Conditionally Tractable Density Estimation using Neural Networks

Abstract

Tractable models such as cutset networks and sum-product networks (SPNs) have become increasingly popular as they admit polynomial time inference in some cases. Among them, cutset networks, which model the mechanics of Pearl's cutset conditioning algorithm, demonstrate great scalability and prediction accuracy. Existing research on cutset networks has mainly focused on discrete domains, and the best mechanism to extend cutset networks to continuous domains is unclear. We propose one possible alternative to cutset networks that models the full joint distribution as the product of a general, complex distribution over a small subset of variables and a fully tractable conditional distribution whose parameters are controlled by a neural network. This model admits exact inference when all variables in the general distribution are observed, and although the model is not fully tractable in general, we show that "cutset" sampling can be employed to efficiently generate accurate predictions in practice. We show that our model performs comparably or better than existing competitors on a variety of real datasets.

1 INTRODUCTION

Tractable probabilistic models such as cutset networks Rahman et al. [2014], sum-product networks Poon and Domingos [2011], arithmetic circuits Darwiche [2003] and probabilistic sentential decision diagrams Kisa et al. [2014] have gained popularity in recent years, primarily because of their ability to accurately answer various reasoning queries in linear time in the size of the model while maintaining other desirable properties such as high expressivity and flexibility. However, to date, with a few exceptions Poon and Domingos [2011], Molina et al. [2018], Uria et al. [2013], a majority of work on learning tractable models has focused on discrete random variables and not on continuous variables. This is primarily due to the fact that, unlike discrete domains, guaranteeing tractability of models in continuous domains while also maintaining their high expressivity and flexibility is very challenging.

Existing tractable models for continuous variables operate under the assumption that the underlying data can be modeled using a Gaussian mixture. For instance, sum-product networks Poon and Domingos [2011], Molina et al. [2018], Uria et al. [2013] use latent discrete (variable) architectures distributed over sum and product nodes via the *distributive law* to efficiently represent a Gaussian mixture. Unfortunately, these architectures often exhibit poor predictive performance despite the fact that their average test set log-likelihood scores can be quite high. This is a well-known phenomena in continuous models where test set likelihood scores are unable to discriminate low-performing models from high-performing ones. This is because unlike discrete likelihood which always lies between 0 and 1, the continuous likelihood can be potentially infinite or close to it, and thus even one test point that has high likelihood may skew the numbers.

Our aim then, in this work, is to design a tractable model for continuous domains that is flexible enough to be broadly applicable while also yielding good predictive performance in practice. To this end, we propose a novel tractable model for continuous domains motivated by cutset networks (CNs) Rahman et al. [2014]. CNs and their subsequent extensions Roy et al. [2021], Rahman and Gogate [2016] make use of cutset conditioning Pearl [2014]: select a set of variables to condition on (namely cutset variables), such that the distribution over the rest of the variables (leaf variables) can be well-represented by tractable tree structured Bayesian Networks (BNs) Chow and Liu [1968]. Cutset networks are expressive and have shown great scalability and predictive accuracy in high-dimensional discrete domains. A natural extension of cutset networks to continuous domains is to condition each cutset variable over a finite

Submitted to the 37th Conference on Uncertainty in Artificial Intelligence (UAI 2021). **To be used for reviewing only**.

number of ranges, similar to decision tree regressors, and then model the leaf distributions using conditional linear Gaussian (CLGs) Bayesian networksGrzegorczyk [2010]. Although simple and appealing, in this representation it is hard to ensure that the number of cutset variables is bounded by a small number in order to guarantee tractability without incurring a significant loss in expressive power, and partitioning the space into rectangular blocks is not natural in many practical applications.

To overcome these limitations of cutset networks in continuous domains, and motivated by the work of Rahman et al. [2019] on cutset Bayesian networks, we propose to model the joint distribution as a product of two distributions: (1) a general, complex unconditional distribution defined over a small subset $\mathbf{X}$ of variables and (2) a fully tractable conditional distribution defined over the remaining variables $\mathbf{Y}$ whose parameters are controlled by a neural network. Like cutset Bayesian networks, this model admits exact inference when all variables in the set $\mathbf{X}$ are observed, and although the model is not fully tractable, we show that "cutset" sampling can be employed to efficiently generate accurate predictions in practice. We demonstrate the competitive performance of our approach against two well-known models in the literature: SPNs Lorenzo [2020], Molina et al. [2019], which use latent discrete sum-product architectures and RNADE Uria et al. [2013], which uses neural density estimators, on a prediction task generated from real datasets.

To summarize our contributions, we present (1) a novel tractable architecture, inspired by cutset networks, (2) a novel neural network architecture for choosing the parameters of a conditional linear Gaussian, (3) and an extensive study (90 different experiments) to evaluate the (marginal MAP) prediction accuracy of our model, SPNs, and RNADE on a variety of real datasets.

2 CONDITIONALLY TRACTABLE DENSITY ESTIMATION

In this section, we describe our approach to building tractable models for continuous domains. Note that we use bold uppercase letters for a set of random variables, e.g., $\boldsymbol{X}$, while a single random variable is denoted using uppercase letters, e.g., A. The instantiation (configuration) of random variables are denoted as lowercase letters. For example, $\boldsymbol{x}$ is one possible configuration for all variables in $\boldsymbol{X}$ and a is a possible value that the random variable A can take. All random variables considered in this paper are assumed to be defined over the real domain $\mathbb{R}$ unless otherwise noted.

Given a set of random variables $\boldsymbol{Z}$, we model the full joint distribution over $\boldsymbol{Z}$ as the product of a general, complex distribution $p(\boldsymbol{X})$ over a small subset of variables $\boldsymbol{X} \subset \boldsymbol{Z}$ and a fully tractable conditional distribution $p(\boldsymbol{Y}|\boldsymbol{X})$ over variables $\boldsymbol{Y} = \boldsymbol{Z} \backslash \boldsymbol{X}$ whose parameters are controlled by a neural network. Specifically, we use a mixture of multivariate Gaussian (MixMG) distribution to model $p(\boldsymbol{X})$, and a fully connected Gaussian Bayesian network (fGBN) to model the distribution $p(\boldsymbol{Y}|\boldsymbol{X})$. In addition, we assume that the parameters of the fGBN are given by a neural network (NN) that takes the values of $\boldsymbol{X}$ as input and outputs all the parameters inside the fGBN model, i.e., $p(\boldsymbol{Y}|\boldsymbol{X} = \boldsymbol{x}) \sim fGBN(\boldsymbol{Y}; \theta)$ where $\theta = NN(\boldsymbol{x})$. Note that any GBN can be converted into an equivalent Multivariate Gaussian (MG) distribution with full covariance matrix Koller and Friedman [2009]. Therefore, the conditional distribution $p(\boldsymbol{Y}|\boldsymbol{X})$ is fully tractable, which means our model admits tractable inference when all variables in the general distribution $p(\boldsymbol{X})$ are observed. In the cases where $\boldsymbol{X}$ is not fully observed, cutset sampling can be employed to efficiently generate accurate predictions in practice.

