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

THOMAS E. POTOK, CATHERINE SCHUMAN, STEVEN YOUNG, and ROBERT PATTON, Oak Ridge National Laboratory
FEDERICO SPEDALIERI, JEREMY LIU, and KE-THIA YAO, USC Information Sciences Institute
GARRETT ROSE and GANGOTREE CHAKMA, University of Tennessee

Current deep learning approaches have been very successful using convolutional neural networks trained on large graphical-processing-unit-based computers. Three limitations of this approach are that (1) they are based on a simple layered network topology, i.e., highly connected layers, without intra-layer connections; (2) the networks are manually configured to achieve optimal results, and (3) the implementation of the network model is expensive in both cost and power. In this article, we evaluate deep learning models using three different computing architectures to address these problems: quantum computing to train complex topologies, high performance computing to automatically determine network topology, and neuromorphic computing for a low-power hardware implementation. We use the MNIST dataset for our experiment, due to input size limitations of current quantum computers. Our results show the feasibility of using the three architectures in tandem to address the above deep learning limitations. We show that a quantum computer can find high quality values of intra-layer connection weights in a tractable time 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.

CCS Concepts: • Computing methodologies → Deep belief networks; Evolvable hardware; Genetic algorithms; Model development and analysis; • Computer systems organization → Heterogeneous (hybrid) systems; • Hardware → Quantum computation;

Notice: This manuscript was authored by UT-Battelle, LLC under contract DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

This material was based on work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Robinson Pino, program manager, under contract DE-AC05-00OR22725. 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.

Authors’ addresses: T. E. Potok, C. Schuman, S. Young, and R. Patton, Computational Data Analytics Group, Oak Ridge National Laboratory, P.O. Box 2008, Oak Ridge, TN 37831; emails: {potokte, schumancd, youngsr, pattonrm}@ornl.gov; F. Spedalieri, J. Liu, and K.-T. Yao, University of Southern California, Information Sciences Institute, 4676 Admiralty Way, Suite 1001, Marina del Rey, CA 90292; emails: f spedali@isi.edu, jeremyjl@usc.edu, kyao@isi.edu; G. Rose and G. Chakma, Department of Electrical Engineering and Computer Science, University of Tennessee, 1520 Middle Dr, Suite 401, Knoxville, TN 37996 USA.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2018 ACM 1550-4832/2018/07-ART19 $15.00
https://doi.org/10.1145/3178454

ACM Journal on Emerging Technologies in Computing Systems, Vol. 14, No. 2, Article 19. Pub. date: July 2018.
1 INTRODUCTION

Deep learning is inspired by the networks of neurons in the visual cortex of the brain. Early versions of these neural networks have been simulated on a computer to analyze imagery. While promising, a significant limitation of this approach has been the computational time required to train or optimally set the weights within a network. Graphical processing units (GPUs) 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. These more complex networks containing multiple hidden layers are known as deep learning networks.

In this article, we explore the current limitations of deep learning and test potential solutions on the emerging computational architectures of quantum computing, high performance computing, and neuromorphic computing. For this article, we will focus on Boltzmann machines (BMs), convolutional neural networks (CNNs), and spiking neural networks (SNNs). Our purpose is to assess the feasibility of a combination of the three architectures on a deep learning problem, rather than conducting a head-to-head performance comparison of the three approaches.

There are multiple designs for deep learning networks, with CNNs being the most widely used deep learning models (LeCun et al. 1998a). They are most commonly used for image classification with remarkably good results. CNNs have many similarities to traditional multi-layer perceptron neural networks. They utilize stochastic gradient descent and backpropagation for training a set of learnable weights in a supervised manner. The defining feature of convolutional networks are their convolutional layers and pooling layers. Convolutional layers consist of multiple sets of weights, or kernels, that are convolved with their input to form a set of feature maps. Pooling layers are used to subsample the outputs of a convolutional layer to produce a smaller feature map, usually by using a max or average operation. These layers combine to utilize shared weights and pooling to take advantage of the locality that exists within images. The concept of shared weights builds upon the assumption that if a feature is useful in one location of an image, it is also useful in other locations. Thus, the network need not calculate a unique set of weights for every position in the image. Since these weights are shared, activations from one position in an image are calculated using the same weights as other locations, and these weights can be pooled in a meaningful way. This pooling operation allows the network to be resilient to shifts of features within the image (Scherer et al. 2010). Subsequent improvements to CNNs, such as dropout (Hinton et al. 2012), rectified linear units (Jarrett et al. 2009; Nair and Hinton 2010), and efficient GPU deep learning codes (Abadi et al. 2015; Al-Rfou et al. 2016; Jia et al. 2014), have allowed CNNs to achieve impressive results on a variety of benchmarks. These include the ImageNet (Russakovsky et al. 2015) dataset where CNNs have surpassed human level performance at object recognition (He et al. 2015) and the Labeled Faces in the Wild (Huang et al. 2008) dataset where CNNs have surpassed human level performance at face recognition (Schroff et al. 2015).
A BM is a recurrent neural network consisting of neurons that make binary stochastic decisions based on the states of their symmetrically connected neuron neighbors (Ackley et al. 1985). BMs 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. A BM training algorithm was proposed in (Ackley et al. 1985). This training algorithm relies on iterations of updating the states until a thermal equilibrium is reached, and then updating weights based on a simple learning rule. Though this process can be improved by utilizing simulated annealing to reach thermal equilibrium, it is a very slow process for large networks. This is because there are $2^N$ possible states the network could take for a network comprised of $N$ neurons along with $2^N$ weights to be learned. Identifying the state of thermal equilibrium and calculating equilibrium statistics becomes a challenge for large networks even for the best computational resources available as the computation grows exponentially with $N$. As such, the training of a BM is impractical for complex network topologies. This has given rise to the development of a restricted Boltzmann machine (RBM). The RBM network topology is restricted to a bipartite graph (Hinton and Salakhutdinov 2006). Deep belief networks can be created by composing many RBM layers (Hinton et al. 2006).

