Use Coupled LSTM Networks to Solve Constrained Optimization Problems

Zheyu Chen, *Student Member, IEEE,* Kin K. Leung, *Fellow, IEEE,* Shiqiang Wang, *Member, IEEE,* Leandros Tassiulas, *Fellow, IEEE,* Kevin Chan, *Senior Member, IEEE,* Don Towsley, *Fellow, IEEE*

***Abstract*—Gradient-based iterative algorithms have been widely used to solve optimization problems, including resource sharing and network management. When system parameters change, it requires a new solution independent of the previous parameter settings from the iterative methods. Therefore, we propose a learning approach that can quickly produce optimal solutions over a range of system parameters for constrained optimization problems. Two Coupled Long Short-Term Memory networks (CLSTMs) are proposed to find the optimal solution. The advantages of this framework include: (1) near-optimal solution for a given problem instance can be obtained in few iterations during the inference, (2) enhanced robustness as the CLSTMs can be trained using system parameters with distributions different from those used during inference to generate solutions. In this work, we analyze the relationship between minimizing the loss functions and solving the original constrained optimization problem for certain parameter settings. Extensive numerical experiments using datasets from Alibaba reveal that the solutions to a set of nonconvex optimization problems obtained by the CLSTMs reach within 90% or better of the corresponding optimum after 11 iterations, where the number of iterations and CPU time consumption are reduced by 81% and 33%, respectively, when compared with the gradient descent with momentum.**

***Index Terms*—Optimization method, resource management, neural networks, iterative methods**

I. Introduction

CONSTRAINED optimization problems are widely used to study and resolve various technical issues in the networks and computer infrastructures, such as the resource allocation [2]–[4], and SDN-based network management [5]–[7]. Since the system parameters are constituent elements in the constrained optimization problems, it requires a new optimal solution when the system parameters change, independent of the previous parameter settings for the same system that have been considered. It is time-consuming to obtain new optimal solutions by using the conventional gradient-based iterative algorithms. Therefore, it is helpful to develop machine-learning solution frameworks that can quickly produce solutions over a range of system parameters.

The constrained optimization problem (P1) under consideration is given as follows:

$$\text{(P1)}\quad \min_{x} f(x) \quad \text{s.t.}\quad h(x) \le 0,$$

where $x \in \mathbb{R}^n$. Note that we include only one constraint in the above problem to simplify our presentation, although the proposed CLSTM can be applied to solve optimization problems with multiple constraints, as shown in Section IV.

To solve this problem, it requires to find the optimal x to minimize the objective function $f(x)$ and satisfy the constraint that the value of the function $h(x)$ is non-positive. Note that this is a very general formulation of the constrained optimization, and the objective and constraint functions can be either convex or nonconvex.

Researchers have proposed to use supervised learning techniques to solve constrained optimization problems. For example, in [8] and [9], the supervised learning techniques are modified and enhanced to predict the optimal solutions for given sets of optimization problem parameters. However, to train a well-performed prediction model needs sufficient data samples containing the input features and the ground truth labels. Thus, it requires extra effort to generate plenty of optimal solutions for these constrained optimization problems, which may be difficult to solve.

Using unsupervised learning techniques to solve these constrained optimization problems is more efficient. For instance, Gao *et al.* in [11] focus on solving optimization problems with constraints in the form of Symmetric Positive Definite matrices, while a LSTM network is used to find the optimal policy for the constrained Markov decision process in [12]. However, the aforementioned work is restricted only to specific types of constrained optimization problems and for more general constrained optimization problems, the question about how to use unsupervised learning techniques (e.g., LSTMs) to solve them is still open.

To address this open issue, the main objective of this work is to develop an unsupervised-learning-based solution

Zheyu Chen and Kin K. Leung are with the Department of Computing, Imperial College, London, UK (e-mail:{z.chen19, kin.leung}@imperial.ac.uk)

Shiqiang Wang is with IBM T.J. Watson Research Center, Yorktown Heights, NY, USA (e-mail: wangshiq@us.ibm.com)

Leandros Tassiulas is with Yale University, New Haven, CT, USA (e-mail: leandros.tassiulas@yale.edu)

Kevin Chan is with DEVCOM Army Research Laboratory, Adelphi, MD, USA (e-mail: mailto:kevin.s.chan.civ@army.mil)

Don Towsley is with the Department of Computer Science, University of Massachusetts, Amherst, MA, USA. (e-mail: towsley@cs.umass.edu)

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. L. Tassiulas work was partially supported by project ONR N00173-21-1-G006:

Early results of this work have been presented in 2021 IEEE Military Communications Conference [1].

framework that can quickly generate optimal solutions for general constrained optimization problems over a range of system parameters. The contributions of this paper are as follows:

- Propose 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.
- Identify the need and propose a projection function to ensure the Lagrangian multiplier non-negative and avoid computational issues, and present appropriate the design criteria for selecting the projection function that can enable the CLSTMs to find the optimal solutions.
- Analyze the impact of the projection function on the optimal solutions and the relationship between minimizing the loss functions and solving the original constrained optimization problem with special parameter settings.
- Formulate 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 [13].
- Our evaluation results demonstrate that the CLSTMs can achieve 90% of the optimum after only 11 iterations even when the optimization problems are nonconvex. Furthermore, we conduct an experiment to validate and show the robustness of the proposed technique where the CLSTMs are 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 details of the coupled LSTM networks. Section III analyzes of the proposed CLSTMs and training process. Section IV describes the resource-allocation problem under study and the formulated constrained optimization problem. Section V presents the evaluation of the coupled LSTM networks using the cluster trace. Finally, section VI discusses related research and section VII concludes the paper.

II. Proposed CLSTMs

In this section, we propose the CLSTMs for solving constrained optimization problem P1 in the last section.

By introducing a Lagrange multiplier λ, a Lagrange function can be formed for the optimization problem (P1):

$$J(x,\lambda) = f(x) + \lambda h(x). \tag{1}$$

The dual optimization problem of (P1) is

$$\begin{aligned} \text{(P2)}\ \ & \max_{\lambda} J(x^*,\lambda) \\ & \text{s.t.}\ \ x^* = \text{argmin}\ J(x,\lambda), \\ & \qquad \lambda \geq 0. \end{aligned}$$

According to the duality theory [14], the dual optimization problem (P2) has the same optimal solution for the original (primal) problem (P1) under the condition of zero duality gap. Therefore, our objective is to find the optimal λ^* for maximizing the function J and the associated x^*.

