A performance comparison between CPU and GPU in TensorFlow

ERIC LIND
ÄVELIN PANTIGOSO
A performance comparison between CPU and GPU in TensorFlow

ERIC LIND
ÄVELIN PATNIGOSO
Abstract

The fast-growing field of Machine Learning has in the later years become more common, as it has gone from a restricted research area to actually be in general use. Frameworks such as TensorFlow have been developed to scale and analyze artificial neural networks, which are used in one of the areas in Machine Learning called Deep Learning. This paper will study how well the framework TensorFlow performs in regard to time and memory allocation on the processor units CPU and GPU since these are the factors that are often the restraining resources. Three neural networks have been used to measure how TensorFlow allocates the resources and computes operations used to process the neural network during the training phase. By using TensorFlows profiler we could trace how each operation was executed in the CPU and GPU, from the gathered data we could analyse how the operations allocated memory and time. Our results show that the training of a more complex neural network benefits from being executed on the GPU, while a simple neural network has no or an insignificant profit from being executed on the GPU over the CPU. The result also indicates possible findings for further research such as processor utilisation as the gaps in the scheduling has not been studied in this paper.
Sammanfattning

Acknowledgement

Thank you to our supervisor Stefano Markidis, to Steven W. D. Chien for your support with the AlexNet code, and thanks to all others who have helped us with this paper.
# Contents

1 Introduction .................................................. 1
   1.1 Problem Statement ........................................ 2
   1.2 Restrictions .............................................. 2
   1.3 Scope of the study ........................................ 3
   1.4 Outline .................................................. 3

2 Background .................................................. 4
   2.1 Terminology .............................................. 4
   2.2 Artificial neural networks ................................. 5
   2.3 TensorFlow ............................................... 5
       2.3.1 Predecessor to TensorFlow ......................... 5
       2.3.2 Computational graphs ............................... 6
       2.3.3 Tensors ............................................ 7
   2.4 Processor units .......................................... 7
       2.4.1 CPU ............................................... 8
       2.4.2 GPU .............................................. 8
   2.5 Neural networks being compared ......................... 9
       2.5.1 AlexNet ........................................... 9
       2.5.2 The text classification ............................. 9
       2.5.3 Mnist digit classification ......................... 10

3 Method ....................................................... 11
   3.1 Obtaining performance data ............................... 11
   3.2 Verification ............................................. 12
   3.3 Profiling ................................................ 12
   3.4 Parsing and formatting of results ....................... 12

4 Results ...................................................... 14
   4.1 Execution time on CPU and GPU .......................... 14
4.2 Average Memory allocation ........................................ 18
4.3 Selected Tracing ...................................................... 22

5 Discussion ................................................................ 24
  5.1 Results ................................................................. 24
  5.2 Method ................................................................. 26
  5.3 Further research .................................................... 27

6 Conclusions ................................................................ 28

7 Appendix ................................................................ 29
Chapter 1

Introduction

Machine Learning is a growing field within Computer Science that aims to make computers learn from a great amount of data. A more scientific explanation about Machine Learning can be formulated as: “A computer program is said to learn from experience E with respect to some class of task T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.” [1]

In the field of Machine learning, a newer branch has emerged called Deep Learning and has become more common in both research and practical applications. Deep Learning uses an artificial neural network, loosely based on biological neural networks, to learn predictions from sets of data. At the core of the learning process is matrix multiplication, a calculation known to be a heavy task even with the most optimised operation [2]. It requires quite a powerful framework to compute a result from matrices, TensorFlow is one of many frameworks aimed towards providing a heightened abstraction for the complex calculations in the field of Machine Learning.

CPU and GPU are both processing units made for computation and can be used when executing the calculations on neural networks. The difference between the processing units from a computational view is very simple, the GPU, or graphical processing units, is simply specialised for parallel computation while the CPU, central processing unit, is not. TensorFlow supports multiple CPU’s and GPU’s usage[3], resulting in computations being run on multiple threads to reduce the overall computation time. The usage of both time and memory in the computational stage of TensorFlow usage is what this study will focus on.
1.1 Problem Statement

In this study, we will look into the performance of TensorFlow with respect to time and memory allocation.

- Does the performance of TensorFlow generally benefit from using the GPU over the CPU when using TensorFlow, with regard to time efficiency and memory allocation?

