Spring May 13, 2018

Hardware-based Restricted Boltzmann Machine Implementation Using Probabilistic Spin Logic Devices

Adedoyin Adepegba

Hardware-Based Restricted Boltzmann Machine Implementations

Using CMOS and Beyond-CMOS Technologies

Adedoyin Adepegba

University of Central Florida

Objective

This project focuses on using emerging beyond-CMOS (complementary metal-oxide semiconductors) technologies, such as spintronics, to overcome software limitations and achieve an energy-efficient Resisted Boltzmann Machine(RBM) implementation. This research project will attempt to create a hardware device that can implement the RBM network. Ultimately, the goal is to have these hardware devices efficiently imitate the neural network of the human brain.

Background and Literature Review

The interrelated fields of machine learning (ML), artificial intelligence (AI), and artificial neural networks (ANN) have grown significantly in previous decades (Zand et al., 2017). Most ML techniques in-use today rely on supervised learning, where the systems are trained on patterns with a known desired output. However, intelligent biological systems exhibit unsupervised learning by evolving as they encounter and analyze more data samples. One interesting class of the unsupervised learning approach that has been extensively researched is the Restricted Boltzmann machine (RBM)(Zand et al., 2017). According to a research project by Bojnordi and Ipek titled "*Memristive Boltzmann machine: A hardware accelerator for combinational optimization and deep learning,"* the invention of RBM was accredited to Hinton in 1983 (Fahlman, Hinton, & Sejnowski, 1983). Most research has focused on software-based implementation of the RBM, which provides flexibility but requires significant execution time and energy when implemented on conventional computing architectures. Thus, research into hardware-based RBM designs has recently sought to alleviate these constraints on software. Successful applications of RBM implementation include Natural Language Understanding, Image Classification and Speech Recognition (Sarikaya, Hinton, & Deoras, 2014). Sarikaya et al. applied the RBM for natural language understanding with the goal to enable communication

between humans and machines. However, being able to increase this network size has come at the cost of performance as explained by Sarikaya et al. (Sarikaya et al., 2014).

Researchers and scientist have been pushing for ways to increase the performance of the neural network by creating specialized RBM hardware to perform probabilistic computing. A hardware-accelerated method was introduced in (Bojnordi & Ipek, 2016) using Resistive Random-Access Memory (RRAM). This method eliminated the need for data exchanged between memory and a computational unit using RRAM's memory arrays. This analysis of newer hardware yielded a 25% less energy usage and a 57% higher performance rate as compared to older traditional hardware devices. However, this was performed on a small-scale architecture, therefore, it could not provide enough concrete data to reach a valid generalization. Likewise, Sheri et al. (Sheri, Rafique, Pedrycz, & Jeon, 2015) and Eryilmaz et al. (Eryilmaz et al., 2016) utilized RRAM devices but found that RRAM devices were limited by power consumption overheads.

Nevertheless, researchers continued to determine proper implementation of neural network through RBMs that could be scaled up into larger architecture. Some researchers devised a way to increase scalability and create larger networks using Field Programmable Gate Arrays (FPGA) such as the work introduced by Kim et al. (Kim, McAfee, McMahon, & Olukotun, 2009). This is a unique method because FPGA chips can be programmed by users to do specialized functions and yield a 25 times greater performance over Central Processing Units (CPU) (Kim, McMahon, & Olukotun, 2010). However, this method has been limited by the availability of FPGA which restricts the number of neurons that can be represented (Li, Najafi, & Lilja, 2015). Instead, the researchers found themselves reverting to software, and exploiting stochastic bit streams (how the network learns). This yielded a method that did not require as

much hardware resources as before, allowing for a bigger network in the same space. However, this did not solve the problem of limited resources on FPGAs. The stochastic Complementary metal-oxide-semiconductor (CMOS) based RBM implementation proposed in (Ardakani, Leduc-Primeau, Onizawa, Hanyu, & Gross, 2017) leveraged the low complexity of stochastic CMOS arithmetic to save area and power. All these efforts were successful, but some limitations called for more research on hardware-based RBM implementations.

