Summer 2021 UROP

Exploration and Further Development of the fpgaConvNet Simulated Annealing Optimizer

Michal Palič

Imperial College London

EEE department

Supervised by:

Mr. Alexander Montgomerie-Corcoran

Dr. Christos Savvas Bouganis

Introduction

At the time of my induction to the project, the fpgaConvNet framework relied on simulated annealing as its main method of optimisation. Simulated annealing was an effective tool when applied to the problem at hand, but there were also significant problems. The sample simulated annealing parameter values, generalized poorly across various standard neural networks of various sizes. The main goal of my UROP thus evolved to become an investigation into using various methods to estimate the value of these parameters. An accurate estimation of these parameters would be expected to significantly decrease the number computations necessary to reach a desired degree of optimality and increase repeatability between runs of the optimiser. Both were big issues with modern deep network models, each with many parameters to be optimized. With the current implementation on modern high-performance hardware, it still may take days to approach the optimum for large networks. This problem is expected to get worse given the trend towards deeper CNN models.

Summary of notable contributions to the project:

- Building on previous work to implement and test an algorithm to estimate the optimal starting temperature. (And an attempt at its parallelization)
- Optimization of Iteration number for select CNNs
- Development of tools to automate and or otherwise simplify the investigation of relationships between the annealing parameters and FPGAconvnet performance
- Enhanced annealer cooling schedule logging

Summary of Performance metrics used

- Average throughput – As the global minimum was not always reached during a run, more than one run's throughput was often averaged, to give an indication of the performance of the annealer with the given settings
- Count – A histogram was used to visualize the distribution of the final annealer temperatures, the higher the count for higher throughput numbers, the better
- Fraction of minimum reached – First the cost of the minimum of all the visited states for all the runs in each data set was found, and then the performance for a given run was expressed as the fraction of this minimum reached. This was then averaged across all annealer runs for a given data point.
-

Starting temperature estimation

Several methods with similarities exist to attempt to solve this problem. These methods take a random sample of transitions between states and then apply an algorithm to estimate the optimal starting temperature. Two methods were investigated, the simple method of setting the initial temperature to the maximum observed difference between states proposed by Kirkpatrick et.al [2] and the Ben-Ameur method [1], both implemented and tested on the LeNet network.

The rough shape of the starting temperature performance maximum was mapped out in the order of the suggested temperatures given by the estimation methods. There exists a maximum, whose existence I justified with the following reasoning. For a higher than optimal starting temperature, too much time is spent at temperatures much higher than the maximum difference between neighbouring states. This results in no meaningful differentiation between transitions that marginally decrease performance and transitions that heavily decrease performance. Conversely, with a lower than optimal starting temperature, the algorithm approaches greedy behaviour too early.

Figure 1 below shows the existence of a performance optimum as a function of starting temperature. Each point represents the average throughput of 100 runs of LeNet.

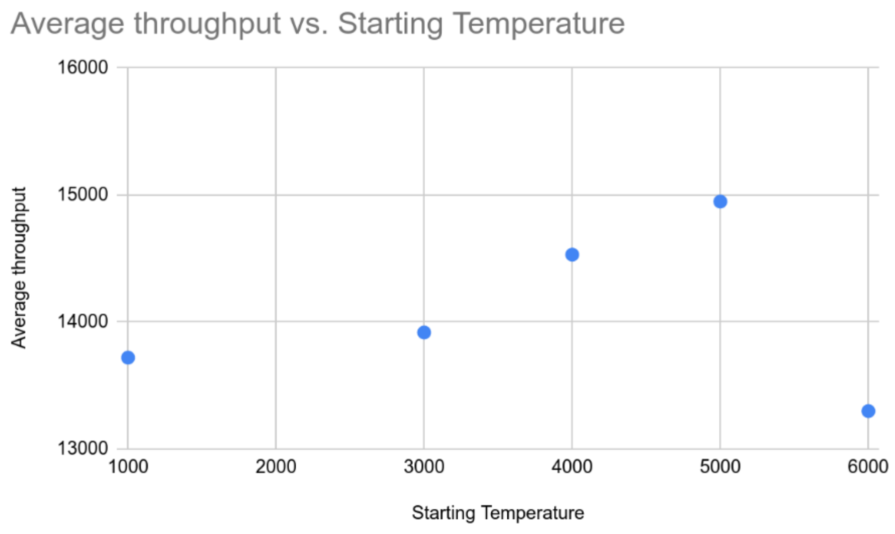

Figure 1: LeNet starting temperature optimum

In Figure 2 we can see how the distribution of throughputs looked for the starting temperature of 4000. There were only relatively small differences between the counts at each local minimum between the runs at different starting temperatures.