This means analysing TensorFlow’s performance under the training stage and the inference stage of neural networks. TensorFlow is very optimised and finding methods to enhance performance could be difficult but this thesis could help for future optimisation.

1.2 Restrictions

The restriction for this study is mostly caused by our resources and prior experience in this field, which limits the amount of research on neural networks that have been made before this research. Most neural networks that will be used with TensorFlow are, because of the knowledge restrictions, well-known models and not of the most complex nature. The final three neural networks were chosen as their structure was well documented, they were easy to set up and their training was configurable with TensorFlow and its profiler. Further, the neural networks are not to be analysed or compared in this study, rather we are focused on analysing the TensorFlow operation performance for each neural network and on each method used. The performance tests of TensorFlow is based on the time and memory needed to train each neural network on the two types of processing units, GPU and CPU.

Furthermore, there are also technical restrictions of the study. All test will be done on a Linux environment, and further, the GPU test will be running on an Anaconda environment inside the Linux environment which uses the CUDA Toolkit 10.0 to perform computation on the GPU. Both processing units are going to be using the latest stable version of TensorFlow respectively, as the GPU uses TensorFlow_gpu 1.13 and the CPU uses TensorFlow 1.13.1. The processing units are more specifically an Intel Core i7-6500U, 2.5GHz CPU and Nvidia GTX 1070 GPU. These are standard processing units that are freely available on the market and, with a Linux environment, they are easy to set up. However, these are also the technical resources that we have available.
1.3 Scope of the study

The focus of this study is to compare how well TensorFlow uses memory and time efficiently to find, therefore we are not looking for how a code can be written to perform with maximum efficiency with respect to those parameters or the processing unit that is run on. The results will be generated from a limited number of neural networks to test TensorFlow's performance, more specifically each training step of the neural network generates data about duration and memory. Further, each training will be analysed and compared between the two different processing units.

1.4 Outline

We first look at the earlier related work in chapter 2, Background, to see the earlier result that has been generated from the research made in the same field. This is followed by chapter 3, Methods, where we describe what we have done to answer our question and how we did it. Following this comes chapter 4, Results, where we present what our research and experiments have generated. Last comes the Discussion, chapter 5, where we discuss the results, the limitations and what kind of further research we recommend.
Chapter 2

Background

2.1 Terminology

The following are short explanations of the terms related to and used in the study. Some are further explained later in this chapter.

**Artificial Neural Network:** Interconnected group of nodes where state of the input is changed to produce an output.

**CPU:** The Central Processing Unit is the primary processor for general computation and consists of a cache, a RAM, a control unit and an arithmetic logic unit.

**GPU:** The Graphical Processing Unit consist of multiple of the same components as the CPU to parallelize computation.

**Profiler:** A tool for analysing and measuring how a program uses resources, such as memory and time.

**CUDA:** The Compute Unified Device Architecture is a parallel computing platform and programming model used to produce custom code for the utilisation GPU.

**Memory Allocation:** The process of reserving parts, or all, of the computer memory for execution of programs and processes.

**Data flow graph:** A data-flow graph is a directed graph in which assignments and references to variables are represented by the nodes, and information flow is represented by the edges.
2.2 Artificial neural networks

Artificial neural networks (ANN) are based on biological neural networks as it mimics the human brains own neural network. ANNs are graphs that consist of two parts, artificial neurons as nodes and weighted edges in between. All artificial neurons have incoming edges that provide data, the neuron itself contains a function that computes the output value, which is often dependent on a threshold, an activation function. A higher weight of the input implies a higher probability that the function will output a positive value. Adjusting the weight of edges gets the desired output of a network, which can be achieved with training or learning algorithms. Generally, the algorithm will feed training data into the network and give an output, depending on the accuracy of the output the weight will be increased or decreased[4][5].

A simple neural network can consist of one layer of nodes which has one type of function for the network. The layer itself has a function to modify the input data for its dedicated purpose, one example is comparing an input integer to the node value for classifying the input[6]. There are multiple kinds of layers used in the Deep Learning field, three of those are encountered in this study. A fully connected layer has nodes where for each node, there is an edge to each other node in the next layer. These layers are used to compare the input data to the current state of the node, and modifying the state accordingly to a given function. A convolutional layer uses mathematics to find patterns in a matrix that is a representation of the neural network. The layer uses a filter on a node and its neighbouring nodes and computes a dot product between them to calculate the value in the node in the next layer. The last layer is the max-pooling layer which is used to decrease the amounts of nodes in the next layer, decreasing the number of nodes decreases the computational burden of the overall computation[6].

