# Project Report

on

Benchmarking densely connected neural networks on MNIST dataset

for the course

High Performance Architectures and Their Compilers

CS 243

Authors: Akshay Sudhir Bapat, Ananthakrishnan Pushpendran, Kanika Baijal

Guided by: Alexandru Nicolau, Neftali Watkinson

**Table of contents:**

|  |  |  |
| --- | --- | --- |
| Chapter No. | Title | Page No. |
| 1 | Introduction | 2 |
| 2 | Related work | 3 |
| 3 | Proposed work | 3 |
| 4 | Analysis | 4 |
| 5 | Observations | 5 |
| 6 | Results | 10 |
| 7 | Conclusions | 11 |
| 8 | References | 11 |

1. Introduction

Neural Network:

A neural network is an interconnected group of natural or artificial neurons that uses a computational model for information processing based on a connectionist approach to computation. In most cases an ANN is an adaptive system that changes its structure based on external or internal information that flows through the network. In more practical terms neural networks are non-linear statistical data modeling or decision-making tools. They can be used to model complex relationships between inputs and outputs or to find patterns in data.

Advantages of neural networks:

* They can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques.
* Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.
* Real Time Operation: ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.

Parameters of neural networks

* Learning rate
* Loss function
* Batch size
* Number of training iterations

Performance of neural networks

* Training and testing accuracy
* Time required for training
* Memory required for training

1. Related work

<https://papers.nips.cc/paper/548-benchmarking-feed-forward-neural-networks-models-and-measures.pdf>

<https://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/viewFile/9993/9347>

<https://arimo.com/data-science/2016/bayesian-optimization-hyperparameter-tuning/>

1. Proposed work

The objective of this project is to benchmark a densely-connected neural network using stochastic gradient descent on MNIST dataset on TensorFlow. In this regard, we have decided to focus on certain performance measures – namely, computation time, memory required, testing accuracy. We will determine the optimal values of batch size and number of iterations (epochs) so that we can get the best possible performance. Furthermore, we will evaluate the performance on different tensor types – half-precision (fp16), single precision (fp32) and double precision (fp64). Also, we will compare performances on different devices, including both CPUs and GPUs.

Devices:

* Nvidia GTX 1050 Ti
* Nvidia GT 940M
* CPU: Intel i5 7287U – 3.1 GHz

Performance measures:

1. Accuracy vs batch size
2. Accuracy vs epochs
3. Computation time vs batch size
4. Computation time vs epochs
5. Memory required vs batch size
6. Memory required vs epochs
7. Analysis
8. Testing accuracy vs batch size

Stochastic gradient descent methods are algorithms of choice in Machine Learning. These methods involve using small batches wherein a fraction of the training data, usually 32-512 data points, to compute an approximation to the gradient. It has been observed that if a larger batch is used, there is a significant degradation in the performance of the model. Having a batch size which is too large results in poorer generalization while small-batch methods usually generalize well. Inherent noise in the gradient estimation is widely considered to be the reason for this effect.

1. Testing accuracy vs epochs

In a neural network, both the training and testing accuracy usually improves with epochs. However, as you increase the number of epochs, even though you increase the training accuracy, you might end up overfitting the data. In such a case, the testing accuracy will decrease after a point. This usually means that the network has learnt the input data too well and has lost the ability to generalize. Thus, we usually take the number of epochs as the value where the testing accuracy is the maximum, or in other words, testing error is the minimum.

1. Computation time vs batch size

Computation time generally drops with batch size. The larger the batch size, the less number of independent computations are needed during training. Higher batch size implies that the gradient is a better approximation of the actual gradient and thus will converge faster. Thus, after a few epochs, the calculations will be trivial and the code will run faster.

1. Computation time vs epochs

The relation between computation time and number of epochs is usually linear, that is, computation time is approximately directly proportional to the number of epochs.

1. Memory required vs batch size

There is no direct relation between the memory (RAM) required and the batch size. It is expected that memory required will not vary much based on batch size. It can be noted that if we vary the batch size, some batch sizes will have more usage of number of data points that others, so there will be slight variations.