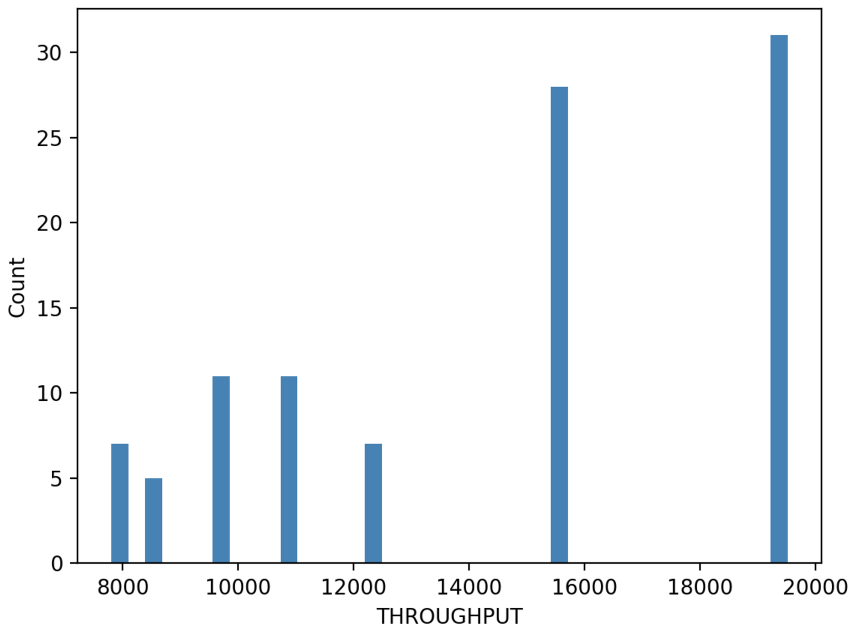

Figure 2: Example distribution for T = 4000

The Ben-Ameur [1] method relies on several random transitions being generated. If a transition leads to a decrease in performance, the performance metric value is recorded before and after the transition, in our case throughput. I call this pre and post transition value pair a sample. Each additional sample helps characterize the behaviour of network state transitions, which results in more accurate estimate of the optimal starting temperature. Once the desired number of samples is gathered, the following recursive formula is applied.

$$T_{n+1} = T_n \left(\frac{\ln \left(\hat{\chi}(T_n) \right)}{\ln \left(\chi_0 \right)} \right)^{\frac{1}{p}}$$

Equation 1: Recursive temperature formula [1]

Where $\hat{\chi}(T_n)$ is equal to

$$\hat{\chi}(T_n) = \frac{\sum_{t \in S} e^{-\frac{E_{max_t}}{T}}}{\sum_{t \in S} e^{-\frac{E_{min_t}}{T}}}$$

Equation 2: Estimate of acceptance probability

Here χ_0 is the target acceptance probability (usually set around 0.8), p is an arbitrary power whose change may improve convergence for some sets of samples. E_{max_t} and E_{min_t} are the throughput values after and before a transition respectively.

Below in Table 1, the averages for ten runs of temperature estimation are presented for the 50-250 sample range. 250 samples would account for a little under half the total annealing time.

Samples	Ben-Ameur (Average 10 runs, p = 0.8)	Ben-Ameur (Average 10 runs, p = 0.94)	Maximum (Average 10 runs)
50	3040	2551	2562.898
100	3667	5548	3561
150	2780	3855	4190
200	2483	3965	3673
250	2842	4193	3025

Table 1: Estimated starting temperatures for various sample sizes

The Ben-Ameur method [1] has additional parameters to be set, most importantly the target acceptance probability (p), i.e., the probability that a random transition decreasing performance is accepted. In the table above, this was set to 0.8. This was at the suggestion of the author but not reasoned and is presumed to be somewhat arbitrary. The results with this parameter did not entirely match the expected maximum around T = 5000. The target acceptance probability was thus raised to better match the observed plateau and the intuition that all transitions are initially accepted. It should still be noted that lots of variation is present between runs as this method uses exponential weighting and is therefore heavily influenced by the largest observed transition. Ultimately the probability parameter was set to 0.94 to bring the temperature results closer to the maximum but will likely have to be re-adjusted as its accuracy is cross-checked with other networks.

