Using neural networks to approximate complex algorithms

Kevin Chen Trieu J11803099002
Petros Debesay J11803099003

January 2018

1 Introduction of the paper

This report is on the paper "On Quality Trade-off Control for Approximate Computing Using Iterative Training".
Given large sets of training data of hand drawn numbers, a neural network is able to identify the number on these images to a relatively high degree of accuracy. This paper utilizes this property to identify the results of algorithms, given large sets of training data of the parameters of an algorithm and the subsequent output, given any input parameters for the algorithm can the neural network approximate the output to a good enough accuracy? And the following question is then asked, how can we ensure quality control without running the original algorithm?

Certain algorithms are very complex to calculate. Some of them belong to the set of NP-complete problems where there is no known polynomial time algorithm for finding the solution. Other problems, there are polynomial solutions but they still might be too slow for our demands. For these problems, approximation algorithms must be developed to at least find a reasonably good solution in a reasonable time frame. The training of the data is very time consuming and so is the generation of the data. To generate the data first hand, the algorithm must be run enough times to generate a large enough training data. However, the idea is that this time will be gained back in the inference. This training data is then used to train the neural network and the neural network is used to predict the result of the algorithm. We label this network the accelerator.

All data cannot be accurately approximated. There needs to be some quality control, to asses if the data approximated is assumed to be close enough to the actual solution. We cannot run the original algorithm to make sure that the approximation is good enough, since that would constitute a running time greater than just running the original algorithm from the beginning. To assess whether the accelerator can approximate a given input, another neural network

is trained. We call this network the classifier. The classifier tells, given an input, if the output of this input can be approximated with our trained neural network to a good enough accuracy. Here the issue lies. If the classifier says that the accelerator cannot be used to a high degree this results in no efficiency gain. So, while the error of the accelerator has to be minimized, the percentage of times it is fine to run the accelerator has to be maximized. This percentage is called the invocation. With a low invocation, this process is extremely inefficient. We can tune the accepted error to reach a higher invocation, but an approximation with high degree of error is essentially useless. So this paper also deals with finding the best invocation rate for a given desired threshold.

2 Experiment result on improving efficiency

The architecture takes longer time to train than the original architecture due to the iterative method used. We wanted to see if we could lower the overall training time while still achieving good efficiency. We used the same topology as theirs and changed the epoch size to 20 and used different batch sizes. We noticed that with smaller batch sizes the invocation got higher and the mean relative error decreased which can be seen in figure 1 below. For large batch size (512) the invocation reduced drastically and the mean relative error is lower but this is due to the low invocation of C for batch size 512.

The time for one iteration also increased with smaller batch sizes as shown in figure 2. Therefore it is not trivial to conclude what the optimal batch size is. It ultimately depends on what is desired. If faster training is wanted, increasing the batch size would be preferred, but if faster inference is wanted then a smaller batch size would be better due to more invocations.

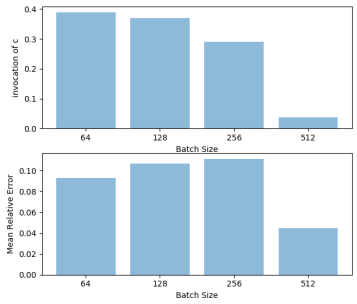

Figure 1: Average invocation and mean relative error

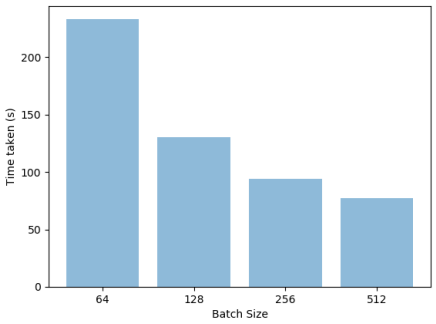

Figure 2: Time taken for one iteration

3 Experiment result of new algorithm

In order to test the paper, we choose an algorithm which solves the Quartic Function:

$$ax^4 + bx^3 + cx^2 + dx + e = 0$$

We generated 100 000 input and output data. Out of that, 70 000 was training data and 30000 was test data. Our accelerator was almost 22 percent faster than the original method. On an average we got 10 percent invocation of the classifier. If we increase the error bound to between .3-.5 then the invocation of classifier increases up to 99 percent. If we increase the learning rate .1 to .3 or .4 the invocation rate also increases. But after that if we still keep increasing the learning rate then we see there is tendency of lower invocation rate. Our accelerator gives better result when there is one parameter in the output. The invocation rate goes down from 11 percent to 4 percent when there is 4 parameters in the output. We also compared our results with the existing algorithms and the result was quite satisfactory.

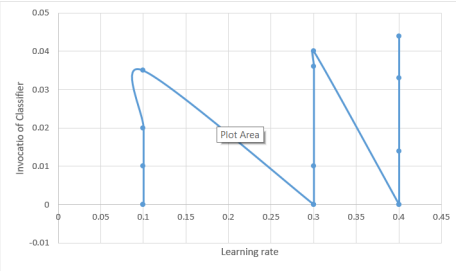

Figure 3: Learning rate vs Invocation of Classifier

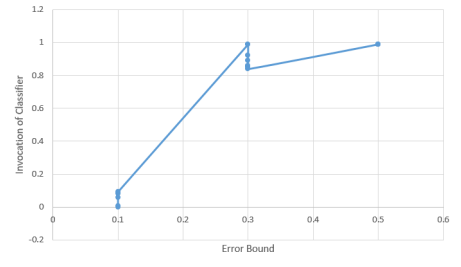

Figure 4: Error bound vs Invocation of Classifier

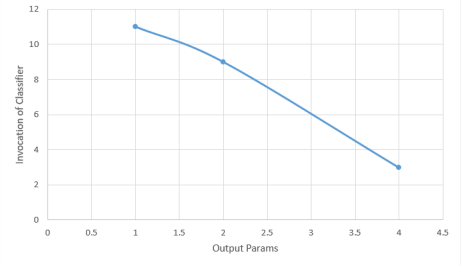

Figure 5: Output param vs Invocation of Classifier

If the error bound is .1 and if we consider 1 parameter as output for our new quartic function, then the evaluation of our method is almost 55 percent while

with similar configuration blackscholes gives an evaluation of 65 percent. And that is because our new quartic equation is much more complex than that of blackscholes.

4 Summary

In summary the paper shows a promising architecture for approximating algorithms. The benchmark used for their results may not be the best examples of real-world scenarios, although they were sufficient to compare against other architectures used for approximation. Our experiments shows that it is possible to reduce the time for training but there is no optimal choice as a certain batch size has a trade-off between time-efficiency and approximating quality. We also tried a more complex algorithm and showed that the trained network is faster than using the original algorithm. So in some real life cases, this property might help to reduce the computation power and improve the computational speed.