There are currently three main challenges in deep learning. The first is how to train models with complex topologies, such as those networks that contain intra-layer connections. Training these models on conventional computers proves to be intractable, yet having these additional connections should provide greater connectivity that will likely increase the representational capabilities of the model as has been shown in (Wiebe et al. 2016). Current deep learning networks limit the scale and complexity of the network models they use. While very successful in solving challenging classification problems, these network models are not comparable to those produced by nature. Early deep learning models proposed networks that contained intra-layer connections, but proved to be intractable to train on conventional computer systems. We believe that quantum computing may offer a potential solution with the ability to sample from complex probability distributions like those generated by neural networks that contain intra-layer connections.

The second challenge is how to automatically determine optimal or near-optimal neural network hyperparameters and topologies. Current deep learning models are created, trained, and tested on reference datasets, with high performing network configurations reported in the literature. A significant challenge is how to construct a high performing network from previously unexamined data. GPU-based high performance computing provides an opportunity to train, test, and evolve thousands of deep learning networks to find well performing network hyperparameters.

The last challenge is how to natively implement a complex deep learning model using a simulated neuron and synapse hardware architecture, as opposed to a CPU/memory-based model. While conventional computers can and are being used to deploy deep learning networks, the power requirements are high, especially when compared with nature. Spiking neuromorphic devices provide neuron and synapse hardware architecture that incorporates a time component (spike), and when implemented with memristive technology has the potential to run deep learning networks with very low power consumption.

There has been significant work in mapping CNN architectures to spiking neuromorphic systems (Esser et al. 2016; Indiveri et al. 2015). However, we believe there is potential to utilize the underlying processing power of SNNs (such as temporal processing) to contribute to a deep learning task, rather than relying on a conversion from a CNN.

Our hypothesis is that these three deep learning challenges can be addressed through a combination of quantum, high performance, and neuromorphic computing. To test this hypothesis we use a simple deep learning problem, MNIST, using a native deep learning network representation.
for each of the three computing platforms, i.e., a BM for quantum, a CNN for high performance computers, and an SNN for neuromorphic.

This article will provide a brief background on the challenges of deep learning related to quantum, high performance, and neuromorphic computing, followed by our experimental approach, results, and future research.

2 RELATED WORK

First, we look at the current state-of-art quantum, high performance, and neuromorphic computing as related to the challenges in deep learning as stated previously.

2.1 Quantum Computing

Computing using quantum computers was first discussed by (Feynman 1982) 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 (Shor 1997) 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 (Farhi et al. 2000). All three have been shown to have the same computational power. In this article, we focus on a restricted form of the adiabatic quantum computing model, which performs adiabatic quantum optimization 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 (Johnson et al. 2011).

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 on traditional computer systems, thus the development of a RBM restricts the network topology to that of a bipartite graph (Hinton and Salakhutdinov 2006). Bipartite graphs allow for techniques like contrastive divergence to efficiently draw approximate samples in linear time from the probability distribution defined by the BM. This enables the evaluation of a network of restricted BMs. 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 an NP-hard problem. Adiabatic quantum machines have the potential to efficiently sample a richer set of graph topologies that are supersets of bipartite graphs. Several approaches to exploit this feature have been attempted using the D-Wave processor for different choices of graph size and topology (Adachi and Henderson 2015; Benedetti et al. 2016a, 2016b).

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 (Coates et al. 2013). 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 to apply deep learning to a new dataset.

