A Study of Complex Deep Learning Networks on High Performance, Neuromorphic, and Quantum Computers

Thomas E. Potok, Catherine D. Schuman, Steven R. Young, Robert M. Patton
Oak Ridge National Laboratory
1 Bethel Valley Road
Email: potokte@ornl.gov

Federico Spedalieri, Jeremy Liu, Ke-Thia Yao
USC Information Sciences Institute
Marina del Rey, CA, USA
Email: fspedali@isi.edu

Garrett Rose, Gangotree Chakma
University of Tennessee
Knoxville, TN, USA
Email: garose@utk.edu

***Abstract*—Current Deep Learning models use highly optimized convolutional neural networks (CNN) trained on large graphical processing units (GPU)-based computers with a fairly simple layered network topology, i.e., highly connected layers, without intra-layer connections. Complex topologies have been proposed, but are intractable to train on current systems. Building the topologies of the deep learning network requires hand tuning, and implementing the network in hardware is expensive in both cost and power.**

In this paper, we evaluate deep learning models using three different computing architectures to address these problems: quantum computing to train complex topologies, high performance computing (HPC) to automatically determine network topology, and neuromorphic computing for a low-power hardware implementation. Due to input size limitations of current quantum computers we use the MNIST dataset for our evaluation.

The results show the possibility of using the three architectures in tandem to explore complex deep learning networks that are untrainable using a von Neumann architecture. We show that a quantum computer can find high quality values of intra-layer connections and weights, while yielding a tractable time result as the complexity of the network increases; a high performance computer can find optimal layer-based topologies; and a neuromorphic computer can represent the complex topology and weights derived from the other architectures in low power memristive hardware.

This represents a new capability that is not feasible with current von Neumann architecture. It potentially enables the ability to solve very complicated problems unsolvable with current computing technologies.

1. Introduction

Deep Learning is inspired by the networks of neurons in the visual cortex of the brain. A significant limitation of the deep learning approach has been the computational time required to train or optimally set the weights within a deep learning network. Graphical processing units have provided a significant speedup in training, due to their ability to perform multiple simple network weight calculations in parallel, which allows for larger and more complex networks to be studied. One of the limitations of this approach is the limited complexity of the neuron models within these networks. While very successful in solving challenging classification problems, the scale and complexity of the neuron models are relatively modest and not comparable to those produced by nature.

There are multiple designs for deep learning networks, but the two that we will focus on in this study are convolutional neural networks and Boltzmann Machines (BMs).

Currently, convolutional neural networks (CNNs) are the most widely used deep learning models [1] and are most commonly used for image classification. They utilize stochastic gradient descent and backpropagation for training in a supervised manner. The defining feature of convolutional networks are their convolutional layers consisting of multiple sets of weights, or kernels, that are convolved with their input to form a set of feature maps.

A Boltzmann Machine (BM) is a recurrent neural network consisting of neurons that make binary stochastic decisions based on the states of their symmetrically connected neuron neighbors [2]. Boltzmann Machines are well-suited for solving constraint satisfaction tasks with many weak constraints. These tasks include digit recognition, object recognition, compression/coding and natural language processing. The training of a BM is impractical given the complex topology, which has given rise to the development of a Restricted Boltzmann Machine (RBM). The RBM network topology is restricted to a bipartite graph [3].

To explore the capabilities of complex deep learning networks, we need to look beyond traditional von Neumann computers. In this paper we examine three different computer architectures to explore this issue: quantum computing,

978-1-5090-3882-4/16 $31.00 © 2016 IEEE
DOI 10.1109/MLHPC.2016.9

high performance computing, and neuromorphic computing. While the CNN approach has had great success, it appears that the BM approach may provide the ability to create a more complex deep learning network. A quantum computer has the ability to sample a probability distribution of a network of a BM, and may provide a feasible approach to training a BM network. Automatically determining an optimal topology of this network is an open problem, but using high performance computing, the topology can be optimized using evolutionary algorithm to evolve a high performing network. Using this trained network to classify data typically requires a central cluster of GPU. Neuromorphic computing has the potential to implement these networks in distributed, low-power hardware.

2. Related Work

There are three main challenges in Deep Learning. The first is how to train models with complex topologies that are closer representations of nature. The second challenge is how to automatically configure a network to an optimal or near optimal topology. The last challenge is how to implement such a model in hardware.

2.1. Quantum Computing

Computing using quantum computers was first discussed by Feynman [4] who was motivated by the fact that simulating a quantum system using a classical computer seems to be intractable. Interest in quantum computing increased dramatically with the discovery of the Shor's polynomial quantum algorithm for factoring numbers [5], because all known classical probabilistic factoring algorithms require exponential time. Several approaches to quantum computing were since developed, and they include the well-known quantum circuit model (used by Shor's algorithm), the measurement-based quantum computing model, and the adiabatic quantum computing model [6]. All three have been shown to have the same computational power. In this paper, we focus on a restricted form of the adiabatic quantum computing model which performs adiabatic quantum optimization (AQO) to find the minimum energy state of an Ising Hamiltonian system. Actual implementations of adiabatic quantum machines, such as the D-Wave, operate at a finite temperature [7]. The output of these machines is a sequence of samples from a probability distribution defined by the Ising Hamiltonian system. The ability to draw samples from complex probability distributions is at the core of probabilistic deep learning approaches, like the BM. As stated above, the training of a BM is impractical, thus the development of a RBM restricts the network topology to that of a bipartite graphs [3]. Bipartite graphs allow for techniques like contrastive divergence to efficiently draw samples in linear time from the probability distribution defined by the BM. Sampling is a fundamental building block and part of the inner loop of the Boltzmann learning algorithm. Without the bipartite restriction, sampling a BM with a general topology is a NP-hard problem. However, adiabatic quantum machines, like the D-Wave, have the potential to efficiently sample a richer set of graph topologies that are supersets of bipartite graphs.

