AI-based Robust Convex Relaxations for Supporting Diverse QoS in Next-Generation Wireless Systems

Steve Chan
Vit Tall and University of Arizona
Orlando, USA
schan@vittall.org

Marwan Krunz
University of Arizona
Tucson, USA
krunz@arizona.edu

Bob Griffin
University of Arizona
Tucson, USA
bobgriffin@me.com

***Abstract*—Supporting diverse Quality of Service (QoS) requirements in 5G and beyond wireless systems often involves solving a succession of convex optimization problems, with varied approaches to optimally resolve each problem. Even when the input set is specifically designed/architected to segue to a convex paradigm, the resultant output set may still turn out to be nonconvex, thereby necessitating a transformation to a convex optimization problem via certain relaxation techniques. This transformation in itself may spawn yet other nonconvex optimization problems, highlighting the need/opportunity to utilize a Robust Convex Relaxation (RCR) framework. In this paper, we explore a particular class of Convolutional Neural Networks (CNNs), namely Deep Convolutional Generative Adversarial Network (DCGANs), to solve not only the QoS-related convex optimization problems but also to leverage the same RCR mechanism for tuning its own hyperparameters. This approach gives rise to various technical challenges. For example, Particle Swarm Optimization (PSO) is often used for hyperparameter reduction/tuning. When implemented on a DCGAN, PSO requires converting continuous/discontinuous hyperparameters to discrete values, which may result in premature stagnation of particles at local optima. The involved implementation mechanics, such as increasing the inertial weighting, may spawn yet other convex optimization problems. We introduce a RCR framework that capitalizes upon the feed-forward structure of the "You Only Look Once" (YOLO)-based DCGAN. Specifically, we use a squeezed Deep Convolutional-YOLO-Generative Adversarial Network (DC-YOLO-GAN), hereinafter referred to as a Modified Squeezed YOLO v3 Implementation (MSY3I), combined with convex relaxation adversarial training to improve the bound tightening for each successive neural network layer and to better facilitate the global optimization via a specific numerical stability implementation within MSY3I.**

Keywords—Quality of Service, 5G Networks, Nonconvex Optimization, Convex Relaxation, Particle Swarm Optimization, Deep Convolutional Generative Adversarial Networks, Numerical Implementation, You Only Look Once, Mixed Integer Non-linear Programming, Robust Convex Relaxation Framework.

I. INTRODUCTION

The cellular industry, including wireless operators and device manufacturers, is racing to deliver the Fifth Generation (5G) wireless technology to end users through three main service categories: Enhanced Mobile Broadband (eMBB), Ultra-Reliable Low-Latency Communications (URLLC), and massive Machine-Type Communications (mMTC). These service categories will support a wide range of QoS needs by existing and emergent applications, such as connected and autonomous vehicles, AR/VR, Industrial IoT, and others. While the concepts of network slicing and Software-Defined Networks (SDNs) offer a framework for supporting diverse sets of QoS, ultimately it comes down to the resource management algorithm within an operator's control plane to ensure that these QoS sets are met without excessive allocation of network resources. Various providers that launched their own 5G networks have made QoS a top priority and articulated the need for new approaches for the envisioned Beyond 5G (B5G) networks, or what is referred to as "6G."

Efficient support for 5G/B5G/6G QoS often necessitates formulating nonconvex optimization problems. Examples include: Radio Resource Allocation (RRA) (whose aim is to maximize the spectral efficiency, subject to certain performance guarantees), Multi-Radio Access Technology (RAT) handling for multi-connectivity (each with its own QoS requirements), and Radio Resource Management (RRM) for connections with varied QoS requirements. The involved optimization formulations are, in essence, mixed integer non-linear programming (MINLP) problems that need to be optimally solved. When the objective and constraint functions are nonconvex, these MINLPs are construed to be nonconvex. For instance, an RRA problem may be formulated as a problem of optimally assigning frequency-time blocks (integer variables) to a number of served connections while simultaneously determining the appropriate transmit powers (continuous variables) for these blocks over various frequency subcarriers.

Prototypical approaches to solving a nonconvex MINLP problem involve transforming it into a convex surrogate, e.g., via reformulation, convex approximation, or a series of convex relaxations. Typically, convex relaxations are derived on a problem-by-problem basis, but there are indeed forays, such as Langevin Diffusions (with the possibility of premature

stagnation of particles at local optima) for nonconvex problems, Alternating Direction Method of Multipliers (ADMM) for nonconvex and nonsmooth functions, and yet others, such as for transforming a nonconvex function to the sum of a smooth function, a concave continuous function, and a convex lower semi-continuous function [1]. Once the nonconvex function has been transformed into a decomposed form, other general-purposes approaches, such as Convex Relaxation Regression (CoRR) and Lassere's Semidefinite Programming (SDP) Relaxation (a.k.a., Linear Matrix Inequality or LMI) can be used. From a numerical implementation standpoint, the aforementioned techniques may not be optimal, in terms of transparency at each neural network layer; this is addressed by the proposed RCR framework.

Historically, various neural network architectures have been experimented with for tackling nonconvex MINLP problems. Each has had its drawbacks, and as one simple example, in several instances, performance tends to degrade with continued training [2]. One approach that has gained great interest due to its robustness and accuracy leverages convex relaxation adversarial training aboard a DCGAN [3]. However, the computational cost has often been quite high [4], as in the case of the YOLO implementation for DCGAN, which is often referred to as a DC-YOLO-GAN. Accordingly, to reduce the computational cost, the notion of fire modules/layers from SqueezeNet (a deep neural network) was utilized to replace convolution layers (a.k.a. Conv) with Fire Layers (FL) [5], and a SqueezeDet adaptation was incorporated for the replacement of certain Conv with Special Fire Layers (SFL) [6]. In essence, this process optimized the DCGAN. The FL and SFL reduced network is referred to as a Modified Squeezed YOLO v3 Implementation (MSY3I). Prior research showed that although the number of parameters in MSY3I are reduced in comparison to a prototypical YOLO implementation, the average precision and accuracy remain relatively high [7].