For CNNs, 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 (Young et al. 2015). This work utilized an evolutionary algorithm (EA) distributed across the nodes of Oak Ridge National Laboratory’s (ORNL’s) Titan supercomputer 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 SNNs, such as IBM’s TrueNorth (Cassidy et al. 2013) or Darwin (Shen et al. 2016), and building devices based on CNNs, such as Google’s Tensor Processing Unit (Jouppi 2016) or Nervana’s Nervana Engine (Nervana 2016), to serve as deep learning accelerators. The neuromorphic devices that have been built based on SNNs or built to simulate more 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 CNNs and recurrent neural networks such as long short-term memories (Hochreiter and Schmidhuber 1997). In SNN-based neuromorphic systems, the neuron is typically some form of spiking neuron, such as a leaky integrate-and-fire (IAF) 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 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. 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 (Esser et al. 2015). This mapping of an existing neural network trained with a well-studied algorithm (in this case, backpropagation) has been used for a variety of other neural network types as well, beyond CNNs, such as spiking Hopfield networks and spiking restricted BMs (Arthur et al. 2012). The algorithms that have been developed for spiking neuromorphic systems typically impose some sort of restriction for the network, or they have not yet been shown to be widely applicable. For example, a variation of back-propagation for SNNs
(SpikeProp) has been developed (Bohte et al. 2002; Schrauwen and Van Campenhout 2004), but it is restricted to feed-forward networks. Learning rules based on spike-timing-dependent plasticity or STDP has also been commonly used in SNN architectures (Song et al. 2000). Though STDP has been shown to be useful on some tasks, the true impact of STDP on real applications has not yet been demonstrated. However, we believe STDP mechanisms have great potential to be used as unsupervised weight training with a supervised algorithm that can help to determine network topology and/or parameters such as thresholds and delays.

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 SNN 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 (Williams 2008). Memristors have similar behavior to biological synapses, and as such, have been frequently utilized to implement neuromorphic systems (Jo et al. 2010; Kim et al. 2011; Prezioso et al. 2015).

3 APPROACH

The three platforms we are studying, (quantum, high performance, and neuromorphic computing), are quite different in the way they each process data. Selecting a deep learning problem that can be used on all three is constrained by the amount of data that each can support. Currently D-Wave supports 1,000 qubits, which limits the size of a problem to the inputs and deep learning network. MNIST is a collection of hand-written digits that has been very widely studied in the deep learning community (LeCun et al. 1998b). The images of the digits are very small (28 × 28 pixels totaling 784 pixels) that can be analyzed using 1,000 qubits of the quantum computer as well as the other architectures.

The next challenge is selecting the type of deep learning network that can be natively supported on the three platforms. While many deep learning methods have been proposed over the years, CNNs have consistently provided the highest accuracy on standard datasets, and typically would be the network of choice for such a comparison. However, the quantum architecture provides a native BM representation that does not restrict intra-layer connections, which is computationally impractical for conventional computers. Likewise, for a neuromorphic computer, a SNN provides a native time-based analysis model. Both a strongly connected BM, and time-dependent SNN operate quite differently than a CNN, but in conjugation with their respective platforms, provide distinctive capabilities that we believe can augment or strengthen a CNN model.

The last challenge is how to compare the experimental results of the three platforms. A nominal performance comparison of the three approaches provides little insight to the deep learning challenges we are addressing.

Likewise, an accuracy comparison on the all but solved MNIST problem is not helpful either. Instead, we look to compare the capabilities of the three architectures in addressing the deep learning challenges we have stated previously. We believe that a combination of these architectures may provide a significant benefit over that of a single platform.

Specifically, a quantum computer will be used to address the first deep learning challenge we list, namely, training a network that contains intra-layer connections. Quantum computers have the ability to sample a probability distribution of a network of BM, and may provide a feasible approach to training a highly connected BM network. We will use high performance computers to address the second challenge of how to automatically configure a network to an optimal or near optimal topology using EAs to evolve a high performing network. Lastly, we will address how to
Fig. 1. Chimera graphs are composed of eight-qubit cells, with bipartite connectivity. Each cell is connected to the adjacent cells.

natively implement such a model using a neuron and synapse hardware architecture simulation of a memristive-based SNN.

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. We propose a network of BMs to represent the MNIST problem. 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 (Ackley et al. 1985; Salakhutdinov and Hinton 2009). RBMs 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. First, we will create and train an RBM on D-Wave, applying it to the MNIST handwritten digit classification problem (LeCun et al. 1998b) to establish a reference result.

For the RBM experiment, we will be using a 1,000 qubit D-Wave adiabatic quantum computer, with 784 qubits as an input layer for the MNIST image set, 80 qubits as a hidden layer, and 10 qubits as an output layer. The hidden layer consists of 10 units of eight qubits each, with no intra-layer connections. The network will be trained over 25 epochs, then evaluated against the validation set.

Next, we will consider a more complex topology that allows for intra-layer connections between nodes. We call this semi-restricted BM a limited Boltzmann machine (LBM). The LBM has two layers: one visible and one hidden. With a 1,000 qubit, the visible layer requires 784 qubits, and 80 qubits for the hidden layer. 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 intra-layer couplings.

The hidden layer topology of the LBM is based on the Chimera graph, which represents the underlying connection topology of the D-Wave processor. Chimera graphs are composed of eight-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 1).
For the LBM, the presence of intra-layer couplings in the hidden layer makes sampling from its distribution non-trivial, even in the positive-phase (which is usually easy for RBMs). It is for this task that we leverage the use of the D-Wave processor. By choosing the hidden layer connection as a native subset of the Chimera graph, we can obtain approximate samples by running the quantum device. This also allows us to sidestep the complicated issue of embedding a given graph in the restricted architecture of the device. It is in this sampling step that we expect a potential advantage of the quantum approach to manifest itself.