In the following sections, we will first introduce the detailed architecture of our model, this includes how we select the conditional variables $\boldsymbol{X}$ from $\boldsymbol{Z}$, as well as the architecture of the NN, and then we will discuss learning and inference algorithms.

2.1 DESIGN OF PARAMETER GENERATION NEURAL NETWORK

In this work, we propose a neural network architecture that we dub parameter generation neural networks (PGNNs) that can be used to generate the parameters for other models (specifically GBNs), see Figure 1. PGNNs are composed of a series of building blocks that are used to extract a shared feature vector stage by stage and a collection of headers that are used to compute the parameters based on the feature vector. The architecture consists of a basic building block (BBlock) that is used to extract non-linear features. It consists of three layers: (1) a (fully connected) linear layer that produces linear features, (2) a one-dimension batch normal layer, and (3) finally a ReLU layer. The Header blocks are used to refine the shared features and include a linear layer that is used to compute the parameters. We stack $s + 1$ building blocks to extract a feature vector of size k that is shared by all Headers. Given n raw input features, the first building block creates a large and expressive feature vector of size $k \cdot r^s$, and the following s building blocks compress the large vector stage by stage until we reach a feature vector with size k. The size of the feature vector is reduced by a factor of r each time it passes through a building block.

Once the final feature vector is built, the individual headers (output layers) generate the parameters for the underlying model. Note that we do not use a single giant header for all parameters to encourage diversity among the different types of generated parameters. Similarly, the model does not include a header for each parameter as, due to the large number of parameters, the computation complexity can be pro-

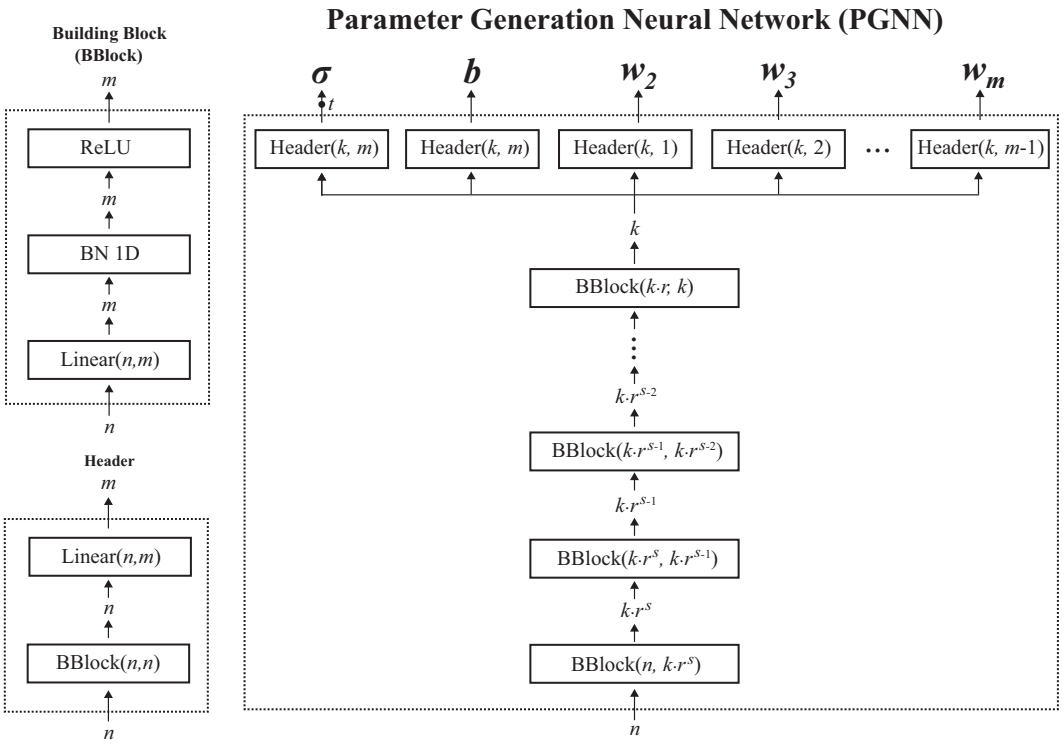

Figure 1: Structure of Parameter Generation Neural Network (PGNN), n is the size of input feature, k is the size of extracted feature, r is the compress ratio and s controls number of stages.

hibitive and this could lead to overfitting. The compromise, which we adopt here, is to use multiple smaller headers, one for each group of parameters. Note that the grouping of parameters is highly dependent on the underlying model. Here, we are generating the parameters of a fully connected GBN over m variables and the conditional distribution of X_j given its parents $\boldsymbol{X}_{1..j-1}$ is characterized by a conditional linear Gaussian (CLG), i.e.,

$$p(X_j|\boldsymbol{X}_{1..j-1} = \boldsymbol{x}_{1..j-1}) \sim N(X_j; \mu = \boldsymbol{w}_j^T \cdot \boldsymbol{x}_{1..j-1} + b_j, \sigma_j^2).$$

The mean of the normal distribution over X_j is a linear function of its parents controlled by the weight vector $\boldsymbol{w}_j^T$ and a bias term b_j and σ_j to denotes the conditional standard deviation, which must be greater than zero. We first create a group $\{\sigma_i|i = 1..m\}$ and use a single header to predict all standard deviations. Then, we do the same for all bias terms $\{b_i|i = 1..m\}$. And finally, we construct one header for each of the weight vectors $\boldsymbol{w}_i$. Note that there is no $\boldsymbol{w}_1$ since X_1 has no parents and the μ is determined solely by the bias term b_1. In addition, the output of the $\boldsymbol{\sigma}$ header clipped to a predefined threshold $t > 0$ in order to guarantee validity of $\boldsymbol{\sigma}$ as well as to ensure numerical stability.

2.2 VARIABLE SELECTION METHODS

As was the case for cutset networks, the selection of the conditional variables, X, can have a significant impact on the quality of the final model. We need to determine both the number of conditional variables to select as well as which specific variables. In general, the expressiveness of the model increases as the number of conditional variables and the number of mixture components in $p(X)$ increases. As picking too many or too few random variables to condition on can lead to overfitting or underfitting, we use a validation set to automatically determine the best number of variables to condition on.

