NeuroBE: Escalating NN Approximations of Bucket Elimination

Abstract

A major limiting factor in graphical model inference is the complexity of computing the partition function. Exact message-passing algorithms such as Bucket Elimination (BE) require exponentially high levels of memory to compute the partition function, therefore approximations are necessary. In this paper, we build upon a recently introduced methodology called *Deep Bucket Elimination (DBE)* that uses classical Neural Networks (NNs) to approximate messages generated by BE when buckets have large memory requirements. The main feature of our new scheme called $NeuroBE$ is that it customizes the architecture and learning of the NNs to the message size and its distribution. We also explore a new loss function for training taking into account the estimated message cost distribution. Our experiments demonstrate significant improvements in accuracy and time compared with DBE.

1 INTRODUCTION

Two of the critical goals of probabilistic modeling are the compact representation of probability distributions and the efficient computation of their marginals and modes. Probabilistic graphical models, such as Markov networks (Pearl, 1988; Darwiche, 2009; Dechter, 2013) provide a framework to represent distributions compactly as normalized products or factors : $P(X) = \frac{1}{Z}\prod_\alpha f_\alpha(X_\alpha)$, where X is a set of variables, each potential f_α is a function over a subset X_α of the variables (its scope) and $Z = \sum_X \prod_\alpha f_\alpha(X_\alpha)$ is the *partition function*. Computing the partition function is still exponential in the induced width of the model's graph even for distributions that admit a compact representation.

The partition function Z is defined by two types of operations: sums and products. It can be evaluated efficiently if $\sum_X \prod_\alpha f_\alpha(X_\alpha)$ can be reorganized using the distributive law along a variable ordering (Dechter, 2003). This organization can be described using buckets as data structures, one for each variable in the ordering. When a bucket is processed, its associated variable is removed by creating a bucket output function, also called a *message*, that is passed to a subsequent bucket. The complexity of computing this function is exponential in its number of arguments, called scope or the bucket's width. Overall, Bucket Elimination (BE) (Dechter, 1999b) is time and memory exponential in the induced-width of the model's graph along the ordering.

Providing good approximations to *BE* is important not only because it generates an answer to a query, but primarily because it compiles a structure and a set of messages that can be used to answer multiple queries (e.g., the probability of evidence for various evidence variables Darwiche (2009)). Also because, the messages can be used as building blocks for generating heuristics for search to further improve performance. We will therefore consider and evaluate $NeuroBE$ within the class of *approximate BE* schemes.

Schemes that approximate *BE* include the (weighted) mini-bucket scheme (Dechter and Rish, 2003; Liu and Ihler, 2012) and generalized belief propagation schemes (Yedidia et al., 2000; Mateescu et al., 2010). Recently introduced, *Deep Bucket Elimination (DBE)* (Razeghi et al., 2021) approximates each bucket function with a neural network (NN). While this approach is inherently time consuming requiring the independent training of many NNs to compute the partition function of a single problem, it has yielded more accurate approximations on several benchmarks. It is important to note that unlike weighted bucket-elimination and belief propagation schemes, DBE can improve with time even with bounded memory, yielding an anytime framework for reasoning. Still, DBE's original design can be improved significantly which is the focus of this paper.

Contributions. We present $NeuroBE$, a re-design of DBE, that addresses its *one size fits all* policy by customizing the NN construction and training sample size to each

Submitted to the 38th Conference on Uncertainty in Artificial Intelligence (UAI 2022). **To be used for reviewing only**.

bucket separately, in proportion to its message size. In particular, we introduce a new loss function that is sensitive to bucket's message distribution. We also provide an analysis relating the local errors to a bound on the global error. In an extensive empirical evaluation we show that $NeuroBE$ outperforms DBE across all benchmarks using far less training samples, yielding higher accuracy with less time.

The paper is organized as follows. We first provide a background to *BE* and *DBE*; then we present $NeuroBE$; followed by error analysis; lastly, we demonstrate the efficiency of $NeuroBE$ empirically.

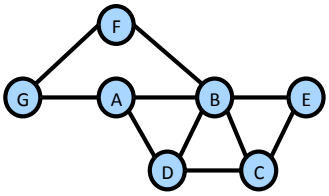
Related work. As noted, approximating and bounding *Bucket Elimination* has been carried out extensively over the years for all probabilistic queries. Well known is the *Mini-Bucket Elimination* scheme (Dechter and Rish, 2003) and its variants, such as *Weighted Mini-Bucket Elimination (WMBE)*, augmented with message-passing cost-shifting (Liu and Ihler, 2011).

Neural network approximation to *BE* was introduced in Razeghi et al. (2021). The idea is closest in spirit to the Neuro-Dynamic Programming scheme as outlined in Bertsekas and Tsitsiklis (1996) where the cost-to-go functions (similar to messages) generated by dynamic programming can be approximated by NNs. This technique is also highly related to Deep Reinforcement Learning (DRL) (Mnih et al., 2015) where, in the absence of a model, the value function is approximated by neural networks learned from temporal trajectories.

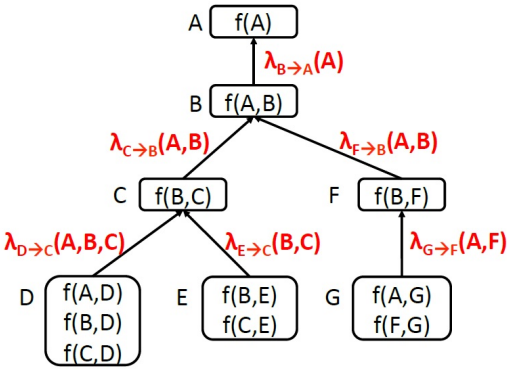
Recently, *Graph Neural Networks (GNNs)* (Scarselli et al., 2009) are used to learn *messages* following the message-passing reasoning methods in graphical models (Abboud et al., 2020; Yoon et al., 2018; Heess et al., 2013). However, Yoon et al. (2018); Heess et al. (2013) is restricted to small instances (i.e., ∼40 variables) and Abboud et al. (2020) tackles problems with a known polynomial-time approximation. GNN based methods derive a supervised end-to-end learning algorithm which generalize across different problem instances. In contrast, we consider a different class of algorithms, where we confine learning to within a problem instance *only*.

2 BACKGROUND

A graphical model can be defined by a 3-tuple $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{F})$, where $\mathbf{X} = \{X_i : i \in V, V = \{1, ..., n\}\}$ is a set of n variables indexed by V and $\mathbf{D} = \{D_i : i \in V\}$ is the set of finite domains for each X_i (i.e. each X_i can only assume values in D_i, and each D_i is finite). Each function $f_\alpha \in \mathbf{F}$ is defined over a subset of the variables called its scope, X_α, where $\alpha \subseteq V$ are the indices of variables in its scope and D_α denotes the Cartesian product of their domains, so that $f_\alpha : D_\alpha \rightarrow R\geq 0$.



(a) A primal Graph



(b) Bucket Elimination example

Figure 1: (a) A primal graph of a GM with 7 variables. (b) Illustration of *BE* with an ordering A B C E D F G.

