Robust Solutions to Constrained Optimization Problems by LSTM Networks

Zheyu Chen*, Kin K. Leung†, Shiqiang Wang‡, Leandros Tassiulas§, Kevin Chan¶
*†Imperial College London, UK
‡IBM T.J. Watson Research Center, Yorktown Heights, NY, USA
§Yale University, New Haven, CT, USA
¶U.S. Army Research Lab, Adelphi, MD, USA
{*z.chen19, †kin.leung}@imperial.ac.uk, ‡wangshiq@us.ibm.com, §leandros.tassiulas@yale.edu, ¶kevin.s.chan.civ@mail.mil

***Abstract*—Many technical issues for communications and computer infrastructures, including resource sharing, network management and distributed analytics, can be formulated as optimization problems. Gradient-based iterative algorithms have been widely utilized to solve these problems. Much research focuses on improving the iteration convergence. However, when system parameters change, it requires a new solution from the iterative methods. Therefore, it is helpful to develop machine-learning solution frameworks that can quickly produce solutions over a range of system parameters.**

We propose here a learning approach to solve non-convex, constrained optimization problems. Two coupled Long Short Term Memory (LSTM) networks are used to find the optimal solution. The advantages of this new framework include: (1) near optimal solution for a given problem instance can be obtained in very few iterations (time steps) during the inference process, (2) the learning approach allows selections of various hyper-parameters to achieve desirable tradeoffs between the training time and the solution quality, and (3) the coupled-LSTM networks can be trained using system parameters with distributions different from those used during inference to generate solutions, thus enhancing the robustness of the learning technique. Numerical experiments using a dataset from Alibaba reveal that the relative discrepancy between the generated solution and the optimum is less than 1% and 0.1% after 2 and 12 iterations, respectively.

***Index Terms*—Constrained optimization, LSTM, optimization, SDC, stochastic optimization**

I. Introduction

Using learning techniques to solve optimization problems has attracted great attention in recent years. In [1] and [2], the supervised learning techniques are modified and enhanced to predict the optimal solutions for given optimization problem parameters. Because the ground truth labels are required for the supervised learning techniques, to train a well-performed prediction model for optimization needs sufficient problem parameters, and the optimal solutions of these optimization problems for the training process, which requires extra effort to generate. Besides the supervised learning, reinforcement learning techniques and meta-learning techniques are also used for solving optimization problems. For instance, authors in [3] propose to model the optimization solving process as a Markov Decision process and employ a deep neural network as the decision policy, while authors in [6] propose to embed the optimization problem into a meta-learning problem and employ a Long Short-Term Memory (LSTM) network as the optimizer.

Motivated by the success of these prior works, researchers start to focus on more specific types of optimization problems. For instance, authors of [4] propose a framework, named TwinL2O, consisting two LSTMs for solving minimax optimization problems. Different from the Twin-L2O, we propose here to use the two Coupled LSTM networks, referred to as *CLSTMs*, for solving non-convex, constrained optimization problems with user-defined objective and constraint functions. Since the CLSTMs and the Twin-L2O are derived for solving different types of problems, different considerations are required in the design of the loss functions and the overall workflows, although the proposed CLSTMs and the Twin-L2O both employ two LSTM networks. Instead of solving the minimax optimization problems as by the Twin-L2O, the constrained optimization problem we consider here is given as follows:

$$\begin{aligned} &\min_{\theta} \ f(\theta) \\ &\text{s.t. } h(\theta) \leq 0. \end{aligned} \tag{1}$$

By introducing a Lagrange multiplier λ, a Lagrange function can be formed for the optimization problem in (1):

$$J(\theta, \lambda) = f(\theta) + \lambda h(\theta). \tag{2}$$

The dual optimization problem of (1) is

$$\begin{aligned} &\max_{\lambda} \ J(\theta^*, \lambda) \\ &\text{s.t. } \theta^* = \text{argmin } J(\theta, \lambda), \lambda \geq 0 \end{aligned} \tag{3}$$

According to the duality theory, the dual optimization problem (3) has the same optimal solution for the original (primal) problem (1) under the condition that the duality gap