Given its advantages in terms of the reduced number of hyperparameters to tune, Particle Swarm Optimization (PSO) is often implemented within a DCGAN, [8, 9]. However, a challenge arises when instantiating PSO aboard the DCGAN, as the continuous or discontinuous hyperparameters must be converted to discrete values (e.g., integers) [10]; yet, rounding the calculated velocities to discrete integer values creates an artificial paradigm, wherein particles may stagnate prematurely. Certain techniques, such as increasing the inertia (allowing particles to advance past their current local optimum) can somewhat obviate this issue [11], but a final challenge remains in addressing the original intent of convex optimization for enhanced 5G QoS via the convex optimization problem of hyperparameter reduction/tuning — effectuating an RCR adversarial training mechanism aboard the utilized MSY3I. Perhaps, the irony can best described as follows. Producing the tightest possible relaxation for the neural network layers of the MSY3I turns out to be no easier than the original problem of providing the tightest possible relaxation for solving 5G QoS convex optimization problems.

The remainder of this paper is organized as follows. Section II provides background and presents related work. Section III discusses three numerical challenges: effectuating an RCR adversarial training mechanism via MSY3I; reducing the computational cost via fire layers; and utilizing adaptive inertia weighting to operationalize PSO (to mitigate against the premature stagnation of particle velocities when implementing PSO onto the MSY3I). Common solution set mechanics are described. Section IV articulates the experimentation findings. Section V provides concluding thoughts and outlines future work.

II. Background and Related Works

Obtaining the globally optimal solution to an MINLP problem requires exploring a vast search space. This can be done through robust mixed-integer convex relaxations of the MINLP. Underscoring the robustness aspect, the Institute for Operations Research at ETH Zurich phrases it quite nicely: "… it is necessary to identify those key combinatorial substructures, induced by integral variables, which can be leveraged so as to improve the involved bound tightening and global optimization algorithms" [12].

A. Traversing the Search Space

1) Stochastic Search

Stochastic search approaches are essentially general-purpose problem-agnostic algorithms that can utilize qualitative or quantitative (computational) modules tailored to the considered problem and/or are combined with problem-specific algorithms. Most referenced algorithms reside within the swarm intelligence subfield of AI. They include, among others, genetic, differential evolution, colony optimization, and PSO algorithms. These algorithms share the commonality that several search entities are created and individually utilized in hyper-locale optimization actions while contemporaneously liaising with each other to derive a globally optimal solution. On the one hand, the challenge in utilizing these algorithms resides in the fact that if the chosen swarm size is too small, the algorithm will more likely gravitate to a local minimum without ascertaining a globally optimal solution; on the other hand, if the chosen swarm size is too large, the likelihood of ascertaining a viable globally optimal solution increases, but the computational overhead increases as well.

While the methods encompassing genetic and evolutionary algorithms cannot prove optimality of the solution, PSO applies the dual approach of global exploration and local search methods to ascertain an optimum solution. PSO is a meta-heuristic algorithm, i.e., no guarantee that a globally optimal solution can be found for some classes of problems. However, even relatively small swarm sizes are fairly consistent in providing "good enough" near-optimum solutions in relatively few iterations [13]. Hence, PSO is often utilized to solve MILP and MINLP problems.

2) Implementation of PSO Search

Fundamentally, a PSO approach simulates a set of particles or candidate solutions that traverse the search space. The method for PSO initializes the swarm at a random point within the space. Each particle has an assigned position and a velocity. The objective function is evaluated for each particle, and a global optimum G is ascertained. Iteratively, the position and velocity for each particle progress towards its individual best,

represented by the vector I, as well as the global best, represented by the vector G, as shown in Equations 1 and 2, respectively:

$$x_i^{(k+1)} = x_i^{(k)} + v_i^{(k+1)} \quad (1)$$

$$v_i^{(k+1)} = \iota^{(k)} v_i^{(k)} + \alpha_1[\beta_{1,i}(I_i - x_i^{(k)})] + \alpha_2[\beta_{2,i}(G - x_i^{(k)})] \quad (2)$$

where $x_i^{(k)}$ and $v_i^{(k)}$ denote, respectively, the position and velocity of particle i at generation [time step] k with particle inertia $\iota^{(k)}$, which induces a certain momentum with regards to the involved particles; the parameters $\beta_{1,i}$ and $\beta_{2,i}$ are uniformly distributed random variables over [0,1], $\alpha_1$and α_2 are acceleration constants, and I_i denotes particle i's optimum; the solution to (1) and (2) gives the global optimum [14].

An initial challenge arises when integrating PSO search into the DC-YOLO-GAN implementation, as most of the parameters are continuous or discontinuous, and they need to be transformed to discrete values (e.g., integers). To maximally preserve the original semantics, each attribute of a PSO particle is a distribution over its possible values rather than a specific value [9]. In addition to inertia $\iota^{(k)}$, the (cognitive component) vector I represents the individual best position in the search space that the involved particle has seen, and the (social component) vector G represents the best position in the search space that any particle in the swarm has seen; these three parameters — inertia, cognitive component, and social component — dictate a particle's behavior [9]. Furthermore, the updated velocity $v_i^{(k)}$ is added to the particle's position $x_i^{(k)}$, thereby moving the particle through the search space. However, as the position $x_i^{(k)}$ represents a set of parameters, the rounding of the calculated velocities $v_i^{(k)}$to discrete integer values creates an artificial environment, wherein particles may stagnate prematurely (i.e., get trapped into local optima [15]) with a non-graceful degradation of the particle inertia $\iota^{(k)}$.