Firstly, we describe the inference process that the two coupled LSTM networks, m and $\hat{m}$, are utilized to find the optimal

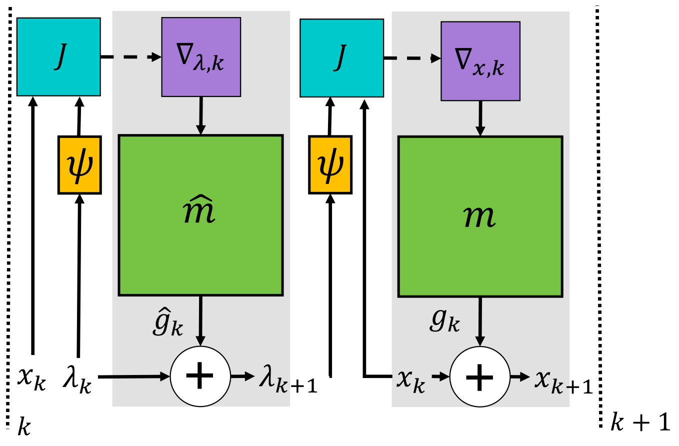

Fig. 1. Computation graph of the coupled LSTM for iteration k, where $\nabla_{x,k} = \nabla_x J(x_k, \psi(\lambda_{k+1}))$, $\nabla_{\lambda,k} = \nabla_\lambda J(x_k, \psi(\lambda_k))$.

x^* and λ^* for a given Lagrange function J, respectively, by iterations. In each iteration, to satisfy the requirement of non-negative Lagrangian multiplier, the projection function ψ is used to map any λ into a value larger than or equal to 0. (The projection function will be further discussed in Section II-A.) The overall workflow of the proposed method is shown in Fig. 1 where iterations indexed by k progress from left to right. Note that x and λ are updated alternately and the iteration can start from either of them. Without loss of generality, we define that the iteration starts with updating λ. In each iteration k, the update step sizes (changes) of x and λ are denoted by g_k and $\hat{g}_k$, respectively. Specifically, λ is updated according to:

$$\begin{bmatrix} \hat{g}_k \\ \hat{h}_{k+1} \end{bmatrix} = \hat{m}(\nabla_\lambda J(x_k, \psi(\lambda_k)), \hat{h}_k, \hat{\phi}^*), \tag{2}$$

$$\lambda_{k+1} = \lambda_k + \hat{g}_k, \tag{3}$$

where $\hat{\phi}^*$ denotes the optimal parameters in the LSTM $\hat{m}$, $\nabla_\lambda J(x_k, \psi(\lambda_k))$ is the gradient of function J with respect to (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. Then x is updated from iteration k to $k+1$ by the following equations:

$$\begin{bmatrix} g_k \\ h_{k+1} \end{bmatrix} = m(\nabla_x J(x_k, \psi(\lambda_{k+1})), h_k, \phi^*), \tag{4}$$

$$x_{k+1} = x_k + g_k, \tag{5}$$

where ϕ^* denotes the optimal parameters in the LSTM m, $\nabla_x J(x_k, \psi(\lambda_{k+1}))$ is the gradient of function J w.r.t. x, and h_k, h_{k+1} are the hidden state for m in iteration k and $k+1$, respectively. For the sake of conciseness, the hidden states h_k, h_{k+1}, $\hat{h}_k$ and $\hat{h}_{k+1}$ are omitted from the CLSTM architecture in Fig. 1.

The training process is to find the optimal parameters for m and $\hat{m}$. During training, we define K consecutive iterations as a frame. In each iteration k within frame i, for a given Lagrange function J, the update step sizes $\hat{g}_k$ and g_k are generated according to:

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

$$\begin{bmatrix}g_k \\ h_{k+1}\end{bmatrix} = m(\nabla_x J(x_k, \psi(\lambda_{k+1})), h_k, \phi_i), \tag{7}$$

where $\hat{\phi}_i$ and ϕ_i denote the parameters in $\hat{m}$ and m, respectively. Using the generated $\hat{g}_k$ and g_k, λ and x are updated according to (3) and (5). At the end of frame i (i.e., after K iterations), the parameters ϕ_i are updated to minimize the loss function:

$$L(\phi_i) = \mathbf{E}\Bigg[\sum_{k=(i-1)K}^{iK-1} w_k J(x_k, \psi(\lambda_{k+1})) + w_{iK} J(x_{iK}, \psi(\lambda_{iK}))\Bigg], \tag{8}$$

where $w_{(i-1)K}, \dots, w_{iK}$ are weighting factors and the sum of them equals 1. Similarly, we update the parameters $\hat{\phi}_i$ to minimize the loss function:

$$\hat{L}(\hat{\phi}_i) = -\mathbf{E}\Bigg[\sum_{k=(i-1)K}^{iK} \hat{w}_k J(x_k, \psi(\lambda_k))\Bigg], \tag{9}$$

where $\hat{w}_{(i-1)K}, \dots, \hat{w}_{iK}$ are another set of weighting factors and the sum of them also equals 1. It is worth noting that the forms of the objective and constraint functions are assumed to be fixed. However, the associated function parameters are randomly chosen from some distributions for the training process. To consider such random functions, the expectation is needed in (8) and (9).

Furthermore, to obtain sufficient sampling and experience in the search process for the optimal solutions, we define a group of I consecutive frames as an epoch, where the variables (i.e., λ and x) 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 1, which works as follows. After the parameters ϕ_0 and $\hat{\phi}_0$ are randomly initialized (Line 1), the variables (i.e., λ and x) and the hidden states (i.e., h_k and $\hat{h}_k$) are set to randomly generated values at the beginning of each epoch (Lines 3-4). Then at the beginning of a frame, all variables and Lagrange multipliers are updated in K iterations (Lines 6-15). In every iteration, for each sample (J, x, λ) in the training dataset, the Lagrange multipliers λ are updated (Lines 8-10) before the variables x are updated (Lines 11-13). Finally, at the end of each frame, the parameters ϕ_i and $\hat{\phi}_i$ are updated (Lines 16-17).

A. Projection Function

Note that the Lagrange multiplier λ is required to be non-negative in the dual optimization problem (P2). To satisfy this constraint and avoid potential numerical issues, we propose to use a projection function $\psi : \mathbb{R} \to \mathbb{R}$ to map any λ into a value larger than or equal to 0. That is,

Requirement 1: The basic requirement of the projection function ψ is that $u = \psi(\lambda)$ where $u \in [0, \infty)$ for all $\lambda \in \mathbb{R}$.

Algorithm 1 Training Procedure

```
1:  Randomly initialize the parameters $\phi_0$ and $\hat{\phi}_0$ for $m$ and $\hat{m}$, respectively;
2:  for epoch =1,2,... do
3:    Randomly initialize the values of $x, \lambda$ for each function $J$ in the training dataset;
4:    Randomly initialize the hidden state $h_k, \hat{h}_k$ for $m$ and $\hat{m}$, respectively;
5:    for frame $i = 1, 2, \dots, I$ do
6:      for iteration $k = (i-1)K, \dots, iK-1$ do
7:        for $(J, x, \lambda)$ in the training dataset do
8:          Calculate the gradient of function $J$ w.r.t. $\lambda$;
9:          Generate the update step size of $\lambda$ by (6)
10:         Update $\lambda$ using (3);
11:         Calculate the gradient of function $J$ w.r.t. $x$;
12:         Generate the update step size of $x$ by (7);
13:         Update $x$ using (5);
14:       end for
15:     end for
16:     Calculate the loss functions $L(\phi_i)$ and $\hat{L}(\hat{\phi}_i)$ using (8)(9), respectively;
17:     Update the parameters $\phi_i$ and $\hat{\phi}_i$ using the gradients $\nabla_{\phi_i} L(\phi_i)$ and $\nabla_{\hat{\phi}_i} \hat{L}(\hat{\phi}_i)$, respectively;
18:   end for
19: end for
```

Since the projection function is used in the training process, its impact on the training also needs to be considered. By applying the chain rule, the derivative of $\hat{L}$ w.r.t each parameter $\hat{\phi}_i$ can be expanded as

$$\frac{\partial \hat{L}}{\partial \hat{\phi}_i} = -\mathbb{E}\Bigg[\sum_{k=(i-1)K}^{iK} \hat{w}_k \frac{\partial J}{\partial u_k}\frac{\partial u_k}{\partial \lambda_k}\frac{\partial \lambda_k}{\partial \hat{\phi}_i}\Bigg], \tag{10}$$

where $u_k = \psi(\lambda_k)$.