The **primal graph** of a graphical model associates each variable with a node. An edge between node i and node j is created if and only if there is a function containing X_i and X_j in its scope. Figure 1a shows a primal graph of a graphical model with variables indexed from A to G with functions over pairs of variables that are connected by an edge. Graphical models can be used to represent a global function, often a probability distribution, defined by $Pr(X) \propto \prod_\alpha f_\alpha(X_\alpha)$. An important task is to compute the normalizing constant, also known as the partition function $Z = \sum_X \prod_\alpha f_\alpha(X_\alpha)$.

2.1 BUCKET ELIMINATION

Bucket Elimination (BE) (Dechter, 1999a) is a universal exact algorithm for probabilistic inference. It is a variable elimination algorithm that can answer a wide-range of queries, including the partition function ranging from constraint satisfaction, to pure combinatorial optimization (e.g., Most Probable Explanation (MPE/MAP)), and weighted counting (Partition Function, Probability of Evidence).

Given a variable ordering d, BE (presented in Algorithm 1, omitting steps 9-12) creates a *bucket tree* where each node is a bucket representing a variable in the ordering d. Figure 1b shows a bucket tree for the primal graph in Figure 1a along an ordering. Each bucket in this tree contains a set of the model's functions depending on the given order of processing. For example, Bucket G in Figure 1b has functions $\{f(A, G), f(F, G)\}$, an exhaustive set of model's

functions with variable G in its scope. There is an arc from a bucket, say B_c, to a parent bucket, B_p, if X_p is the latest variable in bucket B_c's message scope along the ordering (constants are placed in B_1). In the same example, there is an arc from Bucket G to Bucket F.

BE then, performs inference along the bucket tree as a 1-iteration message-passing algorithm (bottom-up). It processes each bucket from leaves to the root passing messages from child (c) to parent (p). For a child variable X_c, it considers all the functions in bucket B_c. This includes the original functions in the graphical model as well as the messages received by processing previous variables. It then marginalizes X_c out from the product of functions in B_c generating a new, so called, *bucket function* or message, denoted $\lambda_{c\to p}$, or λ_c for short:

$$\lambda_c = \sum_{X_c} \prod_{f_\alpha \in B_c} f_\alpha \tag{1}$$

The λ_c function is placed in B_p, the bucket of X_p. Once all the variables are processed, *BE* outputs all the messages and the exact value of Z by taking the product of all the constant present in the bucket of the first variable. We illustrate *BE* message flow on our example problem in Figure 1b.

Complexity. Both the time and space complexity of *BE* are exponential in the **induced width**, which is the size of the largest number of variables in the scope of any message over all buckets (Dechter, 2013). Clearly, *BE* becomes impractical if the induced width is large.

2.2 DEEP BUCKET ELIMINATION

Given a variable ordering d, *Deep Bucket Elimination (DBE)* (Razeghi et al., 2021) approximates each message generated in the bucket tree by training a NN when the scope of any bucket message (S) is high ($> i$-bound). For example, in figure 1b, if we use an i-bound $= 2$, instead of sending an exact function from the bucket of D to the bucket of C, $\lambda_{D\to C}(A, B, C)$, DBE sends a NN approximation $\mu_{\theta,D\to C}(A, B, C)$ with parameters θ, as we describe next.

Let B be a bucket with width $w > i$-bound and output message $\lambda(S)$ with scope S. DBE then, constructs a fully-connected feed-forward NN having w nodes in the input layer. This is followed by L hidden layers with a constant h hidden nodes per layer with $ReLU$ activation function. Finally, the output layer contains one node with a real-valued output. Subsequently, DBE generates a training set $\{(s_n, \lambda(s_n))\}$ of size N, where s_n denotes a configuration over S sampled uniformly at random and $\lambda(s_n)$ is the message value defined in Eq. 1. The NN parameters, θ, are then trained to minimize the mean square error loss :

$$L(\theta) = \frac{1}{N} \sum_{n=1}^{N} (\lambda(s_n) - \mu_\theta(s_n))^2$$

where s_n is the nth sample in the training set and $\mu_\theta(s_n)$ is the NN output. Once training is complete, DBE passes the trained NN, μ_θ to its parent bucket. Typically, DBE needs to approximate many buckets in order to compute the partition function Z.

Even though DBE has yielded more accurate approximations when competing with the weighted bucket-elimination scheme on several benchmarks, each message approximation procedure requires a large training sample size, increasing DBE's total time and memory requirements substantially. Therefore, we re-design each message approximation procedure, as elaborated in the following section.

3 NEUROBE

We rename DBE to $NeuroBE$ since we use mostly shallow neural networks (2 layers). Algorithm 1 presents *NeuroBE*. $NeuroBE$ first creates a bucket tree along a given ordering (line 2). While processing each bucket, if it's width $\leq i-$bound, then the message, $\mu^*_{c\to p}$, is computed exactly (line 7). Otherwise, the message is approximated using a NN (line 10). Note that in either case, if a bucket contains a NN function, then computing μ^* in Line 7 or in the NN-$train$ method in Algorithm 3 (line 5) requires evaluating the trained NN. Finally, line 14 calculates the partition function using the functions in bucket B_1.

Note that we denote $\mu^*_{c\to p}$ as the exact message computed in a bucket while we reserve the notation $\lambda_{c\to p}$ to the messages computed by exact BE. We do this to distinguish the exact local computation of a message that may be based on inexact functions in the bucket from the globally exact messages λ computed by BE. In the latter, each bucket function is computed exactly where all the functions are exact. Hence, we refer to λ as the *global exact message*; μ^*_c as the *local exact message* and by $\mu_{\theta,c\to p}$ the NN approximation of the *local* exact message, $\mu^*_{c\to p}$.

The difference between $NeuroBE$ and DBE is solely in the individual message approximation scheme, NN-$train$. In contrast to DBE, $NeuroBE$ dynamically customizes the NN architecture and its training set size to the bucket message complexity (see Vapnik (1999)).

NN Architecture selection Clearly, the NN size should be dependant on the dimensionality of the message function. In our case, the function's scope size is the induced-width, w. We therefore propose to adjust the NN size by making the number of hidden units, h, a function of w while keeping the number of layers, L, constant. Specifically, we select $h = b \cdot w$, where b is a constant satisfying $b \geq 1$. Figure 2 illustrates an example NN model with an input layer of size w and 2 hidden layers with dimension h, varying linearly with b. Through such a rule, $NeuroBE$ *fits* the NN's architecture to the message size. We next quantify the capacity of such

Algorithm 1 NeuroBE

Input: Graphical model $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{F})$, Ordering $d = X_1, ..., X_n$
Parameters: i-bound i ; #layers L; constants b, η;
Output: the partition function constant and bucket messages

for c in n...1 **do**
  (Initialize buckets) put all unplaced functions mentioning X_c in B_c.
end for
for c in n...1 **do**
  Let X_p be the parent variable of X_c in the bucket-tree
  if $width(B_c) < i$ **then**
    compute $\mu^*_{c\to p} \leftarrow \sum_{X_c} \prod_{f_\alpha \in B_c} f_\alpha$,
  else
    $\mu_{\theta,c\to p} \leftarrow$ NN-train ($\{f_\alpha | f_\alpha \in B_c\}$, L, b, η)
  end if
  Put $\mu^*_{c\to p}$ or $\mu_{\theta,c\to p}$ in B_p
end for
$Z = \sum_{X_0} \prod_{f_\alpha \in B_1} f_\alpha$
return Z and all messages generated

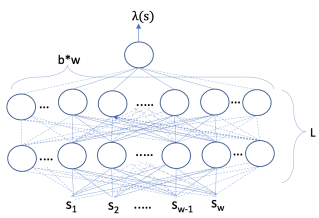


Figure 2: For a bucket of width w, we illustrate a NN architecture with L layers and $\#bw$ hidden-units with $b \geq 1$.

