NeuroBE: NN Approximations to Bucket Elimination

Anonymous author

Anonymous Institution

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 evaluate the partition function, therefore approximations are often investigated. In this paper, we build upon one such recently introduced methodology called *Deep Bucket Elimination (DBE)* that uses classical Neural Networks (NNs) to approximate messages generated by BE in cases when buckets have large memory requirements. In the new scheme, called $NeuroBE$, we personalize the construction and learning of these NNs by utilizing prior information about message size and distribution. Our experiments demonstrate that this new scheme provides significant improvements over DBE, yielding more accurate estimates with less time. We also study the impact of the local errors in message approximations on the global accuracy of the estimand of the partition function.

Introduction

Two of the critical goals of probabilistic modeling is 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, or performing inference, 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 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 the bucket's scope or the bucket's width. Overall, this Bucket Elimination (BE) scheme (Dechter 1999), is time and memory exponential in the induced-width of the model's graph along the ordering. One approach for approximating BE is to substitute each bucket message with a surrogate in cases when it cannot be computed exactly. Namely, when its width is too high.

Such schemes that bound the time and space complexity of *BE* include the (weighted) mini-bucket scheme (Dechter and Rish 2003; Liu and Ihler 2012) and generalized belief propagation schemes (Yedidia, Freeman, and Weiss 2000; Mateescu et al. 2010). A recent approach called *Deep Bucket Elimination (DBE) (Razeghi et al. 2021)* approximates each bucket function with a neural network (NN). This new approach yielded far more accurate approximations to the partition function on several benchmarks compared with its competition (Razeghi et al. 2021). While this approach is inherently time consuming compared with other approximations, requiring training numerous NN functions for solving a single problem, it can improve with bounded memory unlike some of the other schemes.

Yet, DBE's original design can be improved significantly as we show in this paper.

Contributions. We present $NeuroBE$, that addresses the *one size fits all* policy in Deep Bucket Elimination by customizing the NN training to each bucket separately. In particular, we construct NN and sample sizes proportional to the size of messages. We also consider a new loss function for training, that takes into account the distribution of messages. We compare $NeuroBE$ with DBE and with other approximation schemes of BE. Lastly, we provide an analysis, relating local errors associated with individual messages to the global error associated with the estimated partition function.

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

Related work. As noted, approximating and bounding the *Bucket Elimination* algorithm has been carried out extensively over the years for all probabilistic queries. Well

Copyright © 2022, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

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 approximations to *BE* was introduced in (Razeghi et al. 2021). The idea was closest in spirit to the Neuro-Dynamic Programming scheme as outlined in (Bertsekas and Tsitsiklis 1996) where the cost-to-go functions generated by dynamic programming (that parallel messages) can be approximated by NNs. This 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)* were used to explicitly learn *messages* following the message-passing reasoning methods in graphical models. However, since a supervised end-to-end learning algorithm is derived to estimate the parameters of a GNN, these methods are either restricted to very small instances (Yoon et al. 2018; Heess, Tarlow, and Winn 2013) (i.e., ∼40 variables) or tackle problems with a known polynomial-time approximation (Abboud, Ceylan, and Lukasiewicz 2020). The class of algorithms we consider differs significantly, where we confine learning to within a problem instance *only*.

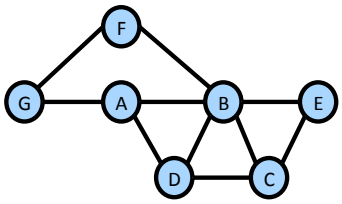
Background

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 $f_\alpha : D_\alpha \rightarrow R\geq 0$.

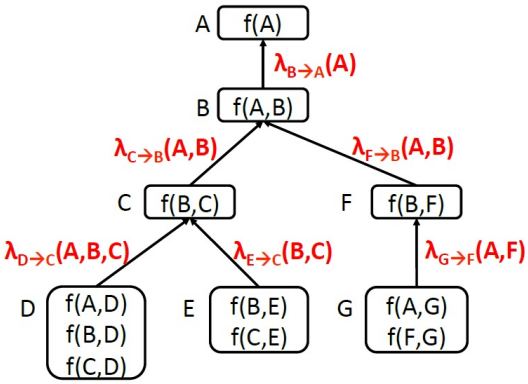
The **primal graph** of a graphical model associates each variable with a node. An edge between node i and node j is created if and only if there is a function containing X_i and X_j in its scope. Figure 1a shows a primal graph of a graphical model with variables indexed from A to G with functions over 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 $Z = \sum_X \prod_\alpha f_\alpha(X_\alpha)$.

Bucket Elimination

Bucket Elimination (BE) (Dechter 1999) is a universal exact algorithm for probabilistic inference. It is a variable elimination algorithm that can answer wide-range of queries, including the partition function. Given a variable ordering d, BE creates a *bucket tree* where each node is a bucket representing a variable in the ordering d. Each bucket contains a set of the model's functions depending on the given order of processing. There is an arc from bucket B_c to a parent bucket B_p, if X_p is the latest variable in λ_c's scope along the ordering (constants are placed in B_1). Figure 1b shows a bucket tree for the primal graph in Figure 1a along an ordering.