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.

We also included 10 "classification" digits used to calculate gradients and determine the LBM’s output label, i.e., what digit has been read.

We chose 6,000 images from the MNIST database for training the LBM. We use a subset of randomly selected images instead of using the full MNIST dataset due to time considerations. Our D-Wave system is time-shared between different organizations, and within each time-share there are many users. Limiting ourselves to 10% of the data allowed us to run experiments in a timely manner.

Each 28 x 28 image is represented by a vector of 784 length, each unit holding a value representing pixel intensity in the range (0, 1). In addition to these 784 units representing pixel intensities, we include the previously mentioned 10 additional units to represent the image’s digit label (0 through 9) using 1-hot encoding. For example, an image of the digit “3” will have label unit 3 marked “on” while all other classification nodes will be off. Each image is thus represented with a vector of length 794. When we show each image to the RBM, we hide the image labels and have the RBM attempt to reconstruct the label units. We choose the unit label of highest “on” probability as the RBM’s digit classification choice for each image.

Training weights for the LBM are randomly initialized, just as they are in the RBM. Also as in the RBM, their values are sampled from a standard normal distribution and are updated using the same gradient descent procedure.

3.2 HPC

The high performance 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 extremely valuable in automatically finding 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 utilize an evolutionary optimization on Titan to evaluate tens of thousands of topologies (Young et al. 2015); 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.

For this experiment, 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.

We represent each individual within the population of the 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, and so on), 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, i.e.,
individual networks that have high accuracy are considered to be the most fit. Typically, generating the results for a single network on a small dataset like MNIST will require a modest amount of GPU/CPU time, and memory. However, creating, training, and evaluating tens of thousands networks requires a significant number of GPUs, like those in the Titan high performance computer.

After all the individuals in the population are evaluated, the top performing individuals are selected to generate a new population of individuals that represent the next generation of the EA. These new generations contain a mix of the well performing hyperparameters from the best performing networks in the population. Successive generations of individuals gradually lead to an improved set of hyperparameters over time. This method is called Multi-Node Evolutionary Neural Networks for Deep Learning (MENNDL) (Young et al. 2015).

For this experiment, we are looking to automatically discover hyperparameters of a well performing deep learning network on the MNIST dataset. 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 (LeCun et al. 1998a). This network utilizes two convolutional layers, two pooling layers, and one hidden fully connected layer. This is the network that is most often used with the MNIST dataset in the literature.

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 for improved results, and the ability to process 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 spiking neuromorphic approach to the MNIST problem is not the ideal solution since there is not a temporal component in the task of recognizing a handwritten digit. 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. The goal is to understand the deployment benefits of using an SNN in memristive hardware, as opposed to classification accuracy on this problem.

We then propose to evaluate the performance of this network on memristor hardware. Memristor hardware has the potential to provide a low power hardware implementation of a deep learning network.

As noted in Section 2.3, there are not very many SNN training methods or neuromorphic training methods that can be applied to spiking 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 (Schuman et al. 2014). NIDA is a simple SNN model composed of IAF neurons and synapses with delays and weights that are affected by processes similar to long-term potentiation and long-term depression in biological brains. A digital hardware implementation based on NIDA, called Dynamic Adaptive Neural Network Array (DANNA), has also been created and is currently implemented on FPGA with a digital VLSI implementation in progress (Disney et al. 2016). NIDA synapses have analog weight values, while DANNA is restricted to a finite set of digital weight values. DANNA also has restricted connectivity between neurons, whereas the NIDA model allows for fully connected networks. The NIDA model allows for us to study neuromorphic models in software and determine how restrictions different in hardware (such as weight resolution or connectivity) affect performance.
The EA approach for training networks for the MNIST problem was previously applied to the NIDA SNN (Schuman et al. 2014) and to DANNA (Disney et al. 2016). For both NIDA and DANNA, an ensemble approach is utilized, where 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% accuracy for NIDA and around 80% accuracy for DANNA were created.

The memristive device technology assumed for this simulation is characterized by a low resistance state (LRS) of 60kΩ, about an order of magnitude larger than the resistance of a typical deep-submicron CMOS transistor. This relatively high LRS for the memristor is desirable such that the CMOS channel resistance can effectively be neglected. The on-off ratio is assumed to be 10, providing a high resistance state (HRS) of 600kΩ. Such characteristics for LRS, HRS, and the associated on-off ratio have been observed for a range of memristive devices, including hafnium-oxide (HfO2) (Cady et al. 2016), tantalum-oxide (TaO2) (Yang et al. 2010), and titanium-oxide (TiO2) (Medeiros-Ribeiro et al. 2011). All of these memristive material stacks consist of an oxide layer sandwiched between two metallic layers. Depending on the polarity and magnitude of an applied voltage bias, the oxide layer transitions between being less or more conductive, providing the switching characteristics desirable for representing synaptic weights.