2.3 TensorFlow

2.3.1 Predecessor to TensorFlow

In 2011, Google started The Google Brain project aimed to "explore the use of very-large-scale deep neural networks" [7][8]. The result of their work can be seen today in the framework TensorFlow, an API widely used for ease of use.
However, the team first published the predecessor called DistBelife, a first-generation scalable and distributed training API for Machine Learning. Scaling was a difficult task prior to the development of the tool since other tools did not support the distribution of the computing processes over multiple processing units. In late 2015 The Google Brain team released a second generation training and deployment API, TensorFlow[9], as a distributed system for training neural networks. TensorFlow is one of the more successful and fast-growing open source projects for neural networks and has over 1800 contributors on Github at the beginning of 2019.

### 2.3.2 Computational graphs

While DistBelife used more traditional algorithm model, TensorFlow uses data flow graph as a programming model for all operations, making it more efficient when parallelizing computation. Data flow graphs, also called computational graphs, is a form of a directed graph where nodes represent operations and edges represents the flow of data between operations[8]. For example, if an operation $F$ gives the output $Z$ by applying the input $X$ and $Y$, $F$ has then been represented as a node, $Z$ is the output of the computation of $F$[8].

![Data flow graph](image)

**Figure 2.1**: Show a Data flow graph with function $F$, input $x,y$ and output $z$

TensorFlow has gained some benefits from using computational graphs as a programming model. One being the gained overview of the programming model that improves the general clearness of how values and computation are
related, and another being the more efficient distribution of operations in parallel computing. As all operations will take 0 or more inputs, and give 0 or more outputs, the graph represents computational dependencies\[7\] that can be used later in the process. These computational dependencies can, in TensorFlow, easily be recognised and used when distributing operations for parallel computation. TensorFlow can also partition the operations to execute on different computational units, and even on different machines, provided that the communications are handled.

2.3.3 Tensors

Tensors are an important part of the data flow graph model that TensorFlow uses, as all edges are connected with one tensor each. A tensor is a multi-dimensional collection of homogeneous values with a fixed, static type. The number of dimensions of the tensor determines its rank, and the number of elements in each dimension determines its shape\[8\]. Input and output to all nodes in the data flow graph of a TensorFlow program take and gives tensors of different ranks and sizes, for example, a multiplication takes two 2D tensors and produces one 2D tensor\[7\].

2.4 Processor units

Machine learning, and neural networks in particular, often need to process a large amount of training data to make precise calculations. Particularly the calculations require a large amount of processing power. With today’s processing unit architectures, we have the capacity to train even larger models\[10\] then before. Processing units consists of three major components: memory, arithmetic logic unit and control unit. The memory, or the registers, saves data that can be retrieved later, while the arithmetic logic unit, or ALU, processes arithmetical and logical operations. Last, the control unit the flow of instruction between the ALU, the main memory and I/O devices\[11\]. The combination of these components determines the processing units characteristics. The two main processing units of modern computers are the CPU and the GPU.


2.4.1 CPU

Every computer has one primary component for performing arithmetic logic and control, the central processing unit (CPU). The primary function of the CPU is to sequentially execute instructions that are kept in the computer memory [11]. The CPU has an essential role in the calculation of neural networks since it processes the general arithmetic calculations during the training phase. CPUs are commonly built with a few strong processing cores clocked at 2 to 3 GHz, making the CPU ideal for performing sequential tasks [12]. Further, all I/O in the training phase, such as loading training data, is handled through the CPU regardless of the usage of the GPU for computation [13].

2.4.2 GPU

The graphical processing unit (GPU), much like the CPU, is the main component in a computer used to process instructions, with the distinction that the GPU can operate multiple instructions at the same time using parallelization. One GPU is often built of several weak processing cores with a clock speed much lower than that of the CPU. The multiple processing core system has one major purpose and one reason for its development, to parallelize calculations by using threads to process instructions and therefore speed up calculations generally made for a longer period of time in the CPU [12].

