Thanks for the interest of our work!

The code consists of four parts, following the four steps introduced in the technical report ‘Ensuring DNN Solution Feasibility for Optimization Problems with Convex Constraints and Its Application to DC Optimal Power Flow Problems’ and the published version of ‘Ensuring DNN Solution Feasibility for Optimization Problems with Linear Constraints’ in ICLR 2023

The provided codes are for IEEE Case30 and the implementation for the other test cases can be conducted via the similar steps by replacing the configuration file and adjusting the DNN/problem parameters. Please see the following instructions.

1. removing\_non\_active\_constraints: run max-final30.py

It identifies the constraints that is always non-active, which can then be excluded from the following calibration and worst-case violation identification procedures;

1. maximum\_calibration\_rate: run kkt\_milp.py

It identifies the maximum calibration rate allowed that can still guarantee the feasibility of all the load input;

1. sufficient\_DNN\_size: run run1.sh and run2.sh

We provided different warm-start points to the APOPT solver to find the (sub)-optimal solution of the worst-case input. This corresponds to the program of mainsolve1-mainsolve6 with total 42 initial points; the programs s-main1-s-main4 in parallel finds the possible worst-case inputs among all the existing inputs (including the uniform sampled ones) and regard it as the approximated worst-case input in the entire input region;

Note: one can also try m.max3 or directly apply the mixed-integer reformulation introduced in the paper to express the DNN with ReLU constraints; The Sigmoid functions at the output layer present the same effect of the last two clipped ReLU operations to ensure feasibility of the predicted variables.

1. adversarial\_sample\_algorithm

Step1. First generate training data in the sub-folder 0.generate\_training\_data and run DC\_generate\_data.py to get data with different calibration rate. Here the configuration file case30\_rev\_xx.py refers to the slack bus capacity with different calibration rate, e.g., case30\_rev\_070.py refers to the slack bus capacity with 7% calibration rate and case30\_rev\_100.py is the default test case. The calibration on the branch limits are included in the DC\_generate\_data.py file, e.g., line87. Here we also scale the branch limit by a certain number for easy of calculation.

Step2. Pre-train the DNN with 7% calibration rate: run PreDCOPF\_case30\_dnn\_cpu-new.py

Step3. Use ASAW algorithm to post-train the DNN: run mainsolve1-mainsolve6 with 42 different initial points; run s-main.py for the post-train with the identified worst-case violation. For each iteration, the post-train stops when the feasibility around the identified worst-case inputs are recovered (line2520 in s-main.py) and the algorithm terminates when no more input with violation can be found (line1327 in s-main.py).

Note: the provided ASAW train the DNN on the entire load region [100%,130%], corresponding to the results in Table V in <https://arxiv.org/abs/2112.08091>; the results in each light-load and heavy-load can be reproduced by the similar pre-train and post-train procedures.

Dr. Tianyu ZHAO

School of Data Science

City University of Hong Kong