COMBINING LEARNED REPRESENTATIONS FOR COMBINATORIAL OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose a new approach to combine Restricted Boltzmann Machines (RBMs) that can be used to solve combinatorial optimization problems. This allows synthesis of larger models from smaller RBMs that have been pretrained, thus effectively bypassing the problem of learning in large RBMs, and creating a system able to model a large, complex multi-modal space. We validate this approach by using learned representations to create "invertible boolean logic", where we can use Markov chain Monte Carlo (MCMC) approaches to find the solution to large scale combinatorial optimization problems. Using this method, we are able to solve 64 bit addition based problems, as well as factorize 16 bit numbers. We find that these combined representations can provide a more accurate result for the same sample size as compared to a fully trained model.

1 INTRODUCTION

The Ising Problem has long been known to be in the class of NP-Hard problems, with no exact polynomial solution existing. Because of this, a large class of combinatorial optimization problems can be reformulated as Ising problems and solved by finding the ground state of that system (Barahona, 1982; Kirkpatrick et al., 1983; Lucas, 2014). The Boltzmann Machine (Ackley et al., 1987) was originally introduced as a constraint satisfaction network based on the Ising model problem, where the weights would encode some global constraints, and stochastic units were used to escape local minima. The original Boltzmann Machine found favor as a method to solve various combinatorial optimization problems (Korst & Aarts, 1989). However, learning was very slow with this model due to the difficulties with sampling and convergence, as well as the inability to exactly calculate the partition function. More recently, the Restricted Boltzmann Machine (RBM) has experienced a resurgence as a generative model that is able to fully approximate a probability distribution over binary variables due to its ease of computation and training via the contrastive divergence method (Hinton, 2002). The success of the RBM as a generative model has been limited due to the difficulties in running the Markov chain Monte Carlo (MCMC) algorithm to convergence (Tieleman, 2008; Tieleman & Hinton, 2009).

In this work, we propose a generative model composed of multiple learned modules that is able to solve a larger problem than the individually trained parts. This allows us to circumvent the problem of training large modules (which is equivalent to solving the optimization problem in the first place), thus minimizing training time. As RBMs have the ability to fill in partial values and solutions, this approach is very flexible to the broad class of combinatorial optimization problems which can be composed of individual atomic operations or parts. The ability of RBMs to fully model a probability distribution ensures model convergence and gives ideas about the shape of the underlying distribution. Many other generative models, such as Generative Adverserial Neural Networks (GANs), (Goodfellow et al., 2014) Generative Stochastic Networks (Alain et al., 2015) and others do not explicitly model the probability distribution in question, but rather train a generative machine to draw samples from the desired distribution. Although these can be useful for modeling high dimensional data, they do not provide the same guarantees that RBMs do. Because we use an RBM as our generative model we can perform a full Bayesian analysis, and condition on any subset of the variables to solve a variety of problems. This leads to increased model flexibility, and the ability to generalize the learned models further.

2 RELATED WORK

People have shown that learned and trained features have the ability to outperform hand calculated features in a variety of tasks, from image classification to speech detection and many others. In addition, other network architectures, such as Recurrent Neural Networks (RNNs) have been shown to be Turing complete, and posses the ability to be used as a conventional Turing machine (Graves et al., 2014; Zaremba & Sutskever, 2014). However, these architectures have mostly been used as generalized computers, rather than to solve specific problems. In addition, these models are not generative or reversible.

Many attempts have also been made to use quantum computers to solve such Ising model based problems. Large scale realizations of quantum computers, however, are still far from completion (Whitfield et al., 2012; Lucas, 2014; Roland & Cerf, 2002; Xu et al., 2012). Therefore, methods than can exploit a classical hardware are of critical importance.

In this regard, one recent work has proposed the use of "p-bits", to realize a form of Boltzmann Machine (Camsari et al., 2017). The work by (Traversa & Di Ventra, 2017) similarly tries to create an "invertible boolean logic", but does so in a deterministic manner, rather than the probabilistic one presented in this work.

Our approach falls within the broad category of transfer learning and compositional learning. There have been other works on using combinations of RBMs for object recognition and computer vision tasks (Ranzato et al., 2010), and on combinations of RBMs with weight sharing for collaborative filtering (Salakhutdinov et al., 2007). Nonetheless, the specific method presented here, as described in the following sections, to the best of our knowledge has not been used before.

3 APPROACH

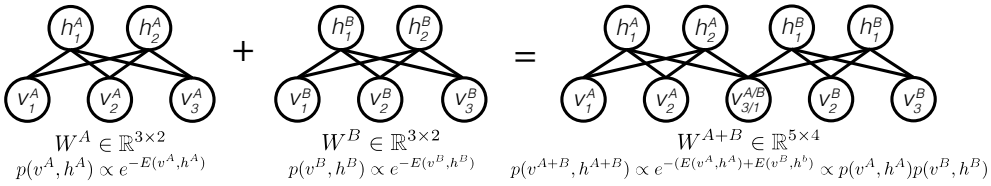

Figure 1: Merging procedure to combine RBMs. This combination scheme retains both the bipartite nature of the graph, as well as the product of experts nature. This allows the combined model to retain the sharp distributions of the original models.

An RBM is a binary stochastic neural network, which can be understood as a Markov random field of binary variables divided into a bipartite graph structure, where all of the visible units are conditionally independent of each other given the hidden unit states, and all hidden units are conditionally independent given the visible unit states. Here we denote v as the visible state, h as the hidden states, and $E(v, h)$ as the energy associated with those states. The probability assigned to a given state $p(v, h) = \frac{1}{Z}e^{-E(v,h)}$ where $Z = \sum_{v,h} e^{-E(v,h)}$ is the normalizing constant of the distribution.