This research was partly sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

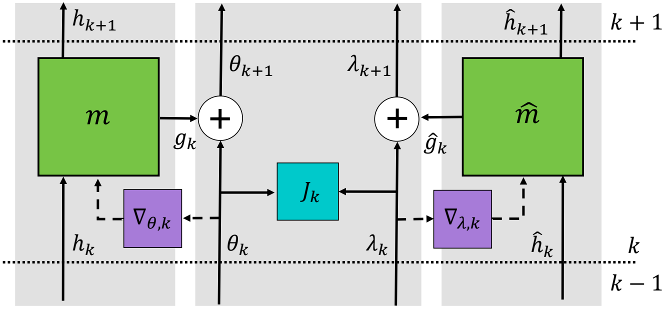

Fig. 1: Computation graph of the coupled LSTM for the iteration k, where $\nabla_{\theta,k} = \nabla_\theta J(\theta_k, \lambda_k), \nabla_{\lambda,k} = \nabla_\lambda J(\theta_{k+1}, \lambda_k)$.

is zero. Therefore, our objective is to find the optimal θ and λ for minimizing and maximizing the function J, respectively. Note that the dual optimization problem (3) can be regarded as a special minimax optimization problem. Thus, the proposed CLSTMs can also be applied to solve minimax optimization problems.

We also formulate here a resource-allocation problem in the cloud cluster as a constrained optimization problem and apply the proposed CLSTMs to solve it with practical data from Alibaba [7]. Our evaluation results demonstrate that the CLSTMs can achieve the 99% optimality accuracy after 2 iterations and the mean relative discrepancy between the generated solution and the optimum is less than 0.1% after 12 iterations. Furthermore, we explore and demonstrate the impact of various hyper-parameters in the CLSTMs on performance. Specifically, our numerical results show that selecting these hyper-parameters can serve as a mechanism to achieve desirable tradeoffs between the training time and the solution quality. Finally, we conduct an experiment to validate and show robustness where the CLSTMs can be trained using system parameters with distributions different from those used during inference to generate solutions.

The rest of the paper is organized as follows. Section II presents the detail of the coupled LSTM networks. Section III describes the resource-allocation problem under study and the formulated constrained optimization problem. Section IV presents the evaluation of the coupled LSTM networks using the cluster trace [7]. Finally, section V discusses related research and section VI concludes the paper.

II. Proposed CLSTMs

In this section, we propose the CLSTMs for solving constrained optimization problems.

Firstly, we describe the inference process that the CLSTMs are utilized to find the optimal θ and λ for a given Lagrange function J, respectively, by iterations. The overall workflow is shown in Fig. 1 where iterations indexed by k progress from the bottom upward. In each iteration k, the update step sizes of θ and λ are denoted by g_k and $\hat{g}_k$, respectively. Specifically, θ

Algorithm 1: Training Process of the CLSTMs in a frame

1: **for** iteration $k = (i-1)K, (i-1)K+1, \ldots, iK$ **do**
2: **for** $(J(\theta,\lambda), \theta, \lambda)$ in the training data set **do**
3: Calculate the gradient of function J w.r.t. θ: $\nabla_\theta J(\theta_k, \lambda_k)$;
4: Generate the update step size g_t by: $\begin{bmatrix} g_k \\ h_{k+1} \end{bmatrix} = m(\nabla_\theta J(\theta_k, \lambda_k), h_k, \phi_i)$
5: Update θ using (5);
6: Calculate the gradient of function J w.r.t. λ: $\nabla_\lambda J(\theta_{k+1}, \lambda_k)$;
7: Generate the update step size $\hat{g}_t$ by: $\begin{bmatrix} \hat{g}_k \\ \hat{h}_{k+1} \end{bmatrix} = \hat{m}(\nabla_\lambda J(\theta_{k+1}, \lambda_k), \hat{h}_k, \hat{\phi}_i)$
8: Update λ using (7);
9: **end for**
10: **end for**
11: Calculate the loss functions $L(\phi_i)$ and $\hat{L}(\hat{\phi}_i)$ using (8)(9), respectively;
12: Update the parameters ϕ_i and $\hat{\phi}_i$ using the gradients $\nabla_{\phi_i} L(\phi_i)$ and $\nabla_{\hat{\phi}_i} \hat{L}(\hat{\phi}_i)$, respectively;

