Parallel Computing Architecture 048874 (aka Manycores for Machine Learning)

November 2017

Homework 1: MNIST inference with 2-layer 1000 hidden units NN as a sequential C program

1. Create and compute MNIST inference on any personal computer (e.g., your laptop), as described here. We prefer Windows and Intel architecture because our parallel programming tool (to be used in follow-on homework assignments) is designed for that environment, and the code created in this homework should eventually be ported to there.
2. See definitions and explanations in <http://eecourses.technion.ac.il/048874/2016-11-05-RomanKaplan-ShortNNintro.pdf>
3. Focus on the fully connected net with 784 inputs with one byte per input pixel, 1000 neurons in the single hidden layer with ReLU activation function, and 10 neurons in the output layer.
4. The output should be softmax (a vector of ten values in [0,1] whose sum is 1.0).
5. Create a sequential program for inference. Best is to use C – it will be simplest to port to the parallel programming tool. Alternatively you may also use C++ or Matlab, and convert to C in the future.
6. We provide one parameter file, [1000HU\_BinaryFloatWeights.txt](http://eecourses.technion.ac.il/048874/HW1-fall-2016/1000HU_BinaryFloatWeights.txt), binary float weights file for a 1000 hidden units ReLU net. Weights are provided for both the hidden layer and the output layer. These parameters were computed by Roman Kaplan using back propagation training, as described in the above linked document (we will also perform training in a future HW).
7. You may wish to use the code segment in [Read\_Weights.c](http://eecourses.technion.ac.il/048874/HW1-fall-2016/Read_Weights.c) to allocate the data structures and to read the weights from the parameter file.
8. The code segment in [mnist\_image\_parser.c](http://eecourses.technion.ac.il/048874/HW1-fall-2016/mnist_image_parser.c) could be useful in obtaining the input data (MNIST images). The file includes instructions how to download, ungzip and read the data.
9. A second file, [1000HU\_BinaryFloatWeights-testLabels.txt](http://eecourses.technion.ac.il/048874/HW1-fall-2016/1000HU_BinaryFloatWeights-testLabels.txt), contains the Net's inference output labels on the MNIST test set. Use it to compare your results and validate that your code functions properly.
10. Once your network works properly, let’s measure performance. We define three times of execution:
    1. T\*, time to solve a problem on one processor using the best sequential algorithm.
    2. T­P , time to solve a problem on a parallel processor with P cores using a parallel algorithm.
    3. T­1 , time to solve a problem on a parallel processor using a parallel algorithm with only one core running.

We are not interested in T\*. Let’s compute T1. Eventually (in future HW assignments), we will take two more steps: First, move the algorithm to a simple simulator of a parallel machine, running on your same machine, and start with only one core. We wish T1 then to be roughly the same as the one you get now at this homework. Next, we will parallelize the code for any P, and we wish that when trying P=1 we will indeed achieve T1 execution time.

Trusting Windows OS to measure execution time (or cycles) is not reliable for most users and the outcome may be tainted by sampling uncertainties and by poor separation of user processes from system tasks. We bypass that limitation by creating our own counters, as follows.

1. Consider the following simple code segment for matrix-vector multiplication (useful for computing the linear combination in a layer of the neural net):

#define ROWS 1000

#define COLS 784

float HiddenWeights[ROWS][COLS], Input[COLS], HiddenOut[ROWS];

// Load HiddenWeights and Input from file

int i,j;

float sum;

for (i=0; i< ROWS; i++) {

sum = 0;

for (j=0; j< COLS; j++)

sum += HiddenWeights[i][j] \* Input[j];

HiddenOut[i] = sum;

}

1. Let’s add performance counters (red code below). We need to count not only the floating-point Multiply and Add visible in the code, but also the invisible code for loop management, array indexing and loading and storing to memory, as defined by the extra #define statements. Actual times are arbitrary below—use them for this homework; they should be carefully selected to reflect the architecture and microarchitecture of the parallel processor under study.