Our memristive NIDA simulation setup also includes analog IAF 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++. Specifically, we verified the high-level C++ model versus the circuit level implementation using small networks that were simulated using both Cadence Spectre and the high-level NIDA simulator. Larger networks, specifically MNIST, were simulated using the high-level NIDA simulator to determine neuron and synapse activity information.

The memristive NIDA simulation is based on two significant steps. Initially an evolutionary optimization training process is used to generate optimized networks for the low level simulation. At the same time, the transistor level simulation is done using Cadence Spectre simulator. Estimates are collected for the design components in different conditions (neuron accumulating but not firing, neuron firing, etc.). These “per component” energy estimates are used in conjunction with activity information from the high-level NIDA simulation to calculate the total energy consumed.

For this article, we extend the prior work by simulating an SNN (specifically, a NIDA network) implemented in memristive hardware to demonstrate the potential of significant power reductions for simulating the behavior of neural networks.

4 RESULTS

4.1 Quantum

We utilize common parameters to control the learning progress, the same ones found in training RBMs (Hinton 2010). For the initial test, we chose to conduct training over 25 epochs (1 epoch is a complete run over all the training data) 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
Fig. 2. A record of reconstruction error and classification rate versus training epochs using 6,000 MNIST images for training and 6,000 for testing. Reconstruction error steadily decreases as the classification rate rises, showing that the LBM is learning from the MNIST data.

Fig. 3. RBM and LBM performance on the MNIST digit classification task. The LBM tends to label the digits slightly better, and it produces significantly lower reconstruction error than the RBM.

The training of the LBM would behave correctly. Figure 2 shows the input reconstruction error and the classification rate for an LBM, confirming that it learns the MNIST data. In Figure 3, we show a comparison of RBM performance against LBM performance on this MNIST digit classification task.

The RBM and LBM were both implemented on D-Wave and on MNIST images using the same number of hidden and visible units. For this test, we trained over 10 epochs. The RBM configuration, as discussed, has no intra-layer connections, whereas the LBM configuration has limited connections between the hidden nodes.

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 one of our trained LBMs was 88.53%, which is comparable with other RBM results on MNIST (Diehl et al. 2015). 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.

The D-Wave quantum computing device is still an experimental platform, and it has a few important limitations, mainly its size and connectivity. But it is important to also note that currently available systems are not optimized for wall-clock performance, so we decided not to present any timing information but rather show the performance as a function of the number of training epochs. Hence, the main point of this part of the project was to validate use of this device to train a neural network using different underlying architectures and approximations than have been used previously (Adachi and Henderson 2015; Benedetti et al. 2016a, 2016b). For example, Adachi and Henderson mapped a whole RBM to the D-Wave but had to down sample the data for it to work, whereas we did not down sample any of the data. The hope is that this approach will provide a better scaling with problem size for this task, but the technology is still not mature enough to make a judgment one way or the other. However, it is worth noting that we implemented an LBM on a standard desktop and found that the amount of time to anneal and then collect enough samples on a desktop is significantly more than the quantum approach.

There is one aspect of the superconducting technology, on which the D-Wave processor is based, that is worth mentioning. The energy consumption of the system is dominated by the cooling of the processor, but the actual computation requires a negligible amount of energy. The cooling required has remained basically flat for four generations of the D-Wave device, and that is not expected to change in the foreseeable future. So even though it is not the usual feature of quantum devices that is touted when comparing them with classical systems, energy consumption may very well be an important reason for preferring quantum computing systems in the future.

4.2 HPC

We used the Titan computer, which utilizes one Nvidia Tesla K20x per node, and the MENNDL system to discover a near optimal topology of a deep network trained on the MNIST handwritten
Fig. 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.

digit dataset (LeCun et al. 1998b) by utilizing the method presented in Young et al. (2015). The hyper-parameters optimized were the kernel size, the number of hidden units for each 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. No data augmentation is performed such as rotations, scaling, or warping of the images. Utilizing 500 nodes of Titan, the EA was trained for 32 generations with 500 individuals in the population allowing us to evaluate 16,000 networks in 3 hours. With a modern GPU such as Nvidia’s Titan X Pascal, we would expect this time to be reduced to less than 30 minutes. Each hyper-parameter is encoded as an integer gene, and the range of this integer is limited to avoid evaluating hyper-parameter values that are not of interest. A single node of Titan evaluates the core of the EA 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 99.16 ± 0.01% classification accuracy, representing a statistically significant increase over the baseline network’s performance of 99.09 ± 0.01% even for the well-studied MNIST dataset. 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

This experiment was done in two parts. The first was to implement an SNN using NIDA to demonstrate a native spiking neuromorphic solution to they MNIST problem is feasible. The second part was to simulate the characteristics of this SNN implemented on memristive hardware. We started by simulating 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
Fig. 6. Network being optimized. This figure shows the typical hyper-parameters used (red) and the hyper-parameters learned through the evolutionary algorithm (green*).