Once the size of X is fixed, we also need to determine which variables to pick in order to achieve the the best practical performance. Exhaustive search is prohibitive as there are $\binom{n}{k}$ possible choices if we want to choose $k < n$ variables out of a total of n variables. Conversely, randomly selecting k variables to be the conditional variables is efficient, but some random variables may not be informative for the prediction task. In some settings domain expertise can be used to select the conditional variables, but this expertise is not always available in practice. Consequently, we propose two sets of heuristic methods for selecting the conditional variables $\boldsymbol{X}$.

The first set of heuristic methods select the variables based on the amount of variance explained by each features in the dataset. Specifically, we use Principal Components Analysis (PCA) as a feature selection method to determine the amount of variance explained by each feature (variable).

After that, we rank those features based on how much variance they explain from highest to lowest. Intuitively, the more variance a feature explains, the more informative it is. One reasonable heuristic would be select variables that have the largest amount of information, which gives us the first heuristic method `MostVar` that selects the first k variables based on the rank. Note that different variables might share some similar information, especially among variables that explain the most variance. Hence, we also consider an alternative heuristic method called `MixVar` that selects k evenly spaced variables based on the rank order. Compared to `MostVar`, `MixVar` produces a more diverse set of variables that have both high and low variance, but the total useful information might be higher than that of `MostVar`. Another alternative to `MixVar` is to randomly select k variables proportional to their contribution to the total variance.

The second set of heuristic methods selects conditional variables based on the correlations between each pair of variables. Recall that, the parameters of the distribution $p(\boldsymbol{Y}|\boldsymbol{X})$ are controlled by a neural network that takes $\boldsymbol{x}$ as input. If the $\boldsymbol{X}$ random variables have no correlation with the $\boldsymbol{Y}$ variables, the neural network will be unable to produce an appropriate guess for the model parameters. In other words, we want all $Y_i \in \boldsymbol{Y}$ to be highly correlated to some or all variables in the set $\boldsymbol{X}$. With this idea in mind, we designed our third variable selection heuristic `MaxMincorrHard`. This heuristic initializes $\boldsymbol{X}$ as an empty set $\varnothing$ and initializes $\boldsymbol{Y} = \boldsymbol{Z}$. Then it iteratively selects a variable $A \in \boldsymbol{Y}$ and adds it into $\boldsymbol{X}$ such that the minimum correlation between those two sets is maximized. Specifically, the algorithm first calculates the Pearson correlation coefficient $R_{A,B}$ to measure the degree of correlation between each pair of random variables A and B. Then, for each variable $V \in \boldsymbol{Y}$, it calculates the minimal correlation of all variable $Y_i \in \{\boldsymbol{Y} \backslash V\}$ to the set $\boldsymbol{S} = \boldsymbol{X} \cup V$. Note that the correlation of a single variable Y_i to a set of variables $\boldsymbol{S}$ is defined as $R_{Y_i,\boldsymbol{S}} = \sum_{S_i \in \boldsymbol{S}} R_{Y,S_i}$. After that, the variable A that maximizes the minimal correlation will be picked as conditional the variable and added into the set $\boldsymbol{X}$ before the next iteration starts, i.e.,

$$A \in \underset{V}{\operatorname{argmax}} \left(\min_{Y_i \in \{\boldsymbol{Y} \backslash V\}} R_{Y_i,\boldsymbol{X} \cup V} \right).$$

Finally, we also consider another variable picking strategy named `MaxMincorrSoft`. It follows the same iterative strategy as `MaxMincorrHard` but uses a slightly different criteria to select conditional variables. Specifically, in addition to maximizing the minimum correlation, `MaxMincorrSoft` also takes the correlation of V to all variables $Y_i \in \{\boldsymbol{Y} \backslash V\}$ into consideration. Obviously, if $R_{V,\boldsymbol{Y} \backslash V}$ is high, it means V contains a lot of useful information about other variables in $\boldsymbol{Y}$, and this should encourage the algorithm to select V as the next conditional variable. In short, `MaxMincorrSoft` selects variable B to be the conditional variable where

$$B \in \underset{V}{\operatorname{argmax}} \left\{ R_{V,\boldsymbol{Y} \backslash V} * \left(\min_{Y_i \in \{\boldsymbol{Y} \backslash V\}} R_{Y_i,\boldsymbol{X} \cup V} \right) \right\}.$$

We have introduced four heuristic methods for selecting the conditional variables $\boldsymbol{X}$. Again, as the best strategy may be dataset dependent, we make it a tunable hyperparameter and use a validation set to automatically determine the variable picking method.

2.3 INFERENCE

In this paper, we focus on marginal Max-A-Posterior (MMAP) inference, which reflects the model's predictive performance in the case of missing values. Given evidence $\boldsymbol{Z}_e = \boldsymbol{z}_e$, variables of interest (query variables) $\boldsymbol{Z}_q$ and missing variables $\boldsymbol{Z}_m$, MMAP inference aims to find the best assignment $\boldsymbol{z}_q^*$ such that $p(\boldsymbol{Z}_q|\boldsymbol{z}_e) = \int_{\boldsymbol{Z}_m} p(\boldsymbol{Z}_q, \boldsymbol{Z}_m|\boldsymbol{z}_e) d\boldsymbol{Z}_m$ is maximized. As our model has partitioned the variables into two sets, we denote the query, missing, and evidence variables from the general distribution as $\boldsymbol{X}_q, \boldsymbol{X}_m, \boldsymbol{X}_e$ and those in the conditional part as $\boldsymbol{Y}_q, \boldsymbol{Y}_m, \boldsymbol{Y}_e$. The inference task can be formulated as finding $\boldsymbol{x}_q^*$ and $\boldsymbol{y}_q^*$ such that

$$\begin{aligned}
\boldsymbol{x}_q^*, \boldsymbol{y}_q^* &\in \underset{\boldsymbol{x}_q, \boldsymbol{y}_q}{\operatorname{argmax}} \int_{\boldsymbol{x}_m} \int_{\boldsymbol{y}_m} p(\boldsymbol{x}_{q,m}, \boldsymbol{y}_{q,m}|\boldsymbol{x}_e, \boldsymbol{y}_e) d\boldsymbol{y}_m d\boldsymbol{x}_m \\
&\in \underset{\boldsymbol{x}_q, \boldsymbol{y}_q}{\operatorname{argmax}} \int_{\boldsymbol{x}_m} \int_{\boldsymbol{y}_m} p(\boldsymbol{x}_{q,m,e}, \boldsymbol{y}_{q,m,e}) d\boldsymbol{y}_m d\boldsymbol{x}_m \\
&\in \underset{\boldsymbol{x}_q, \boldsymbol{y}_q}{\operatorname{argmax}} \int_{\boldsymbol{x}_m} \int_{\boldsymbol{y}_m} p_{\boldsymbol{X}}(\boldsymbol{x}_{q,m,e}) p_{\boldsymbol{Y}}(\boldsymbol{y}_{q,m,e}|\boldsymbol{x}_{q,m,e}) d\boldsymbol{y}_m d\boldsymbol{x}_m \\
&\in \underset{\boldsymbol{x}_q, \boldsymbol{y}_q}{\operatorname{argmax}} \int_{\boldsymbol{x}_m} p_{\boldsymbol{X}}(\boldsymbol{x}_{q,m,e}) p'_{\boldsymbol{Y}}(\boldsymbol{y}_{q,e}|\boldsymbol{x}_{q,m,e}) d\boldsymbol{x}_m
\end{aligned}$$