#define t\_FPMULT 4

#define t\_FPADD 2

#define t\_FPDIV 16

#define t\_LOAD 2

#define t\_STORE 2

#define t\_FXMULT 1

#define t\_FXADD 1

#define t\_BRANCH 1

#define ROWS 1000

#define COLS 784

float HiddenWeights[ROWS][COLS], Input[COLS], HiddenOut[ROWS];

// Load HiddenWeights and Input from file

int i,j;

float sum;

int PrfCntr = 0;

for (i=0; i< ROWS; i++) {

PrfCntr += 2 \* t\_FXADD + t\_BRANCH; // each loop iteration,

// increment i, compare i to ROWS, branch

sum = 0;

PrfCntr +=1; // 1 cycle for resetting a FP register

for (j=0; j< COLS; j++) {

PrfCntr += 2 \* t\_FXADD + t\_BRANCH;

sum += HiddenWeights[i][j] \* Input[j];

PrfCntr += 2 \* t\_LOAD + t\_FPMULT + t\_FPADD;

// highly architecture dependent

}

HiddenOut[i] = sum;

PrfCntr += t\_STORE ;

}

// report PfrCntr

1. Submit by 10 December 2017, using email to ran@ee with subject line “048874-F2017-HW1”:
   1. Your code
   2. Performance – total and for each step of the inference computation. Invent a good graphic way for showing how performance develops over the steps. Submit a short PDF explanation and your graphing code (e.g., excel tables or Matlab).

HW1 FAQ

Quick Links

[*Q: Where did the Weights File Come From?*](#_Toc468177436)

[*Q: What is the Structure of the Weights File?*](#_Toc468177437)

[*Q: What is the Purpose of the Labels File?*](#_Toc468177438)

[*Q: What is the Structure of the Labels File?*](#_Toc468177439)

[*Q: How to Parse the MNIST dataset?*](#_Toc468177440)

## Q: Where did the Weights File Come From?

We have created a neural network with one hidden layer, containing 1000 neurons, and trained the net for a short time on the MNIST training set. We than wrote all net's weights to a single file, which we call 'weights file'.

## Q: Which the Activation Functions are Used?

The hidden layer neuron use a ReLU activation function.

The output layer, since the neural network is used for classification, uses softmax.

## Q: What is the Structure of the Weights File?

The weights file contains a sequence of 4-byte single precision floating point numbers (*float* typein C). There are no delimiters such as commas, space or newline characters.

Each weight is encoded in little-endian format, the standard encoding of an Intel processor.

We will refer to the numbers in the weights file as 'floats'. Assume we start counting from 1, float 1 matches weight 1, of neuron 1, in hidden layer 1 (). The last weight of neuron 1, , is the 785th float and matches the bias input of that neuron. The bias is assumed to be a constant +1 input. The 786th float is weight 1, of neuron **2** in hidden layer 1. The first hidden layer has a total of (784+1)×1000=785,000 weights.

The weights of layer 2, in our case it is the output layer, start immediately after the weights of the first hidden layer, I.e., float 785,001.

An accompanying file containing an example program for weights-parsing accompanies the HW.

## Q: What is the Purpose of the Labels File?

Every image in the MNIST test set has its correct class, available for download in the [MNIST website](http://yann.lecun.com/exdb/mnist/). However, no neural network is 100% accurate and will classify at least some test set images wrong. Therefore, each neural net has its own classification for the test set.

The labels file accompanying the HW contains the labels assigned to the MNIST test set images by the neural network used to produce the weights file. The purpose of the labels is for you to verify your implementation of a feedforward neural network by comparing the classification of your network with that of our original net.

*Note:* Our neural network wasn't trained to reach high accuracy on the MNIST test set. Therefore, its classifications will mostly mismatch the correct ones. It has low accuracy rate (about 18%).

## Q: What is the Structure of the Labels File?

The labels file contains 10,000 lines, one for each test set image. Every line contains the sample number and its label, separated by a comma. Every line also ends with a *newline* character ('\n'). I.e., the structure of a line is as follows: *Sample number, label.* It is a human readable format, you can open the file with a simple text editor and see its contents.

## Q: How to Parse the MNIST dataset?

In addition to the weight and label files, an example file for parsing the MNIST dataset written in C is part of the HW files. The MNIST dataset file structure is explained in the [MNIST website](http://yann.lecun.com/exdb/mnist/).

If you are using a different language than C, first try to search for a parsing code online, to be used as a starting point for you to edit later. The MNIST dataset is old and popular enough to have an existing parser in every available programming language.