Sparse Probabilistic Circuits via Pruning and Growing

Abstract

Probabilistic circuits (PCs) are a tractable representation of probability distributions allowing for exact and efficient computation of likelihoods and marginals. There has been significant recent progress on improving the scale and expressiveness of PCs. However, PC training performance plateaus as model size increases. We discover that most capacity in existing large PC structures is wasted: fully-connected parameter layers are only sparsely used. We propose two operations: *pruning* and *growing*, that exploit the sparsity of PC structures. Specifically, the pruning operation removes unimportant sub-networks of the PC for model compression and comes with theoretical guarantees. The growing operation increases model capacity by increasing the dimensions of latent states. By alternatingly applying pruning and growing, we increase the capacity that is meaningfully used, allowing us to significantly scale up PC learning. Empirically, our learner achieves state-of-the-art likelihoods on MNIST-family image datasets and an Penn Tree Bank language data compared to other PC learners and less tractable deep generative models such as flow-based models and variational autoencoders (VAEs).

1 INTRODUCTION

Probabilistic circuits (PCs) [Vergari et al., 2020, Choi et al., 2020b] are a unifying framework to abstract from a multitude of tractable probabilistic models. The key property that separates PCs from other deep generative models such as flow-based models [Papamakarios et al., 2021] and VAEs [Kingma and Welling, 2013] is their *tractability*. It enables them to compute various queries, including marginal probabilities, exactly and efficiently [Vergari et al., 2021]. Therefore, PCs are increasingly used in inference-demanding applications such as enforcing algorithmic fairness [Choi et al., 2020a, 2021], making predictions under missing data [Khosravi et al., 2019], and data compression [Liu et al., 2022a].

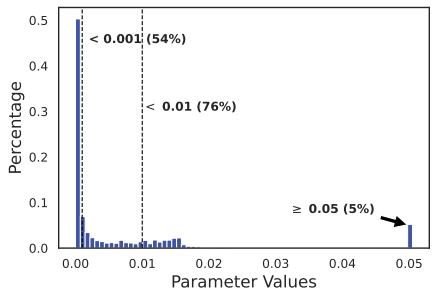


Figure 1: Histogram of parameter values for a state-of-the-art PC with 2.18M parameters on MNIST.

Recent advanced of PC learning [Rahman et al., 2014], regularization [Shih et al., 2021, Liu and Van den Broeck, 2021] and efficient parallelism implementation [Peharz et al., 2020a] have been pushing the limits of PC's expressivity and scalability such that PCs can even match the performance of less tractable deep generative models such as Flows and VAEs. This lead to a trend of building larger PCs. However, PC training performance plateaus as model size increases. This indicates that to go even further, simply scaling up might not suffice, and we need to be better at using the capacity available. We discover that this might be caused by the fact that the capacity of large PCs are wasted. As shown in Figure 1, most parameters in a PC with 2.18M parameters have close-to-zero values, which have little effect to the PC distribution. Since existing PC structures usually have fully-connected parameter layers [Liu and Van den Broeck, 2021, Rahman et al., 2014], this indicates that the parameter values only are sparsely used.

In this work, 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-

Submitted to the 5th Workshop on Tractable Probabilistic Modeling at UAI (TPM 2022). **To be used for reviewing only**.

networks of a PC. This is done by quantifying the importance of PC edges 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. Applying pruning and growing iteratively greatly refine the structure and parameters of a PC. Empirically, the log-likelihoods metric can improve from 2% to 10% after a few iterations. Compared to existing PC learners and 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 Penn Tree Bank language modeling task.

2 PROBABILISTIC CIRCUITS

Probabilistic circuits (PCs) Vergari et al. [2020], Choi et al. [2020b] model probability distributions with a structured computation graph. They are an umbrella term for a large family of tractable probabilistic models including arithmetic circuits Darwiche [2002, 2003], sum-product networks (SPNs) Poon and Domingos [2011], cutset networks Rahman et al. [2014], and-or search spaces Marinescu and Dechter [2005], and probabilistic sentential decision diagrams Kisa et al. [2014]. The syntax and semantics of PCs are defined as follows.

Definition 1 (Probabilistic Circuit)**.** A PC $\mathcal{C} := (\mathcal{G}, \boldsymbol{\theta})$ represents a joint probability distribution $p(\mathbf{X})$ over random variables $\mathbf{X}$ through a directed acyclic (computation) graph (DAG) $\mathcal{G}$ parameterized by $\boldsymbol{\theta}$. 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 p_n defined recursively as follows:

$$p_n(\boldsymbol{x}) := \begin{cases} f_n(\boldsymbol{x}) & \text{if } n \text{ is an input unit} \\ \prod_{c \in \mathsf{in}(n)} p_c(\boldsymbol{x}) & \text{if } n \text{ is a product unit} \\ \sum_{c \in \mathsf{in}(n)} \theta_{c|n} \cdot p_c(\boldsymbol{x}) & \text{if } n \text{ is a sum unit} \end{cases} \tag{1}$$

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., $\sum_{c \in \mathsf{in}(n)} \theta_{c|n} = 1$.

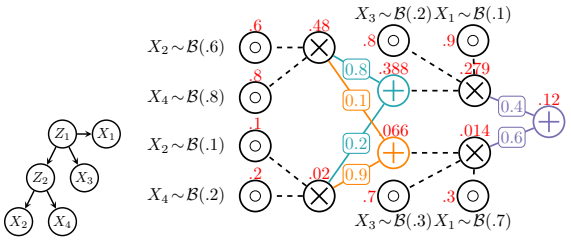
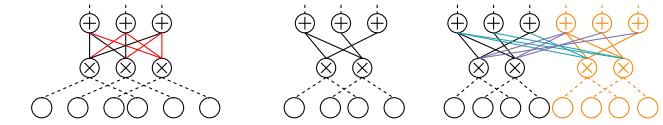


Figure 2: A smooth and decomposable PC (right) and equivalent Bayesian network (left). The probability of each unit given input $\bar{x_1} x_2 \bar{x_3} x_4$ is in red.



(a) fully-connected layers (b) after pruning (c) after growing

Figure 3: Pruning (3a to 3b) and growing (3b to 3c).

Intuitively, a product unit defines a factorized distribution over its inputs, and a sum unit represents a mixture over its input with weights $\{\theta_{c|n} : c \in \mathsf{in}(n)\}$. Finally, the probability distribution of a PC (i.e., $p_{\mathcal{C}}$) is defined as the distribution represented by its root unit r (i.e., $p_r(\boldsymbol{x})$), that is, its output neuron. The size of a PC, denoted $|\mathcal{C}| = |\boldsymbol{\theta}|$, is the number of parameters in $\mathcal{C}$. Figure 2 shows an example of PC.

Computing the (log)likelihood of PC $\mathcal{C}$ given a sample $\boldsymbol{x}$ is evaluating its computation units in $\mathcal{G}$ in a feedforward manner following Equation 1.

The key property that separates PCs from other deep probabilistic models such as flows [Dinh et al., 2014] and VAEs [Kingma and Welling, 2013] is their *tractability*, which is the ability to exactly and efficiently answer various probabilistic queries. This paper focuses on PCs that support linear time (w.r.t. model size) marginal probability computation, as they are increasingly used in downstream applications such as data compression [Liu et al., 2022b] and making predictions under missing data [Khosravi et al., 2019], and also achieve on-par expressiveness [Liu et al., 2022b, Liu and Van den Broeck, 2021, Liang et al., 2017]. To support efficient marginal inference, PCs need to be *smooth* and *decomposable*.

Definition 2 (Smoothness and Decomposability [Darwiche and Marquis, 2002])**.** For a PC, the *scope* $\phi(n)$ of a PC unit n is the input variables that it depends on; then, (1) a product unit is *decomposable* if its children have disjoint scope; (2) a sum unit is *smooth* if its children have identical scope. A PC is decomposable (or smooth) if all of its produce units are decomposable (or smooth).