Since the GPU has the ability to run numerous processes at the same time, it has been useful for training the neural networks that entail computationally intensive matrix multiplications. The training of a neural network requires a large number of computations and the GPU is used to optimise calculations by using multiple memory channels and streaming processors [14][12].

GPUs were originally built for rendering graphics, therefore running custom code with ease on the GPU requires APIs to achieve a higher abstraction level, from low-level to high-level programming languages. Since the GPUs architecture enabled every type of parallelisms such as multithreading, MIMD, SIMD, and instruction-level, CUDA was developed to use low-level instructions for utilising these types of parallelism [10]. TensorFlow can, together with CUDA, make use of the whole GPU architecture to further optimise computation time.
2.5 Neural networks being compared

Some machine learning problems can be solved by using neural networks, however, this requires that their application and implementations are flexible, easy to use and has fast computation. Varying conditions are often generated from the different models and sets of data which interferes with the general use of neural networks. Therefore, the neural networks that will be compared and tested in this study are selected from three criteria: structural complexity, their already existing documentation and accessibility. This study will analyse how TensorFlow performs on the general neural network and therefore we have chosen three of different structural complexity to cover a wide spectrum. While some have different amounts of layers, others have different types of layers and some have supporting functions. All there are explained in more detailed below.

2.5.1 AlexNet

AlexNet is deep convolutional neural network consisting of five convolution layers, three max-pooling layers and three fully connected layers. The design is complex as it is made for higher resolution image classification, which requires a lot of learning. The set of data used for training and testing the performance of TensorFlow is the Caltech 101 dataset [15], which consists of coloured images with the rough size of 300x200 pixels and are categorised into 101 categories. AlexNet is well studied and well documented, both in research and in general use.

2.5.2 The text classification

The selected text classifier is a convolutional neural network and a commonly used sentence classifier[16]. The neural network consists of four convolution layers, one max pooling layer and one fully connected layer. The design is fairly advanced and well documented as it is was developed at the beginning of the 21st century. Training the neural network is done with the movie review dataset [17] which consists of both positive and negative reviews an are labelled with respect to sentiment polarity, subjective rating and subjectivity status. Each review is one sentence long and roughly consist of around 75 characters.
2.5.3 Mnist digit classification

The digit classification is five extensions of a neural network, where every version built on top of the previous version with gradually increasing complexity. The first version was built with one fully connected layer, the three later versions use five fully connected layers and different activation functions, optimisers and a dropout functions. The fifth version is a convolutional neural network using three convolution layers, one fully connected layer and one output layer. These neural networks all use the mnist dataset for training, which consisting of black and white images of handwritten digits, with a size of 28x28 pixels [18].
Chapter 3

Method

In this chapter, we define the method and motivate the tools used to test the performances of the processing units. We explain how the results are verified using multiple neural networks and how we extracted the results from the gathered data.

3.1 Obtaining performance data

The performances of TensorFlow, on both CPU and GPU, was tested by measuring time and memory allocation under the training phase of the neural networks as TensorFlows data flow graphs are utilised primarily to optimise training time. Training is an iterative process where the model is tested and then changed so that it can be tested again. Executions of the training were specified to be done on the two different processor units by using CUDA and specifying whether or not the GPU was to be used. To know if the performances were to be changed during the iterations, measurements were obtained during multiple iterations and between 5000 and 10000 training iterations were executed on the neural networks. 1% of training iterations was extracted in even intervals and later analysed, to get a good overview of the possible changes in the performance. This process was implemented on both CPU and GPU.
3.2 Verification

Verifying the results is essential to the work since systematic errors easily can occur. It also ensured that the result is reliable and holds true for general processes. The performance tests were therefore performed on three different neural networks, which are described in section 2.5 "Neural networks being compared". The neural networks vary in size, complexity and training data. Comparing the neural networks and their results does not only ensure general reliability in the results, but also gives insight into how design affects performance.

