Relation of work project for Intelligent Systems course

Compressed Network in EML

Ugo Marchesini 1

[1] Affiliation 1; ugo.marchesini@studio.unibo.it

Abstract:neural networks are applications of Machine Learning, they allow to solve regression and classification problems by modifying their weights and bias during the training phase. More complex networks in general allow to solve problems with more complex datasets, but at the cost of increasing the network footprint in terms of training time, prediction, disk size and network load in case of application performed remotely. Even neural networks with very high precision do not provide a clear understanding of the nature of the problem or of how the network "solves" the problem. A neural network modifies its weights and bias to better fit the predicted value with the expected value for the purpose. to improve prediction accuracy, it is clear that as the size of the trainable parameters increases, the time and energy cost of the training itself increases. Networks like CNNs (ResNet, Inception, VGG) contain up to tens of millions of weights, their accuracy can be very high, but their footprint is also very high. The compression techniques of neural networks allow to obtain reduced footprints with acceptable losses of precision. This does not solve optimization problems, where it is necessary to impose an objective function and define constraints between variables. EML allows to incorporate an ML model into a combinatorial problem, in this case a neural network. A combinatorial problem being declarative could allow a better understanding and therefore a better "explainability" of the problem learned from a neural network. The field of application of this research is transprecision computing. The aim of this research is the exploration of the compression techniques of neural networks and its incorporation in a combinatorial optimization model in order to obtain a pipeline that processes a dataset obtained from the processing of different computing functions on different hardware.

Keywords:neural network; machine learning; footprint; compression; combinatorial optimization;

transprecision computing

1. Transprecision computing

Transprecision Computing is a paradigm that allows users to trade the energy associated with computation in exchange for a reduction in the quality of the computation results. Guaranteed numerical precision of each elementary step in a complex computation has been the mainstay of traditional computing systems for many years. The possibility of obtaining an acceptable precision below a predetermined error with a lower use of bits in FP operations allows energy savings opens up to the possibility of obtaining a greater computing power without a proportional energy cost.

2. Empirical Model Learning

EML can be considered as a technique that allow to encapsulate the learned model in a number of optimization techniques. Library used codifies each neuron as a variable in the combinatorial problem, while each edge is considered as a constraint on the neurons connected.

3. Compressed Neural Network

Among the various compression techniques, the research took into consideration: weight pruning and quantization, where weight pruning is intended to set the weight value to zero. The TensorFlow Model Optimization Toolkit (TMOT) was used for both techniques. TMOT allows the conversion to a compressed tflite format which, however, is not useful in order to use it within a combinatorial problem as the EML library accepts models in Keras or PyTorch which are incompatible with the tflite format.

4. Dataset

The dataset was taken fromhttps://zenodo.org/record/6575841#.YtE3kS8QNQJ, with subsequent additions.

The dataset consists of 9 files divided by benchmark and hardware.

There are 3 types of benchmarks: Convolution, Correlation and Saxpy, and they correspond to as many algorithms used to produce samples.

There are 3 types of hardware: g100, pc and vm.

Each file has a name like <benchmark> _ <hardware> .csv

The samples contain variables (var_0, var_1 ... var_n) whose content is the number of bits used in the calculation, the error (error), the time taken (time), the average memory usage (memory_mean) and the peak memory usage (memory_peak).

5. Used Tools

The environment used is Visual Studio 2022, the Python language for the ML part and C # for the data preparation. The libraries are: TensorFlow \ Keras, Emlib and Util.

6. Process

In a first phase, compression, quantization and pruning techniques were explored on an epidemiological problem (first phase), and subsequently on the transprecision computing problem (second phase).
The Epidemics problem implemented is the version with n neural networks of an existing pipeline taken from a course by Prof. Lombardi on GitHub (phd-ml-co-2021-02 / notebooks at main phd-course-ml-co-2021 / phd-ml-co-2021-02 GitHub). The pipeline of the original example is as follows.