NNs, and use it to determine its train sample sizes.

NN complexity. *Pseudo-dimension* (Pollard, 1984; Anthony and Bartlett, 2002) is often used to measure the expressive power of a set of functions that can be learned by any statistical regression algorithm. Bounds to the pseudo-dimension of NNs used in our work with ReLU activation function is provided by Bartlett et al. (2019). We use the lower bound as a proxy to estimate the pseudo-dimension (ρ) of the NN (μ_θ), with L layers, and $b \cdot w$ hidden units, approximating μ^* having width w, yielding (see appendix for derivation):

$$\rho(w) \propto (L * b * w)^2 log[(b * w)] \tag{2}$$

The above equation correlates the complexity of the candidate NN with the width of the message μ^* it approximates.

Sample Complexity As suggested in Vapnik (1999), we choose a sample size N that is proportional to the pseudo-dimension (Eq. 2) of each NN as a function of its # of

Algorithm 2 generate-samples(F, N, X)

Input: F, a set of functions over scope $S \cup \{X\}$, where X, a variable to be eliminated. μ^* denotes the output message over S, N, an integer,
Output: D, a set of N samples $\{(s, \mu^*_n(s))\}$

initialize D = { },
for i=1..N **do**
  $s \leftarrow$ sample uniformly from domain(S)
  $\mu^*(s) \leftarrow \sum_x \prod_{f \in F} f(s, x)$ {Eq. 4}
  $\mu^*_{nor}(s) \leftarrow$ Normalize $\mu^*(s)$ {Eq. 5}
  Add (s, $\mu^*_{nor}(s)$) to D
end for
return D

arguments w:

$$N(w) = \eta * (L * b * w)^2 log(b * w) \tag{3}$$

where η is a constant. The sample size $N(w)$ often exceeds memory limits for higher width buckets with even the simplest NN architecture ($L = 1, b = 1$). So, we use η as a control over the sample size and threshold sample size per bucket to 10^6. In general, for high induced-width problems, we keep η small and train NNs with small sample sizes. However, for problems with small induced-width, we let η take high values.

Sample Generation Given a generic bucket B of a variable X and a local bucket function $\mu^*(S)$, where S is the scope of the bucket's output function, we create a training set D_T (see Algorithm 2), by generating N (from Eq. 3) number of training examples $(s, \mu^*_{nor}(s))$ by sampling configurations $\{S = s\}$ uniformly at random from the domain of scope S. We compute the message value for each configuration s as

$$\mu^*(s) = \sum_{x \in domain(X)} \prod_{f \in B} f(s, x) \tag{4}$$

Each generated configuration s then, is paired with its normalized message value $\mu^*_{nor}(s) \in [0, 1]$ as,

$$\mu^*_{nor}(s) = \frac{\mu^*(s) - \mu^*_{min}(D_T)}{\mu^*_{max}(D_T) - \mu^*_{min}(D_T)} \tag{5}$$

where μ^*_{min} and μ^*_{max} are minimum and maximum μ^* values in D_T. Note that we also normalize the input sample configurations across benchmarks with domain values > 2 to be in $[-1, 1]$ to accelerate training, as suggested in Cun et al. (1991). Some benchmarks in our experiments have very large message values (e.g. e^{51}). In such cases, we use a log transform i.e. $log\mu^*$, $log\mu^*_{min}(D_T)$ and $log\mu^*_{max}(D_T)$ instead of their corresponding usual μ^* values to compute

the target NN values μ^*_{nor} in Eq. 5. (For simplicity we will ignore the normalization subscript on the output function μ^* and will explicitly denote it when relevant.)

Loss Function DBE used the mean square error loss function for training each NN. However, in the context of the partition function we want to weigh errors associated with high cost tuples higher than low cost tuples. Namely, squared error on message values that are large seem to be potentially much costlier than smaller message values because of their significant contribution to the sum-product operation in Eq. 1. Hence, we propose an importance mean square error (abbreviated as I.m.s.e) as a loss function where the weights are dependent on the message values.

We define the importance weight of a sample as its relative weight in the function distribution in the given dataset D. Namely, given a dataset D, the importance weight $W(s)$ is normalized over the μ^* values:

$$W(s) = \frac{\mu^*(s)}{\sum_{s \in D} \mu^*(s)} \tag{6}$$

Definition 1 (Importance mean square error loss (I.m.s.e))**.** Let μ_θ be the learned NN function for approximating function μ^*. Then, the I.m.s.e loss function for a given minibatch, $D \in D_T$ of size $\#D$ is defined by

$$L_D(\mu^*, \mu_\theta) = \frac{1}{\#D} \sum_{s \in D} (\mu^*(s) - \mu_\theta(s))^2 * W(s) \tag{7}$$

log transformation. For problem instances whose functions costs and message costs are very large, we apply log transformation everywhere, replacing μ^* by $\log \mu^*$ in the above equations. Here, the weight function becomes:

$$W^{log}_D(s) = \frac{log\mu^*(s) - log\mu^*_{min}(D)}{\sum_{s \in D}(log\mu^*(s) - log\mu^*_{min}(D))} \tag{8}$$

Algorithm 3 describes the procedure, $NN\text{-}Train$. Its input parameters are L, b, η where L is the number of layers, b is a constant to determine the number of hidden-units, $b \cdot w$ (line 1), η is another constant to determine the training sample size N (line 2, Eq. 3). A major step occurs next where the algorithm generates a Data set of samples (line 3, see also Algorithm 2) and splits it to create the training set of size N, validation set of size $N/4$ and testing set of size $50k$ (line 4). If the bucket B contains a trained NN, then this step requires evaluating that NN. Lines 8-12 then performs batch training for updating the NN parameters (θ) using the I.m.s.e loss function (Eq. 7), and the Adam optimizer (Kingma and Ba, 2014) (line 14) with a learning rate of 0.001 and a batch-size (N_B) of 256 across all benchmarks. At the end of each epoch, the current model is evaluated on a holdout validation set (line 14). We evaluate the early-stopping criteria (line 15), which is assigned $True$ when either the maximum limit $\#epochs$ is reached or the validation error increases for two consecutive epochs. Once training is complete, we compute the maximum log relative error between the target and NN approximated messages over a test set (lines 18-19). In the next section, we use this to analyse error propagation in $NeuroBE$. The $NN\text{-}train$ procedure then returns the approximated message μ_θ, along with its estimated error.

