_{c \in \mathsf{in}(n)}$)
6     else n is a sum unit
7         sample an input $c^*$ proportional to $\{\theta_{c|n}\}_{c \in \mathsf{in}(n)}$; return SAMPLE($c^*$)
8 return SAMPLE($r$) where $r$ is the root of PC $\mathcal{C}$
```

probabilistic circuits.

Pruning by generative significance. A more informed pruning strategy needs to consider the global impact of edges on the distribution represented by the output of the PC. To achieve this, instead of viewing the distribution $\Pr_{\mathcal{C}}$ in a feedforward manner following Equation 2.1, we quantify the significance of a unit or edge by the probability that it will be "activated" when drawing samples from the PC. Indeed, if the presence of an edge is hardly ever relevant to the generative sampling process, removing it will not significantly affect the PC's distribution.

Algorithm 1 shows how to draw samples from a PC distribution through a recursive implementation: (1) for an input unit n defined on variable X (line 3), the algorithm randomly samples value x according to its input univariate distribution; (2) for a product unit (line 5), by decomposability its children have disjoint scope, thus we draw samples from all input units and then concatenate the samples together; (3) for a sum unit n (line 7), by smoothness its children have identical scope, thus we first randomly sample one of its input units according to the categorical distribution defined by sum parameters $\{\theta_{c|n} : c \in \mathsf{in}(n)\}$, and then sample from this input unit recursively. Besides actually drawing samples from the

PC, we can also compute the probability that n will be visited during the sampling process. This provides a good measure of the importance of unit n to the PC distribution as a whole, which we define as the *top-down probability.*

Definition 7 (Top-down Probability)**.** The top-down probability of each unit n in a PC with parameters $\boldsymbol{\theta}$ is defined recursively as follows, assuming alternating sum and product layers:

$$q(n;\boldsymbol{\theta}) := \begin{cases} 1 & \text{if } n \text{ is the root unit,} \\ \sum_{m \in \mathsf{out}(n)} q(m;\boldsymbol{\theta}) & \text{if } n \text{ is a sum unit,} \\ \sum_{m \in \mathsf{out}(n)} \theta_{n|m} \cdot q(m;\boldsymbol{\theta}) & \text{if } n \text{ is a product unit,} \end{cases}$$

where $\mathsf{out}(n)$ are the units that take n as input in the feedforward computation. Moreover, the top-down probability of a sum edge (n, c) is defined as $q(n, c;\boldsymbol{\theta}) = \theta_{c|n} \cdot q(n;\boldsymbol{\theta})$.

The top-down probability of the root is always 1; a product unit passes its top-down probability to all its inputs, and a sum unit distributes its top-down probability to its inputs proportional to the corresponding edge weights. Therefore, the top-down probability of a non-root unit is summing over all probabilities it receives from its outputs.

The top-down probability of all PC units and sum edges can be computed in a single backward pass over the PC's computation graph. Following the intuition that the top-down probability defines the probability that units will be visited during the sampling process, pruning edges with the smallest top-down probability constitutes a reasonable pruning strategy.

Pruning by circuit flows. The top-down probability $q(n;\boldsymbol{\theta})$ represents the probability of reaching unit n in an unconditional random sampling process. Despite its ability to capture global information of PC parameters, the top-down probability is not tailored to a specific dataset. Therefore, to further utilize the dataset information, we can measure the probability of reaching certain units/edges in the sampling process *conditioning on some instance $\boldsymbol{x}$ being sampled.* To bridge this gap, we define circuit flow as a sample-dependent version of the top-down probability.

Definition 8 (Circuit Flow[1])**.** For a given PC with parameters $\boldsymbol{\theta}$ and example $\boldsymbol{x}$, the circuit flow of unit n on example $\boldsymbol{x}$ is the probability that n will be visited during the sampling procedure conditioned on $\boldsymbol{x}$ being sampled. This can be computed recursively as follows, assuming alternating sum and product layers:

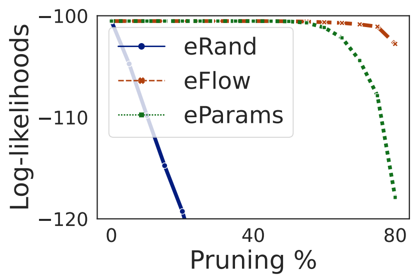
$$\mathrm{F}_n(\boldsymbol{x}) = \begin{cases} 1 & \text{if } n \text{ is the root unit,} \\ \sum_{m \in \mathsf{out}(n)} \mathrm{F}_m(\boldsymbol{x}) & \text{if } n \text{ is a sum unit,} \\ \sum_{m \in \mathsf{out}(n)} \frac{\theta_{n|m} \cdot \mathrm{Pr}_n(\boldsymbol{x})}{\mathrm{Pr}_m(\boldsymbol{x})} \cdot \mathrm{F}_m(\boldsymbol{x}) & \text{if } n \text{ is a product unit.} \end{cases}$$

Similarly, the edge flow $\mathrm{F}_{n,c}(\boldsymbol{x})$ on sample $\boldsymbol{x}$ is defined by $\mathrm{F}_{n,c}(\boldsymbol{x}) = \theta_{c|n} \cdot \mathrm{Pr}_c(\boldsymbol{x})/\mathrm{Pr}_n(\boldsymbol{x}) \cdot \mathrm{F}_n(\boldsymbol{x})$. We further define $\mathrm{F}_{n,c}(\mathcal{D}) = \sum_{\boldsymbol{x} \in \mathcal{D}} \mathrm{F}_{n,c}(\boldsymbol{x})$ as the *aggregate edge flow* over dataset $\mathcal{D}$.

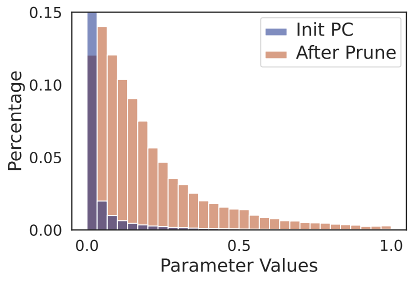
Effectively, we can think of $\theta^{\boldsymbol{x}}_{n|m} := \theta_{n|m} \cdot \mathrm{Pr}_n(\boldsymbol{x})/\mathrm{Pr}_m(\boldsymbol{x})$ as the posterior probability of component n in the mixture of sum unit m *conditioned on observing sample $\boldsymbol{x}$*. Then, circuit flow is the top-down probability under this $\boldsymbol{\theta}^{\boldsymbol{x}}$ reparameterization of the circuit: $\mathrm{F}_n(\boldsymbol{x}) = q(n; \boldsymbol{\theta}^{\boldsymbol{x}})$ and $\mathrm{F}_{n,c}(\boldsymbol{x}) = q(n, c; \boldsymbol{\theta}^{\boldsymbol{x}})$.

Circuit flow $\mathrm{F}_n(\boldsymbol{x})$ defines the probability of reaching unit n in the top-down sampling procedure of Algorithm 1, given that the sampled instance is $\boldsymbol{x}$. Therefore, edge flow $\mathrm{F}_{n,c}(\boldsymbol{x})$ is a natural metric of the importance of edge (n, c) given $\boldsymbol{x}$. Intuitively, the aggregate circuit flow measures how many expected samples "flow" through certain edges. We write eFlow to refer to the heuristic that prunes edges with the smallest aggregate circuit flow.

[1]We also introduce "circuit flow" or "expected circuit flow" in the context of parameter learning [CDB21, LB21, DVB20] in Chapter 2, without observing the connection to sampling. We contribute its more intuitive sampling semantics here.



(a) Comparison of heuristics eRand, eParam, and eFlow. Heuristic eFlow can prune up to 80% of the parameters without much loglikelihoods decrease.



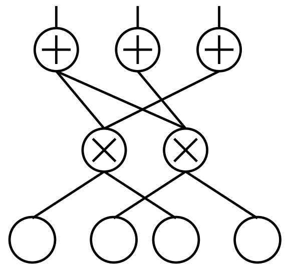
(b) Histogram of parameters before (the same as in Figure 3.1) and after pruning. The parameter values take higher significance after pruning.

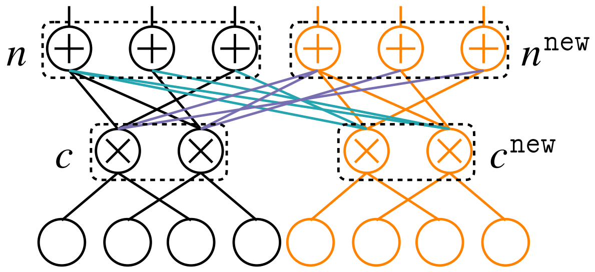
Figure 3.4: Empirical evaluation of the pruning operation.

Empirical Analysis. Figure 3.4a compares the effect of pruning heuristics eParam, eFlow, as well as an uninformed strategy, prune randomly, which we denote as eRand. It shows that both eParam and eFlow are reasonable pruning strategy, however, as we increase the percentage of pruned parameters, eFlow has less log-likelihoods drop compared with eParam. Using eFlow heuristics we can pruning up to 80% of the parameters without much log-likelihoods drop. As shown in Figure 3.4b, the parameter distribution is more balanced after pruning compared to Figure 3.1, indicating a higher significance of each edge. Section 3.4 will provide more empirical results.

3.3 Scalable Structure Learning

If we treat PCs as hierarchical mixtures of components, pruning can be regarded as an implicit structure learning step that removes the "unimportant" components for each mixture. However, since pruning only decreases model capacity, it is impossible to get a more expressive PC than the original one. To mitigate this problem, we propose a *growing* operation to increase the capacity of a PC by introducing more components for each mixture. Pruning


(a) PC before growing operation



(b) PC after growing operation.

Figure 3.5: A demonstration of the growing operation. Each unit is doubled, and each parameterized edge is copied 3 times: $(n^{\texttt{new}}, c^{\texttt{new}})$ (orange), $(n^{\texttt{new}}, c)$ (purple), and $(n, c^{\texttt{new}})$ (green).

and growing together define a scalable structure learning algorithm for PCs.

Growing. *Growing* is an operator that increases model size by copying its existing components and injecting noise. As shown in Figure 3.5, the growing operation is applied to units, edges, and parameters respectively: (1) for units, growing operates on every PC unit n and creates another copy $n^{\texttt{new}}$; (2) for edges, the sum edge (n, c) from the original PC (Figure 3.5a) are copied three times to the grown PC (Figure 3.5b): from new parent to new child $(n^{\texttt{new}}, c^{\texttt{new}})$, from old parent to new child $(n, c^{\texttt{new}})$, and from new parent to old child $(n^{\texttt{new}}, c)$; product edges are added to connect the copied version of a product unit and its copied inputs; (3) a new parameter $\theta^{\texttt{new}}_{c|n}$ is a noisy copy of an old parameter $\theta_{c|n}$, that is $\theta^{\texttt{new}}_{c|n} \leftarrow \epsilon \cdot \theta_{c|n}$ where $\epsilon \sim \mathcal{N}(1, \sigma^2)$ and σ^2 controls the Gaussian noise variance. Gaussian noise is added to the copied parameters to ensure that after we apply the growing operation, parameter learning algorithms can find diverse parameters for different copies. After a growing operation, the PC size is 4 times the original PC size.

Structure Learning through Pruning and Growing. The proposed pruning and growing algorithms can be applied iteratively to refine the structure and parameters of an initial

PC. Specifically, since the growing operator increases the number of PC parameters by a factor of 4, applying growing after pruning 75% of the edges from an initial PC keeps the number of parameters unchanged. We propose a joint structure and parameter learning algorithm for PCs that uses these two operations. Specifically, starting from an initial PC, we apply 75% pruning, growing, and parameter learning iteratively until convergence. We utilize HCLTs [LB21] as initial PC structure as it has the state-of-the-art likelihood performance. Note that this structure learning pipeline can be applied to any PC structure.