Fig. 7. An example memristive neuromorphic circuit with integrated-and-fire neuron (left) and a NIDA network trained to classify digit “0” MNIST images (right). This NIDA network is meant to show that the network itself is not organized into layered structures and that it has recurrent connections; in particular, the exact topology of the network has been optimized for recognizing the digit 0, and different NIDA networks will have different topologies.

part of an ensemble of networks that classifies MNIST images with an accuracy of approximately 90%, which is comparable with other non-convolutional SNN approaches (Diehl et al. 2015). Energy consumption was also estimated for a NIDA representation of this network where synaptic elements are implemented using metal-oxide memristors.

The network on the right in Figure 7 shows a network of input neurons (shown as yellow spheres), hidden neurons (shown as teal spheres), and a single output neuron (shown as a red sphere). Each of those neurons corresponds to an IAF neuron circuit as indicated on the left of
Figure 7. Each of the synapses in the network is shown as an arrow in the network on the right, where red arrows correspond to inhibitory synapses (with negative weights) and blue arrows correspond to excitatory synapses (with positive weights). Each synapse in the NIDA network on the right has both a weight value and a delay value. To implement each synapse in the memristive network, we use a synapse buffer to implement the delay and two memristors to implement the weight, which allows us to realize both positive and negative weight values (Sayyaparaju et al. 2017).

To collect energy data, we considered three different phases of operation for the neurons: fire (neuron generates an output spike), accumulation (neuron accumulates input spikes from synapse), and idle phase (no input and no output spikes). Using Cadence Spectre, the energy of each phase has been determined as per spike energy. A similar method was also followed for the memristive synapses. Here we considered the active and passive phases of a synapse, specifically when the synapse is passing an input spike (active) and when it is not (passive). For the programmable delay chain, we collected energy data for each spike existing in the delay chain. The evolutionary optimization is used with the high-level simulator to generate a network with some activity factors for specific applications. For instance, the MNIST dataset was used as input for networks that recognize the handwritten characters, and the total energy of the system was calculated. To calculate the average energy for a single image run, the total energy consumed by the network is calculated for the total number of runs (which, for example, is 10,000 testing images for MNIST dataset application). The network has 128 neurons and 357 synapses associated with 357 programmable delays. The neurons and synapses are mostly analog circuit components but the programmable logic delay consists of digital components. As such, the energy consumed is mostly due to the digital parts of the synapse.

Simulations for energy and power estimates are based on experimental memristive devices that have exhibited thresholds in the range of 1V. Thus, the supply voltage for our simulation is 1.2V. The average power observed for the total network with 16.67MHz clock speed is 304.3mW and the respective energy is 18.26nJ (including the digital programmable delays). However, if we consider the core analog neuromorphic logic, the energy per spike is 5.24nJ and the average power is 87.43mW, which is consistent with similar memristor-based neuromorphic systems (Liu et al. 2016b). Research has also shown that memristive neuromorphic systems are typically 20× more energy-efficient than their CMOS counterparts (Liu et al. 2016a), 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.

As noted, the voltage supply for the memristive neuromorphic system simulation is 1.2V due to memristor thresholds in the range of 1V. Further, the resistance levels of many experimental metal-oxide memristors explored to date are in the range of a few kΩ to a few tens of kΩ (Cady et al. 2016). While the power results presented are in line with expectations for existing memristive devices and subsequent memristive neuromorphic systems (Liu et al. 2016a), power reductions can be expected with future improvements at the device level. Specifically, increased resistance levels, increased on/off ratios, and reduced threshold voltages will all help reduce system power consumption.

5 DISCUSSION

In this article, we perform a comparison on a standard benchmark problem (MNIST) on three different computing platforms (quantum, HPC, and neuromorphic) to provide baseline performance
for each. It is clear from these results that each of these platforms have their own strengths. The quantum approach is able to model a highly connected network that is impractical for traditional computers, perhaps leading to a richer representation of data for a given task. The HPC approach is able to study CNNs on a very large scale to produce an optimal CNN topology for a given task. The neuromorphic approach is able to realize a neural network solution with low power and with a smaller footprint than other computational approaches. However, it is also clear that the MNIST problem is not ideally suited to showcase the capabilities of either the quantum or neuromorphic system, as it has been essentially solved using CNNs.

For example, the potential increased representational power of the quantum LBM approach is likely better utilized on a more complex dataset. For spiking neuromorphic systems, datasets that include temporal components are better suited to this approach. As we propose in Figure 9, we foresee an architecture that provides the ability to leverage the strengths of each of these computing platforms for future, more complex datasets.

The goal of this article is to explore ways of addressing some of the current limitations of deep learning, namely, that contain intra-layer connections, automatically configuring the hyperparameters of a network, and natively implementing a deep learning model using energy efficient neuron and synapse hardware. We use three architectures, quantum computing, high performance computing, and neuromorphic computing, and three different deep learning models (LBM, CNN, and SNN) to address these issues.

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 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 the topologies that produce the highest accuracy networks for these data. It is 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 a native implementation and a low-power memristor-based hardware to implement an 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 that does not rely on conversions from non-SNN types.

