Assignment P3 – Neural Network

Formal assignment description for P3 - INFOMOV
Jacco Bikker, 2018

Universiteit Utrecht

Introduction

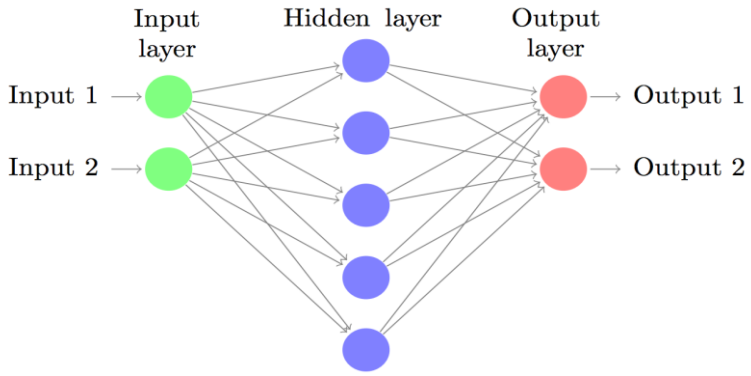
This document describes the requirements for the second assignment for the INFOMOV course. For this assignment, you will improve the performance of the training phase of a neural network, using vectorization.

Neural Network

The supplied C++ application implements a basic backpropagation neural network, and is based on code by Bobby Anguelov. The net is trained to recognize handwritten characters, using the MNIST training data set, which consists of 60,000 categorized images of 28x28 pixels. To keep training time reasonable only a subset of this data is used. You can find the implementation of the neural net training code in source file `NeuralNetwork.cpp/.h`.

The point of this assignment is obviously not to learn about neural networks, but to better understand your task, the main concepts are outlined below.

A neural net is a collection of virtual 'brain cells', which take some input and may produce output:



In this image, as in the application, three layers are used. The *input layer* is fed input data. In our case this is a 28x28 pixel image; we therefore use 784 input cells. The input cells are connected to the *hidden layer*. Each connection has a *weight*, which scales the value provided by the input layer. The cells in the hidden layer each sum the incoming weighted values, and if the result exceeds a certain threshold, a value is sent to the *output layer*, again using weighted connections. In the application the output layer consists of 10 cells: one for each number between zero and nine. We read the result by finding the output with the highest value: this is the 'best guess' of the net. This functionality is implemented in the `Evaluate` function in `NeuralNetwork.cpp`.

Training the network is a matter of adjusting the weights. The functionality for this can be found in the `Backpropagate` function, which uses the difference between the result produced by `Evaluate` and the desired result to improve the weights.

We train the network using a large number of images, for which we know the correct classification. Training happens in waves, called *epochs*. For each epoch, we feed the network all images of the training set, and validate the result using a second (smaller) set of different images. This functionality can be found in function `RunEpoch`.

Running the original code makes clear that training is time consuming: a single epoch, using only 4,000 of the 60,000 MNIST images, takes roughly four seconds. The network requires about 300 epochs to reach a reasonable level of accuracy, and reaches optimal (not perfect) accuracy after about 600 epochs, which takes about 40 minutes.

There appears to be plenty of room for parallel execution: there are no loop dependencies in the for loops in Evaluate and Backpropagate, which suggests we can easily update many cells at the same time.

Data Layout

The data for the neural net is specified in class `Network`. We have simple arrays of floats for the cells themselves: `inputNeurons`, `hiddenNeurons` and `outputNeurons`. Additional arrays store the weights for the connections between the input layer and the hidden layer (`weightsInputHidden`) and between the hidden layer and the output layer (`weightsHiddenOutput`).

The training set is stored in a separate class `TrainingSet`, which contains an array of `TrainingEntry`'s. Each TrainingEntry stores greyscale values for the pixels (scaled to 0..1 and stored as floats), and a set of expected values: 0 for each wrong number, 1 for the correct one.

Finally, we have a collection of data needed for backpropagation: `deltaInputHidden`, `deltaHiddenOutput`, `errorGradientsHidden` and `errorGradientsOutput`.

Vectorization

For this assignment, you will apply SIMD (using SSE or AVX) to speed up the training phase. This means that you will be modifying the three functions called by `RunEpoch`: `Evaluate`, `Backpropagate` and `UpdateWeights`. On top of that you may need to alter the data layout to be more suitable for vector operations.

After vectorization, the code should still be able to train the network. Do not expect exactly the same results: in my experiments the code is not entirely deterministic, even without SIMD code; given the large number of floating point operations any change is expected to yield slightly different results. For reference, your MSE after 50 epochs should be in the range 0.0135...0.0138, and the accuracy of the network should be in the range 43.5%...44.2%.

Vectorization is expected to yield a significant increase in efficiency. For reference: if your original training time is 3 seconds per epoch, you should at the very least expect training times below 2 seconds after vectorization with SSE, and below 1.5 seconds with AVX.

Team

You may work on this assignment alone, or with one partner. You may team with one partner for all assignments, but it is also allowed to change teams per assignment. You cannot change your team

halfway an assignment; if for whatever reason you don't want to finish the project with your partner, both of you will work alone. Both team members may continue working with the code that was produced up till the split.

You may exchange information about the project with other students, online or in real life. Do not share code snippets, limit the exchange to ideas, hints, and concepts.

Deliverables

For this assignment you may hand in just the project files. Make sure the code compiles out-of-the-box in VS2017. If any other tools are required to produce the intended executable, please add a `readme.txt` that contains build instructions.

Grading

The grading for this assignment is based on the performance of the submitted program, running on an Intel Core i7-7700HQ @ 3.6Ghz. On this machine, the original program roughly takes ~2900ms per epoch. Reduce this to 1800ms for a 6; to 1500 for a 7. For an 8, you need to either reduce epoch time to 1100ms using SSE (__m128), or to 900ms using AVX (__m256). Grades higher than 8 are reserved for even better performance levels. Best performance in 2017 was 240ms using AVX.

Important: your performance improvements must primarily be the result of the use of SIMD.

Deadline

The deadline for this assignment is **Thursday October 18th, 23:59**. Please submit your work using the SUBMIT system. If you fail to meet this deadline, you may submit one day later. One point will be subtracted from your grade in this case.

Academic Conduct

The work you hand in must be your own original work, or properly referenced. If you used materials from other sources, please specify this clearly.

Do not store your work in a publicly accessible location. If other students use your work (now or in the future), you may be reported along with the perpetrators.

The End

Questions and comments:

bikker.j@gmail.com or room 4.24.

INFOMOV 2018