Parameter Estimation. We use a stochastic mini-batch version of Expectation-Maximization optimization [CFB20]. Specifically, at each iteration, we draw a mini-batch of samples $\mathcal{D}_B$, compute aggregated circuit flows $\mathrm{F}_{n,c}(\mathcal{D}_B)$ and $\mathrm{F}_n(\mathcal{D}_B)$ of these samples (E-step), and then compute new parameter $\theta_{c|n}^{\texttt{new}} = \mathrm{F}_{n,c}(\mathcal{D}_B) / \mathrm{F}_n(\mathcal{D}_B)$. The parameters are then updated with learning rate α: $\theta^{t+1} \leftarrow \alpha\theta^{\texttt{new}} + (1-\alpha)\theta^t$ (M-step). Empirically this approach converges faster and is better regularized compared to full-batch EM.

3.4 Experiments

We now evaluate our proposed method pruning and growing on two different sets of density estimation benchmarks: (1) the MNIST-family image generation datasets including MNIST [LCB10], EMNIST [CAT17], and FashionMNIST [XRV17]; (2) the character-level Penn Tree Bank language modeling task [MMS93].

Section 3.4.1 first reports the best results we get on image datasets and language modeling tasks via the structure learning procedure proposed in Section 3.3. Section 3.4.2 then shows the effect of pruning and growing operations via two detailed experimental settings. It studies two different constrained optimization problems: finding the smallest PC for a given likelihood via model compression and finding the best PC of a given size via structure learning.

Settings. For all experiments, we use hidden Chow-Liu Trees (HCLTs) [LB21] with the number of latent states in $\{16, 32, 64, 128\}$ as initial PC structures. We train the parameters of PCs with stochastic mini-batch EM (cf. Section 3.3). We perform early stopping and hyperparameter search using a validation set and report results on the test set. We use mean test set bits-per-dimension (bpd) as the evaluation criteria, where $\mathsf{bpd}(\mathcal{D}, \mathcal{C}) = -\mathcal{LL}(\mathcal{D}, \mathcal{C})/(\log(2) \cdot m)$ and m is the number of features in dataset $\mathcal{D}$.

3.4.1 Density Estimation Benchmarks

Image Datasets. The MNIST-family datasets contain gray-scale pixel images of size 28×28 where each pixel takes values in $[0, 255]$. We split out 5% of training data as a validation set. We compare with two competitive PC learning algorithms: HCLT [LB21] and RatSPN [PVS20], one flow-based model: IDF [HPV19], and three VAE-based methods: BitSwap [KAH19], BB-ANS [TBB18], and McBits [RUS21]. For a fair comparison, we implement RatSPN structures ourselves and use the same training pipeline and EM optimizer as our proposed method. Note that EinsumNet [PLV20] also uses RatSPN structures but with a PyTorch implementation so its comparison is subsumed by comparison with RatSPN. All 7 methods are tested on MNIST, 4 splits of EMNIST and FashionMNIST. As shown in Table 3.1, the best results are bold. We see that our proposed method significantly outperforms all other baselines on all datasets, and establishes new state-of-the-art results among PCs, flows, and VAE models.

Language Modeling Task. We use the Penn Tree Bank dataset with standard processing from [MSD12], which contains around 5M characters and a character-level vocabulary size of 50. The data is split into sentences with a maximum sequence length of 288. We compare with three competitive normalizing-flow-based models: Bipartite flow [TVA19] and latent flows [ZR19] including AF/SCF and IAF/SCF, since they are the only comparable work with non-autoregressive language modeling. As shown in Table 3.2, the proposed method

Table 3.1: Density estimation performance on MNIST-family datasets in test set bpd.

Dataset	Sparse PC (ours)	HCLT	RatSPN	IDF	BitSwap	BB-ANS	McBits
MNIST	**1.14**	1.20	1.67	1.90	1.27	1.39	1.98
EMNIST(MNIST)	**1.52**	1.77	2.56	2.07	1.88	2.04	2.19
EMNIST(Letters)	**1.58**	1.80	2.73	1.95	1.84	2.26	3.12
EMNIST(Balanced)	**1.60**	1.82	2.78	2.15	1.96	2.23	2.88
EMNIST(ByClass)	**1.54**	1.85	2.72	1.98	1.87	2.23	3.14
FashionMNIST	**3.27**	3.34	4.29	3.47	3.28	3.66	3.72

Table 3.2: Character-level language modeling results on Penn Tree Bank in test set bpd.

Dataset	Sparse PC (ours)	Bipartite flow [TVA19]	AF/SCF [ZR19]	IAF/SCF [ZR19]
Penn Tree Bank	**1.35**	1.38	1.46	1.63

outperforms all three baselines.

3.4.2 Evaluating Pruning and Growing

What is the Smallest PC for the Same Likelihood? We evaluate the ability of pruning based on circuit flows to do effective model compression by iteratively pruning a k-fraction of the PC parameters and then fine-tuning them until the final training log-likelihood does not decrease by more than 1%. Specifically, we take pruning percentage k from $\{0.05, 0.1, 0.3\}$. As shown in Figure 3.6, we can achieve a compression rate of 80-98% with negligible performance loss on PCs. Besides, by fixing the number of latent parameters (x-axis) and comparing bpp across different numbers of latent states (legend), we discover that compressing a large PC to a get smaller PC yields better likelihoods compared to directly training an HCLT with the same number of parameters from scratch. This can be explained by the sparsity of compressed PC structures, as well as a smarter way of finding good parameters: learning a better PC with larger size and compressing it down to a smaller one.

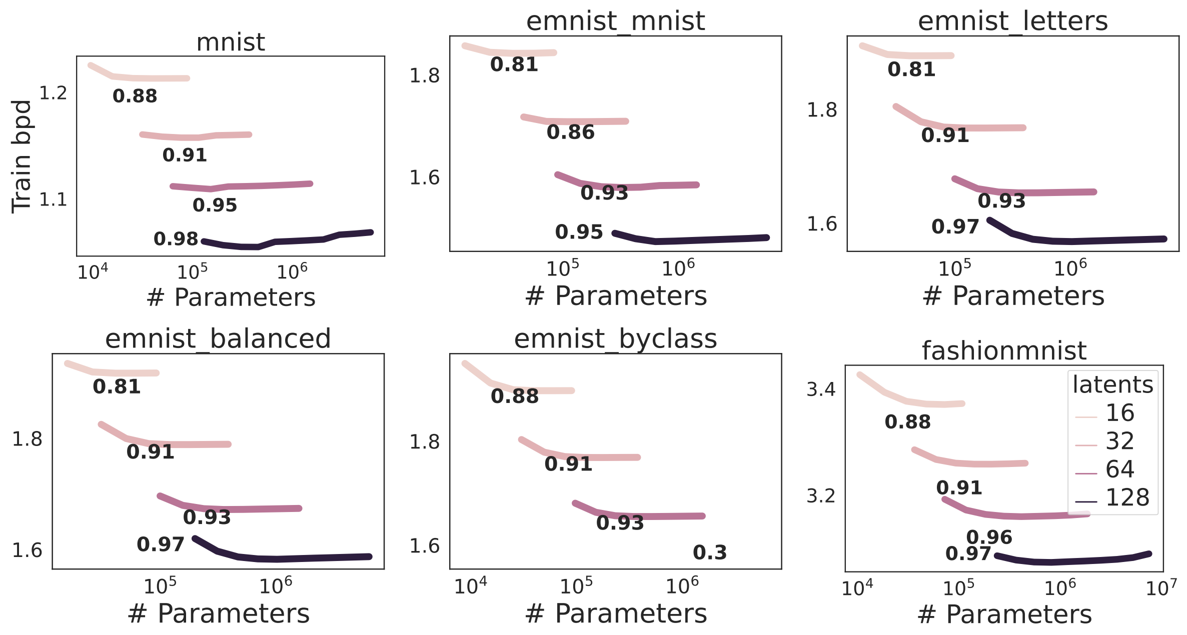


Figure 3.6: Model compression via pruning and finetuning. We report the training set bpd in terms of the number of parameters for different numbers of latent states. For each curve, compression starts from the right and ends at the left; compression rate is annotated next to each curve.

What is the Best PC for the Same Size? We evaluate structure learning that combines pruning and growing as proposed in Section 3.3. Starting from an initial HCLT, we iteratively prune 75% of the parameters, grow again, and fine-tune until meeting the stopping criteria. As shown in Figure 3.7, our method consistently improve the likelihoods of initial PCs for different numbers of latent states among all datasets.

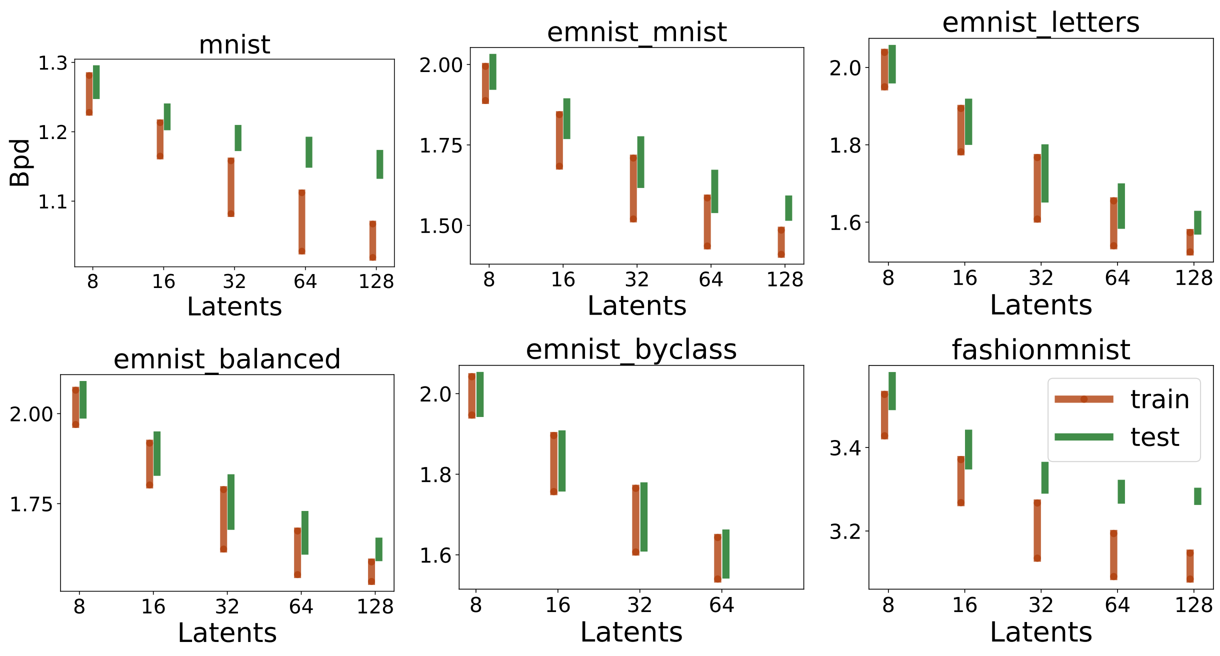


Figure 3.7: Structure learning via 75% pruning, growing and finetuning. We report bpd (y-axis) on both train (red) and test set (green) in terms of the number of latent states (x-axis). For each curve, training starts from the top (large bpd) and ends at the bottom (small bpd).

CHAPTER 4

Modeling Genetic Variation Data

In Chapter 3, we introduce a learning algorithm that exploits the sparsity properties of PCs to learn large yet efficient models. In this section, we will leverage the effectiveness and efficiency of PCs to model genetic variation data and to generate artificial genomes (AGs).

First, we propose to learn PCs that capture the long-range dependencies among Single Nucleotide Polymorphisms (SNPs), which obtain the highest log-likelihood across SNPs chosen across the genome and from a contiguous genomic region. Moreover, we show that the AGs generated by PCs more accurately resemble the source data set in their patterns of allele frequencies, linkage disequilibrium, pairwise haplotype distances, and population structure.

4.1 Background

Generative models of genetic sequence data play a central role in population genomics [Nor19, Wak20]. By modeling dependencies across individuals and sites, these models have empowered genomic analyses such as genotype imputation [MH10], haplotype phasing [BB11], and ancestry inference [MBI19]. Such models also form the basis for programs that simulate artificial genomes (AGs) [Hud02, HS07, EF11, KEM16, BBG21] that, in turn, have played a critical role in testing evolutionary hypothesis, inferring population genetic models, validating empirical results, and benchmarking methods. The ability to accurately and efficiently simulate AGs has been important to foster reproducibility and equity in research: trained models (or data simulated under the models) can be made available without restriction thereby side-stepping privacy restrictions associated with sharing primary genetic data.