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 units are well-defined over the same set of variables. Both structural properties will be the key to guaranteeing the effectiveness of the structure learning algorithms proposed in the following sections.

Algorithm 1: PC Sampling

Input : a PC $\mathcal{C}$ over variables $\mathbf{X}$
Output : an instance $\boldsymbol{x} \sim p_{\mathcal{C}}$

1 $Q \leftarrow$ a queue initialized with the root node r of $\mathcal{C}$
2 **while** Q *is not empty* **do**
3 $\quad n \leftarrow Q.\text{pop}()$
4 $\quad$ **if** n is an input unit **then** sample the value of $\text{var}(n)$ following the distribution defined by n
5 $\quad$ **else if** n is a product unit **then** $Q.\text{push}(c)$ for each $c \in \text{in}(n)$
6 $\quad$ **else if** n *is a sum unit* **then**
7 $\quad\quad$ sample $i \sim \text{Categorical}(\{\theta_{c|n} : c \in \text{in}(n)\})$; $Q.\text{push}(c_i)$, where c_i is the ith input of n

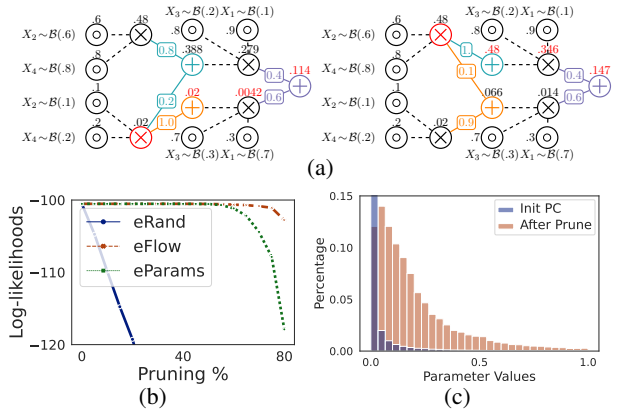


Figure 4: (4a) A case study of comparing pruning heuristics EPARAM (left) and EFLOW (right) pruned from PC in Figure 2 given sample $\boldsymbol{x} = \bar{x}_1 x_2 \bar{x}_3 x_4$. (4b): Comparison of pruning heuristics (1) ERAND (2) EPARAM, (3) EFLOW for different percentage (4c): Histogram of parameters before/after the pruning operation.

3 PROBABILISTIC CIRCUIT MODEL COMPRESSION VIA PRUNING

Given a PC $\mathcal{C}$ and a dataset $\mathcal{D}$, our goal is to efficiently identify a set of k edges $\mathcal{E}$ such that the performance gap between the pruned PC $\mathcal{C}_{\setminus\mathcal{E}}$ and the original PC $\mathcal{C}$ is minimized:

$$\begin{aligned} &\operatorname*{argmin}_{\mathcal{E}} \mathcal{LL}(\mathcal{D}, \mathcal{C}) - \mathcal{LL}(\mathcal{D}, \mathcal{C}_{\setminus\mathcal{E}}) \\ &\text{such that } \mathcal{E} \subseteq \{(n, c) : \theta_{c|n} \in \boldsymbol{\theta}\} \text{ and } |\mathcal{E}| = k, \end{aligned} \tag{2}$$

where $\mathcal{LL}(\mathcal{D}, \mathcal{C}) = \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x} \in \mathcal{D}} \log p_{\mathcal{C}}(\boldsymbol{x})$ is the averaged log-likelihood of PC $\mathcal{C}$ given dataset $\mathcal{D}$. The edges $\mathcal{E}$ are chosen among all parameterized edges (i.e., all input edges of sum units). Figure 3b illustrates the result of pruning five (red) edges from the PC in Figure 3a.

Pruning by parameters. Figure 1 shows that most parameters in a large PC are very small. Hence, by pruning away these unimportant components, it is possible to significantly reduce model size while maximally retaining model expressiveness. This leads to thee EPARAM heuristic, which selects k edges with the smallest parameters. The parameters of a sum unit are normalized to be 1 so they only contain local information and are insufficient to quantify the importance of inputs to a sum unit in the entire PC's distribution. In Figure 4a (left), we prune the edge with the smallest parameter. However, as shown in Figure 4a (right), 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 PC semantics and we will introduce it next with its corresponding heuristics EFLOW.

Pruning by generative significance. A more informed pruning strategy needs to consider the global impact of edges on the PC distribution. To achieve this, we quantify the significance of a unit or edge by the probability that it will be "activated" when drawing samples from the PC.

Algorithm 1 shows that the PC sampling process proceeds in a top-down manner: a queue Q is initialized with the root unit (line 1). The algorithm then processes every unit in the queue until it is empty. For a sum unit n (lines 6-7), the sampler randomly adds one of its input units to the queue according to the categorical distribution defined by sum parameters $\{\theta_{c|n} : c \in \text{in}(n)\}$; for a product unit (line 5), all its inputs are added to the queue; for an input unit n defined on variable X (line 4), the algorithm randomly samples value x according to its input distribution. Algorithm 1 is designed to sample instances following the PC distribution, therefore the probability of adding a unit n to the queue Q is the probability that n will be sampled, which we call as the *top-down probability* of n.

Definition 3 (Top-down probability)**.** The top-down probability of the output unit r is 1: $p^{\text{TD}}_{r,\boldsymbol{\theta}}(r) = 1$. The top-down probability of sum/product units n is defined recursively:

$$p^{\text{TD}}_{r,\boldsymbol{\theta}}(n) = \sum_{m \in \text{pa}_{\text{sum}}(n)} \theta_{n|m} \cdot p^{\text{TD}}_{r,\boldsymbol{\theta}}(m) + \sum_{m' \in \text{pa}_{\text{prod}}(n)} p^{\text{TD}}_{r,\boldsymbol{\theta}}(m'),$$

where $\text{pa}_{\text{sum}}(n)$ and $\text{pa}_{\text{prod}}(n)$ are the sum and product units that take n as input. The top-down probability of a sum edge (n, c) is defined as $p^{\text{TD}}_{r,\boldsymbol{\theta}}(n, c) = \theta_{c|n} \cdot p^{\text{TD}}_{r,\boldsymbol{\theta}}(n)$.

Pruning by circuit flows. The top-down probability $p^{\text{TD}}_{\mathcal{C},\boldsymbol{\theta}}(n)$ represents the probability of reaching n in an unconditional sampling process. However, the pruning objective of Equation 2 requires that the sampled instance is some $\boldsymbol{x} \in \mathcal{D}$. Therefore, we define circuit flow as a sample-dependent version of the top-down probability.

Definition 4 (Circuit Flow)**.** For a given $\mathcal{C} = (\mathcal{G}, \boldsymbol{\theta})$ and input $\boldsymbol{x}$, let $\boldsymbol{\theta}^{\boldsymbol{x}}$ denote a new set of parameters such that $\theta^{\boldsymbol{x}}_{c|n}$ is the probability of component c in the mixture represented by sum unit n *after observing sample* $\boldsymbol{x}$. The node flow $\text{F}_n(\boldsymbol{x})$ of a unit n is then defined as the top-down probability under this reparameterization of the circuit:

$$\text{F}_n(\boldsymbol{x}) = p^{\text{TD}}_{r,\boldsymbol{\theta}^{\boldsymbol{x}}}(n), \text{where} \theta^{\boldsymbol{x}}_{c|n} = \frac{\theta_{c|n} \cdot p_c(\boldsymbol{x})}{\sum_{c' \in \text{in}(n)} \theta_{c'|n} \cdot p_{c'}(\boldsymbol{x})}.$$

Similarly, the edge flow $\mathrm{F}_{n,c}(\boldsymbol{x})$ w.r.t. PC $\mathcal{C}$ and sample $\boldsymbol{x}$ is defined by $p^{\mathtt{TD}}_{r,\boldsymbol{\theta}^{\boldsymbol{x}}}(n,c)$. We further define $\mathrm{F}_{n,c}(\mathcal{D}) = \sum_{\boldsymbol{x}\in\mathcal{D}} \mathrm{F}_{n,c}(\boldsymbol{x})$ as the *aggregate flow* over $\mathcal{D}$.