(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.

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 $\lambda_{c\rightarrow p}$, 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 the bucket of X_p, B_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 in a graph (Dechter 2013). Clearly, *BE* becomes impractical if the induced-width is large.

Deep Bucket Elimination

Given a variable ordering d, *Deep Bucket Elimination (DBE)* (Razeghi et al. 2021) approximates each message generated in sequence by training a NN when the scope of

any bucket message (S_c) is high ($\geq$ 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, $\lambda_{D\rightarrow C}(A, B, C)$, it sends a NN approximation $\mu_{D\rightarrow C}(\theta, A, B, C)$, as we describe next.

Approximating a bucket message in *DBE* is carried out by training a neural network. Given a generic bucket B_c with output message λ_c (to be approximated by a NN); whose width $w_c \geq i$-bound, DBE constructs a fully-connected feed-forward NN having w_c nodes in the input layer, L hidden layers with h hidden nodes per layer with $ReLU$ activation function and the output layer has one node with real-valued output. It then generates a train set, $\{(s_n, \lambda_c(s_n))\}$, of size N where s_n denotes a configuration of the variables in the bucket's message λ_c sampled uniformly at random over the bucket's scope and $\lambda_c(s_n)$ the message value defined by eq. 1. The parameters, θ, of the NN are trained to minimize the mean square error loss:

$$L(\theta) = \frac{1}{N}\sum_{n=1}^{N}(\lambda_c(s_n) - \mu_c(\theta, s_n))^2$$

where s_n is the nth sample in the training set and $\mu_c(\theta, s_n)$ is the output of the NN. Once the training is complete, DBE passes this NN representation $\mu_c(\theta)$ as a surrogate to the exponential sized exact message table to its parent bucket.

Usually, during the course of eliminating variables in the ordering d, DBE *approximates* multiple bucket messages. We refer to exact messages computed by BE as global exact message λ, and the messages approximated by a NN as μ. In contrast, we denote by μ^* the exact function computed by DBE when $w_c \geq i$-bound. Note that this function may or may not be the global exact message, λ.

Drawback The design of NNs in DBE however, are constrained to a constant size and always presented with a fixed (often large) train sample size. DBE also samples uniformly from the message scope across different messages, ignoring their individual distributions. Therefore, we ask the following questions: 1. Can we re-design the algorithm to sample small train set sizes while still maintaining overall performance? 2. Will tuning the NN and train sample sizes to accommodate information about the size of bucket messages help improve performance? 3. Should we sample the train set from the message distribution instead of sampling them uniformly?
We address the above questions in the design of our algorithm.

NeuroBE

We rename DBE to $NeuroBE$ acknowledging using shallow neural networks (2 layers) to approximate each message. Algorithm 1 presents *NeuroBE*. It differs from DBE in each message approximation scheme, "NN-message" in line 10. The "NN-message" procedure is described in detail in algorithm 2The procedure aims to enhance the accuracy and time taken for each message approximation by *constructing* and *learning* each NN to customize bucket message; i.e. its width and distribution. Following we describe the procedure in detail.

Algorithm 1: NeuroBE

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

1: **while** c $\in$ reverse(d) **do**
2: (Initialize buckets) put all unplaced functions mentioning X_c in B_c.
3: **end while**
4: **while** c $\in$ reverse(d) **do**
5: Let X_p be the closest ancestor variable of X_c in d that is in bucket B_c; if none exists, let $p = 0$.
6: Formulate message: $\mu^*_{c\rightarrow p} \leftarrow \sum_{X_c}\prod_{f_\alpha \in B_c} f_\alpha$
7: **if** $width(B_c) \leq i$ **then**
8: Put $\mu^*_{c\rightarrow p}$ in B_p
9: **else**
10: $\mu_{c\rightarrow p}(\theta) \leftarrow$ NN-message ($\mu^*_{c\rightarrow p}$, L, b, η)
11: Put $\mu_{c\rightarrow p}(\theta)$ in B_p
12: **end if**
13: **end while**
14: $Z = \prod_{f_\alpha \in B_0} f_\alpha$
15: **return** Z and All messages generated

Constructing NNs

The fixed sample size used by DBE to approximate each message was usually rather large, resulting in an unreasonable amount of training time. Instead, in $NeuroBE$ we aim for a small sample size to train each NN. In cases when the NN size is large, and small sample sizes are used to learn these complex NN architectures, over-fitting occurs. To avoid such a scenario, we design the capacity of NNs to reflect a-priori knowledge about the data it models as suggested by (Vapnik 1999). We, hence, express NN size as a function of the size of each message, which is exponential in it's width.

We propose a simple rule to adjust the NN size by making the number of hidden units, h, a function of w. Specifically, we select h as $b * w$, where b is a constant $\geq$ 1. Figure 2 illustrates an example NN model with fixed input size w and varying h by a constant b. In the simplest case, the size of h is the same as bucket width, w. Through such a rule, $NeuroBE$ *personalizes* NN capacity to message size. To guage the expressive power of the resulting NN architectures, we estimate their *pseudo-dimension* (Pollard 1984; Anthony and Bartlett 2002). Bounds to the pseudo-dimension of NN architectures with ReLU activation functions is provided by (Bartlett et al. 2019). We use the lower bound as a proxy to estimate the pseudo-dimension, ρ_c, of the NN approximating a message λ_c with width w_c as follows:

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

For the derivation please see the Appendix.