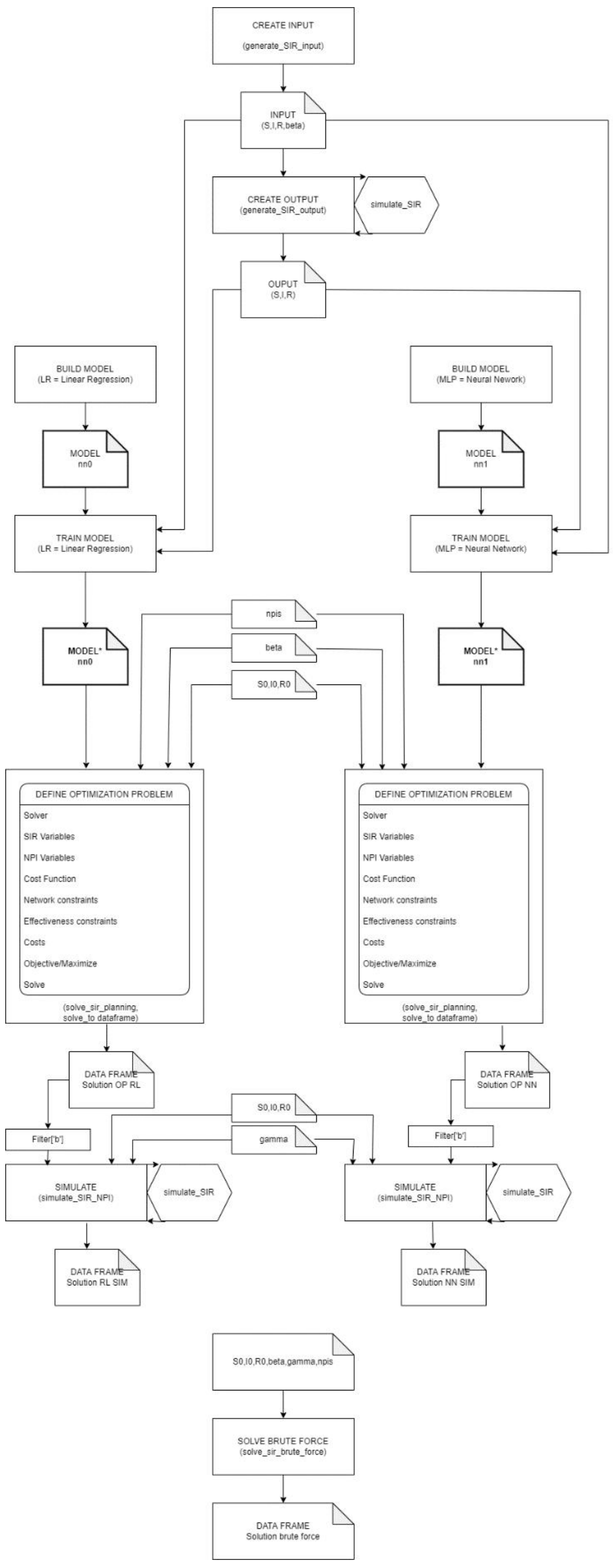


Figure1: Original Pipeline Epidemics

The version with n neural networks is the following.

