

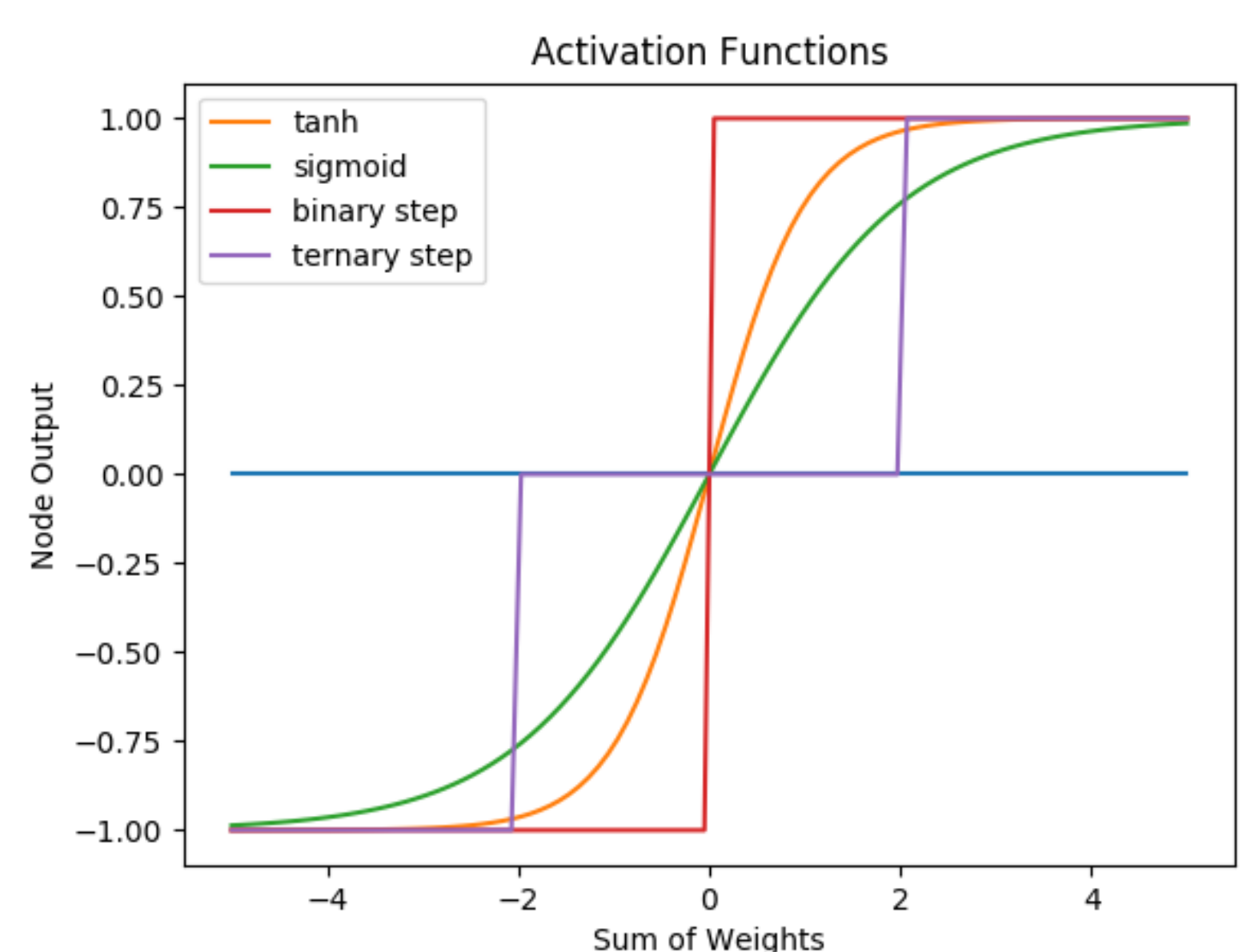


Optimising Neural Networks for Embedded Systems

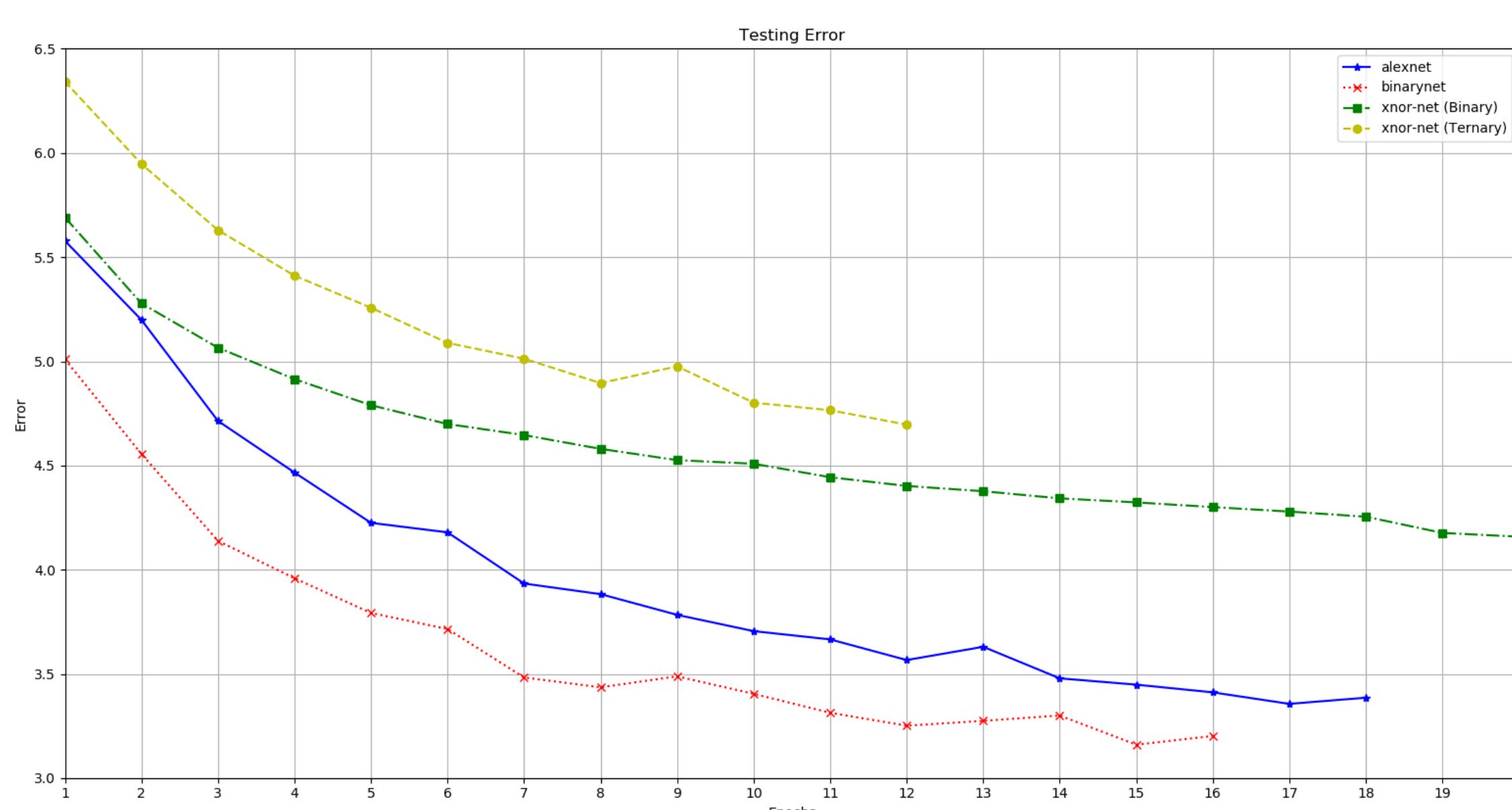
Supervisor: Prof Tom Drummond

Current implementations of Artificial Neural Networks and other Machine Learning algorithms are designed to run on dedicated systems with large quantities of memory and parallel computation cores (e.g. GPUs). However these requirements are beyond the current capacity of many embedded systems such as mobile phones, microcontrollers, and IoT connected devices. Current solutions to this problem focus on the concept of "Quantising" specific aspects of neural networks to allow them to train, test, and function faster on all systems including embedded devices. This project aims to provide a method of quantising specific networks aimed at image classification by changing the weights and activation functions of certain well-known networks like AlexNet and Inception from floating point numbers to integer based numbers. The main advantage of this change is the ability to use bitwise operations on specific layers instead of floating point operations that require large amounts of processing power.

The basic functionality of a neural network comes from the activation functions. Each neuron in a network sums the weighted inputs to itself and applies an activation function which creates the output value for that node. On a relatively large network there are hundreds of thousands of nodes, each with an activation function that takes thousands of inputs, quite quickly the amount of computing power needed can reach super-computing levels. Most activation functions are non-linear (sigmoid, tanh, etc.), by using non-linear functions networks have the ability to model problems that are themselves non-linear. These functions require floating point precision to be accurate, however by approximating these functions with binary, and ternary steps the required