Certain techniques, such as increasing the inertia (e.g., weighting the distance from the particle's local optimum) allow the involved particles to progress past their current local optimum instead of stagnating prematurely; these techniques beget calculating varying inertial weights. The chosen platform for the experimentation herein is the GNU Octave platform. As a numerical computation platform, it is mostly compatible with comparable platforms, such as MATLAB™; as GNU Octave is released under a GNU GPLv3 license, the source code was modified for the experiments conducted herein, which resulted in a Modified GNU Octave (M-GNU-O) platform [16], that can better leverage certain accelerants to deal with the PSO adaptive inertial weighting issue (yet another convex optimization problem) [17] as well as the various convex relaxations discussed herein.

The selection of the PSO was predicated upon its performance robustness (good performance even under a small swarm size) and ability to converge in relatively few iterations. As an architectural construct, the discussed DCGAN instantiation exhibits robust performance for the case herein, while other constructs may degrade in performance with prolonged training. For the 5G/B5G/6G functions needed for QoS, the PyTorch Machine Learning library was utilized, and the PyTorch implementation of the neural network framework, YOLO v3, was utilized; accordingly, the specific DCGAN implementation was that of DC-YOLO-GAN.

B. Resolving Gradations of Mixed-integer Convex Relaxations to Facilitate Convex Optimizations

Hybridizing local and global optimization algorithms has become an accepted strategy for deriving valid bounds for near-optimal convex optimization solutions [18]. This can also be operationalized by denoting and resolving gradations of mixed-integer convex relaxations. Accordingly, the nonlinearities are typically replaced by convex under-estimators and concave over-estimators. The tightest convex under-estimator and the tightest concave over-estimator are referred to as the convex envelope and the concave envelope of a function, respectively. Prior findings indicate that RCR, which facilitates convex optimization-based methods, can be well addressed by a MSY3I combined with convex relaxation adversarial training.

1) Squeezed YOLO v3 Implementation

Darknet, an open-source neural network written in C and CUDA, is a CNN implementation framework for the widely YOLO [19]. YOLO v2 utilizes a variant of Darknet-19, which starts with a 19-layer neural network, supplemented with 11 additional layers, thereby yielding a 30-layer convolutional architecture for YOLO v2. As documented in the literature, the performance of YOLO v2 is sub-optimal for the need described herein [20]; hence, a variant of YOLO v3 was selected. YOLO v3 utilizes a variant of Darknet-53, which starts with a 53-layer network that is supplemented with 53 additional layers, thereby yielding a 106-layer convolutional architecture for YOLO v3. Compared with YOLO v2, the classification performance of YOLO v3 is greatly enhanced, but prior experimental findings show that the computational performance is slower. This should be axiomatic, as a search space approach for a 106-layer YOLO network, even with the constraining decision of only optimizing the number of neurons in the layers at ten test values each, would still necessitate the training of 10^{106} models. Hence, to decrease the number of parameters for the YOLO instantiation, the use of fire layers (of SqueezeDet) to optimize the network structure segues to a MSY3I. In essence, certain SFLs replace certain Conv layers, and the number of hyperparameters as well as the number of filters of the compression portion of the fire layers are reduced; prior research has indicated that the number of model parameters in MSY3I will be lower than that of just YOLO v3 with only the slightest degradation in performance [5, 6]

2) Convex Relaxation Adversarial Training

In addition to analyzing its performance, MSY3I must be examined for robustness, and this often relates to the performance of the layer-wise optimal convex relaxations implemented within the involved DCGAN (or MSY3I in this case) [21]. In essence, a certain convex relaxation is posited for the purpose of ascertaining an upper bound for a worst-case instability scenario. This is of critical importance, as

prototypical DCGANs exhibit non-graceful degradation in performance even at imperceptible perturbation levels, which results in numerical instability. Given the abundance of perturbations/variability in contemporary environs, a prototypical approach for mitigating numerical instability, such as batch normalization (batchnorm), can have counterproductive consequences if not implemented in a proven fashion. Batchnorm is a method for imbuing stability into a neural network via normalization of the input layer, such that each layer can learn a bit more independently of other layers. Simply applying batchnorm to all the layers of the neural network can result in oscillation and instability. Prior research has shown that this instability can be avoided by *selectively* applying batchnorm, e.g., only at the generator output layer and/or the discriminator input layer (the adversarial components of a DCGAN).

To verify the performance of the layer-wise convex relaxations implemented in MSY3I, a hybridized approach vector is utilized in our work: (1) exact (complete), and (2) relaxed (incomplete). Prototypical exact verifiers are predicated upon Mixed Integer Programming (MIP) (specifically, MINLP for the experimentation discussed herein), Branch-and-Bound (BnB), or Satisfiability Modulo Theories (SMT). By definition, these exact verifiers are not beset by false positives or false negatives, but they must contend with resolving NP-hard optimization problems, which in turn obviates their scalability. Prototypical relaxed verifiers are predicated upon MILP or Mixed-Integer Convex Programming (MICP), which is more compact than MILP. MILP/MICP can be more quickly resolved and are more scalable, but their effectiveness (i.e., false negative rate) degrades quickly [22], thereby potentially obviating the ability to verify robustness. Hence, it can be seen that there are two aspects of relaxation: (1) convex relaxations implemented at each layer of the MSY3I, and (2) the relaxation schema verifier implemented to ascertain robustness of the MSY3I both layer-wise and overall [23]. These are the key elements of the RCR framework, which has a counterpoised objective of the tightest possible relaxation.