From (10), we can observe that the derivative of the projection function w.r.t. λ_k is a part of the derivative of the loss function $\hat{L}$. Note that the gradient-based optimization method (e.g., Adam [15]) and the backpropagation are used to update the parameters $\hat{\phi}_i$ using the gradients $\nabla_{\hat{\phi}_i} \hat{L}(\hat{\phi}_i)$ (line 17 in Algorithm 1). Thus, we need the following criteria to ensure that $\frac{\partial \hat{L}}{\partial \hat{\phi}_i}$ will not be distorted by $\frac{\partial \psi(\lambda_k)}{\partial \lambda_k}$.

Firstly, $\frac{\partial \hat{L}}{\partial \hat{\phi}_i}$ should be defined everywhere so that the parameters of the CLSTMs can be updated and optimized. We note that $\frac{\partial \hat{L}}{\partial \hat{\phi}_i}$ cannot be determined if $\frac{\partial \psi(\lambda_k)}{\partial \lambda_k}$ is not defined. Therefore, we have the following:

Requirement 2: The projection function ψ should be differentiable everywhere.

Obviously, the most ideal situation is that $\frac{\partial \psi(\lambda_k)}{\partial \lambda_k}$ is 1 everywhere so that its impact on $\frac{\partial \hat{L}}{\partial \hat{\phi}_i}$ can be eliminated as much as possible. However, this requirement is hard to fulfil since a function with a slope equal to 1 cannot map both negative and positive values of λ_k to positive values simultaneously. Therefore, we propose the following criteria as a relaxation:

Requirement 3: (a) The derivative of the projection function ψ becomes a constant, which can be different from 1, when $|\lambda| \to \infty$; (b) The two constants (i.e., the two values of the derivative of the function ψ when $\lambda \to \infty$ and $\lambda \to -\infty$, respectively) are non-zero; (c) The two constants should not to be too small or large to avoid numerical issues.

By using a projection function that satisfies the aforementioned three requirements, the magnitude of gradient $\frac{\partial \hat{L}}{\partial \hat{\phi}_i}$ will not be extremely large or small because the derivative of the projection function $\frac{\partial \psi(\lambda_k)}{\partial \lambda_k}$ is not too small or large. This is important because the extreme magnitude of the gradient will cause the CLSTMs unstable during the training process and the parameters of the CLSTMs will not be properly updated if the gradients become too small. Therefore, using the projection function satisfying the requirement can lead to proper computations without hindering the convergence of the training process, as our experiments show.

It is interesting to observe that the function returning the absolute value of input (i.e., $\psi(\lambda) = |\lambda|$) satisfies Requirement 3, although it does not meet Requirement 2. Nevertheless, inspired by this observation, we use a rough approximation for the function $\psi(\lambda) = |\lambda|$ as the projection function as follows:

$$\psi(\lambda) = \begin{cases} -a\lambda - (a-1), & \text{if } \lambda < -1 \\ \lambda^a, & \text{if } -1 \le \lambda \le 1 \\ a\lambda - (a-1), & \text{if } \lambda > 1 \end{cases} \tag{11}$$

where a can be any even number including 2. This projection function is obtained by the following two steps. Firstly, we set $\psi = \lambda^a$ when $-1 \le \lambda \le 1$ where a is a positive, even integer. Secondly, when $1 \le \lambda$, we choose $\psi(\lambda) = a\lambda - (a-1)$. The main idea is that when λ becomes very large, $\psi(\lambda) \approx a\lambda$ reflects the linear growth of $|\lambda|$ as λ increases. Similarly, we select $\psi(\lambda) = -a\lambda - (a-1)$ for $\lambda \le -1$. It is important to note that such $\psi(\lambda)$ defined in (11) captures only the increasing or decreasing trend of $|\lambda|$ as λ changes. However, (11) does not give the precise value of $|\lambda|$ as it is not needed in search for the optimal solution.

Clearly, the proposed projection function in (11) possesses the following properties.

Lemma 1: The proposed projection function ψ is differentiable and the derivative of function ψ is equal to a and $-a$ when $\lambda \to \infty$ and $\lambda \to -\infty$, respectively.

Note that the symmetric property is not required although the proposed projection function in (11) is symmetric. Furthermore, although a is not limited to be specific even numbers, 2 is a better choice than other even numbers since the function ψ with a equal to 2 can better approximate $|\lambda|$ than any other integer larger than 2.

B. Reduction of Computation Complexity and Storage Needs

For each LSTM in the CLSTMs structure, 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 [17] 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.

Furthermore, to reduce the 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 x and λ are independent of ϕ and $\hat{\phi}$, respectively (i.e., $\frac{\partial \nabla_x J}{\partial \phi_i} = 0$, $\frac{\partial \nabla_\lambda J}{\partial \hat{\phi}_i} = 0$).

III. Analysis

In this section, we present the analysis of the proposed CLSTMs and training process. Firstly, we make the following assumption for the constrained optimization problem P1.

Assumption 1: We assume that:

1) The strong duality holds (i.e., the duality gap is zero).

2) There exists at least a dual optimal λ^* and a primal optimal x^*.

Note that the strong duality is not equivalent to the convexity and non-convex optimization problems can also possess the strong duality property [18]. As a result of Assumption 1, the optimization problems P2 and P1 have same optimal solution. The zero-duality gap property helps us solve a wide range of constrained optimization problems of interest in communications and networking [18]–[21].

A. Transformed Optimization Problem

Since the projection function ψ is employed to keep the Lagrangian multiplier non-negative and avoid numerical issues, the dual optimization problem P2 for which the CLSTMs are trained to solve can be transformed into:

$$\begin{aligned} \text{(P3)}\ \ &\max_{\lambda} J(x^*, \psi(\lambda)) \\ &\text{s.t.}\ \ x^* = \text{argmin}\ J(x, \psi(\lambda)). \end{aligned}$$

Theorem 1: Having λ^* as the optimal solution to the problem P3 is equivalent to having u^* as the optimal solution to the problem P2, where $u^* = \psi(\lambda^*)$.