where $P'_{\boldsymbol{Y}}$ is the marginal distribution obtained by integrating out $\boldsymbol{Y}_m$ from distribution $P_{\boldsymbol{Y}}$ and $\boldsymbol{x}_{q,m}$ denotes the union of $\boldsymbol{x}_q$ and $\boldsymbol{x}_m$.

The difficulty of the above MMAP problem varies depends on whether $\boldsymbol{X}_q$ and $\boldsymbol{X}_m$ are empty sets.

Case 1: All $\boldsymbol{X}$ variables are observed. This means $\boldsymbol{X}_q, \boldsymbol{X}_m = \varnothing$, and the distribution over the remaining variables $\boldsymbol{Y}$ is an $fGBN$ with parameters $\theta = NN(\boldsymbol{x})$. In this case, we can do *exact* MMAP inference by converting the $fGBN$ into a multivariate Gaussian distribution.

Case 2: $\boldsymbol{X}_q \neq \varnothing$ while $\boldsymbol{X}_m = \varnothing$. In this case, since the joint distribution is not tractable, strategy similar to cutset sampling can be employed to efficiently answer the above query. Compared to standard sampling techniques that sample all the variables, cutset sampling only samples a subset of variables and does exact inference over the remaining

variables, which makes the prediction more accurate Rahman et al. [2019]. In our case, we can first generate n samples $\boldsymbol{x}_q^{(1)}, ..., \boldsymbol{x}_q^{(n)}$ for $\boldsymbol{X}_q$ from the distribution $p_{\boldsymbol{X}}$ given evidence $\boldsymbol{x}_e$. After that, for each $\boldsymbol{x}_q^{(i)}$, we do exact MMAP inference on $p'_{\boldsymbol{Y}}$ to find the best assignment $\boldsymbol{y}_q^{(i)}$ for $\boldsymbol{Y}_q$ such that $P'_{\boldsymbol{Y}}(\boldsymbol{y}_q^{(i)}, \boldsymbol{y}_e | \boldsymbol{x}_q^{(i)}, \boldsymbol{x}_e)$ is maximized. Finally, we evaluate the joint density for each pair $(\boldsymbol{x}_q^{(i)}, \boldsymbol{y}_q^{(i)})$ along with the evidence and treat the one with maximum density as the MMAP result. Note that the size of $\boldsymbol{X}$ is usually small in our case. Therefore, the number of variables being sampled is small, which enables the cutset sampling inference to generate high quality approximations in practice.

Case 3: There are missing variables in $\boldsymbol{X}$, which means $\boldsymbol{X}_m \neq \varnothing$. In this case, we treat those missing variables as part of the query variables and use the cutset sampling method described in Case 2 to find best assignment $\boldsymbol{x}^*_{q,m}$ for both $\boldsymbol{X}_q$ and $\boldsymbol{X}_m$. Then we simply discard $\boldsymbol{x}^*_m$ and return $\boldsymbol{x}^*_q$ as the MMAP result for $\boldsymbol{X}_q$. This is a popular method in practice used for approximating MMAP by using the partial assignment from the MAP results Liu and Ihler [2013], Poon and Domingos [2011].

2.4 LEARNING

Our model can be learned with two steps given a dataset D over variables $\boldsymbol{Z}$ and conditional variables $\boldsymbol{X} \subset \boldsymbol{Z}$. First, we learn a mixture of Gaussians over just the X variables (using the expectation maximization algorithm), where the number of components can be a fixed number or automatically determined through simple tuning. Then, we train a PGNN for generating the parameters of the underlying $fGBN$ model defined over random variables $\boldsymbol{Y} = \boldsymbol{Z} \backslash \boldsymbol{X}$, using the negative log-likelihood of the dataset D as the loss function.

3 EXPERIMENTS

We evaluated our model's MMAP inference prediction performance, as measured with respect to root mean square error (RMSE), on ten publicly available datasets from the UCI Machine Learning Repository Dua and Graff [2017]. Following Uria et al. [2013], we pre-process the datasets by eliminating discrete valued features and one of the attributes from every pair of attributes whose Pearson correlation coefficient is greater than 0.98. The number of instances and features for each dataset after pre-processing are shown in Table 1. The number of instances range from 5,875 to 150,000 while the number of features range from 12 to 117. All datasets were normalized by subtracting the mean and dividing by the standard deviation.

We consider four competitors in our experiments.

Chow-Liu GBNs (CLGBNs) are GBNs where the underlying structure is a tree learned using the Chow-Liu algorithm. This is considered as a baseline model.

Multivariate Gaussians (MGs) with full covariance matrix. Another baseline that is equivalent to a fully connected GBN.

Real-valued Neural Autoregressive Density Estimators (RNADEs) Uria et al. [2013, 2016] are equivalent to a fully connected Bayesian Network where each of the conditional distributions that define the model are given by a mixture of multivariate Gaussians whose parameters are controlled by neural networks. For MMAP inference, we used two approximate inference algorithms for RNADE. The first one is based on sampling, where we employ likelihood weighted sampling to generate N samples for both the query and the missing variables given the evidence, and then return the sample that achieves the highest likelihood as the prediction result. We denote this algorithm as RNADE-LW. The second algorithm is based on gradient ascent, which is denoted as RNADE-GA. It initializes all query and missing variables to a fixed value, iteratively optimizes the assignment through gradient ascent with the log-likelihood as the objective function, and returns the iteration with the highest log-likelihood. Note that the gradient is approximated using numerical methods since a closed form gradient is not available for RNADE. In addition, backtracking line search is used to determine an appropriate step size in each iteration. For the tuning of hyperparameters, we follow the same strategy as Uria et al. [2013]: we tune the number of components, $\{2, 5, 10, 20\}$; the weight-decay, $\{2.0, 1.0, 0.1, 0.01, 0.001, 0\}$; and the learning rate $\{0.1, 0.05, 0.025, 0.0125\}$. The remaining hyperparameters, where possible, are set using the discussion in Uria et al. [2013], left at the default value determined by existing code, or set to a value that is equivalent to our method if we have the same hyperparameter. For example, both our model and RNADE use the same number of pre-trainings.