3.3 Profiling

The performances measurements of all tests were done by gathering data on the execution time and memory allocation. TensorFlow’s own profiler, tf.profiler [19], was used for the measurings because of its ease of use and precision. The training iterations were not to be executed in python, which meant that the conventional profilers would not be able to measure important values and give insight into what occurred during the training iteration. The profiler also gave us the ability to save timelines for all test runs, which later gave tracings of the operations executed on the different threads of the processing units. It also gathered data on how much memory was allocated at any time of the test and which operation that allocated the memory. Data of the tracings were also saved in JSON-files for every test run for later inspection. When analysing the results, the tracings were inspected with chrome’s tracing tool to get further insight into what happened under the training. The Tensorborad tool was also considered for this task, as it visualises the information similarly to chrome’s tool, but since chrome’s tool gave the information needed it was the one to be used. Chrome’s tracing tool were used to visualise that data measured by the profiler, visualised tracings can be found in the Appendix under “Tracings”.

3.4 Parsing and formatting of results

Profiling data gathered from the profiling-tool was formatted to reveal how performances differentiated between the two processing units and during all training executions. First, the profiling data in the JSON-files was parsed to
get the specific values of interest, more specifically time and average memory. The duration of the training phase was determined by calculating the absolute difference of the timestamps of the first and last executed operation. The average memory allocation was calculated by adding all memory allocations under the test and multiply by the time it was allocated, to then divided it by execution time. Average memory allocation was, therefore, given as bytes per microsecond as the profiler measured the change in memory allocation each microsecond. The parsed data was then be formatted into diagrams that gave an overview of performance.
Chapter 4

Results

In this chapter, all results from every neural network will be presented. Results are divided into the duration of test runs, average memory allocation and timeline tracing. The results in the two first sections are represented by plots for every neural network showing some of all test runs. The plots show results of both CPU and GPU respectively. The test results are displayed in sequential order in the plots from the first to last iteration. However, only the first training iteration seems to have a systematic difference in execution, which means changes along the x-axis is of minor concern. In this chapter, we also explain timeline tracings which give a detailed description of TensorFlows processing during test runs. Selected timelines will also be available in the Appendix.

4.1 Execution time on CPU and GPU

This section consist of plots of the execution time for each neural networks training phase. The execution time for the CPU and GPU are represented by the blue and orange dots respectively. Time is measured in microseconds in all plots. There is also a liner trendline to help visualise overall difference of CPU and GPU results.
Figure 4.1: Execution time for AlexNet using the Caltech 101 dataset during the training on both CPU and GPU.

Figure 4.2: Execution time for the text classifier using the movie review dataset during the training on both CPU and GPU.
Figure 4.3: Execution time for mnist version 1, using 1 fully connected layer during the training on both CPU and GPU.

Figure 4.4: Execution time for mnist version 2, using 5 fully connected layers during the training on both CPU and GPU.
Figure 4.5: Execution time for mnist version 3, using 5 fully connected layers with the RELU as activation function and the Adam optimizer during the training on both CPU and GPU.

Figure 4.6: Execution time for mnist version 4, using 5 fully connected layers with the RELU as activation function, the Adam optimizer and Dropout[20] during the training on both CPU and GPU.
18  CHAPTER 4. RESULTS

Figure 4.7: Execution time for mnist version 5, using five layers with three convolutional layers during the training on both CPU and GPU.

4.2 Average Memory allocation

Like the last section, this section also consists of plots for every neural network. The plots give an overview of the average memory allocation of the CPU and GPU, both are represented by blue and orange dots respectively. The average memory allocation is presented in kBytes and measured as the average memory allocated under each test for every microsecond. There is also a liner trendline to help visualise overall difference of CPU and GPU results.
Figure 4.8: Average memory allocation for AlexNet using the Caltech 101 dataset during the training on both CPU and GPU.

Figure 4.9: Average memory allocation for the text classifier using the movie review dataset during the training on both CPU and GPU.
Figure 4.10: Average memory allocation for mnist version 1, using 1 fully connected layer during the training on both CPU and GPU.

Figure 4.11: Average memory allocation for mnist version 2, using 5 fully connected layers during the training on both CPU and GPU.
Figure 4.12: Average memory allocation for mnist version 3, using 5 fully connected layers with the RELU as activation function and the Adam optimizer during the training on both CPU and GPU.