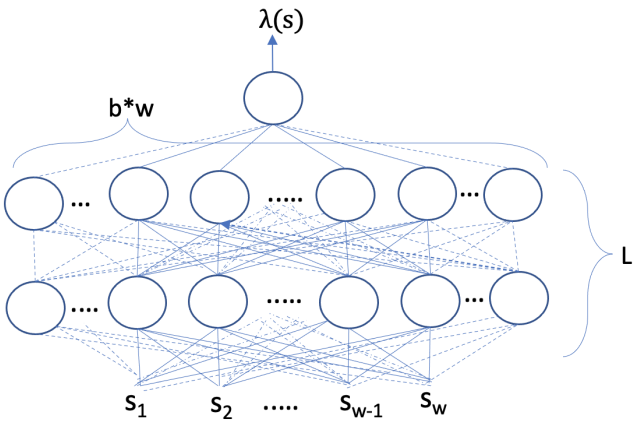


Figure 2: For a bucket of width w, we illustrate a variable L-layer, $\#bw$ hidden-units NN architecture.

Adapting Sample Complexity Inspired by (Vapnik 1999), we choose a sample size (N) proportional to the pseudo-dimension (eq. 2) of each NN:

$$N = \eta * (L * b * w_c)^2 log(b * w_c) \quad (3)$$

where η is a constant. Since N is also a function of w_c, we refer to sample size as $N(w_c)$ from now on. To avoid creating large training sets ($\geq$ threshold) when problems have high induced-width, we are forced to choose $\eta < 20$. Otherwise, for smaller problems, we let $\eta > 20$.

Learning NNs

Algorithm DBE used the mean square error as a loss. We propose to also take into account the message distribution in our loss function. We define the distribution of an output message $\lambda_c(S_c)$, $F_c(S_c)$, by:

$$F_c(S_c) = \frac{\lambda_c(S_c)}{\sum \lambda_c(S_c)} \quad (4)$$

Intuitively we think it may be good to sample from the above distribution. Since this is hard, we sample from any proposal distribution (e.g. uniform), and then (inspired by importance sampling), we use a weighted estimate of mean square error (w.m.s.e) as the loss:

$$L(\theta_c) = \frac{1}{N(w_c)} \sum_{n=1}^{N(w_c)} (\lambda_c(s_n) - \mu_c(\theta_c, s_n))^2 \frac{\hat{F}(s_n)}{\hat{U}(s_n)} \quad (5)$$

where c is an index to the bucket, s_n is a uniformly sampled configuration in scope S_c and $\mu(\theta_c, s_n)$ is the approximated message value computed by the NN; $\hat{F}(s_n)$ is the probability of s_n (eq 4), and $\hat{U}(s_n)$ is the probability of s_n sampled from the uniform distribution.

Algorithm 2 describes this. For a bucket B_c with output message λ_c; NN-message (algorithm 2) takes a set of input hyper-parameters L, b, η. It first constructs a NN with L layers, $b * w_c$ hidden units in line 2, then determines the sample size N in line 4. It then generates N samples uniformly in line 5 and approximates the bucket message by training the NN parameters θ_c using the w.m.s.e loss function in lines 8-9. After each epoch, the trained model is evaluated on a holdout validation set. We stop training when the maximum limit of epochs is reached or the validation error meets the early stopping criteria, that is if the validation error starts to increase for two consecutive epochs. We compute the maximum and average absolute errors on a test set to evaluate the performance of the trained model in line 13. The NN-message procedure then returns the approximated message $\mu_c(\theta)$ along with the error approximations on test set.

Algorithm 2: NN-message(λ,L,b, η, $\#epochs$)

Input: λ message on a set of variables X
Parameters: L: # layers in NN, $\#epochs$: a bound on the number of epochs, η, b: constants
Output: $\mu_c(\theta_c)$, NN message approximation, $\hat{\epsilon}$, an estimated bucket error bound, $\hat{\epsilon}_{avg}$ estimated average bucket error

1: $w_c \leftarrow$ scope(λ_c)
2: $\#h \leftarrow b * w_c$
3: Initialize NN with L layers, $\#h$ hidden-units
4: $N \leftarrow$ sample-size(L, w_c, η)
5: train, val, test $\leftarrow$ generate-data(λ, N)
6: p=1
7: **while** p $\leq$ #epochs and $\neg$ early_stopping(val_error) **do**
8: loss $\leftarrow$ w.m.s.e(train)
9: $\mu_c(\theta_c) \leftarrow$ train(NN, p ,train, loss)
10: val_error $\leftarrow$ test(NN, p ,val)
11: $p \leftarrow p + 1$
12: **end while**
13: $\hat{\epsilon}_c, \hat{\epsilon}_c^{avg} \leftarrow$ absolute_error(NN, test)
14: **return** $\mu_c(\theta_c)$, $\hat{\epsilon}_c$, $\hat{\epsilon}_c^{avg}$,

Complexity. Clearly, the time and space complexity for learning in $NeuroBE$ is linear in number of samples. In contrast to DBE it is customized to message complexity. Namely to w.