1. Memory required vs epochs

There is no direct relation between the memory (RAM) required and number of epochs. In fact, increasing epochs just improves the accuracy while using the same memory. Thus, it is expected that memory required will be the same regardless of the number of epochs.

1. Memory required vs tensor type

It is expected that fp64 will occupy more memory than fp32 which will exceed memory of fp16.

1. Time required vs tensor type

It is expected that fp64 will be slower than fp32 which will be slower fp16.

1. Accuracy vs tensor type

It is expected that fp64 will be more accurate than fp32 which will be more accurate than fp16.

1. Observations

We are working on comparing tensor types fp32, fp64. We tried fp16 but there is no advantage due to lack of GPU support for consumer cards. In consumer cards such as the ones we had access to, 16-bit operations were performed under a reduced clock speed. Thus, even though we managed to speed up the performance two times faster, it would still be slower.

The resources to support this can be found here -

Link 1

<https://www.reddit.com/r/MachineLearning/comments/6p8hcs/d_floating_point_precision_in_deep_learning/>

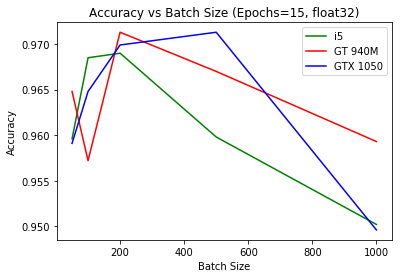
Link 2

<https://devtalk.nvidia.com/default/topic/934562/cuda-programming-and-performance/nvidia-pascal-titan-x-geforce-gtx-1080-gtx-1070-gtx-1060-gtx-1050-amp-gt-1030/5>

Thus, for this benchmarking, we did not perform operations using Half-Precision floating points (fp16).

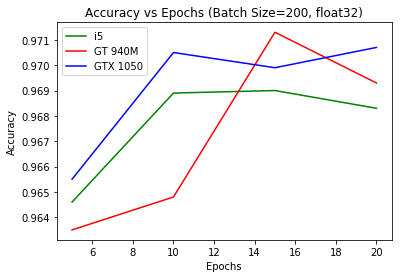
We did, however, perform a variety of benchmarks which can be seen below:

* 1. Accuracy vs batch size



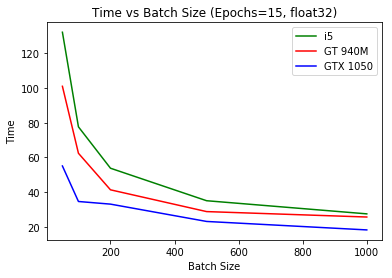
The plot for accuracy vs batch size across different devices has a clear peak in the middle. This shows that we obtain the best accuracy an optimal batch size, 200, which we are going to use for computing the later parts of the graph. The choice of epochs and data type was based on the best accuracy we got across all the values we used.

* 1. Accuracy vs epochs



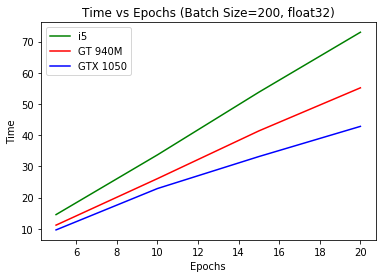
The testing accuracy increases as we increase the number of epochs. This is expected as the network is trained multiple times, increasing the accuracy and updating the weights over time. There is a dip in the accuracy after a certain number of epochs as the network starts to overfit the data and even though the training accuracy will keep on increasing, the testing accuracy is adversely affected and starts to decline.