3) Facilitated RCR

Ultimately, the final rendition of the MSY3I is dictated by the PSO deployment; the PSO determines the reduction in the number of hyperparameters and the tuning thereof for the MSY3I. In turn, the M-GNU-O facilitates the adaptive inertial weighting to facilitate more robust PSO performance by addressing the premature particle velocity stagnation issue.

III. Numerical Challenges and Solutions

To the extent that RCR is a key enabler of computationally tractable 5G/B5G/6G solutions, its underlying architecture is central to the equation. While certain mathematical approaches hold great promise, their implementation may present a barrier due to various numerical issues. Our RCR "architectural stack," shown in Figure 1, tackles three successive challenges: (1) effectuating an RCR paradigm via MSY3I, (2) reducing the computational costs via PSO-tuned MSY3I, and (3) utilizing adaptive inertial weighting via M-GNU-O to operationalize the PSO. M-GNU-O serves as a key enabler for facilitating (3), which in turn facilitates (2), and in turn enables the bespoke MSY3I to effectuate an RCR paradigm.

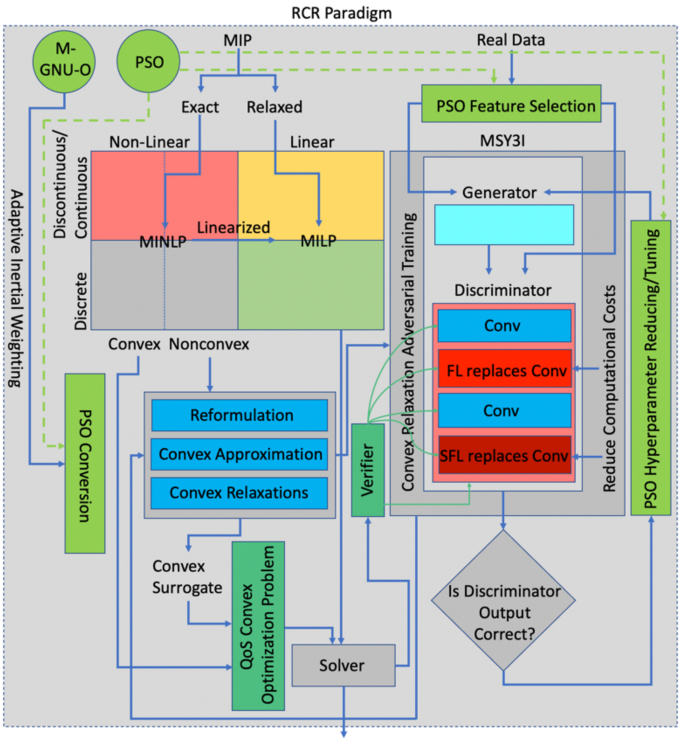

Fig. 1. RCR architectural stack and components

As noted in Section II-A-1, initialization of the swarm size in PSO is critical, as it impacts how robustly and quickly the involved optimization algorithm converges to a globally optimal solution. Furthermore, a nonoptimal initialization schema can segue to unstable gradients, which may have a profound impact on the stability of the involved optimization algorithms. For example, parameter updates that are excessively large (i.e., exploding gradient) or excessively small (i.e., vanishing gradient) may obviate the intended MSY3I deep learning.

Along this vein, the choice of the YOLO version to be utilized in deep learning is nontrivial. The reviews of recent versions of YOLO (e.g., v5) or even other variants (e.g., PP-YOLO) are still forthcoming, and peer-reviewed publications are not yet available. Our selection criteria narrowed to the YOLO version that could interoperate with the more robust set of 5G/BFG/6G-related tools. Given the performance disadvantages of YOLO v1 and v2, they were ruled out. YOLO v4 is a Darknet implementation, and YOLO v3 and v5 are PyTorch implementations; as the PyTorch library has a repertoire of 5G/BFG/6G-related tools, the choice was narrowed to YOLO v3 and v5. Because YOLO v5 has not been peer-reviewed yet, the decision was made to proceed with YOLO v3.

In addition to the CNN framework, the selection and utilization of various functions from the available ML libraries/toolkits are also important. By way of background information, Facebook operates two well-known open-source

ML libraries/toolkits: PyTorch and the Convolutional Architecture for Fast Feature Embedding (Caffe2). In March 2018, the Caffe2 repository was merged into the PyTorch repository on Github. Maintainers, core developers, and users noted several incompatibility issues (although Open Neural network Exchange or ONNX is intended to help resolve that) Typically, the onus is on 5G/BFG/6G researchers/programmers to understand and address the intricacies of the underlying numerical implementation. In this case, the numerical stability implementation challenge was nearly on par with the devising of the numerical stability strategy itself.

IV. Experimentation Findings

For the experiments described herein, we utilized two different RCR paradigms with different versions of components at the MSY3I level (MSY3I #1 and MSY3I #2), augmented with a TensorFlow-based DCGAN implementation, which is considered stable. MSY3I #1 was targeted for solving QoS convex optimization problems. As such, it required a high degree of numerical stability; accordingly, PyTorch v0.4.1 was utilized. MSY3I #2 was intended for solving 5G/BFG/6G-related functions (e.g., STFT), with lower utilization rate. Accordingly, the recently released PyTorch v1.7.0 was utilized, allowing MSY3I #2 to focus on its intrinsic stability training, so as to mitigate against the numerical instability issues from PyTorch v1.7.0 (as contrasted to v0.4.1). A "forward stable" TensorFlow-based DCGAN implementation (hereinafter, DCGAN #3) was utilized via an additional generator (hence, a mixture of generators) to assist in mitigating mode failure (a.k.a. mode collapse), which occurs when two competing neural networks that are being trained concurrently fail to converge or have an unusual convergence. Note that a forward stable DCGAN does not amplify perturbations of the input set, e.g., due to noise. The experimental testbed with the described components is delineated in Figure 2.