Figure 4.13: Average memory allocation for mnist version 4, using 5 fully connected layers with the RELU as activation function, the Adam optimizer and Dropout[20] during the training on both CPU and GPU.
4.3 Selected Tracing

Tracing results give a detailed look into specific test runs, through a timeline. Tracings include information of all used processing threads, such as duration of computation and what method was operating at an exact time. It also includes a section for memory allocation under the test run. Images of selected tracings can be found in the appendix, the tracings are selected to complement the discussion in section 5.1 *Discussion of results*. Each tracing has multiple computing sections and an allocators section. The computing sections visualises what operations run on which processing unit and also when each operation starts and finishes. It will also show which thread the operation is executed in. The allocators’ section in the tracing gives a representation of the memory allocation for all processing units used during the test run. Results of GPU test runs also show a cuda_host_bfs allocator, however, this is of minor importance as it demonstrates how much of the CPU memory the Cuda toolkit is consuming. The tracings of the GPU includes some computing sections for Cuda streams denoted by section names ending with GPU:n/stream:m with the exception of GPU:n/memcpy that is a specific stream for loading data from
memory. Cuda streams are made for concurrent execution of asynchronous methods, every stream has a sequence of operations that will be executed in order, concurrently from other streams. The tracings also allow us to review information for every operation and memory allocation at a given time of the tracing. This feature is important, but unfortunately, it can only be utilised in Tensorboard or chromes tracing tool.
Chapter 5

Discussion

In this chapter, we discuss results, method and further research of this study. Results show that more complex neural networks benefit more from GPU performance than the CPU, while training more simple neural network yields an insignificant performance difference between CPU and GPU. We also discuss more general findings for further research on this subject, and how our method and restrictions affected the study.

5.1 Results

The results do not indicate any constant performance bottleneck, for either time or memory. Performance between the CPU and GPU had more or less some difference for all neural networks on both processing units and bottlenecks can be found by analysing the profiling data in the tracing results. However, these bottlenecks are particular to the neural networks themselves and not caused by the procession units. Performance differences in the processing units will be discussed further later in this chapter.

Timeline tracing can still show some common operations that all neural networks use. These operations are important, yet simple. An example is MatMul that is used multiple times under AlexNets 81:s test run shown in figure 7.3. MatMul is a standard function for matrix multiplication that is used to change the weight of the neural network under the training step. Common operations are important for performance since they are widely used in neural networks. However, the results do not give an insight into how well these operations per-
form. Tracing results also show that there are often gaps in threads where no operation are registered, as seen in figure 7.5. Further, the gaps indicate that CPU and GPU are not fully utilised during the training execution. We can not determine why these gaps appear and how to minimize these gaps from the data we gatherd in the tests.

Comparing our results from the CPU and GPU are sufficient to show that performance only benefits from GPU’s processing power when a complex neural network is being trained. Training duration results clearly show this relationship between the CPU and GPU usage, while memory allocation seems to mostly depend on the training data size. In figure 4.2 we can observe that the duration time on the GPU is lower compared to the same test on the CPU. However, figure 4.4 show that tests on both CPU and GPU have equivalent duration. This is also true for most of the other neural networks using the mnist dataset. It is only the most complex of the mnist neural networks that utilise the GPU and has performance profits, as shown in figure 4.7. The main difference in the mnist neural networks is whether or not they use convolution layers. The results show that neural network with a higher complexity benefits from the GPUs processing power.

This is not the only interesting result found when comparing the CPU and GPU test. In most of the first tests, the execution time is considerably slower than later test on the GPU. When comparing figure 7.1 and 7.2 it is clear that the operation Conv2D takes a substantial amount of time, on the first run. The Conv2D operation is computed in the convolution layers of the neural network. When TensorFlow is used with a GPU, the CUDA toolkit will analyse how to optimise succeeding operations for the convolution layer, which seems to impact memory allocation when comparing the neural networks using mnist data. If figure 4.13 is compared to the results of the next iteration of the neural network in figure 4.14 the average memory allocation is doubled in the later iteration of the neural network, on both processing units. The differences in the next iteration of the mnist neural network are the use of three convolutional layers instead of the fully connected used in all iterations before. The difference in average memory allocation between CPU and GPU is only clear in the results of AlexNet in figure 4.8 and the Text classifier in figure 4.9. This could be caused by the use of other data sets than mnist. AlexNet uses the Cal-tech 101 dataset, which is a set of fully coloured images with a rough size of 300x200 pixels while the mnist dataset contains black and white 28x28 pixel images and the text classifier uses the movie review dataset that consists of sentences around 75 characters each.
In figure 4.8 it is clear that the GPU uses more memory than CPU for AlexNet steps, and figure 4.9 show that the average memory allocation of the CPU is higher than the GPU. The datasets format could indicate why the CPU or GPU has a higher average memory allocation for the test runs of the two neural networks. Although, with the results of this study we can not conclude that the differences in average memory allocation are due to the differences in data used by the neural networks.