Present researchers devised a robust solution to the limited space problem by fundamentally changing the type of hardware used, instead of repurposing current technology. Zand et al. attempted this by creating models of Probabilistic Spin Logic devices (p-bits) (Zand et al., 2017). These devices work off the electron's spin to control current and resistance. These p-bits were designed to use a sigmoidal function for decision making. As the system learned, it was able to rapidly achieve a minimal 3.8% error, after 5000 training examples. If an actual device could be fabricated, this would mean the space in a computational unit could be smaller, allowing for more resources on the same chip. This approach gives future researchers a place to start in the effort to achieve a working hardware device that successfully implements RBM using Spin Logic devices.

Methods

This method involves the replication of the method proposed in *R-DBN: A Resistive Deep Belief Network Architecture Leveraging the Intrinsic Behavior of Probabilistic Devices* (Zand et al., 2017).

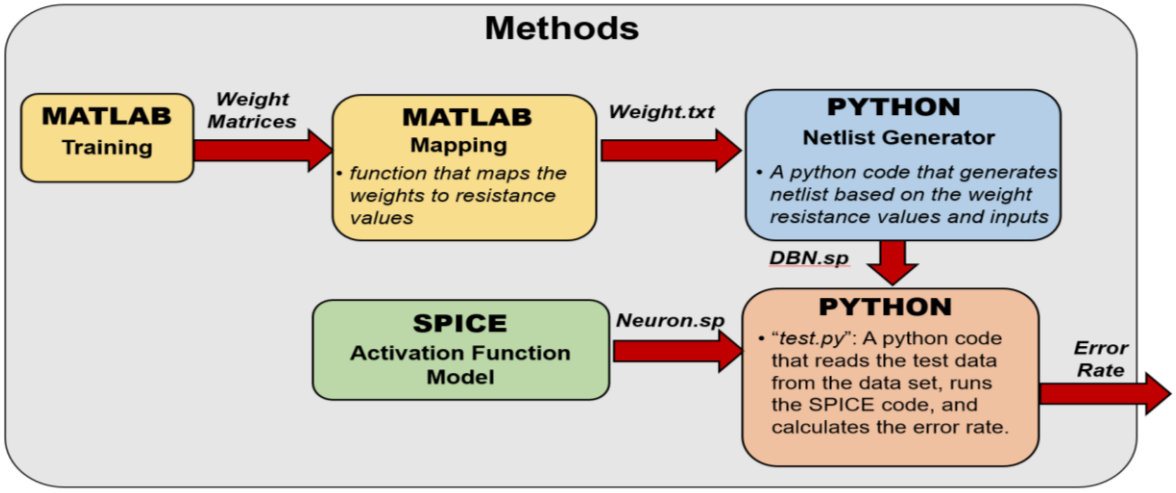

Fig. 1: Flowchart of Method

The figure above summarizes the method involved in this project. Firstly, the weights (that is a connection between the nodes, i.e., inputs and the neurons in a Restricted Boltzmann Machine, RBM network are trained using supervised training, unsupervised learning or other types of training processes. Supervised learning involves training the weights to produce the desired output from an input data while unsupervised learning involves training the weights without a specific desired output. There are other various ways to train the weights, but for this project, unsupervised training is used to train the weights. The training process in this project was done using Deep Neural Network by Masayuki Tanaka (a Matlab file), which was modified for this project's application. Th training process was done using MatLab, a programming language that allows for matrix manipulation, which produced some weight matrices, i.e. a matrix of the inputs and the neurons.

The weight matrices produced from MatLab is mapped using the MatLab software. This process involves using a function on MatLab to map the weight matrices to resistance values (this allows a circuit to be built to represent RBM network). From the mapping, a text file is

produced that contains the weight resistance. The next step involves using a Python, another programming language, code to generate Netlist, circuit representing RBM network, based on the weight resistance values and inputs (voltages). To successfully represent the RBM network, the neuron (activation function) is needed. For this project, an activation function produced by another collaborator of this project will be used. A Python code is used to call the activation function SPICE code and Netlist. Then the new Phyton code reads the test data (i.e., hundreds of input data like numbers from 0 – 9) from the data set. Then the Python code runs the called codes (activation function code and Netlist) and calculates the error rate in the actual output of the circuit and the desired output.