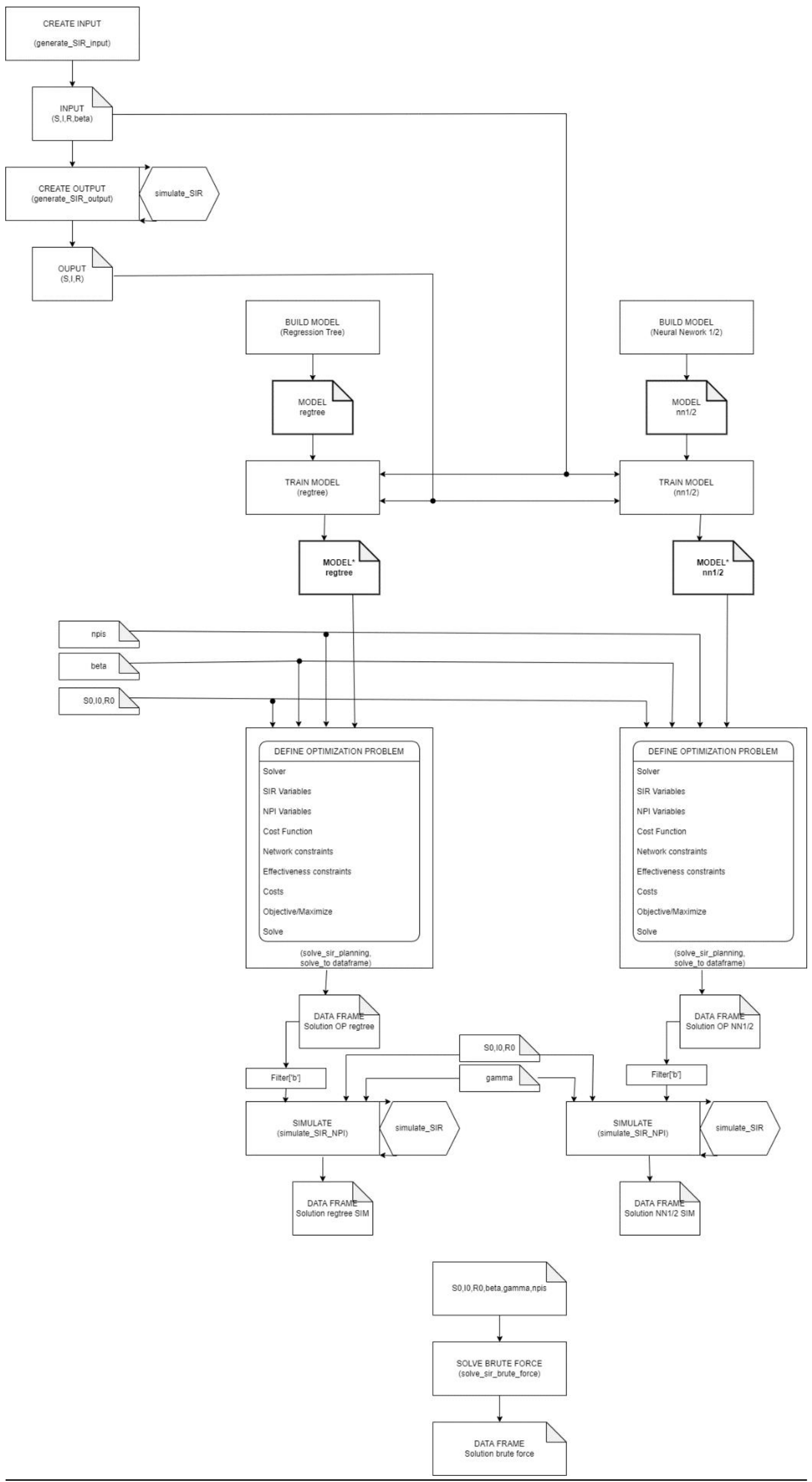


Figure2: Pipeline for multiple NN Epidemics

6.1. Epidemics

The first example on which to explore compression techniques was a dataset of a SIR (Susceptible, Infectious, or Recovered) epidemic model.
The purpose of the model is the prediction of SIR values and the objective function of the combinatorial optimization problem is the minimization of the cost of an NPI intervention (non pharmaceutil interventions)

6.1.1. Build and Train Model

Functions have been used to create configurable input and output data. The functions create the data from scratch if required and save it, or load the data if it exists.

Two types of networks were trained, one with DecisionTreeRegressor and the other with Keras models with 3 different topologies. They essentially differ only in depth and the number of neurons per layer.

All networks are saved with the extension h5.