5.2 Method

Each neural network used during the test were chosen from specific criteria where previous documentation had much importance, and the neural network itself was not built at our hand. Since our knowledge of neural networks was restricted, and we had not yet obtained the knowledge to build a neural network, we could custom make tests by modifying the neural network. Using an own built neural network could have made it easier to modify and run tests for common functions and their impact on the time and memory, but also the edge cases of how programming can affect these factors. The three neural networks used in this study are quite different, one major factor being the datasets used for training. Our comparison of the neural networks are independent of each other, therefore eliminating the impact of the different datasets between them. However, by using the same neural network for all test with a database where the data had different sizes would have included further results on how the input size is handled in the CPU and GPU.

TensorFlows profiling capabilities were important for getting an insight into how TensorFlows works under different conditions, although it did not give a general view of TensorFlows performance. Profiling gives information on the operations that are active during execution, if well studied the results can often give an idea of where bottlenecks may have appeared. By profiling multiple neural networks test runs we hopped we could evaluate the general performances of TensorFlow. However, with the amount of data the profiling gave the results were overwhelming and made it difficult to assess. Profiling data should have been formatted to form general results for both the neural networks and TensorFlow.
5.3 Further research

The research conducted from this paper can be extended into different scopes. A few suggestions for further research that we have thought about are described below.

Using one set of data for training, with changeable input sizes, could indicate how the input size affects TensorFlow’s performance and identify possible breaking point for when the input size has a considerable effect. While we have been restricted to completed neural networks, building a neural network would allow modifications of the input since the neural network we have used has been specified to their own set of data.

The results show significant gaps in computations, our research has not had this into consideration therefor we have no indications about why they occur. We believe the gaps could stem from either the programming itself, the hardware limitations or TensorFlow performing poorly.

Using TensorFlow’s profiler can also record the data flow graph of training. The graph can then be visualised in TensorBoard. We did not take advantage of this feature, however, it could be used in further research. Analysing the data flow graph could make it easier to determine how computation is related. By using the TensorBoard to visualise data flow graph, a study could determine relations in TensorFlow computation and process.
Chapter 6

Conclusions

The differences in the performance of TensorFlow depends significantly on the processing unit and the more complex neural networks benefit from the GPUs parallelizing capabilities, which makes using GPU with TensorFlow well worth it in most cases. However, the benefits of the GPU becomes insignificant when a simplistic neural network is trained with small instances of training data. It’s hard to draw any conclusion on the memory management of the GPU and CPU as the results indicate that the average memory allocation was affected mostly by the training data.

Using a profiler to measure test runs was easy, however, it was time-consuming. Profiling test runs gave an insight into operations run by TensorFlow and a lot of data about each one. However, it did not give any results of TensorFlows general performance on either processing units. Although no specified conclusion could be drawn from the results, further research into the subject could generate guidelines for the general optimisation of Training neural networks in TensorFlow.
Chapter 7

Appendix

Tracings
Figure 7.1: Tracing of test 0 on GPU, training the Text classifier with the movie review dataset.
Figure 7.2: Tracing of test 10 on GPU, training the Text classifier with the movie review dataset.
Figure 7.3: Tracing of test 81 on CPU, training AlexNet with the Caltech 101 dataset.
Figure 7.4: Tracing of test 81 on GPU, training AlexNet with the Caltech 101 dataset.
Figure 7.5: Tracing of test 12 on GPU, training mnist version 4 using 5 fully connected layers with the RELU as activation function, the Adam optimizer and Dropout.
Figure 7.6: Tracing of test 13 on GPU, training mnist version 4 using 5 fully connected layers with the RELU as activation function, the Adam optimizer and Dropout.
Bibliography


