MULTIPROCESSOR PROGRAMMING, DV2597/DV2606
—
LAB 3: ACCELERATING DEEP LEARNING FOR HANDWRITTEN DIGITS USING A GPU

Lars Lundberg, Sai Prashanth Josyula, Håkan Grahn
Blekinge Institute of Technology

2021 fall
uppdated: 2025-10-16

The objective of this laboratory assignment is to give some experience of how GPU programming can be used to speed up a computationally intensive real-world application.

The laboratory assignments should be conducted, solved, implemented, and presented in groups of two students! It is ok to do them individually, but groups of more than two students are not allowed.

Home Assignment 1. Prerequisties and preparations

You are expected to have acquired the following knowledge before the lab session:

- You are supposed to have good programming experience and good knowledge of C / C++ programming.
- You are supposed to have knowledge about GPU programming concepts and be familiar with writing GPU programs.

Read the following sections in the course book [3]:

- Chapter 16 (Application case study–machine learning)

End of home assignment 1.

Home Assignment 2. Plagiarism and collaboration

You are encouraged to work in groups of two. Groups larger than two are not accepted.

Discussions between laboratory groups are positive and can be fruitful. It is normally not a problem, but watch out so that you do not cross the border to cheating. For example, you are *not allowed* to share solution approaches, solutions to the different tasks, source code, output data, results, etc.

The submitted solution(s) to the laboratory assignments and tasks should be developed by the group members only. You are not allowed to copy code from somewhere else. The only other source code that you are allowed to use is the one provided with the laboratory assignment.

End of home assignment 2.

1 Introduction

In this laboratory assignment, we will work with graphics processing units (GPUs) and accelerate a deep learning algorithm. The MNIST database[1] contains 70,000 images, each containing 28×28 pixels. Each image represents a handwritten digit (0-9). The database is divided into a training set with 60,000 images and a test set with 10,000 images. Figure 1 shows how the data in the MNIST dataset looks like[2]. Chapter 16 in the course book [3] describes a deep learning system called LeNet-5[3]. This system is capable of *learning* from the training set to predict correct labels for future inputs. However, the training phase is computationally intensive, particularly when run sequentially. In this lab, the goal is to parallelize the computations in the training phase of LeNet-5 using a GPU.

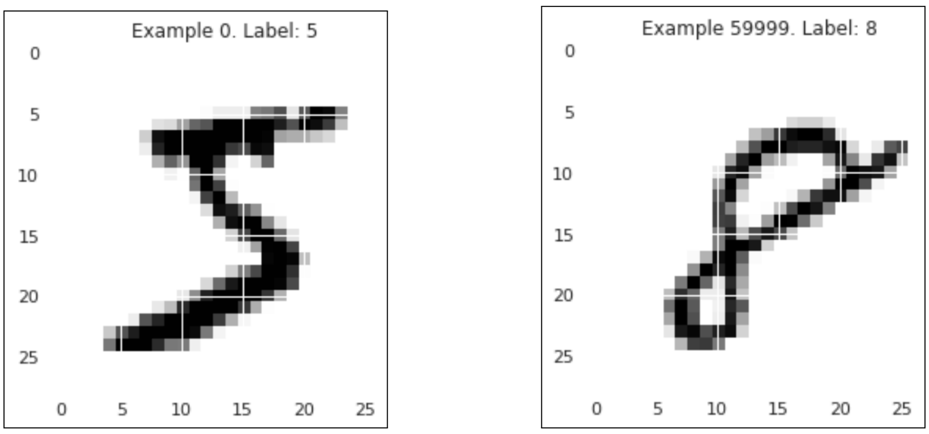


Figure 1: Example of data from the MNIST database

The sequential C program provided to you has been tested on a Windows laptop. However, it should also work on the computers in the lab room as well as your personal computers. The source code files and programs necessary for the laboratory are available on Canvas. You shall compile the C program provided to you using:

```
gcc -O2 -o output_file_name sourcefile1.c ... sourcefileN.c
```

2 Examination and grading

Note: Present and discuss your solution with a teacher / lab assistant when it is done, i.e., before submitting on Canvas.