Proof: If u^* is not the optimal solution to P2, there at least exists the optimal solution u^{**} according to Assumption 1. Then, we have $J(x^{**}, u^{**}) > J(x^*, u^*)$, where $x^{**} = \text{argmin}\ J(x, \psi(\lambda^{**}))$ and $x^* = \text{argmin}\ J(x, \psi(\lambda^*))$. For any given u^{**}, there at least exists a λ^{**} such that $u^{**} = \psi(\lambda^{**})$. As a result, we have $J(x^{**}, \psi(\lambda^{**})) > J(x^*, \psi(\lambda^*))$ and thus, λ^* is not the optimal solution to P3. Therefore, we can conclude that if λ^* is the optimal solution to P3, then $u^* = \psi(\lambda^*)$ is the optimal solution to problem P2. Obviously, if u^* is the optimal solution to P2, we have $J(x^*, u^*) \ge J(x, u)$ for all u, where $u = \psi(\lambda)$ and $x = \text{argmin}\ J(x, \psi(\lambda))$. Since $u^* = \psi(\lambda^*)$, we then have $J(x^*, u^*) = J(x^*, \psi(\lambda^*)) \ge J(x, u) = J(x, \psi(\lambda))$ for all λ. Therefore, if u^* is the optimal solution to P2, λ^* is the optimal solution to the problem with $u^* = \psi(\lambda^*)$. ■

This theorem has following corollaries:

Corollary 1.1: There exists at least one λ^* that is optimal for the problem P3.

Since there must exist u^* which is optimal for the problem P2 due to Assumption 1, Theorem 1 implies that λ^* is optimal for the problem P3 where $u^* = \psi(\lambda^*)$.

Corollary 1.2: Using the optimal λ^* for the problem P3, we can find the primal optimal solution x^*.

Finding the optimal solution x^* is straightforward since solving argmin $J(x, \psi(\lambda^*))$ and argmin $J(x, u^*)$ are identical because $u^* = \psi(\lambda^*)$ and u^* is optimal for the problem P2.

B. Loss Function

In the training process, (8) and (9) are approximated by using D training samples (i.e., D different instances of the primal optimization problem P1 and the corresponding instances of the dual optimization problem P2) as:

$$L(\phi_i) = \frac{1}{D}\sum_{d=1}^{D}\Big[\sum_{k=(i-1)K}^{iK-1} w_k J^{(d)}(x_k^{(d)}, \psi(\lambda_{k+1}^{(d)})) + w_{iK} J^{(d)}(x_{iK}^{(d)}, \psi(\lambda_{iK}^{(d)}))\Big], \quad (12)$$

$$\hat{L}(\hat{\phi}_i) = -\frac{1}{D}\sum_{d=1}^{D}\Big[\sum_{k=(i-1)K}^{iK} \hat{w}_k J^{(d)}(x_k^{(d)}, \psi(\lambda_k^{(d)}))\Big], \quad (13)$$

where $x_k^{(d)}$ and $\lambda_k^{(d)}$ are the optimization variables and Lagrange multiplier, respectively, for the Lagrange function $J^{(d)}$ in iteration k.

Theorem 2: Assuming that each frame includes only one iteration (i.e., K=1) and all weighting factors, w_k and $\hat{w}_k$, are set to 1, the problem P1 is solved during the training process, when the loss functions (12) and (13) reach their minimal values.

Proof: When $K = 1$, the parameters of the two CLSTMs will be updated at the end of each iteration and each frame includes only one iteration. Thus, by assigning K to 1 in the loss functions (12) and (13) we have:

$$L(\phi_i) = \frac{1}{D}\sum_{d=1}^{D}\Big[w_{i-1} J^{(d)}(x_{i-1}^{(d)}, \psi(\lambda_i^{(d)})) + w_i J^{(d)}(x_i^{(d)}, \psi(\lambda_i^{(d)}))\Big], \quad (14)$$

$$\hat{L}(\hat{\phi}_i) = -\frac{1}{D}\sum_{d=1}^{D}\Big[\hat{w}_{i-1} J^{(d)}(x_{i-1}^{(d)}, \psi(\lambda_{i-1}^{(d)})) + \hat{w}_i J^{(d)}(x_i^{(d)}, \psi(\lambda_i^{(d)}))\Big], \quad (15)$$

where $x_i^{(d)}$ and $\lambda_i^{(d)}$ are the optimization variables and Lagrange multiplier, respectively, for the Lagrange function $J^{(d)}$ in iteration i, and

$$\begin{bmatrix} g_i^{(d)} \\ h_{i+1}^{(d)} \end{bmatrix} = m(\nabla_x J^{(d)}(x_{i-1}^{(d)}, \psi(\lambda_i^{(d)})), h_i^{(d)}, \phi_i),$$
$$x_i^{(d)} = x_{i-1}^{(d)} + g_i^{(d)},$$
$$\begin{bmatrix} \hat{g}_i^{(d)} \\ \hat{h}_{i+1}^{(d)} \end{bmatrix} = \hat{m}(\nabla_\lambda J^{(d)}(x_{i-1}^{(d)}, \psi(\lambda_{i-1}^{(d)})), \hat{h}_i^{(d)}, \hat{\phi}_i),$$
$$\lambda_i^{(d)} = \lambda_{i-1}^{(d)} + \hat{g}_i^{(d)}.$$

Here two different cases are considered according to the position of the current iteration within an epoch:

1) *Case 1*: the current iteration is the first iteration in an epoch (i.e., $i = 1$), then the values of $x_{i-1}^{(d)}$ and $\lambda_{i-1}^{(d)}$ are randomly initialized.

2) *Case 2*: the current iteration is not the first iteration, the values of $x_{i-1}^{(d)}$ and $\lambda_{i-1}^{(d)}$ are given by the following equations:

$$\begin{bmatrix} g_{i-1}^{(d)} \\ h_i^{(d)} \end{bmatrix} = m(\nabla_x J^{(d)}(x_{i-2}^{(d)}, \psi(\lambda_{i-1}^{(d)})), h_{i-1}^{(d)}, \phi_{i-1}),$$
$$x_{i-1}^{(d)} = x_{i-2}^{(d)} + g_{i-1}^{(d)},$$
$$\begin{bmatrix} \hat{g}_{i-1}^{(d)} \\ \hat{h}_i^{(d)} \end{bmatrix} = \hat{m}(\nabla_\lambda J^{(d)}(x_{i-2}^{(d)}, \psi(\lambda_{i-2}^{(d)})), \hat{h}_{i-1}^{(d)}, \hat{\phi}_{i-1}),$$
$$\lambda_{i-1}^{(d)} = \lambda_{i-2}^{(d)} + \hat{g}_{i-1}^{(d)}.$$

We can see that in both cases, the values of $x_{i-1}^{(d)}$ and $\lambda_{i-1}^{(d)}$ do not depend on the parameters ϕ_i and $\hat{\phi}_i$, respectively. Therefore, the gradients of the first term of (14) (i.e., $w_{i-1}J^{(d)}(x_{i-1}^{(d)}, \psi(\lambda_i^{(d)}))$) w.r.t. the parameters ϕ_i and the gradients of the first term of (15) (i.e., $\hat{w}_{i-1}J^{(d)}(x_{i-1}^{(d)}, \psi(\lambda_{i-1}^{(d)}))$) w.r.t. the parameters $\hat{\phi}_i$ are zero. Based on this observation, the loss functions (14) and (15) can be simplified as:

$$L(\phi_i) = \frac{1}{D}\sum_{d=1}^{D} J^{(d)}(x_i^{(d)}, \psi(\lambda_i^{(d)})),$$
$$\hat{L}(\hat{\phi}_i) = -\frac{1}{D}\sum_{d=1}^{D} J^{(d)}(x_i^{(d)}, \psi(\lambda_i^{(d)})),$$

where w_i and $\hat{w}_i$ are eliminated since they are set to 1.