$\mathrm{F}_n(\boldsymbol{x})$ defines the probability of reaching unit n in Algorithm 1, given sampled $\boldsymbol{x}$. Therefore, edge flow $\mathrm{F}_{n,c}(\boldsymbol{x})$ is a natural metric of the importance of edge (n,c) given $\boldsymbol{x}$, measuring how many expected samples "flow" through certain edges, which we call EFLOW heuristic.

Empirical Analysis. Figure 4b compares the effect of pruning heuristics EPARAM, EFLOW, and ERAND(an uninformed strategy, prune randomly). 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 4c, the parameter distribution is more balanced after pruning compared to Figure 1, indicating a higher significance of each edge. We also theoretically verify the effectiveness of the EFLOW heuristic in Appendix A.

4 SCALABLE STRUCTURE LEARNING

The pruning operator benefits PCs in two aspects: (1) model parameters are more balanced after pruning (Figure 4c); (2) pruning removes sub-circuits with negligible contribution to the model distribution. In another direction, we propose a *growing* operation to increase the PC capacity by introducing more sub-circuits. Pruning and growing together defines an scalable structure learning algorithm for PCs.

Growing Operation. *Growing* operation increases model size by coping its existing components and injecting noise. Specifically, the growing operation is applied to units, edges, and parameters respectively: (1) for units, growing creates new copy $n^{\mathtt{new}}$ for every unit n; (2) for edges, the sum edge (n,c) PC (Figure 3b) are copied three times to the grown PC (Figure 3c): from new parent to new child $(n^{\mathtt{new}}, c^{\mathtt{new}})$, from old parent to new child $(n, c^{\mathtt{new}})$, and from new parent to old child $(n^{\mathtt{new}}, c)$; the product edge is simply connecting new copied sum units; (3) for parameters, new parameter $\theta^{\mathtt{new}}_{c|n}$ are a noisy copy from old parameter $\theta_{c|n}$, that is $\theta^{\mathtt{new}}_{c|n} \leftarrow \epsilon \cdot \theta_{c|n}$ where $\epsilon \sim \mathcal{N}(1,\sigma^2)$ and σ^2 controls the Gaussian noise variance. Gaussian noise is injected to encourage parameter learning algorithms to find diverse parameters for different copies. After a growing operation, the PC size is 4 times the original PC size. Algorithm 4 in Appendix B shows a feed-forward implementation of the growing operation.

Structure Learning through Pruning and Growing. We utilize pruning and growing operations and propose a joint structure and parameter learning algorithm for PCs. Specifically, starting from an initial PC, we apply 75% pruning, growing, and parameter learning iteratively until convergence. We utilize HCLT [Liu and Van den Broeck, 2021] as initial PC structure as it has the state-of-the-art likelihood performance, however this structure learning pipeline can be applied to any PC structure.

5 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 [LeCun et al., 2010], EMNIST [Cohen et al., 2017], and FashionMNIST [Xiao et al., 2017]; (2) character level Penn Tree Bank language modeling task [Marcus et al., 1993]. We report the test set bits-per-dimension (bpd) we get via structure learning proposed in Section 4. Details of experiments are in Appendix C.

Image Datasets. We compare with 2 competitive PC learning algorithms: HCLT [Liu and Van den Broeck, 2021] and RatSPN [Peharz et al., 2020b], one flow-based model: IDF [Hoogeboom et al., 2019], and 3 VAE based methods: BitSwap [Kingma et al., 2019], BB-ANS [Townsend et al., 2018], and McBits [Ruan et al., 2021]. As shown in Table 1, 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.

Table 1: Test set bpd on MNIST-family datasets.

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

Language Modeling Task. We use the Penn Tree Bank dataset with standard processing from Mikolov et al. [2012]. We compare with 3 competitive normalizing flow based models: Bipartite flow [Tran et al., 2019] and latent flows [Ziegler and Rush, 2019] including AF/SCF and IAF/SCF, since they are the only comparable work with non-autoregressive language modeling. As shown in Tab. 2, the proposed method outperforms all 3 baselines.

Table 2: Character-level language modeling results.

Dataset	SparsePC	Bipartite flow	AF/SCF	IAF/SCF
Penn Tree Bank	**1.35**	1.38	1.46	1.63

Conclusions. We propose structure learning of PCs by combining pruning and growing operations to exploit the sparsity of PC structures and show significant empirical improvements in the density estimation tasks.

References

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*, 2020a.

YooJung Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic models. *Technical report*, 2020b.

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.

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)*, pages 2921–2926. IEEE, 2017.

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.

Adnan Darwiche. A logical approach to factoring belief networks. In *Proceedings of KR*, pages 409–420, 2002.

Adnan Darwiche. A differential approach to inference in bayesian networks. *Journal of the ACM*, 50(3):280–305, 2003.

Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

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

Emiel Hoogeboom, Jorn Peters, Rianne Van Den Berg, and Max Welling. Integer discrete flows and lossless compression. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

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.

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

Friso Kingma, Pieter Abbeel, and Jonathan Ho. Bit-swap: Recursive bits-back coding for lossless compression with hierarchical latent variables. In *International Conference on Machine Learning*, pages 3408–3417. PMLR, 2019.

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.

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

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.

Anji Liu and Guy Van den Broeck. Tractable regularization of probabilistic circuits. In *Advances in Neural Information Processing Systems 35 (NeurIPS)*, 2021.

Anji Liu, Stephan Mandt, and Guy Van den Broeck. Lossless compression with probabilistic circuits. In *International Conference on Learning Representations (ICLR)*, 2022a.

Anji Liu, Stephan Mandt, and Guy Van den Broeck. Lossless compression with probabilistic circuits. In *International Conference on Learning Representations (ICLR)*, 2022b.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2): 313–330, 1993. ISSN 0891-2017.

Radu Marinescu and Rina Dechter. And/or branch-and-bound for graphical models. In *IJCAI*, pages 224–229, 2005.

Tomáš Mikolov, Ilya Sutskever, Anoop Deoras, Hai-Son Le, and Stefan Kombrink. Subword language modeling with neural networks. 2012.

George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 22(57):1–64, 2021.

Robert Peharz, Steven Lang, Antonio Vergari, Karl Stelzner, Alejandro Molina, Martin Trapp, Guy Van den Broeck, Kristian Kersting, and Zoubin Ghahramani. Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In *International Conference on Machine Learning*, pages 7563–7574. PMLR, 2020a.

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

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

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*, pages 630–645. Springer, 2014.

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 *International Conference on Machine Learning*, pages 9136–9147. PMLR, 2021.

Andy Shih, Dorsa Sadigh, and Stefano Ermon. Hyperspns: Compact and expressive probabilistic circuits. *Advances in Neural Information Processing Systems*, 34, 2021.

James Townsend, Thomas Bird, and David Barber. Practical lossless compression with latent variables using bits back coding. In *International Conference on Learning Representations*, 2018.

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*, volume 32, 2019.

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

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*, volume 34, 2021.

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

Zachary Ziegler and Alexander Rush. Latent normalizing flows for discrete sequences. In *International Conference on Machine Learning*, pages 7673–7682. PMLR, 2019.

A BOUNDING AND APPROXIMATING THE LOSS OF LIKELIHOOD

This section establish a theoretical upper bound on the log-likelihood drop $\Delta\mathcal{LL}(\mathcal{D},\mathcal{C},\mathcal{E}) = \mathcal{LL}(\mathcal{D},\mathcal{C}) - \mathcal{LL}(\mathcal{D},\mathcal{C}_{\setminus\mathcal{E}})$ (cf. Equation 2) caused by pruning away edges $\mathcal{E}$. Interestingly, the EFLOW heuristic proposed in the previous section is a good approximation of the derived upper bound.