2.2. High Performance Computing

Deep learning, being an early adopter of GPU technology, has benefited greatly from the speedup offered by these accelerated computing devices and has received great support from device manufacturers in the form of deep learning-specific GPU libraries. General purpose GPUs are the basic building blocks of today's HPC platforms and next generation machines will rely on them to an even greater degree. Thus, deep learning provides a great opportunity to fully utilize these machines, as they will have multiple GPUs per compute node. This leaves the question of how to best utilize thousands of GPUs for deep learning, as previous work has only utilized a maximum of 64 GPUs before encountering scaling problems when trying to exploit model parallelism to spread the weights of the network across multiple GPUs [8]. HPC provides the unique opportunity to address the problem of network specification. This refers to the problem of deciding upon the set of hyper-parameters needed to specify the network and training procedure in order to apply deep learning to a new dataset.

For convolutional neural networks, this could involve specifying parameters such as the number of layers, the number of hidden units, or the kernel size. For more general networks, such as RBMs, this could involve defining much more complicated connectivity between neurons.

Previously, it has been shown that HPC can be utilized to optimize the hyper-parameters of a deep learning network [9]. This work utilized an evolutionary algorithm distributed across the nodes of Oak Ridge National Laboratory's (ORNL's) Titan supercomputer in order to optimize the performance of deep learning algorithms. Hyper-parameters in deep learning refer to the model parameters, i.e., the activation function used, the number of hidden units in a layer, the kernel size of a convolutional layer, and the learning rate of the solver. As the size of the network grows, the hyper-parameter space grows increasingly larger. The size of deep learning networks used today have resulted in a hyper-parameter space that cannot be searched on a single machine or a small cluster. This is a result of the computational complexity of training and evaluating these networks. Without utilizing the computational capabilities provided by supercomputers, evaluating a sufficient number of hyper-parameter sets to search the enormous hyper-parameter space of these methods would be impossible.

2.3. Neuromorphic Computing

Neuromorphic computing architectures have historically been developed with one of two goals in mind: either developing custom hardware devices to accurately simulate biological neural systems with the goal of studying biological brains or building computationally useful architectures that are inspired by the operation of biological brains and have some of their characteristics. In developing neuromorphic

computing devices for computational purposes, there have been two main approaches: building devices based on spiking neural networks (SNNs), such as IBM's TrueNorth [10] or Darwin [11], and building devices based on convolutional neural networks, such as Google's Tensor Processing Unit [12] or Nervana's Nervana Engine [13], to serve as deep learning accelerators.

The neuromorphic devices that have been built based on SNNs or built to simulate biologically-accurate systems have vastly different characteristics than those that have been built based on deep learning networks, such as CNNs or RBMs. The neurons in SNN-based systems are typically not organized in layers, and there are fewer restrictions on connectivity between neurons. The neuron and synapse models also differ from those in convolutional neural networks. In SNN-based neuromorphic systems, the neuron is typically some form of spiking neuron, such as a leaky-integrate-and-fire neuron, and the synapses have a delay value in addition to a weight value, thus introducing a temporal component to the processing of the network.

The primary computational issue associated with SNN-based systems is that very few algorithms that train native networks for those systems have been developed. Two of the key reasons why algorithms have not been developed are the computational difficulty introduced by broader connectivity and the computational difficulty introduced by the inclusion of the temporal component in both the neurons and synapses. In fact, one approach for training networks for neuromorphic computers has been to train a CNN offline and then create a mapping process from the CNN to the associated SNN-based neuromorphic hardware [14].

One of the key properties of neuromorphic systems is their potential for more energy-efficient computation. To achieve energy-efficiency, we (and many others) have explored an implementation of a spiking neural network system utilizing memristors. Memristors are one of the four fundamental circuit elements. They are "memory resistors" in that their resistance can be altered depending on the magnitude of the voltage applied. Likewise, when no voltage is applied across a memristor, the most recent resistance value is retained [15]. Memristors have similar behavior to biological synapses, and as such, have been frequently utilized to implement neuromorphic systems [16], [17], [18].

3. Approach

We began our investigation by evaluating a simple classification problem using existing DL networks on a quantum computer, an HPC platform, and a neuromorphic system in order to understand the differences offered by each approach, and also to project the potential advantages and disadvantages of each computer platform on a deep learning problem. These three architectures are quite different in many ways and run at significantly different scales. Our goal is not to do a performance versus accuracy comparison of the three platforms, but rather to look for the potential advantages of each architecture on complex deep learning problems. The limiting factor of the three architectures is the quantum computer. It has roughly 1,000 qubits, and thus limits the size of the problem that can be addressed.