The general approach to solving these combinatorial optimization problems is to recognize the atomic unit necessary to solve the problem, and use this as a base trained unit. We combine these units by performing a merge operation of different RBMs. If we know two different units share a connection (such as the output of one logical unit being the input of another), then these units would be merged to create a larger unit (see Figure 1). The probability distribution of this merged distribution can be recognized as the multiplication of two product of experts models.

$$E_A(v, h) = -v^T W_A h - a_A^T h - b_A^T v; p_A(v, h) = \frac{1}{Z_A} e^{-E_A(v,h)}$$

$$E_B(v,h) = -v^T W_B h - a_B^T h - b_B^T v; p_B(v,h) = \frac{1}{Z_B} e^{-E_B(v,h)}$$

$$W^A = \begin{bmatrix} w_{11}^A & w_{12} \\ w_{21}^A & w_{22} \\ w_{31}^A & w_{32}^A \end{bmatrix} W^B = \begin{bmatrix} w_{11}^B & w_{12}^B \\ w_{21}^B & w_{22}^B \\ w_{31}^B & w_{32}^B \end{bmatrix} W^{A+B} = \begin{bmatrix} w_{11}^A & w_{12} & 0 & 0 \\ w_{21}^A & w_{22} & 0 & 0 \\ w_{31}^A & w_{32}^A & w_{11}^B & w_{12}^B \\ 0 & 0 & w_{21}^B & w_{22}^B \\ 0 & 0 & w_{31}^B & w_{32}^B \end{bmatrix}$$

$$E_{A+B}(v,h) = E_A(v_A,h_A) + E_B(v_B,h_B), v_A = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}, v_B = \begin{bmatrix} v_3 \\ v_4 \\ v_5 \end{bmatrix}, h_A = \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} h_B = \begin{bmatrix} h_3 \\ h_4 \end{bmatrix}$$

$$p_{A+B}(v,h) = \frac{1}{Z_{A+B}} e^{-E_{A+B}(v,h)} \propto p_A(v_A,h_A) p_B(v_B,h_B)$$

Because of the probabilities approximately multiplying, we can also say that if each of the distributions differed from the "ideal" distribution (denoted here by q), then we can expect the error (as measured by the KL divergence) to increase approximately linearly with the number of connections. This shows that the error should increase with the number of connected units, rather than directly with the number of bits. This is especially important with combinatorial optimization problems where the deterministic algorithm run time increases exponentially with the number of bits. However, we note that as the number of connections between the units increases, the KL divergence increasingly diverges from linear.

$$D_{\mathrm{KL}}(q\|p) \approx D_{\mathrm{KL}}(q_A\|p_A) + D_{\mathrm{KL}}(q_B\|p_B);$$

$$p = p_A(v_A,h_A)p_B(v_B,h_B), \quad q = q_A(v_A,h_A)q_B(v_B,h_B)$$

Multiplying distributions like this ensures that the combined distribution retains the sharp structures of the original distribution. In addition, due to the nature of the multiplication, we know exactly where the modes of the multiplied distribution will be, and it is possible to make a deterministic estimation of the shape of the final distribution. Those states corresponding to spurious modes in the distribution also get depressed as they correspond to multiple low probability states getting multiplied together. However, careful consideration must be taken to modes created by high probability states combining with low probability states, as these can lead to spurious modes causing further samples being needed to perform proper inference. To prevent this, training must be done carefully to ensure that the model distribution properly reflects the data during training.

To solve the actual optimization problem, the visible units are partially clamped to data, and the remaining units are sampled via Gibbs sampling steps starting from the partial data distribution. This gives the most likely estimate for the RBM distribution given the partial data distribution. Given enough sampling steps, the sampled distribution is guaranteed to converge to the model distribution, whose mode is expected to be the correct answer as described above.

3.1 MCMC & Mixing Time

As sampling from the full distribution of an RBM is intractable due to the the partition function, a Markov chain Monte Carlo based technique is used to generate samples. Due to the bipartite graph nature of RBMs, Gibbs sampling is very efficient and can be done in 2 steps.

Convergence to a solution in this scheme is not generally based on the size of the problem, but rather on the convergence of the Markov chain Monte Carlo (MCMC) algorithm, which is measured by the mixing time. Although the Markov chain is guaranteed to converge to the target distribution and find the most likely estimate, in many cases it does not do so in a computationally efficient manner. There are many things that can effect the mixing time of the Markov chain, such as the magnitude of the weight matrices, the modality of the phase space, and the sparsity of the weight matrix.

3.1.1 WEIGHT MATRIX MAGNITUDE

Typically large magnitude in the weight matrices corresponds to higher probabilities assigned to states from the data distribution. However, it is well known that as the magnitude of the weight matrix increases, the MCMC mixes more slowly during inference and sampling (Hinton, 2002; Tieleman, 2008).This is especially a problem when sampling from large combined matrices as the magnitude of the weight matrix needs to be controlled for reasonable mixing times. The magnitude of the weights can be controlled by using finite weight decay during training, allowing for finite probability in the spurious modes making mixing between states easier. (Hinton, 2002).