Algorithm 2: Training Procedure

1: **for** epoch =1,2,... **do**
2: Randomly initialize the values of θ, λ for each optimization problem $J(\theta, \lambda)$;
3: Randomly initialize the hidden state $h_k, \hat{h}_k$ for m and $\hat{m}$, respectively;
4: **for** frame $i = 1, 2, \ldots, I$ **do**
5: Algorithm 1;
6: **end for**
7: **end for**

is updated from iteration k to $k+1$ by the following equations:

$$\begin{bmatrix} g_k \\ h_{k+1} \end{bmatrix} = m(\nabla_\theta J(\theta_k, \lambda_k), h_k, \phi^*), \tag{4}$$

$$\theta_{k+1} = \theta_k + g_k, \tag{5}$$

where ϕ^* denotes the optimal parameters in m, $\nabla_\theta J(\theta_k, \lambda_k)$ is the gradient of function J with respect to (w.r.t.) θ, and h_k, h_{k+1} are the hidden state for m in iteration k and $k+1$, respectively. Then λ is updated according to:

$$\begin{bmatrix} \hat{g}_k \\ \hat{h}_{k+1} \end{bmatrix} = \hat{m}(\nabla_\lambda J(\theta_{k+1}, \lambda_k), \hat{h}_k, \hat{\phi}^*), \tag{6}$$

$$\lambda_{k+1} = \lambda_k + \hat{g}_k, \tag{7}$$

where $\hat{\phi}^*$ denotes the optimal parameters in $\hat{m}$, $\nabla_\lambda J(\theta_{k+1}, \lambda_k)$ is the gradient of function J w.r.t. λ, and $\hat{h}_k, \hat{h}_{k+1}$ are the hidden state for $\hat{m}$ in iteration k and $k+1$, respectively.

We define K consecutive iterations are a frame. The training process is to update the parameters for m and $\hat{m}$ at the end of

each frame. Specifically, at the end of frame i, the parameters ϕ_i are updated to minimize the loss function:

$$L(\phi_i) = \mathbf{E}\big[\sum_{k=(i-1)K+1}^{iK+1} w_k J(\theta_k, \lambda_k)\big], \quad (8)$$

where w_k are weighting factors and the sum of all w_k equals 1. Meanwhile, we update the parameters $\hat{\phi}_i$ to minimize the loss function:

$$\hat{L}(\hat{\phi}_i) = -\mathbf{E}\big[\sum_{k=(i-1)K+1}^{iK} \hat{w}_k J(\theta_{k+1}, \lambda_k) + \hat{w}_{iK+1} J(\theta_{iK+1}, \lambda_{iK+1})\big], \quad (9)$$

where $\hat{w}_k$ are weighting factors and the sum of all $\hat{w}_k$ equals 1. The expectation is needed because that the objective functions $f(\theta)$ used for training are sampled from a set of random functions, while the constraint functions $h(\theta)$ are chosen from another set of random functions. That is, the form of the random functions is fixed, while function parameters are randomly chosen from some distributions.

The detailed training process in a frame is illustrated in Algorithm 1, which works as follows. At the beginning of a frame, all variables and Lagrange multipliers are updated in K iterations (Lines 1-10). In every iteration, for each sample $(J(\theta, \lambda), \theta, \lambda)$, the variables θ are updated (Lines 3-5) before the Lagrange multipliers λ are updated (Lines 6-8). Finally, at the end of a frame, the parameters ϕ_i and $\hat{\phi}_i$ are updated (Lines 11-12). Furthermore, to reduce the computational complexity of computing the gradients $\nabla_{\phi_i} L(\phi_i)$ and $\nabla_{\hat{\phi}_i} \hat{L}(\hat{\phi}_i)$, we assume that the gradients of J w.r.t. to θ and λ are independent of ϕ and $\hat{\phi}$, respectively (i.e., $\frac{\partial \nabla_\theta J}{\partial \phi_i} = 0$, $\frac{\partial \nabla_\lambda J}{\partial \hat{\phi}_i} = 0$).

A. Training Procedure

To obtain sufficient sampling and experience about how to optimize near and far from the optimum, a group of I consecutive frames are defined as an epoch, where the optimization variables (i.e., λ and θ) and the hidden states (i.e., h_k and $\hat{h}_k$) are randomly initialized at the beginning of each epoch. The detailed training procedure is provided in Algorithm 2.