Therefore, minimizing the above two loss functions in each iteration simultaneously corresponds to solving the following minimax optimization problem:

$$\text{(P4)} \quad \max_{\hat{\phi}_i} \min_{\phi_i} \frac{1}{D}\sum_{d=1}^{D} J^{(d)}(x_i^{(d)}, \psi(\lambda_i^{(d)})).$$

When the two loss functions achieve their minimal values, this minimax optimization problem is solved. Since these D optimization problem instances $J^{(d)}$ are independent, $x_i^{(d)}$ and $\lambda_i^{(d)}$ generated by the CLSTMs with the optimal parameters ϕ_i and $\hat{\phi}_i$ should form a saddle-point for the function $J^{(d)}(x_i^{(d)}, \psi(\lambda_i^{(d)}))$, for all d. Thus, the generated $x_i^{(d)}$ and $\psi(\lambda_i^{(d)})$ form a saddle-point for the function $J^{(d)}$, for all d. Consequently, for the original optimization problem P1 for which the Lagrange function $J^{(d)}$ is formed, $x_i^{(d)}$ and $u^{(d)}$ are primal and dual optimal points where $u^{(d)} = \psi(\lambda_i^{(d)})$. ■

IV. Formulation of Resource-Allocation Problem

A. System Model

The resource-allocation problem under consideration is to allocate cluster resources to competing jobs for maximizing the sum of job utilities. Specifically, there are N jobs competing for one type of resource and the amount of available resource is denoted by C. For each job 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 job n must be allocated with a minimum amount of resource to provide satisfactory 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 jobs. By introducing two parameters $\alpha < 1$ and $\beta > 1$, the minimum and the maximum resource requirement of job n are denoted by αR_n and βR_n, respectively.

B. Optimization Problem

By using these notations, we can formulate the resource-allocation problem as the following optimization problem:

$$\max_{x_1,\ldots,x_N} \sum_{n=1}^{N} u_n(x_n) \tag{17a}$$

$$\text{s.t.} \quad \sum_{n=1}^{N} x_n \leq C, \tag{17b}$$

$$x_n \geq \alpha R_n, \forall n, \tag{17c}$$

$$x_n \leq \beta R_n, \forall n, \tag{17d}$$

where C is the amount of available resource, and x_n, R_n, and $u_n(\cdot)$ denote the amount of resource allocated to job n, the resource requirement, and the utility function of job n, respectively.

The objective function (17a) maximizes the sum utilities of all jobs by resource allocation. The first constraint (17b) ensures that the amount of allocated resources for all jobs must not exceed the amount of available resources, while the constraints (17c) guarantee that the minimum resource requirements for jobs are satisfied. The constraints (17d) ensure that the amount of allocated resources for each job must not exceed its maximum resource requirement.

C. Solve the Problem with the CLSTMs

Let $\boldsymbol{x}$ denote the vector of variables $[x_1, \ldots, x_N]$ and define the objective function $f(\boldsymbol{x})$ and the constraint function $\boldsymbol{h}(\boldsymbol{x})$ as

$$f(\boldsymbol{x}) = -\sum_{n=1}^{N} u_n(x_n),$$

$$\boldsymbol{h}(\boldsymbol{x}) = \begin{bmatrix} \sum_{n=1}^{N} x_n - C \\ \alpha R_1 - x_1 \\ \ldots \\ \alpha R_N - x_N \\ x_1 - \beta R_1 \\ \ldots \\ x_N - \beta R_N \end{bmatrix}.$$

The optimization problem above represents a particular instance of P1:

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) \tag{18a}$$

$$\text{s.t.} \quad \boldsymbol{h}(\boldsymbol{x}) \leq 0. \tag{18b}$$

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

$$J(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}\boldsymbol{h}(\boldsymbol{x}).$$

Finally, by substituting $J(\boldsymbol{x}, \boldsymbol{\lambda})$ for $J(x, \lambda)$, $\boldsymbol{x}$ for x, $\boldsymbol{\lambda}$ for λ into Algorithm 1 and applying the projection function to $\boldsymbol{\lambda}$ element by element, we can use Algorithm 1 to train the CLSTMs for solving this constrained optimization problem. It is worth noting that we do not make specific assumptions about the forms of the functions $f(x)$ and $h(x)$ in (18), although the problem (18) is assumed to satisfy the zero-duality gap.

V. Numerical Experiments

A. Setup

The Alibaba cluster trace [13] 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 machines in terms of utilization in percentage to various jobs, where the CPU requirement of a job is the aggregate CPU requirement of all its tasks. We employ a cluster of 5 machines to provide CPU resource to 10 competing jobs in all optimization problem scenarios considered in the following experiments. In each problem scenario, the amount of available CPU resource and the CPU requirements of jobs are randomly selected from the Alibaba cluster trace. Therefore, for all problem scenarios used for the training and the evaluation, each scenario has 10 control variables and 21 constraints. The associated optimization problem is convex when problem scenarios have convex utility functions, and the problem is nonconvex when the sigmoid utility functions are used.

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 the layer with 20 neural units. For both the training and evaluation (inference) phases, we employ the function defined in (11) with a equal to 2 as the projection function. During the training process, the CLSTMs are trained with 5,120 optimization problem scenarios. The training process consists of 50 epochs where each epoch has $\lfloor \frac{2000}{K} \rfloor$ frames ($I = \lfloor \frac{2000}{K} \rfloor$) and each frame consists of K iterations. We set $w_{k,\forall k}$ to 1 and the learning rate in frame i to $0.01 \times 0.95^{\frac{i-1}{100}}$. For the evaluation, the trained CLSTMs are used to solve 1,000 problem scenarios with randomly selected parameters. For each problem scenario, the optimization (control) variables are updated using the trained CLSTMs by iterations and the final solutions are saved after 1,000 iterations.

We employ the fmincon from the Optimization-toolbox in Matlab R2016a to produce the optimal solutions, which is guaranteed to find the optimal solutions for convex problems. Furthermore, two gradient-based methods are used to serve as baselines for comparison with the trained CLSTMs. To solve the optimization problem P2, these two methods update the variables x and λ by iterations. In each iteration, the first method, referred to as Gradient Descent (GD), updates x and λ by gradient descent and gradient ascent, respectively, while

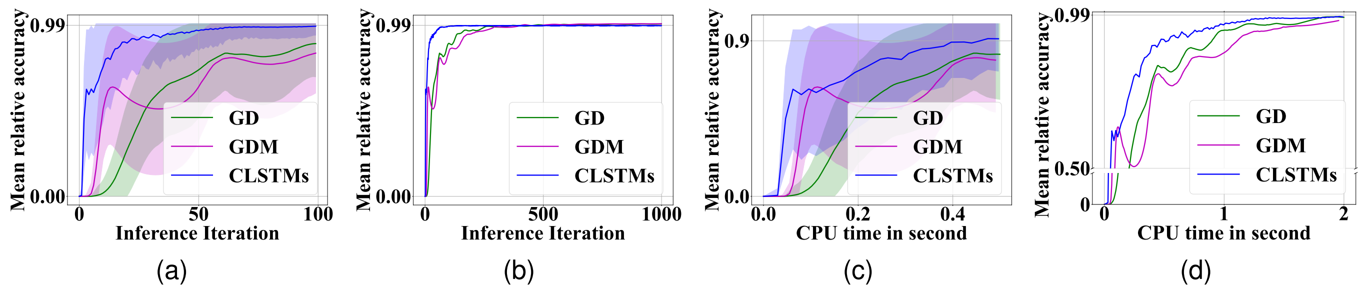

Fig. 2. The mean relative accuracy over (a) 100 iterations, (b) 1,000 iterations, (c)(d) CPU time in seconds.