The traditional and widely used population genetic approach to simulate AGs relies on the coalescent model [Hud83] which simulates genomes given a demographic history and additional parameter such as mutation and recombination rates and an alternate class of models that aim to directly approach which produces “new data from old”. The coalescent with recombination [GM97] describes the probability of genetic variation in chromosomes sampled across individuals to population genetic parameters (population size, rates of mutation, and recombination) through latent gene genealogies along the genome (with the genealogies varying along the genome due to recombination) [WH99]. While expressive in principle (*i.e.*, able to generate genetic variation AGs with realistic properties), inference under this model is computationally challenging due to the non-Markovian dependence induced among the latent genealogies. As a result, exact computation of the likelihood function under the coalescent with recombination is intractable. This difficulty is limiting as the ability to simulate realistic AGs depends critically on the parameters of the model, such as the demographic history, which, in turn, need to be learned from data. This limitation has motivated investigation into tractable approximations to the coalescent. One class of approaches improve tractability by approximating the coalescent as a Markovian process along the genome [MC05, MW06]. These approximations are the cornerstones of population coalescent-based simulators [KEM16, BBG21]. An alternate class of approximations, exemplified by the product of approximate conditionals (PAC) model and its extensions [LS03], aim to directly model the distribution of genetic variation without invoking a well-defined genealogical process. These models improve tractability by imposing a Markovian assumption which leads naturally to a hidden Markov model (HMM) [RJ86] (while tractable, these models can still be computationally intensive leading to additional approximations [SS06]). The move away from an explicit connection to a genealogical process can limit the evolutionary inferences from such a model. Nevertheless, the PAC models and their variants are used in settings where the goal is to obtain an accurate probability model for the data distribution and have been widely used in applications such as haplotype phasing [SS06, DMZ12], genotype imputation [HFS12, MH10], and ancestry inference [BPS12, PTP09].

Recent advances in deep learning have led to the application of deep generative models to genetic variation [YDO21, BCK21]. While these models are more expressive than a HMM, current deep models proposed for this task (based on GANs [GPM14], variational autoencoders VAEs [KW13], and RBMs [Smo86, Hin02]) are limited in important ways. First, these models do not permit exact probabilistic inference. As a result, it is not possible to compute likelihoods on held-out data for GANs or RBMs while VAEs only allow computation of a lower bound. Importantly, this difficulty precludes the application of these models to tasks such as genotype imputation (all of which involve marginalizing over the joint probability distribution to be able to compute the probability of the observed variables or the conditional probability of the missing variables given the observed variables). Further, these models are challenging to learn due to the complicated cost functions that need to be optimized as a means of learning parameters or hyperparameters. For example, training GANs involve a minimax objective whose optimization can lead to degenerate distributions (termed mode collapse [AAC18]). Learning VAEs requires approximately maximizing a lower bound on the marginal likelihood while learning RBMs involves approximating the gradient of the log-likelihood which is typically achieved by running a Markov Chain Monte Carlo sampling algorithm [Hin02].

4.2 Experiments

In this section, we empirically demonstrate the effectiveness of PCs in terms of modeling genome sequencing data. In order to show the generality of PC structures, we use HCLT and Strudel as introduced in Chapter 2.

4.2.1 Dataset: 1000 Genomes Project

We use 2504 genomes from the 1000 Genomes Project [CFZ16] to evaluate our models and generate artificial genomes (AGs). When analyzing global structure (Section 4.2.3), we use a set of 805 highly differentiated SNPs from across the genome that are a subet of the SNP set identified from Colonna et al [CAC14]. When analyzing local structure (Section 4.2.4), we use a set of 10K SNPs drawn from a single genomic locus on chromosome 15. For all the experiments, we apply a 0.8/0.2 train/test split to phased data. Models are trained on the training set and evaluated on the test set. For every model, we simulate 5000 AGs and compare them to the test set genomes.

4.2.2 Baselines and Evaluation

Baselines To benchmark our model performance in estimating density and simulating artificial genomes, we first compare it to three popular probabilistic graphical models (PGMs) that support tractable likelihood computation: fully-factorized distributions (Indep), higher-order Markov chain models (Markov) and non-homogeneous hidden Markov models (HMM). In order to estimate the parameters of these models, Indep and Markov have closed-form MLE solutions; while for HMM, we perform EM algorithm with random initialization. To facilitate the implementation benefits of PCs, we transform all these PGMs as equivalent PCs for efficient parameter estimation. We also compare against existing neural network methods: (1) generative adversarial networks (GAN) and (2) Restricted Boltzmann machines (RBM) as implemented in [YDO21]. For both neural network baselines, we use the samples generated by the corresponding authors for comparison.[1]

Evaluation criteria We evaluate these models using the following metrics: (1) log-likelihood on test data to assess the capability of each model as a density estimator; (2) summaries of

[1]Note that [YDO21] did not do train/test splits so GAN and RBM are actually trained on train+test.

Table 4.1: Density estimation results in 805 and 10K SNPs data. Averaged training and test log-likelihoods and models sizes (number of parameters in the PCs) for Indep, Markov (order is 10), HMM, CLT, Strudel, and HCLT. The bold values highlight the best averaged test set log-likelihoods.

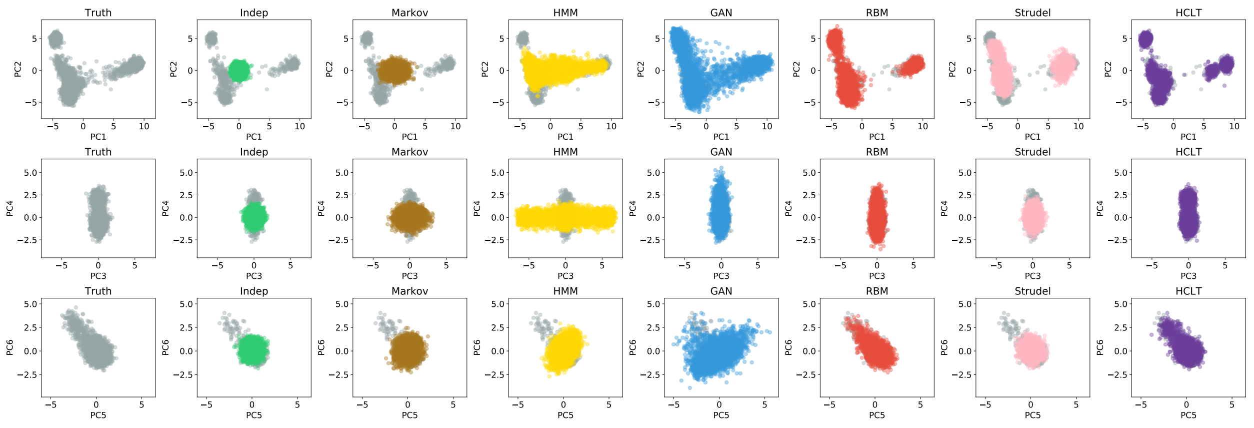
Dataset	**Category**	Indep	Markov	HMM	CLT	Strudel	HCLT
805	**train LL**	-490.73	-433.57	-402.45	-414.68	-402.02	-387.97
	test LL	-491.10	-438.64	-402.50	-415.83	-407.26	**-389.20**
	#params	1.61k	51.26k	231.10k	4.55k	77.80k	61.12k
10K	**train LL**	-2389.69	-626.18	-1192.78	-444.06	-444.03	-282.07
	test LL	-2390.09	-633.14	-1194.72	-456.39	-459.61	**-310.93**
	#params	20.0k	20461.57k	2879.26k	49.0k	2194.32k	5661.95k

AGs sampled from each model that include the top principal components that summarize the dominant axes of variation in the samples; (3) allele frequencies at individual SNPs, which calculate marginal probabilities and act as a one-point estimation; (4) linkage disequilibrium at pairs of SNPs, which calculate pairwise probabilities and act as a two-point estimation.

4.2.3 Reconstructing Global Population Structure

We first compared the ability of different models to represent genetic variation across 805 SNPs sparsely sampled from across the genome. We simulate AGs with Strudel, HCLT and all five baselines (Indep, Markov, HMM, GAN, and RBM) for comparison. We use 2504 diploid genomes (5008 haploid genomes) from the 1000 Genomes Project [CFZ16] as our dataset, and all models are learned on 805 SNPs which are sparsely sampled from across the genome. We tune hyper-parameters on a small split of training data as validation: we use the order of 5 for higher-order Markov chain, and hidden states of 16 for HCLT and HMM.

Table 4.1 shows that Strudel and HCLT learn more accurate probabilistic models than



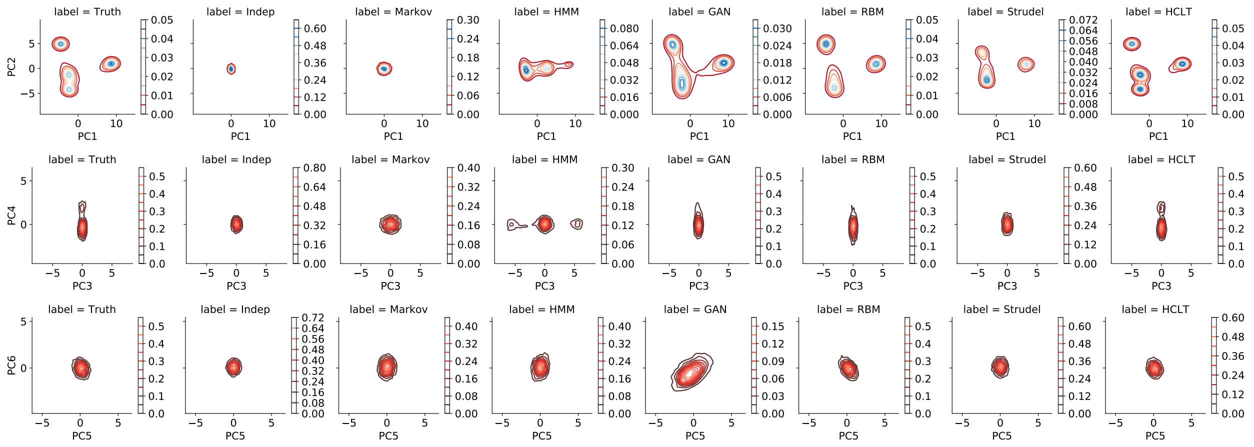


Figure 4.1: Principal components analysis for models trained on the 805 dataset. The first six axes of a single PCA applied to the test set of the 805 dataset (gray) and AGs generated via INDEP (green), MARKOV (brown), HMM (orange), GAN (blue), RBM (red), STRUDEL (pink), and HCLT (purple). The test set contains 961 haplotypes, and each model generate 5000 haplotypes as AGs. The top three panels plot the samples while the bottom three panels show the density plot of these samples.

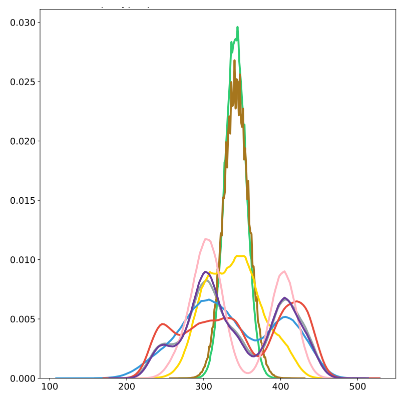
fully-factorized distributions, Markov chains and HMMs as measured by their log-likelihood on the test set. Additionally, the PC models are relatively lightweight: they have similar sizes compared to the adopted Markov chains. Note that we do not compare with GAN and

Table 4.2: Evaluating the performance in preserving population structure using principal component analysis. Wasserstein 2D distances between the PCA representations of real versus generated individuals. (within, between): Wasserstein distance between the pairwise Euclidean distances of haploid genomes within a single dataset or between the real and generated individuals. r^2: Squared Pearson correlations between real and generated LD across all pairs of samples. We denote Real for the testset. Bolded values indicate the best among all compared models.

