hls4ml

Caiden Swanson

William-Rainey Harper College CCI Intern

July 19, 2024

Abstract:

hls4ml (high level synthesis for machine learning) Is a Python package used to translate commonly used open-source machine learning models into HLS. This is useful in machine learning applications on FPGAs. Machine learning algorithms are only as fast as the hardware that they are used on, and some applications require high speed without sacrificing accuracy. In these situations, an FPGA is a good choice since it is faster than a CPU or a GPU, but programming an FPGA is difficult. This is where hls4ml can be used to simplify the process, as a well-known learning model can be converted to HLS and more easily deployed onto an FPGA. There are many use cases for a machine learning algorithm running on an FPGA. For example, detectors in a particle accelerator cannot keep every event that they detect, and so a computer must decide which events to keep and which to discard. Using an FPGA with a machine learning algorithm would be a good way to keep as many events as possible.

Features:

hls4ml has support for three different machine learning frameworks. Qkeras/Keras is a user friendly and easily modifiable framework. With Qkeras being an extension of Keras. This is the most supported framework, but hls4ml also supports PyTorch, which is useful for deep learning algorithms, which are written in a way that resembles the way that the human brain works. ONNX and QONNX are both still in development but will eventually be fully supported.

The supported neural network architectures are fully connected neural networks, convolutional neural networks, recurrent neural networks, and graph neural networks. Fully connected neural networks are critical in deep learning neural networks. They are called “fully connected” because each node is connected to every node in the previous layer. A convolutional neural network is a lighter model than a fully connected network. In the case of processing an image, a convolutional neural network would simplify the image into chunks to make processing require less resources, and then use a fully connected layer to make a final prediction. A recurrent neural network works by processing information, and then passing the output to the next layer. This is useful for text-based machine learning models where the previous words in a sentence are used to predict the next word of the sentence. A graph neural network has neurons arranged in the shape of a graph data structure, which has nodes and edges. This neural network is used for analyzing any data stored in a graph structure. One example of data that would be good to store in a graph would be the structure of a complex molecule.

hls4ml supports 3 HLS backends, Vivado HLS, Intel HLS, and Vitis HLS. Vivado is much more focused on designing hardware, where Vitis uses a software-based approach to design hardware or software. To be more specific, FPGAs are good for prototyping circuits since they are much faster than GPUs and CPUs and can still be modified, whereas an ASIC cannot. Vivado is focused on this application of an FPGA and does not allow you to deploy software onto an FPGA. Vitis can be used to write software or hardware for an FPGA, but it uses a software approach for both. Vitis and Vivado each have use cases where one is superior to the other, but in the case of hls4ml, they are essentially the same since they are just being used by the software, so Vitis is mostly replacing Vivado since it is newer. Intel HLS is the third backend and is unrelated to Vitis and Vivado, which are both for programming AMD FPGAs. It is the main competitor of the other two and is used for programming Intel FPGAs.

How it works:

To understand the purpose of hls4ml, you must first understand the advantages of working with FPGAs. FPGAs are chips which can be programmed to perform a certain task. Once programmed to do this task they are faster than both CPUs and GPUs, with the major drawback of being far less flexible than both. In the case of Neural networks, FPGAs are ideal since they don't need to be constantly reprogrammed, and they can evaluate a neural network much faster than CPUs or GPUs while also using much less power. However, converting a neural network to a hardware design language or high-level synthesis code is time consuming, which is where hs4ml is helpful.

The input that hls4ml begins with is a multilayer neural network. Each layer contains some number of neurons, which as mentioned earlier may be connected in a variety of different ways. An output value must be calculated for each neuron in the layer. hls4ml calculates the output values in each layer independently of one another in order. Pipelining is used to speed up this process.