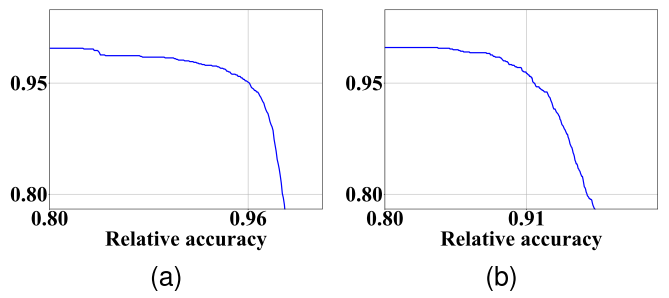

Fig. 3. The complementary cumulative distribution function (CCDF) of relative accuracy with (a) convex utility functions, (b) sigmoid utility functions.

the second method, named Gradient Descent with Momentum (GDM) [22], [23], updates x and λ by gradient descent with momentum and gradient ascent with momentum, respectively. Therefore, the second method is supposed to converge faster than the first one.

To measure performance of the proposed CLTMs approach, we define the *relative accuracy* of a solution as $1-\frac{|\hat{f}-f|}{|f|}$ where $\hat{f}$ and f are the optimal values of the objective function found by the CLSTMs and the fmincon, respectively. Moreover, we define the *mean relative accuracy* as the relative accuracy averaged over the 1,000 problem scenarios solved by the trained CLTMs in the evaluation process.

B. Results and Analysis with Convex Utility Functions

In this subsection, we consider convex utility functions for which the optimal results can be accurately obtained by the Matlab tool fmincon for the purpose of demonstrating the performance and the robustness of the proposed CLSTMs. For each job n, its utility function given the allocated CPU utilization x_n is given by

$$u_n(x_n) = -\mu_n(\frac{x_n}{R_n} - 1)^2 + \frac{x_n}{R_n},$$

where μ_n is a constant randomly selected from the uniform distribution in the range of $[0.001, 10)$ and R_n is the CPU utilization requirement of job n. Moreover, α and β are set to 0 and 5, respectively, for all jobs.

1) *Compare with the baselines*

In this experiment, we demonstrate that the CLSTMs can find the near-optimal solutions in a few iterations and obtain extremely high relative accuracy at the end. During the training process, the number of iterations in a frame is set to 20 (K=20) so that there are 100 frames in an epoch. To select the optimal parameters for the two baselines, we exhaustively use all possible combinations of parameters to solve the problem scenarios used in the evaluation phase and choose the one that obtains the highest relative accuracy after 1,000 iterations. Specifically, in the GD method, the possible learning rates of gradient descent and gradient ascent are 0.001, 0.01, 0.1, 0.5 or 1. In the GDM method, the possible learning rates of gradient descent with momentum and gradient ascent with momentum are 0.001, 0.01, 0.1, 0.5 or 1, while the possible momentum factors varies from 0.1 to 1 with a step size of 0.1. Finally, the optimal learning rates of gradient descent and gradient ascent in the GD method are 0.001 and 0.5, respectively. For the GDM method, the optimal learning rate and the momentum factor for gradient descent with momentum is 0.001 and 0.7, respectively, while the optimal learning rate and the momentum factor for gradient ascent with momentum are 0.1 and 0.5, respectively.

Fig. 2a shows the relative accuracy obtained by the CLSTMs and the baselines in the first 100 iterations during the evaluation process, where the solid curves present the mean relative accuracy and the coloured shadows show the mean relative accuracy plus and minus one standard deviation of accuracy. We can observe that the mean relative accuracy for the CLSTMs achieves 0.90 after 22 iterations and reaches to 0.98 after 67 iterations, while the baselines still show obvious fluctuation. Furthermore, the narrower shadow region for the CLSTMs indicates that the standard deviation for the CLSTMs is lower than the baselines after the same number of iterations. This confirms that the CLSTMs is much quicker in producing accurate and stable solutions than the conventional gradient-based methods. Fig. 2b shows the mean relative accuracy obtained by the CLSTMs and the baselines until 1,000 iterations during the evaluation process. We can see that the mean relative accuracy of the two baselines gradually increases to 1, which confirms that the gradient-based methods are able to find the optimal solutions after enough iterations. Furthermore, we can see that the mean relative accuracy of CLSTMs converges at 0.99 and the change of the CLSTMs results from 100 to 1,000 iterations becomes negligible. Although it is possible to improve the solution quality with more iterations in the evaluation process, the improvement will be quite marginal.

Fig. 2c and 2d present the relative accuracy obtained by the CLSTMs and the baselines in the first 0.5 and 2 seconds

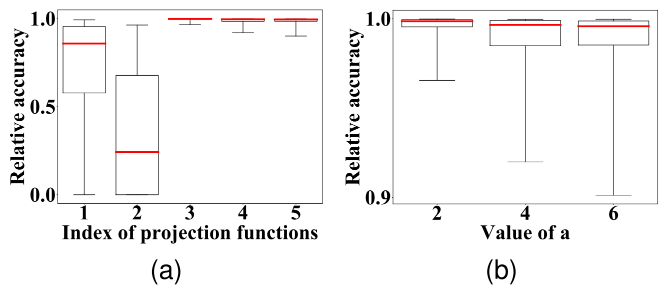

Fig. 4. The impact of (a) projection functions (b) value of a on the relative accuracy

of CPU time during the evaluation, respectively, where the solid curves correspond to the mean relative accuracy and the coloured shadows to the mean relative accuracy plus and minus one standard deviation of the relative accuracy. Note that the y-axis values between 0.1 and 0.5 are omitted in Fig. 2d so that the difference between the three curves can be made clear. We can see that the mean relative accuracy for the CLSTMs is higher than those of the two baselines after consuming the same amount of CPU time and achieves the accuracy of 0.99 earlier than the two baselines with respect to the CPU time consumption from Fig. 2c and Fig. 2d. Moreover, the observation can be made from Fig. 2c that the standard deviation for the CLSTMs is lower than the two baselines after 0.5 seconds of CPU time. Therefore, one can confirm from Fig. 2 that using the CLSTMs require less the CPU time and iterations to generate accuracy solutions when compared with the conventional gradient-based methods.

Fig. 3a shows the complementary cumulative distribution function (CCDF) of the relative accuracy of solutions generated by the CLSTMs after 1,000 iterations. From the figure, we can make the observation that with 95% of probability, the relative accuracy is higher than 0.96 and the mean relative accuracy is 0.99, which are excellent for general applications.

