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

Include brief information on setting up the CNN problem.

1.1 Field-Programmable Gate Arrays

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?

2 Problem Definition

Set up the goals of the project.

3 Proposed Implementation

Brief overview of implementation.

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, σ, 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.

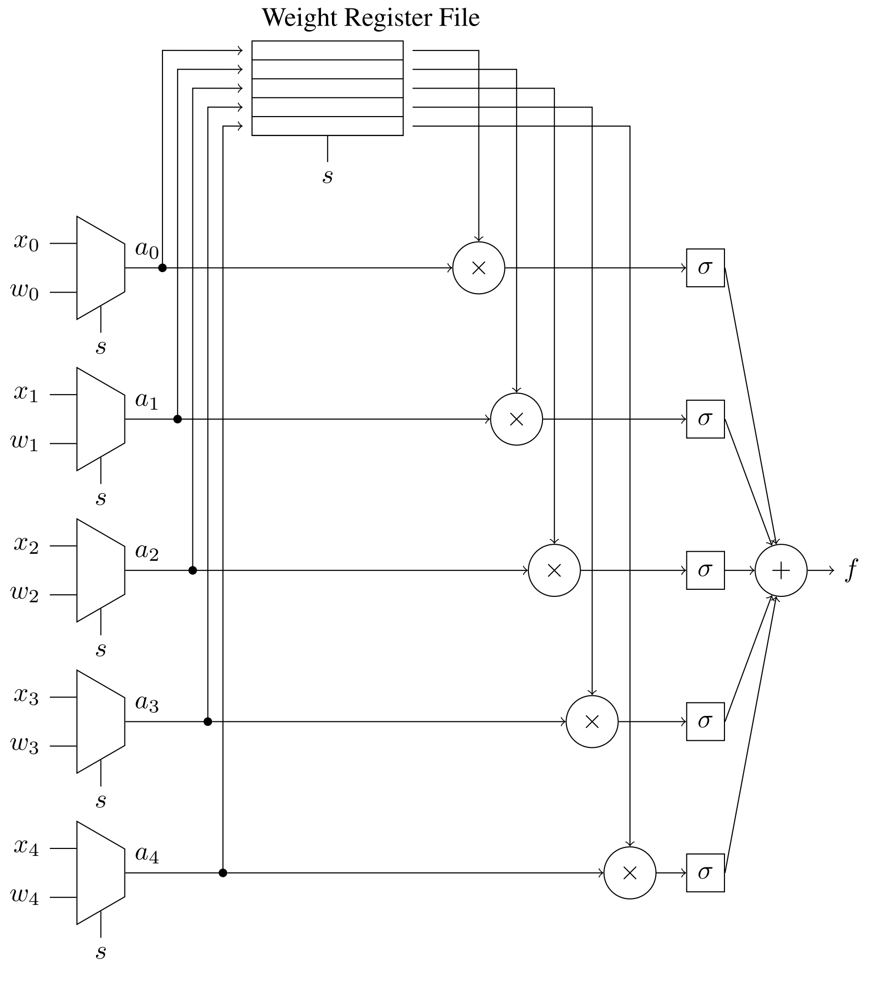

Figure 1: A unit-neuron implementation for an FPGA with a filter size of 5

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

Talk about analysis that we are targeting.

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.