3.1.2 STATE SPACE & MODEL MODALITY

Many combinatorial optimization problems are highly multimodal, which can lead to problems of getting stuck in local minima and spurious modes during MCMC. This problem can manifest itself in a large combined structure, where many of the individually trained units are in a high probability states and only a few are in low probability states, but mixing between such a state and a higher probability overall state would only occur if all of the units were to change states. This can be equivalently thought of as a traversal between two modes of state space that are separated by a large set of states of low probability. The traversal from one of these modes to another goes as p^n where p is the probability of transition between states and n is the number of transitions needed to go from one mode to another. As the distribution becomes increasingly multimodal, pseudo-convergence of the MCMC becomes a problem.

3.1.3 SPARSITY

A very sparse weight matrix (as is pictured in Figure 2) can cause a relatively sparse 1 step transition probability matrix while running MCMC. This means that for visible units on the either end of the model to mix, they would have to cross a large number of intermediate states. This can be seen in figure 2 where in the 16 bit adder composed of multiple one 1 bit adders, there is no hidden unit directly connecting the first full adder to the last full adder. Thus, for information about the change of state in the first full adder must pass through all of the individual adders before it can manifest itself as a change in the last full adder. This means additional gibbs sampling steps must be taken, causing a slower mixing rate for the RBM during inference. This effect is partially mitigated by the fact that very sparse matrices can be more computationally efficient for matrix multiplications, causing the actual computation time to be less.

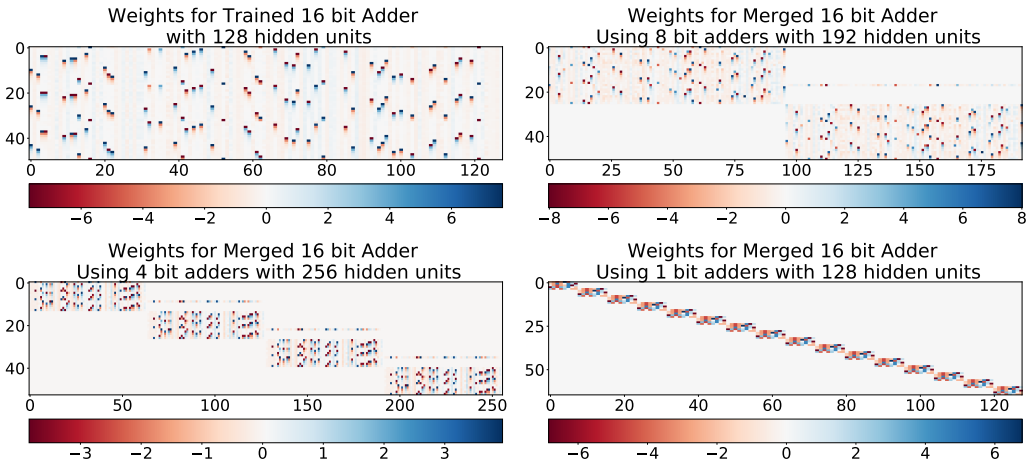

Figure 2: Weights matrix for a 16 bit adder circuit created by directly training the device (top left), combination of 8 bit units (top right), combinations of 4 bit units (bottom left) and combination of 1 bit units (bottom right). The sparsity can cause matrix multiplication to be more efficient, but can also lead to poor mixing between states.

3.2 TRAINING VS. SETTING MODEL PARAMETERS

If we are aware of the exact distribution we are trying to model, we can also directly set the weight matrix and biases for the RBM instead of training using samples from this exact distribution. As the RBM is a universal function approximator, we are guaranteed to have the correct distribution for an arbitrary sized model. Using the method outlined in the proof of universal function approximation outlined in (Le Roux & Bengio, 2008) we can create an "ideal" RBM for this problem, and merge copies of these ideal RBMs together.

There are a few problems with this method, most notably that directly calculated RBMs are not compact function approximators. To exactly model these distributions, 1 hidden unit is needed to approximate every data vector meaning the number of hidden units would scale as 2^n for multiplier and adder units. This type of scaling is not expected for most highly structured problems, as many combinatorial optimization tasks are. The directly calculated RBM is more susceptible to being trapped in local minima than a trained RBM is due to poor mixing between modes. This is because they have a uniformly low probability in states that are not exactly part of the data distribution as described above. This is also seen in Figure 4 where the directly calculated unit has a slower mixing rate, owing to the slower decay to an autocorrelation of 0.

4 EXPERIMENTS

To experimentally demonstrate the viability of this method of combination, we have used it to create "invertible boolean logic". With this, we are able to create logical units of varying sizes, and combine them to create adders and multipliers. Using the property of RBMs that they can partially fill in visible units conditioned on the values of other units, we can use these adder RBMs to solve addition, subtraction, reverse sum carry, and combine these adders with multiplier RBMs to solve multiplication, division, and factorization tasks. Here, we are able to use a circuit that is traditionally used to multiply numbers to also divide, factorize, and solve any problem involving partial multiplication and division, as we are just sampling from the joint distribution of variables over multiplication and conditioning over variables we are interested in.

Performance on these various tasks was characterized by using Gibbs sampling, and taking samples from the distribution conditioned on partial inputs. After a number of samples, we check the mode of the distribution against the expected minimal energy state. In Figure 6 we see that as sampled distribution approaches its stationary distribution it becomes more likely to correctly identify the minimal energy state (highest probability) and solve the combinatorial optimization problem.

If the models are trained such that the highest probability states correspond to the data distribution without spurious modes, the Markov chain for the merged model will always converge to the state corresponding to the combination of correct states of its individual parts. However, this requires full convergence of the MCMC algorithm, which could easily become an extremely time consuming exercise. Empirically, we see that adders seem to converge much quicker to the correct distribution than multipliers do.