Clearly, these numerical results validate that the CLSTMs can find a near-optimal solution quickly (e.g., achieving 0.98 for the mean relative accuracy after 67 iterations) and the relative accuracy of the solutions found by the CLSTMs after enough iterations is practically equal to 100% (e.g., achieving the mean relative accuracy of 0.99 after 1,000 iterations).

2) *Impact of projection functions*

In this experiment, we study the impact of different projection functions. Specifically, we train five CLSTMs with the same training procedure except that the projection functions used by the CLSTMs are different. Furthermore, the five trained CLSTMs are evaluated with the same projection function as used during the training. These five projection functions that are examined here include: $\psi(\lambda) = |\lambda|$, $\psi(\lambda) = \frac{1}{2}(\sqrt{\lambda^2 + 0.25} + \lambda)$, and another three functions defined in (11) with a equal to 2, 4 and 6, respectively. Note that we index these projection functions by 1 to 5 in this order.

Fig. 4a shows the summary of the relative accuracy obtained by the five trained CLSTMs after 1,000 iterations. Specifically, the lower whisker, the bottom of the box, the red horizontal line, the top of the box and the upper whisker represent the 5th, 25th, 50th, 75th and 95th percentile of the relative accuracy, respectively. From this figure, we can see that the CLSTMs trained with the first and second projection functions (i.e., $\psi(\lambda) = |\lambda|$ and $\psi(\lambda) = \frac{1}{2}(\sqrt{\lambda^2 + 0.25} + \lambda)$) fail to find the optimal solutions. This is expected so because that the first projection function does not satisfy Requirement 2 since the first function is not differentiable at 0 and the second projection function violates Requirement 3 that the first derivative of the projection function should be a non-zero constant when $\lambda \rightarrow -\infty$.

Fig. 4b demonstrates the summary of the relative accuracy obtained by the three CLSTMs trained with three projection functions defined in (11) with a equal to 2, 4 and 6, respectively, after 1,000 iterations. Specifically, the whiskers, the boundaries of the box and the red line indicate the 5th, 25th, 50th, 75th and 95th percentile of the relative accuracy from the bottom upward. We can see that the 5th percentile of the relative accuracy for three different a values are all higher than 0.9. This confirms that the projection functions satisfying Requirements 1, 2 and 3 can be used by the CLSTMs to generate accurate solutions. Furthermore, the 5th, 25th and 50th percentile of the relative accuracy with $a = 2$ is higher than those with $a = 4$ and 6, which implies that the CLSTMs trained with $a = 2$ can generally produce more accurate solutions.

3) *Robustness: Impact of K and M*

In this experiment, we firstly show the impact of the parameter K in (8) and (9) on the performance of CLSTMs. Specifically, we select six different K values from 10 to 60 with a step size of 10 and each K value is used to train five CLSTMs with randomly initialized weights for neural networks. Then, we use five datasets that are generated from distributions different from those used during training to evaluate the trained CLSTMs. Specifically, for each utility function (19), the parameters μ_n are randomly selected from the uniform distribution in the range of $[0.001, M)$. To generate the training dataset, M is set to 1. To produce the six evaluation datasets, the value of M is increased by 0, 20, 40, 60, 80 and 100%, respectively.

Fig. 5a presents the average, maximum and minimum of the mean relative accuracy for five CLSTMs using the same K value, while Fig. 5b shows the average, maximum and minimum of the standard deviation of the relative accuracy for five CLSTMs using the same K value. We can see from the figure that the mean relative accuracy improves and the standard deviation decreases with the increase of K from 10 to 20. This is so because that the loss function with a larger value of K contains a longer future impact of each iteration. Thus, the CLSTMs learning to minimize such loss function can find better update step sizes to minimize the objective functions. On the other hand, we can further observe that when the value of K increases from 20 to 60, the mean relative accuracy and standard derivation only show slight fluctuation. This confirms that training the CLSTMs with various K values (i.e., from 20 to 60) can obtain similarly good performance. Based on these observations from Fig. 5a and Fig. 5b, we find that the CLSTMs is robust to various K values, whereas we still suggest to avoid small K values since they may lead to

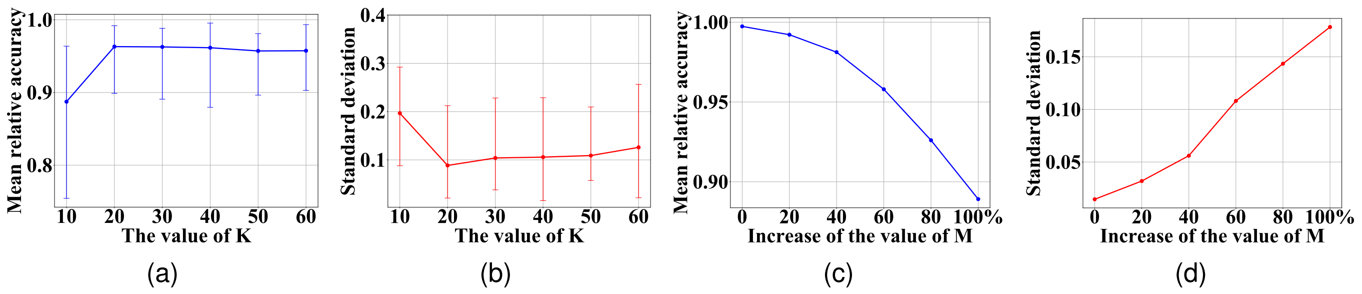

Fig. 5. The impact of the value of K on (a) the mean relative accuracy and (b) the standard deviation where K is the parameter in (8) and (9); the impact of the increase of the value of M on (c) the mean relative accuracy and (d) the standard deviation where M is the upper bound of the range for the uniform distribution that used to randomly select the parameters μ_n for each utility function

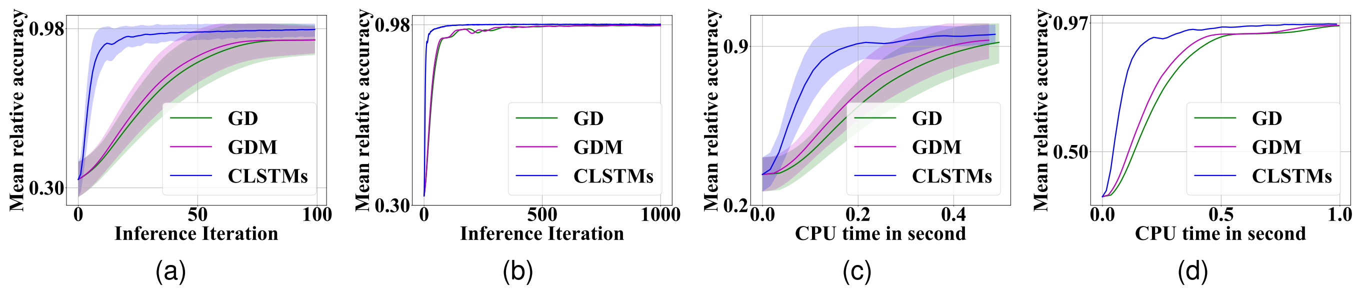

Fig. 6. The mean relative accuracy over (a) 100 iterations, (b) 1,000 iterations, (c)(d) CPU time in seconds.

the performance degradation.