Sum-Product Networks (SPNs) Poon and Domingos [2011], Molina et al. [2018] admit tractable MAP inference (under certain constraints). However for MMAP inference, SPNs are not tractable, but MMAP can be approximated by first approximately marginalizing out the missing variables and then applying MAP inference over the marginalized SPN. We also consider another inference method called SPN-S to estimate the MMAP results using the same sampling technique as described above for RNADE. We implement SPNs using existing libraries Molina [2019], Molina et al. [2019], Lorenzo [2020], and tune three different hyperparameters: the instance threshold for creating leaf nodes, $\{50, 100, 200, 300, 500, 1000\}$; the row splitting method, $\{kmeans, gmm, rdc\}$; and the column splitting method,

Table 1: Datasets information

Name	#instance	#feature
airquality	9357	12
cropmapping	50000	117
energy	19735	24
hepmass	150000	21
miniboone	36488	43
onlinenews	39644	32
parkinson	5875	15
sdd	58509	29
superconduct	21263	68
wec-sydney	72000	49

Table 2: One-to-One comparison of win/tie/lose achieved by one model (row) over the other (column).

	CLGBN	MG	RNADE	SPN	OURS
CLGBN	-	00/13/77	65/08/17	10/13/67	00/04/86
MG	77/13/00	-	90/00/00	45/07/38	00/11/79
RNADE	17/08/65	00/00/90	-	01/10/79	00/00/90
SPN	67/13/10	38/07/45	79/10/01	-	15/09/66
OURS	86/04/00	79/11/00	90/00/00	66/09/15	-

$\{rdc, gvs\}$. Note that we restrict the leaf nodes in SPNs to univariate Gaussian distributions and thus we do not need to tune the feature threshold for creating product nodes.

PGNNs (our model) is trained using the Adam optimizer from pyTorch Paszke et al. [2019] with fixed batch size 100, and weight decay $1e^{-6}$ for 150 epochs. For each of the experiments, the number of stages $s = 3$, the reduction factor $r = 2.5$, and the standard deviation threshold $t = \sqrt{5e^{-3}}$ are fixed. As for the parameter k, which is the size of the feature vector generated by the building blocks, we make it 2 times the size of the input, i.e., $k = 2n$. The learning rate update is scheduled using a linear warm up of 5 epochs starting from 1% of the maximum learning rate, followed by cosine decay. Note that the complexity of the PGNN varies depending on the parameter k, and we decided to scale the max learning rate to encourage the convergence of training when k is relatively large [1]. Given a base max learning rate α, we scale it to $\beta = {}^{2\alpha}/_{1+\sqrt{k}}$, and use β as the maximum learning rate in the training of the PGNN. At last, following Uria et al. [2013, 2016], we pre-train 4 times for 15 epochs each and select the model that achieves best loss for subsequent training. For hyperparameter tuning, we considered base maximum learning rate, $\{0.3, 0.16, 0.08, 0.04\}$; maximum number of conditional variables, $\{10\%, 20\%, 30\%, 40\%, 50\%\}$; and the variable selection method from the four heuristics we discussed in Section 2.2.

For each dataset, we randomly selected 200 instances for both validation and test sets and the remaining instances are used as the training split. All models (SPN, RNADE, OURS) are tuned based on the best RMSE achieved on the validation set. In addition, for sample based approximate inference schemes, the sample size is based on the number of query and missing variables during inference, with a minimum number of 200 samples.

[1] Learning rate tuning can also be used to guarantee the convergence of training. However, the range of LR being tuned is usually roughly divided and is not appropriate for every case.

All experiments were conducted on a workstation with a 16-core Intel Xeon Gold 6130 CPU and two Quadro P5000 GPUs. Note that running on a GPU is not required for our model as the PGNNs are typically small compared to modern deep neural networks. For a small dataset like parkinson, CPU only performance is 2x faster than using GPUs. For larger datasets like cropmapping, running our model on GPUs can achieve a 3x to 5x speedup over CPUs. All code will be made publicly available on Github post acceptance.

3.1 PREDICTIVE PERFORMANCE

We conducted a comprehensive experiment using nine different settings of the query and missing variables where the percentage of query and missing variables is chosen from the set $\{10\%, 20\%, 30\%\}$. For each setting, we randomly assign query and missing variables for *each* of the instances in test set and we report the mean RMSE over all query variables averaged across all instances in test set. Table 3 shows the average RMSE of our model along with the other four competitors on the ten different datasets under the above nine settings. Numbers in bold denote the lowest RMSE, we consider two RMSEs within 5% difference as a tie. In addition, Table 2 presents a one-to-one comparison of win/tie/lose for one model (row) over another (column) based on the results in Table 3.

From the above results, we have the following observations. (1) Our model achieves the lowest average RMSE over all datasets, regardless of the setting of the query and missing variables. Further, it achieves the best or a comparable RMSE on seven of the datasets for all nine settings. For the other three datasets (airquality, energy and sdd), we lose 2, 5, and 8 out of 9 settings respectively, to only the SPN model. The poorer performance on the sdd dataset is likely the result of overfitting. (2) SPNs are the second best model in terms of the average RMSE over all datasets. However, their predictive performance is not stable across datasets: they produce very accurate predictions for some datasets (sdd) while the predictions on other datasets can be worse

Table 3: Average prediction RMSE of ten real datasets under nine query and missing settings.