We chose the MNIST dataset as the DL problem to explore on these three architectures. MNIST is a collection of hand-written digits that has been very widely studied in the deep learning community [19]. The images of the digits are very small (28 X 28 pixels totaling 784 pixels) that can be analyzed using a quantum computer as well as the other architectures. Figure 1 shows a notional diagram of the deep learning networks that will be applied to each of the different architectures.

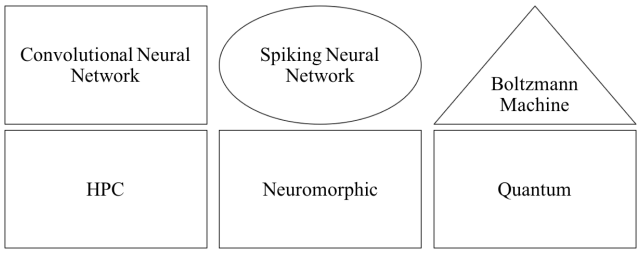

Figure 1. The native deep learning methods for each hardware platform.

3.1. Quantum

The quantum computer we are using is a D-Wave adiabatic quantum computer located at the University of Southern California Lockheed Martin Quantum Computing Center. The approach we propose to represent the MNIST problem is to use a network of BMs. As discussed in Section 2.1 a deep learning network of BM has been previously proposed (Deep Boltzmann Machine); however, learning is intractable for a BM with a fully connected topology, since computing expected values over the model requires computing sums over an exponentially large state space. Restricted Boltzman Machines were introduced to circumvent this issue by discarding couplings between nodes within the same layer. Removing intra-layer couplings introduces conditional independence between nodes within the same layer. This computational advantage comes at the cost of lower representational power. The D-Wave device provides an opportunity to test this approach. We propose to implement an experiment to test this hypothesis. First, we will create and train a RBM on D-Wave, applying it to the MNIST handwritten digit classification problem [19]. Next, we will consider a more complex topology that allows for some intra-layer connections between nodes. This semi-restricted BM might also be called a limited Boltzmann machine (LBM). We applied the LBM to the same MNIST digit classification problem on which we trained the RBM.

The LBM has two layers: one visible and one hidden. The visible layer is the same as in the RBM, with no intra-layer couplings. All the nodes of the visible layer are connected to all the nodes of the hidden layer. However, in contrast to the RBM, the LBM's hidden layer is allowed to have some intra-layer couplings. As a consequence, the

probability distribution of the hidden layer nodes no longer factorizes when the values of the visible layer nodes are fixed. To estimate the expected values required for the learning process, we used the D-Wave processor to generate samples of the hidden layer configuration and estimate probabilities.

3.2. HPC

The HPC computer we are using is the ORNL's Titan computer with roughly 300,000 cores, and 18,000 GPUs. This is currently the fastest open science computer in the world.

Clearly a supercomputer is not needed to solve the MNIST problem; however, a supercomputer is needed to automatically find an optimal deep learning topology for such a problem. Rather than using a trial-and-error method for finding a well performing network topology, we propose to use evolutionary optimization on Titan to evaluate tens of thousands of topologies; therefore, systematically finding the best performing networks on this problem. If achievable, this would solve one of the major challenges in building deep learning networks.

We use a CNN as our deep learning network since CNNs are currently producing the top results. We approach the network topology problem of selecting optimal hyper-parameters as a massive search problem, where Titan can be used to quickly search the space. Using an evolutionary algorithm, we represent each individual within the population of the evolutionary algorithm (EA) as a single deep neural network or CNN. An individual consists of a genome where the genes represent the various hyper-parameters that define the network topology, i.e., the number of layers, type of layers (convolution, pooling, etc.), and order of the layers. We then apply parameters defined in the genes of the individual to construct and train a deep learning network on the MNIST dataset. The results of the network's performance in testing are then used as the "fitness" of the individual in the EA population. After all the individuals in the population are evaluated, the individuals that performed better than others are selected to generate a new population of individuals that represent the next generation of the EA. Successive generations of individuals gradually improve in performance over time. This method is called Multi-node Evolutionary Neural Networks for Deep Learning (MENNDL).

To test this hypothesis, we used a simple EA that limits the search to the number of neurons per layer and the kernel size of convolutional layers. The network architecture utilized was LeNet [1]. We have shown that even with this widely studied MNIST dataset, better hyper-parameters can be found than those widely reported in the literature. An EA that can evolve the topology provides the opportunity to provide even greater results, and tackles more challenging datasets. Such an EA will also provide the opportunity to meaningfully utilize the entirety of Titan's capacity. It will provide challenging data management problems on a machine designed primarily for modeling and simulation, as opposed to these deep learning algorithms which require heavy amounts of data input in addition to heavy computation.