* 1. Computation time vs batch size



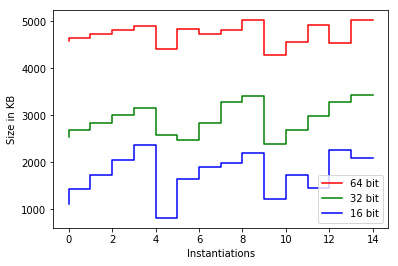
The graph clearly shows the computational prowess of GPUs over a CPU. For very low batch size, the time taken by the i5 CPU is more than double the time taken by the GTX 1050. This is because the forward and backward propagation and the updates to the weights are made much more frequently, thus increasing the computational requirement, which is easily handled by a hardware like GPU that has multiple cores and is optimal for computing matrix but not by a CPU.

* 1. Computation time vs epochs



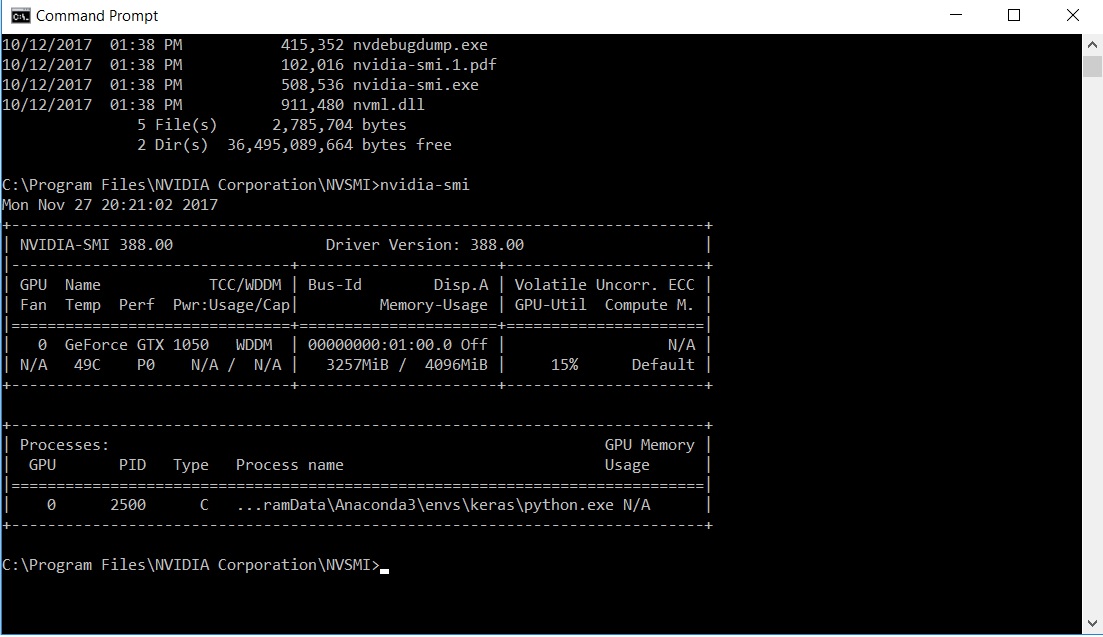
The graph shows the time taken for the hardware to train the neural network. As expected, the time increases with the increase in number of epochs. Again, the GPUs overshadow the CPU by taking much less time to complete the process. The higher end GTX1050 takes less than half the time of the CPU, while GT940M takes slightly more than half the time taken by the CPU for the highest epoch.