B. Parameters Shared in the CLSTMs

For each LSTM in the CLSTMs, directly feeding the vector of gradients w.r.t. every variable into the fully connected input layer of the LSTM will require a rather large LSTM network if there are thousands of variables. Consequentially, two large LSTM networks will impose a tremendous burden on computation and storage. To reduce the computation and storage requirements, the coordinate-wise LSTM structure proposed in [6] is adopted here. Specifically, the gradients of function J w.r.t. every variable are fed into the LSTM successively so that they can share the parameters of an LSTM network. Thus, the number of LSTM parameters can keep small when there are thousands of variables. Meanwhile, the hidden states for each variable are independent so that the LSTM can generate different update step sizes for different variables even their gradients are equal.

C. Scale function

Note that there is a constraint on the Lagrange multiplier λ required to be non-negative in the dual optimization problem (3). Thus, the scale function $\psi(\lambda)$, as defined in (10), is used to ensure that the values of λ are larger than or equal to 0. Using the scale function $\psi(\lambda)$ has the following two advantages. Firstly, compared with the absolute value of λ or ReLU function $max(0, \lambda)$, the derivative of the scale function $\psi(\lambda)$ exists everywhere. Compared with the square function λ^2, the scale function $\psi(\lambda)$ remains the linear property expect the range $[-1, 1]$. These two advantages help the CLSTMs to converge and obtain better performance as confirmed in our experiments.

$$\psi(\lambda) = \begin{cases} -2\lambda - 1, & \text{if } \lambda < -1 \\ \lambda^2, & \text{if } -1 \leq \lambda \leq 1 \\ 2\lambda + 1, & \text{if } \lambda > 1 \end{cases} \quad (10)$$

III. Problem Formulation

A. System Model

The resource-allocation problem is to allocate cluster resources to competing tasks for maximizing the sum of task utilities. Specifically, there are N tasks competing for a type of resource and the amount of available resource is denoted by C. For each task n, let r_n, R_n, $u_n(r_n)$ denote the amount of resource allocated to it, its resource requirement and its utility function given the allocated resource r_n, respectively. Moreover, each task n must be allocated with a minimum amount of resource to provide good service, while it also cannot receive more than a maximum amount of resource in order to guard against occupying a large amount of resources by few tasks. By introducing two parameters $\alpha > 1$ and $\beta < 1$, the maximum and the minimum resource requirement of task n are denoted by αR_n and βR_n.

B. Optimization Problem

By using these notations, we can have the optimization problem derived from the resource allocation problem:

$$\max_{r_1, \ldots, r_N} \sum_{n=1}^{N} u_n(r_n) \quad (11a)$$

$$\text{s.t.} \sum_{n=1}^{N} r_n \leq C \quad (11b)$$

$$r_n \geq \beta R_n, \forall n \quad (11c)$$

$$r_n \leq \alpha R_n, \forall n. \quad (11d)$$

The first constraint (11b) ensures that the amount of allocated resources for all tasks must not exceed the amount of available resources, while the constraints (11c) guarantee that the minimum resource requirements for tasks are satisfied. The constraints (11d) ensure that the amount of allocated resources for each task must not exceed its maximum resource requirement.

C. Solve the Problem with the CLSTMs

Let $\boldsymbol{\theta}$ denotes the vector of variables $[r_1, \ldots, r_N]$ and define the objective function $f(\boldsymbol{\theta})$ and the constraint function $\boldsymbol{h}(\boldsymbol{\theta})$ as:

$$f(\boldsymbol{\theta}) = -\sum_{n=1}^{N} u_n(r_n)$$

$$\boldsymbol{h}(\boldsymbol{\theta}) = \begin{bmatrix} \sum_{n=1}^{N} r_n - C \\ \beta R_1 - r_1 \\ \cdots \\ \beta R_N - r_N \\ r_1 - \alpha R_1 \\ \cdots \\ r_n - \alpha R_N \end{bmatrix} \tag{12}$$