This merging method is validated by the equilibrium distributions of a merged adder vs. a trained adder in Figure 3. Although the equilibrium distributions are very similar, the two distributions mix very differently. The trained full adder mixes rapidly between modes while the full adder composed of merged logical units is more likely to get get stuck in a one mode and reach a pseudo-convergence condition. This condition can be mitigated by starting multiple chains from various starting locations in the state space and combining their distributions using the mulistart heuristic (Brooks et al., 2011). However, it is better practice to design the weight matrix to have better mixing properties and to use more advanced sampling techniques, as we cannot be guaranteed about convergence properties when using larger networks.

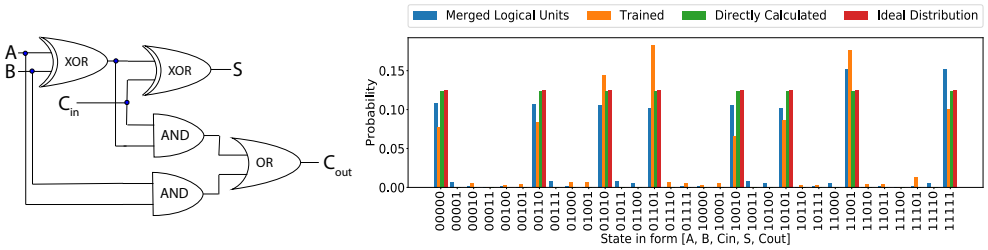

Figure 3: Probability distributions calculated for a trained full adder, as well as a full adder composed of merged logical units as on the left using the method in Section 3. The comparison shows that merging units can approach similar performance to a fully trained unit, with similar error levels even with a reasonable number of merged units. These probabilities were directly calculated and normalized by the exact partition function (no gibbs sampling was done). We also show directly calculated (as described in Le Roux & Bengio (2008)) units have the distribution closest to the ideal distribution.

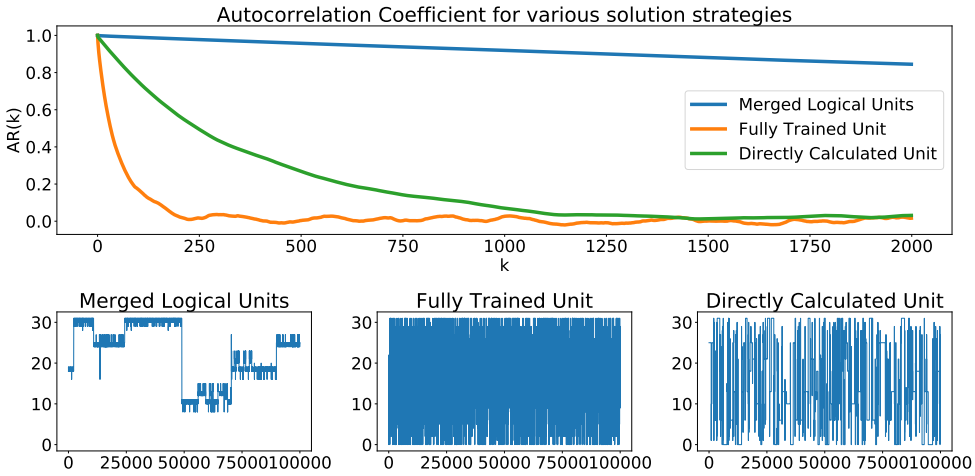

Figure 4: Autocorrelation coefficients (above) and time series (below) of MCMC algorithm comparison. All three units are attempting to model the behavior of a full adder circuit using either Merged Logical Units, a fully trained unit, or a directly calculated unit. The probability distribution modeled by all of these units is seen in Figure 3. We can see that the mixing time for a fully trained unit is much faster than that of both a directly calculated unit and of merged logical units as the autocorrelation coefficient quickly decays to 0 for this case. From the time series analysis below we can also see that the merged logical units take a longer time to visit the entirety of the state space, as is expected by the autocorrelation coefficients.

4.1 ADDERS

Multi bit adders were created using the combination scheme outlined in Figure 5. From the data based on trained and merged 64 bit adders, we are unable to improve performance by training larger individual units. One of the biggest issues is the exponential growth in the size of the training space (and time) needed to effectively train larger units. As the training set grows as 2^n, training on the entirety of an addition dataset becomes intractable at 16 bit adders, where a training set that would encompass the entire space would contain $2^{33} \approx 8$ billion training examples. Even though we expect these models to generalize effectively, training a good model becomes increasingly difficult as the dataset grows. This can be seen in greater degree in the 32 bit adder unit, whose performance is significantly worse than merged units even after training the model for over a week. The authors attempted to train a 64 bit adder unit but were unsuccessful in reducing the error by any reasonable

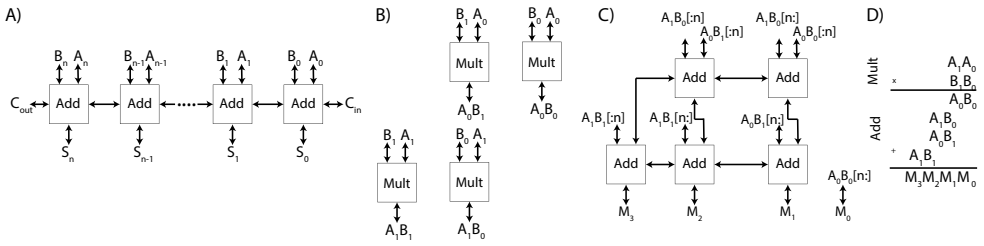

Figure 5: Diagram of the creation of larger logical units by combining smaller individual units. We can create a n-bit adder can be created by cascading n copies of a 1 bit adder (or $n/2$ copies of a 2 bit adder, etc.) as in A). To create a multiplier, we divide n bit multiplication as described in D) into n/2 bit multiplication by dividing the problem into multiplication as in B) along with addition of partial sums, as in C). This can also be extended to create n-bit multiplication using smaller units.

amount.