* 1. Memory required
* Hard disk space

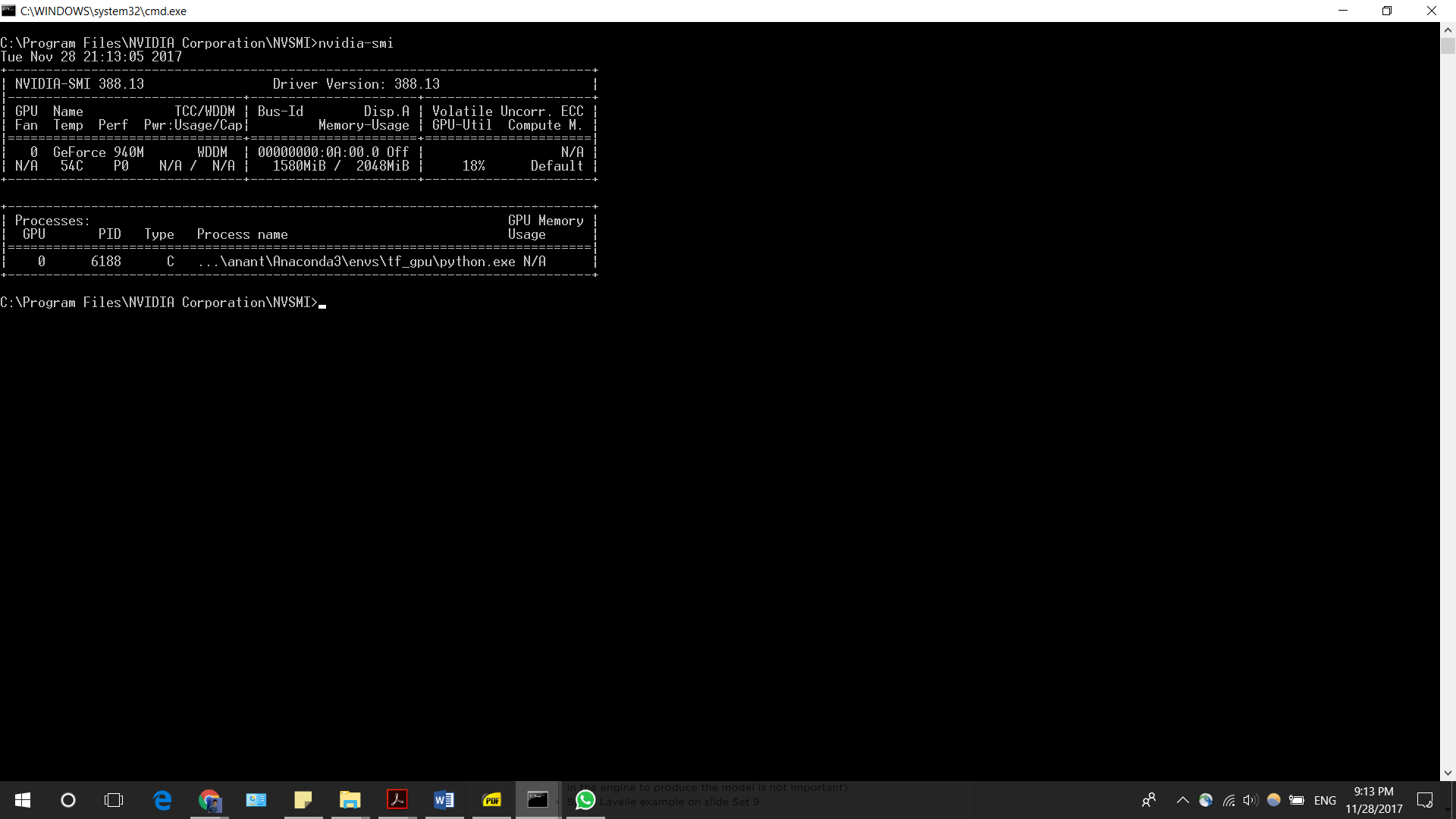


The graph shows that the file size of the` saved neural network is more or less constant for a fixed tensor type. The file size does not change at all with changing epochs as expected, and fluctuates a bit with changing batch size. It was observed that the size does not vary across models (GTX 1050, GT 940 and Intel i5 CPU) and the only significant changes were across tensor types (half precision, single precision, double precision)

* Memory (RAM) required



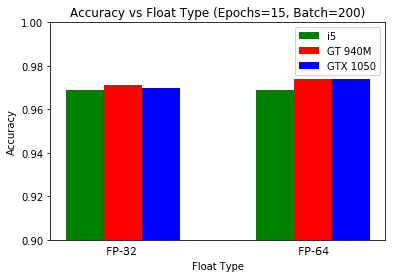
GPU memory required by the code is 3257MB/4096MB which is 79.51% of the total RAM. Usually, memory usage of 80-100% means that the code is consuming the GPU resources well and thus, having maximum speeds.