Then, we can have an optimization problem whose form is similar to (1):

$$\begin{aligned} &\min_{\boldsymbol{\theta}} \ f(\boldsymbol{\theta}) \\ &\text{s.t. } \boldsymbol{h}(\boldsymbol{\theta}) \leq 0. \end{aligned} \tag{13}$$

The Lagrange function can be derived by introducing a Lagrange multiplier vector $\boldsymbol{\lambda} = [\lambda_0, \ldots, \lambda_{2N}]$:

$$J(\boldsymbol{\theta}, \boldsymbol{\lambda}) = f(\boldsymbol{\theta}) + \boldsymbol{\lambda}\boldsymbol{h}(\boldsymbol{\theta}) \tag{14}$$

Finally, by substituting $J(\boldsymbol{\theta}, \boldsymbol{\lambda})$ for $J(\theta, \lambda)$, $\boldsymbol{\theta}$ for θ, $\boldsymbol{\lambda}$ for λ in the Algorithm 2, we can apply Algorithm 2 to train the CLSTMs for solving this constrained optimization problem. It is worth noting that we have not made specific assumptions about the forms of the functions $f(\theta)$ and $h(\theta)$ in (13) as long as the problem (13) satisfies the zero-duality gap.

IV. Experiment

A. Setup

The Alibaba cluster trace [7] presents the resource utilization of 4,000 machines and the resource requirements of the batch workloads. In the Alibaba's cluster, the batch workloads are described by the 'Job-Task-Instance' structure, where each job has multiple tasks and each task contains multiple instances. Furthermore, the resource requirements of each instance in a given task are identical. In our experiments, we allocate the available CPU in the cluster in terms of utilization in percentage to tasks where the resource requirement of a task is the aggregate resource requirement of all its instances. We employ a cluster of 10 machines randomly selected from the Alibaba cluster trace to provide CPU resource to competing tasks in all optimization problem scenarios considered in the following experiments. In addition, for each problem scenario, we randomly select 10 tasks and each task is allowed to have most 100 instances. For each task n, its utility function given the allocated CPU utilization r_n is given by

$$u_n(r_n) = \frac{1}{1 + e^{-\mu_n (r_n - R_n)}}, \tag{15}$$

where μ_n is a constant randomly selected from the uniform distribution in the range of $[0.5, 1)$ and R_n is the CPU utilization requirement of task n. Moreover, α and β are set to 1.4 and 0.7, respectively, for all tasks.

In the experiments, our algorithm is implemented with Python and Tensorflow 2.1 and evaluated on an Ubuntu 20.04 LTS server with a NVIDIA TITAN Xp graphics card. Each LSTM of the CLSTMs has two layers and each layer has 20 neural units. During the training process, the CLSTMs are trained with 10,240 optimization problem scenarios. The training process consists of 30 epochs where each epoch has 50 frames ($I = 50$) and each frame consists of 10 iterations ($K = 10$). We set $w_{k,\forall k}$ to 1 and the learning rate in the frame i to $0.01 \times 0.97^{\frac{i-1}{300}}$. For the evaluation, the trained CLSTMs are used to solve 1,000 optimization problem scenarios with randomly selected parameters. For each problem scenario, the optimization (control) variables are updated using the trained CLSTMs by iterations and the solutions are saved after 1,000 iterations.

We employ a gradient-based method to produce the optimal solutions, which serve as a baseline for comparison with the CLSTMs. Given the optimization problem (3), this method updates the variables θ and λ by iterations. In each iteration, this method first finds the optimal θ for the given λ through the gradient-descent and then updates the λ with the found optimal θ by gradient ascent. After the iterations converge, the optimal solutions are saved.

To measure performance of the proposed CLTMs approach, we define the *relative accuracy* of a solution as:

$$\alpha = 1 - \frac{|\hat{x} - x|}{|x|}, \tag{16}$$

where $\hat{x}$ and x are the optimal values of the objective function found by the CLSTMs and the gradient-based method, respectively. Moreover, we define *Mean relative accuracy* as the relative accuracy averaged over the 1,000 problem scenarios solved by the trained CLTMs in the evaluation (inference) process.

B. Results and Analysis

1) *Compare with the baseline*: In this experiment, we demonstrate that the CLSTMs can find the near-optimal solutions in a few iterations and obtain extremely high relative accuracy in the end.