When you have discussed your solution orally, prepare and submit a `tar`-file or `zip`-file containing:

- **Source code:** The source-code for working solutions to the task in Section 3.2. Specifically, you shall submit your well-commented source code for the task at hand.
- Corresponding `Makefile`(s), or a text-file describing how to compile the respective project / task.
- **Written report:** You should write a short report (approximately 2-3 pages) describing your implementation and your measurements (results), as outlined below. The format of the report must be pdf.
  - **Implementation:** A general description of your parallel implementation, i.e., you should describe how you have partitioned the work between the GPU threads, how the data structures are organized, how the GPU threads are synchronized, etc.
  - **Measurements:** You should provide execution times for two cases: (i) the sequential version of the program given to you, and (ii) the parallel program running on a GPU (using as many threads as you like).
    Further, analyze and discuss your results.

All material (except the code given to you in this assignment) must be produced by the laboratory group alone.

The examiner may contact you within a week, if they need some oral clarifications on your code or report. In this case, all group members must be present at that oral occasion.

[1]For more details, see `http://yann.lecun.com/exdb/mnist/`
[2]To see the first 100 entries in the dataset, click here.
[3]For a demo of Lenet-5, see `http://yann.lecun.com/exdb/lenet/`

3 Task

You are going to parallelize (using CUDA) the training phase of a learning algorithm based on a neural network. Section 3.1 presents a brief introduction to neural networks. Feel free to skip the section if you are already familiar with these concepts. Section 3.2 briefly describes LeNet-5 and presents the task you are supposed to complete and submit for this lab.

3.1 A brief introduction to neural networks

In the context of artificial intelligence, Figure 2 shows the simplest artificial *neuron*, called *threshold logic unit (TLU)* [1]. This neuron's job is to perform a weighted sum of its input values $x_0, x_1 \dots x_n$ and then output a "1" if this sum exceeds a threshold, and a "0" otherwise [1]. An artificial neuron mimics the functionality of biological neurons in a human brain.

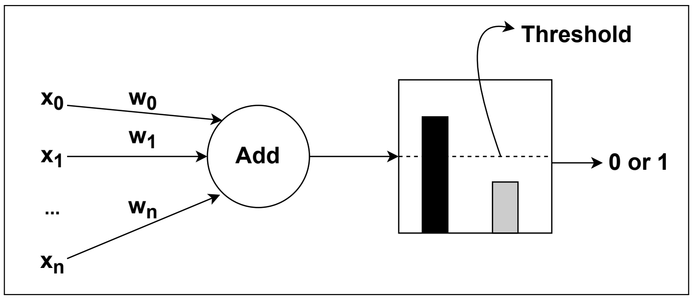
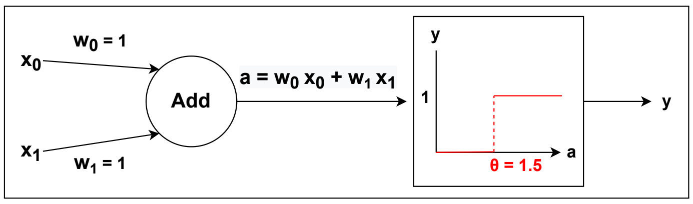
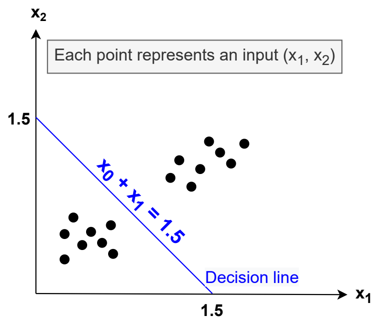


Figure 2: The simplest neuron [1]

Pattern classification: The above introduced neuron may be thought of as classifying its input patterns into two groups: (i) the group with input patterns that give output *1*, and (ii) those that give output *0* [1]. For example, consider a two-input TLU with (w_1, w_2) as weights and θ as the threshold. Whenever the value of $w_1x_1 + w_2x_2 < \theta$, the input pattern is classified into group 0 and whenever $w_1x_1 + w_2x_2 > \theta$, the input pattern is classified into group 1. Geometrically, the equation $w_1x_1 + w_2x_2 = \theta$ represents a straight line and points on either side of the line will belong to the two alternative groups of our example. For our TLU, different values of (w_1, w_2, θ) would result in different *decision lines*. Figure 3 shows a simple example of how a TLU can be used for classification.