Error Analysis

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

Definition 0.1 (local and global errors)**.** Let λ_c be the (global) exact message generated in B_c by the exact BE algorithm, μ_c^* be the local exact message in B_c computed by the functions in it and $\mu_c =$ NN-message(μ_c^*) be its NN approximation. The Local Bucket Error is $E_c = log\mu_c^* - log\mu_c$. The Global Bucket Errors is $G_c = log\lambda_c - log\mu_c$.

Note that the true partition function, $Z^* = \lambda_1$. So, the global bucket error G_1 is the error in the estimated partition function, thus measuring $NeuroBE$'s performance.

Theorem 1. *Let B_c be a bucket in a bucket chain along an ordering d; λ_c be the (global) exact message generated*

Problem Description					ref Z	WMB		DBE (#h=100, N=320k)				NeuroBE (#h=3w, N(w,η=10), N_{min} = 49k)						
i-bound=20								statistics on error over 10 runs				Statistics on resources			statistics on error over 10 runs			
Id	name	k	#v	w		error	#NB	avg error	min error	stdev	time	h_{max}	N_{avg}	N_{max}	avg error	min error	std	time (h)
1	pedigree13	3	888	33	-31.18	6.4696	127	5.32	2.62	3.06	11.4	96	218k	706k	1.11	0.76	0.25	7.5
2	pedigree41	5	885	32	-76.04	4.1497	92	4.27	3.25	0.734	9.9	93	190k	658k	0.47	0.153	0.21	6.2
3	pedigree51	5	871	35	-77.27	9.7624	120	23.92	9.23	12.74	13.1	102	259k	809k	3.51	1.96	0.89	10.4
4	pedigree34	5	922	33	-64.23	7.0762	106	5.91	1.57	5.98	11.1	96	211k	706k	0.65	0.23	0.29	6.96
5	pedigree7	4	867	34	-64.82	6.0012	108	11.26	5.18	7.8	10.5	99	350k	900k	1.75	1.21	0.7	10.7
6	pedigree19	5	693	28	-59.020	2.5809	43	6.054	5.41	0.92	9.35	71	149k	482k	2.61	1.91	0.6	5

(a) pedigree

Problem Description					ref Z	WMB		DBE (#h=100, N=320k)				NeuroBE (#h=w, N(w,η=4), N_{min} = 19k)						
i-bound=20								statistics over 10 runs				Statistics on resources			statistics on error over 10 runs			
Id	name	k	#v	w		error	#NB	avg error	min error	stdev	time	h_{max}	N_{avg}	N_{max}	avg error	min error	std	time (h)
1	grid4040f10	2	1600	55	5490	215.45	308	**97.1**	**11.81**	65.15	11.8	55	60k	182k	**24**	**16.51**	8	5.4
2	grid4040f5	2	1600	55	2800	84.92	308	**39.9**	**6.28**	34.96	11.7	55	60k	182k	**21.2**	**18.87**	6.4	5.3
3	grid4040f2	2	1600	55	1220	25.24	308	**7.34**	**1.2**	5.4	11.5	55	60k	182k	**5.1**	**3.46**	1.8	5.2
4	grid4040f15	2	1600	55	8200	338.2	308	**83.46**	**41.78**	34.2	13.4	55	60k	182k	**22.4**	**5.38**	13.3	5.4
5	grid4040f10w	2	1600	114	5637	297.7	376	100.5	**6.4**	82.15	21.2	114	209k	900k	**51.05**	**29.72**	18.9	13.7
6	grid4040f5w	2	1600	114	2819	136.99	376	78.2	**72.62**	5.6	**21.3**	114	209k	900k	**16.6**	**12.28**	3.92	13.9
7	grid4040f2w	2	1600	114	1231	32	376	**15.12**	**0.92**	20.52	18.2	114	209k	900k	25.78	**12.61**	17.9	11.8
8	grid4040f15w	2	1600	114	8230	657.03	376	**220.91**	**95.23**	192.2	17.7	114	209k	900k	103.9	79.59	27.8	**13.7**