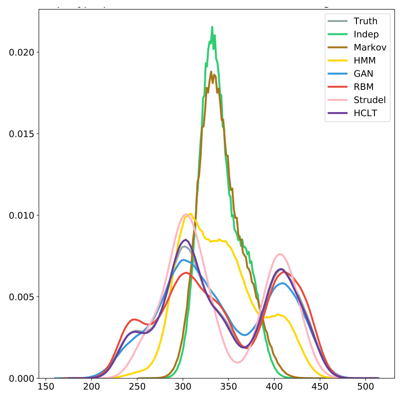
Dataset		Real	Indep	Markov	HMM	GAN	RBM	Strudel	HCLT
805	PCA1-2	0.0010	0.2272	0.1666	0.0758	0.0040	0.0089	0.0065	**0.0015**
	PCA3-4	0.0015	0.0082	0.0280	0.0588	0.0175	0.0045	0.0107	**0.0020**
	PCA5-6	0.0013	0.0019	0.0213	0.0270	0.0043	0.0017	0.0017	**0.0013**
	within	0.98	43.92	42.10	24.99	4.96	6.96	9.25	**2.41**
	between	0.49	37.17	35.96	20.27	2.34	3.28	5.90	**1.26**
	r^2	0.99	0.67	0.76	0.73	0.95	0.98	0.96	**0.99**
10K	PCA1-2	0.0012	0.1905	0.0881	0.0946	0.0065	0.0144	0.0056	**0.0029**
	PCA3-4	0.0014	0.1655	0.0148	0.0572	**0.0018**	0.0107	0.0037	0.0022
	PCA5-6	0.0013	0.0889	0.0091	0.0169	**0.0014**	0.0063	0.0036	0.0020
	within	1.41	177.86	1223.19	148.65	107.84	29.88	36.69	**21.28**
	between	0.81	128.95	678.85	115.65	44.77	47.61	36.74	**24.51**
	r^2	0.99	0.38	0.66	0.50	0.95	0.94	0.94	**0.96**

RBM since they do not support tractable exact likelihood computation.

We then analyze the quality of AGs generated by all models. Figure 4.1 shows that the AGs generated by Indep and Markov fall within the center of the variation seen in the test samples. The AGs generated by the HMM cover some spaces of PC1 and PC2, whereas GAN and RBM can capture most of the global structure. Regardless, AGs generated by the GAN tend to cover a larger space than the real test data. In contrast, Strudel and HCLT are able to capture more details: the PC1 and PC2 plots of HCLT is almost identical



(a) within each dataset



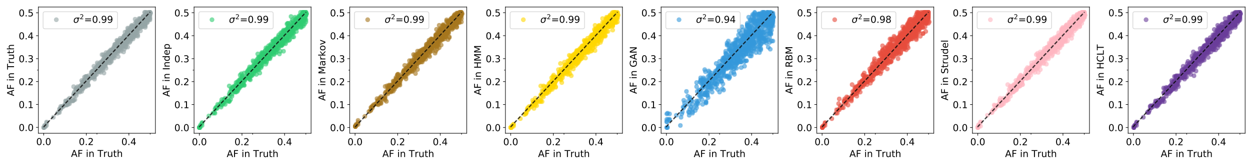
(b) between datasets and ground truth

Figure 4.2: Distribution of haplotypic pairwise Euclidean distances within (4.2a) datasets and between (4.2b) AG datasets and test set from 805 dataset using different models.

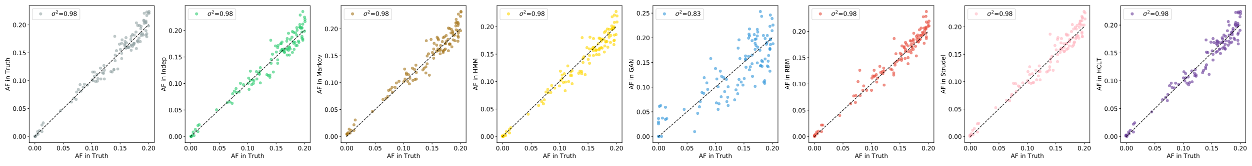
to the ground truth, and the PC3 and PC4 are also well captured.

To quantify the accuracy of the PCs computed from the AGs, we computed Wasserstein distances between the 2D PCA representations of test data versus simulated data (Table 4.2). Wasserstein distances between the 2D PCA representations of test data versus simulated data are lower (closer to 0) for HCLT than for RBM and GAN along every pair of dimensions. We additionally compute the pairwise differences of haploid genomes within a single dataset or between the test dataset and AG datasets. Figure 4.2 shows the distribution of the pairwise difference while Table 4.2 rows 4-5 show the Wasserstein distances between these distributions. GAN, RBM, Strudel and HCLT all capture the three modes in the distribution while the histogram of GAN and RBM are more uniform.

Next, we examined the allele frequencies in the AGs relative to the allele frequencies in the test data. As shown in Figure 4.3, the allele frequencies of Strudel and HCLT are more centered around the diagonal, which indicates that they yield better calibrated probabilities.



(a) Comparison for the whole range.



(b) Comparison with a focus on low frequencies SNPs.

Figure 4.3: Comparison of allele frequency between ground truth genomes from the 805 SNP dataset and the AGs counterparts generated using Indep, Markov, GAN, RBM, Strudel and HCLT models (a) for the whole range and (b) with a focus on low frequencies SNPs. The plot legend σ^2 refers to squared Pearson correlations between ground truth genomes and AGs.

4.2.4 Reconstructing Local Population Structure

To evaluate the ability of PCs to generate genome sequences across a dense set of SNPs from a single genomic region, we applied the Strudel and HCLT learner to a region with 10K SNPs from chromosome15 in the 1000 Genomes data. The log-likelihoods in Table 4.1 show that Strudel and HCLT can still deliver expressive PC models with moderate sizes. This suggests that these PC learners can handle high-dimensional genetic datasets and accurately capture long-range correlations. Note that although HMM has similar likelihoods as Strudel in 805 SNPs dataset, the performance is much worse in the 10K SNPs dataset, which shows that HMM is hard to capture long-range correlations for it is a linear model. PCA results and comparison of pairwise distances of AGs (Table 4.2) show that the AGs generated by HCLT are most similar to the test data on the most important principal components and in terms of pairwise distances.

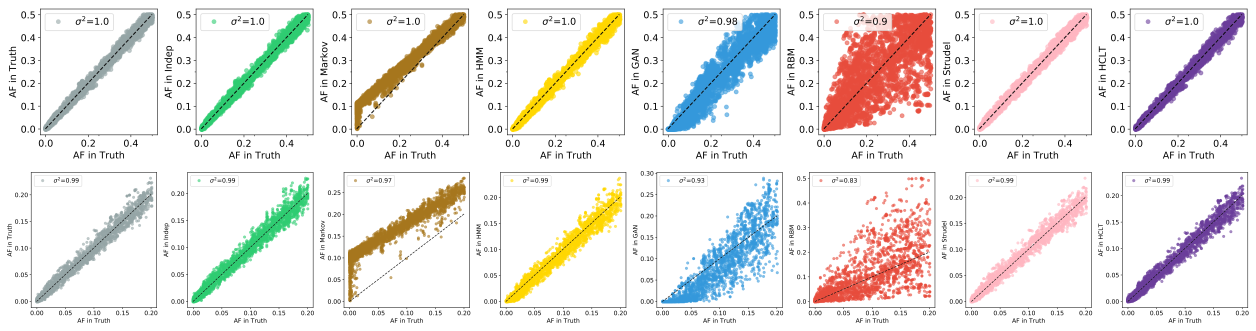


Figure 4.4: Allele frequency comparison of corresponding SNPs between ground truth genomes from 10K dataset and AGs counterparts generated using Indep, Markov, GAN, RBM, Strudel and HCLT models (a) for the whole range and (b) zoomed to low frequencies.

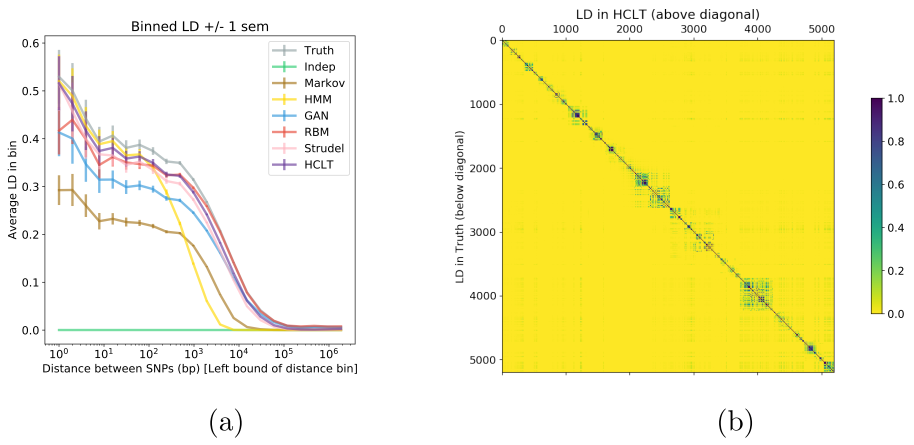


Figure 4.5: Linkage disequilibrium analysis. (4.5a) LD as a function of SNP distance on all methods. (4.5b) Correlation matrices (r^2) of SNPs where the lower triangular parts are in real genomes from 10K dataset and upper triangular parts in AGs generated by HCLT.

Allele frequency analysis in Figure 4.4 shows that RBM and GAN perform poorly especially for low-frequency alleles while Strudel and HCLT still show well-calibrated correlations. Allele frequency analysis can be seen as a first dimension correlation analysis of AGs. Since SNPs from a given genomic region tend to be correlated, we examined patterns of linkage disequilibrium (LD) to assess how the pairwise short and long-range correlations of SNPs can be captured by AGs. The pairwise LD matrix in Figure 4.5b shows that HCLTs

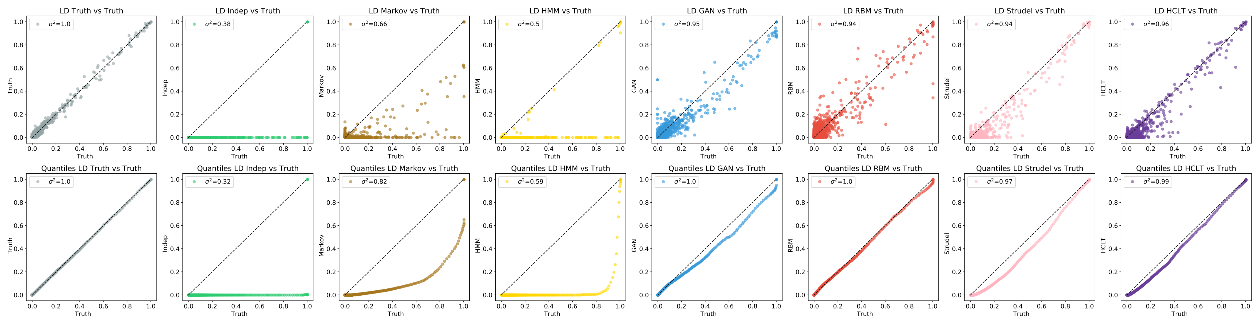


Figure 4.6: Linkage disequilibrium comparison. The first row plots the pairwise LD between pairs of points from AGs vs. real test set for all models on 10K dataset. The second row shows the respective QQ-plots, which illustrate the corresponding quantiles.

accurately capture patterns of LD in this region. Plotting LD as a function of SNP distance in Figure 4.5a demonstrates that HCLT, STRUDEL and HMM better capture better correlation across shorter length scales while all models expect for HMM are accurate at longer length scales. On the other hand, while the HMM accurately captures LD at shorter length scales, it performs poorly across longer length scales. Correlations between real and AGs shown in the last row of Table 4.2 and in Figure 4.6 comes to a similar conclusion that HCLT more accurately captures the distribution of LD across SNPs within the region.

CHAPTER 5

Modeling Natural Language for Controllable Generation

In Chapter 3, we introduce a scalable and efficient learning algorithm for tractable probabilistic models. In this chapter, we demonstrate one usage of tractability in controllable language generation.

We first introduce the task of controllable language generation: sampling from the conditional distribution $\Pr(\text{text}|\alpha)$ with some constraints α. Then we propose to use tractable probabilistic models to impose lexical constraints in autoregressive text generation models. To demonstrate the effectiveness of this framework, we use distilled hidden Markov models, where we *can* efficiently compute $\Pr(\text{text}|\alpha)$, to guide autoregressive generation from GPT2.