Models	Datasets										Average
	airquality	cropmapping	energy	hepmass	miniboone	onlinenews	parkinson	sdd	superconduct	wec-sydney	
Query: 10% Missing: 10%											
CLGBN	0.3622	0.4040	0.3924	0.8249	0.5810	0.5466	0.3769	0.5058	0.4061	0.7729	0.5173
MG	0.2407	0.2546	0.2810	0.8146	0.4165	0.3875	0.3495	0.4744	0.1529	0.5586	0.3930
RNADE-LW	0.5696	1.1820	0.3416	0.9014	0.5376	1.0871	0.4495	0.6939	0.5369	0.8246	0.7124
RNADE-GA	0.4394	0.9755	0.3799	0.9265	0.4935	0.7387	0.4841	0.6624	0.6968	0.8200	0.6617
SPN	**0.1666**	0.3245	0.1926	0.9019	0.4987	0.4970	0.3957	**0.3469**	0.2076	0.5709	0.4102
SPN-S	**0.1679**	0.3555	0.1815	0.9237	0.5101	0.4842	0.3830	**0.3355**	0.2222	0.6542	0.4218
OURS	**0.1607**	**0.2173**	**0.1523**	**0.7600**	**0.3594**	**0.3325**	**0.2786**	0.4362	**0.0849**	**0.4656**	**0.3247**
Query: 10% Missing: 20%											
CLGBN	0.4146	0.4018	0.4350	0.8013	0.6522	0.6143	0.4476	0.5015	0.4316	0.7996	0.5500
MG	0.2879	0.2556	0.3251	0.7875	0.4598	0.4963	**0.4126**	0.4508	0.1929	0.5961	0.4265
RNADE-LW	0.5764	1.1881	0.3860	0.8457	0.6059	1.0052	0.5453	0.7064	0.6264	0.8277	0.7313
RNADE-GA	0.5342	0.9257	0.4109	0.8810	0.5259	0.7354	0.5556	0.6350	0.7562	0.7967	0.6756
SPN	**0.1585**	0.3399	0.1921	0.8496	0.5479	0.5367	0.4437	**0.3245**	0.2119	0.5622	0.4167
SPN-S	0.2147	0.3786	0.1898	0.9061	0.6076	0.5582	0.4669	**0.3185**	0.2378	0.6462	0.4524
OURS	0.1747	**0.2153**	**0.1775**	**0.7078**	**0.3631**	**0.4364**	**0.3938**	0.3858	**0.1010**	**0.4824**	**0.3438**
Query: 10% Missing: 30%											
CLGBN	0.4369	0.4336	0.4748	0.8136	0.6623	0.6231	0.4219	0.5278	0.4328	0.7828	0.5610
MG	0.3677	0.2890	0.3581	0.7840	0.4744	0.5286	0.3842	0.4977	0.1975	0.6175	0.4499
RNADE-LW	0.6512	1.2668	0.3918	0.9214	0.6445	1.0219	0.5557	0.7558	0.5887	0.8450	0.7643
RNADE-GA	0.6167	0.9624	0.4065	0.9120	0.5462	0.7793	0.5168	0.6654	0.7125	0.8047	0.6923
SPN	0.2463	0.3351	**0.1960**	0.8598	0.5547	0.5367	0.4265	0.3996	0.1944	0.5644	0.4313
SPN-S	0.3014	0.3940	0.2088	0.9033	0.6316	0.6183	0.4137	**0.3725**	0.2341	0.6574	0.4735
OURS	**0.2345**	**0.2381**	**0.2037**	**0.7373**	**0.3913**	**0.4925**	**0.3420**	0.4514	**0.1166**	**0.5042**	**0.3712**
Query: 20% Missing: 10%											
CLGBN	0.4346	0.4461	0.4804	0.8714	0.6463	0.6153	0.4545	0.5559	0.4339	0.8007	0.5739
MG	0.3192	0.2975	0.3723	0.8573	0.4655	0.4895	0.4237	0.5266	0.1940	0.6087	0.4554
RNADE-LW	0.7073	1.2279	0.4641	1.0131	0.6084	1.1404	0.6089	0.7821	0.5810	0.8463	0.7979
RNADE-GA	0.7443	0.9861	0.4531	1.0091	0.5553	0.7482	0.5779	0.7113	0.7262	0.8314	0.7343
SPN	**0.1942**	0.3472	**0.2047**	0.9412	0.5296	0.5848	0.4520	0.4072	0.2209	0.5676	0.4449
SPN-S	0.2849	0.3950	**0.2079**	0.9823	0.5762	0.5680	0.4581	**0.3858**	0.2558	0.6533	0.4767
OURS	0.2082	**0.2430**	**0.2129**	**0.7991**	**0.4027**	**0.4388**	**0.3854**	0.4780	**0.1152**	**0.4851**	**0.3768**
Query: 20% Missing: 20%											
CLGBN	0.4728	0.4445	0.5496	0.8879	0.7033	0.6911	0.4688	0.5473	0.4615	0.8194	0.6046
MG	0.3400	0.2877	0.4359	**0.8744**	0.5023	0.5878	**0.4219**	0.5089	0.2091	0.6526	0.4821
RNADE-LW	0.7446	1.2752	0.5010	1.0413	0.6759	1.1328	0.5995	0.7711	0.6181	0.8759	0.8235
RNADE-GA	0.8407	0.9658	0.4979	1.0174	0.5914	0.8278	0.5772	0.6963	0.7480	0.8599	0.7622
SPN	0.2864	0.3257	**0.2082**	0.9682	0.5705	0.6587	0.4617	**0.4224**	0.2118	0.6091	0.4723
SPN-S	0.3152	0.3902	0.2283	0.9850	0.6216	0.7176	0.4609	**0.4124**	0.2410	0.7068	0.5079
OURS	**0.2624**	**0.2406**	0.2724	**0.8539**	**0.4381**	**0.5398**	**0.4145**	0.4703	**0.1247**	**0.5295**	**0.4146**
Query: 20% Missing: 30%											
CLGBN	0.4759	0.4418	0.5497	0.8446	0.6871	0.6970	0.4558	0.5673	0.4778	0.8259	0.6023
MG	0.3879	0.2850	0.4355	**0.8285**	0.5154	**0.6271**	0.4111	0.5405	0.2425	0.6754	0.4949
RNADE-LW	0.7501	1.3056	0.4952	0.9787	0.6596	1.1117	0.6167	0.7803	0.6306	0.8682	0.8197
RNADE-GA	0.9083	0.9389	0.5145	1.0012	0.5888	0.8160	0.6007	0.6870	0.7736	0.9285	0.7757
SPN	**0.2935**	0.3330	**0.2119**	0.8901	0.5670	0.6957	0.4536	**0.4698**	0.2175	0.6053	0.4737
SPN-S	0.3602	0.3908	0.2304	0.9871	0.6338	0.7407	0.4423	**0.4504**	0.2611	0.7096	0.5206
OURS	**0.3041**	**0.2401**	0.3004	**0.7991**	**0.4533**	**0.6344**	**0.3753**	0.4886	**0.1435**	**0.5390**	**0.4278**
Query: 30% Missing: 10%											
CLGBN	0.4740	0.4709	0.5253	0.8958	0.6796	0.6332	0.5025	0.5865	0.4580	0.8476	0.6073
MG	0.3607	0.3140	0.4178	**0.8805**	0.4871	0.5288	0.4584	0.5443	0.2151	0.6640	0.4871
RNADE-LW	0.9037	1.2967	0.4909	1.0222	0.6702	1.1271	0.6770	0.8089	0.6029	0.8865	0.8486
RNADE-GA	0.9686	0.9791	0.5074	1.0061	0.5749	0.7843	0.6260	0.7427	0.7538	0.8538	0.7797
SPN	**0.2837**	0.3420	**0.2298**	0.9710	0.5389	0.6137	0.4878	**0.4946**	0.2160	0.6196	0.4797
SPN-S	0.4138	0.4166	**0.2371**	1.0145	0.6086	0.6606	0.4983	**0.4746**	0.2502	0.7184	0.5293
OURS	**0.2874**	**0.2643**	0.2700	**0.8597**	**0.4090**	**0.5002**	**0.4323**	0.5236	**0.1210**	**0.5430**	**0.4210**
Query: 30% Missing: 20%											
CLGBN	0.5203	0.4713	0.5375	0.9041	0.7035	0.6931	0.5317	0.5898	0.5025	0.8580	0.6312
MG	0.4232	0.3040	0.4234	**0.8978**	0.5293	**0.6120**	0.4942	0.5623	0.2564	0.7001	0.5203
RNADE-LW	0.9098	1.3372	0.5079	1.0684	0.6933	1.1058	0.6826	0.8114	0.6452	0.8763	0.8638
RNADE-GA	1.1027	0.9676	0.5265	1.0267	0.6063	0.8151	0.6532	0.7242	0.7948	0.9439	0.8161
SPN	**0.3332**	0.3372	**0.2415**	0.9733	0.5579	0.6852	0.5065	**0.5110**	0.2173	0.6250	0.4988
SPN-S	0.4599	0.4075	**0.2385**	1.0351	0.6299	0.7228	0.5114	**0.5027**	0.2619	0.7384	0.5508
OURS	**0.3292**	**0.2625**	0.3103	**0.8936**	**0.4675**	**0.5942**	**0.4570**	**0.5184**	**0.1513**	**0.5936**	**0.4578**
Query: 30% Missing: 30%											
CLGBN	0.5736	0.5044	0.5504	0.8963	0.7415	0.7505	0.5095	0.5996	0.5261	0.8894	0.6541
MG	0.4928	0.3362	0.4390	**0.8857**	0.5758	**0.7050**	0.4864	0.5776	0.2893	0.7503	0.5538
RNADE-LW	0.9009	1.3642	0.5675	1.0708	0.7618	1.0816	0.6531	0.8351	0.6915	0.9264	0.8853
RNADE-GA	1.2655	0.9623	0.5697	1.0226	0.6621	0.8354	0.6385	0.7242	0.8214	1.0205	0.8522
SPN	**0.4191**	0.3590	**0.2384**	0.9928	0.6087	0.8815	0.5016	**0.5105**	0.2282	**0.6479**	0.5388
SPN-S	0.6605	0.4350	0.2589	1.0471	0.6829	0.9594	0.5241	**0.5163**	0.2757	0.7630	0.6123
OURS	**0.4029**	**0.2832**	0.3559	**0.9175**	**0.5020**	**0.7095**	**0.4612**	0.5363	**0.1840**	**0.6239**	**0.4976**