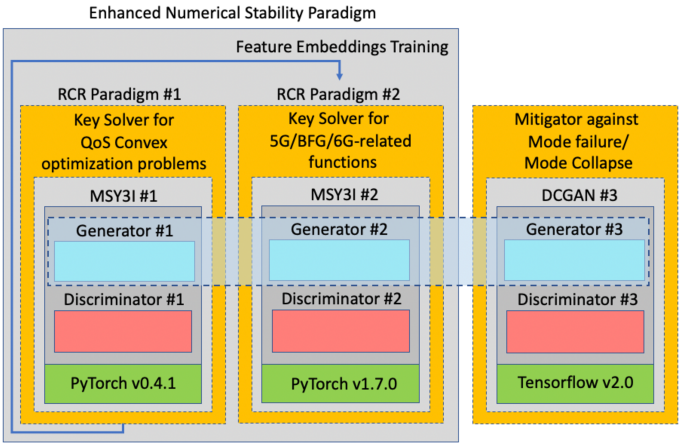

Fig. 2. Experimentation with a stable RCR, composed of two MSY3I implementation that are augmented with a third DCGAN.

The selection and utilization of various functions from the available ML libraries/toolkit is crucial. It is equally important for the 5G/BFG/6G researcher/programmer to understand and contend with the implementation intricacies (e.g., signature, dependency, etc.) of the numerical algorithms being utilized. For example, signature consistency intricacies have been shown to result in errors or incorrect results. Likewise, dependency intricacies are also an issue, as they introduce variances that result in errors or incorrect outcomes [26].

A. STFT Signature Consistency Challenges

Short-time Fourier transform (STFT) is a key functionality in many OFDM-based wireless systems and is often used as the basis for signal detection and classification in 5G and beyond. Previous PyTorch implementations of STFT (including the spectrogram transform) had slower performance than Librosa, a well-known Python package for signal processing and analysis [24]. Accordingly, the developers changed the function signature to be consistent with Librosa at v0.4.1 (various PyTorch versions are available at https://pypi.org/project/torch/0.4.1/#history). The significance is that the STFT signature for PyTorch versions prior to v0.4.1 can cause errors or return incorrect results [25].

B. STFT Dependency Challenges

In SciPy et al., it was noted that the various implementations of STFT in TensorFlow often introduce a phase skew dependency on the stored window which, if not addressed during the conversion, would have severe effects on any ensuing processing or phase analysis [26]. Hence, when phase information is processed, it is crucial to be aware of the phase conventions by which the STFT is being computed and adjust the processing schema accordingly. For example, conversion between conventions typically equates to point-wise multiplication of the STFT with an a priori determined matrix of phase factors.

The previously discussed PyTorch STFT issue (#9308 fixes #7883 by changing STFT to have a consistent signature with Librosa [27]) is emblematic of the numerical algorithm implementation challenges involved. First, the substantive portion of numerical algorithm/numerical analysis problems cannot be solved precisely; they need to be solved in an approximate fashion, and this is achieved by supplanting the infinite object with a finite approximation, as simplistically exemplified in the following Taylor-series polynomial approximation of the exponential function and a composite trapezoidal approximation of a definite integral, respectively:

$$e^x \approx 1 + x + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!} \quad (3)$$

$$\int_a^b f(x)dx \approx \frac{h}{2}\left[f(a) + 2\sum_{i=1}^{n-1} f(x_i) + f(b)\right] \quad (4)$$

The approximation errors are, in essence, truncation errors. Second, errors also stem from the limitations (i.e., round-off errors) in the representation of real numbers within the involved computing platform; these include not only the case of irrational numbers (which require an infinite number of digits for their exact representation), but also for various cases involving rational numbers (which are often represented as floating-point numbers). The accuracy of the floating-point representation is underpinned by the number of significant digits utilized;

axiomatically, a higher number of significant digits equates to a higher computational load. Third, extremely large or small numbers cannot readily be represented in floating-point arithmetic due to the phenomenon of overflow (an arithmetic operation yields a resultant, which is outside the range of the computational platform's floating-point numbers) and underflow (an arithmetic operation yields a resultant nonzero fraction, which is not readily able to be represented as a nonzero floating-point number).

Along this vein, the implementation of STFT in some of the software libraries/toolkits (e.g., TensorFlow, among others) does not follow either the frequency-invariant STFT convention or the time-invariant STFT, wherein the time resolution and frequency resolution of the STFT is the same across the time-frequency plane, such as shown below for the latter case,

$$STFT_g^{ti}(s)[m,n] = \sum_{l=-\lfloor \frac{Lg}{2} \rfloor}^{\lceil \frac{Lg}{2} \rceil - 1} s[l+na]g[l]e^{-\frac{2\pi iml}{M}} + b = \gamma \quad (5)$$

where L denotes the time separation between the short-time segments, and signal s is being filtered with STFT window g. Unconventionally, the window g is stored as vector of length Lg (typically, $Lg \ll L$), and the peak is not at $g[0]$, as expected, but at $g[\lfloor Lg/2 \rfloor]$ with the Simplified Time-Invariant STFT calculated, as shown in the following equation:

$$STFT_g^{sti}(s)[m,n] = \sum_{l=0}^{Lg-1} s[l+na]g[l]e^{-2\pi iml/M} + b = \gamma \quad (6)$$

Comparing (5) and (6), it can be discerned that (4) imbues a delay as well as a phase skew that is dependent on the (stored) window length L_g [26]. Researchers have also noted that the implementation of STFT in some of the software libraries/toolkits does not consider s circularly, but only for $n \in [0, \ldots, \lfloor (L - L_g)/a \rfloor]$ [26]. The documentation for the utilized functions (e.g., phased=gabphasederiv[dflag,method,…]) for use on the modified [experimentation] GNU Octave platform described herein (i.e., M-GNU-O) notes that "phased is scaled such that (possibly non-integer) distances are measured in samples … the computation of phased is inaccurate when the absolute value of the Gabor coefficients [of the Gabor transform, which is a special case of STFT] is low. This is due to the fact [that] the phase of complex numbers close to the machine precision is almost random" [27].