6.1.2. Evaluate

The evaluation of the networks was carried out with the accuracy obtained during the training phase, the rmse (root mean square error), mae (mean absolute error) and r2 (coefficient of determination).

The test dataset is 20% of the train dataset.

6.1.3. Post Training Quantization

According to the TensorFlow documentation: “Post-training quantization is a conversion technique that can reduce model size while also improving CPU and hardware accelerator latency, with little degradation in model accuracy. You can quantize an already-trained float TensorFlow model when you convert it to TensorFlow Lite format using the TensorFlow Lite Converter. "

6 quantization techniques are offered: Dynamic range quantization,Full integer quantization, Integer with float fallback, Float16 quantization, Integer only: 16-bit activations with 8-bit weights (V1) and Integer only: 16-bit activations with 8-bit weights + builtins (V2)

In order to evaluate the metrics of these techniques, it is necessary to train the model (baseline) before converting to a quantized and compressed model in tflite format.

The metrics are: size, rmse,mae and r2;

6.1.4. Pruning

TMOT offers the possibility of pruning a trained network, the size of the network after pruning is greater than the baseline network as the ultimate goal is compression through TFLite conversion.

6.1.5. Compress

TMOT allows the compression of nn in a tflite format, this format is not useful in order to insert a combinatorial optimization problem offered by the emlib libraries as the network is not a Keras type which instead uses emlib.

6.1.6. Simulation and Brute Force

The combinatorial optimization problem outputs the beta vector (infected per day) which is used together with the initial values of S0, I0 and R0 in the simulation function provided by emlib.

The same is done with the Brute_Force function to check the validity of the process pipeline of the dataset.

6.1.7. Conclusion

The metrics detected concern the size of the compressed models compared to the baseline and the accuracy of the predictions. In the following table we used a network with 4 layers and 8 neurons per layer (4X8).

It has been found that with the increasing complexity of the network it is not guaranteed to obtain an optimal or feasible solution out of the combinatorial problem, for this reason a relatively simple model has been chosen (4X8)

	TrainingTime	size	accuracy	loss	rmse	mae	r2
Reference	74573	22176	0.988	6.68519e-05	0.00817	0.00578	0.99711
Pruned	24947	33984	0.989	5.25832e-05	0.00725	0.00499	0.99770
DynamicRangeQuantization		3576			0.00817	0.00578	0.99711
FullIntegerQuantizationIntegerOnly		3392			2.14685	1.62933	0.99687
FullIntegerQuantizationIntegerWithFloatFallback		3736			0.00811	0.00607	0.99708
Float16Quantization		4304			0.00818	0.00580	0.99710
IntegerOnly16BitActivationsWith8BitWeightsV1		3576			0.00817	0.00578	0.99711
IntegerOnly16BitActivationsWith8BitWeightsV2		3576			0.00817	0.00578	0.99711

The evaluation that can be given is that the compression offered by TMOT is interesting, although it cannot be inserted within an optimization problem due to the different data structure incompatible with emlib, and the precision of a pruned network does not differ from the baseline. but it has a higher file size. In the next phase it will be necessary to investigate whether different types of nn topologies allow an improvement in the solving time of the optimization problem.

Having verified the feasibility of implementing the same pipeline on a different environment, the original pipeline is on Jupiter Notebook while the current one and the Transprecision Computing problem are on the VisualStudio \ Python stack, and gained confidence with the libraries, we moved on to the second phase.

6.2. Transprecision Computing

The execution of the pipeline is carried out with a battery of networks on the whole dataset, therefore for each network the baseline version and the pruned version for each benchmarck are processed, so there are 6 different logs for each processed network.

The runs are defined as “run”, a folder with the name run <datetime> is created which is the destination for the logs, templates and **run.summary.csv**.