GPU memory required by the code is 1580MB/2048MB which is 77.14% of the total RAM. Usually, memory usage of 80-100% means that the code is consuming the GPU resources well and thus, having maximum speeds.

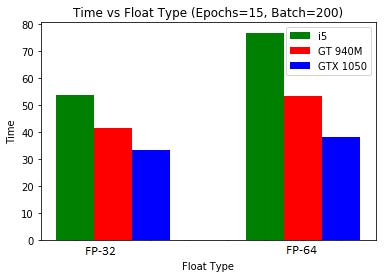
We could not determine the exact RAM usage for CPU because CPU handles many others tasks simultaneously and the readings might not have been accurate.

* 1. Accuracy vs float type



The accuracy of the neural network on the training data for different precisions and devices are shown in the graph. As expected, the accuracy is almost the same across the devices and the data type. For single precision, the GT 940M achieves the best accuracy while for double precision, both the GPUs achieve the same, best accuracy. The reason behind this is how the training batches were provided to the neural network. Even the CPU could have achieved the same accuracy, with the downside of taking a much higher training time. There is also no significant difference between the single and double precision data type. The single precision data type is able to compute the weights of the neural network without any loss of precision. This reaffirms our real-world observation where single precision is the industry standard. Switching to double precision is not worth the extra computational space and time required since there is no significant improvement in the accuracy.

* 1. Time required vs float type



The bar graph shows the time taken to train the network for a given epoch and batch size across all devices and single and double precision floating points. The time taken while using single precision is less than the time taken for double precision for all hardware benchmarked across. The time taken for double precision is almost double for the CPU while the difference is not that high for the GPUs. This, coupled with our observation of accuracy achieved by the neural network from the previous graph shows that using higher precision does not offer any significant advantage. This is especially true when the hardware at disposal is not a GPU which is oriented towards high performance tasks.

1. Results
2. *Accuracy vs. batch size:*

Best accuracy was achieved at a batch size of 200

1. *Accuracy vs. epochs:*

The test accuracy increased as the number of epochs was increased

1. *Computation time vs. batch size:*

Computation time decreased with increasing batch size. The performance test on the GPU gave better results than that on the CPU

1. *Computation time vs. epochs:*

The computation time increases with the increase in number of epochs. Here as well the GPU outperforms the CPU

1. *Memory required vs. batch size:*

The file size varies for different tensors type. The file size is maximum for tensor double precision across all three systems

1. *Accuracy vs. float type:*

The accuracy of double precision is only slightly better than the single precision

1. *Time required vs. float type*

Training a single precision model is faster than a double precision one. The difference of training time for the two-tensor types is significant on a CPU than that on a GPU

1. Conclusions and future scope

Performance analysis was done across three machines with different hardware i.e. CPU’s and GPU’s. The conclusions are as follows:

1. Accuracy achieved by single and double precision is better on a GPU than CPU. The training time on a GPU is almost half that on a CPU
2. The accuracy of double precision is only slightly better than the single precision but switching to double precision is not worth the extra computational space and time require
3. The file size varies across different tensors type. The file size is maximum for tensor double precision across different systems.

Future Scope

Since the current GPU support for half precision is limited, this benchmark can be extended to systems which support fast half precision calculations such as the Nvidia Tesla series.

8. References

* Keskar, Nitish Shirish, et al. "On large-batch training for deep learning: Generalization gap and sharp minima." *arXiv preprint arXiv:1609.04836* (2016).
* <http://www.sciencedirect.com/science/article/pii/0895717794002075>
* <https://papers.nips.cc/paper/548-benchmarking-feed-forward-neural-networks-models-and-measures.pdf>
* <https://arimo.com/data-science/2016/bayesian-optimization-hyperparameter-tuning/>
* https://www.reddit.com/r/MachineLearning/comments/6p8hcs/d\_floating\_point\_precision\_in\_deep\_learning/