# Project Proposal

**Kyle Daruwalla**
Department of Electrical and Computer Engineering
University of Wisconsin – Madison
daruwalla@wisc.edu

**Akhil Sundararajun**
Department of Electrical and Computer Engineering
University of Wisconsin – Madison
asundararaja@wisc.edu

## Abstract

*Insert abstract.*

## 1 Introduction

Machine learning systems tackle problems ranging from content filtering and recommender systems to object recognition and text classification. Solving these problems involves finding classifiers that can detect patterns in input data. In learning of deep neural networks, feature extraction is performed automatically by using many layers of representation; each layer involves passing inputs of the previous layer through a nonlinear activation function. Compositions of successive layers can enable learning of classifiers corresponding to nonlinear decision boundaries, and this has led to recent successes in image classification and speech recognition.

### 1.1 Field-Programmable Gate Arrays

Even though learning algorithms are inherently serial, speedup might be possible by using specialized hardware to reduce the cost per iteration.

Field-programmable gate arrays (FPGAs) are reconfigurable hardware units. An FPGA is comprised of *slices*, which are the fundemental hardware unit from which any designed hardware is constructed. Each slice is comprised of *look-up tables* (LUTs) and *flip-flops* (FFs). When reporting the resource consumption of a particular design, it is common to report the metric in terms of slices or LUTs+FFs.

Hardware on an FPGA is designed using a *hardware description language* (HDL). The most common HDL is Verilog. While Verilog shares some syntax with C, it should not be confused for a sequential programming language. HDLs allow a designer to spatially describe the hardware.

FPGAs are commonly used for real-time control, because the design freedom they offer allows for lean, efficient controller design. Furthermore, designs are not hampered by hardware limitations, because the designer can create any hardware he desires. As the boundary between control theory and optimization has blurred, FPGAs have become suitable hardware platforms for machine learning algorithms such as neural networks [1] [2]. Similarly, FPGAs are an attractive option to make object-recognition algorithms real-time [3].

While previous work has largely focused on deployment of neural networks on FPGAs, this project will focus on the training phase. Specifically, can FPGAs be utilized to build efficient parallel hardware to speedup the lengthy training process for convolutional neural networks?

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

## 2 Problem Definition

*Comparing Hardware Platforms* We seek to compare training deep neural networks on different computing platforms. The first aspect of the comparison will be to study training performance of the ImageNet dataset on Amazon EC2, which will serve as a baseline for a traditional single-machine CPU setting. We then propose to design an FPGA hardware system on which to train deep neural networks. Potential speedups in training the ImageNet database will be investigated in the FPGA system.

TensorFlow *Varying CNN Structure and Activation Function* Due to the variety of structural and parameter design choices typical in building a CNN, we propose to examine several CNN architectures in combination with the different hardware platforms considered in this project. Several activation functions including sign, ReLU, and sigmoid will be explored.

*Efficacy of Parallelization* Furthermore, as an extension to the single-machine CPU setting, GPU acceleration on Amazon ec2 will be studied using TensorFlow. Due to TensorFlow's distributed execution capability, we also propose to train CNNs using the Hogwild! algorithm.

## 3 Proposed Implementation

In order to perform an effective comparison between software implementations of CNNs and hardware implementations, we propose using TensorFlow, Amazon EC2, and Xilinx FPGAs. TensorFlow will be used to implement various software implementations of a given CNN structure for ImageNet. These CNN implementations will be trained and tested on Amazon EC2. A hardware implementation of the CNN will be created for Xilinx FPGAs in Vivado. Simulations will be performed to verify its functionality, and it will be deployed on a physical FPGA to test its timing characteristics and accuracy.

### 3.1 TensorFlow on EC2

*Information on TensorFlow implementation on EC2. Talk about CPU baseline. Talk about speed up using GPU and Hogwild!*

### 3.2 Neural Networks on FPGAs

Each filter in the CNN will be modeled as a *unit-neuron* on the FPGA (shown in Figure 1). During the compute phase, the selector signal $s$, will feed the current patch $(x_0, x_1, x_2, x_3, x_4)$ into the unit-neuron. A weight register file will hold the current weights, $(w_0, w_1, w_2, w_3, w_4)$. The activation function, $\sigma$, will be approximated using a lookup table if it is not piecewise linear. The output $f$ will store a single pixel of output for a given filter.

A controller will adjust $(x_0, x_1, x_2, x_3, x_4)$ so that it corresponds to the current patch being evaluated. After the compute phase is complete, it will update the $(w_0, w_1, w_2, w_3, w_4)$ values and drive $s$ high so that the weight register file can be updated. There will be latching (not shown in Figure 1) on the output values of the filters so that they can be held while the weights are updated.