Algorithm 3 NN-train(F,X,L,b, η, $\#epochs$)

Input: F: a set of functions over scope $S \cup \{X\}$ where X is to be removed, w scope size.
Parameters: L: # layers in NN, $\#epochs$, η, b: constants
Output: μ_θ: NN message approximation, $\hat{\epsilon}$: an estimated bucket error bound, $\hat{\epsilon}_{avg}$: estimated average bucket error

1: $\#h \leftarrow b * w$
2: $N \leftarrow$ #samples(w, η, L, b) {Eq. 3}
3: Data $\leftarrow$ generate-samples($F, N + N/4 + 50k, X$)
4: $D_{Train},\ D_{Val},\ D_{Test} \leftarrow$ Split(Data)
5: Initialize NN parameters θ, p=1, early-stopping $\leftarrow$ False
6: **while** p $\leq \#epochs$ and $\neg$ early-stopping **do**
7: $D_1, .., D_k \leftarrow$ divide D_{Train} to minibatch
8: **for** $i = 1..k$ **do**
9: Let D_i ={(s,μ^*)}
10: Compute $\{\mu_\theta(s) \mid s \in D_i\}$
11: $loss_{D_i} \leftarrow L_{D_i}(\mu^*, \mu_\theta)$ {Eq. 7}
12: $\theta \leftarrow$ update θ by optimize(Adam, $loss_{D_i}$, θ)
13: **end for**
14: loss$_{D_{val}} \leftarrow L_{D_{val}}(\mu^*, \mu_\theta)$ {For stop condition}
15: early-stopping $\leftarrow$ evaluate early-stoping($loss_{D_{Val}}$)
16: $p \leftarrow p + 1$
17: **end while**
18: Unnormalize { $\mu_\theta(s)$,$\mu^*(s)|s \in D_{Test}$} {Inverse of Eq. 5}
19: $\hat{\epsilon} \leftarrow \max_{s \in D_{Test}}(log\mu^*(s) - log\mu_\theta(s))$
20: **return** μ_θ, $\hat{\epsilon}$

Complexity. The time and space complexity for learning a single message in $NeuroBE$ is linear in the sample size. In contrast to DBE, here the sample size varies with the bucket's width.

4 ERROR ANALYSIS

We next analyse the relationship between the local errors contributed by each approximated message and the global partition function error.

Definition 2 (local and global bucket errors)**.** Let λ be the (global) exact message generated in B by the exact BE algorithm, μ^* be the (local) exact message in B computed by the functions in it and $\mu = NN$-train(μ^*) be its NN

Problem Description					ref Z	WMB	#NB	DBE (#h=100, N=320k)				NeuroBE (#h=3w, N_{min} = 49k)										
i-bound=20												Statistics on resources			Loss : mean square error				Loss : Imp. mean square error			
								statistics on error (5 runs)							statistics on error (5 runs)				statistics on error (5 runs)			
Id	name	k	#v	w		error		avg error	min error	stdev	time	h_{max}	N_{avg}	N_{max}	avg error	min error	std	time (h)	avg error	min error	std	time (h)
1	pedigree13	3	888	33	-31.18	6.4696	127	5.32	2.62	3.06	11	96	218k	706k	9.04	7.86	0.8	7.5	1.11	0.76	0.25	7.5
2	pedigree41	5	885	32	-76.04	4.1497	92	4.27	3.25	0.73	9.9	93	190k	658k	10.2	8.4	2.16	6.2	0.47	0.153	0.21	6.2
3	pedigree51	5	871	35	-77.27	9.7624	120	23.92	9.23	12.7	13	102	259k	809k	11.73	10.37	1.26	10.4	3.51	1.96	0.89	10.4
4	pedigree34	5	922	33	-64.23	7.0762	106	5.91	1.57	5.98	11	96	211k	706k	6.14	4.56	4.14	6.96	0.65	0.23	0.29	6.96
5	pedigree7	4	867	34	-64.82	6.0012	108	11.26	5.18	7.8	11	99	350k	900k	#	#	#	10.7	1.75	1.21	0.7	10.7
7	pedigree19	5	693	28	-59.020	2.5809	43	6.054	5.41	0.92	9.4	71	149k	482k	9.14	8.7	0.6	5	2.61	1.91	0.6	5

(a) pedigree

Problem Description					ref Z	WMB	#NB	DBE (#h=100, N=320k)				NeuroBE (#h=w, Nmin = 19k)										
i-bound=20												Statistics on resources			Loss : mean square error				Loss : Imp. mean square error			
								statistics on error (5 runs)							statistics on error (5 runs)				statistics on error (5 runs)			
Id	name	k	#v	w		error		avg error	min error	stdev	time	h_{max}	N_{avg}	N_{max}	avg error	min error	std	time (h)	avg error	min error	std	time (h)
1	grid4040f10	2	1600	55	5490	215.45	308	97.1	11.8	65.2	12	55	120k	364k	14.57	0.15	8.94	7.69	9.71	2.1	0.91	6.4
2	grid4040f5	2	1600	55	2800	84.92	308	39.9	6.28	35	12	55	120k	364k	8.86	2.05	8.21	7.66	3.7	0.2	3.06	6.3
3	grid4040f2	2	1600	55	1220	25.24	308	7.34	1.2	5.4	12	55	120k	364k	3.15	1.48	1.66	7.48	2.28	1.2	0.91	6.1
4	grid4040f15	2	1600	55	8200	338.2	308	83.46	41.8	34.2	13	55	75k	228k	24.75	6.62	18.1	5.7	17.87	7.45	9.16	5.7
5	grid4040f10w	2	1600	114	5637	297.7	376	100.5	6.4	82.2	21	114	150k	670k	67.76	32.7	26	11	54.18	25.01	21.1	11.1
6	grid4040f5w	2	1600	114	2819	136.99	376	78.2	72.6	5.6	21	114	150k	670k	5.37	1.65	4.77	11.6	9.62	4.62	5.27	12.1
7	grid4040f2w	2	1600	114	1231	32	376	15.12	0.92	20.5	18	114	150k	670k	10.15	8.81	1.44	11.8	5.56	2.92	2.11	10

(b) Grid-hard

Problem Description					ref Z	WMB	#NB	DBE (#h=100, N=320k)				NeuroBE (#h=w, Nmin = 20k)										
i-bound=10												Statistics on resources			Loss : mean square error				Loss: Imp. mean square error			
								statistics on error (5 runs)							statistics on error (5 runs)				statistics on error (5 runs)			
Id	name	k	#v	w		error		avg error	min error	stdev	time	h_{max}	N_{avg}	N_{max}	avg error	min error	std	time (h)	avg error	min error	std	time (h)
1	grid1010f10w	2	100	21	333.32	32	31	4.45	0.89	3.71	1.5	21	51k	87k	1.58	0.28	0.75	0.35	1.83	0.98	0.76	0.48
2	grid1010f10	2	100	13	303.09	1.58	8	0.7	0.05	0.54	0.3	13	23k	30k	1.28	0.59	0.41	0.06	1.18	0.86	0.32	0.05
3	grid2020f2	2	400	27	291.73	11.24	114	1.98	0.36	1.28	5.3	27	68k	177k	0.4	0.013	0.36	2.24	0.13	0.015	0.14	2.5
4	grid2020f10	2	400	27	1312	80.86	114	10.04	1.05	9.3	5.5	27	68k	177k	2.51	1.62	1.31	2.22	2.4	0.37	1.98	2.27
5	grid2020f5	2	400	27	665.12	39.44	114	5.75	0.24	3.1	5.5	27	68k	177k	0.84	0.37	0.53	1.95	0.8	0.081	0.78	2.3
6	grid2020f15	2	400	27	1963	122.91	114	17.8	3.08	9.8	3	27	68k	177k	2.39	0.41	2.16	2.2	2.68	1.08	2.54	2.2