For the aforementioned reasons, our experimental architecture utilizes specific versions of PyTorch and TensorFlow for specific purposes. As was depicted in Figure 2, because numerical stability is needed for RCR Paradigm #1, PyTorch v0.4.1 was utilized for MSY3I #1. Whereas consistency and accuracy was needed for certain key functions (e.g., STFT) of RCR Paradigm #2, PyTorch v1.7.0 was utilized although the overall numerical stability decreased. To compensate/mitigate against this deficiency, we added DCGAN #3 to prevent mode failure/mode collapse. Collectively, this framework provides the basis for the 5G/B5G/6G testing amidst function/method issues that have arisen within ML libraries/toolkits. A core set was examined, which include FFT, IFFT, Real-Valued FFT (RFFT), Inverse RFFT (IRFFT), STFT, and Inverse STFT (ISTFT). A sampling of the issues/bugs encountered in various libraries/toolkits/frameworks (e.g., Caffe, Caffe2, Julia, PyTorch, SciPy, and TensorFlow) is shown in Figure 3.

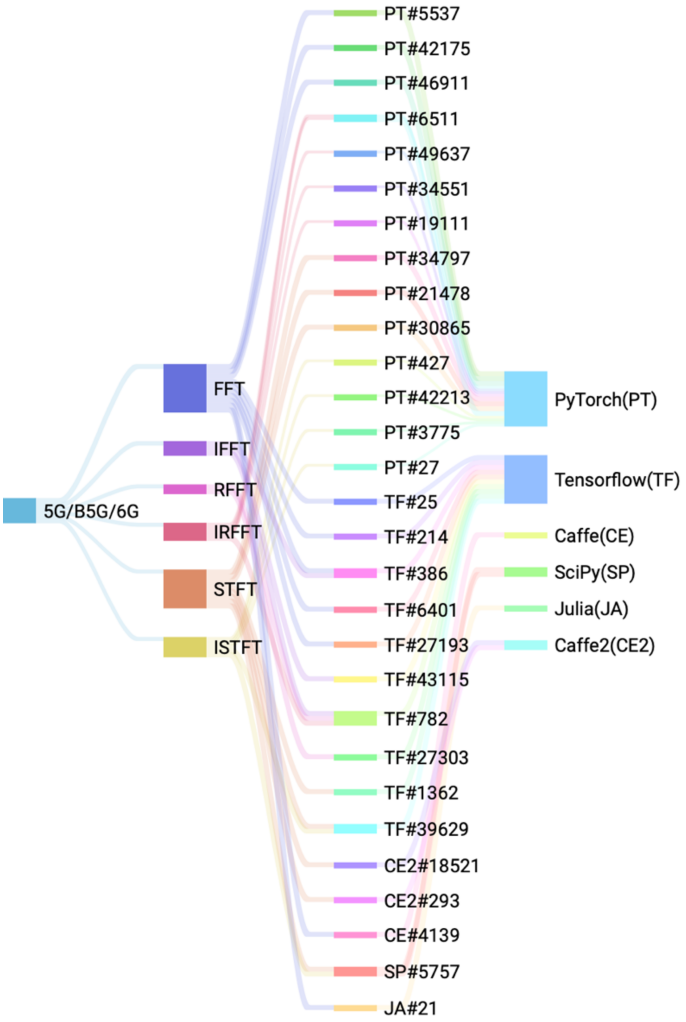

Fig. 3. Sample of numerical issues found in various ML libraries/toolkits.

Our experimentation period spans 4/24/18 through 12/10/20, which aligns with the release of PyTorch v0.4.0 and the release of PyTorch v1.7.1, respectively. During this period of time, the 5G/B5G/6G-related research that necessitated the functions/methods of FFT, IFFT, RFFT, IRFFT, STFT, and ISTFT triggered the modification ability of M-GNU-O as well as an architectural testbed that would address the aforementioned numerical issues.

C. *Conversion to Semi-Definite Programming Problem*

Ironically, although our original intent is to resolve QoS-related convex optimization problems, the process involves formulating successive gradations of convex optimizations, with varied approaches to resolve each class of convex optimizations. For example, the requisite adaptive inertial weighting (used to facilitate PSO, which in turn would facilitate

MSY3I to process yet other convex optimizations) is itself comprised of a succession of convex optimizations problems.

Indeed, many of the intermediate enabling steps involve Quadratic Programming (QP) step-down algorithms, including Quadratically Constrained Quadratic Programming (QCQP), which would compute the QCQP special class convex optimization problem in polynomial time. A QCQP takes the following general form:

$$\underset{x}{\text{minimize}} \left((\tfrac{1}{2}) x^T P_0 x + q_0^T x + r_0 \right)$$

$$\text{subject to} \quad (\tfrac{1}{2}) x^T P_i x + q_i^T x + r_i \leq 0, \; i = 1, \ldots, m$$

$$Ax = b$$

where $P_0, \ldots, P_m$ are n-by-n matrices; $P_i \in \mathbf{S}_+^n$ and $x \in \mathbf{S}^n$ is the optimization variable (7)

Accordingly, there are two envelopes for this aspect of discerning a gradation among convex optimization problems: (1) if $P_1, \ldots, P_m \in \mathbf{S}_{++}^n$, where $\mathbf{S}_{++}^n$ denotes the set of positive semidefinite matrices, the involved problem is convex, and (2) a QP with a semi-definite Hessian is still convex.