The potential for speedup comes from parallelizing the filter operation, using faster fixed-point computation units, and approximation of the activation function.

## 4 Proposed Analysis

After creating an FPGA implementation of a neural network, timing characteristics for a particular CNN can be created. For example, the proposed unit-neuron will have some physical path delay, $\tau$. Using this known constant and the structure of our CNN, we would like to define a design space. This will allow a designer targeting neural networks on FPGAs to know the time per iteration as a function of $\tau$, the number of layers, and the CNN structure. This will provide some theoretical closed form for the computational complexity on an FPGA.

Additionally, after collecting empirical results, we would like to do perform the following analysis:

1. Comparing generalization error between the CPU, GPU, Hogwild!, and FPGA implementations.
2. Comparing convergance rates between the CPU, GPU, Hogwild!, and FPGA implementations.
3. Provide a metric of when FPGAs might provide a larger speedup than Hogwild!

## References


[1] J. Wang, Y. Chen, J. Xie, B. Chen, and Z. Zhou, "FPGA based neural network PID controller for line-scan camera in sensorless environment," Fourth International Conference on Natural Computation, 2008.

[2] J. Skodzik, V. Altmann, B. Wagner, P. Danielis, and D. Timmermann, "A highly integrable FPGA-based runtime-configurable multilayer perceptron," IEEE 27th International Conference on Advanced Information Networking and Applications, 2013.

[3] B. Ahn, "Real-time video object recognition using convolutional neural network," International Joint Conference on Neural Networks, 2015.

[4] F. Niu, B. Recht, C. Ré, and S. J. Wright, "Hogwild!: A lock-free approach to parallelizing stochastic gradient descent." arXiv:1106.5730v2, 2011.

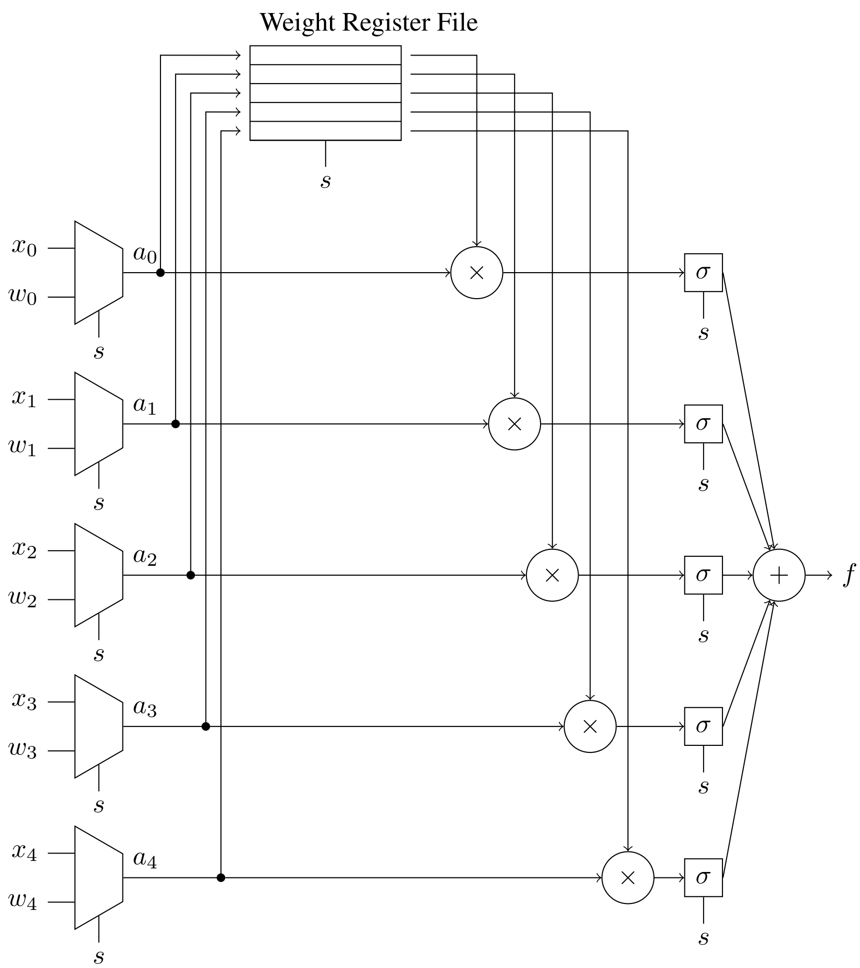


Figure 1: A unit-neuron implementation for an FPGA with a filter size of 5