(b) Grid-hard

Problem Description					ref Z	WMB		DBE (#h=100, N=320k)				NeuroBE (#h=w, N(w,η=20), N_{min} = 10k)						
i-bound=10								statistics on error over 10 runs				Statistics on resources			statistics on error over 10 runs			
Id	name	k	#v	w		error	#NB	avg error	min error	stdev	time	h_{max}	N_{avg}	N_{max}	avg error	min error	std	time (h)
1	grid1010f10w	2	100	21	333.32	32	31	**4.45**	**0.89**	3.71	1.53	21	25k	48k	**3.51**	**1.45**	1.21	0.35
2	grid1010f10	2	100	13	303.086	1.58	8	**0.7**	**0.05**	0.54	0.32	13	11.6k	15.5k	**1.1**	**0.34**	0.7	0.055
3	grid2020f2	2	400	27	291.733	11.24	114	**1.98**	**0.36**	1.28	5.34	27	68k	480k	**0.37**	**0.174**	0.21	2.24
4	grid2020f10	2	400	27	1311.98	80.86	114	**10.04**	**1.05**	9.3	5.5	27	68k	480k	**1.37**	**0.08**	1.05	2.22
5	grid2020f5	2	400	27	665.12	39.44	114	5.75	**0.24**	3.1	5.5	27	121k	965k	**0.67**	**0.018**	0.56	0.91
6	grid2020f15	2	400	27	1962.98	122.91	114	17.8	**3.08**	9.8	**2.95**	27	68k	480k	**2.35**	**0.53**	1.57	2.2

(c) Grid-easy

Problem Description					ref Z	WMB		DBE (#h=100, N=320k)				NeuroBE (#h=3w, N(w,η=30), N_{min} = 147k)						
i-bound=20								statistics on error over 10 runs				Statistics on resources			statistics on error over 10 runs			
Id	name	k	#v	w		error	#NB	avg error	min error	stdev	time	h_{max}	N_{avg}	N_{max}	avg error	min error	std	time (h)
1	rbm20	2	40	21	58.53	0.0007	20	0.22	0.034	0.17	1.05	60	147k	147k	0.23	0.002	0.21	0.45
2	rbm21	2	42	22	63.15	6.39	22	0.48	**0.27**	0.19	1.25	63	163k	164k	0.78	0.133	0.8	0.46
3	rbm22	2	40	21	66.55	8.65	24	0.47	**0.14**	0.35	1	66	180k	182k	0.32	0.035	0.26	0.57
4	rbm-ferro20	2	44	23	151.16	0.005	20	1.33	0.29	1.21	1.03	60	147k	147k	1.58	0.38	1.15	0.44
5	rbm-ferro21	2	42	22	152.62	1.98	22	3.43	**0.83**	1.89	1.17	63	163k	164k	2.79	0.125	2.57	0.48
6	rbm-ferro22	2	44	23	166.11	0.517	24	6.52	3.86	1.5	1.3	66	180k	182k	4.69	1.09	3.2	0.59
i-bound=10												NeuroBE (#h=5w, N(w,η=20), N_{min} = 20k)						
1	rbm20	2	40	21	58.53	7.85	30	0.49	0.11	0.35	1.57	100	82k	87k	1.35	0.46	0.96	0.42
2	rbm21	2	42	22	63.15	15.73	32	0.73	0.14	0.47	1.71	105	90k	109k	0.43	0.14	0.35	0.5
3	rbm22	2	40	21	66.55	27.46	34	0.65	0.19	0.41	1.83	110	99.5k	121k	0.46	0.19	0.49	0.54

(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 train 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 $error$:. *Note: Here, referenced Z is approximated by (Kask et al. 2020)

in B_c, μ_c^* be the local exact message in B_c and μ_c = NN-message(μ_c^*) its approximation. Let $E_c(x) = ln\mu_c^*(x) - ln\mu_c(x)$ and and $\epsilon_c = max_{x \in S_{B_c}} |E_c(x)|$. Then,

$$ln\lambda_c - ln\mu_c \leq \sum_{k=0}^{n-c} \epsilon_{c+k}$$

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

$$lnZ - ln\mu_1 \leq \sum_{k=0}^{n-1} \epsilon_{1+k} \tag{6}$$

For the proof see the Appendix.

Step 13 in algorithm 2 estimates the local bucket error bound ϵ_c as $\hat{\epsilon}_c$ and an average local bucket error, $\hat{\epsilon}_c^{avg}$ over a test set. According to eq. 6, summing over bucket error bounds, $\hat{\epsilon}_c$, limits the global error in the estimated partition function. Clearly, this estimated global error bound is very lose. Therefore, we intuitively also sum over the average global bucket error, $\hat{\epsilon}_{avg}$ to estimate an average global error. In the next section, in addition to evaluating the performance of $NeuroBE$, we attempt to understand the behaviour of such global error estimates with local bucket errors across a few problem instances.

Experiments

Experiment Setup

We ran experiments comparing *NeuroBE* against the Weighted Mini Bucket Elimination scheme (*WMBE*) (Dechter and Rish 2003; Liu and Ihler 2012) and DBE (Razeghi et al. 2021). Following the methodology in DBE, we evaluate $NeuroBE$ 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 each benchmark, we distinguish between problems that can be solved exactly, which we call "easy", and those that cannot be solved, called "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 14 instances from Grids, with easy ones (i.e., width 20-30) and hard ones (i.e., 1600 variables, width 55), 6 from pedigrees, which posses high level of determinism and 6 from DBNs, totalling 25 instances. To trigger bucket message approximations, we put i-bound=10 for easy problems and the highest, 20 for hard ones.

For problems with determinism, such as pedigree, the NN has a slightly different architecture, where the same input layer is connected to two parallel set of hidden layers along with their individual output layers, one of which indicates 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 weighted mean square loss (from the second output layer).