(c) Grid-easy

Problem Description					ref Z	WMB	#NB	DBE (#h=100, N=320k)				NeuroBE (#h=3w, Nmin = 147k)										
i-bound=20												Statistics on resources			Loss : mean square error				Loss: Imp. mean square error			
								statistics on error (5 runs)							statistics on error (5 runs)				statistics on error (5 runs)			
Id	name	k	#v	w		error		avg error	min error	stdev	time (h)	h_{max}	N_{avg}	N_{max}	avg error	min error	std	time (h)	avg error	min error	std	time (h)
1	rbm20	2	40	21	58.53	0.0007	20	0.22	0.03	0.17	1.1	60	147k	147k	0.37	0.08	0.29	0.45	0.37	0.06	0.25	1.73
2	rbm21	2	42	22	63.15	6.39	22	0.48	0.27	0.19	1.3	63	163k	164k	0.9	0.59	0.2	0.46	0.57	0.15	0.26	1.75
3	rbm22	2	40	21	66.55	8.65	24	0.47	0.14	0.35	1	66	180k	182k	0.59	0.02	0.38	0.57	0.75	0.48	0.36	1.76
4	rbm-ferro20	2	44	23	151.16	0.005	20	1.33	0.29	1.21	1	60	147k	147k	1.11	0.38	0.75	0.44	0.75	0.37	0.32	1.13
5	rbm-ferro21	2	42	22	152.62	1.98	22	3.43	0.83	1.89	1.2	63	163k	164k	2.3	1.27	1.08	0.48	1.82	1.06	0.87	2.5
6	rbm-ferro22	2	44	23	166.11	0.517	24	6.52	3.86	1.5	1.3	66	180k	182k	5.32	2.515	2.69	0.59	4.17	3.8	0.32	1.29
i-bound=10												NeuroBE (#h=3w, Nmin = 82k)										
1	rbm20	2	40	21	58.53	7.85	30	0.49	0.11	0.35	1.6	100	82k	87k	1.57	0.89	1.04	0.3	1.43	0.83	0.82	0.8
2	rbm21	2	42	22	63.15	15.73	32	0.73	0.14	0.47	1.7	105	90k	109k	1.62	0.68	0.98	0.4	0.89	0.51	0.42	1.02
3	rbm22	2	40	21	66.55	27.46	34	0.65	0.19	0.41	1.8	110	99.5k	121k	0.89	0.03	0.6	0.46	0.94	0.83	0.61	1.3

(d) DBN

Figure 3: Results on performance of *NeuroBE* against *DBE* and *WMB*. k:domain size, $\#v$:variables, w:induced width, $\#NB$: number of buckets that are trained with NNs, $\#h$: number of hidden units per layer (reported maximum $\#h$ for *NeuroBE*), N: number of training samples (reported minimum, average and maximum $\#N$ for *NeuroBE*), $error$: L1 error for referenced and estimated $log(Z)$ (reported minimum, average, and standard deviation over 5 runs for *DBE* and *NeuroBE*), $time$: average time taken to get the estimated $error$, # in a cell denotes estimated partition function is $-\infty$. *Note: Here, referenced Z is approximated by Kask et al. (2020)

approximation. Then, the *Local Bucket Error* is the function

$$E = log\mu^* - log\mu$$

The *Global Bucket Error* is

$$G = log\lambda - log\mu$$

The above error corresponds to a log of the relative errors. We use log relative error here because bounding the global error as a function of the local errors turned out to be easier.

Theorem 1. *Assume a bucket-chain along an ordering d and let B be a bucket along the chain at index c. Let $E(s) = ln\mu^*(s) - ln\mu(s)$ as defined above and let $\epsilon = max_{s \in D(S)}|E(s)|$, where S is the scope of outgoing message from B and $D(S)$ is the set of all possible configurations on S. Then,*

$$E = ln\lambda - ln\mu \leq \sum_{k=0}^{n-c} \epsilon_{c+k}$$

In particular, since $\lambda_1 = Z$, the partition function

$$E_1 = lnZ - ln\mu_1 \leq \sum_{k=0}^{n-1} \epsilon_{1+k} \tag{9}$$

For the proof see the Appendix.

In the next section we evaluate $NeuroBE$.

5 EXPERIMENTS

5.1 EXPERIMENT SETUP

We ran experiments comparing *NeuroBE* against the Weighted Mini Bucket Elimination scheme (*WMB*) (Dechter and Rish, 2003; Liu and Ihler, 2012) and DBE (Razeghi et al., 2021).

Benchmarks Following the methodology in DBE, we evaluated $NeuroBE$ on instances selected from three well-known benchmarks from the UAI repository used in Kask et al. (2020), i.e. grids (vision domain), pedigree (genetic linkage analysis) and DBNs. We targeted diverse benchmarks (in structure and level of determinism) and aimed for different levels of hardness. Thus, in the grids benchmark, we distinguish between problems that can be solved exactly, which we call "grid-easy", and those that cannot be solved, called "grid-hard". We also distinguish benchmarks that possess *determinism*, namely have a high proportion of zero probabilities, a feature which can impact training. We randomly selected 13 instances from Grids, with easy ones (width 20-30) and hard ones (1600 variables, width 55 or 114), 6 from pedigrees, which posses high level of determinism and 6 from DBNs, totalling 25 instances.

NN architectures and sample sizes. To trigger bucket message approximations, we used i-bound=10 for easy problems and i-bound=20 for hard ones. For problems with determinism, such as pedigree, the NN architecture has 2 set of output layers, one indicating whether the input configuration has non-zero probability while the other specifies the message value if so. The loss is then a sum of the cross-entropy loss (from the first output layer) and the (importance) mean square loss (from the second output layer). We keep the #layers fixed (=2) across all benchmarks. Through a process of trial and error applied to a selected representative instance from each benchmark, we used as static parameters of the architecture and sample sizes as follows. We selected $h = 3w$ and $N_{avg} \in [149k, 350k]$ for pedigrees; $h = 3w$ and $N_{avg} \in [80k, 180k]$ for DBN; $h = w$ and $N_{avg} \in [23k, 68k]$ for grid-easy; and $h = w$ and $N_{avg} \in [75k, 150k]$ for grid-hard.