5.1 Background

Large pre-trained language models (LMs) [RWC19, LLG20] have achieved remarkable performance on a wide range of challenging language generation tasks such as machine translation [BCB15, LPM15], summarization [LFT15, XD19] and open-domain creative generation [YPW19, TP22]. Nevertheless, many practical language generation applications require fine-grained control of LMs to follow complex lexical constraints (e.g., given a source document, generate a summary that contains certain keywords). The common paradigm for controlling pre-trained LMs is to either finetune them on task-specific datasets or to condition them on certain prompts. However, fine-tuning and prompting are by nature approximate

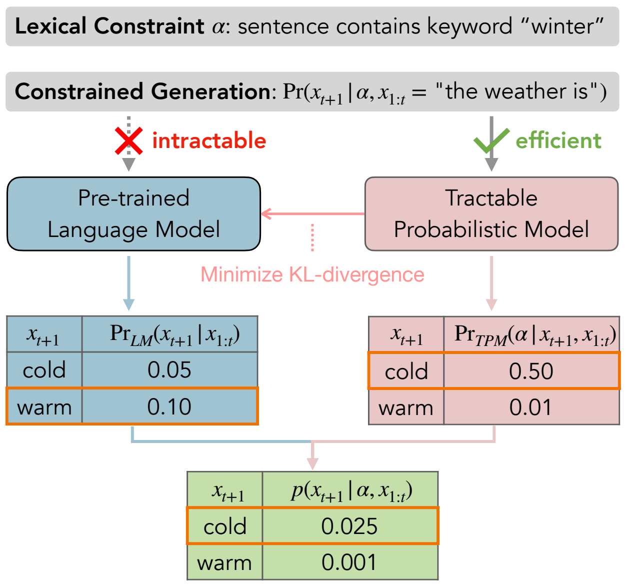

Lexical Constraint α: sentence contains keyword "winter"
Constrained Generation: $\Pr(x_{t+1} | \alpha, x_{1:t} =$ "the weather is")
intractable
efficient
Pre-trained Language Model
Tractable Probabilistic Model
Minimize KL-divergence
x_{t+1} | $\Pr_{LM}(x_{t+1} | x_{1:t})$
cold 0.05
warm 0.10
x_{t+1} | $\Pr_{TPM}(\alpha | x_{t+1}, x_{1:t})$
cold 0.50
warm 0.01
x_{t+1} | $p(x_{t+1} | \alpha, x_{1:t})$
cold 0.025
warm 0.001

Figure 5.1: Given some lexical constraint α that we want our pretrained language models to follow in generation, the conditional distribution $\Pr(x_{t+1}|x_{1:t}, \alpha)$ is often intractable. We propose to control and guide the autoregressive generation process of pre-trained LMs via tractable probabilistic models, which do support efficient computation of $\Pr(x_{t+1}|x_{1:t}, \alpha)$.

solutions and do not guarantee that the desired constraints are satisfied [MLP22, ZLM22]. The major difficulty of constrained language generation lies in the autoregressive nature of LMs: they only model the next token distribution given some prefix $\Pr_{\text{LM}}(x_{t+1}|x_{1:t})$, while the conditional distribution $\Pr_{\text{LM}}(x_{1:n}|\alpha)$ given a constraint α as simple as, e.g., a keyword appearing at the end of a sentence, is often intractable [Rot96].

Aside from language models based on neural architectures, one line of research in machine learning focuses on the development of *tractable probabilistic models* (TPMs) [PD11, KT12, CVB20, ZJB21]. TPMs model joint probability distributions and allow for efficient conditioning on various families of logical constraints [KVC14, CBD15, BDC15]. In this chapter, we propose **GeLaTo** (**Ge**nerating **La**nguage with **T**ractable C**o**nstraints), where we use TPMs to impose lexical constraints in autoregressive text generation. Given a pre-trained autoregressive LM $\Pr_{\text{LM}}$, e.g., GPT3 [BMR20], our goal is to generate text effectively following the conditional distribution $\Pr_{\text{LM}}(x_{1:n}|\alpha)$ for arbitrary lexical constraints α. As illustrated in Figure 5.1, our proposed framework consists of two major components: (1) we train a TPM $\Pr_{\text{TPM}}$ via maximum likelihood estimation (MLE) on samples drawn from $\Pr_{\text{LM}}$, which is equivalent to minimizing the KL-divergence between $\Pr_{\text{TPM}}$ and $\Pr_{\text{LM}}$; then (2) at generation time, we compute $\Pr_{\text{TPM}}(x_{t+1}|x_{1:t}, \alpha)$ efficiently and combine it with $\Pr_{\text{LM}}(x_{t+1}|x_{1:t})$ to approximate $\Pr_{\text{LM}}(x_{t+1}|x_{1:t}, \alpha)$ for reliable control. Note that we assume nothing about the lexical constraint α as we train $\Pr_{\text{TPM}}$, which means that the TPM does not need to be re-trained for different types of constraints: given a trained TPM that approximates $\Pr_{\text{LM}}$ well enough, we can use it to impose any lexical constraints α, as long as $\Pr_{\text{TPM}}(.|\alpha)$ can be efficiently computed.

Throughout this chapter, we use hidden Markov models (HMMs) [RJ86] as an example TPM to demonstrate the effectiveness of GeLaTo. Specifically, (1) we show that, when trained as probabilistic circuits [CVB20, LZB23], HMMs can approximate the GPT2-large model fine-tuned on downstream tasks well enough and (2) we propose a dynamic programming algorithm that efficiently computes conditional probabilities $\Pr_{\text{HMM}}(\cdot|\alpha)$, for αs that encode constraints as conjunctive normal forms (CNFs):

$$(I(w_{1,1}) \vee \cdots \vee I(w_{1,d_1})) \wedge \cdots \wedge (I(w_{m,1}) \vee \cdots \vee I(w_{m,d_m}));$$

here each $w_{i,j}$ is *a string of tokens*, and $I(w_{i,j})$ is an indicator variable denoting whether or not w_{ij} appears in the generated text. Intuitively, constraint α requires that a set of m keywords must appear somewhere in the generated text, in any of their inflections, where each inflection is encoded as a string of one or more tokens. We evaluate the performance of GeLaTo on challenging constrained text generation datasets: CommonGen [LZS20], News [ZWL20], and Yelp!Review [CZZ19]. GeLaTo not only achieves state-of-the-art generation quality but also guarantees that the constraints are satisfied 100%; for both unsupervised and supervised settings, GeLaTo beats strong baselines belonging to different families of constrained generation approaches by a large margin.

Our study demonstrates the potential of TPMs in controlling large language models and motivates the development of more expressive TPMs.

5.2 Guiding Autoregressive Generation with Probabilistic Circuits

In this section, we present the general GeLaTo framework for guiding autoregressive generation with tractable probabilistic models. Throughout this paper, we use uppercase letters X_t for random variables and lowercase letters x_t for their assignment.

Let $\Pr_{\text{LM}}(x_{1:n})$ be the distribution of an autoregressive LM (e.g., GPT) over n tokens and α a lexical constraint defined over $X_{1:n}$; our goal is to generate from the following conditional distribution:

$$\Pr_{\text{LM}}(x_{1:n}|\alpha) = \prod\nolimits_t \Pr_{\text{LM}}(x_{t+1}|x_{1:t}, \alpha)$$

Though $\Pr_{\text{LM}}(x_{t+1}|x_{1:t}, \alpha)$ is intractable, we can assume that $\Pr_{\text{TPM}}(x_{t+1}|x_{1:t}, \alpha)$ can be efficiently computed.

The first step of GeLaTo is to train our TPM model such that $\Pr_{\text{TPM}}$ approximates $\Pr_{\text{LM}}$ as well as possible. We train the TPM model via maximum likelihood estimation (MLE) on data drawn from $\Pr_{\text{LM}}$, that is, we maximize

$$\mathbb{E}_{x_{1:n} \sim \Pr_{\text{LM}}} \log \Pr_{\text{TPM}}(x_{1:n}),$$

which effectively minimizes their KL-divergence:

$$\begin{aligned}
&D_{\text{KL}}(\Pr_{\text{LM}} \parallel \Pr_{\text{TPM}}) \\
&= \mathbb{E}_{x_{1:n} \sim \Pr_{\text{LM}}} \log \Pr_{\text{LM}}(x_{1:n}) - \mathbb{E}_{x_{1:n} \sim \Pr_{\text{LM}}} \log \Pr_{\text{TPM}}(x_{1:n})
\end{aligned}$$

With the recent development of scaling up TPMs [CR20, DLB22, LZB23], we show in Section 5.3 that it is possible to train TPMs as good enough approximations of LMs.

Now given some TPM as a good enough approximation for the LM that we want to generate from, we combine both models for constrained generation, where the TPM is responsible for providing guidance on incorporating lexical constraints and LM responsible for generating fluent texts. To derive our formulation, in addition to lexical constraint α, we assume that there exists some "quality" constraint β such that $\Pr_{\text{TPM}}(|\beta)$ is even closer to $\Pr_{\text{LM}}$; intuitively we interpret β as some constraint characterizing the high-quality (fluent & grammatical) sentences that are likely to be sampled from our base LM $\Pr_{\text{LM}}$. Hence, in order to generate a high-quality sentence satisfying some lexical constraint α, we generate from

$$\Pr_{\text{TPM}}(x_{1:n}|\alpha, \beta) = \prod\nolimits_t \Pr_{\text{TPM}}(x_{t+1}|x_{1:t}, \alpha, \beta);$$

in particular, in addition to the assumption that $\Pr_{\text{TPM}}(\cdot|\beta)$ is a good enough approximation for $\Pr_{\text{LM}}$, we also assume the *key independence assumption*: α and β are conditionally independent given $x_{1:t+1}$. By applying Bayes rule, it follows from our assumptions that:

$$\begin{aligned}
&\Pr_{\text{TPM}}(x_{t+1}|x_{1:t}, \alpha, \beta) \\
&\quad \propto \Pr_{\text{TPM}}(\alpha|x_{1:t+1}, \beta) \cdot \Pr_{\text{TPM}}(x_{t+1}|x_{1:t}, \beta) \\
&\quad \propto \Pr_{\text{TPM}}(\alpha|x_{1:t+1}) \cdot \Pr_{\text{LM}}(x_{t+1}|x_{1:t}).
\end{aligned}$$

Now we examine whether our key independence assumption holds for the *unsupervised* and *supervised* settings.

Unsupervised setting. In the unsupervised setting, we assume that the base pre-trained LM is *not* fine-tuned given task-specific supervision; that is, $\Pr_{\text{LM}}$ is not fine-tuned to generate texts satisfying α provided as input, but is possibly fine-tuned or prompted for the purpose of domain adaptation. In this setting, there is no easy way for the "quality" constraint β to obtain any information about the lexical constraint α and our key independence assumption should roughly hold. In other words, satisfying the lexical constraint α should not help or hinder the fluency of the generated sentence according to the pre-trained LM, it merely biases what the sentence talks about. Hence for the unsupervised setting, we generate autoregressively following the next-token distribution defined as:

$$p(x_{t+1}|x_{1:t}, \alpha) \propto \Pr_{\text{TPM}}(\alpha|x_{1:t+1}) \cdot \Pr_{\text{LM}}(x_{t+1}|x_{1:t}). \tag{5.1}$$

This formulation is also adopted in FUDGE [YK21] and NADO [MLP22], which train auxiliary models to approximate $\Pr_{\text{LM}}(\alpha|x_{1:t+1})$; the key difference is that such auxiliary models take α as input during training while our TPM training is unconditional.