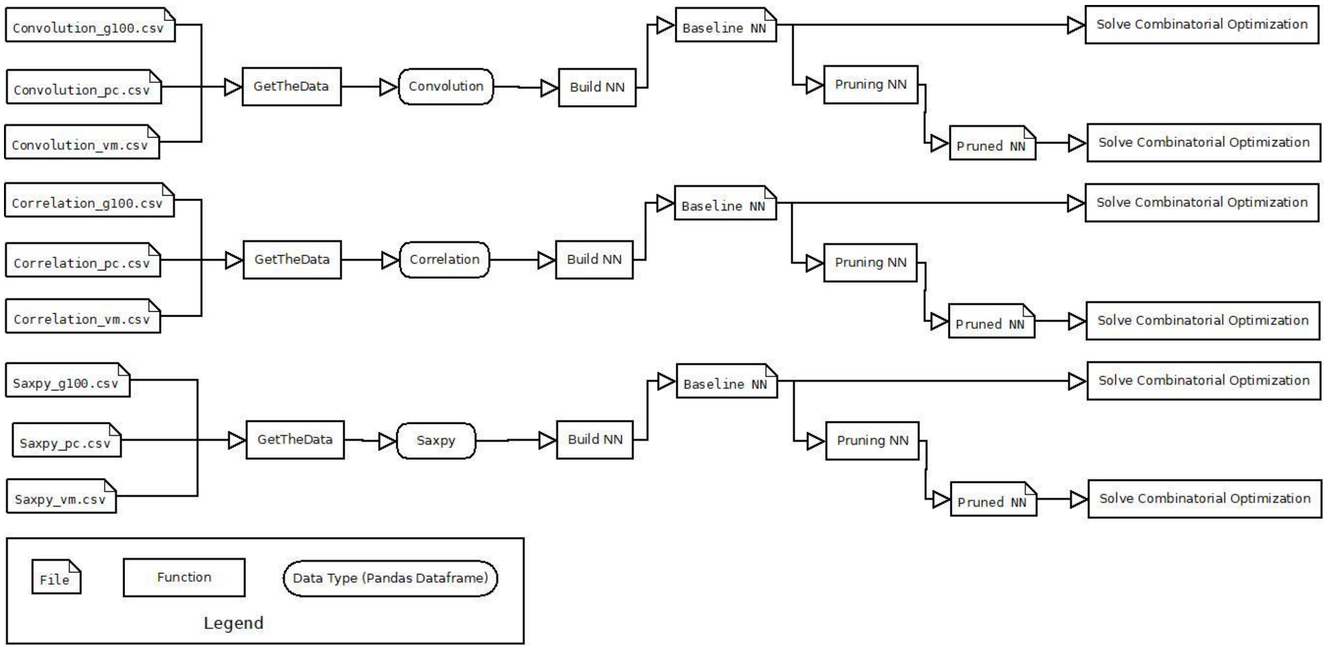


Figure3: Pipeline Tpc

Block with same name means it’s the same data\function.

6.2.1. GetTheData

The baseline is divided into 9 files: 3 for benchmarks (Convolution, Correlation and Saxpy) and 3 for hardware (g100, pc and vm).

The models are benchmark oriented so a function has been developed that transforms the dataset of each benchmark into a one-hot-encoding dataframe for the values of g100, pc and vm.

The values were normalized in the interval between 0 and 1, the upper and lower bound values were stored to denomalize the predicted results.

6.2.2. Build and Train Model

As in the previous phase it was necessary to train various networks using the present function in Util. In this phase, the accuracy, loss, traning time and size are obtained.

6.2.3. Evaluate

The metrics obtained are: evaluation time, rmse,mae and r2

6.2.4. Pruning

As for Epidemics, network pruning is performed. The parameters used are initial_sparsity and final_sparsity and indicate the percentage of weights to be set to 0 at the beginning and at the end of the pruning process.

6.2.5. Encoding

The objective function is the minimization of the number of bits of the variables, with constraints on time and error with preset value.
The constrains used are : time < 100 , error < 0.01.