Traditionally, computing the Hessian matrix for large-scale problems is computationally impractical [28]. However, given a particular Hessian matrix in a resolvable form, proxies (i.e., approximations) of the Hessian matrix can be obtained in alternative ways, e.g., Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm. However, to avoid false curvature information, additional initialization conditions are required. To accommodate this, resolving of the QCQP can assist in the determination of the involved *trust regions* (the subset of the objective function region that is approximated). Furthermore, there are a variety of trust-region methods given a sufficiently low-rank positive semidefinite matrix; in essence, this equates to computing $\hat{R}_1 = \hat{R}_c + \hat{R}_n$, where $\hat{R}_c$ and $\hat{R}_n$ are examined, via resolving the following Rank Minimization Problem (RMP) :

$$(\hat{R}_c, \hat{R}_n) = \arg \min_{R_c, R_n} \text{rank}\,(\hat{R}_c), \tag{8}$$

$$\text{subject to} \begin{cases} R_c + R_n = R_s \\ R_c \succeq 0 \\ R_n \text{ diagonal} \end{cases}$$

In many instances, even when the input set is designed/architected to be convex, the resultant output set may turn out to be nonconvex. Accordingly, when the rank function is nonconvex and discontinuous, the RMP cannot be solved directly. To accommodate this, and to transform the problem to a convex form, the rank function is replaced with the trace function, giving rise to a Trace Minimization Problem (TMP):

$$(\hat{R}_c, \hat{R}_n) = \arg \min_{R_c, R_n} \text{tr}\,(\hat{R}_c), \tag{9}$$

$$\text{subject to} \begin{cases} R_c + R_n = R_s \\ R_c \succeq 0 \\ R_n \text{ diagonal} \end{cases}$$

Because the rank function tallies the number of nonzero eigenvalues and the trace function computes the sum of the involved eigenvalues, the above equation can be reconstrued as an equivalent Semi-Definite Programming (SDP) problem:

$$(\hat{R}_c, \hat{R}_n) = \arg \min_{R_c, R_n} \text{tr}\,(\hat{R}_c), \tag{10}$$

$$\text{subject to} \begin{cases} \begin{bmatrix} W_1 & R_c \\ R_C^H & W_2 \end{bmatrix} \\ R_c + R_n = Rs \\ Rc \succeq 0 \\ R_n \text{ is diagonal} \end{cases}$$

Once in this form (i.e., the nonconvex QCQP has been relaxed to a convex SDP), there are numerous SDP solvers (e.g., SDPT3, which is a MATLAB/GNU Octave Semi-Definite Programming or SDP software package) available for these types of problems; the aforementioned (regarding Equations 7, 8, 9, and 10) is implemented atop the M-GNU-O platform, which readily supports various high-performance SDP solvers [16]. Prior testing was effectuated in Ilog Cplex Optimizer (a commercial software package for optimization) and, subsequently, on AD Model Builder (ADMB) (an open-source software package for non-linear statistical modeling) as well as Interior Point OPTimizer (IPOPT) (a software package for large-scale nonlinear optimization).

V. Concluding Remarks and Future Work

This paper articulated some of the issues/intricacies of 5G/BFG/6G-related function implementations, of various ML frameworks, which affected their accurate resolution of QoS convex optimization problems for 5G/BFG/6G. For example, mathematical equivalence does not necessarily segue to correct results when using certain functions of particular versions of numerical computation libraries. By way of example, in some cases, sub-operations needed to be combined, as performing the sub-operations separately would be computationally slower and more numerically unstable (e.g., as the softmax output approaches 0, the log output approaches infinity, which causes instability).

To compound this issue, it was found that the optimization challenge of transforming nonconvex to convex optimization problems may spawn yet other nonconvex optimization problems, thereby highlighting the need/opportunity to utilize an RCR framework. This paper presented an RCR architecture (presented in Figures 1 and 2), which could not only resolve the tasked 5G QoS-related convex optimization problems but could also leverage the same RCR mechanisms for tuning its own hyperparameters; the RCR architectural stack achieved this via three distinct phases: (1) effectuating a RCR paradigm, via a bespoke MSY3I, (2) using a PSO to tune the MSY3I so as to reduce the associated computational costs, and (3)

operationalizing the PSO via an adaptive inertial weighting mechanism facilitated by an M-GNU-O. For future work, an additional DCGAN will be added to the RCR architectural stack to derive further key combinatorials for optimizing computations.

ACKNOWLEDGMENTS

The authors would like to thank Vit Tall and the University of Arizona for the collaborative framework pertaining to this 5G/B5G/6G-related white paper series. The authors would also like to acknowledge various organizations, such as ED2, for their encouragement related to advancing matters within the 5G/B5G/6G ecosystem.

REFERENCES

[1] K. Khamaru and M. Wainwright, "Convergence guarantees for a class of non-convex and non-smooth optimization problems," *Journal of Machine Learning Research*, vol. 20, pp. 1-52, 2019.

[2] H. Shin, J. Lee, Ja. Kim, Ji. Kim, "Continual Learning with Deep Generative Replay," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[3] Y. Chen, Y. Shi, and B. Zhang, "Optimal control via neural network: a convex approach," *Seventh International Conference on Learning Representations*, 2019, https://openreview.net/forum?id=H1MW72AcK7

[4] N. Nguyen, T. Do, T. Ngo, and D. Le, "An evaluation of deep learning methods for small object detection," *Journal of Electrical and Computer Engineering*, vol. 2020, Apr 2020, doi: 10.1155/2020/3189691.