Constructing NNs. We keep #layers fixed (=2) across all our experiments and vary $h \in [w, 5w]$. We restrict the

Problem Description					ref Z	NeuroBE		Statistics on Global Error (over 5 runs)					
i-bound=20						(h=3w)		m.s.e loss			w.m.s.e loss		
Id	name	k	#v	w		#NB	N_{avg}	avg error	min error	stdev	avg error	min error	stdev
1	pedigree13	3	888	33	-31.18	127	218k	9.04	7.86	0.8	1.11	0.76	0.25
2	pedigree41	5	885	32	-76.04	92	190k	10.2	8.4	2.16	0.47	0.153	0.21
3	pedigree51	5	871	35	-77.27	120	259k	11.725	10.37	1.26	3.51	1.96	0.89
4	pedigree34	5	922	33	-64.23	106	211k	6.14	4.56	4.14	0.65	0.23	0.29
6	pedigree19	5	693	28	-59.020	43	149k	9.14	8.7	0.6	2.61	1.91	0.6

Figure 4: Comparing m.s.e and w.m.s.e loss function with NeuroBE on pedigrees. $\#NB$: # buckets trained, N_{avg}: average samples, $avg\ error$: average global error, $stdev$: standard deviation on global error (over 5 runs).

average number of samples for a problem instance, N_{avg} to be small, i.e. $\leq 300k$. For each problem instance per benchmark, we pick the constant η such that the maximum sample size N, corresponding to bucket with the highest width, is small. Keeping η fixed and varying $h \in [w, 5w]$, we pick the configuration of h which gives the smallest average error to estimate the partition function. We then use the particular such a configuration for the rest of the problem instances across a benchmark. For instance, we chose the configuration $h = 3w$ and $N_{avg} \in [149k, 350k]$ for the pedigree benchmark; $h = \{3w, 5w\}$ and $N_{avg} \in [80k, 180k]$ for the DBN benchmark; $h = w$ and $N_{avg} \in [12k, 121k]$ for the grid-easy benchmark; and $h = w$ and $N_{avg} \in [60k, 209k]$ for the grid-hard benchmark.

Training NNs Bucket messages can either be small (eg. $exp(-11)$) in the case of pedigree benchmarks (deterministic) or large (eg. $exp(51)$) for grid and DBN benchmarks. To handle large values of messages in the non-deterministic benchmark domains, we use $log\lambda$ transformations. We normalize the output values for NNs across benchmarks to be in $[0, 1]$ to accelerate training (Cun, Kanter, and Solla 1991)). In the case of pedigree benchmarks with higher domain size, $k \in [3, 5]$, we similarly normalize the input configuration values to be in $[-1, 1]$. We create a data set with $N(w_c)$ training samples (eq 3), a fraction $\frac{N(w_c)}{9}$ as validation samples, and $50k$ test samples. We train the network using the Adam optimizer (Kingma and Ba 2014) with a learning rate of 0.001 and a batchsize of 256. The NN model is evaluated on a holdout validation set after each epoch. We terminate training either when the bound #epochs (500) is reached or the early stopping criteria is met, that is if the validation error starts to increase for two consecutive epochs. Subsequently, we evaluate the trained model on the test set.

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

	N_{avg} = 60k, h=w								N_{avg} = 150k, h=w							
	Statistics on bucket errors				Statistics on global errors				Statistics on 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 & Global bucket errors errors over 5 runs for 4 grid-hard instances with w=55, h=w, # buckets trained, $\#NB = 308$. test wmse: weighted mean square error in NN approximations over a test set; local bucket error: statistics on L1 error for $log\lambda$ approximations over all buckets; estimated bounds: approximated bounds from eq 8; empirical error is the global error over 5 runs.

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	888	33	218	0.25	1.11	7.5	437	0.17	0.34	14.3
3	871	35	259	0.89	**3.51**	10.4	518	1.57	**1.42**	18.3
6	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)
1	1600	55	60	7.98	24.01	5.4	150	5.17	8.12	9.1
2	1600	55	60	6.4	**21.2**	5.3	150	4.12	**6.4**	9.2
4	1600	55	60	1.8	5.1	5.2	300	1.23	2.08	14
7	1600	114	209	3.92	16.6	13.9	392	1.01	6.67	22.7

(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	25	1.21	3.51	0.35	42	0.88	1.38	0.4
2	100	13	11.6	0.7	**1.1**	0.055	19	0.4	**0.42**	0.07

(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	0.21	0.23	0.45	489	0.09	0.2	1.1
2	42	22	163	0.8	**0.78**	0.46	544	0.33	**0.55**	1.5
4	44	23	147	1.15	1.58	0.44	489	0.63	0.62	1.13
5	42	22	164	2.57	2.79	0.48	13	0.82	1.12	0.16
6	44	23	182	3.2	4.69	0.59	600	0.78	2.87	1.7

(d) DBN

Figure 6: Performance of NeuroBE when increasing # samples &/or NN complexity. N_{avg}: average samples, $t(h)$: average time, $Error$: global error (reported average and standard deviation over 5 runs; except instance 7 from grid-hard(#runs=2)).