6.2.6. Metrics

The build, pruning and encoding functions output detailed logs for each model and for each benchmark and part of these values are collected in a file called run.summary.csv.
The following table contains the metrics collected in the run.summary.csv file

First name	**Description**
Topology	Describes the topology is of the type: <neurons layer> <space> <neurons layer> ... <neurons layer>
Train	Training time in ms
Size	Byte of file h5
Evaluate	Time in ms of exucution of the rmse, rme and r2 funtions
Accuracy	Accuracy obtained from the training phase
Rmse	Root mean square error
Encode	Time in ms for encoding in the optimization problem
Solver	Time in ms of solution of the optimization problem
Objective	Result of the objective function (sum of the input variables var_x) normalized
Status	Status of the solution of the optimization problem (OPTIMAL, FEASIBLE ...)

Tabella 1 : Table metrics of run.summary

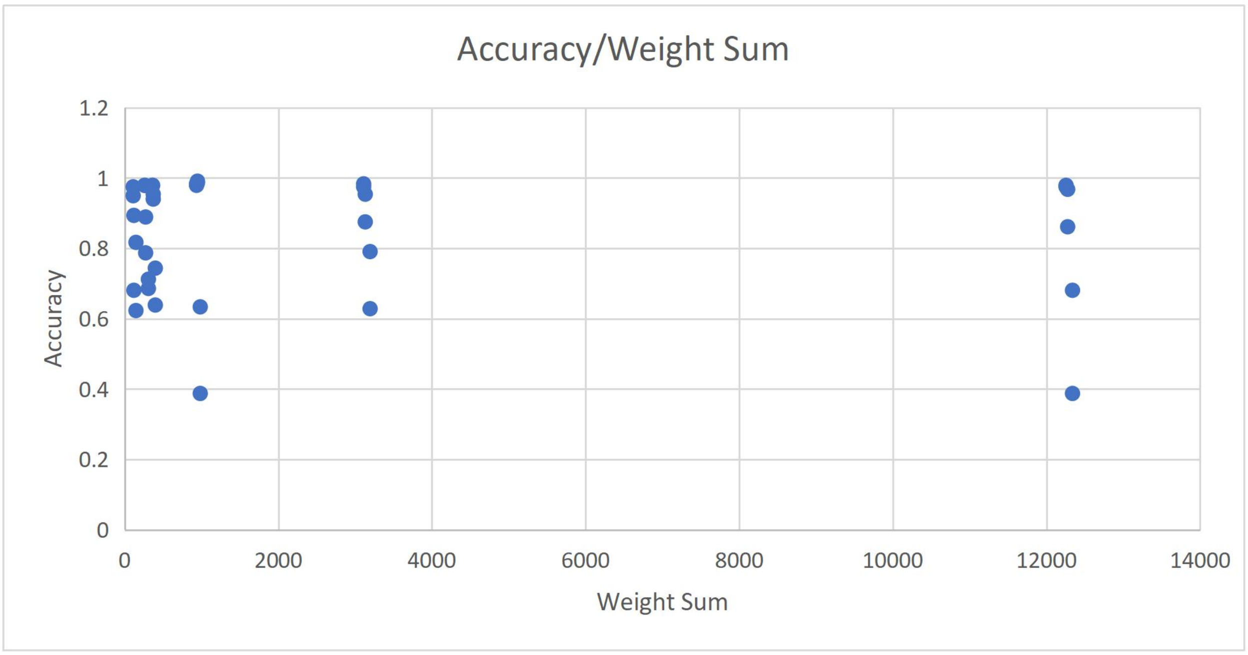
6.2.7. Result

The evaluation is performed on a specific run (run20220901103017) in which 6 different networks were used.