Supervised setting. In this setting, we assume that the language model $\Pr_{\text{LM}}$ is fine-tuned in a sequence-to-sequence (seq2seq) manner; that is, during training, α is explicitly supplied to the LM together with some gold sentences: e.g., for keyword-type constraints, the LM is fine-tuned over texts of the form *"weather winter cold = the weather is cold in winter,"* where the prompt *"weather winter cold = "* encodes the constraint that all words before *"="* should be used. In this case, our key independence assumption no longer holds because $\Pr_{\text{LM}}$ is already trained to satisfy the lexical constraint α, which is provided as part of the prefix $x_{1:t+1}$. Hence for the supervised setting, we adopt an alternative formulation by viewing $\Pr_{\text{TPM}}(x_{t+1}|x_{1:t}, \alpha)$ and $\Pr_{\text{LM}}(x_{t+1}|x_{1:t})$ as classifiers trained for the same task yet with different biases; by [SBF14], if we assume that each model predicts the true logits up to additive Gaussian noise, then the most likely logits can be found by taking a geometric mean of the models. Hence, in the supervised setting, we generate autoregressively following the next-token distribution defined as their weighted geometric mean [Hin02, GE18]:

$$p(x_{t+1}|x_{1:t}, \alpha) \propto \Pr_{\text{TPM}}(x_{t+1}|x_{1:t}, \alpha)^{w} \cdot \Pr_{\text{LM}}(x_{t+1}|x_{1:t})^{1-w}; \tag{5.2}$$

here $w \in (0, 1)$ is a hyper-parameter to be tuned.

To summarize, GeLaTo consists of two major steps: (1) distillation: we train a TPM on samples drawn from the pre-trained LM via MLE to effectively minimize the KL divergence between $\Pr_{\text{LM}}$ and $\Pr_{\text{TPM}}$; (2) probabilistic reasoning: for each step of autoregressive generation, we compute $\Pr_{\text{TPM}}(\cdot|\alpha)$ and generate from the conditional next-token distribution $p(x_{t+1}|x_{1:t}, \alpha)$ defined above. In addition to better generation quality, which we demonstrate in Section 5.3, GeLaTo has two major advantages compared to its counterparts for constrained generation:

- The sentences generated following $p(x_{t+1}|x_{1:t}, \alpha)$ are *guaranteed* to satisfy the lexical constraint α; in autoregressive generation, as we generate the next token x_{t+1}, it follows from the definition that for choices of x_{t+1} such that α *cannot be satisfied*, $\Pr_{\text{TPM}}(x_{t+1}, x_{1:t}, \alpha)$ is 0, thus $p(x_{t+1}|x_{1:t}, \alpha)$ is also 0.

- The TPM training is independent of the lexical constraint α, which is only enforced at inference time; it immediately follows that we do not need to re-train the TPM model no matter how α changes; on the other hand, constrained decoding approaches that train auxiliary neural models, e.g., FUDGE and NADO, need to re-train their model for different types of constraints.

Throughout the rest of this paper, we use **hidden Markov models** (HMMs) as example TPMs to demonstrate the practicality and effectiveness of GeLaTo. In the following section, we propose an efficient algorithm for computing $\Pr_{\text{TPM}}(\alpha|x_{1:t+1})$ and $\Pr_{\text{TPM}}(x_{t+1}|x_{1:t}, \alpha)$.

5.3 Experiments: Common Sense Generation

In this section, we demonstrate the effectiveness of GeLaTo on a challenging benchmark for constrained generation: CommonGen [LZS20]. For both unsupervised and supervised settings, GeLaTo achieves state-of-the-art performance in terms of various automatic evaluation metrics including BLEU score while guaranteeing 100% constraint satisfaction.

5.3.1 Dataset and Baselines

CommonGen [LZS20] is a benchmark for constrained generation with lexical constraints: the input of each example consists of three to five concepts (keywords) and the goal is to generate a natural sentence using all concepts; in particular, the given keywords can appear in any order or in any form of inflections in the generated sentences. For example, given *"car snow drive"* as concepts, both *"a man drives a car on a snow covered road"* and *"the car drove through the snow"* are considered acceptable. We also evaluate GeLaTo on the

Yelp!Review [CZZ19] and the News [ZWL20] datasets. Compared to CommonGen, both Yelp!Review and News share similar formats, except that they require all keywords to be generated in the forms as given (i.e. no inflections allowed) and to follow specific orders.

We compare GeLaTo against constrained generation approaches belonging to different families:

InsNet [LMP22] is a class of insertion-based language models [SCT20] that generate text by repeatedly inserting new tokens into the sequence. InsNet guarantees that the keywords appear in the generated sentence by initializing the token sequence as the keywords, arranged in some order.

NeuroLogic (A*esque) Decoding [LWZ21, LWW22] are search-based decoding algorithms; they are inference-time algorithms like beam search and do not use any auxiliary models. Leveraging look-ahead heuristics, NeuroLogic A*esque decoding not only optimizes the probability of the generated sentence but also steers the generation towards satisfying the lexical constraints.

NADO [MLP22] trains an auxiliary neural model approximating the conditional distribution $\Pr(\alpha | x_{1:t}, x_{t+1})$ to guide constrained generation of the base model. As mentioned in Section 5.2, NADO needs to re-train the auxiliary model for different types of α (e.g., ten keywords) while GeLaTo does not need re-training.

5.3.2 Experiemnt Setup

Following the experiment setup of [LWZ21] and [MLP22], we evaluate GeLaTo under both unsupervised and supervised settings, as described in Section 5.2.

Fine-tuning GPT2-large All baselines, except for InsNet, perform generation with GPT2-large [RWC19] as the *base model*. Following prior works [MLP22], we use fine-tuned GPT2-large as base models:

1. Unsupervised Setting: we perform *domain adaptation* (DA) by fine-tuning GPT2-large on all gold (reference) sentences of the training split of CommonGen *without* supplementing the keywords. We fine-tune the model for 1 epoch with learning rate = 1e-6.

2. Supervised Setting: following the template proposed in [LZS20], we fine-tune the GPT2-large model in a *sequence-to-sequence* (seq2seq) manner; in particular we fine-tune the model on sequences of the form *"car snow drive = a car drove through snow"* for 3 epochs with learning rate = 1e-6.

Training HMMs. We use HMMs as an example TPM to enforce lexical constraint in autoregressive generation from GPT2-large. Following Section 5.3, we sample sequences of length 32 from the fine-tuned GPT2-large models and train HMMs with 4096 hidden states to approximate the base model distributions; we train HMMs with the expectation-maximization (EM) algorithm for 40 epochs, and we re-sample 0.2 million examples for each epoch. The HMM models are trained as probabilistic circuits with the *Juice.jl* framework [DKL21] and the training procedure leverages the latent variable distillation technique proposed in [LZB23]; we refer readers to the original papers for more details.

Constraint Formulation. For CommonGen, as described in Section 5.3.1, the goal is to generate a sentence using the given concepts (keywords) and we encode this lexical constraint as a CNF. For example, given the concepts "catch frisbee snow", the lexical constraint can be represented as:

$$\begin{aligned}&[I(\text{catch}) \lor I(\text{caught}) \lor \dots] \\ \land&[I(\text{fr} \oplus \text{is} \oplus \text{bee}) \lor I(\text{fr} \oplus \text{is} \oplus \text{bees}) \lor \dots] \\ \land&[I(\text{snow}) \lor I(\text{snow} \oplus \text{ing}) \lor I(\text{snow} \oplus \text{ed}) \lor \dots];\end{aligned}$$

here each clause encodes the constraint that a keyword has to appear, in any form of its inflections; each literal $I(w)$ indicates the occurrence of a string of tokens w (i.e. keystring), which represents the tokenization of a specific inflection of a keyword and $\oplus$ denotes the concatenation of individual tokens. For the keywords, we use LemmInflect[1] to generate their inflections. We also enforce the constraint that each keystring, whenever it appears in the generated text, is followed by either a space, a comma or an $\langle \text{eos} \rangle$ token.

Decoding. $p(x_{t+1}|x_{1:t}, \alpha)$ defined in Section 5.2 (see Eq. 5.1 and 5.2) induces the conditional distribution $p(x_{1:n}|\alpha) = \prod_t p(x_{t+1}|x_{1:t}, \alpha)$. We adopt beam search to greedily search for $x_{1:n}$ that maximizes $p(x_{1:n}|\alpha)$; we experiment with different beam sizes: 16, 32, 64 and 128. Finally, we re-rank all beams generated by beam search by their log-likelihood given by the domain-adapted GPT2-large model and select the top beam.

Metrics. We evaluate the quality of generation via human evaluation and some commonly used automatic metrics including ROUGE [LH03], BLEU [PRW02], CIDEr [VLP15], and SPICE [AFJ16]. In addition to generation quality, we also measure the constraint satisfaction performance via *coverage*, the average percentage of concepts presented in the generated sentences and *success rate*, the percentage of generated sentences that perfectly satisfy the constraints.

5.3.3 Results and Analysis

Main evaluation results are presented in Table 5.1. GeLaTo outperforms all baselines in both unsupervised and supervised settings by a large margin, achieving not only significantly higher BLEU and ROUGE scores but also 100% constraint satisfaction. The unsupervised setting is more challenging given that the base model is never trained with task-specific supervision; despite this, GeLaTo achieves 30.3 BLEU score in the *unsupervised* setting, while NADO (the best performing baseline) obtains 30.8 BLEU score in the *supervised* setting. To provide more insight into GeLaTo, we also conduct the following ablation studies.

Generation Quality vs. Approximation Performance. As discussed in Section 5.2,

[1] `https://github.com/bjascob/LemmInflect`

Method	*Generation Quality*						*Constraint Satisfaction*			
	ROUGE-L		BLEU-4		CIDEr		Coverage		Success Rate	
Unsupervised	*dev*	*test*	*dev*	*test*	*dev*	*test*	*dev*	*test*	*dev*	*test*
InsNet [LMP22]	-	-	18.7	-	-	-	**100.0**	-	**100.0**	-
NeuroLogic [LWZ21]	-	41.9	-	24.7	-	14.4	-	96.7	-	-
A*esque [LWW22]	-	**44.3**	-	28.6	-	**15.6**	-	97.1	-	-
NADO [MLP22]	-	-	26.2	-	-	-	96.1	-	-	-
GeLaTo	**44.3**	43.8	**30.3**	**29.0**	**15.6**	15.5	**100.0**	**100.0**	**100.0**	**100.0**
Supervised	*dev*	*test*	*dev*	*test*	*dev*	*test*	*dev*	*test*	*dev*	*test*
NeuroLogic [LWZ21]	-	42.8	-	26.7	-	14.7	-	97.7	-	93.9[†]
A*esque [LWW22]	-	43.6	-	28.2	-	15.2	-	97.8	-	97.9[†]
NADO [MLP22]	44.4[†]	-	30.8	-	16.1[†]	-	97.1	-	88.8[†]	-
GeLaTo	**46.2**	**45.9**	**34.0**	**34.1**	**17.2**	**17.5**	**100.0**	**100.0**	**100.0**	**100.0**

Table 5.1: Performance comparison of different generation methods for *unsupervised* and *supervised* settings on the CommonGen dataset, measured by *generation quality* and *constraint satisfaction*.

GeLaTo assumes that distilled HMMs are good enough approximations for base models; our hypothesis is that the better the HMM approximates the base model, the better the generation quality. With GeLaTo, we generate from different HMM checkpoints from the distillation procedure, and report the average log-likelihoods and BLEU scores (without re-ranking the beams). As shown in Figure 5.2, as the training proceeds, both log-likelihood and BLEU score improves, exhibiting a clear positive correlation. This finding motivates the development of better tractable probabilistic models for language modeling.

Robustness of Hyperparameter w. As described in Section 5.2, for the supervised setting, the formulation of GeLaTo involves a hyperparameter $0 \leq w \leq 1$ that decides how much the TPM or the base model contributes to generation. For our experiments, w is set to 0.3 based on cross-validation results on the training set. Figure 5.3 shows the BLEU score (after re-ranking) on the validation set of CommonGen given different values of w.

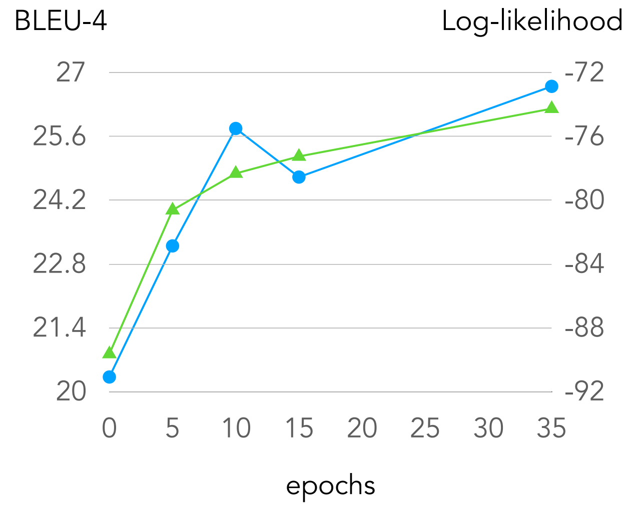


Figure 5.2: HMM log-likelihoods on data sampled from GPT2-large (triangles) and the corresponding BLEU scores (circles) w.r.t. # of training epochs. As the HMM model approximates GPT2-large better, the generation quality also improves.