Performance measures We evaluate the performance of $NeuroBE$ using: $error = |log_e Z - log_e \hat{Z}|$ where $\hat{Z}$ is the estimate of the partition function, Z. When the exact Z is not available (for hard Grid benchmark), Z^* is a surrogate to Z, which is obtained using an advanced sampling scheme for a duration of $100 * 1hr$ (Kask et al., 2020).

5.2 RESULTS

Figure 3 compares $NeuroBE$ against WMB and DBE over the 3 benchmarks. The first few columns show the problem statistics for instances in the respective benchmarks (pedigree, DBN and grids). We then show WMB error, followed by the average error, minimum error, standard deviation and average time (in hours) over 5 runs (due to stochasticity) for both DBE and $NeuroBE$ with m.s.e and I.m.s.e loss functions. For $NeuroBE$, we also report the average and maximum #training samples, (N_{avg}, N_{max}) and maximum #hidden units, (h_{max}) across all buckets of the instance.

Pedigrees We observe immediately that overall $NeuroBE$ has far superior performance to DBE, especially with the I.m.s.e loss function. In particular, it is ≥ 5 times more accurate than DBE for almost all the instances and takes less time, since it uses far less training samples. Also, $NeuroBE$ outperforms WMB on most of the instances. Here, DBE yields either a similar (or even worse) accuracy as WMB.

Grids. Here too we observe that $NeuroBE$ outperforms DBE in accuracy, particularly with the I.m.s.e loss, as reflected by the average error and standard deviation, even though it uses far less time. In most cases, we see a reduction in time by a factor of 2 while still producing a far better estimate. $NeuroBE$ and DBE outperform WMB across all problem instances.

i-bound=20			h=3*w				h=5*w			
k	#v	w	#N_{avg} (10^3)	standard deviation	error	t(h)	#N_{avg} (10^3)	standard deviation	error	t(h)
3	888	33	218	0.25	1.11	7.5	437	0.17	0.34	14.3
5	871	35	259	0.89	3.51	10.4	518	1.57	1.42	18.3
5	693	28	149	0.6	2.61	5	298	0.51	0.68	8.2

(a) pedigree

i-bound=20			h=w				h=w			
Id	#v	w	#N_{avg} (10^3)	standard deviation	error	t(h)	#N_{avg} (10^3)	standard deviation	error	t(h)
5	1600	114	120	10.82	55.11	12	262	16.3	17.76	17
7	1600	114	120	3.6	5.5	10	262	1.39	2.34	14.5

(b) Grid-hard

i-bound=10			h=w				h=3*w			
Id	#v	w	#N_{avg} (10^3)	standard deviation	error	t(h)	#N_{avg} (10^3)	standard deviation	error	t(h)
1	100	21	51	0.76	1.83	0.48	202	0.08	1.2	1.13
2	100	13	23	0.32	1.18	0.05	38	0.15	0.12	0.09

(c) Grid-easy

i-bound=20			h=3*w				h=5*w			
Id	#v	w	#N_{avg} (10^3)	standard deviation	error	t(h)	#N_{avg} (10^3)	standard deviation	error	t(h)
1	40	21	147	0.25	0.37	1.73	245	0.07	0.2	3.2
2	42	22	163	0.26	0.57	1.75	272	0.24	0.77	3.7
3	40	21	163	0.36	0.75	1.76	300	0.22	0.51	4.5
4	44	23	147	0.32	0.75	1.13	245	0.61	0.67	2.17
5	42	22	164	0.87	1.82	1.3	272	0.23	1.06	2.6
6	44	23	182	0.32	4.17	1.29	300	0.86	3.43	2.9

(d) DBN

Figure 4: Performance of NeuroBE when increasing # sample &/or NN complexity. N_{avg}: average samples, $t(h)$: average time, $Error$: global error (reported average and standard deviation over 5 runs)

DBN We report results for the DBN benhcmark for 2 i-bounds. For i-bound=20, $NeuroBE$, mostly with the I.m.s.e loss achieves a higher accuracy than DBE with far less #training samples (but with more training time). It is superior to WMB on instances 2,3,5. However, WMB performs better on instance 1,4 & 6, as the induced-width is closer to the i-bound. For i-bound=10, $NeuroBE$ shows better accuracy than WMB for all three instances. It is comparable to DBE on most instances taking less time.

Overall, compared with DBE, $NeuroBE$ using I.m.s.e is about 50% faster while also far more accurate on pedigrees, twice as fast and 5 to 10 fold more accurate on hard grids. It is also faster and more accurate on easy grids and has a comparable performance on the DBN benchmark.

The impact of loss functions. We observe that $NeuroBE$ with the I.m.s.e loss shows better performance (lower average error and standard deviation) than $NeuroBE$ with the m.s.e loss for the pedigree instances, and for the majority of DBN, grid-easy and grid-hard instances. An F-test on the two groups of partition function estimates (each consisting of 5 approximations) showed that the means are *significantly* different for pedigrees, (Fig. 3(a)). For the grids and DBN, there was no statistical difference between the two means. However, by inspection, we see a reduction in the standard deviation for almost all instances.

Impact of architecture size. Figure 4 shows the impact of architecture size on time and accuracy on a few problem instances. We show the average error of the partition function estimate (what we call global error), and standard deviation (over 5 runs) and the computation time for 2 different NN architecture and their associated sample sizes. As expected we see that increasing the NN and sample sizes increases the time and accuracy for pedigrees. For grid-hard instances, we just increased the sample sizes for the same architecture having $h = w$, and observe that the average error is reduced, as expected. Instances from grid-easy and DBN (except, instance #2) show a similar improvement in performance with a larger NN and training sample size. This shows that the algorithm has an anytime characteristic, as it can improve its performance by controlling the size of the approximating NN matched by a suitable sample size for training.

6 CONCLUSION & FUTURE WORK

In this work, we advance the earlier theme of using Neural Networks to approximate the class of bucket-elimination algorithms that is at the heart to probabilistic reasoning. *NeuroBE* can be viewed as a realization of Neuro-Dynamic Programming schemes Bertsekas and Tsitsiklis (1996), in the context of graphical models. That being said, it requires training of numerous NN per problem instance and thus the central aim of $NeuroBE$'s design (customizing NN architectures, training samples and the loss function to the message) is to enhance *efficiency* of such schemes. We presented *NeuroBE* and illustrated, on challenging instances over three benchmarks that it can be far more accurate and requires less time compared with *Deep Bucket Elimination* (*DBE*) and is also superior to *weighted mini-bucket* (*WMB*) that cannot improve its accuracy once their memory is exhausted. We believe that *NeuroBE* has the potential to become more time efficient and may also extend into learning across a set of instances from the same benchmark domain.

Future Work. While $NeuroBE$ adjusts the NN architectures according to a bucket's scope, it keeps the #layers fixed, a hyper-parameter we wish to explore varying dynamically. We will also explore training a single function per union of buckets, which yield a cluster in a tree-decomposition (Dechter, 2013). This can significantly reduce the number of trained functions at the cost of more time for sample generation, a trade-off we plan to study. We will also explore the task of parameter sharing and training multiple bucket functions simultaneously.

References

Ralph Abboud, Ismail Ceylan, and Thomas Lukasiewicz. Learning to reason: Leveraging neural networks for approximate dnf counting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3097–3104, 2020.