Fig. 2a shows the mean relative accuracy obtained by the CLSTMs and the baseline in the first 200 iterations during the evaluation process. Note that the mean relative accuracy after the first iteration is omitted in Fig. 2a so that the difference between the two curves can be made clear. Although not shown in the figure, the mean relative accuracy for the CLSTMs and the baseline after the first iteration is in fact 0.972 and 0.729, respectively. We can further observe that the mean relative accuracy for the CLSTMs achieves 0.9929 after 2 iterations and reaches to 0.9993 after 12 iterations, while the baseline still presents obvious fluctuation. This confirms that the CLSTMs is much quicker in producing accurate and stable solutions than the conventional gradient-based method. Furthermore, we can see that the mean relative accuracy

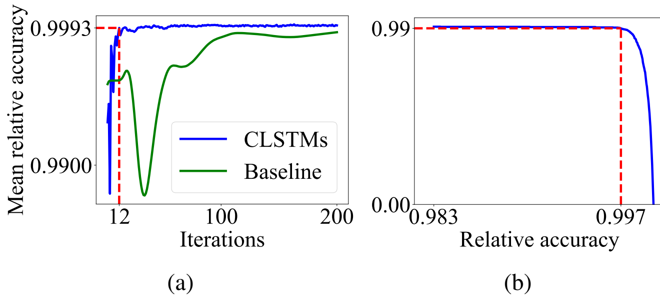

Fig. 2: (a) The mean relative accuracy over iterations and (b) the complementary cumulative distribution function (CCDF) of relative accuracy.

achieves 0.9995 at the end of 1,000 iterations. Although it is possible to improve the solution quality (i.e., from 0.9993 to 0.9995) with more iterations in the evaluation process, the improvement is quite marginal.

Fig. 2b shows the complementary cumulative distribution function (CCDF) of relative accuracy of solutions generated by the CLSTMs after 1,000 iterations. From the figure, we can make the observation that the 99% relative accuracy is larger than 0.997 and the minimal relative accuracy is 0.983.

Clearly, these numerical results validate that the CLSTMs can find a near-optimal solution quickly (e.g., achieving 0.993 and 0.9993 for the mean relative accuracy after 2 and 12 iterations, respectively) and the relative accuracy of the solutions found by the CLSTMs after enough iterations is practically equal to 100% (e.g., achieving 0.9995 for the mean relative accuracy after 1,000 iterations).

2) *Impact of neural network structures*: The purpose of this aspect of our experiment is to study the impact of number of neural units in each LSTM layer on the performance of the CLSTMs. Toward this goal, we set the number of neural units in each LSTM layer to 10, 15, 20, 25 and 30 as five different settings. Consequently, the corresponding numbers of parameters in the CLSTMs are 2662, 5792, 10122, 15652 and 22382 in the five settings, respectively. Through comparing the optimal solutions generated by CLSTMs of five different settings after 1,000 iterations, we find that their mean relative accuracy are similar (i.e., the difference between them is less than 0.001) within the range of (0.998, 0.999). Therefore, these confirm that the CLSTMs can perform well with different neural network structures for the considered problem scenarios.

3) *Impact of K*: In this experiment, we show the impact of the parameter K in (8) and (9) on the performance of CLSTMs. Specifically, we train five CLSTMs with the same training procedure except that the K value is set to 1, 3, 5, 7 and 10, respectively.

Fig. 3a presents the mean relative accuracy and the standard deviation of relative accuracy when K is set to the five different values. We can see from the figure that the mean relative accuracy improves and the standard deviation decreases with the increase of K. This is so because that the loss function with a larger value of K contains a longer future impact of a single iteration. Thus, the CLSTMs learning to minimize such loss function can find better update step sizes for minimizing the objective functions. On the other hand, the training process for K equal to 1, 3, 5, 7, 10 consumes 14, 31, 51, 69, 113 minutes of CPU time, respectively. The reason for longer training time for increased K is that the training procedure contains a fixed number of frames and the number of iterations in a frame increases as the K value grows.

Based on these observations from Fig. 3a, we can improve the CLSTMs performance by increasing the value of K, although it will consume more time for training.