We note the relative success of merging addition models in comparison to multiplication models. This can be explained by the fact that addition models are less prone to getting stuck in local minima, as each of the adders is clamped to a piece of the data in all modes of operation (addition, subtraction and reverse sum carry). The 64 bit adder unit can explore a state space of $2^{64} \approx 2 \times 10^{19}$ states, and find the exact answer within only $\approx 10^3$ samples, showing a remarkably fast mixing rate.

4.2 MULTIPLIERS

Multi bit multipliers were created by the combination scheme outlined in Figure 5, similar to Adders. Multipliers require heterogeneous integration of different components (adders and multipliers), which means that the magnitude of multiple different weight matrices need to be matched to get good mixing and to ensure there are minimal spurious modes.

The design of bitwise multipliers faces more challenging issues than that of adders. The state space of multipliers is much sparser, tends to have more spurious modes, is larger for the same number of bits. For an adder, the state space is 2^{3n+2} states ($2 \times n$ inputs, n output, 1 C_{in}, 1 C_{out}) with 2^{2n+1} of those states representing valid answers to the addition problem. However, the multiplier has a state space that is 2^{4n} ($2 \times n$ inputs, $2 \times n$ outputs) and 2^{2n} of them are valid answers. This means that an 8 bit multiplier has a state space that is 128 times sparser than an 8 bit adder, and that the sparsity of an adder scales as $\approx 2^n$ while the sparsity of a multiplier scales as $\approx 2^{2n}$. This increased sparsity makes mixing between modes more difficult, as there are larger areas of low probability space in between modes.

Multipliers have a more complex state space that leads them to get stuck in spurious modes. This is due to the fact that not all units in the multiplier are clamped to pieces of the data distribution, and intermediate units have many equivalent local minima. Because of this, individual units may be in local minima where their individual constraints are satisfied (i.e. an adder satisfies the condition that its inputs add to its outputs, or a multiplier satisfies the constraints that the output is equal to the two inputs are multiplied together), but the global constraint that the output is equal to the inputs multiplied together is unsatisfied. This can be because one of the units is in an incorrect state, or the local minima of the other units corresponds to an unsatisfiable constraint on one of the units (i.e. an adder that is forced into a state where the numbers cannot add up to one another, or a multiplier that is forced to factorize a number whose factors are outside the input bit range).

Even with the complexities of multipliers, we have shown that an 16 bit multiplier created from trained and merged is able to outperform a multiplier created just by training, as can be seen in Figure 6. Currently, the number of samples needed to compute the factors is not competitive with a

direct search. However, we believe that this situation will be significantly improved by engineering of the mixing rate of the MCMC algorithm, and using more advanced sampling techniques (as mentioned below in Section 5).

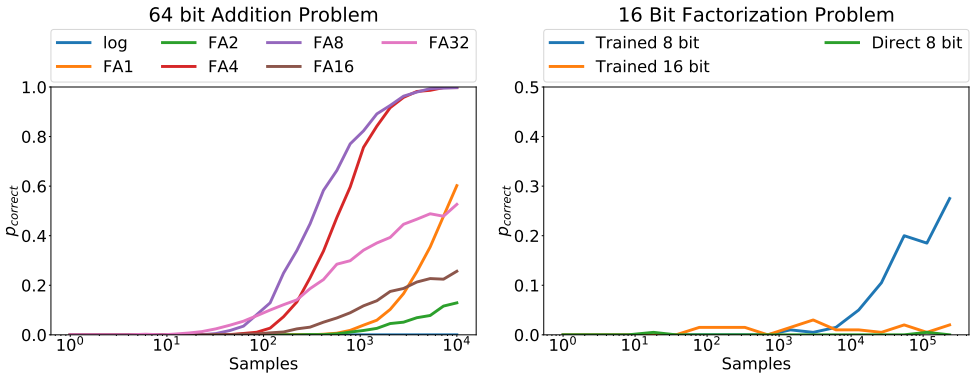

Figure 6: In both the factorization and addition problems, as we approach higher bit problems it becomes increasingly difficult to fully train a module. At 8 bits for factorization (8 bit factors, 16 bit number) and 64 bits for addition (64 bit inputs), it becomes clear that merged logical units can outperform fully trained units. There is a clear trade off between the ability to train the model and number of samples needed to perform accurate inference.

5 Conclusion & Future Work

In this work we have shown the viability of merging RBMs of individually trained distributions and using this merged RBM to solve combinatorial optimization tasks. Although mixing in these merged RBMs is a problem, these merged representations can outperform fully trained structures on tasks where training is difficult. We have used factorization as an example problem to validate this technique. However, the approach is generally applicable to a variety of other combinatorial optimization tasks.