Results

Figure 3 compares $NeuroBE$ against DBE and WMB on 3 benchmarks. It shows the problem statistics for the benchmarks (pedigree, DBN and grids) in the first few columns. For $NeuroBE$, we report the statistics (average and maximum) for #train samples, (N) and maximum #hidden units, (h) across buckets. We report the statistics of global error over 5 runs for DBE and $NeuroBE$. We consider the average global error as a representation of the global error for each instance. Lastly, we report the average time (in hours).

Pedigree We see a consistent decrease in both global average error and standard deviation for partition function estimates using $NeuroBE$ when compared with DBE, even with less number of training samples. $NeuroBE$ further provides more accurate estimates in less time. In particular, $NeuroBE$ is $\geq$5 times more accurate than DBE for 5 instances, and WMB for 4 instances. $NeuroBE$ outperforms WMB across all problem instances for the same i-bound $= 20$, except instance 6 which shows similar performance. DBE, on the other hand is either similar or even worse (instances 3,5,6) in performance when compared with WMB.

Grids Here, we again observe better performance of $NeuroBE$ compared with DBE in both error and standard deviation while taking far less time. In many problem instances, we reduce the time by half or more (IDs 1,3,4,5 from grid-easy and IDs 1,2,3,4,6,7 from grid-hard). However, for 2 instances (easy #2, hard #5) we see slightly worse performance than DBE. $NeuroBE$ and DBE outperform WMB across all problem instances.

DBN When i-bound=20, *NeuroBE* achieves a higher accuracy in far less time for instances 3,5,6 when compared with DBE, and instances 2,3 when compared with WMB. WMB performs much better for instance 1,4 & 6, typically expected since the induced-width is closer to the i-bound. For instances 1 & 4, $NeuroBE$ reaches an average error similar to DBE in half the time. When i-bound=10, $NeuroBE$ performs better in terms of accuracy and time for instances 2,3 when compared with DBE. $NeuroBE$ outperforms WMB for all three instances in terms of accuracy.

Overall, when compared with DBE, $NeuroBE$ is more than 2 times faster on 11 grid + DBN instances; 6 times more accurate for 4 easy grid instances; more than 2 times accurate for 5 grid-hard instances; 5 times more accurate for 5 pedigree instances.

Loss Functions. Figure 4 shows the comparison between the mean square error and weighted mean square error loss function for the pedigree benchmark using the proposed $NeuroBE$ method where $h = 3w$. We observe that with

w.m.s.e loss function, $NeuroBE$ performs much better for all instances (for instance 2, training with w.m.s.e is 20 times more accurate).

Time & **accuracy.** Figure 6 shows the trade-off between time and accuracy for a few problem instances selected randomly across benchmarks. In this table, we report the average global error, standard deviation (over 5 runs) and time for 2 different configurations in NN and sample complexity. For pedigree instances, we see that increasing the NN and sample complexity takes more time for the algorithm to compute the partition function. However, the estimates are more accurate. For grid-hard instances, we simply increase the sample sizes for $h = w$, and observe that with more samples, the average global error is reduced by approximately a factor of 3. Instances from grid-easy and DBN show a similar improvement in performance with a larger NN and a corresponding larger train sample size.

Global & **local Bucket Errors.** Figure 5 reports statistics on bucket errors, namely the weighted mean square error over a test set and local bucket errors, defined in definition 0.1; followed by estimates to global errors from eq. 6; and lastly, empirical global errors averaged over 5 runs. We report them for 2 sets of average sample size $N_{avg} = \{60k, 150k\}$ for 4 representative instances in grid-hard problems. 1) We observe a consistent decrease in the local bucket and global errors when more samples are given. However, instance 4 shows an increase in the empirical global error, which we account to stochasticity. 2) We also observe a direct correlation between the estimated local bucket errors and empirical average global errors. For example, instance 1 gives an average local bucket error of 1.67 and a corresponding empirical global error of 24.01; instance 2 gives an average local bucket error of 0.784 with 21.15 as the empirical global error. Note that both errors simultaneously decrease. This trend continues for instance 3, with a local bucket error of 0.336 while the global error is 5.05. We observe such a dependency of the empirical average global errors on the estimated local bucket errors across all instances, which is affirmative.

Conclusion & Future Work

The central theme around this work is to *efficiently* approximate messages to compute the partition function. We present *NeuroBE* and show, on challenging instances from three benchmarks, that it can be far more accurate when compared with *Deep Bucket Elimination DBE* and *weighted mini-bucket WMB*, the two most powerful approximations of Bucket Elimination. With customizing NN architectures and train samples to fit each bucket function, *NeuroBE* is capable of more accurate estimations, given more time while *DBE* on the other hand did not offer such a flexibility. *WMB*, and other iterative *BE* approximations (Mateescu et al. 2010) also cannot improve their accuracy once their memory is exhausted. We believe that *NeuroBE* has great 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 and seeks for user inputs for such variations. We will explore how to fine-tune NN capacity dynamically during training. Since the # buckets trained have a direct effect on $NeuroBE$'s accuracy and time performance, we will explore reducing the number of trained functions by 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 thus, training of multiple bucket functions simultaneously.

