NN-PES, a generic parametric equations solver using hybrid neural networks

J-P. Merlet, INRIA Sophia-Antipolis
Jean-Pierre.Merlet@inria.fr

We consider an arbitrary parametric equation system $\mathbf{F}(\mathbf{P}, \mathbf{X} = \mathbf{0}$ with n equations in $\mathbf{F}$ and n unknowns in $\mathbf{X}$ while $\mathbf{P}$ are parameters that appear in $\mathbf{F}$ and are constrained to lie in a bounded domain. Our objective is to develop a specific solver for this system which will provide the possible multiple solutions for arbitrary values for $\mathbf{P}$. Beside providing $\mathbf{F}$ and its jacobian the only input for constructing the solver is an *initial solution set* i.e. a small number of sample $P_j, \{S_j^1, S_j^2, \ldots S_j^n\}$ where S_j^k are solutions of $\mathbf{F}$ for the parameters value P_j and is not necessary to provide all the solutions. With this input we are able to structure the solutions space in different components and create learning sets for each of them. These sets are used to train Multi-Layer Perceptron (MLP) using an original strategy: we classically try to minimize a loss function but we also look for a MLP that maximizes the *success rate* i.e. the number of time the Newton methods with as guess the MLP predictions converges toward the expected solution over the whole learning set (a success rate of 100 implies that hybrid MLP will provide the correct solution for all samples of the learning set). Whenever a success rate of 100 is not reached for a given learning set we use the missed solution(s) to create new MLPs until we get a set of MLPs that will find the solutions for all samples of all learning sets. The solver then just consists in providing the desired $\mathbf{P}$ to all the m existing MLPs, collect their predictions $\{\hat{P_1}, \ldots \hat{P_m}\}$ and run the Newton method with these predictions as guess. If extended arithmetic is available we are able to provide **exact** solutions meaning that we can make the distance between the approximate solution and the real one arbitrary low.

Building the solver is computer intensive while the solving time is very low. We cannot guarantee to find all solutions for any instance of $\mathbf{P}$ but we have implemented a self-learning capacity that allows one to add hybrid MLPs deduced from missed solutions.

We have implemented this approach in a generic software that will be released soon. It has been tested on difficult problems in robotics, mechanical and electrical engineering: typically building the solver requires 60 hours but the solving time is about 500ms for problems that require several hours with alternate methods.