One of the biggest challenges in making this approach viable as a stochastic optimization algorithm is to improve the mixing rate of the merged RBMs. In addition to the weight decay approaches already implemented in this work, significant improvement in this regard is expected from using advanced sampling techniques such as tempered transitions (Desjardins et al., 2010), fast weight decay (Tieleman & Hinton, 2009), adaptive MCMC (Salakhutdinov, 2010), annealed importance sampling (Neal, 2001), parallel tempering (Cho et al., 2010), or using a different transition operation for the RBM (Brgge et al., 2013). It will also be necessary to develop mathematical models that can describe the differences in the mixing rate of the merged model as compared to that of the constituent units.

RBMs have been used for classical combinatorial optimization problems such as the travelling salesman (Shim et al., 2012; Aarts & Korst, 1989). The method presented here is generally applicable to all such optimization tasks. As a specific example, a natural application for the adders, that we used as an example case, could be to solve 3SUM and subset sum problems . There have already been initial results on these problems using cascaded, fully connected Boltzmann Machines (Hassan et al., 2018).

Acknowledgments

This work was funded through the ASCENT center, one of the six centers within the DARPA/SRC JUMP initiative.

REFERENCES

Emile HL Aarts and Jan HM Korst. Boltzmann machines for travelling salesman problems. *European Journal of Operational Research*, 39(1):79–95, 1989.

DAVID H. Ackley, GEOFFREY E. Hinton, and TERRENCE J. Sejnowski. A Learning Algorithm for Boltzmann Machines*. In *Readings in Computer Vision*, pp. 522–533. Morgan Kaufmann, San Francisco (CA), 1987. ISBN 978-0-08-051581-6. doi: 10.1016/B978-0-08-051581-6.50053-2.

Guillaume Alain, Yoshua Bengio, Li Yao, Jason Yosinski, Eric Thibodeau-Laufer, Saizheng Zhang, and Pascal Vincent. GSNs : Generative Stochastic Networks. *arXiv:1503.05571 [cs]*, March 2015. arXiv: 1503.05571.

F. Barahona. On the computational complexity of Ising spin glass models. *Journal of Physics A: Mathematical and General*, 15(10):3241, 1982. ISSN 0305-4470. doi: 10.1088/0305-4470/15/10/028.

Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. *Handbook of markov chain monte carlo*. CRC press, 2011.

Kai Brgge, Asja Fischer, and Christian Igel. The flip-the-state transition operator for restricted Boltzmann machines. *Machine Learning*, 93(1):53–69, October 2013. ISSN 0885-6125, 1573-0565. doi: 10.1007/s10994-013-5390-3.

Kerem Yunus Camsari, Rafatul Faria, Brian M. Sutton, and Supriyo Datta. Stochastic p-Bits for Invertible Logic. *Physical Review X*, 7(3):031014, July 2017. doi: 10.1103/PhysRevX.7.031014.

K. Cho, T. Raiko, and A. Ilin. Parallel tempering is efficient for learning restricted Boltzmann machines. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, July 2010. doi: 10.1109/IJCNN.2010.5596837.

Guillaume Desjardins, Aaron Courville, Yoshua Bengio, Pascal Vincent, and Olivier Delalleau. Tempered Markov chain Monte Carlo for training of restricted Boltzmann machines. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 145–152, 2010.

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

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Orchi Hassan, Kerem Y Camsari, and Supriyo Datta. Voltage-driven building block for hardware belief networks. *arXiv preprint arXiv:1801.09026*, 2018.

Geoffrey E. Hinton. Training Products of Experts by Minimizing Contrastive Divergence. *Neural Computation*, 14(8):1771–1800, August 2002. ISSN 0899-7667, 1530-888X. doi: 10.1162/089976602760128018.

Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

Jan HM Korst and Emile HL Aarts. Combinatorial optimization on a boltzmann machine. *Journal of Parallel and distributed computing*, 6(2):331–357, 1989.

Nicolas Le Roux and Yoshua Bengio. Representational power of restricted Boltzmann machines and deep belief networks. *Neural computation*, 20(6):1631–1649, 2008.

Andrew Lucas. Ising formulations of many NP problems. *Frontiers in Physics*, 2, 2014. ISSN 2296-424X. doi: 10.3389/fphy.2014.00005.

Radford M. Neal. Annealed importance sampling. *Statistics and computing*, 11(2):125–139, 2001. ISSN 1573-1375. doi: 10.1023/A:1008923215028.

MarcAurelio Ranzato, Alex Krizhevsky, and Geoffrey Hinton. Factored 3-Way Restricted Boltzmann Machines For Modeling Natural Images. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 621–628, March 2010.

Jrmie Roland and Nicolas J. Cerf. Quantum search by local adiabatic evolution. *Physical Review A*, 65(4):042308, March 2002. doi: 10.1103/PhysRevA.65.042308.

Ruslan Salakhutdinov. Learning deep Boltzmann machines using adaptive MCMC. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 943–950, 2010.

Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted Boltzmann Machines for Collaborative Filtering. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pp. 791–798, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-793-3. doi: 10.1145/1273496.1273596.

Vui Ann Shim, Kay Chen Tan, and Kok Kiong Tan. A hybrid estimation of distribution algorithm for solving the multi-objective multiple traveling salesman problem. In *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pp. 1–8. IEEE, 2012.

Tijmen Tieleman. Training restricted Boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th international conference on Machine learning*, pp. 1064–1071. ACM, 2008.

Tijmen Tieleman and Geoffrey Hinton. Using fast weights to improve persistent contrastive divergence. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 1033–1040. ACM Press, 2009. ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553506.