There are several metrics that can be modified by the user to optimize their individual model to suit their needs. The size of the model can be specified to best use the FPGA's resources. The precision of the individual calculations can also be set, which improves processing times. The user can also specify a reuse factor, which decides how many times a multiplier is used to compute the output values in a layer. This is useful again to preserve resources at the expense of low latency and throughput.

hls4ml has a frontend, and a backend. The front end parses the Neural Network into an internal graph model. The frontends are the parsers for the machine learning frameworks. The backends are the HLS compilers. In summary the front end is used to take the input, and the back end is what determines what kind of output is produced.

hls4ml supports multiple ways to pass data between the layers of the neural network. The way that the data is handled is called the I/O type. There are two different options for how to pass the data. The first is io_parallel, which as the name suggests passes the data in parallel between layers. This is good for MLP networks. MLP stands for multilayer perceptron, which is an older Neural network that is still widely used as part of more complicated networks. Io_parallel can also be used for small convolutional neural networks, but synthesis will generally fail when the network is large. The other option is io_stream, which is where data is passed one piece at a time, and it is most useful for large convolutional neural networks. When io_stream is used, each layer is connected to the next using FIFO (First in First Out) style buffers, which helps to use resources more efficiently. The minimum depth needed for each step is hard to determine, and so by default hls4ml uses the largest possible minimum depth. However, the depth of the buffers can be modified by the user to suit their individual use case.

Testing:

The hls4ml package contains many tests to make sure that hls4ml is working properly. Some of the tests are bash scripts, but most of the tests are python scripts. The non-synthesis tests are currently all Python. The tests serve many different purposes and range in the number of components they contain, with some having only a couple of items while others contain many more. The Python tests are run in a Python environment using Pytest, which is a framework used to make simple and easy to read tests. Pytest is useful because it offers detailed information about why and where the test failed, which can help to narrow down what the problem may be. Although synthesis cannot be performed on all tests since it would take too long, the current synthesis tests are depreciated, and are not compatible with all of the backends that they need to be compatible with, so currently we are adding synthesis to some of the python tests, which is what I have been working on. My first job was to add a synthesis to one of the python tests. Doing this was simple and just involved adding a few lines of code. However, synthesis does not have to succeed for the test to pass. If the synthesis fails, the function simply doesn't return the results, but this doesn't raise an exception, and so the test still appears to pass, which is not very helpful. The build function which is used to perform synthesis returns a dictionary containing the results of the synthesis. If the synthesis fails it only returns the results for the c simulation, so I added an if statement which takes a dictionary and checks if it contains all the keys that it should contain were synthesis to succeed. If the dictionary does not contain all the keys an exception is raised informing the user that synthesis has failed. However, whether the synthesis succeeded or not is not the only thing we are interested in testing. We also would like to test the results of the synthesis against a baseline. The goal is for the code maintainers to be able to decide what directory the baseline synthesis results will come from, and then for Jenkins to automatically run both the synthesis test and the comparison between the two. Currently the comparison simply checks if the dictionaries are identical or not, but eventually it may test to see if the results are within a certain range of the baseline results. Currently I am trying to figure out how to include the name of the directory where the results will be stored in the Jenkins file, and I am also working on making it configurable. Once that is done, I will work on making the comparison test more advanced and making it so that the script can take an argument from Jenkins since the comparison test will require the name of the results directory to retrieve the baseline results.

References

Fast Machine Learning Lab. (2024). *Hls4ml*. Retrieved from Hls4ml: https://fastmachinelearning.org/hls4ml/index.html

Holger Krekel and Pytest-Dev Team. (n.d.). *pytest: helps you write better programs*. Retrieved from Pytest: https://docs.pytest.org/en/stable/

Intel High Level Synthesis Compiler. (n.d.). Retrieved from Intel: https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/hls-compiler.html

Vitis HLS. (n.d.). Retrieved from AMD: https://www.xilinx.com/products/design-tools/vitis/vitis-hls.html

This manuscript has been authored by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy, Office of Science, Office of High Energy Physics.

This work was supported in part by the U.S. Department of Energy, Office of Science, Office of Workforce Development for Teachers and Scientists (WDTS) under the Community College Internship (CCI)