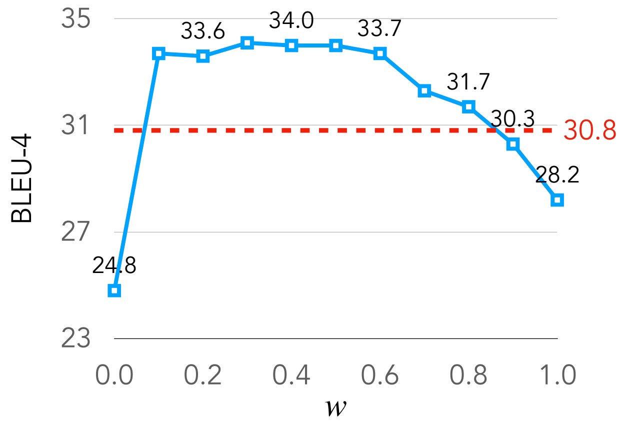


Figure 5.3: BLEU score on CommonGen (dev) for different values of w. GeLaTo achieves SoTA performance for $0.1 \leq w \leq 0.8$.

The performance of GeLaTo is very robust with respect to different choices of w, achieving SoTA BLEU scores for $0.1 \leq w \leq 0.8$.

CHAPTER 6

Conclusion

In recent years, deep generative models have achieved remarkable performance in generating texts and images. Yet, the black-box issue of neural networks makes it challenging to reliably control the behavior of deep generative models. One potential solution is to use tractable probabilistic models, in particular, probabilistic circuits (PCs), for generative modeling.

In this thesis, we study probabilistic circuits as a tractable counterpart of deep generative models based on neural networks. To overcome the bottleneck of scaling up PCs and adapting them to model real-world data from different domains, 1) we develop the Juice.jl library for efficient learning and probabilistic reasoning with PCs, and 2) we propose the "pruning and growing" algorithm for learning sparse structures of PCs. Then, we demonstrate the expressiveness and tractability of PCs on important real-world applications. Specifically, we show that (1) PCs encapsulate very expressive structures for modeling genetic sequences and (2) the tractability of PCs allows for efficient conditioning on various logical constraints, which can be used to reliably control the behavior of large language models.

REFERENCES

[AAC18] Sudarshan Adiga, Mohamed Adel Attia, Wei-Ting Chang, and Ravi Tandon. "On the tradeoff between mode collapse and sample quality in generative adversarial networks." In *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pp. 1184–1188. IEEE, 2018.

[AFJ16] Peter Anderson, Basura Fernando, Mark Johnson, and Stephen Gould. "Spice: Semantic propositional image caption evaluation." In *European Conference on Computer Vision (ECCV)*. Springer, 2016.

[BB11] Sharon R Browning and Brian L Browning. "Haplotype phasing: existing methods and new developments." *Nature Reviews Genetics*, **12**(10):703–714, 2011.

[BBG21] Franz Baumdicker, Gertjan Bisschop, Daniel Goldstein, Graham Gower, Aaron P Ragsdale, Georgia Tsambos, Sha Zhu, Bjarki Eldon, Castedo E Ellerman, Jared G Galloway, et al. "Efficient ancestry and mutation simulation with msprime 1.0." *bioRxiv*, 2021.

[BCB15] Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015.

[BCK21] CJ Battey, Gabrielle C Coffing, and Andrew D Kern. "Visualizing population structure with variational autoencoders." *G3*, **11**(1):1–11, 2021.

[BDC15] Jessa Bekker, Jesse Davis, Arthur Choi, Adnan Darwiche, and Guy Van den Broeck. "Tractable Learning for Complex Probability Queries." In *Advances in Neural Information Processing Systems 28 (NIPS)*, 2015.

[BMR20] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. "Language models are few-shot learners." *Advances in Neural Information Processing Systems 33 (NeurIPS)*, 2020.

[BPS12] Yael Baran, Bogdan Pasaniuc, Sriram Sankararaman, Dara G Torgerson, Christopher Gignoux, Celeste Eng, William Rodriguez-Cintron, Rocio Chapela, Jean G Ford, Pedro C Avila, et al. "Fast and accurate inference of local ancestry in Latino populations." *Bioinformatics*, **28**(10):1359–1367, 2012.

[CAC14] Vincenza Colonna, Qasim Ayub, Yuan Chen, Luca Pagani, Pierre Luisi, Marc Pybus, Erik Garrison, Yali Xue, Chris Tyler-Smith, 1000 Genomes Project Consortium, Goncalo R Abecasis, Adam Auton, Lisa D Brooks, Mark A DePristo, Richard M Durbin, Robert E Handsaker, Hyun Min Kang, Gabor T Marth, and

Gil A McVean. "Human genomic regions with exceptionally high levels of population differentiation identified from 911 whole-genome sequences." *Genome Biology*, **15**(6):R88, 2014.

[CAT17] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. "EMNIST: Extending MNIST to handwritten letters." In *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017.

[CBD15] Arthur Choi, Guy Van den Broeck, and Adnan Darwiche. "Probability Distributions over Structured Spaces." In *Proceedings of the AAAI Spring Symposium on KRR*, 2015.

[CD17] Arthur Choi and Adnan Darwiche. "On Relaxing Determinism in Arithmetic Circuits." In *Proceedings of the Thirty-Fourth International Conference on Machine Learning (ICML)*, 2017.

[CDB21] YooJung Choi, Meihua Dang, and Guy Van den Broeck. "Group Fairness by Probabilistic Modeling with Latent Fair Decisions." In *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, 2021.

[CFB20] YooJung Choi, Golnoosh Farnadi, Behrouz Babaki, and Guy Van den Broeck. "Learning Fair Naive Bayes Classifiers by Discovering and Eliminating Discrimination Patterns." In *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, 2020.

[CFZ16] Laura Clarke, Susan Fairley, Xiangqun Zheng-Bradley, Ian Streeter, Emily Perry, Ernesto Lowy, Anne-Marie Tassé, and Paul Flicek. "The international Genome sample resource (IGSR): A worldwide collection of genome variation incorporating the 1000 Genomes Project data." *Nucleic Acids Research*, **45**(D1):D854–D859, 09 2016.

[CL68] C. K. Chow and C. N. Liu. "Approximating discrete probability distributions with dependence trees." *IEEE Transactions on Information Theory*, 1968.

[CPC20] Alvaro Correia, Robert Peharz, and Cassio P de Campos. "Joints in random forests." In *Advances in Neural Information Processing Systems 33 (NeurIPS)*, 2020.

[CR20] Justin Chiu and Alexander M Rush. "Scaling Hidden Markov Language Models." In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

[CVB20] YooJung Choi, Antonio Vergari, and Guy Van den Broeck. "Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Models." *Technical report*, 2020.

[CZZ19] Woon Sang Cho, Pengchuan Zhang, Yizhe Zhang, Xiujun Li, Michel Galley, Chris Brockett, Mengdi Wang, and Jianfeng Gao. “Towards Coherent and Cohesive Long-form Text Generation.” In *Proceedings of the First Workshop on Narrative Understanding*, pp. 1–11, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[Dar02] Adnan Darwiche. “A Logical Approach to Factoring Belief Networks.” In *Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 2002.

[Dar03] Adnan Darwiche. “A Differential Approach to Inference in Bayesian Networks.” *Journal of the ACM*, 2003.

[DKB14] Laurent Dinh, David Krueger, and Yoshua Bengio. “Nice: Non-linear independent components estimation.” *arXiv preprint arXiv:1410.8516*, 2014.

[DKL21] Meihua Dang, Pasha Khosravi, Yitao Liang, Antonio Vergari, and Guy Van den Broeck. “Juice: A Julia Package for Logic and Probabilistic Circuits.” In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (Demo Track)*, 2021.

[DLB22] Meihua Dang, Anji Liu, and Guy Van den Broeck. “Sparse Probabilistic Circuits via Pruning and Growing.” In *Advances in Neural Information Processing Systems 35 (NeurIPS)*, 2022.

[DLW22] Meihua Dang, Anji Liu, Xinzhu Wei, Sriram Sankararaman, and Guy Van den Broeck. “Tractable and Expressive Generative Models of Genetic Variation Data.” In *Proceedings of the International Conference on Research in Computational Molecular Biology (RECOMB)*, 2022.

[DM02] Adnan Darwiche and Pierre Marquis. “A knowledge compilation map.” *Journal of Artificial Intelligence Research*, 2002.

[DMT22] Marc Dietrichstein, David Major, Martin Trapp, Maria Wimmer, Dimitrios Lenis, Philip Winter, Astrid Berg, Theresa Neubauer, and Katja Bühler. “Anomaly Detection Using Generative Models and Sum-Product Networks in Mammography Scans.” In *MICCAI Workshop on Deep Generative Models*, 2022.

[DMZ12] Olivier Delaneau, Jonathan Marchini, and Jean-François Zagury. “A linear complexity phasing method for thousands of genomes.” *Nature methods*, **9**(2):179–181, 2012.

[DVB20] Meihua Dang, Antonio Vergari, and Guy Van den Broeck. “Strudel: Learning Structured-Decomposable Probabilistic Circuits.” In *Proceedings of the 10th International Conference on Probabilistic Graphical Models (PGM)*, 2020.

[DVB22] Meihua Dang, Antonio Vergari, and Guy Van den Broeck. "Strudel: A Fast and Accurate Learner of Structured-Decomposable Probabilistic Circuits." *International Journal of Approximate Reasoning*, **140**:92–115, jan 2022.

[EF11] Laurent Excoffier and Matthieu Foll. "Fastsimcoal: a continuous-time coalescent simulator of genomic diversity under arbitrarily complex evolutionary scenarios." *Bioinformatics*, **27**(9):1332–1334, 2011.

[GE18] Aditya Grover and Stefano Ermon. "Boosted generative models." In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[GM97] Robert C. Griffiths and Paul Marjoram. "An Ancestral Recombination Graph." In P. Donnelly and S. Tavare, editors, *Progress in Population Genetics and Human Evolution, IMA Volumes in Mathematics and its Applications, vol. 87*, pp. 257–270. Springer, 1997.

[GPM14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial nets." In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[HFS12] Bryan Howie, Christian Fuchsberger, Matthew Stephens, Jonathan Marchini, and Gonçalo R Abecasis. "Fast and accurate genotype imputation in genome-wide association studies through pre-phasing." *Nature genetics*, **44**(8):955–959, 2012.

[Hin02] Geoffrey E Hinton. "Training products of experts by minimizing contrastive divergence." *Neural Computation*, 2002.

[HPV19] Emiel Hoogeboom, Jorn Peters, Rianne Van Den Berg, and Max Welling. "Integer discrete flows and lossless compression." In *Advances in Neural Information Processing Systems 32 (NeurIPS)*, 2019.

[HS07] Garrett Hellenthal and Matthew Stephens. "msHOT: modifying Hudson's ms simulator to incorporate crossover and gene conversion hotspots." *Bioinformatics*, **23**(4), 2007.

[Hud83] Richard R Hudson. "Properties of a neutral allele model with intragenic recombination." *Theoretical Population Biology*, **23**(2):183–201, 1983.

[Hud02] Richard R Hudson. "Generating samples under a Wright–Fisher neutral model of genetic variation." *Bioinformatics*, **18**(2):337–338, 2002.

[JNS06] Manfred Jaeger, Jens D. Nielsen, and Tomi Silander. "Learning probabilistic decision graphs." *International Journal of Approximate Reasoning*, **42**(1):84–100, 2006. PGM'04.

[KAH19] Friso Kingma, Pieter Abbeel, and Jonathan Ho. "Bit-swap: Recursive bits-back coding for lossless compression with hierarchical latent variables." In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 2019.