[5] A. Wang, M. Wang, K. Jiang, M. Cao, and Y. Iwahori, "A dual architecture combined SqueezeNet with OctConv for lidar data classification," *Sensors*, vol. 19, Nov 2019, doi: 10.3390/s19224927.

[6] B. Wu, A. Wan, F. Iandola, P. H. Jin and K. Keutzer, "SqueezeDet: unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving," *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 446-454, doi: 10.1109/CVPRW.2017.60.

[7] R. Peiming, L. Wang, W. Fang, S. Song, S. Djahel, "A novel squeeze YOLO-based real-time people counting approach," *Int. J. of Bio-Inspired Computation*, vol. 16, pp. 94-101, Sep 2020, doi: 10.1504/IJBIC.2020.109674.

[8] F. Yei, "Particle swarm optimization-based automatic parameter selection for deep neural networks and its applications in large-scale and high-dimensional data," *PLOS One*, Dec 2017, doi: 10.1371/journal.pone.0188746.

[9] S. Strasser, R. Goodman, J. Sheppard, S. Butcher, "A new discrete particle swarm optimization algorithm," *Proc of the Genetic and Evolutionary Computation Conference*, pp. 53-60, doi: 10.1145/2908812.2908935.

[10] X. Liu, Q. Wang, H. Liu, and L. Li, "Particle swarm optimization with dynamic inertia weight and mutation," *2009 Third International Conference on Genetic and Evolutionary Computing*, Feb 2010, pp. 620–623, doi: 10.1109/WGEC.2009.99.

[11] B. Borowska, "Dynamic inertia weight in particle swarm optimization," *Advances in Intelligent Systems and Computing II*, Nov 2017, pp. 79-88, Springer, Cham, doi: 10.1007/978-3-319-70581-1_6.

[12] "Mixed integer optimization," Accessed on: Jan. 7, 2021. [Online]. Available:https://math.ethz.ch/ifor/research/mixed-integer-optimization.html

[13] Y. Sun and Y. Gao, "An efficient modified particle swarm optimization algorithm for solving mixed-integer nonlinear programming problems," *International Journal of Computational Intelligence Systems*, vol. 12, Apr 2019, pp. 530-543, doi: 10.2991/ijcis.d.190402.001.

[14] M. Pourabdollah, E. Silvas, N. Murgovski, M. Steinbuch, B. Egardt, "Optimal sizing of a series PHEV: comparison between convex optimization and particle swarm optimization," *4th International Federation of Automatic Control Workshop on Engine and Powertrain Control, Simulation and Modeling*, 2015, pp. 16-22, doi: 10.1016/j.ifacol.2015.10.003.

[15] C. Worasucheep, "A particle swarm optimization with stagnation detection and dispersion," *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, Hong Kong, June 2008, pp. 424-429, doi: 10.1109/CEC.2008.4630832.

[16] S. Chan, "Mitigation Factors for Multi-domain Resilient Network Distributed Tessellation Communications," *The Fifth International Conference on Cyber Technologies and Cyber-Systems*, 2020, pp. 66-73, http://www.thinkmind.org/articles/cyber_2020_2_60_80061.pdf.

[17] M. Jakubcova, P. Maca, and P. Pech, "A Comparison of Selected Modifications of the Particle Swarm Optimization Algorithm," *Journal of Applied Mathematics*, vol. 2014, June 2014, doi: 10.1155/2014/293087.

[18] R. Pellgrini, A. Serani, G. Liuzzi, F. Rinaldi, S . Lucidi, and M. Diez, "Hybridization of Multi-Objective Deterministic Particle Swarm with Derivative-Free Local Searches," *Mathematics*, vol. 8, April 2020, doi: 10.3390/math8040546.

[19] J. Redmon, S. Divvala, R. Girshick, and A. Farhardi, "You only look once: unified, real-time object detection," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 779-788, doi: 10.1109H/CVPR.2016.91.

[20] "YOLO: Real-Time Object DetectionAccessed," Accessed on: Mar. 1, 2021. [Online]. Available: https://pjreddie.com/darknet/yolo/.

[21] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," *IEEE European Symposium on Security and Privacy,* 2016, pp. 372–387, doi: 10.1109/EuroSP.2016.36.

[22] B. Anderson, Z. Ma, J. Li, and S. Sojoudi, "Tightened convex relaxations for neural network robustness certification," *Proceedings of the 59th IEEE Conference on Decision and Control*, 2020, pp. 2190-2197, doi: 10.1109/CDC42340.2020.9303750.

[23] H. Salman, G. Yang, H. Zhang, C. Hsieh, and P. Zhang, "A convex relaxation barrier to tight robustness verification of neural networks," *Advances in Neural Information Processing Systems*, vol. 30, 2019.

[24] "Torch.stft," Accessed on: Mar. 1, 2021. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.stft.html.

[25] "Change stft to have consistent signature with librosa #9308," Accessed on: Mar. 1, 2021. [Online]. Available: https://github.com/pytorch/pytorch/pull/9308.

[26] A. Marafioti, N. Holighaus, N. Perraudin, P. Majdak, "Adversarial Generation of Time-Frequency Features," *Proceedings of Machine Learning Research*, vol. 97, 2019, pp. 4352-4362, http://proceedings.mlr.press/v97/marafioti19a/marafioti19a.pdf.

[27] "Function: gabphasederiv," Accessed on: Mar. 1, 2021. [Online]. Available:https://octave.sourceforge.io/ltfat/function/gabphasederiv.html

[28] J. Rafati and R. F. Marcia, "Improving L-BFGS Initialization for Trust-Region Methods in Deep Learning," *17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2018, pp. 501-508, doi: 10.1109/ICMLA.2018.00081.