3.3. Neuromorphic

A SNN approach to the MNIST problem is not the ideal solution since there is not a temporal component in the task. To add a temporal component, we use a streaming scan of the digits as input to the SNN. The SNN learns to recognize digits based on this scan pattern. We then propose to evaluate the performance of this network on memristor hardware having the potential to enable a low power hardware implementation of a deep learning network, further, with the ability to represent spatial and temporal data. As noted in Section 2.3, there are not very many SNN training methods or training methods that can be applied to neuromorphic networks. To train both SNN models and neuromorphic networks, we utilize an EA approach to determine the structure (e.g., number of neurons and synapses and how they are connected) and parameters (e.g., weight values of synapses and threshold values of neurons).

The neuromorphic system we will use to explore the MNIST problem is a memristive implementation of the neuroscience-inspired dynamic architectures (NIDA) system [20]. The EA approach for training networks for the MNIST problem was previously applied to the NIDA SNN [20] and to a digital neuromorphic architecture based on NIDA called dynamic adaptive neural network array (DANNA), currently implemented on FPGA and with a VLSI implementation in progress [21]. For both NIDA and DANNA, an ensemble approach is utilized - each network in the ensemble is responsible for recognizing a particular digit type. For example, a network may be trained to recognize zeros, in which case the network will take the handwritten digit image as input and its output corresponds to either "yes, it is a zero" or "no, it is not a zero." Using this approach, ensembles that achieve around 90 percent accuracy for NIDA and around 80 percent accuracy for DANNA were created.

For this work, we extend the prior work by simulating a SNN implemented in memristive hardware in order to demonstrate the potential of significant power reductions for simulating the behavior of neural networks.

4. Results

4.1. Quantum

For our experiment the LBM has two layers, one visible and one hidden. The visible layer is the same as in the RBM, with no intra-layer couplings. All the nodes of the visible layer are connected to all the nodes of the hidden layer. However, in contrast to the RBM, the LBM's hidden layer is allowed to have some intra-layer couplings. As a consequence, the probability distribution of the hidden layer nodes no longer factorizes when the values of the visible layer nodes are fixed. To estimate the expected

values required for the learning process, we used the D-Wave processor to generate samples of the hidden layer configuration and estimate probabilities.

The hidden layer topology is based on the Chimera graph, which represents the underlying connection topology of the D-Wave processor. Chimera graphs are composed of 8-qubit cells. Within each cell, the nodes have a four-by-four fully bipartite connectivity. The bipartite cells are arranged in a grid pattern. The four nodes from one side of the bipartite graph are linked horizontally to two other nodes of adjacent cells in horizontal direction. The other four nodes are linked vertically to two other nodes of adjacent cells in the vertical direction (Figure 2).

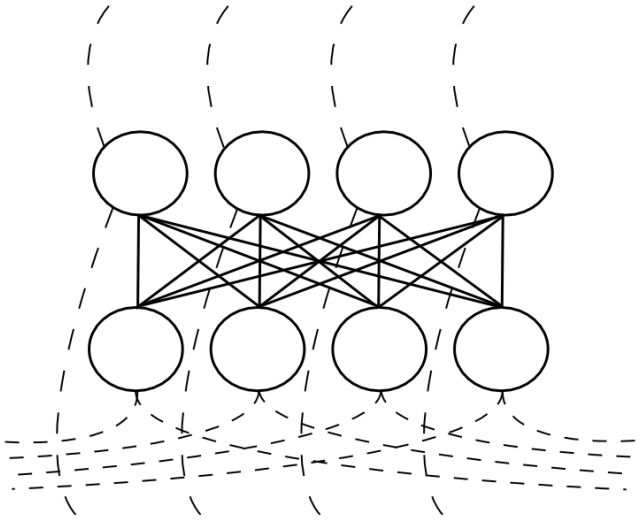

Figure 2. One cell of the Chimera graph. Chimera graphs are composed of 8-qubit cells, with bipartite connectivity within the cell. Note that this figure only depicts the hidden units, and these units are fully connected to the input. The dashed lines indicate connections to neighboring cells. These neighboring connections are not used in this work.

We chose 6,000 images from the MNIST database for training the LBM. When loading the pixels of each image, we also included 10 "classification" digits, which are not initially shown in the visible layer and are used to calculate gradients and determine the LBM's output label. Each image sets exactly one classification node to the proper label (for example, an image of number 3 will have output node 3 while all other classification nodes will be off).

We utilize common parameters to control the learning progress, the same ones found in training RBMs [22]. We chose to conduct training over 25 epochs instead of 10 epochs to get a better idea of what performance we can potentially achieve. Another parameter is the learning rate, or how much the LBM learns from each example. Setting the parameter too high can cause unstable behavior. This can be thought of as the LBM compensating too much for an error. Setting the parameter too low has the obvious downside of the LBM not learning anything of value from an example. We chose a relatively standard learning rate of 0.1 for visible-to-hidden edges and 0.0001 for hidden-to-hidden edges.

We wanted to determine if any performance advantage would be gained from using this LBM topology instead of the traditional RBM topology. First we ran a small experiment to confirm that the training of the LBM would behave correctly. Figure 3 shows the input reconstruction error and the classification rate for a LBM, confirming that it learns the MNIST data. The RBM and LBM were both implemented on D-Wave and on MNIST images using the same number of hidden and visible units over ten training epochs (one epoch is a complete run over all the training data). The RBM configuration, as discussed, has no intra-layer connections, whereas the LBM configuration has limited connections between the hidden nodes.