[KCL19] Pasha Khosravi, YooJung Choi, Yitao Liang, Antonio Vergari, and Guy Van den Broeck. "On Tractable Computation of Expected Predictions." In *Advances in Neural Information Processing Systems 32 (NeurIPS)*, 2019.

[KEM16] Jerome Kelleher, Alison M. Etheridge, and Gilean McVean. "Efficient Coalescent Simulation and Genealogical Analysis for Large Sample Sizes." *PLoS Computational Biology*, **12**(5):1–22, 2016.

[KF09] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning.* The MIT Press, 2009.

[KT12] Alex Kulesza and Ben Taskar. "Determinantal Point Processes for Machine Learning." *Foundations and Trends® in Machine Learning*, 2012.

[KVC14] Doga Kisa, Guy Van den Broeck, Arthur Choi, and Adnan Darwiche. "Probabilistic Sentential Decision Diagrams." In *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 2014.

[KW13] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes." *arXiv preprint arXiv:1312.6114*, 2013.

[LB21] Anji Liu and Guy Van den Broeck. "Tractable Regularization of Probabilistic Circuits." In *Advances in Neural Information Processing Systems 34 (NeurIPS)*, 2021.

[LBB17] 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.

[LCB10] Yann LeCun, Corinna Cortes, and CJ Burges. "MNIST handwritten digit database." *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, **2**, 2010.

[LFT15] Fei Liu, Jeffrey Flanigan, Sam Thomson, Norman Sadeh, and Noah A Smith. "Toward Abstractive Summarization Using Semantic Representations." In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, 2015.

[LH03] Chin-Yew Lin and Eduard Hovy. "Automatic evaluation of summaries using n-gram co-occurrence statistics." In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, 2003.

[LLG20] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension." In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.

[LMB22] Anji Liu, Stephan Mandt, and Guy Van den Broeck. "Lossless Compression with Probabilistic Circuits." In *Proceedings of the 10th International Conference on Learning Representations (ICLR)*, 2022.

[LMP22] Sidi Lu, Tao Meng, and Nanyun Peng. "InsNet: An Efficient, Flexible, and Performant Insertion-based Text Generation Model." In *Advances in Neural Information Processing Systems 35 (NeurIPS)*, 2022.

[LPM15] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. "Effective Approaches to Attention-based Neural Machine Translation." In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2015.

[LS03] Na Li and Matthew Stephens. "Modeling linkage disequilibrium and identifying recombination hotspots using single-nucleotide polymorphism data." *Genetics*, **165**(4):2213–2233, 2003.

[LWW22] Ximing Lu, Sean Welleck, Peter West, Liwei Jiang, Jungo Kasai, Daniel Khashabi, Ronan Le Bras, Lianhui Qin, Youngjae Yu, Rowan Zellers, Noah A. Smith, and Yejin Choi. "NeuroLogic A*esque Decoding: Constrained Text Generation with Lookahead Heuristics." In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, 2022.

[LWZ21] Ximing Lu, Peter West, Rowan Zellers, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. "NeuroLogic Decoding:(Un) supervised Neural Text Generation with Predicate Logic Constraints." In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, 2021.

[LZB23] Anji Liu, Honghua Zhang, and Guy Van den Broeck. "Scaling Up Probabilistic Circuits by Latent Variable Distillation." In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.

[LZS20] Bill Yuchen Lin, Wangchunshu Zhou, Ming Shen, Pei Zhou, Chandra Bhagavatula, Yejin Choi, and Xiang Ren. "CommonGen: A Constrained Text Generation Challenge for Generative Commonsense Reasoning." In *Findings of the Association for Computational Linguistics: EMNLP*, 2020.

[LZV21] Wenzhe Li, Zhe Zeng, Antonio Vergari, and Guy Van den Broeck. “Tractable Computation of Expected Kernels.” In *Proceedings of the 37th Conference on Uncertainty in Aritifical Intelligence (UAI)*, 2021.

[MBI19] Daniel Mas Montserrat, Carlos Bustamante, and Alexander Ioannidis. “Class-conditional vae-gan for local-ancestry simulation.” *arXiv preprint arXiv:1911.13220*, 2019.

[MC05] Gilean A T McVean and Niall J Cardin. “Approximating the coalescent with recombination.” *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, **360**(1459):1387–93, 2005.

[MD05] Radu Marinescu and Rina Dechter. “AND/OR branch-and-bound for graphical models.” In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, 2005.

[MH10] Jonathan Marchini and Bryan Howie. “Genotype imputation for genome-wide association studies.” *Nature Reviews Genetics*, **11**(7):499–511, 2010.

[MJ00] Marina Meila and Michael I Jordan. “Learning with mixtures of trees.” *Journal of Machine Learning Research*, **1**(Oct), 2000.

[MLP22] Tao Meng, Sidi Lu, Nanyun Peng, and Kai-Wei Chang. “Controllable Text Generation with Neurally-Decomposed Oracle.” In *Advances in Neural Information Processing Systems 35 (NeurIPS)*, 2022.

[MMS93] Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. “Building a Large Annotated Corpus of English: The Penn Treebank.” *Computational Linguistics*, 1993.

[MSD12] Tomáš Mikolov, Ilya Sutskever, Anoop Deoras, Hai-Son Le, and Stefan Kombrink. “Subword language modeling with neural networks.” *preprint (http://www.fit.vutbr.cz/imikolov/rnnlm/char.pdf)*, 2012.

[MVS19] Alejandro Molina, Antonio Vergari, Karl Stelzner, Robert Peharz, Pranav Subramani, Nicola Di Mauro, Pascal Poupart, and Kristian Kersting. “Spflow: An easy and extensible library for deep probabilistic learning using sum-product networks.” *arXiv preprint arXiv:1901.03704*, 2019.

[MW06] Paul Marjoram and Jeff D. Wall. “Fast ”coalescent” simulation.” *BMC Genetics*, **7**(1):16, 2006.

[Nor19] Magnus Nordborg. “Coalescent theory.” *Handbook of Statistical Genomics: Two Volume Set*, pp. 145–30, 2019.

[PD11] Hoifung Poon and Pedro Domingos. "Sum-product networks: A new deep architecture." In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, 2011.

[PLV20] 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 *Proceedings of the 37th International Conference on Machine Learning (ICML)*, 2020.

[PNR21] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. "Normalizing flows for probabilistic modeling and inference." *Journal of Machine Learning Research*, 2021.

[PRW02] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. "Bleu: a method for automatic evaluation of machine translation." In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics (ACL)*, 2002.

[PTP09] Alkes L Price, Arti Tandon, Nick Patterson, Kathleen C Barnes, Nicholas Rafaels, Ingo Ruczinski, Terri H Beaty, Rasika Mathias, David Reich, and Simon Myers. "Sensitive detection of chromosomal segments of distinct ancestry in admixed populations." *PLoS genetics*, **5**(6):e1000519, 2009.

[PVS20] 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 *Proceedings of the 36th Conference on Uncertainty in Aritifical Intelligence (UAI)*, 2020.

[RJ86] Lawrence Rabiner and Biinghwang Juang. "An introduction to hidden Markov models." *IEEE ASSP Magazine*, **3**(1), 1986.

[RKG14] Tahrima Rahman, Prasanna Kothalkar, and Vibhav Gogate. "Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of Chow-Liu trees." In *Joint European conference on machine learning and knowledge discovery in databases*, 2014.

[Rot96] Dan Roth. "On the hardness of approximate reasoning." *Artificial Intelligence*, 1996.

[RUS21] Yangjun Ruan, Karen Ullrich, Daniel S Severo, James Townsend, Ashish Khisti, Arnaud Doucet, Alireza Makhzani, and Chris Maddison. "Improving lossless compression rates via monte carlo bits-back coding." In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021.

[RWC19] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. "Language models are unsupervised multitask learners." *OpenAI blog*, 2019.

[SBF14] Ville A Satopää, Jonathan Baron, Dean P Foster, Barbara A Mellers, Philip E Tetlock, and Lyle H Ungar. "Combining multiple probability predictions using a simple logit model." *International Journal of Forecasting*, 2014.

[SCT20] Raymond Hendy Susanto, Shamil Chollampatt, and Liling Tan. "Lexically Constrained Neural Machine Translation with Levenshtein Transformer." In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.

[Smo86] Paul Smolensky. "Information processing in dynamical systems: Foundations of harmony theory." Technical report, Colorado Univ at Boulder Dept of Computer Science, 1986.

[SS06] Paul Scheet and Matthew Stephens. "A fast and flexible statistical model for large-scale population genotype data: applications to inferring missing genotypes and haplotypic phase." *The American Journal of Human Genetics*, **78**(4):629–644, 2006.

[SSE21] Andy Shih, Dorsa Sadigh, and Stefano Ermon. "HyperSPNs: Compact and Expressive Probabilistic Circuits." In *Advances in Neural Information Processing Systems 34 (NeurIPS)*, 2021.

[TBB18] James Townsend, Thomas Bird, and David Barber. "Practical lossless compression with latent variables using bits back coding." In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2018.

[TP22] Yufei Tian and Nanyun Peng. "Zero-shot Sonnet Generation with Discourse-level Planning and Aesthetics Features." In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, 2022.

[TVA19] Dustin Tran, Keyon Vafa, Kumar Agrawal, Laurent Dinh, and Ben Poole. "Discrete flows: Invertible generative models of discrete data." In *Advances in Neural Information Processing Systems 32 (NeurIPS)*, 2019.

[VCL21] Antonio Vergari, YooJung Choi, Anji Liu, Stefano Teso, and Guy Van den Broeck. "A Compositional Atlas of Tractable Circuit Operations for Probabilistic Inference." In *Advances in Neural Information Processing Systems 34 (NeurIPS)*, 2021.

[VCP20] Antonio Vergari, YooJung Choi, Robert Peharz, and Guy Van den Broeck. "Probabilistic circuits: Representations, inference, learning and applications." *AAAI Tutorial*, 2020.

[VLP15] Ramakrishna Vedantam, C Lawrence Zitnick, and Devi Parikh. "Cider: Consensus-based image description evaluation." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[Wak20] John Wakeley. "Developments in coalescent theory from single loci to chromosomes." *Theoretical population biology*, **133**:56–64, 2020.

[WH99] Carsten Wiuf and Jotun Hein. "Recombination as a point process along sequences." *Theoretical Population Biology*, **55**(3):248–259, 1999.

[XD19] Jiacheng Xu and Greg Durrett. "Neural Extractive Text Summarization with Syntactic Compression." In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019.

[XRV17] Han Xiao, Kashif Rasul, and Roland Vollgraf. "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms." *arXiv preprint arXiv:1708.07747*, 2017.

[YDO21] Burak Yelmen, Aurélien Decelle, Linda Ongaro, Davide Marnetto, Corentin Tallec, Francesco Montinaro, Cyril Furtlehner, Luca Pagani, and Flora Jay. "Creating artificial human genomes using generative neural networks." *PLOS Genetics*, **17**(2):1–22, 02 2021.

[YK21] Kevin Yang and Dan Klein. "FUDGE: Controlled Text Generation With Future Discriminators." In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, 2021.

[YPW19] Lili Yao, Nanyun Peng, Ralph Weischedel, Kevin Knight, Dongyan Zhao, and Rui Yan. "Plan-and-write: Towards better automatic storytelling." In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

[ZDP23] Honghua Zhang, Meihua Dang, Nanyun Peng, and Guy Van den Broeck. "Tractable Control for Autoregressive Language Generation." In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, jul 2023.

[ZJB21] Honghua Zhang, Brendan Juba, and Guy Van den Broeck. "Probabilistic Generating Circuits." In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021.

[ZLM22] Honghua Zhang, Liunian Harold Li, Tao Meng, Kai-Wei Chang, and Guy Van den Broeck. "On the paradox of learning to reason from data." *arXiv preprint arXiv:2205.11502*, 2022.

[ZR19] Zachary Ziegler and Alexander Rush. "Latent normalizing flows for discrete sequences." In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 2019.

[ZWL20] Yizhe Zhang, Guoyin Wang, Chunyuan Li, Zhe Gan, Chris Brockett, and Bill Dolan. "POINTER: Constrained Progressive Text Generation via Insertion-based Generative Pre-training." In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 8649–8670, Online, November 2020. Association for Computational Linguistics.