We start from the case of pruning one edge (i.e., $k = 1$ in Equation 2). In this case, the loss of likelihood can be quantified exactly using flows and edge parameters:

Theorem 1 (Log-likelihood drop of pruning one edge)**.** *For a PC $\mathcal{C}$ and a dataset $\mathcal{D}$, the loss of log-likelihood by pruning away edge (n, c) is*

$$\begin{aligned}
&\Delta\mathcal{LL}(\mathcal{D},\mathcal{C},\{(n,c)\}) \\
&\quad= \frac{1}{|\mathcal{D}|}\sum_{\boldsymbol{x}\in\mathcal{D}} \log\left(\frac{1-\theta_{c|n}}{1-\theta_{c|n}+\theta_{c|n}\,\mathrm{F}_n(\boldsymbol{x})-\mathrm{F}_{n,c}(\boldsymbol{x})}\right) \\
&\quad\leq -\frac{1}{|\mathcal{D}|}\sum_{\boldsymbol{x}\in\mathcal{D}} \log(1-\mathrm{F}_{n,c}(\boldsymbol{x})).
\end{aligned}$$

See proof in Appendix A.1.1. By computing the second term from Theorem 1, we can pick the edge with the smallest log-likelihood drop. Additionally, the third term characterizes the log-likelihood drop without re-normalizing parameters of $\theta_{\cdot|n}$. It suggests pruning the edge with smallest edge flow. A key insight from Theorem 1 is that the log-likelihood drop depends explicitly on the edge flow $\mathrm{F}_{n,c}(\boldsymbol{x})$ and unit flow $\mathrm{F}_n(\boldsymbol{x})$. This matches the intuition from Section 3 that the circuit flow of an edge is sufficient to quantify its importance in the PC.

Next, we bound the drop of pruning multiple edges.

Theorem 2 (Log-likelihood drop of pruning multiple edges)**.** *Let $\mathcal{C}$ be a PC and $\mathcal{D}$ be a dataset. For any set of edges $\mathcal{E}$ in $\mathcal{C}$, if $\forall\boldsymbol{x}\in\mathcal{D}, \sum_{(n,c)\in\mathcal{E}} \mathrm{F}_{n,c}(\boldsymbol{x}) < 1$, the log-likelihood drop by pruning away $\mathcal{E}$ is bounded and approximated by*

$$\begin{aligned}
\Delta\mathcal{LL}(\mathcal{D},\mathcal{C},\mathcal{E}) &\leq -\frac{1}{|\mathcal{D}|}\sum_{\boldsymbol{x}} \log(1-\sum_{(n,c)\in\mathcal{E}} \mathrm{F}_{n,c}(\boldsymbol{x})) \\
&\approx \frac{1}{|\mathcal{D}|}\sum_{(n,c)\in\mathcal{E}} \mathrm{F}_{n,c}(\mathcal{D}).
\end{aligned} \tag{3}$$

Proof of this theorem is provided in Appendix A.1.2. We first look at the second term of Equation 3. Although it provides an upper bound to the performance drop, it cannot be used as a pruning heuristic since the bound does not decompose over edges. And hence finding the set of edges with the lowest score requires evaluating the bound exponentially (w.r.t. k) many times. Therefore, we do an additional approximation step of the bound via Taylor expansion, which leads to the third term of Equation 3. This approximation

matches the EFLOW heuristic by a constant factor $1/|\mathcal{D}|$, which theoretically justifies the effectiveness of the heuristic. As shown in Figure 5, the approximate bound (EFLOW heuristic) matches closely to the actual log-likelihood drop.

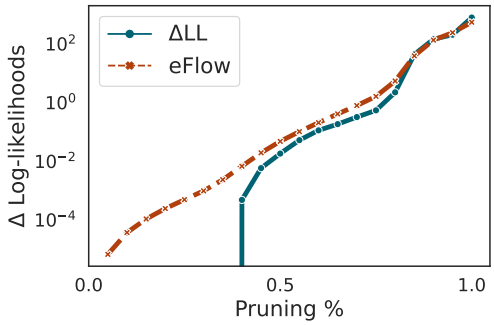


Figure 5: Comparing the actual log-likelihood drop (ΔLL) and quantity computed from EFLOW heuristics (which is also the approximated upper bound in Equation 3) for different percentage of pruned parameters (x-axis).

A.1 PROOFS

In this section, we provide detailed proofs of Theorem 1 (Section A.1.1) and Theorem 2 (Section A.1.2).

A.1.1 Pruning One Edge over One Example

Lemma 1 (Pruning One Edge Log-Likelihood Lower Bound)**.** *For a PC $\mathcal{C}$ and a sample $\boldsymbol{x}$, the loss of log-likelihood by pruning away edge (n, c) is*

$$\begin{aligned}&\Delta\mathcal{LL}(\{\boldsymbol{x}\}, \mathcal{C}, \{(n,c)\})\\ &\quad= \log\left(\frac{1-\theta_{c|n}}{1-\theta_{c|n}+\theta_{c|n}\,\mathrm{F}_n(\boldsymbol{x})-\mathrm{F}_{n,c}(\boldsymbol{x})}\right)\\ &\quad\leq -\log(1-\mathrm{F}_{n,c}(\boldsymbol{x})).\end{aligned}$$

Proof. For notation simplicit, denote the probability of units m (resp. n) in the original (resp. pruned) PC given sample $\boldsymbol{x}$ as $p_m(\boldsymbol{x})$ (resp. $p'_n(\boldsymbol{x})$). As a slight extension of Definition 4, we define $F_n(\boldsymbol{x}; m)$ as the flow of unit n w.r.t. the PC rooted at m.

The proof proceeds by induction over the PC's root unit. That is, we first consider pruning (n, c) w.r.t. the PC rooted at n. Then, in the induction step, we prove that if the lemma holds for PC rooted at m, then it also holds for PC rooted at any parent unit of m. Instead of directly proving the statement in Lemma 1, we first prove that for any root node m, the following holds:

$$\begin{aligned}&p_m(\boldsymbol{x}) - p'_m(\boldsymbol{x})\\ &= F_n(\boldsymbol{x}; m)\cdot p_m(\boldsymbol{x})\cdot\left(\frac{1}{1-\theta}\frac{\mathrm{F}_{n,c}(\boldsymbol{x}; m)}{\mathrm{F}_n(\boldsymbol{x}; m)} - \frac{\theta}{1-\theta}\right).\end{aligned} \tag{4}$$

Base case: pruning an edge of the root unit. That is, the root unit of the PC is n. In this case, we have

$$\begin{aligned}&p_n(\boldsymbol{x}) - p'_n(\boldsymbol{x})\\ &\quad= \sum_{c'\in\mathsf{in}(n)}\theta_{c'|n}\cdot p_c(\boldsymbol{x}) - \sum_{c'\in\mathsf{in}(n)\setminus c}\theta'_{c'|n}\cdot p'_c(\boldsymbol{x})\\ &\quad= \theta_{c|n}\cdot p_c(\boldsymbol{x})\\ &\qquad+\sum_{c'\in\mathsf{in}(n)\setminus c}\theta_{c'|n}\cdot p_c(\boldsymbol{x}) - \sum_{c'\in\mathsf{in}(n)\setminus c}\theta'_{c'|n}\cdot p_c(\boldsymbol{x}),\end{aligned} \tag{5}$$

where $\theta'_{c|n}$ denotes the normalized parameter corresponding to edge (n, c) in the pruned PC. Specifically, we have

$$\forall m\in\mathsf{in}(n)\setminus c,\quad \theta'_{m|n} = \frac{\theta_{m|n}}{\sum_{c'\in\mathsf{in}(n)\setminus c}\theta_{c'|n}} = \frac{\theta_{m|n}}{1-\theta_{c|n}}.$$