(a) A two-input neuron with values of $(w_0, w_1) = (1, 1)$ and $\theta = 1.5$



(b) The decision line for the neuron of Figure 3a

Figure 3: An example of how a neuron can be used to classify inputs into groups

For a TLU to perform a given classification task, it must have the desired decision boundary. Since this boundary is determined by the values of weights and threshold, it is important to adjust these parameters to bring about the functionality we need [1]. Typically, the weights and thresholds are adjusted through an iterative process by repeatedly presenting examples of the required classification task [1]. In each iteration, small changes are made to the weights and threshold to bring them closer to the desired values. This process is known as *training* the network, and the set of examples as the *training set* [1]. The network *learns* or adapts to the training set by changing its parameters at each step. In *supervised training*, we present a set of inputs and for each input, the corresponding desired output that the network is supposed to return.

In the real world, there are many problems where the points that we want to classify cannot be separated by a simple decision line, but need a decision boundary that is non-linear. Figure 4 shows an example where the aim is to classify an input image as a car or not-a-car based on two specific pixel locations given as input. A single neuron cannot handle such classification tasks. To handle such interesting problems, we need a *neural network*, e.g., a collection of *layers* with many neurons in each layer. One such interesting problem is to identify images of handwritten digits by classifying them correctly.

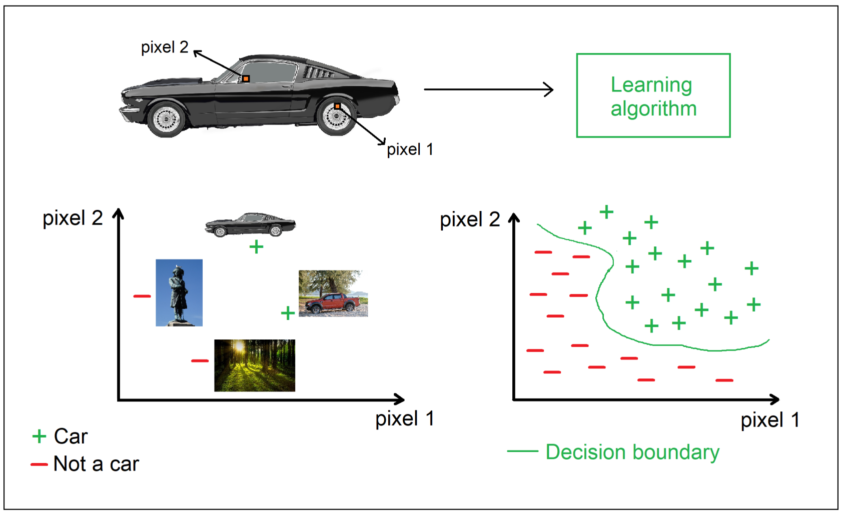


Figure 4: A toy illustration to show how an algorithm classifies images by finding a decision boundary (inspired from [4])

3.2 LeNet-5 neural network

A convolution neural network (CNN) is a specific type of neural network that typically comprises of *convolutional layers*, *subsampling layers*, and *fully connected layers*. For a quick overview of convolutions, subsampling, etc. in this context, see, e.g., [2]. LeNet-5 is a simple CNN architecture that was introduced in the 1990's. Figure 5 gives an overview of LeNet-5 and shows how an input image is processed to produce an output. The aim of this neural network is to classify input grey images of size 32 $\times$ 32 pixels into ten groups, based on the output value.

In Figure 5, the CNN appears to return an output value for an input image. For the CNN to return the correct output value, it has several parameters, which need to be determined appropriately. In the training phase, the known *(input, output)* pairs are fed to the network and the parameters are adjusted accordingly. In LeNet-5, the first four layers, shown in Figure 5, have 156, 12, 1516, and 32 parameters, respectively [5]. In the last layers, there are 58,284 parameters [5]. In total, there are 60,000 parameters that are adjusted in the training phase. The goal is to train the network in such a way that after the training phase, it gives the correct output for a new input.