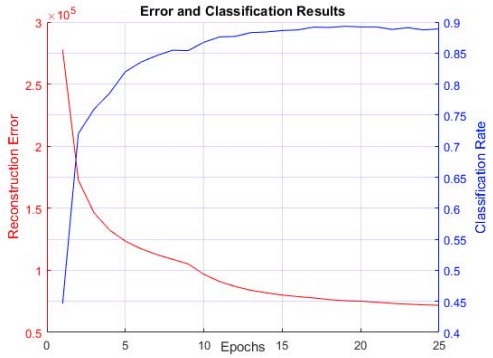

Figure 3. A record of reconstruction error and classification rate versus training epochs. Reconstruction error steadily decreases as the classification rate rises, showing that the LBM is learning from the MNIST data.

We initially found that the LBM configuration performed worse than the RBM configuration when we included couplings between nodes in the hidden layer. This was not what we expected, so we introduced a hybrid learning scheme where these intra-layer couplings were redrawn from a random normal distribution for the first three training epochs. From epoch 4 on, the weights were allowed to follow the typical learning rule used in BMs. The results can be seen in the blue series in Figure 4. The choice of using a three epoch duration for randomization was rather arbitrary and the full effects of choosing different durations can be explored in future work. We were primarily interested in providing some randomization while retaining a modest amount of learning time (seven epochs).

The final classification rate for our trained LBM was 88.90 percent. Reconstruction error dropped in a regular, expected manner as seen in Figure 4. In practical terms, we want to compare the cost of training LBMs on quantum devices versus training LBMs on traditional architectures.

4.2. HPC

We used the Titan computer and the MENNDL system to discover an optimal topology of a deep network trained on the MNIST handwritten digit dataset [19] by utilizing the method presented in [9]. The hyper-parameters optimized were the kernel size, the number of hidden units for each

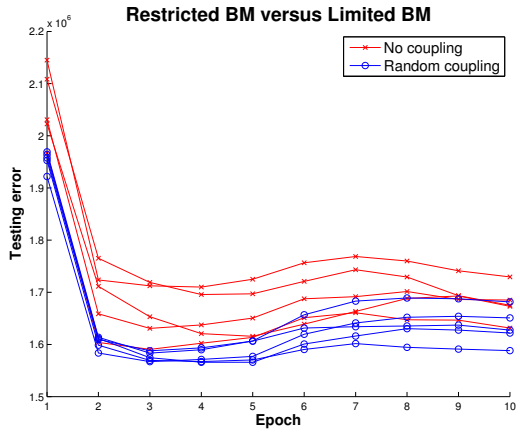

Figure 4. Reconstruction error (not normalized) of BMs trained using no intra-layer connections (red) and using random intra-layer connections (blue).

of the convolutional layers, and the number of hidden units in the fully connected layer. The structure of the network is shown in Figure 6. Utilizing 500 nodes of Titan, the evolutionary algorithm was trained for 32 generations with 500 individuals in the population allowing us to evaluate $16,000$ networks. Each hyper-parameter is encoded as an integer gene, and the range of this integer is limited in order to avoid evaluating hyper-parameter values that are not of interest. A single node of Titan evaluates the core of the evolutionary algorithm and distributes the fitness function to the rest of the nodes to evaluate the network using Titan's GPUs.

The optimal network, shown in Figure 6, demonstrates some significant differences from the starting network. The optimal network achieved 98.5 percent classification accuracy, representing a 0.3 percent increase over the baseline network. This demonstrates that accuracy can be improved by optimizing hyper-parameters even on networks and datasets that have been widely used. Figure 5 highlights the best performing networks and shows their corresponding hyper-parameters. It is interesting to note that the best performing networks had a wide variety of values for the number of hidden units in the fully connected layer. However, there was little variation in the kernel size of the convolutional layers which indicates that the performance of the network is much more sensitive to this parameter, and the kernel size of the second layer converged to a much smaller value than the value that is typically used. This indicates that even for very well studied problems, i.e., MNIST, the networks typically used are not optimal since it is difficult to find the correct hyper-parameters without an automated search process and significant computing resources.

4.3. Neuromorphic

In order to explore memristor-based neuromorphic systems, we simulated a memristive implementation of a NIDA network trained to classify MNIST images (Figure 7). The NIDA network itself was generated using evolutionary optimization and was part of an ensemble of networks that classifies MNIST images with an accuracy of approximately 90 percent. The memristive device technology assumed for this simulation is characterized by a 60kΩ low resistance state and an on-off ratio of just 10. While the on-off ratio is fairly low, this model is representative of characteristics achievable for experimentally observed hafnium-oxide memristors [23]. Our memristive NIDA simulation setup also includes analog integrate-and-fire neurons, implemented using a 65nm CMOS process technology. Neuromorphic elements (neurons and synapses) were simulated using Cadence Spectre and system-level energy and power estimates were calculated using a high-level simulator written in C++.

