Contents

Part I:
Introduction

Neural network training and inference in embedded environments

Estimates put the number of tiny embedded systems devices north of 20 billion (attach reference) and the potential of running machine learning applications on that compute is enormous. Furthermore the notion of utilising the existing embedded infrastructure for the purpose of performing ML compute as a means for achiever greater utilisation and as opposed to deploying new specialised devices for those applications has an appeal from a sustainability standpoint. However much of the potential of running machine learning appilations on these devices remain unattained due to the difficulties in creating these applications ...

Among the approaches that would be salient on these platforms, neural network approaches are the most sought after owing to the unprecedented progress made in their practical applications. Tiny Machine Learning is a burgeoning field that looks at how this space of embedded devices can be made more suitable to create and explore the potential machine learning applications that it can support. An important feature of machine learning applications are their iterative improvement process. For neural network applications this happens during the training process which traditionally consumes a lot of compute resource

Why perform training and inference on ECUs?

In the world of embedded systems resources such as compute, memory, network bandwith etc. are all limited. The traditional model of sending data from embedded device sensors off-board to compute clusters on the cloud presents several challenges such as bandwith consumption, privacy considerations, and more that makes it attractive to perform both training and inference on-board the embedded device

Federated Learning One approach to making this training loop take place from within these platforms is Federated Learning which cruicially allows for the data to remain on the device

1. Background

Describe ECU systems, Tiny ML, Anomaly Detection, Yocto Project etc

Hypothesis: Training and inference of (small) neural networks in embedded systems can be considerably improved compared to general purpose neural networks frameworks

The space of salient applications for automotive embedded systems is enormous with examples such as anomaly detection within an automobile, a subcomponent of the automobile, or with the interactions between subsystems

- *Introduce general information about artificial neural networks (ANNs), MLOps, etc - state that there is more information in the Theory chapter*
- *Introduce how an anomaly detection application could be run*

1.1 Anomaly Detection On Board

Introduce how the ANN application would be executing on the automobile

1.1.1 MLOps On Embedded Systems

Nature of (CAN) data generated on ECU systems and how they could be consumed - described from an MLOps viewpoint

1.1.2 Considerations Of Embedded Environments

- *State hardware requirements within the context of the ANN functionality*
- *Express intent to benchmark the training phase. State the Motivation*

1.2 Development For Embedded Linux

Introduce build systems for embedded linux. Motivate the section in terms of targetting embedded hardware

1.2.1 The Yocto Project

Outline the Yocto Project Build System

1.3 Development Of Neural Network Application

Contrast general purpose frameworks - TFlite etc with handwritten applications

1.3.1 Different Programming Paradigms

Approaches to doing Machine Learning in Embedded Environments. Emphasis on how these applications are developed - e.g TFLite

2. Theory

Describe the arrangement of this chapter

2.1 Artificial Neural Networks

General introduction to ANNs. Explaining topics from inference, training, till federated learning systems

- *Explain the different ways of building out the ANN applications - Training on board vs off board, associated factors such as uploading data vs learned model*
- *Compare training with inference*

2.2 ANN Performance Optimisations Techniques

Contrast traditional implementations in resource rich environments and the constraints of embedded environment. Layout general strategies to acquire performance improvements with little losses to accuracy - Purning, Quantisation. State the emphasis on training

2.3 ARM's CMSIS-NN

Introduction to ARM CMSIS-NN kernels, mentioned again in Development chapter

2.4 Performance Evaluation

Describe and motivate performance measures used in the Results chapter

Part II: Implementation

Developing multiple benchmark ANN programs for the i.MX6 target

3. Design

Motivate and describe the HDRNN based ANNs used to benchmark training

3.1 ANN Development Process

- *Toolchains and Yocto recipes*
- *State target board exploration explored in later chapter*

3.2 i.MX6 Processor

Information about the processor in general - ISA, Memory, Clock . . .

3.2.1 i.MX6 as an ANN application target

Describe performance optimisations possible using NEON(SIMD) . . .

3.2.2 Available Optimisation Frameworks : CMSIS-NN

Describe ARM's CMSIS-NN as a framework worth adding

3.3 Benchmark ANN - HDRNN

- *Motiviate using HDRNN to measure training performance*
- *Describe the architecture of the HDRNN*

MNIST database is the standard benchmark for ANN applications. ANN heavily derived from neuralnetworksanddeeplearning.com. Our primary target was to evaluate the training phase of an ANN.

3.3.1 The Learning Algorithm

- batched SGD

link with Theory chapter content

4. Development

- *Detail the exploration on i.MX6 based board*
- *Describe the HDRNN implementations*

4.1 Targetting an i.MX6 based custom board

4.1.1 Testing on Device

Process for testing on device - flashing, benchmarking (perf)

4.2 HDRNN Implementation

Motivate why HDRNN via MNIST dataset was chosen

With the primary focus on training, MNIST dataset was primarily loaded in an easily readable format appropriate to the corresponding paradigms

4.2.1 Python Numpy based HDRNN

4.2.2 Tensorflow Lite based HDRNN

- Keras based, easy to write
- TFLite build less easy, still straightforward

4.2.3 C based HDRNN

4.2.4 CPP based HDRNN

4.3 CMSIS-NN based Optimisations

4.4 General Distribution of Work

Part III: Analysis

Results from the implementation

5. Results

The performance of the different neural networks on the i.MX6 processor

5.1 HDRNN comparisons

- *State the reasoning behind comparisons of HDRNN implementations - e.g same accuracy when initialised with the same random weights . . .*
- *Describe the performance measure considered - execution times, network accuracy, power usage . . . motivated in Theory chapter*

5.1.1 Python Numpy based HDRNN

Performance of textbook HDRNN using Python & Numpy

5.1.2 Tensorflow-Lite based HDRNN

Performance of a similar network on Tensorflow lite

5.1.3 C based HDRNN

Performance of a similar network written in C

5.1.4 CPP based HDRNN

Performance of a similar network written in CPP

5.2 CMSIS-NN based Optmisations

Further breakdown of the performance achieved from different optimisation techniques

5.2.1 Quantisation

future: Training Network with Quantized weights

5.2.2 Pruning the Network

future

6. Discussion

- *Contrast development process for the ML programming paradigms*
- *Which optimisation approaches gave the most in improvement?*

7. Conclusion and Future Work

What does it all mean? Where do we go from here?

References

[1] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Shlomi Regev, Rocky Rhodes, Tiezhen Wang, and Pete Warden. Tensorflow lite micro: Embedded machine learning on tinyml systems, 2020.