The method of taking the maximum performance decrease [2] as the starting temperature, also looks somewhat promising, but it is not expected to converge on the observed optimum of T = 5000 with a large enough sample size. For LeNet the largest observed performance improvement during annealing of LeNet was from around –10000 to around –19500. With a large enough sample of random transitions, this jump will eventually be taken in reverse, giving the maximum/estimated temperature as ca. 9500, which is not near the observed maximum of 5000.

There were also other practical problems with the Ben-Amur algorithm [1]. Maximum LeNet throughput were around 20000. The Ben-Amur algorithm requires $e^{throughput/t}$ to be calculated, and when this was done with the standard python float type, its range was exceeded. This was solved by using the arbitrary precision data type Decimal. Ben-Amur states that in practice convergence is obtained in almost all cases. This did not mirror my experience. Empirically, for a sample size of 250, the chance that a failure to converge would occur was about half. The proposed solution in the paper was to raise an intermediate value to a power higher than 1 (see note).

Note: the intermediate result was supposed to be raised to a power or 1/p, not p as stated above, which can be correctly seen in Equation 1. This was my mistake which I only noticed when editing the report. As T_{n+1} (see Equation 1) will only be less extreme as a 1/n power is applied, the precision issue is not expected to re-appear.

In my submitted code the p was fixed at one, so the implementation is correct, but instead of replacing a part of a sample when there is no convergence, a proper implementation of this power 1/p trick should be considered.

Still, it is possible that the Decimal library which was used may cause issues in the future. It is supposed to be an arbitrary precision floating point/fixed point library, but in practice, even a significant increase in the allowed exponent range did not appear to fix overflow issues. An increase in the precision of the decimal mantissa to mirror fixed point arithmetic more closely was attempted but led to impractically poor performance when capturing sufficient precision.

The implemented solution to the convergence problem was to discard a small part of the sample and re-generate it. And then attempt the calculation again. This generally worked, but more than one attempt to replace a part of the sample was required for sample sizes of 200 upward. For sample sizes of 500 it was difficult to get them to converge with this method, with many replacements required.

In case there is a wish to further investigate this, the shortest known sample list that caused a failure to converge was recorded. It has an odd property that if any further attempt is made to reduce this list and any element is removed, the convergence issue is resolved. This was tested exhaustively.

As it was noted that for larger sample sizes, the time taken to sample the transitions becomes significant compared to the actual annealer run. To offset this, an option was added into the implementation of the function which utilizes the multiprocessing module to parallelize the random search. Ultimately, this approach appeared flawed. With each process starting from an identical pre-defined state, the sample collected was less representative as each random walk was shorter and remained closer to the original state. This appeared to skew the temperature estimate downwards which was not acceptable. As such it was removed from the development branch but can still be found in the experimental line of development.

Empirical rules

With the iterations annealing parameter not being used for every problem, there is no body of work to draw on. I hypothesized that at least a meaningful estimate of this parameter could be made as a function of network size or layer number. While this goal was not achieved, the AlexNet and LeNet performance dependence on the iteration number was mapped in detail. The following results suggest that there is a large possible decrease in the computational requirements for annealing while maintaining by choosing the iteration parameter appropriately. Generally, the default configuration value of 50 was much higher than the optimum. As this relationship was investigated late in the project, no interpretation in the context of the specific problem was attempted.

The metric used to quantify the performance of the LeNet and eventually AlexNet CNNs across multiple iteration values, was the achieved percentage of the observed minimum. The value of the observed global minimum was calculated by taking the minimum cost value across all annealing steps of all runs for the network. The value for each data point in Figure 2 and 3 was taken as the average of 96 runs for LeNet and 24 for AlexNet respectively. As the number of iterations is decreased, each cooling step takes proportionally shorter. Therefore, if the number of cooling steps were kept constant, the run with the iteration parameter set to 1 would only take one fiftieth of the time compared to the maximum of 50 iterations. To make the comparison more representative, the cool parameter, describing the exponential rate of cooling, was set so that the product of the iteration parameter and the number of cooling steps between the start and final temperature was constant. This product for each point was arbitrarily fixed at 34,200. Therefore, all the runs in Figure 3 took an approximately constant time.