For notation simplicity, denote $\theta := \theta_{c|n}$. Plug in the above definition into Equation 5, we have

$$\begin{aligned}&p_n(\boldsymbol{x}) - p'_n(\boldsymbol{x})\\ &= \theta_{c|n}\cdot p_c(\boldsymbol{x})\\ &\quad+\sum_{c'\in\mathsf{in}(n)\setminus c}\theta_{c'|n}\cdot p_c(\boldsymbol{x}) - \frac{1}{1-\theta}\sum_{c'\in\mathsf{in}(n)\setminus c}\theta_{c'|n}\cdot p_c(\boldsymbol{x})\\ &= \theta_{c|n}\cdot p_c(\boldsymbol{x}) - \frac{\theta}{1-\theta}\sum_{c'\in\mathsf{in}(n)\setminus c}\theta_{c'|n}\cdot p_c(\boldsymbol{x})\\ &= \theta_{c|n}\cdot p_c(\boldsymbol{x}) - \frac{\theta}{1-\theta}(p_n(\boldsymbol{x}) - \theta_{c|n}p_c(\boldsymbol{x}))\\ &= \frac{1}{1-\theta}\cdot\theta_{c|n}\cdot p_c(\boldsymbol{x}) - \frac{\theta}{1-\theta}\cdot p_n(\boldsymbol{x})\\ &\overset{(a)}{=} \frac{1}{1-\theta}\cdot p_n(\boldsymbol{x})\cdot\frac{\mathrm{F}_{n,c}(\boldsymbol{x}; n)}{\mathrm{F}_n(\boldsymbol{x}; n)} - \frac{\theta}{1-\theta}\cdot p_n(\boldsymbol{x})\\ &= F_n(\boldsymbol{x}; n)\cdot p_n(\boldsymbol{x})\cdot\left(\frac{1}{1-\theta}\frac{\mathrm{F}_{n,c}(\boldsymbol{x}; n)}{\mathrm{F}_n(\boldsymbol{x}; n)} - \frac{\theta}{1-\theta}\right),\end{aligned} \tag{6}$$

where (a) follows from the fact that $F_n(\boldsymbol{x}; n) = 1$ and $F_{n,c}(\boldsymbol{x}; n) = \theta_{c|n}p_c(\boldsymbol{x})/p_n(\boldsymbol{x})$.

Inductive case #1: suppose Equation 4 holds for m. If product unit d is a parent of m, we show that Equation 4 also holds for d:

$$
\begin{aligned}
&p_d(\boldsymbol{x}) - p'_d(\boldsymbol{x}) \\
&= \prod_{n' \in \mathsf{in}(d)} p_{n'}(\boldsymbol{x}) - \prod_{n' \in \mathsf{in}(d)} p'_{n'}(\boldsymbol{x}) \\
&= (p_m(\boldsymbol{x}) - p'_m(\boldsymbol{x})) \prod_{n' \in \mathsf{in}(d) \setminus m} p_{n'}(\boldsymbol{x}) \\
&\overset{(a)}{=} F_n(\boldsymbol{x}; m) p_m(\boldsymbol{x}) \left(\frac{1}{1-\theta} \frac{\mathrm{F}_{n,c}(\boldsymbol{x}; m)}{\mathrm{F}_n(\boldsymbol{x}; m)} - \frac{\theta}{1-\theta} \right) \\
&\quad \cdot \prod_{n' \in \mathsf{in}(d) \setminus m} p_{n'}(\boldsymbol{x}) \\
&\overset{(b)}{=} F_n(\boldsymbol{x}; d) p_d(\boldsymbol{x}) \left(\frac{1}{1-\theta} \frac{\mathrm{F}_{n,c}(\boldsymbol{x}; d)}{\mathrm{F}_n(\boldsymbol{x}; d)} - \frac{\theta}{1-\theta} \right),
\end{aligned}
$$

where (a) is the inductive step that applies Equation 6; (b) follows from the fact that (note that d is a product unit) $F_n(\boldsymbol{x}; m) = F_n(\boldsymbol{x}; d)$ and $F_{n,c}(\boldsymbol{x}; m) = F_{n,c}(\boldsymbol{x}; d)$.

Inductive case #2: for sum unit d, suppose Equation 4 holds for m, where $m \in \mathcal{A}$ iff $m \in \mathsf{in}(d)$ and m is an ancester of n and c. Assume all other children of d are not ancestoer of n, we show that Equation 4 also holds for d:

$$
\begin{aligned}
&p_d(\boldsymbol{x}) - p'_d(\boldsymbol{x}) \\
&= \theta_{m|d} \cdot (p_m(\boldsymbol{x}) - p'_m(\boldsymbol{x})) \\
&= \theta_{m|d} \cdot F_n(\boldsymbol{x}; m) \cdot p_m(\boldsymbol{x}) \cdot \left(\frac{1}{1-\theta} \frac{\mathrm{F}_{n,c}(\boldsymbol{x}; m)}{\mathrm{F}_n(\boldsymbol{x}; m)} - \frac{\theta}{1-\theta} \right) \\
&= \theta_{m|d} \cdot F_n(\boldsymbol{x}; m) \cdot p_m(\boldsymbol{x}) \cdot \left(\frac{1}{1-\theta} \frac{\mathrm{F}_{n,c}(\boldsymbol{x}; d)}{\mathrm{F}_n(\boldsymbol{x}; d)} - \frac{\theta}{1-\theta} \right) \\
&= \theta_{m|d} \cdot F_n(\boldsymbol{x}; d) \cdot \frac{\sum_{m' \in \mathsf{in}(d)} \theta_{m'|d} p_{m'}(\boldsymbol{x})}{\theta_{m|d} p_m(\boldsymbol{x})} \cdot p_m(\boldsymbol{x}) \\
&\quad \cdot \left(\frac{1}{1-\theta} \frac{\mathrm{F}_{n,c}(\boldsymbol{x}; d)}{\mathrm{F}_n(\boldsymbol{x}; d)} - \frac{\theta}{1-\theta} \right) \\
&= F_n(\boldsymbol{x}; d) \cdot \left(\sum_{m' \in \mathsf{in}(d)} \theta_{m'|d} p_{m'}(\boldsymbol{x}) \right) \cdot \\
&\quad \cdot \left(\frac{1}{1-\theta} \frac{\mathrm{F}_{n,c}(\boldsymbol{x}; d)}{\mathrm{F}_n(\boldsymbol{x}; d)} - \frac{\theta}{1-\theta} \right) \\
&= F_n(\boldsymbol{x}; d) \cdot p_d(\boldsymbol{x}) \\
&\quad \cdot \left(\frac{1}{1-\theta} \frac{\mathrm{F}_{n,c}(\boldsymbol{x}; d)}{\mathrm{F}_n(\boldsymbol{x}; d)} - \frac{\theta}{1-\theta} \right).
\end{aligned}
$$

Therefore, following Equation 4 for root r, we have

$$
\begin{aligned}
&\frac{p_r(\boldsymbol{x}) - p'_r(\boldsymbol{x})}{p_r(\boldsymbol{x})} = \frac{1}{1-\theta} \mathrm{F}_{n,c}(\boldsymbol{x}; r) - \frac{\theta}{1-\theta} F_n(\boldsymbol{x}; r) \\
\Leftrightarrow \quad &\frac{p'_r(\boldsymbol{x})}{p_r(\boldsymbol{x})} = 1 + \frac{\theta}{1-\theta} F_n(\boldsymbol{x}; r) - \frac{1}{1-\theta} \mathrm{F}_{n,c}(\boldsymbol{x}; r)
\end{aligned}
$$

Therefore, we have

$$
\begin{aligned}
&\Delta\mathcal{LL}(\{\boldsymbol{x}\}, \mathcal{C}, \{(n,c)\}) \\
&= \log p_r(\boldsymbol{x}) - \log p'_r(\boldsymbol{x}) \\
&= \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x} \in \mathcal{D}} \log \left(\frac{1 - \theta_{c|n}}{1 - \theta_{c|n} + \theta_{c|n} \mathrm{F}_n(\boldsymbol{x}; r) - \mathrm{F}_{n,c}(\boldsymbol{x}; r)} \right) \\
&\overset{(a)}{\leq} -\log(1 - F_{n,c}(\boldsymbol{x})),
\end{aligned}
$$

where (a) follows from the fact that $F_{n,c}(\boldsymbol{x}) \leq F_n(\boldsymbol{x})$. $\square$

Theorem 1 follows directly from Lemma 1 by noting that for any dataset $\mathcal{D}$, $\Delta\mathcal{LL}(\mathcal{D}, \mathcal{C}, \{(n,c)\}) = \frac{1}{|\mathcal{D}|} \Delta\mathcal{LL}(\{\boldsymbol{x}\}, \mathcal{C}, \{(n,c)\})$.