floating point computation can be reduced. By only applying this optimisation to specific "hidden" layers the overall efficiency of the network can be increased at the cost of some accuracy. The game then becomes optimising this loss enough that the network can still perform better at specific tasks than humans. This optimisation is possible because the step functions follow the shape of the non-linear functions enough that they can emulate them over enough iterations.



There are many different frameworks available to create machine learning algorithms. Most early networks were written in Lua using the torch framework, however, current advances in machine learning have led companies to develop their own in house solutions which have gained widespread adoption. The Google built Tensorflow framework works with languages including Python, Java, and C++. It also has many advanced features including a large library, a cloud service for running distributed networks, and a visualization tool called TensorBoard.



By changing several middle layer activation functions, the accuracy of the networks fell, as expected, however the training speed of the network increased, with the average speed of training alexnet being 3 seconds and the improved ternary network training in 2 seconds. By using this technique, it is possible to load networks designed to run on large x86 machines onto smaller ARM computers such as the Raspberry Pi, and Android mobiles. By using three layers instead of two in the ternary function the number of operations compared to a binary function is the same because the third level of zero requires no resources to compute, however, it adds an extra level of precision for free. This allows the network to become more accurate for little to no cost. This technique can be expanded in the future to allow extra discrete levels of computation for more accuracy with less resources being used.