4) *Impact of w_k*: This experiment aims to present the impact of w_k on the performance. We fix K to 10 and employ three different strategies to set weights. Specifically, the random strategy chooses weights with random values sampled from a uniform distribution in $[0, 1)$, the decay strategy sets weights to be exponentially decayed with the decay factor equal to 0.9, and the uniform strategy sets all weights to 1. Then, each strategy normalizes the chosen weights so that the sum of all weights chosen by the strategy equals 1. For the training process, three CLSTMs with weights chosen by the three different strategies are trained for 30 epochs. Then, these trained CLSTMs are applied to solve identical 1,000 problem scenarios and the generated solutions are saved after 1,000 iterations.

Fig. 3b presents the value of loss function (8) with different weights setting strategies throughout the training epoch. First of all, we can observe that the curve of the decay strategy becomes relatively “flat” when it comes to the 20 epoch while the curve of the uniform strategy still has obvious fluctuation until the last few epochs. We can further observe that three curves in this figure reach a similar value of the loss function at the end of the training process after 30 epochs. This explains why the trained CLSTMs can generate similar performance (i.e., very narrow range of y-axis values) in Fig. 3c.

Specifically, Fig. 3c shows the mean relative accuracy and the standard derivation of the relative accuracy with three different weight settings. We can see that the CLSTMs trained with randomly selected weights produces the highest standard deviation and the lowest mean relative accuracy, while the CLSTMs trained with weights selected by the uniform strategy generate the lowest standard deviation and the highest mean relative accuracy. Therefore, the strategy of uniform weights provides the most desirable performance. We can further observe that the mean relative accuracy with the decay strategy decreases by 0.0008 and the standard deviation increases by 0.0002 when compared with the uniform strategy. Since this difference is very small, the decay strategy and the uniform strategy are comparable with respect to the performance.

Based on the observations of Fig. 3c and Fig. 3b, we find that the exponential decayed weights can help the CLSTMs converge despite slight performance degradation.

5) *Robustness*: This experiment aims to show the robustness of a trained CLSTM. We use two datasets that are generated from different distributions to train and evaluate the CLSTMs.

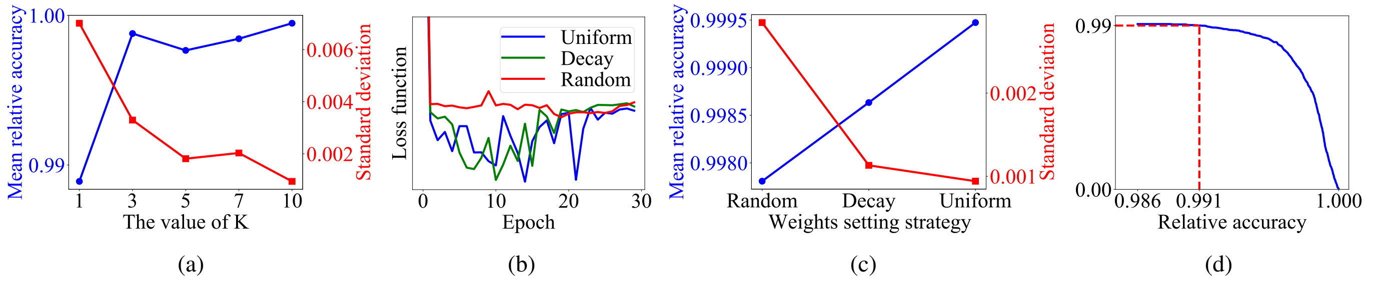

Fig. 3: The impact of (a) the value of K (b) w_k on the the mean relative accuracy and the standard deviation of relative accuracy, (c) The value of the loss function with different weights w_k over epochs, and (d) the CCDF of relative accuracy evaluated using system parameters with distributions different from those used during training

Specifically, for each utility function (15), the parameters μ_n are randomly selected from the set {0.5, 0.6, 0.7, 0.9} in the training dataset, while they are randomly sampled from the uniform distribution in $[0.5, 1.0)$ in the evaluation dataset.