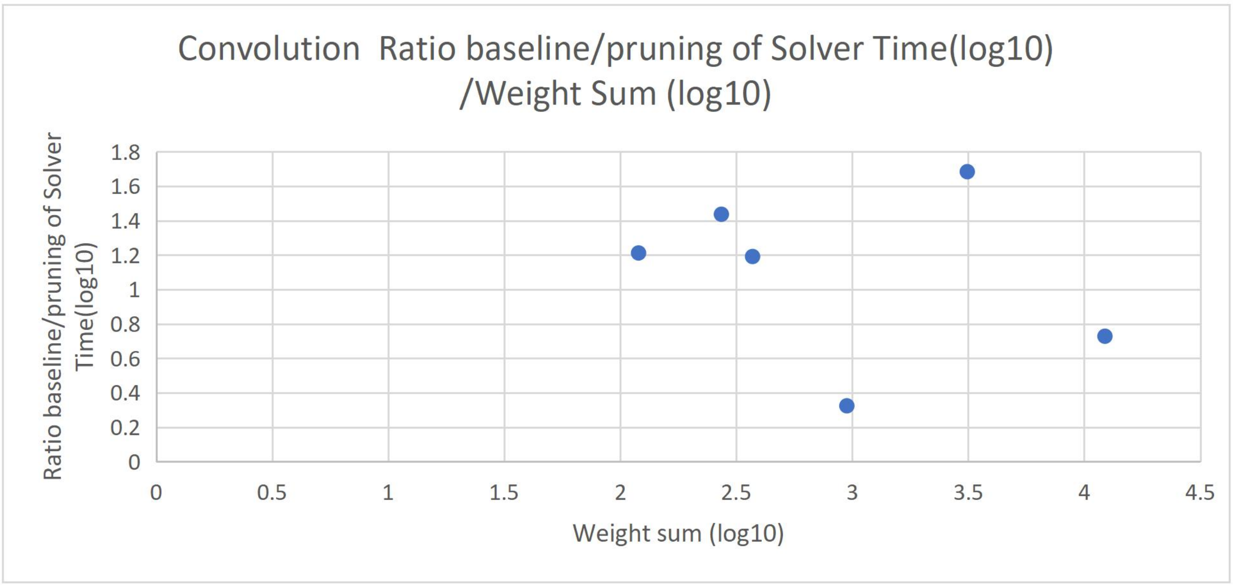
run.summary.csv

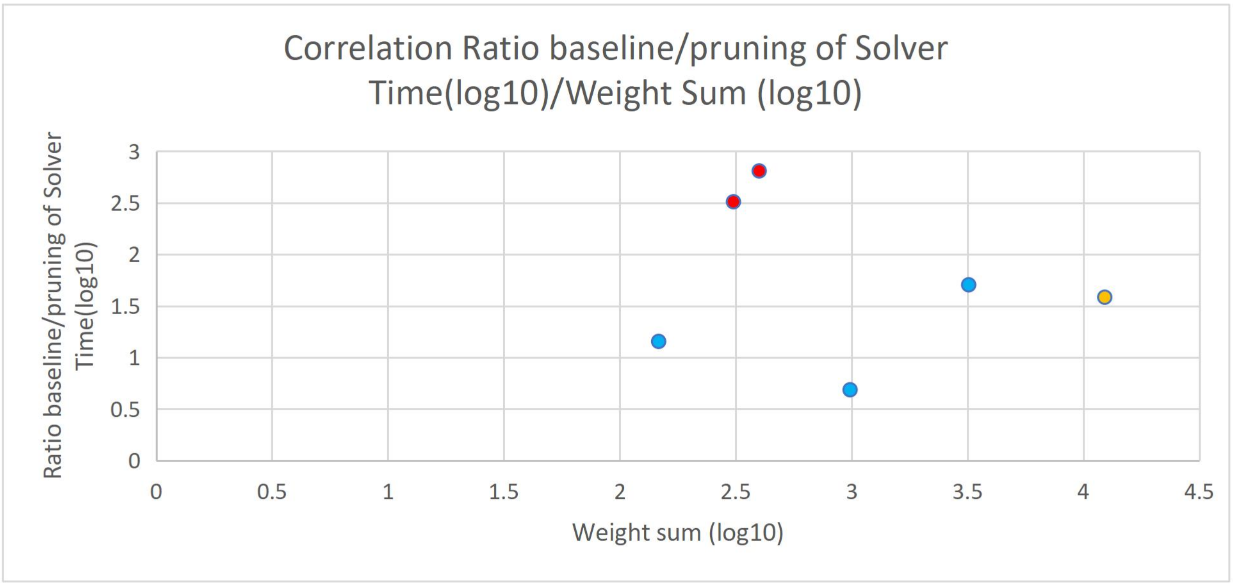
It is possible to provide some evaluations:

- Larger networks don't provide better accuracy
- The Saxpy dataset has better accuracy
- Pruned networks have a shorter solution time of the optimization problem than the corresponding baseline

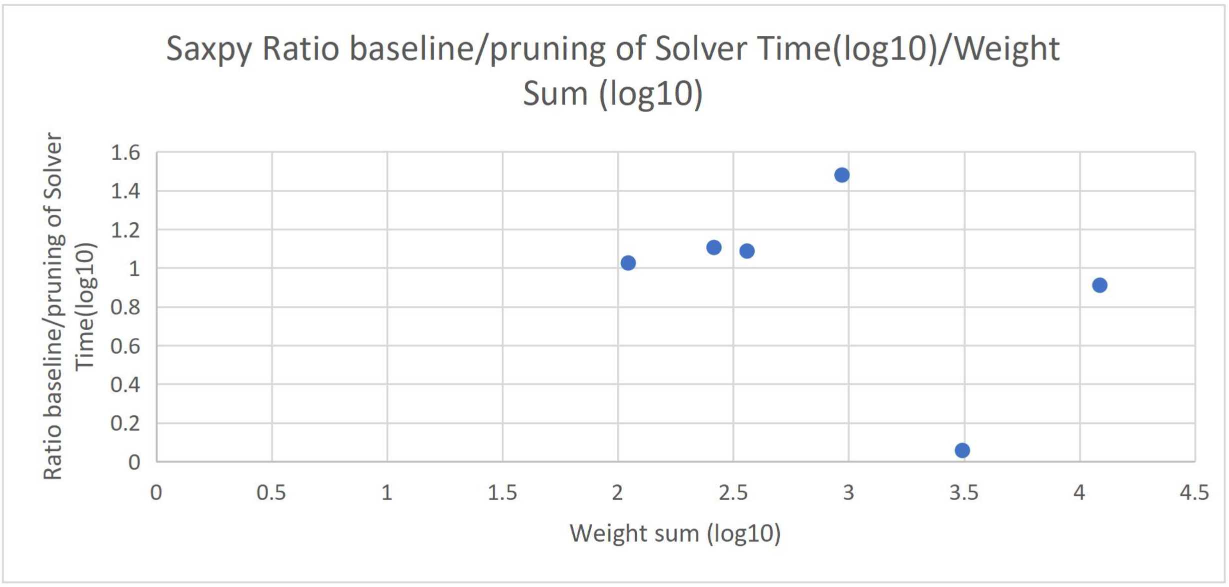


Nexts plot are log10 of solver time (see Table 1 : metrics of run.summary) of the ratio between baseline and pruning version of the same model in Y axes and log10 of weight sum in X axes





The red points are INFEASIBLE problems, the yellow points NOT_SOLVED, all the others are optimal solutions.



In the <benchmark> Solver Time(log10,ms) /Weight Sum (log10) graphs, the abscissas are log10 (topology) and the ordinates are log10 (solver).

It is possible to conclude that the pruned networks always allow, when they obtain an OPTIMAL result, better times than the corresponding baseline networks.

7. Repository

https://github.com/AlmaMaterStudiorum/IntelligentSystems_ProjectWork