A memristive NIDA network used to classify a particular MNIST image (the digit '0') was found to consume an average power of 1.72mW. For the full 500 cycles required to classify the MNIST image, including loading the image and allowing the network to process the data, our memristive implementation was found to consume a total energy of 710nJ. These results are consistent with results determined for similar memristor-based neuromorphic systems [24]. Research has also shown that memristive neuromorphic systems are typically $20\times$ more energy-efficient than their CMOS counterparts [25], and our results are consistent with this estimation. Further improvements in energy-efficiency are possible through the use of memristors with a higher on-off ratio and/or higher low resistance state. Ultra low-power CMOS circuit design techniques (i.e., sub-threshold operation) can also be used to further reduce the power consumption of CMOS neurons. Thus, a CMOS-memristive neuromorphic implementation is particularly well suited for energy-constrained, resource limited application domains.

5. Discussion

The three architectures use three different deep learning approaches to address this DL problem. Given the significant differences in the architectures, we believe these different approaches provide insight into how these three types of machines can be used to create complex and trainable neural networks.

The quantum approach allows the deep learning network topologies to be much more complex than is feasible with conventional computers. The results show training convergence with a high number of intra-layer connections, thus opening the possibility of using much more complex topologies that can be trained on a quantum computer. There is not a time-based performance penalty for increased intra-layer connections, although, there may be the need to do more sampling in order to reduce potential errors.

HPC's contribution to the problem focuses on automatically developing an optimal network topology to create a high performing network. Many of the topologies used today are developed through trial and error methods. This approach works well with standard research datasets since the research community can learn and publish what topologies produce the highest accuracy networks for these data. It is

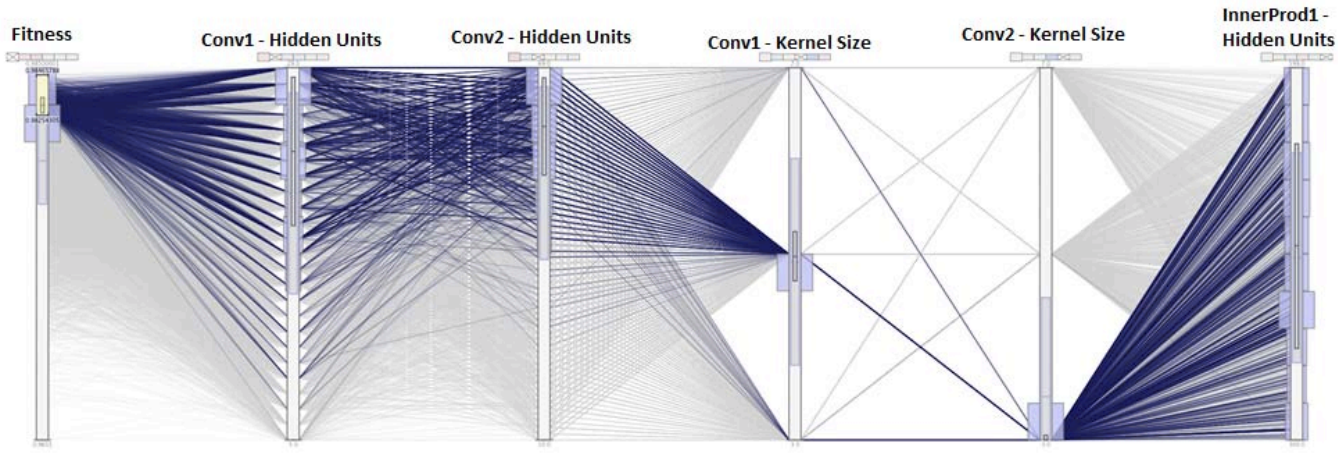

Figure 5. Parallel coordinates plot of networks evaluated by the evolutionary algorithm. The best performing networks are highlighted in dark purple such that the best hyper-parameters can be observed. This demonstrates that the best networks utilized smaller kernel sizes than those typically used and that the number of hidden units in the convolutional layers was much more closely tied to performance than the number of hidden units in the inner product layer.

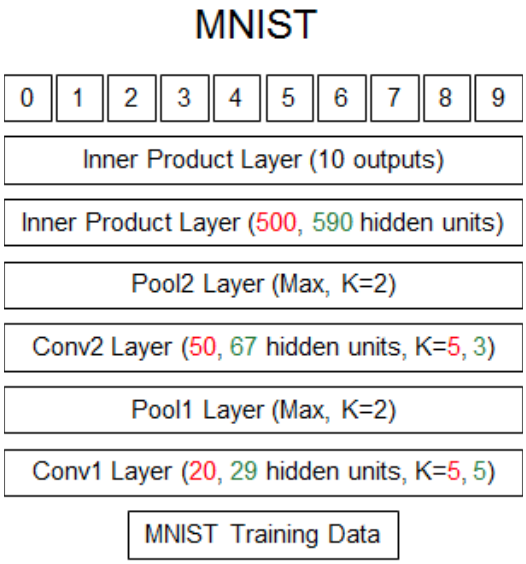

Figure 6. Network being optimized. This figure shows the typical hyper-parameters used (red) and the hyper-parameters learned through the evolutionary algorithm (green).