Martin Anthony and Peter L. Bartlett. *Neural Network Learning - Theoretical Foundations*. Cambridge University Press, 2002. ISBN 978-0-521-57353-5.

Peter L. Bartlett, Nick Harvey, Christopher Liaw, and Abbas Mehrabian. Nearly-tight vc-dimension and pseudodimension bounds for piecewise linear neural networks. *Journal of Machine Learning Research*, 20(63):1–17, 2019. URL http://jmlr.org/papers/v20/17-612.html.

Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-dynamic programming*, volume 3 of *Optimization and neural computation series*. Athena Scientific, 1996. ISBN 1886529108. URL https://www.worldcat.org/oclc/35983505.

Yann Le Cun, Ido Kanter, and Sara A. Solla. Eigenvalues of covariance matrices: Application to neural-network learning. *Phys. Rev. Lett.*, 66:2396–2399, May 1991. doi: 10.1103/PhysRevLett.66.2396. URL https://link.aps.org/doi/10.1103/PhysRevLett.66.2396.

A. Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009.

R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113:41–85, 1999a.

R. Dechter. *Constraint Processing*. Morgan Kaufmann Publishers, 2003.

Rina Dechter. Bucket elimination: A unifying framework for reasoning. *Artif. Intell.*, 113(1-2):41–85, 1999b.

Rina Dechter. Reasoning with probabilistic and deterministic graphical models: Exact algorithms. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 7 (3):1–191, 2013.

Rina Dechter and Irina Rish. Mini-buckets: A general scheme for bounded inference. *Journal of the ACM (JACM)*, 50(2):107–153, 2003.

Nicolas Heess, Daniel Tarlow, and John Winn. Learning to pass expectation propagation messages. In *NIPS*, volume 26, pages 3219–3227, 2013.

Kalev Kask, Bobak Pezeshki, Filjor Broka, Alexander T. Ihler, and Rina Dechter. Scaling up AND/OR abstraction sampling. In *Proceedings of IJCAI 2020*, pages 4266–4274, 2020.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

Qiang Liu and Alexander Ihler. Belief propagation for structured decision making. In *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence*, pages 523–532, 2012.

Qiang Liu and Alexander T. Ihler. Bounding the partition function using holder's inequality. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 849–856, 2011.

Robert Mateescu, Kalev Kask, Vibhav Gogate, and Rina Dechter. Join-graph propagation algorithms. *J. Artif. Intell. Res. (JAIR)*, 37:279–328, 2010.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

D. Pollard. *Convergence of Stochastic Processes*. Springer-Verlag, 1984.

Yasaman Razeghi, Kalev Kask, Yadong Lu, Pierre Baldi, Sakshi Agarwal, and Rina Dechter. Deep bucket elimination. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 4235–4242. International Joint Conferences on Artificial Intelligence Organization, 8 2021. Main Track.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *Trans. Neur. Netw.*, 20(1):61–80, jan 2009. ISSN 1045-9227. doi: 10.1109/TNN.2008.2005605. URL https://doi.org/10.1109/TNN.2008.2005605.

Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, second edition, November 1999. ISBN 0387987800.

Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Generalized belief propagation. In *(NIPS) 2000*, pages 689–695. MIT Press, 2000.

KiJung Yoon, Renjie Liao, Yuwen Xiong, Lisa Zhang, Ethan Fetaya, Raquel Urtasun, Richard S. Zemel, and Xaq Pitkow. Inference in probabilistic graphical models by graph neural networks. 2018.

A APPENDIX

A.1 ESTIMATING THE PSEUDO-DIMENSION OF A NN:

In our work, we use NN architectures with ReLU activation functions. To construct a NN with L layers and a variable h, #hidden-units per layer to model a specific local bucket message μ^*, we pick the rule $h = b * w$ where w is the width and b is a constant. By doing this, the #parameters in the NN is :

$$|\theta| = (L-1)*b^2*w^2 + b*w^2 + (L+1)*b*w + 1 \quad (10)$$

We make use of the lower bound of pseudo-dimension for NNs with ReLU activation functions from the work in Bartlett et al. (2019) to get:

$$\rho = |\theta| * Llog(|\theta|/L) \quad (11)$$

By substituting Eq. 10 in Eq. 11 and ignoring all linear terms in w we get that $\rho(w)$ can be dominated by:

$$\rightarrow \rho_c(w_c) \propto (L*b*w)^2 log[(b*w)]$$

A.2 ESTIMATING ERROR IN PARTITION FUNCTION:

Theorem. 1 *Let B_c be a bucket in a bucket chain along an ordering d; let B_c contain the original functions as ϕ_c and μ_{c+1} as the message passed to it from the previous bucket; let λ_c be the (global) exact message generated in B_c, μ_c^* be the local exact message in B_c and $\mu_c = APP(\mu_c^*)$ its approximation (e.g., by a trained neural network). Let $E_c = ln\mu_c^* - ln\mu_c$ and and $\epsilon_c = max_{B_c}|E_c|$. Then,*

$$ln\lambda_c - ln\mu_c \leq \sum_{k=2}^{n-c} \epsilon_{c+k}$$

In particular, since $\lambda_1 = Z$, the partition function

$$lnZ - ln\mu_1 \leq \sum_{k=1}^{n-1} \epsilon_{1+k} \quad (12)$$

Proof. We will next derive the recursion, starting at the first processed bucket B_n and going down in order. Remember throughout that $\ln\mu_{n-i}^* = \sum_{X_{n-i}} ln(e^{ln\phi_{n-i}+ln\mu_{n-i+1}})$

For B_n $\lambda_n = \mu_n^*$, therefore

$$ln\lambda_n - ln\mu_n = ln\mu_n^* - ln\mu_n = E_n$$

For B_{n-1}, by definition

$$ln\lambda_{n-1} - ln\mu_{n-1} = ln \sum_{X_{n-1}} e^{ln\phi_{n-1}+ln\lambda_n} - ln\mu_{n-1}$$

Substituting $ln\lambda_n$ from B_n

$$= ln \sum_{X_{n-1}} e^{[(ln\phi_{n-1}+ln\mu_n)+E_n]} - ln\mu_{n-1}$$

$$= ln[\sum_{X_{n-1}} e^{(ln\phi_{n-1}+ln\mu_n)} e^{E_n}] - ln\mu_{n-1}$$

If $max_{scope(\mu_n^*)}|E_n| = \epsilon_n$, then,

$$\leq ln[e^{\epsilon_n} \sum_{X_{n-1}} e^{(ln\phi_{n-1}+ln\mu_n)}] - ln\mu_{n-1}$$

$$\leq \epsilon_n + ln \sum_{X_{n-1}} e^{(ln\phi_{n-1}+ln\mu_n)} - ln\mu_{n-1}$$

Since $ln \sum_{X_{n-1}} e^{ln\phi_{n-1}+ln\mu_n} = ln\mu_{n-1}^*$ we get

$$ln\lambda_{n-1} - ln\mu_{n-1} \leq \epsilon_n + ln\mu_{n-1}^* - ln\mu_{n-1} \quad (13)$$

or equivalently,

$$ln\lambda_{n-1} - ln\mu_{n-1} \leq \epsilon_n + E_{n-1} \quad (14)$$

Moving to B_{n-2}, by definition:

$$ln\lambda_{n-2} - ln\mu_{n-2} = ln \sum_{X_{n-2}} e^{ln\phi_{n-2}+ln\lambda_{n-1}} - ln\mu_{n-2} \quad (15)$$