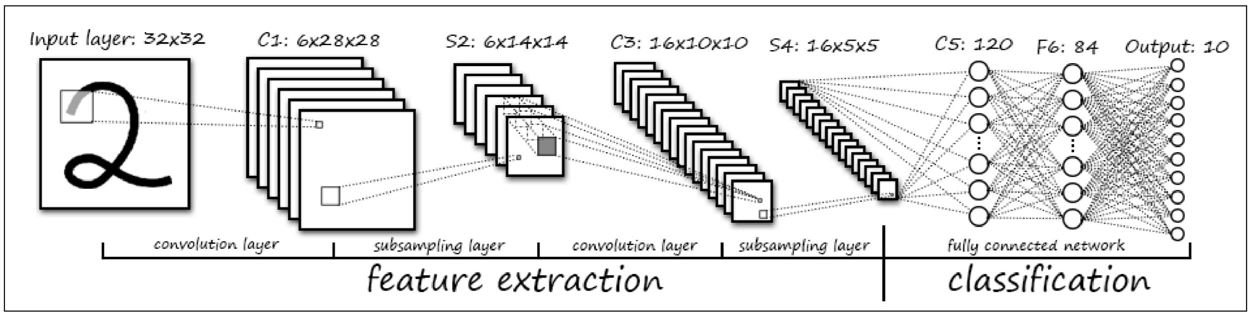


Figure 5: An overview of LeNet-5 [5]

The training phase requires intensive computations that are time consuming. Accelerating the training phase has several advantages, e.g. to train the network quickly and such that it is capable of producing better results, i.e., more correct predictions for its inputs. The following are some steps (in chronological order) involved in training a neural network. Note that these are not an exhaustive list of steps, but a concise subset to give you an idea of the training phase. At the end of the training phase, you will have fixed all the parameters of your neural network.

Let p be the set of parameters in our neural network. A function called the *cost function* $J(p)$ is used to measure how well your parameters are performing on the training set. The goal of the training phase is to minimize the value of $J(p)$. This can be done using

Steps involved in training a neural network:
1. Randomly initialize the parameters p used in the neural network.
2. Implement code to compute *forward propagation* at each layer.
3. Implement code to compute the cost function.
4. Implement code to compute *backpropagation* at each layer.

```
for (each training example $(x_i, y_i)$)
    Perform forward propagation using $(x_i, y_i)$.
    Perform back-propagation using $(x_i, y_i)$.
```

Task 1. Parallel deep learning using LeNet-5

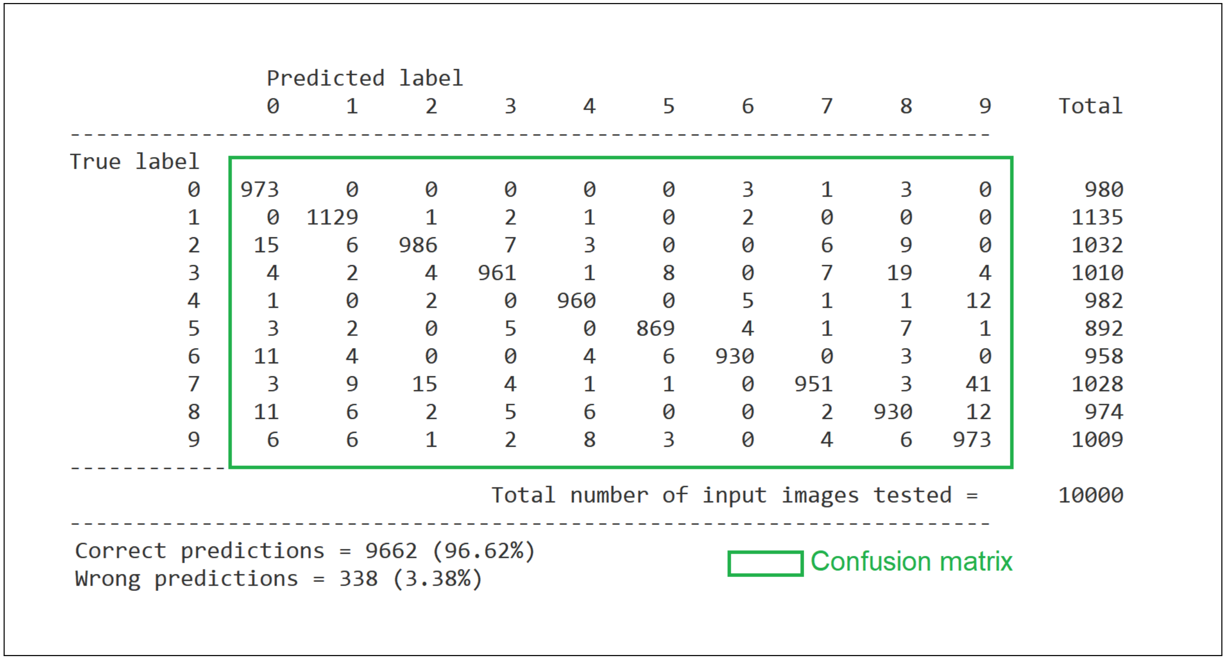
You are supposed to do the following:

- Write a parallel implementation of LeNet-5 using CUDA, based on the code snippets in Chapter 16 of [3].
- Measure the training time using the 60,000 images in the training set of the MNIST database.
- Measure the speedup in training time compared to the sequential code given to you.
- Measure the classification time, i.e., the time it takes to classify the 10,000 images in MNIST's test set.
- Record the confusion matrix for the test set.