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 (see Figure 8). 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. However, the scalability of a quantum machine is a real concern. As we observed, limiting the size of the input layer to 1,000 qubits severely limits the size...
of a problem that can be analyzed using this approach. We believe that the best use for complex networks may be as higher layers in a CNN (see Figure 9). These layers usually combine fairly rich features, and may benefit from increased intra-layer connections. These layers usually have a smaller input size than the original input, which eases the scalability concerns of this approach, and may improve over all accuracy.

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 temporal analysis and very low power hardware. One of the most compelling aspects of this approach is the combination of an SNN and neuromorphic hardware that can analyze the temporal aspects of data. The MNIST problem does not have a temporal component, but one can imagine a dataset that has both image and temporal aspects, such as a video. A CNN approach has been shown to perform well on the image side, perhaps an SNN can provide increased accuracy by analyzing the temporal aspects as well. For example, a CNN could analyze an image to detect objects within the image and output the location and/or orientation of those objects. This output can be used as input for an SNN. As each video frame is processed independently by the CNN, the output can be fed into the SNN, which can aggregate information over time and make conclusions about what is occurring in the video or to detect particular events that occur over time, but to do so in real-time. In this case, the CNN can be trained independently using the labeled frames of the video as input images, and the SNN can also be trained independently utilizing different objects with their locations and orientations as input.

These experiments provide valuable insights into deep learning by exploring the combination of three novel approaches to challenging deep learning problems. We believe that these three architectures can be combined to gain greater accuracy, flexibility, and insight into a deep learning approach. Figure 9 shows a possible configuration of the three approaches that addresses the three deep learning challenges we discussed above. The high performance computer is used to create a well performing CNN on image type data. The final layer or two is then processed by the quantum computer using an LBM network that contains greater complexity than a CNN. The temporal

| Platform     | Algorithm                          | Contribution                                          | Significance                                                                 |
|--------------|------------------------------------|-------------------------------------------------------|------------------------------------------------------------------------------|
| Quantum      | Limited Boltzmann                  | Development of LBM for quantum computers              | Increased representational ability for deep learning, potentially tractable for quantum computer |
| HPC          | Convolutional Neural Network       | Evolutionary optimization design CNN network          | Rapid design of high performing network                                      |
| Neuromorphic | Spiking Neural Network             | Native implementation of SNN on neuromorphic hardware | Very low power implementation of SNN                                          |

Fig. 8. A comparison of the platforms, deep learning approaches, contributions, and significance of the result from the MNIST experiment.
Fig. 9. A proposed architecture that shows how the three approaches, quantum, HPC, and neuromorphic, can be used to improve a deep learning approach. Image data can be analyzed using an HPC rapidly derived CNN with the top layers using an LBM on a quantum computer. The top layers have fewer inputs and require greater representational capabilities, which both play to the strength and limitations of a quantum approach. The temporal aspect of the data can be analyzed using an SNN, finally the image and temporal models will be merged to provide a richer and we believe a more accurate model, with an aim to be deployed in very low-power neuromorphic hardware.

aspects of the data are modeled using an SNN, and the ensemble models are then merged and an output produced. Our belief is that this approach has the potential to yield greater accuracy than existing CNN models.

5.1 Future Work
Our next step is to test the hypothesis that this proposed architecture does indeed provide greater accuracy, flexibility, and insight into a dataset than can be derived from a traditional CNN approach. We will apply this proposed architecture to a large scientific dataset and compare the results of a traditional approach to this proposed architecture.

6 CONCLUSION
Current deep learning networks are loosely based on a 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 consists of convolutional layers with shared weights and fully connected layers, without inter-layer connections, which, while powerful, are quite simplistic.
This article addresses three main limitations in deep learning: (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 native hardware. To address these problems we explored 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 neural networks 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 currently 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.

We propose a new deep learning architecture based on the unique capabilities of the quantum, high performance, and neuromorphic approaches presented in this article. This new architecture addresses the three main limitations we see in current deep learning methods, and holds the promise of higher classification accuracy, faster network creation times, and low power, native implementation in hardware.

REFERENCES

Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, et al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. White Paper. TensorFlow. Software available from http://tensorflow.org.

David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. 1985. A learning algorithm for Boltzmann machines. *Cogn. Sci.* 9, 1 (1985), 147–169.

Steve H. Adachi and Maxwell P. Henderson. 2015. Application of quantum annealing to training of deep neural networks. arXiv:1510.06356v1.

Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, et al. 2016. Theano: A Python framework for fast computation of mathematical expressions. arXiv:1605.02688. http://arxiv.org/abs/1605.02688.

John V. Arthur, Paul A. Merolla, Filipp Akopyan, Rodrigo Alvarez, Andrew Cassidy, Shyamal Chandra, Steven K. Esser, Nabil Imam, William Risk, Daniel B. D. Rubin, et al. 2012. Building block of a programmable neuromorphic substrate: A digital neurosynaptic core. In *Proceedings of the 2012 International Joint Conference on Neural Networks (IJCNN’12)*. IEEE, 1–8.

