IWLS 2020 Programming Contest: ML+LS

Alan Mishchenko (UC Berkeley) and Satrajit Chatterjee (Google AI)
iwls.contest.2020@gmail.com

Overview. The goal of this contest is to learn an unknown boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ from a training set consisting of input-output pairs. The training set is chosen at random from the 2^n possible input minterms of the function (and typically is much smaller than 2^n).

Inputs and Outputs. The training set is provided in the Espresso PLA format. The learned function should be in the form of an And-Inverter Graph (AIG) with *n* inputs and a single output in the Binary AIGER format. However, to make things more interesting, the AIG should have no more than **5000** And gates.

Now, in addition to the training set, we also provide a validation set (also in the Espresso PLA format). You can see how well you have learned the function by evaluating it on the validation set and by counting the fraction of minterms in the validation set on which your learned function agrees with the unknown function, i.e., by measuring the accuracy on the validation set.

Evaluation. However, we will evaluate your function not on the validation set, but on a private test set. (So, if you are feeling brave, you can use the validation set as additional training data to improve your accuracy, but watch out for overfitting!)

There are 100 functions in the benchmark set, i.e., 100 training and validation sets (in files named `exNN.train.pla` and `exNN.valid.pla`). You can download them here.

Your overall score is the sum of test accuracies over these 100 benchmarks. If your AIG for a benchmark is bigger than **5000** And gates, then the accuracy on that benchmark will be considered to be 0. Note that it should be trivial to get a score of at least 50. Do you see why?

(In case the overall score as defined above is not a telling metric to separate the entries, we may consider alternatives such as average rank, etc.)

Submission. We would like you to submit both the program used to generate AIGs given the training data as well as the actual AIG files for each benchmark in a tarball. The AIG corresponding to `exNN.train.pla` should be named `exNN.aig`. (You are free to use any programming language and any open source libraries that are useful but we should be able to reproduce your results independently.)

Tools. You can evaluate the learned AIGs using the command `&mltest` in ABC. For example, assuming that `exNN.aig` is the AIG synthesized using `exNN.train.pla`, it can be evaluated on `exNN.valid.pla` as follows:

```
abc 01> &read exNN.aig; &ps; &mltest exNN.valid.pla
exNN.aig : i/o = 43/ 1  and = 186  lev = 27 (27.00)  mem = 0.00 MB
Finished reading 6400 simulation patterns for 43 inputs.
Probability of 1 at the output is  74.56 %.
Total =  6400.  Errors =  890.  Correct =  5510.  (86.09 %)
```

In this case, the AIG has 186 internal And gates (so it is within the **5000** node limit), and the accuracy on the validation set is 86.09%. Note that `&mltest` also prints the accuracy of naive guessing (i.e., simply learning a constant function) which in this case is 74.56%.

Parting Remarks. One simple solution to this problem is to create a sum-of-products (SOP) from the training minterms and to convert that to an AIG, but of course, we should not expect that to generalize well to the test set. But perhaps if we use Espresso to minimize the SOP by treating the training set minterms as the on-set and the off-set and the rest as don't cares (`.type fr` in the Espresso PLA format) we can do better? Or how about the method described in this paper? Or perhaps even train a neural network and compile it down to an AIG? We can't wait to see what you come up with.

All the best!