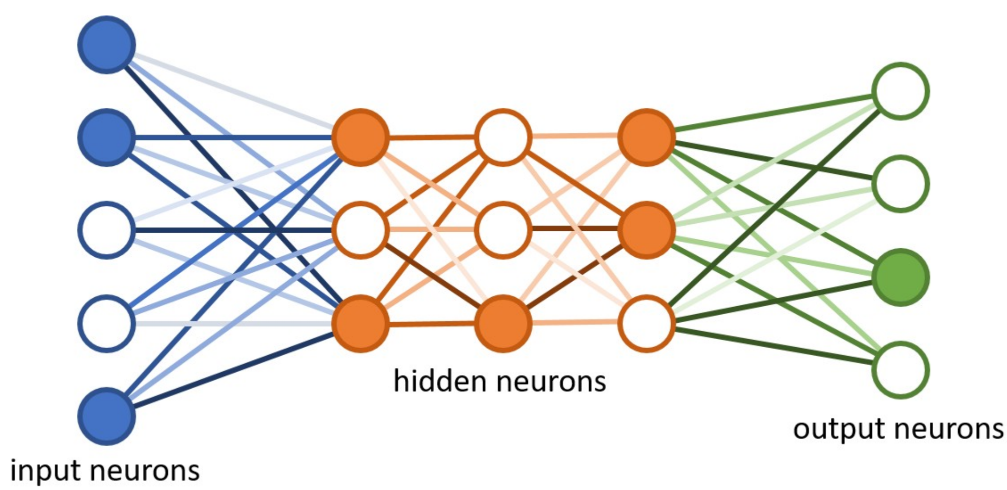
Profiling an Artificial Neural Network implementation

Francesco Cicala

Assignment 2

Abstract

In this assignment we have selected a non-trivial C code and profiled it by means of a popular profiling software, Valgrind; in particular, we used the tools Memcheck, Callgrind and KCacheGrind. The code is the implementation of a simple neural network solving the XOR problem. It was provided by GenANN (https://github.com/codeplea/genann), a minimal library used for generating and training feedforward artificial neural networks in C.



1. Memcheck

Valgrind memcheck is a tool to detect memory errors in an execution. Memory errors could be:

- accessing memory after it has been freed;
- incorrect freeing of heap memory;
- memory leaks, i.e., memory was allocated and not freed before the program termination.

Firstly, we compiled the script and executed it with the following commands:

```
gcc -c example1.c -Wall -Wextra
gcc -c genann.c -Wall -Wextra
gcc example1.o genann.o -o example1.x -pg -lm -Wall -Wextra
./example1.x
```

And the output of the execution was:

```
GENANN example 1.
Train a small ANN to the XOR function using backpropagation.
Output for [0, 0] is 0.
Output for [0, 1] is 1.
Output for [1, 0] is 1.
Output for [1, 1] is 0.
```

After that, we profiled the memory usage of the execution:

```
valgrind --verbose --show-leak-kinds=all --leak-check=full
--track-origins=yes --log-file=valgrind.out ./example1.x
```

The output was:

```
==17118== Memcheck, a memory error detector
==17118== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==17118== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==17118== Command: ./example1.x
==17118== Parent PID: 15821
==17118==
 [...]
==17118== embedded gdbserver: reading from
/tmp/vgdb-pipe-from-vgdb-to-17118-by-fra-on-???
```

```
==17118== embedded gdbserver: writing to
/tmp/vgdb-pipe-to-vgdb-from-17118-by-fra-on-???
==17118== embedded gdbserver: shared mem
/tmp/vgdb-pipe-shared-mem-vgdb-17118-by-fra-on-???
==17118==
==17118== TO CONTROL THIS PROCESS USING vgdb (which you probably
==17118== don't want to do, unless you know exactly what you're doing,
==17118== or are doing some strange experiment):
==17118== /usr/lib/valgrind/../../bin/vgdb --pid=17118 ...command...
==17118==
 [...]
==17118==
==17118== HEAP SUMMARY:
==17118==  in use at exit: 0 bytes in 0 blocks
==17118== total heap usage: 3 allocs, 3 frees, 11,924 bytes allocated
==17118==
==17118== All heap blocks were freed -- no leaks are possible
==17118==
==17118== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==17118== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

In the Heap Summary we see that no memory errors were generated, that is, all the heap blocks that were allocated were also freed before the program termination.

2. Callgrind and KCacheGrind

By calling callgrind with argument the executable we considered in the last section, a file callgrind.out is generated. We analyzed this file by means of KCacheGrind, a graphic user interface. Below, the time elapsed in the different functions is shown:

99.60	0.81	1	main
97.32	56.55	40 000	genann_train
40.78	24.90	40 004	genann_run
10.26	10.26	120 012	genann_act_sigmoid_ca...
9.65	2.81	80 008	genann_act_hidden_ind...
4.82	1.40	40 004	genann_act_output_ind...
1.40	1.40	40 004	__memcpy_avx_unaligned
1.39	0.00	1	genann_init
1.38	0.13	1	genann_init_sigmoid_lo...
1.25	0.21	4 096	genann_act_sigmoid
1.04	0.14	4 096	exp

As we see, the most of the time is passed in the function genann_train, which is the method by which the parameters of the neural network are learned until their convergence. genann_train calls genann_run, which is the method which processes the output for a given train input, so that a loss value is computed by comparing the output and the expected result.