Marcello Benedetti, John Realpe-Gómez, Rupak Biswas, and Alejandro Perdomo-Ortiz. 2016a. Estimation of effective temperatures in quantum annealers for sampling applications: A case study with possible applications in deep learning. *Phys. Rev. A* 94, 2 (2016), 022308. DOI: http://dx.doi.org/10.1103/PhysRevA.94.022308

Marcello Benedetti, John Realpe-Gómez, Rupak Biswas, and Alejandro Perdomo-Ortiz. 2016b. Quantum-assisted learning of graphical models with arbitrary pairwise connectivity. arXiv:1609.02542v2.

Sander M. Bohte, Joost N. Kok, and Han La Poutr. 2002. Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing* 48, 1 (2002), 17–37.
A Study of Complex Deep Learning Networks

X. Liu, M. Mao, B. Liu, B. Li, Y. Wang, H. Jiang, M. Barnell, Q. Wu, J. Yang, H. Li, and Y. Chen. 2016a. Harmonica: A framework of heterogeneous computing systems with memristor-based neuromorphic computing accelerators. IEEE Trans. Circuits Syst. I: Reg. Papers 63, 5 (2016), 617–628.

Gilberto Medeiros-Ribeiro, Frederick Perner, Richard Carter, Hisham Abdalla, Matthew D. Pickett, and R. Stanley Williams. 2011. Lognormal switching times for titanium dioxide bipolar memristors: Origin and resolution. Nanotechnology 22, 9 (2011), 095702.

Vinod Nair and Geoffrey E. Hinton. 2010. Rectified linear units improve restricted Boltzmann machines. In Proceedings of the 27th International Conference on Machine Learning (ICML’10). 807–814.

Nervana. 2016. Nervana Engine. Retrieved from https://www.nervanasys.com/technology/engine/.

M. Prezioso, F. Merrikh-Bayat, B. D. Hoskins, G. C. Adam, K. K. Likharev, and D. B. Strukov. 2015. Training and operation of an integrated neuromorphic network based on metal-oxide memristors. Nature 521, 7550 (2015), 61–64.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, et al. 2015. ImageNet large scale visual recognition challenge. Int. J. Comput. Vis. 115, 3 (2015), 211–252.

Ruslan Salakhutdinov and Geoffrey Hinton. 2009. Deep Boltzmann machines. In Proceedings of the 12th International Conference on Artificial Intelligence and Statistics. 448–455.

Sagarvarma Sayyaparaju, Gangotree Chakma, Sherif Amer, and Garrett S. Rose. 2017. Circuit techniques for online learning of memristive synapses in CMOS-memristor neuromorphic systems. In Proceedings of the Great Lakes Symposium on VLSI. 479–482.

Dominik Scherer, Andreas Müller, and Sven Behnke. 2010. Evaluation of pooling operations in convolutional architectures for object recognition. In Proceedings of the International Conference on Artificial Neural Networks. 92–101.

Benjamin Schrauwen and Jan Van Campenhout. 2004. Extending SpikeProp. In Proceedings of the 2004 IEEE International Joint Conference on Neural Networks. Vol. 1. IEEE, 471–475.

Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. FaceNet: A unified embedding for face recognition and clustering. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 815–823.

Catherine D. Schuman, J. Douglas Birdwell, and Mark E. Dean. 2014. Spatiotemporal classification using neuroscience-inspired dynamic architectures. Proced. Comput. Sci. 41 (2014), 89–97. DOI: http://dx.doi.org/10.1016/j.procs.2014.11.089

Juncheng Shen, De Ma, Zonghua Gu, Ming Zhang, Xiaolei Zhu, Xiaoqiang Xu, Qi Xu, Yangjing Shen, and Gang Pan. 2016. Darwin: A neuromorphic hardware co-processor based on spiking neural networks. Sci. China Inform. Sci. 59, 2 (2016), 1–5.

Peter W. Shor. 1997. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. 26, 5 (1997), 1484–1509. DOI: http://dx.doi.org/10.1137/S0097539795293172

Sen Song, Kenneth D. Miller, and Larry F. Abbott. 2000. Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. Nat. Neurosci. 3, 9 (2000), 919–926.

Nathan Wiebe, Ashish Kapoor, and Krysta M. Svore. 2016. Quantum deep learning. Quantum Inf. Comput. 16 (2016), 0541–0587.

R. S. Williams. 2008. How we found the missing memristor. IEEE Spect. 45, 12 (2008), 28–35. DOI: http://dx.doi.org/10.1109/MSPEC.2008.4687566

J. Joshua Yang, M. X. Zhang, J. P. Strachan, F. Miao, M. D. Pickett, R. D. Kelley, G Medeiros-Ribeiro, and R. S. Williams. 2010. High switching endurance in TaOx memristive devices. Appl. Phys. Lett. 97, 23 (2010), 232102.

Steven R. Young, Derek C. Rose, Thomas P. Karnowski, Seung-Hwan Lim, and Robert M. Patton. 2015. Optimizing deep learning hyper-parameters through an evolutionary algorithm. In Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments (MLHPC’15). ACM, Article 4, 5 pages. DOI: http://dx.doi.org/10.1145/2834892.2834896

Received March 2017; revised November 2017; accepted January 2018