Fig. 3d shows the CCDF of relative accuracy for the uniformly distributed dataset for the CLSTMs evaluation. From this figure, we see that the minimal relative accuracy is 98.6% in Fig. 3d, which is higher than the result of 98.3% in Fig. 2b. On the other hand, the 99 percentile of the relative accuracy is 99.1%, which represents a small reduction of 0.6% when compared with the corresponding result of 99.7% accuracy in Fig. 2b. This slight degradation of 99 percentile of the relative accuracy is intuitively consistent because the combinations of system parameters (i.e., μ_n, R_n, C) in the evaluation dataset are far more random than the combinations included in the training dataset. Nevertheless, these results show that the robustness of the trained CLSTMs as it still can find the optimal solutions for the problem scenarios even when the system parameters are drawn from distributions different from that used for training.

V. Related Work

The term 'learning to optimize' can generally refer to using learning techniques to solve optimization problems. A possible approach is to predict optimal solutions for these optimization problems using the supervised learning techniques. For instance, authors in [1] propose to use a deep neural network to approximate the unknown nonlinear mapping between the parameters of signal processing problems and the optimal solutions. Authors in [2] propose a specific deep neural network structure to predict the optimal solution for constrained optimization problems based on the supervised deep learning techniques and demonstrate that the average prediction error evaluated on a realistic system is as low as 0.2%. Besides the supervised learning techniques, other learning techniques are also applied to solve optimization problems, such as the deep reinforcement learning [3] and the meta-learning [6]. Meanwhile, learning techniques are used to solve other types of optimization problems. For example, authors in [8] focus on solving the Bayesian swarm optimization problem, while authors in [4] and [9] propose to solve the minimax optimization problems by learning. However, none of the aforementioned research considers solving constrained optimization problems without the help of optimal labels.

VI. Conclusion

In this paper, we have proposed the CLSTMs to solve nonconvex, constrained optimization problems. Furthermore, we have formulated a resource-allocation problem and applied the new CLSTMs to solve it by using the practical data from Alibaba. Experiments have been conducted to study the performance of the proposed CLSTMs. By considering 1,000 scenarios of the resource-allocation problem, our numerical results have shown that (1) the trained CLSTMs can find the near-optimal and stable solutions in very few iterations (e.g., achieving 99% and 99.9% optimality accuracy after 2 and 12 iterations, respectively), (2) the CLSTMs include a number of selectable hyper-parameters to trade off the training time for the solution quality, and (3) the proposed approach is robust as the trained CLSTMs can produce excellent solutions for problems with system parameters drawn from distributions different from those used in the training process.

References

[1] Sun, Haoran and Chen, Xiangyi and Shi, Qingjiang and Hong, Mingyi and Fu, Xiao and Sidiropoulos, Nicholas D., "Learning to Optimize: Training Deep Neural Networks for Interference Management," in IEEE Transactions on Signal Processing, vol. 66, no. 20, pp. 5438-5453, 2018.

[2] Fioretto, Ferdinando, Mak, Terrence W.K. and Van Hentenryck, Pascal, "Predicting AC Optimal Power Flows: Combining Deep Learning and Lagrangian Dual Methods", Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34(01), pp. 630-637, 2020.

[3] Li, Ke, and Jitendra Malik. "Learning to optimize." Proceedings of 5th International Conference on Learning Representations (ICLR), 2017.

[4] Shen, Jiayi, Xiaohan Chen, Howard Heaton, Tianlong Chen, Jialin Liu, Wotao Yin, and Zhangyang Wang. "Learning a minimax optimizer: A pilot study." Proceeding of 9th ICLR, 2021.

[5] Sepideh Nazemi, Kin K. Leung and Ananthram Swami. "Distributed Optimization Framework for In-Network Data Processing," IEEE/ACM Trans. on Networking, vol. 27, pp. 2432–2443, 2019.

[6] Marcin Andrychowicz, et al. "Learning to learn by gradient descent by gradient descent," Advances in neural information processing systems, pp. 3981–3989, 2016.

[7] Alibaba Inc. 2018. Alibaba production cluster data v2018. Website. //github.com/alibaba/clusterdata/tree/v2018.

[8] Yue Cao, Tianlong Chen, Zhangyang Wang, and Yang Shen. Learning to optimize in swarms. In Advances in Neural Information Processing Systems, pp. 15018–15028, 2019.

[9] Xiong, Yuanhao, and Cho-Jui Hsieh. "Improved Adversarial Training via Learned Optimizer." In European Conference on Computer Vision, pp. 85-100. Springer, Cham, 2020.