The CUDA implementation should demonstrate an overall performance improvement compared to the sequential version, achieving a speedup greater than 1.

End of task 1.

For our problem, the confusion matrix[4] is a 10×10 matrix with the true values (0-9) as rows and the values predicted (0-9) by your implementation of LeNet-5 as columns (see Figure 6). Since there are 10,000 images in our test set, the sum of all values in your 10×10 confusion matrix should be 10,000.



```
                    Predicted label
                    0      1      2      3      4      5      6      7      8      9      Total
--------------------------------------------------------------------------------------
True label
           0      973      0      0      0      0      0      3      1      3      0        980
           1        0   1129      1      2      1      0      2      0      0      0       1135
           2       15      6    986      7      3      0      0      6      9      0       1032
           3        4      2      4    961      1      8      0      7     19      4       1010
           4        1      0      2      0    960      0      5      1      1     12        982
           5        3      2      0      5      0    869      4      1      7      1        892
           6       11      4      0      0      4      6    930      0      3      0        958
           7        3      9     15      4      1      1      0    951      3     41       1028
           8       11      6      2      5      6      0      0      2    930     12        974
           9        6      6      1      2      8      3      0      4      6    973       1009
------------
                                  Total number of input images tested =                   10000
--------------------------------------------------------------------------------------
Correct predictions = 9662 (96.62%)
Wrong predictions = 338 (3.38%)
```

Figure 6: Example of a confusion matrix for a test set containing 10,000 images

[4]For a quick introduction to confusion matrix, see `https://en.wikipedia.org/wiki/Confusion_matrix`.

References

[1] K. Gurney, "An introduction to neural networks, 1st ed.," *UCL Press*, 1997, ISBN: 0-203-45151-1.

[2] A. Amidi and S. Amidi, "Convolutional neural networks cheatsheet," 2018, URL: `https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks`.

[3] David B. Kirk and Wen-mei W. Hwu, "Programming Massively Parallel Processors: A Hands-on Approach, 3rd Edition," 2016, ISBN-9780128119877 (ebook).

[4] A. Ng, "Machine Learning MOOC lecture slides, Coursera," 2021, URL: `https://www.coursera.org/learn/machine-learning`.

[5] B. Póczos, "Introduction to Machine learning lecture slides, Carnegie Mellon," 2017, URL: `http://www.cs.cmu.edu/~10701/slides/10_Deep_Learning.pdf`.

A Program listings

A simple sequential implementation of LeNet-5, that is written in C language is available online[5] with an open-source license (the *MIT license*). The code does not rely on any third-party libraries and you can compile and execute it as it is. We modified that open-source implementation to print out the confusion matrix and also added some additional comments. The updated code is available to you on Canvas. The three code files provided to you are `lenet.h`, `lenet.c`, and `main.c`. You may use that sequential code: (i) as a starting point for your GPU implementation, (ii) to compare the training speed of your parallel program.

B Note

You can reuse the provided open-source sequential code, e.g., (i) by writing your own GPU kernels for convolution, sub-sampling, and dot product operations that happen in the forward propagation and back-propagation and (ii) calling your GPU kernels in the `forward()` and `backward()` functions of the sequential code (instead of the sequential version of those operations).

You may instead also choose to parallelize only the forward propagation on the GPU (and perform the backpropagation on the CPU). However, note that if you choose to parallelize only the forward propagation, you are expected to (i) not only write your own GPU kernels for convolution, sub-sampling, and dot product operations in the forward propagation, but also (ii) use the GPU to perform the forward propagation on each training image (of the mini batch) in parallel. Note that to perform forward propagation on each training image (of the mini batch) in parallel, you will need to modify the `TrainBatch()` algorithm accordingly, to suit your needs.

Note that the LeNet-5 architecture has a layer with 84 neurons as its sixth layer. However, the sequential implementation provided to you does not implement this layer. For the task in this lab, you are not expected to implement that layer in your GPU parallel program and can simply omit it.

[5]For the original code, see `https://github.com/fan-wenjie/LeNet-5`.