Expected Results

Researchers have been able to produce a working algorithm that imitates RBM using multipliers, FPGAs, RRAM, and Adders (as explained in the literature review). However, this project is expected to achieve the same result without doing as much work. Using a 28 X 28-pixel data set and neurons of 10 cases (0 – 9), the expected result for the first part of this project is matrices of 784 rows and 10 columns produced by MatLab, which equates to 7840 resistance values in the text file produced using MatLab for mapping. Then, the circuit produced using Netlist generator on Python should have voltages and resistance that represents the input and neurons of the RBM network. Using the activation function and circuit, the python code for the final step should be able to read test data from dataset provided and calculate error rate (the number of right outputs based on inputs).

If the implementation of the Python code produces an error rate of less than 5%, the conclusion is that a hardware-device can be made from the circuit produced from this project

thereby confirming the hypothesis that it is possible to have a hardware-based implementation of RBM that imitates the neurons in the brain.

Budget

Item	Reason	Quantity	Cost
Lamdha Quad Graphic Processing Unit • Ubuntu 16.04 Operating System • NVIDIA GeForce GTX 1080Ti GPU • 64GB DDR4 RAM • 1TB SSD + 2TB HDD storage • 1 GB Ethernet network • One year warranty	For a faster running time for the data set and simulations. Faster computers reduce the timeline of implementation from 6 months to about one month.	1	$8693
MatLab License • Neural Network Toolbox	For simulations and implementation	1	$1000
			Total: $9693

Timeline

Month 1: Modification of Tanak Model and application in MatLab (2 weeks)

Mapping – function in MatLab that maps the weights to resistance values (2 weeks)

Month 2: Netlist Generator, python code (2 – 3 weeks)

Month 3: Implementation of code generated in Python to read data sets (1 month)

Month 4: Adjustment of the parameter if the error rate is above 5% (2 weeks)

Month 5: Implementation of code generated in Python to read data sets (1 month)

Month 6: Adjustment of the parameter if the error rate is above 5% (2 weeks)

Month 7: Implementation of code generated in Python to read data sets (1 month)

Month 8: Working code that implements RBM using Beyond-CMOS technology

Note: Implementation of code might take up to 6 months depending on the speed of the computers used.

References

Ardakani, A., Leduc-Primeau, F., Onizawa, N., Hanyu, T., & Gross, W. J. (2017). VLSI implementation of deep neural network using integral stochastic computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 25*(10), 2688-2699.

Bojnordi, M. N., & Ipek, E. (2016). *Memristive Boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning.* Paper presented at the High-Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on.

Eryilmaz, S. B., Neftci, E., Joshi, S., Kim, S., BrightSky, M., Lung, H.-L., . . . Wong, H.-S. P. (2016). Training a probabilistic graphical model with resistive switching electronic synapses. *IEEE Transactions on Electron Devices, 63*(12), 5004-5011.

Fahlman, S. E., Hinton, G. E., & Sejnowski, T. J. (1983). Massively parallel architectures for Al: NETL, Thistle, and Boltzmann machines. *Proceedings of AAAI-83109, 113*.

Kim, S. K., McAfee, L. C., McMahon, P. L., & Olukotun, K. (2009). *A highly scalable restricted Boltzmann machine FPGA implementation.* Paper presented at the Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on.

Kim, S. K., McMahon, P. L., & Olukotun, K. (2010). *A large-scale architecture for restricted Boltzmann machines.* Paper presented at the Field-Programmable Custom Computing Machines (FCCM), 2010 18th IEEE Annual International Symposium on.

Li, B., Najafi, M. H., & Lilja, D. J. (2015). *An FPGA implementation of a restricted Boltzmann machine classifier using stochastic bit streams.* Paper presented at the Application-specific Systems, Architectures, and Processors (ASAP), 2015 IEEE 26th International Conference on.

Sarikaya, R., Hinton, G. E., & Deoras, A. (2014). Application of deep belief networks for natural language understanding. *IEEE/ACM Transactions on Audio, Speech, and Language Processing, 22*(4), 778-784.

Sheri, A. M., Rafique, A., Pedrycz, W., & Jeon, M. (2015). Contrastive divergence for memristor-based restricted Boltzmann machine. *Engineering Applications of Artificial Intelligence, 37*, 336-342.

Zand, R., Camsari, K. Y., Ahmed, I., Pyle, S. D., Kim, C. H., Datta, S., & DeMara, R. F. (2017). R-DBN: A Resistive Deep Belief Network Architecture Leveraging the Intrinsic Behavior of Probabilistic Devices. *arXiv preprint arXiv:1710.00249*.