A.1.2 Pruning Multiple Edge

Proof. Similar to the proof of Lemma 1, we prove Theorem 2 by induction. Different from Lemma 1, we induce a slightly different objective:

$$
\begin{aligned}
&p_m(\boldsymbol{x}) - p'_m(\boldsymbol{x}) \\
&\leq \sum_{(n,c) \in \mathcal{E} \cap \mathsf{des}(m)} F_n(\boldsymbol{x}; m) \cdot p_m(\boldsymbol{x}) \\
&\quad \cdot \left(\frac{1}{1-\theta_{c|n}} \frac{\mathrm{F}_{n,c}(\boldsymbol{x}; m)}{\mathrm{F}_n(\boldsymbol{x}; m)} - \frac{\theta_{c|n}}{1-\theta_{c|n}} \right),
\end{aligned} \tag{7}
$$

where $\mathsf{des}(n)$ is the set of descendent units of n.

Base case: the base case follows directly from the proof of Lemma 1, and lead to the conclusion in Equation 6.

Inductive case #1: suppose for all children of a product unit d, Equation 7 holds, we show that Equation 7 also holds for d:

$$
\begin{aligned}
&p_d(\boldsymbol{x}) - p'_d(\boldsymbol{x}) \\
&= \prod_{m \in \mathsf{in}(d)} p_m(\boldsymbol{x}) - \prod_{m \in \mathsf{in}(d)} p'_m(\boldsymbol{x}) \\
&= \prod_{m \in \mathsf{in}(d)} p_m(\boldsymbol{x}) - \prod_{m \in \mathsf{in}(d)} \Big(p_m(\boldsymbol{x}) - (p_m(\boldsymbol{x}) - p'_m(\boldsymbol{x})) \Big) \\
&\leq \sum_{m \in \mathsf{in}(d)} \Big(p_m(\boldsymbol{x}) - p'_m(\boldsymbol{x})) \Big) \cdot \prod_{m' \in \mathsf{in}(d) \setminus m} p_{m'}(\boldsymbol{x}) \\
&\overset{(a)}{\leq} \sum_{m \in \mathsf{in}(d)} \sum_{(n,c) \in \mathcal{E} \cap \mathsf{des}(m)} \\
&\quad F_n(\boldsymbol{x}; d) \cdot p_d(\boldsymbol{x}) \cdot \left(\frac{1}{1-\theta_{c|n}} \frac{\mathrm{F}_{n,c}(\boldsymbol{x}; m)}{\mathrm{F}_n(\boldsymbol{x}; m)} - \frac{\theta_{c|n}}{1-\theta_{c|n}} \right) \\
&\leq \sum_{(n,c) \in \mathcal{E} \cap \mathsf{des}(d)} \\
&\quad F_n(\boldsymbol{x}; d) \cdot p_d(\boldsymbol{x}) \cdot \left(\frac{1}{1-\theta_{c|n}} \frac{\mathrm{F}_{n,c}(\boldsymbol{x}; d)}{\mathrm{F}_n(\boldsymbol{x}; d)} - \frac{\theta_{c|n}}{1-\theta_{c|n}} \right),
\end{aligned}
$$

where (a) uses the definition that $p_d(\boldsymbol{x}) = \prod_{m \in \mathsf{in}(d)} p_m(\boldsymbol{x})$.

Inductive case #2: suppose for all children of a sum unit d, Equation 7 holds, we show that Equation 7 also holds for d:

$$
\begin{aligned}
&p_d(\boldsymbol{x}) - p'_d(\boldsymbol{x}) \\
&= \sum_{m \in \mathsf{in}(d) \cap (d,m) \notin \mathcal{E}} \theta_{m|d} \Big(p_m(\boldsymbol{x}) - p'_m(\boldsymbol{x}) \Big) \\
&\quad + \sum_{m \in \mathsf{in}(d) \cap (d,m) \in \mathcal{E}} \theta_{m|d} \Big(p_m(\boldsymbol{x}) - p'_m(\boldsymbol{x}) \Big) \\
&\overset{(a)}{=} \sum_{m \in \mathsf{in}(d) \cap (d,m) \notin \mathcal{E}} \theta_{m|d} \Big(p_m(\boldsymbol{x}) - p'_m(\boldsymbol{x}) \Big) \\
&\quad + \sum_{m \in \mathsf{in}(d) \cap (d,m) \in \mathcal{E}} \theta_{m|d} F_n(\boldsymbol{x}; m) p_m(\boldsymbol{x}) A_{n,c}(\boldsymbol{x}, m),
\end{aligned}
$$

where $A_{n,c}(\boldsymbol{x}, m) = \left(\frac{1}{1-\theta_{c|n}} \frac{\mathrm{F}_{n,c}(\boldsymbol{x};m)}{\mathrm{F}_n(\boldsymbol{x};m)} - \frac{\theta_{c|n}}{1-\theta_{c|n}} \right)$, and (a) follows from the base case of the induction. Next, we focus on the first term of the above equation:

$$
\begin{aligned}
&\sum_{m \in \mathsf{in}(d) \cap (d,m) \notin \mathcal{E}} \theta_{m|d} \Big(p_m(\boldsymbol{x}) - p'_m(\boldsymbol{x}) \Big) \\
&\leq \sum_{m \in \mathsf{in}(d) \cap (d,m) \notin \mathcal{E}} \sum_{(n,c) \in \mathcal{E} \cap \mathsf{des}(m)} \\
&\qquad \theta_{m|d} \Big(p_m(\boldsymbol{x}) - p'_m(\boldsymbol{x}) \Big) \\
&\leq \sum_{m \in \mathsf{in}(d) \cap (d,m) \notin \mathcal{E}} \sum_{(n,c) \in \mathcal{E} \cap \mathsf{des}(m)} \\
&\qquad \theta_{m|d} F_n(\boldsymbol{x}; m) p_m(\boldsymbol{x}) A_{n,c}(\boldsymbol{x}, d) \\
&\leq \sum_{(n,c) \in \mathcal{E} \cap \mathsf{des}(d)} F_n(\boldsymbol{x}; d) p_d(\boldsymbol{x}) A_{n,c}(\boldsymbol{x}, d)
\end{aligned}
$$

where the derivation of the last inequality follows from the corresponding steps in the proof of Lemma 1.

Therefore, from Equation 7, we can conclude that

$$
\Delta\mathcal{LL}(\mathcal{D}, \mathcal{C}, \mathcal{E}) \leq -\frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x}} \log(1 - \sum_{(n,c) \in \mathcal{E}} \mathrm{F}_{n,c}(\boldsymbol{x})).
$$

Finally, we prove the approximation step in Equation 3. Let $\epsilon(\cdot) = \sum_{(n,c) \in \mathcal{E}} \mathrm{F}_{n,c}(\cdot) \in [0, 1)$. We have,

$$
\begin{aligned}
\text{RHS} &= -\sum_{\boldsymbol{x} \in \mathcal{D}} \log(1 - \epsilon(\boldsymbol{x})) \\
&= -\sum_{\boldsymbol{x} \in \mathcal{D}} \sum_{k=1}^{\infty} -\frac{\epsilon(\boldsymbol{x})^k}{k} (\text{Taylor expansion}) \\
&\leq \sum_{\boldsymbol{x} \in \mathcal{D}} \sum_{k=1}^{\infty} \epsilon(\boldsymbol{x})^k = \sum_{\boldsymbol{x} \in \mathcal{D}} \frac{\epsilon(\boldsymbol{x})}{1 - \epsilon(\boldsymbol{x})} = \frac{1}{1-\epsilon} \sum_{\boldsymbol{x} \in \mathcal{D}} \epsilon(\boldsymbol{x}) \\
&= \frac{1}{1-\epsilon} \sum_{(n,c) \in \mathcal{E}} \sum_{\boldsymbol{x} \in \mathcal{D}} \mathrm{F}_{n,c}(\boldsymbol{x}) = \frac{1}{1-\epsilon} \sum_{(n,c) \in \mathcal{E}} \mathrm{F}_{n,c}(\mathcal{D}).
\end{aligned}
$$

□

B PSEUDOCODE

In this section, we list the detailed algorithms of pruning operation (Section 3), circuit flows computation (Definition 4), and mini-batch Expectation Maximization (Section 4).