than that of the multivariate Gaussian baseline (cropmapping), which it fails to outperform in roughly half of the cases. Our model, on the the hand, is only outperformed by SPNs and only in roughly 17% of the cases. (3) RNADE, somewhat surprisingly, seems to be the worst model among all models tested. For example, it fails to beat the baseline CLGBN model in over 70% of the cases. We believe there are two likely reasons for this. First, RNADE is a complicated model and is intractable everywhere (even the gradient is numerically approximated). This might results in worse performance of the approximate MMAP procedures used in this case. Further, these MMAP techniques are quite general, i.e., they are not specifically hand-tailored to the RNADE model. Second, the number of hyperparameters that need to be tuned in RNADE is extensive, Uria et al. [2013] manually tuned RNADE on relatively large dataset. In our case, we only tuned a fraction of the possible hyperparameters as otherwise the search would have been computationally expensive. As such, with additional tuning, the performance of RNADE would likely be improved (though the same could be said of more extensive hyperparameter tuning for SPNs and our model as well). (4) MG produces acceptable or even very good results in many of the cases. This is not surprising as MGs admit exact MMAP inference, so no approximations are necessary.

We also evaluated the log-likelihood of each model on the respective test sets (see Appendix A of the supplementary material). In summary, SPNs achieve the highest average log-likelihood on six out of the ten datasets while our model achieves the highest log-likelihood on the remaining four. However, given the observations in Table 3, it does not appear that the highest test set log-likelihood translates into the best predictive performance (at least not on the MMAP task). In addition, if the hyperparameter tuning was done in order to maximize the log-likelihood score on the validation data instead of minimizing RMSE, then the predictive performance of all of the non-baseline models would likely decrease significantly.

4 CONCLUSION AND FUTURE WORK

We described a flexible family of tractable models for marginal MAP prediction tasks in continuous domains. Experimentally, we verified that our approach outperforms existing approaches such as SPNs and RNADE on a variety of real-world prediction tasks. In particular, we found that (1) although SPNs typically produced models with higher test set log-likelihoods, our model typically resulted in the best performance on the prediction task and (2) the number of hyperparameters required to fit RNADE in practice was computationally prohibitive and/or much more careful hand-tuning may be required, which likely limits it practical utility.

We focused primarily on relatively low dimensional datasets in this work, and more detailed studies need to be conducted to assess performance differences between these models on high dimensional datasets. We are also interested in exploring applications of this approach to temporal data or more generally in situations where there may be additional graphical structure that can be exploited. In addition, many more neural network structures are possible for parameter selection, and in different applications, it may be of interest to explore different function characterizations of both general distribution $p(\boldsymbol{X})$ and conditional distribution $p(\boldsymbol{Y}|\boldsymbol{X})$. We leave these extensions for future work.

References

CKCN Chow and Cong Liu. Approximating discrete probability distributions with dependence trees. *IEEE transactions on Information Theory*, 14(3):462–467, 1968.

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

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

Marco Grzegorczyk. An introduction to gaussian bayesian networks. In *Systems Biology in Drug Discovery and Development*, pages 121–147. Springer, 2010.

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)*, pages 1–10, 2014.

Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

Qiang Liu and Alexander Ihler. Variational algorithms for marginal map. *The Journal of Machine Learning Research*, 14(1):3165–3200, 2013.

Loconte Lorenzo. Sum-product networks and normalizing flows for tractable density estimation [mit license]. `https://github.com/loreloc/spnflow`, 2020.

Alejandro Molina. Sum product flow: An easy and extensible library for sum-product networks [apache license v2.0]. `https://github.com/SPFlow/SPFlow`, 2019.

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

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, 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier, 2014.