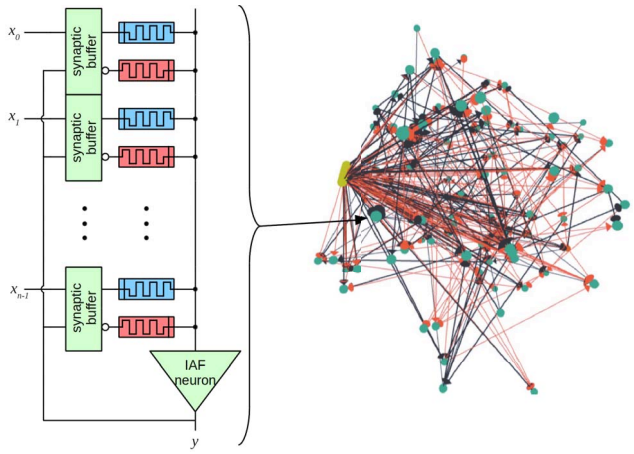

Figure 7. An example memristive neuromorphic circuit with integrated-and-fire (IAF) neuron (left) and a NIDA network trained to classify digit '0' MNIST images (right).

a different matter when working with datasets that have not been widely studied. The HPC approach provides a way to optimize the hyper-parameters of a CNN, saving significant amounts of time when working on new datasets.

The neuromorphic contribution to this problem is to provide low-power memristor-based hardware to implement a SNN. The network has the potential to have broader connections than a CNN and the ability to dynamically reconfigure itself over time. There are many benefits to neuromorphic computers (including robustness, low energy usage, and small device footprint) that could be useful in a real-world environment today if we had a mechanism for finding good network solutions to deploy on those devices.

Reviewing the results of the three experiments opens the possibility of using these three architectures in tandem to create powerful deep learning systems that are beyond our current capabilities. Practically, the current quantum computer is quite limited in the size and scope of the problem it can address, but the ability to train a very complex deep learning network gives it a very interesting potential. It could be used to generate weights for very complex networks that are untrainable using current systems opening the possibilities of potentially solving more complex and challenging problems. The HPC approach of automatically finding optimal deep learning topologies is a fairly robust and scalable capability, though quite expensive

in development and computer costs. Having the ability to use deep learning methods on unstudied datasets (experimental scientific data) can provide a huge time savings and analytical benefit to the scientific community. The neuromorphic approach is limited by the lack of robust neuromorphic hardware and algorithms, yet it holds the potential of analyzing complex data using very low power hardware. Combining the three approaches provides the potential to discover and train complex deep learning topologies and models, and natively represent these models in low-powered hardware. This is a capability that is not feasible with a von Neumann architecture. It holds the potential to solve much more complicated problems than can currently be solved with deep learning.

5.1. Future Work

Our next step is to understand the types of problems that could be solved using complex deep learning networks and the three architectures in tandem. On the quantum side, we will explore more complex BM topologies; on the HPC side, we will look at extending MENNDL concepts to LBM; and on the neuromorphic side, we will be moving toward physical memristor implementations along with exploring a memristor-based LBM implementation. We also plan on moving from an MNIST problem to a scientific dataset.

6. Conclusion

Current Deep Learning networks are loosely based on this neural model of human perception and have been highly optimized using CNNs trained on large clusters of GPUs. This technology has been instrumental in solving problems that have challenged researchers for years, such as object and facial recognition within photographs. The topology of these CNN networks consist of convolutional layers with shared weights and fully connected layers, without inter-layer connections, which while powerful, are quite simplistic.

To study more complex networks, there are three main challenges: 1) training models with complex topologies that contain intra-layer connections; 2) automatically determining an optimal configuration for a network topology; and 3) implementing a complex topology in hardware. To address these problems we explore a simple deep learning problem on three different architectures: quantum, high performance, and neuromorphic computers. These architectures address the three problems: complex topologies with quantum computing; network topology optimization with high performance computing; and low-power implementation with neuromorphic computing.

Given input size limitations of 1,000 qubits, we use the MNIST dataset for this evaluation, and use neuron models and topologies that are best suited to the architectures: CNN for HPC; SNN for neuromorphic; and BMs for quantum.

Our results from these three experiments demonstrate the possibility of using these three architectures to solve complex deep learning networks that are untrainable using a von Neumann architecture.

The quantum computer experiment demonstrated that a complex neural network, i.e., one with intra-layer connections, can be successfully trained on the MNIST problem. This is a key advantage for a quantum approach and opens the possibility of training very complex networks. A high performance computer can be used to take the complex networks as building blocks and compare thousands of models to find the best performing networks for a given problem. And finally, the best performing neural network and weights can be implemented into a complex network of memristors producing a low-power hardware device. This is a capability that is not feasible with a von Neumann architecture. This holds the potential to solve much more complicated problems than can currently be solved with deep learning.

Acknowledgments

This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under contract number DE-AC05-00OR22725.

References

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[2] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for boltzmann machines," *Cognitive science*, vol. 9, no. 1, pp. 147–169, 1985.

[3] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006. [Online]. Available: http://science.sciencemag.org/content/313/5786/504

[4] R. P. Feynman, "Simulating physics with computers," *International journal of theoretical physics*, vol. 21, no. 6, pp. 467–488, 1982.

[5] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM J. Comput.*, vol. 26, no. 5, pp. 1484–1509, Oct. 1997. [Online]. Available: http://dx.doi.org/10.1137/S0097539795293172

[6] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser, "Quantum computation by adiabatic evolution," Massachusetts Institute of Technology, Report MIT-CTP-2936, 2000.