Algorithm 2 shows how to prune k percentage edges from PC $\mathcal{C}$ following heuristic h.

Algorithm 2: Prune($\mathcal{C}$, h, k)

Input : a non-deterministic PC $\mathcal{C}$, heuristic h deciding which edge to prune, h can be EFLOW, ERAND, or EPARAM, percentage of edges to prune k
Output : a PC $\mathcal{C}$' after pruned

1 `old2new` $\leftarrow$ mapping from input PC $n \in \mathcal{C}$ to pruned PC
2 $s(n, c) \leftarrow$ compute a score for each edge (n, c) based on heuristic h
3 $f(n, c) \leftarrow false$
4 $f(n, c) \leftarrow true$ if $s(n, c)$ ranks the last k
5 `// visit children before parents`
6 **foreach** $n \in \mathcal{C}$ **do**
7 **if** n *is a leaf* **then**
8 `old2new`$[n] \leftarrow n$
9 **else if** n *is a sum* **then**
10 `old2new`$[n] \leftarrow \bigoplus$([`old2new`(c) for $c \in$ $\mathsf{in}(n)$ and if $f(n, c)$])
11 **else** n is a product
12 `old2new`$[n] \leftarrow \bigotimes$([`old2new`$(c)$ for $c \in$ $\mathsf{in}(n)$])
13 **return** `old2new`$[n_r]$ where n_r is the root of $\mathcal{C}$

Algorithm 3 computes the circuit flows of a sample $\boldsymbol{x}$ given PC $\mathcal{C}$ with parameters $\boldsymbol{\theta}$ though one forward pass (line 1) and one backward pass (line 2-8).

Algorithm 3: CircuitFlow($\mathcal{C}$,$\boldsymbol{\theta}$,$\boldsymbol{x}$)

Input : a PC $\mathcal{C}$ with parameters $\boldsymbol{\theta}$; sample $\boldsymbol{x}$
Output : circuit flow $\mathsf{flow}[n, c]$ for each edge (n, c) and $\mathsf{flow}[n]$ for each node n

1 $\forall n \in \mathcal{C}$, $\mathsf{p}[n] \leftarrow p_n(\boldsymbol{x})$ computed as in Equation 1
2 For root n_r, $\mathsf{flow}[n] \leftarrow 1$
3 **for** $n \in \mathcal{C}$ *in backward order* **do**
4 $\mathsf{flow}[n] \leftarrow \sum_{g \in \mathsf{pa}(n)} \mathsf{flow}[g]$
5 **if** n *is a sum node* **then**
6 $\forall c \in \mathsf{in}(n), \mathsf{flow}[n, c] \leftarrow \theta_{c|n} \frac{\mathsf{p}[c]}{\mathsf{p}[n]} \mathsf{flow}[n]$
7 **else**
8 $\forall c \in \mathsf{in}(n), \mathsf{flow}[n, c] \leftarrow \mathsf{flow}[n]$

Algorithm 4 shows the growing operation.

Algorithm 4: Grow($\mathcal{C}$, σ^2)

Input : a PC $\mathcal{C}$, Gaussian noisy variance σ^2
Output : a PC $\mathcal{C}$' after growing operation

```
old2new ← a dictionary mapping input PC units n ∈ C to units of the growed PC
foreach n ∈ C do  // visit children before parents
    if n is an input unit then
        old2new[n] ← (n, deepcopy(n))
    else
        chs_1 ← old2new[c][0] for c in in(n)]
        chs_2 ← old2new[c][1] for c in in(n)
        if n is a product unit then
            old2new[n] ← (⊗(chs_1), ⊗(chs_2))
        else if n is a sum unit then
            n_1 ← ⊕([chs_1, chs_2])
            n_2 ← ⊕([chs_1, chs_2])
            ε ~ N(1, σ²) for i in [1, 2]
            θ_{|n_i} ← normalize([θ_{|n}, θ_{|n}]) × ε)
            old2new[n] ← (n_1, n_2)
return old2new[r][0] // r is the root of C
```

Algorithm 5 shows the pipeline of mini-batches Expectation Maximization algorithm given PC $\mathcal{C}$, dataset $\mathcal{D}$, batch size B and learning rate α.

Algorithm 5: StochasticEM($\mathcal{C}$,$\mathcal{D}$;B,α)

Input : a PC $\mathcal{C}$; dataset $\mathcal{D}$; batch size B; learning rate α
Output : parameters $\boldsymbol{\theta}$ estimated from $\mathcal{D}$

```
θ ← random initialization
For root n_r, flow[n] ← 1
while not converged or early stopped do
    D' ← B random samples from D
    flow ← Σ_{x∈D'} CircuitFlow(C, θ, x)
    for sum unit n and its child c do
        θ^{new}_{c|n} ← flow[n, c]/flow[n]
        θ_{c|n} ← α θ^{(new)}_{c|n} + (1 − α) θ_{c|n}
```

C EXPERIMENTS DETAILS

Parameter Estimation. We use a stochastic mini-batch version of Expectation-Maximization optimization [Choi et al., 2020a]. 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 $\theta^{\texttt{new}}_{c|n} = \mathrm{F}_{n,c}(\mathcal{D}_B)/\mathrm{F}_n(\mathcal{D}_B)$, and update the targeting parameter with a learning rate α: $\theta^{t+1} \leftarrow \alpha\theta^{\texttt{new}} + (1-\alpha)\theta^t$ (M-step). Empirically this approach converges faster and is more regularized compared to full-batch EM.

Parallelism Computation. Existing approaches to scale up learning and inferences of PCs such as Einsum networks [Peharz et al., 2020a] utilize fully connected parametrized layers (Figure 3a) of PC structures such as HCLT [Liu and Van den Broeck, 2021] and RatSPN [Peharz et al., 2020b]. These structures can be easily vectorized to utilize deep learning packages such as PyTorch. However, the sparse structure learned by pruning and growing is not easily vectorized as a dense matrix operation. We therefore implement customized GPU kernels to parallelize the computation of parameter learning and inferences based on Juice.jl [Dang et al., 2021], an open-source Julia package for learning PCs. The kernels segment PC units into layers such that the units in each layer are independent thus the computation can be fully parallelized in the GPU. As a result, we can train PCs with millions of parameters in less than half an hour.

Hardware specifications All experiments are performed on a server with 32 CPUs, 126G Memory, and NVIDIA RTX A5000 GPUs with 26G Memory. In all experiments, we only use a single GPU on the server.

C.1 DATASETS

For MNIST-family datasets, we split 5% of training set as validation set for early stopping. For Penn Tree Bank dataset, we follow the setting in Mikolov et al. [2012] to split a training, validation, and test set. Table 3 lists the all the dataset statistics.

For Penn Tree Bank dataset, we use the standard processing from Mikolov et al. [2012]. , which contains around 5M characters and a character-level vocabulary size of $k = 50$. The data is split into sentences with a maximum sequence length of $n = 288$.

C.2 LEARNING HIDDEN CHOW-LIU TREES

HCLT structures. Adopting hidden chow liu tree (HCLT) PC architecture as in Liu and Van den Broeck [2021], we reimplement the learning process to speed it up and use a different training pipeline and hyper-parameters tuning.