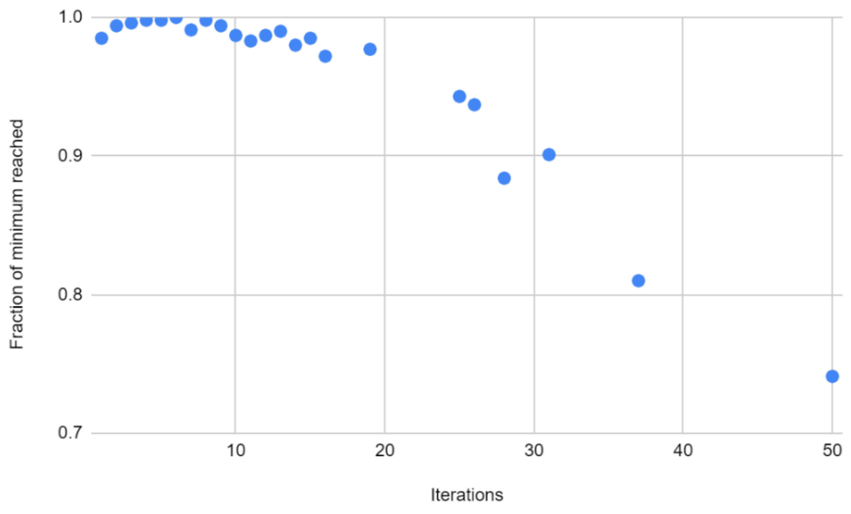

Figure 3: Lenet fraction of minimum reached vs. iterations

In Figure 3 for such a large number of transforms, there is a lack of separation on the lower end of the spectrum. One set of runs with iterations set to 6 even achieved a perfect score. For a more definite localization of the optimal iteration number, the number of allowed transforms would have to be reduced and the test re-run. Regardless, it appears that the optimum number of iterations is in the range of 3-6.

When the problem was reversed, it only took 2% of the computations with the iteration parameter set to 3 to reach the average score of runs with iteration parameter set to default value of 50.

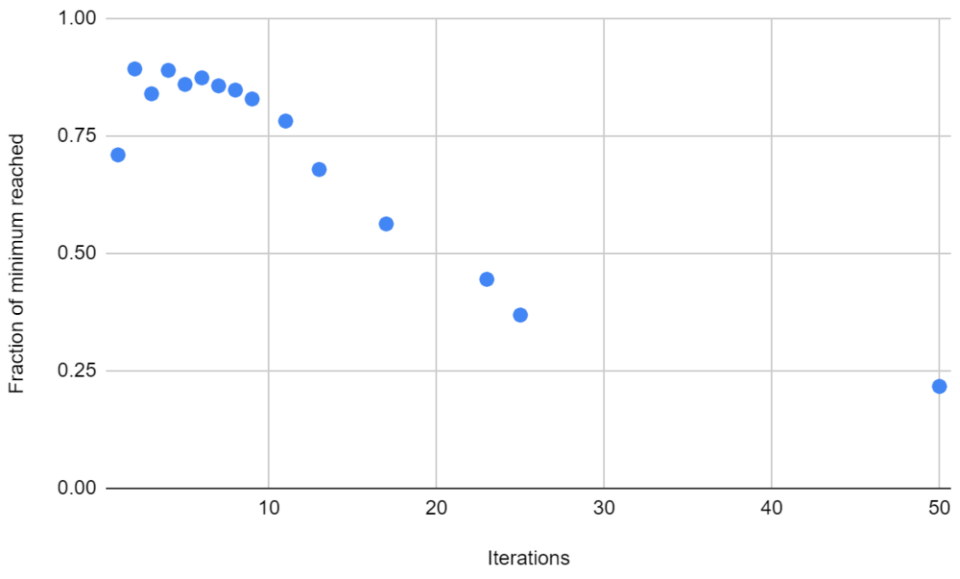

Figure 4: AlexNet Fraction of minimum reached vs Iterations

Figure 4 shows the same methodology applied to AlexNet. Being a larger network and more complex optimization problem, the overall performance values are lower. Due to this complexity, it was not feasible to run ensembles as large as before and there is additional random noise to account for when interpreting Figure 4. There again appears to exist, an optimum range of iterations. This is read from the graph as 2-4.

Based on these two data points the trend in the optimum number of iterations appears to be downward, yet it must be highlighted that LeNet might not be representative due to only residing in one partition. In the absence of more accurate data or methods to estimate this parameter it is thus recommended to set the number of iterations to 3 or 4. It is expected that based on the relatively wide optima in the two studied cases, a small deviation from the optimum iteration number will not have a large effect on performance.

Final temperature

No references could be found in literature to any algorithm or applicable method to estimate the temperature at which the algorithm should be stopped. Owing to the physical analogy of simulated annealing, a “change of state” should be complete once there is insufficient energy for any state transition not directly reducing energy to occur. This would suggest that the minimum temperature might be related to the minimum difference in cost between neighbouring states, a value which may be estimated from the same set of samples used to determine the starting temperature.