[7] M. W. Johnson, M. H. S. Amin, S. Gildert, T. Lanting, F. Hamze, N. Dickson, R. Harris, A. J. Berkley, J. Johansson, P. Bunyk, E. M. Chapple, C. Enderud, J. P. Hilton, K. Karimi, E. Ladizinsky, N. Ladizinsky, T. Oh, I. Perminov, C. Rich, M. C. Thom, E. Tolkacheva, C. J. S. Truncik, S. Uchaikin, J. Wang, B. Wilson, and G. Rose, "Quantum annealing with manufactured spins," *Nature*, vol. 473, no. 7346, pp. 194–198, 05 2011. [Online]. Available: http://dx.doi.org/10.1038/nature10012

[8] A. Coates, B. Huval, T. Wang, D. J. Wu, B. Catanzaro, and A. Y. Ng, "Deep learning with COTS HPC systems," in *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, 2013, pp. 1337–1345. [Online]. Available: http://jmlr.org/proceedings/papers/v28/coates13.html

[9] S. R. Young, D. C. Rose, T. P. Karnowski, S.-H. Lim, and R. M. Patton, "Optimizing deep learning hyper-parameters through an evolutionary algorithm," in *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, ser. MLHPC '15. New York, NY, USA: ACM, 2015, pp. 4:1–4:5. [Online]. Available: http://doi.acm.org/10.1145/2834892.2834896

[10] A. S. Cassidy, P. Merolla, J. V. Arthur, S. K. Esser, B. Jackson, R. Alvarez-Icaza, P. Datta, J. Sawada, T. M. Wong, V. Feldman *et al.*, "Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores," in *Neural Networks (IJCNN), The 2013 International Joint Conference on*. IEEE, 2013, pp. 1–10.

[11] J. Shen, D. Ma, Z. Gu, M. Zhang, X. Zhu, X. Xu, Q. Xu, Y. Shen, and G. Pan, "Darwin: a neuromorphic hardware co-processor based on spiking neural networks," *Science China Information Sciences*, vol. 59, no. 2, pp. 1–5, 2016.

[12] N. Jouppi, "Google supercharges machine learning tasks with tpu custom chip," May 2016, uRL: https://cloudplatform.googleblog.com/2016/05/Google-supercharges-machine-learning-tasks-with-custom-chip.html.

[13] Nervana, "Nervana engine," 2016, uRL: https://www.nervanasys.com/technology/engine/.

[14] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha, "Backpropagation for energy-efficient neuromorphic computing," in *Advances in Neural Information Processing Systems*, 2015, pp. 1117–1125.

[15] R. S. Williams, "How we found the missing memristor," *IEEE Spectrum*, vol. 45, no. 12, pp. 28–35, Dec 2008.

[16] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, "Nanoscale memristor device as synapse in neuromorphic systems," *Nano letters*, vol. 10, no. 4, pp. 1297–1301, 2010.

[17] K.-H. Kim, S. Gaba, D. Wheeler, J. M. Cruz-Albrecht, T. Hussain, N. Srinivasa, and W. Lu, "A functional hybrid memristor crossbar-array/cmos system for data storage and neuromorphic applications," *Nano letters*, vol. 12, no. 1, pp. 389–395, 2011.

[18] M. Prezioso, F. Merrikh-Bayat, B. Hoskins, G. Adam, K. K. Likharev, and D. B. Strukov, "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," *Nature*, vol. 521, no. 7550, pp. 61–64, 2015.

[19] Y. LeCun, C. Cortes, and C. J. Burges, "The MNIST database of handwritten digits," 1998.

[20] C. D. Schuman, J. D. Birdwell, and M. E. Dean, "Spatiotemporal classification using neuroscience-inspired dynamic architectures," *Procedia Computer Science*, vol. 41, pp. 89 – 97, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050914015348

[21] A. Disney, J. Reynolds, C. D. Schuman, A. Klibisz, A. Young, and J. S. Plank, "DANNA: A neuromorphic software ecosystem," *Biologically-Insipred Cognitive Architectures 2016*, p. In press, 2016.

[22] G. Hinton, "A practical guide to training restricted boltzmann machines," *Momentum*, vol. 9, no. 1, p. 926, 2010.

[23] N. Cady, K. Beckmann, H. Manem, M. Dean, G. Rose, and J. V. Nostrand, "Towards memristive dynamic adaptive neural network arrays," in *Proceedings of the Government Microcircuit Applications and Critical Technology Conference (GOMACTech)*, March 2016.

[24] C. Yakopcic, R. Hasan, and T. M. Taha, "Memristor based neuromorphic circuit for ex-situ training of multi-layer neural network algorithms," in *2015 International Joint Conference on Neural Networks (IJCNN)*, July 2015, pp. 1–7.

[25] X. Liu, M. Mao, B. Liu, B. Li, Y. Wang, H. Jiang, M. Barnell, Q. Wu, J. Yang, H. Li, and Y. Chen, "Harmonica: A framework of heterogeneous computing systems with memristor-based neuromorphic computing accelerators," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 5, pp. 617–628, May 2016.