EM parameter learning We adopt the EM parameter learning algorithm introduced in Choi et al. [2021], which computes the EM update target parameters using circuit flows. We use a stochastic mini-batches EM algorithm. Denoting $\theta^{\texttt{new}}$ as the EM update target computed from a mini-batch of samples, and we update the targeting parameter with a learning rate α: $\theta^{t+1} \leftarrow \alpha\theta^{\texttt{new}} + (1-\alpha)\theta^t$. α is piecewise-linearly annealed from $[1.0, 0.1]$, $[0.1, 0.01]$,

Table 3: Dataset statistics including number of variables (**#vars**), number of categories for each variable (**#cat**), and number of samples for training, validation and test set (**#train**, **#valid**, **#test**).

Dataset	n (**#vars**)	k (**#cat**)	**#train**	**#valid**	**#test**
MNIST	28×28	256	57000	3000	10000
EMNIST(MNIST)	28×28	256	57000	3000	10000
EMNIST(Letters)	28×28	256	118560	6240	20800
EMNIST(Balanced)	28×28	256	107160	5640	18800
EMNIST(ByClass)	28×28	256	663035	34897	116323
FashionMNIST	28×28	256	57000	3000	10000
Penn Tree Bank	288	50	42068	3370	3761

$[0.01, 0.001]$, and each piece is trained T epochs.

Hyper-parameters searching. For all the experiments, the hyper-parameters are searched from

- $h \in \{8, 16, 32, 64, 128, 256\}$, the hidden size of HCLT structures;
- $\gamma \in \{0.0001, 0.001, 0.01, 0.1, 1.0\}$, Laplace smoothing factor;
- $B \in \{128, 256, 512, 1024\}$, batch-size in mini-batches EM algorithm;
- α piecewise-linearly annealed from $[1.0, 0.1]$, $[0.1, 0.01]$, $[0.01, 0.001]$, where each piece is called one mini-batch EM phase. Usually the algorithm will start to overfit as validation set and stop at the third phase;
- $T = 100$, number of epochs for each mini-batch EM phase.

The PC size is quadratically growing with hidden size h, thus it is inefficient to do a grid search among the entire hyper-parameters space. What we do is to fist do a grid search when $h = 8$ or $h = 16$ to find the best Laplace smoothing factor γ and batch-size B for each dataset, and then fix γ and B to train a PC with larger hidden size $h \in \{32, 64, 128, 256\}$. The best tuned B is in $\{256, 512\}$, which is different for different hidden size h, and the best tuned γ is 0.01.

C.3 DETAILS OF BENCHMARKS

Sparse PC (ours). Given HCLT learned in Section C.2 as initial PC, we use the structure learning process proposed in Section 4. Specifically, starts from initial HCLT, for each iteration, we (1) prune 75% of the PC parameters, and (2) grow PC size with Gaussian variance ϵ, (3) finetuing PC using mini-batches EM parameter learning with learning rate α. We prune and grow PC iteratively until the validation set likelihood is overfitted . The hyper-parameters are searched from

- $\epsilon \in \{0.1, 0.3, 0.5\}$, Gaussian variance in growing operation;
- α, piecewise-linearly annealed from $[0.1, 0.01]$, $[0.01, 0.001]$;
- $T = 50$, number of epochs for each mini-batch EM phase;
- for γ and B, we use the tuned best number from Section C.2.

HCLT. The HLCT experiments in Table 1 are performed following the original paper (Code `https://github.com/UCLA-StarAI/Tractable-PC-Regularization`), which is different from the leaning pipeline we use as our inital PC (Section C.2).

SPN. We reimplement the SPN architecture ourselves following Peharz et al. [2020b] and train it with the same mini-batch pipeline as HCLT.

IDF. We run all experiments with the code in the GitHub repo provided by the authors. We adopt an IDF model with the following hyperparameters: 8 flow layers per level; 2 levels; densenets with depth 6 and 512 channels; base learning rate 0.001; learning rate decay 0.999. The algorithm adopts an CPU-based entropy coder rANS.

BitSwap. We train all models using the following author-provided script: `https://github.com/fhkingma/bitswap/blob/master/model/mnist_train`.

BB-ANS. All experiments are performed using the following official code `https://github.com/bits-back/bits-back`.

McBits. All experiments are performed using the following official code `https://github.com/ryoungj/mcbits`.

C.4 EVALUATING PRUNING AND GROWING

What is the Smallest PC for the Same Likelihood? We evaluation the ability of pruning operations based on circuit flows (Section 3) to do effective model compression by iteratively pruning $k\%$ 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 $k\%$ ranging between $\{0.05, 0.1, 0.3\}$. As shown in Figure 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 number of latent states (legend), we discover that compressing a large PC to a get smaller PC has better likelihoods compared to directly training a HCLT with the same number of parameters from scratch, due to the sparsity of compressed PC structures, as well as the smarter way to find good parameters: finding a better PC with larger size and compress it to smaller one.

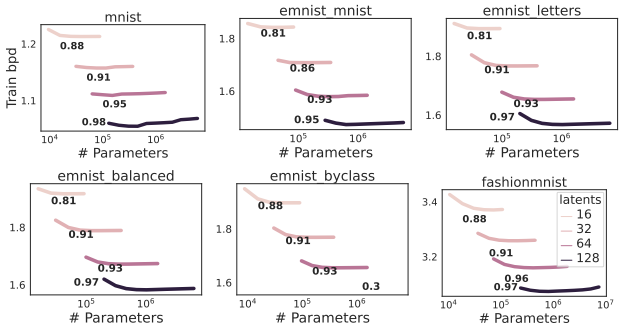


Figure 6: Model compression via pruning and finetuning. We report the training set bpd (y-axis) in terms of the number of parameters (x-axis) for different number of latent states. For each curve, compression starts from the right (initial PC #Params $|\mathcal{C}^{\texttt{init}}|$) and ends at the left (compressed PC #Params $|\mathcal{C}^{\texttt{com}}|$); compression rate (1 - $|\mathcal{C}^{\texttt{com}}|$ / $|\mathcal{C}^{\texttt{init}}|$) is annotated next to each curve.

What is the Best PC Given the Same Size? We evaluate structure learning methods combining pruning and growing proposed in Section 4. Starts from initial HCLT, we iteratively prune 75% of the parameters, growing again, and finetuning until meeting the stopping criteria. As shown in Figure 7, our method consistently improve the likelihoods of initial PCs for different number of latent states among all datasets.

C.5 DETAILS OF SECTION C.4

For all experiments in Section C.4, we use the best tuned γ and B from Section C.2 and hidden size h ranging from $\{16, 32, 64, 128\}$. For experiments "What is the Smallest PC for the Same Likelihood?", the hyper-parameters are searched from

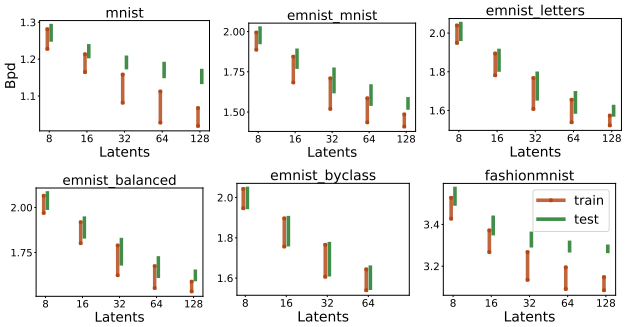


Figure 7: Structure learning via 75% pruning, growing and finetuning. We report bpd (y-axis) on both train (red) and test set (green) in terms of number of latent states (x-axis). For each curve, training starts from the top (large bpd) and ends at the bottom (small bpd).

- $k\% \in \{0.05, 0.1, 0.3\}$, percentage of parameters to prune each iteration;
- α, piecewise-linearly annealed from $[0.3, 0.1]$, $[0.1, 0.01]$, $[0.01, 0.001]$;
- $T = 50$, number of epochs for each mini-batch EM phase;

For experiments "What is the Best PC Given the Same Size?", we use the same setting as in Section C.3.