Substituting $ln\lambda_{n-1}$ from Eq. (13) we get

$$ln\lambda_{n-2} - ln\mu_{n-2} \leq ln \sum_{X_{n-2}} e^{ln\phi_{n-2}+[ln\mu_{n-1}+\epsilon_n+E_{n-1}]} - ln\mu_{n-2}$$

$$\leq ln \sum_{X_{n-2}} e^{ln\phi_{n-2}+\mu_{n-1}} e^{\epsilon_n+E_{n-1}} - ln\mu_{n-2}$$

Taking $\max_{scope(\mu_{n-1}^*)} E_{n-1} = \epsilon_{n-1}$,

$$\leq lne^{\epsilon_n+\epsilon_{n-1}} \sum_{X_{n-2}} e^{ln\phi_{n-2}+\mu_{n-1}} - ln\mu_{n-2}$$

$$\leq \epsilon_n + \epsilon_{n-1} + ln \sum_{X_{n-2}} e^{ln\phi_{n-2}+\mu_{n-1}} - ln\mu_{n-2}$$

$$\leq \epsilon_n + \epsilon_{n-1} + ln\mu_{n-2}^* - ln\mu_{n-2}$$

yielding,

$$ln\lambda_{n-2} - ln\mu_{n-2} \leq E_{n-2} + \epsilon_{n-1} + \epsilon_n \quad (16)$$

Moving to bucket B_{n-3}, by definition

$$ln\lambda_{n-3} - ln\mu_{n-3} = ln \sum_{X_{n-3}} e^{ln\phi_{n-3}+ln\lambda_{n-2}} - ln\mu_{n-3}$$

Substituting for λ_{n-2} from Eq. (16) we get with some algebra

$$ln\lambda_{n-3} - ln\mu_{n-3}$$
$$\leq ln \sum_{X_{n-3}} e^{ln\phi_{n-3}+[ln\mu_{n-2}+E_{n-2}+\epsilon_{n-1}+\epsilon_n]} - ln\mu_{n-3}$$

yielding

$$ln\lambda_{n-3} - ln\mu_{n-3} \leq E_{n-3} + \epsilon_{n-2} + \epsilon_{n-1} + \epsilon_n$$

and so on. Clearly the emerging expression for bucket B_c is

$$ln\lambda_c - ln\mu_c \leq E_c + \epsilon_{c+1} + \epsilon_{c+2} + ... \quad (17)$$

or,

$$ln\lambda_c - ln\mu_c \leq E_c + \sum_{k=0}^{n-c-1} \epsilon_{c+1+k} \quad (18)$$

The general transition from $n-i$ to $n-i-1$ can be easily followed to complete the inductive proof.

Assuming that we control the derivation of μ_c for each B_c to ensure that $E_c = ln\mu_c^* - ln\mu_c \leq \epsilon_c$ and substituting in the expression we get from Eq. (18) that

$$ln\lambda_c - ln\mu_c \leq \epsilon_c + \sum_{k=0}^{n-c-1} \epsilon_{c+1+k} \leq (n-c+1) * \epsilon \quad (19)$$

□

A.3 MISCALLANEOUS EXPERIMENTS ON ANALYZING ERROR

Calculating ϵ from Theorem 1 is hard because it involves computing the local bucket error E over all configurations in the scope of the bucket. Therefore, we calculate the maximum over a sampled test set (lines 18-20 of algorithm 3) as $\hat{\epsilon}$.Additionally, we calculate the average local bucket error, $\hat{\epsilon}^{avg}$ over the same test set. To bound the global error of the approximated partition function from Eq. 9, we sum over all the estimated bucket error bounds, $\hat{\epsilon}$), Clearly, the bound is very lose. We therefore use the average local bucket error, $\hat{\epsilon}^{avg}$ to give us some additional information on the global error empirically:

$$\hat{E}_1 \leq \sum_{k=0}^{n-1} \hat{\epsilon}_{1+k}^{avg} \quad (20)$$

Relationship between local and global errors empirically. Figure 5a) depicts the empirical global errors against the local error bound for 4 grid instances over 2 sample configurations={60k,120k}. Specifically, the local error bound shown is the maximum over the estimated local bucket errors ($\hat{\epsilon}^{avg}$ from Eq. 20) across all buckets and the (empirical) global error is the error in the partition function estimate. As expected, we see a somewhat linear relationship between the global error and the local error bound. We also see that higher samples drive the local and global errors towards the lower-left of the plot and vice-versa.

Impact of sample size on error bounds. Figure 5b &c) depicts the impact of sample size on the estimated local and global error bounds (Eq. 20). Specifically, the local error bound shown is the maximum over the estimated local bucket errors ($\hat{\epsilon}^{avg}$ from Eq. 20) across all buckets. As expected, we see that increasing the training sample size makes the two bounds tighter. For the 4 grid instances (f10, f5, f2, f15), we also observed that the empirical global error in the partition function estimate for for the two sample configurations {(37.21, 7.94), (18.8, 5.9), (10.28, 3.05), (41.3, 27.5)} is in proportion to the global error bound from Fig. 5b).

	N_{avg} = 60k, h=w								N_{avg} = 150k, h=w							
	Statistics on local bucket errors				Statistics on global errors				Statistics on local bucket errors				Statistics on global errors			
Id	test w.m.s.e		local bucket errors		estimated bounds		empirical errors		test w.m.s.e		local bucket errors		estimated bounds		empirical errors	
	avg	max	avg	max	avg	max	avg	max	avg	max	avg	max	avg	max	avg	max
1	2.19E-04	0.06	1.67	3.81	251.7	1664	24.01	37.21	1.63E-05	0.006	0.29	3.71	71.75	1380	8.11	13.62
2	1.74E-04	0.053	0.784	2.44	115.6	796	21.15	31.32	1.40E-05	0.0068	0.131	1.45	34.34	680	6.4	12.3
3	1.06E-04	0.053	0.336	0.661	34.98	264.91	5.05	7.88	7.46E-06	0.004	0.045	0.46	10.28	228.59	4.3	5.95
4	1.94E-04	0.057	1.988	10.97	359.38	2549.28	22.43	27.68	1.85E-05	0.009	0.382	4.665	106.75	2166	31.03	57.46

Figure 5: Statistics of Local & bucket errors compared with global error over 5 runs for 4 grid-hard instances having w=55 with $i-$bound=20, where h=w, # buckets trained, $\#NB = 308$ for two different scales of smaples sizes. *test wmse* is the w.m.s.e of the learned NN over the test set; *local bucket error* is the average L1 error for $log\lambda$ approximations over all buckets; *estimated bounds* is the bound obtained in eq 12; *empirical error* is the average global error over 5 runs.