Fabio L. Traversa and Massimiliano Di Ventra. Polynomial-time solution of prime factorization and NP-hard problems with digital memcomputing machines. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 27(2):023107, February 2017. ISSN 1054-1500, 1089-7682. doi: 10.1063/1.4975761. arXiv: 1512.05064.

J. D. Whitfield, M. Faccin, and J. D. Biamonte. Ground-state spin logic. *EPL (Europhysics Letters)*, 99(5):57004, 2012. ISSN 0295-5075. doi: 10.1209/0295-5075/99/57004.

Nanyang Xu, Jing Zhu, Dawei Lu, Xianyi Zhou, Xinhua Peng, and Jiangfeng Du. Quantum Factorization of 143 on a Dipolar-Coupling Nuclear Magnetic Resonance System. *Physical Review Letters*, 108(13):130501, March 2012. doi: 10.1103/PhysRevLett.108.130501.

Wojciech Zaremba and Ilya Sutskever. Learning to execute. *arXiv preprint arXiv:1410.4615*, 2014.

6 APPENDIX

6.1 TRAINING

In this paper we trained the RBMs by contrastive divergence as described by Hinton (2002). Each of the models in this paper were validated by checking their performance on the problem they were trying to solve (i.e addition and subtraction for an adder, multiplication and factorization for a multiplier). This method was also used to assess model complexity (i.e number of hidden units) and evaluate learning parameters (learning rate, batch size, etc.). The final results for RBM sizes are shown below.

Training was conducted on a computer with 2 Intel Xeon E5-2620 processors, and 2 Nvidia Titan V GPUs with 128Gb RAM. Each RBM was trained for 10 epochs, where an epoch was 4 copies of the full state space, with a learning rate of 1. After 10 epochs, the CDk of learning was increased to combat the worse mixing as the weight matrix increases, with an initial CDk of 2. This process was repeated until the test stopped decreasing. In the case that the state space was too large to train on the full state space (such as the 16 bit and 32 bit adders), a random sample of the training set was used, and a new random sample was reinitialized every epoch of training.

Table 1: Trained RBM Parameters

RBM	Hidden Units	Training Time (minutes)
1 bit Adder	6	1
2 bit Adder	28	13.5
4 bit Adder	64	133
8 bit Adder	96	201
16 bit Adder	128	321
32 bit Adder	192	13000 (approx)
1 bit Multiplier	4	1
2 bit Multiplier	12	46
4 bit Multiplier	64	655
8 bit Multiplier	96	2794

The training time tends to increase with the number of bits in the adder or multiplier due to both the size of the data set (which increases exponentially with the number of bits) and the number of hidden and visible units (which both increase approximately linearly). The slower increase of training time for adders after the 4 bit adder is due to the usage of GPUs to increase the parallelism.

At the 16 bit adder level, the size of the data set was so large that the entire data set could not be used for training ($\approx$ 8 billion data points) and a randomized sample of the set had to be taken. As generalization is not perfect, we can attribute the decrease in their performance (as described in Figures 6 8 7) to this fact. For the 32 bit adder, this problem was exacerbated, and the 32 bit adder was outperformed by most units even after training for a full week.

For multipliers, the 8 bit multiplier has a tractable amount of data, but a good joint density model could not be formed even after a large training time. We believe this is due to an inherent difficult in the multiplication problem that is not present in the addition problem. As there is not as distinct of a correlation between higher level bits in the 8 bit multiplication problem as there is in the addition problem, first level correlations (as an RBM with 1 layer of hidden units would find) are more difficult to find. We believe that using deep boltzmann machines might help fix the problem of training in large multipliers.

6.2 Further Data

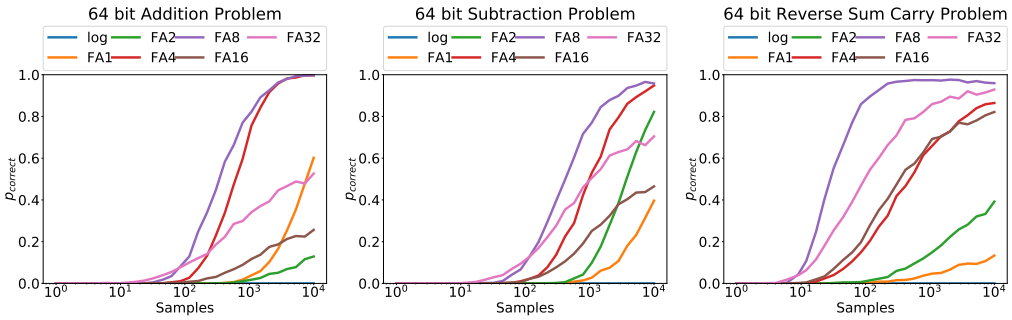

Figure 7: A comparison of probability of being correct vs. number of samples taken for a 64 bit adder unit composed of smaller merged units. It is clear from these figures that smaller bit units that can be trained better outperform larger, harder to train units.