An initial survey was conducted to confirm the existence of an optimum. For all other parameters fixed, the average of 100 runs of LeNet was taken for one data point. The temperature was varied with one data point per order of magnitude between $10 - 10^{-13}$. Surprisingly, no optimum was found with performance increasing as the minimum temperature decreased. This somewhat follows previous observations that LeNet tended to yield better performance with greedier optimization.

It is suspected that this conclusion is specific to LeNet and that for a more definitive conclusion on the validity of this hypothesis and method, the above test would have to be repeated on with larger more relevant networks.

Conclusion

Looking back now at this point in the project I now see that to achieve any appreciable degree of rigor this entire body of work would have to be re-done and testing generalized. Even against my efforts some of my decisions related to the method were influenced by my biases and wishful thinking.

The biggest limitation of this work is the lack of more generalized testing. This was largely due to the time requirements for running a large sample on the next bigger networks such as AlexNet and vgg16. This could take days and combined with subpar stability of the experimental code caused problems.

Still, I believe that in the absence of other approaches to estimate the annealing parameters this work is valuable. With everything combined it is expected to provide a multiple times speedup for the annealer which is bound to be useful in accurately and quickly establishing the ground truth for comparison with other optimization methods.

References

[1] W. Ben-Ameur, “Computing the initial temperature of simulated annealing,” *Comput. Optim. Appl.*, vol. 29, no. 3, pp. 369–385, 2004.

[2] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, “Optimization by simulated annealing,” *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

UROP student perspective

Exploring and helping further develop the fpgaConvNet simulated annealing optimizer

Michal Palic, Undergraduate, Department of Electronics and Electronic Engineering, EIE Y1 (2020 entry)

My motivation for doing the UROP

Knowing that the summer holidays are going to be the longest for me yet, and my interest in learning new skills, I was highly motivated to join a project in my area of interest. In the first year, the teaching in the EEE department (as I imagine in most other departments) consisted entirely of mandatory modules, which when combined with the considerable study load, meant I did not have as much time to further pursue areas of interest as I would like. With Imperial College being at the forefront of many research areas and my very pleasant interactions with some of the college staff, I decided to look for ways to get involved, which was how I came across the UROP scheme. The work of the Intelligent Digital Systems lab of my department matched my interests very well, which was why I decided to inquire about the possibility of doing an UROP there.

Also being intrigued with the possibility of eventually doing a PhD, I wanted to use this opportunity to try to get a better idea of how this further study looks like and whether it would be suitable for me.

How I initiated the UROP

I was admittedly lucky to have the head of the iDSL, Dr. Bouganis, as my tutor for the first year, which is why I approached him after the spring semester asking about the potential availability of a suitable project in his lab. After some meetings and gentle reminders, I received a list of potential tasks I might be able to assist with. Once I asked about the details, I accepted the task titled “Help with fpgaConvNet (tutorial + testing)”. However, the description of the task evolved over time.

Preparation for the placement

In preparation for the UROP I completed the cs231n course from Stanford to get a better understanding of convolutional neural networks. I further began to do some reading on hardware description languages, namely VHDL (wrong choice, turns out Verilog is more popular at Imperial) and the Xilinx HLS tools. Since the papers published relating to internal function of fpgaConvNet were available, that was also important reading to get quickly oriented.

The Placement

With COVID still raging and the placement being held remotely, there were some issues. Unlike a normal year, I did not get to meet any of the other students and academics working on this or similar problems. All communication took place over Teams or email, which meant that I could not immediately turn to someone when I had a problem. And there were many given my knowledge of the code and the Ubuntu and Centos operating systems. I was mainly supervised by Mr. Alexander Montgomerie, a PhD student heading the development of the program with whom I was in touch often tie teams or email whenever I did not manage to figure something out. Our semi-regular meetings were used to report progress and the issues I have encountered as well as to brainstorm solutions and recommend what should be done next. I thoroughly enjoyed the open nature of the investigation and that I was not held accountable to a rigid project brief, but instead got to pursue my own ideas and hunches.

Outcome

I ended up gaining a lot from this placement. Just the impulse to learn in preparation for the placement gave me a lot of insight into the field, but likely the area I learned most about was software development and common tasks with Linux based operating systems. While this was not completely what I expected, I am very happy about it because it is a pre-requisite for many high-performance computing tasks as it was for this project. I significantly improved my proficiency in Python and gained experience in working with a larger number of contributors on a project using git for version control. I also gained an appreciation for the simulated annealing algorithm, which can be used very generally, as well as its shortcomings. This was thanks to the research in existing literature that I conducted, looking for existing solutions to my problems.