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 and Vibhav Gogate. Learning ensembles of cutset networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

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.

Tahrima Rahman, Shasha Jin, and Vibhav Gogate. Cutset bayesian networks: A new representation for learning rao-blackwellised graphical models. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, 2019.

Chiradeep Roy, Tahrima Rahman, Hailiang Dong, Nicholas Ruozzi, and Vibhav Gogate. Dynamic cutset networks. In *International Conference on Artificial Intelligence and Statistics*, pages 3106–3114. PMLR, 2021.

Benigno Uria, Iain Murray, and Hugo Larochelle. Rnade: the real-valued neural autoregressive density-estimator. In *Proceedings of the 26th International Conference on Neural Information Processing Systems (NeurIPS)*, pages 2175–2183, 2013.

Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, and Hugo Larochelle. Neural autoregressive distribution estimation. *The Journal of Machine Learning Research*, 17(1):7184–7220, 2016.

A TEST SET LOG-LIKELIHOOD

Table 4 shows the train and test loglikelihood for all of the models compared in the paper. In addition to the average loglikelihood, we also show the 25%, 50% (median) and 75% percentile of train/test loglikelihood over all train/test instances.

Table 4: Train (marked in grey)/test Loglikelihood, and best test LL is marked in bold.

Models	Datasets									
	airquality	cropmapping	energy	hepmass	miniboone	onlinenews	parkinson	sdd	superconduct	wec sydney
25 percentile										
CLGBN	-11.383	-72.587	-22.489	-29.888	-50.113	-37.358	-10.935	-31.542	-55.350	-65.904
	-12.460	-70.940	-21.627	-30.428	-51.549	-37.241	-11.026	-31.109	-60.863	-64.181
MG	-8.613	-21.946	-16.294	-29.642	-38.912	-26.496	-9.778	-28.944	-11.033	-58.507
	-9.012	-20.699	-15.314	-30.254	-40.705	-26.453	-9.727	-28.511	-15.001	**-56.181**
RNADE	8.110	-151.363	-12.605	-27.548	-32.791	-24.116	-8.871	-17.871	-46.538	-65.660
	7.738	-148.388	-11.969	**-28.100**	-32.791	-25.218	-8.913	-16.645	-43.908	-64.999
SPN	17.239	15.530	-0.160	-28.887	-35.789	37.651	-7.683	12.295	1.841	-54.269
	17.314	10.884	-2.745	-29.640	-35.580	**29.474**	-8.694	**12.100**	-3.045	**-54.230**
OURS	-1.331	63.827	0.342	-26.729	-24.271	-9.646	-4.999	-12.213	26.403	-58.135
	-1.524	**64.601**	**-0.218**	**-27.119**	**-28.240**	-10.994	**-6.011**	-11.579	**25.645**	-57.579
Median										
CLGBN	-8.741	-55.268	-18.906	-27.457	-42.106	-32.656	-8.800	-27.351	-38.429	-60.996
	-8.875	-54.845	-18.505	-27.861	-42.877	-32.500	-8.791	-26.955	-40.621	-61.379
MG	-5.871	-13.910	-12.642	-27.195	-30.665	-21.348	-7.634	-24.708	7.662	-47.397
	-5.949	-14.178	-12.554	-27.726	-31.127	-21.057	-7.657	-24.075	5.400	-48.223
RNADE	17.277	269.283	5.637	-23.267	-13.650	5.278	-0.151	-19.875	125.006	-30.385
	17.416	**266.802**	4.548	**-23.774**	**-16.033**	4.874	**-0.269**	-20.079	**113.721**	**-33.907**
SPN	20.433	53.421	9.641	-26.342	-27.436	63.851	-3.355	17.940	86.134	-37.666
	20.403	50.126	**8.028**	-27.249	-28.500	**61.093**	-4.259	**17.043**	84.018	-38.811
OURS	0.876	82.298	5.487	-23.972	-15.288	-0.939	-1.546	-7.190	62.223	-39.593
	1.057	82.919	4.791	**-24.701**	-17.369	-2.318	-2.041	-6.892	50.711	-41.495
75 percentile										
CLGBN	-7.093	-43.213	-16.626	-25.571	-37.537	-29.948	-7.390	-25.052	-26.318	-57.738
	-7.359	-43.092	-16.141	-25.999	-38.651	-29.691	-7.449	-25.260	-27.085	-58.090
MG	-4.444	-8.425	-10.331	-25.312	-26.020	-18.561	-6.269	-22.397	19.288	-38.185
	-4.583	-8.217	-10.285	-25.731	-27.158	-18.082	-6.280	-22.439	19.122	-38.550
RNADE	13.070	-105.694	-4.115	-22.049	-16.695	-9.410	-1.611	-7.520	48.924	-15.120
	13.127	-104.361	-4.336	**-22.232**	-17.689	-9.010	-1.869	-7.167	47.275	-15.219
SPN	25.283	92.417	20.264	-24.278	-19.996	90.897	0.045	24.068	255.095	1.171
	27.054	**92.352**	**18.648**	-24.539	-21.470	**87.679**	-0.722	**23.833**	**290.550**	**6.218**
OURS	2.812	94.529	10.769	-21.590	-7.345	5.259	1.366	-3.372	91.351	2.496
	3.021	**95.279**	9.513	**-22.133**	**-8.480**	5.284	**1.196**	-2.729	88.547	**5.954**
Average										
CLGBN	-10.156	-63.346	-20.252	-28.178	-47.836	-36.688	-11.614	-30.517	-46.466	-62.348
	-10.596	-62.214	-19.564	-28.584	-48.657	-37.501	-11.713	-29.021	-49.418	-62.054
MG	-7.454	-20.564	-14.210	-27.925	-37.167	-27.551	-10.650	-31.893	-2.042	-49.815
	-7.639	-17.089	-13.627	-28.348	-37.182	-27.195	-10.566	-26.362	-4.764	-48.627
RNADE	10.484	-131.016	-8.591	-25.089	-25.942	-17.803	-5.938	-13.537	2.619	-39.457
	10.242	-128.895	-8.370	**-25.583**	-27.512	-18.257	-6.371	-12.555	2.560	-38.361
SPN	27.953	51.723	11.080	-27.003	-28.953	65.680	-4.634	18.856	180.669	-24.174
	29.579	46.459	**8.657**	-27.562	-30.970	**59.444**	-5.543	**18.526**	**184.080**	**-23.590**
OURS	0.746	73.029	5.312	-24.498	-17.023	-3.078	-2.634	-10.124	55.991	-26.813
	0.763	**74.106**	4.455	**-25.466**	**-19.347**	-4.174	**-4.169**	-7.427	52.627	-26.559