References

Abboud, R.; Ceylan, I.; and Lukasiewicz, T. 2020. Learning to reason: Leveraging neural networks for approximate DNF counting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 3097–3104.

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

Bartlett, P. L.; Harvey, N.; Liaw, C.; and Mehrabian, A. 2019. Nearly-tight VC-dimension and Pseudodimension Bounds for Piecewise Linear Neural Networks. *Journal of Machine Learning Research*, 20(63): 1–17.

Bertsekas, D. P.; and Tsitsiklis, J. N. 1996. *Neuro-dynamic programming*, volume 3 of *Optimization and neural computation series*. Athena Scientific. ISBN 1886529108.

Cun, Y. L.; Kanter, I.; and Solla, S. A. 1991. Eigenvalues of covariance matrices: Application to neural-network learning. *Phys. Rev. Lett.*, 66: 2396–2399.

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

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

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

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

Dechter, R.; and Rish, I. 2003. Mini-buckets: A General Scheme for Bounded Inference. *Journal of the ACM (JACM)*, 50(2): 107–153.

Heess, N.; Tarlow, D.; and Winn, J. 2013. Learning to Pass Expectation Propagation Messages. In *NIPS*, volume 26, 3219–3227.

Kask, K.; Pezeshki, B.; Broka, F.; Ihler, A. T.; and Dechter, R. 2020. Scaling Up AND/OR Abstraction Sampling. In *Proceedings of IJCAI 2020*, 4266–4274.

Kingma, D. P.; and Ba, J. 2014. Adam: A Method for Stochastic Optimization.

Liu, Q.; and Ihler, A. 2012. Belief Propagation for Structured Decision Making. In *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence*, 523–532.

Liu, Q.; and Ihler, A. T. 2011. 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, 849–856.

Mateescu, R.; Kask, K.; Gogate, V.; and Dechter, R. 2010. Join-Graph Propagation Algorithms. *J. Artif. Intell. Res. (JAIR)*, 37: 279–328.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533.

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

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

Razeghi, Y.; Kask, K.; Lu, Y.; Baldi, P.; Agarwal, S.; and Dechter, R. 2021. Deep Bucket Elimination. In Zhou, Z.-H., ed., *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, 4235–4242. International Joint Conferences on Artificial Intelligence Organization. Main Track.

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

Yedidia, J. S.; Freeman, W. T.; and Weiss, Y. 2000. Generalized Belief Propagation. In *(NIPS) 2000*, 689–695. MIT Press.

Yoon, K.; Liao, R.; Xiong, Y.; Zhang, L.; Fetaya, E.; Urtasun, R.; Zemel, R. S.; and Pitkow, X. 2018. Inference in probabilistic graphical models by Graph Neural Networks.

Appendix

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 #hidden-units per layer to model a specific bucket message λ_c, we pick the rule $h = b * w_c$ where w_c is the width and b is a constant. By doing this, the #parameters in the NN is :

$$|\theta_c| = (L-1) * b^2 * w_c^2 + b * w_c^2 + (L+1) * b * w_c + 1 \quad (7)$$

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:

$$\rho_c = |\theta_c| * Llog(|\theta_c|/L)$$

We simplify the dependency of ρ_c on $|\theta_c|$ by taking the term quadratic in width w_c from eq 7 to give the following:

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

Estimating error in partition function:

Theorem. 1 *Let B_c be a bucket in a bucket chain along an ordering d; 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

$$lnZ - ln\mu_1 \leq \sum_{k=1}^{n-1} \epsilon_{1+k} \quad (8)$$

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

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

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

If $max_{B_n}|E_n| = \epsilon_n$, then,

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

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

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

$$ln\lambda_{n-1} - ln\mu_{n-1} \leq \epsilon_n + ln\mu_{n-1}^* - ln\mu_{n-1} \quad (9)$$

or equivalently,

$$ln\lambda_{n-1} - ln\mu_{n-1} \leq \epsilon_n + E_{n-1} \quad (10)$$

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

$$ln\lambda_{n-2} - ln\mu_{n-2} = ln \sum_{X_{n-2}} e^{ln\phi_{n-2}+ln\lambda_{n-1}} - ln\mu_{n-2} \quad (11)$$

Substituting $ln\lambda_{n-1}$ from Eq. (9) 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,

$$ln\lambda_{n-2} - ln\mu_{n-2} \leq E_{n-2} + \epsilon_{n-1} + \epsilon_n \quad (12)$$

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. (12) 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

$$ln\lambda_c - ln\mu_c \leq E_c + \epsilon_{c+1} + \epsilon_{c+2} + ... \quad (13)$$

or,

$$ln\lambda_c - ln\mu_c \leq E_c + \sum_{k=0}^{n-c-1} \epsilon_{c+1+k} \quad (14)$$

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. (14) that

$$ln\lambda_c - ln\mu_c \leq \epsilon_c + \sum_{k=0}^{n-c-1} \epsilon_{c+1+k} \leq (n-c+1) * \epsilon \quad (15)$$

□