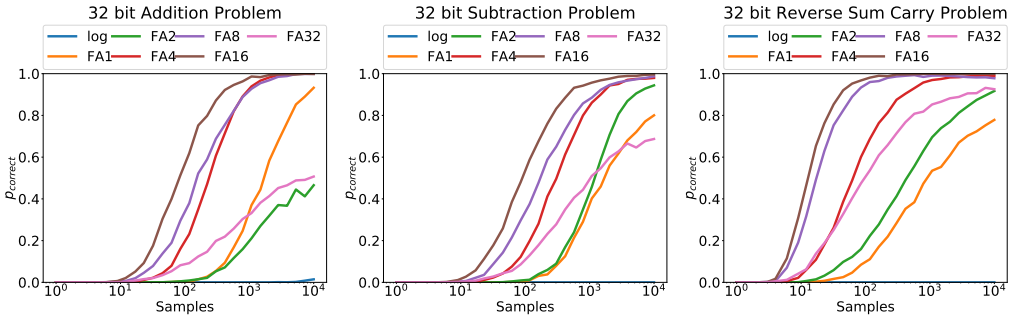

Figure 8: A comparison of probability of being correct vs. number of samples taken for a 32 bit adder unit composed of smaller merged units compared to a fully trained model. The joint density model for the 32 bit adder is an intractable training problem due to the size of the data set (2^{66} datapoints).

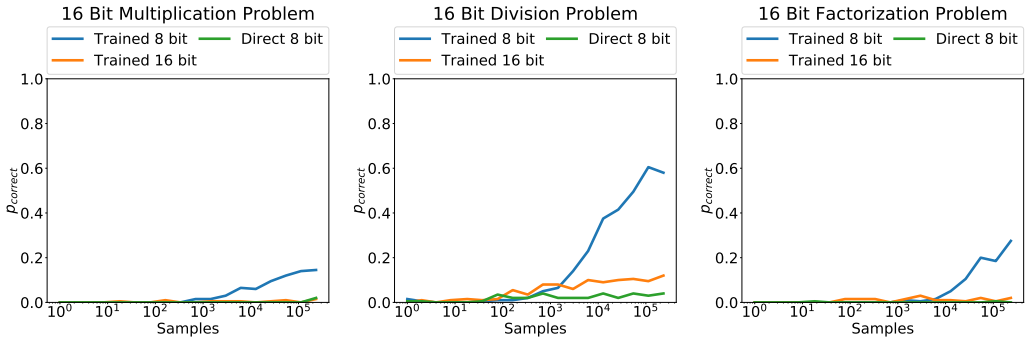

Figure 9: A comparison of probability of being correct vs. number of samples for an 16 bit multiplier unit. Here we compare using a fully trained unit, vs. composing it of 8 bit trained multiplier and adder units, vs. composing it with 8 bit directly calculated unit. The trained unit outperforms both of these units, even after extensive training of the 16 bit multiplier unit (see above 6.1)

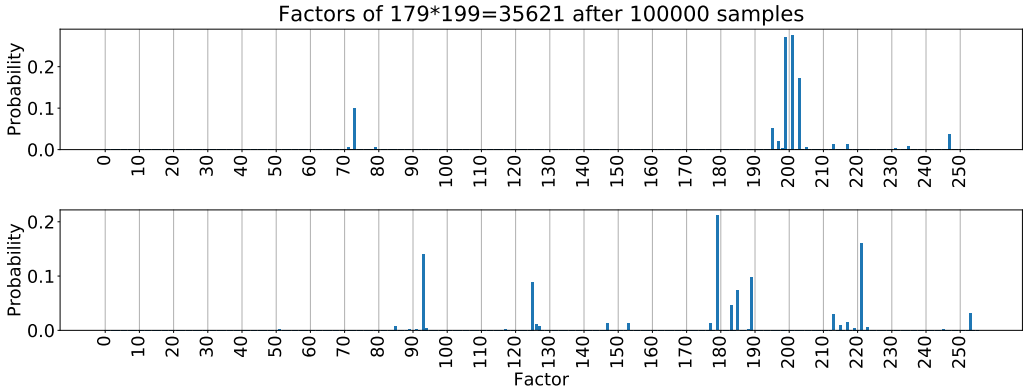

Figure 10: An example output from a factorization problem for the merged 8 bit multiplier formed from 4 bit units. This is an example of the probability distribution generated by MCMC after 100000 samples. In this example, we are trying to find the coprimes that multiply to 35621. The factors it should find are 179 and 199, which it correctly identifies as the most likely factors.

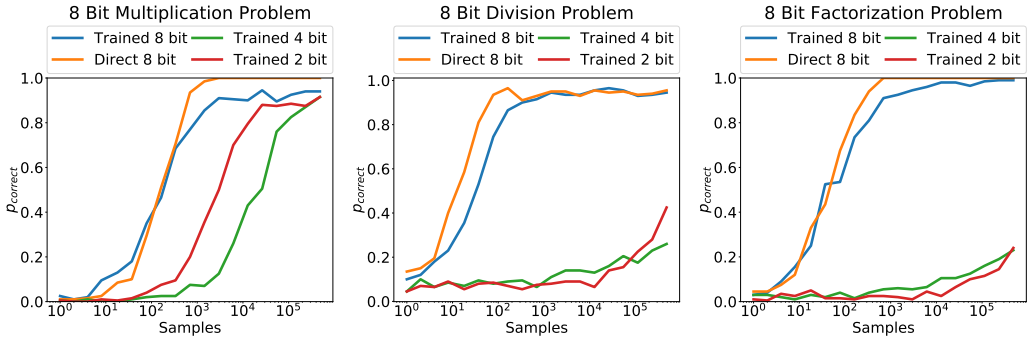

Figure 11: A comparison of probability of being correct vs. number of samples for an 8 bit multiplier unit. Training of an 8 bit multiplier is a tractable problem, and we can see that both the trained and the directly calculated models outperform the merged models. This can be understood as a consequence of worse mixing present in the merged models. (see above 6.1)