Fig. 5c and 5d show the mean relative accuracy and the standard deviation of the relative accuracy for six different datasets for the CLSTMs evaluation, respectively. We observe that the mean relative accuracy gradually decreases from 0.99 to 0.89 in Fig. 5c and the standard deviation increases from 0.01 to 0.18 when the range for the uniform distribution is increased to 200%. This slight degradation of the relative accuracy is intuitively correct because when the range is increased by 100%, 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 the robustness of the trained CLSTMs as it still can find the optimal solutions for the problem scenarios even when their system parameters are drawn from distributions different from those used for training.

C. Results and Analysis with Sigmoid Utility Functions

In this subsection, we replace the convex utility functions used in the last subsection with non-convex utility functions and show the performance of the proposed CLSTMs. Specifically, for each job n, its utility function given the allocated CPU utilization x_n is given by

$$u_n(x_n) = \frac{1}{1 + e^{-\mu_n(x_n - R_n)}}, \tag{20}$$

where μ_n is a constant randomly selected from the uniform distribution in the range of $[0.001, 10)$ and R_n is the CPU utilization requirement of job n. Except the utility function, the values of parameters for the optimization problem scenarios (i.e., α, β, μ_n, the amount of required resource R_n and the available amount of CPU resource C) used during the training and the evaluation are the same as these used in the last subsection. Moreover, parameters for training the CLSTMs (i.e., w_k, $\hat{w}_k$, the learning rate and the projection function) also remain the same except that K is set to 40.

The optimal solutions are generated by the fmincon, which is guaranteed to converge to a local optimal where the infinity norm (maximum) of the gradient of the objective function is less than $1e^{-6}$.

Similarly, we select the optimal parameters for the two baselines, GD and GDM, by exhaustively evaluating potential parameter combinations and the sets of potential parameters are the same as in the last subsection. Finally, the optimal learning rate of gradient descent and gradient ascent in the GD are set to 0.01 and 0.1, respectively. In the GDM method, the optimal learning rate and momentum factor for gradient descent with momentum is 0.01 and 0.1, respectively, and for gradient ascent with momentum are 0.1 and 0.1, respectively.

Fig. 6a and 6b show the relative accuracy of the solutions generated by the CLSTMs and the baselines after the first 100 iterations and after a total of 1,000 iterations, respectively, where the solid curve corresponds to the mean relative accuracy and the coloured shadow to the mean relative accuracy plus and minus one standard deviation of the relative accuracy. Although three curves converge to a similar mean relative accuracy (0.98) in Fig. 6b, we can clearly observe that the mean relative accuracy for CLSTMs improves much faster than the two baselines and achieves the accuracy of 0.97 after 68 iterations in Fig. 6a. Furthermore, the observation can be made from Fig. 6a that the shadow region for the CLSTMs is clearly narrower than the two baselines after 20 iterations. Thus, even when the utility functions are non-convex, the CLSTMs

TABLE I
THE IMPACT OF THE NUMBER OF VARIABLES ON THE RELATIVE ACCURACY AFTER 10 ITERATIONS AND THE NUMBER OF ITERATIONS/THE CPU TIME CONSUMED WHEN THE MEAN RELATIVE ACCURACY ACHIEVES 0.97

Number of jobs in each problem scenario	Number of variables	Number of constraints	Mean relative accuracy / standard deviation after 10 iterations	Number of iterations	CPU time (second)
10	10	21	0.90 / 0.09	68	1.2
50	50	101	0.94 / 0.04	66	1.2
70	70	141	0.92 / 0.06	62	1.0
90	90	181	0.92 / 0.05	70	1.2
100	100	201	0.96 / 0.03	66	1.2

require fewer iterations than the conventional gradient-based methods to produce accurate and stable solutions.

To consider CPU time consumption, Fig. 6c and Fig. 6d shows the mean relative accuracy obtained by the CLSTMs and the baselines in the first 0.5 and 1 seconds of CPU time during the evaluation, respectively, where the solid curve corresponds to the mean relative accuracy and the coloured shadow to the mean relative accuracy plus and minus one standard deviation of the relative accuracy. We can observe that after consuming the same amount of CPU time, the CLSTMs always outperform the two baselines in the mean relative accuracy and the standard deviation. In other words, the CLSTMs generate excellent solutions faster than two baselines with respect to the CPU time.

Fig. 3b portrays the CCDF of the relative accuracy of solutions generated by the CLSTMs after 1,000 iterations. We can see that with 95% probability, the relative accuracy is larger than 0.91 and the mean relative accuracy is 0.98, which are lower than the corresponding results of 0.96 and 0.99 in Fig. 3a, respectively. The reason for this slight degradation is that the constrained optimization problem (17) with non-convex utility functions is much difficult to solve, thus requiring more iterations and CPU time than those with convex utility functions.

Nevertheless, these results validate the performance of CLSTMs, which can find the optimal solutions for the non-convex utility functions quickly.

D. Results and Analysis for Large Numbers of Variables and Constraints

To demonstrate the impact of large numbers of variables on performance, the proposed CLSTMs is used to find the solutions for the problem scenarios with the different numbers of jobs. Specifically, we change the number of jobs in each problem scenarios to 50, 70, 90, and 100, respectively. Consequently, the number of decision variables becomes 50, 70, 90, and 100, and the number of constraints is 101, 141, 181, and 201, respectively. Except the number of jobs, the utility function and all other parameters are the same with these used in the experiment with sigmoid utility functions.

As shown in Table. I, the number of variables and constraints increases when there are more jobs in the problem scenarios. Furthermore, we can see that the mean relative accuracy obtained after 10 iterations in all settings are higher than 0.9 and the standard deviation is lower than 0.1. Furthermore, we can observe from Table. I that the mean relative accuracy in all settings achieve 0.97 within 70 iterations or 1.2 seconds of the CPU time.

Therefore, the results clearly show that the proposed CLSTMs performs consistently well over a wide range of problem sizes. These results have led us to expect that the CLSTMs method can quickly generate the optimal or near-optimal solutions even when the numbers of variables and constraints further increase.

VI. RELATED WORK

A. Learning to optimize

The term 'learning to optimize' can generally refer to solve optimization problems by using learning techniques, such as the supervised learning [8]–[10] and the deep reinforcement learning [25]–[27]. A possible approach is to predict optimal solutions to these optimization problems using the supervised learning techniques. For instance, authors in [8] propose to use a deep neural network to approximate the unknown nonlinear mapping between the parameters of signal processing problems and the optimal solutions. Another approach is to apply learning techniques as a specific component in an optimization algorithm. For example, Zhang *et al.* [10] propose to use supervised learning techniques to learn the optimal pruning policy in the branch-and-bound algorithm, while authors in [24] use deep belief networks for reducing the number of variables in the optimization problems so that the size of the considered problem is reduced.

Besides the aforementioned two approaches, learning techniques also can be applied for updating variables in optimization problems by iterations. For example, Ke *et al.* [25] propose a reinforcement learning based optimization framework, where a deep neural network is employed as a policy and this policy is used to generate the update step sizes for variables. Because this type of approach can adopt different learning techniques that optimal solutions are not required for training, it can be more efficient and easy-to-implement when generating the optimal solutions is difficult.

B. Learning to optimize by RNNs

In recent decades, the recurrent neural networks (RNNs) become more attractive for designing novel 'learn to optimize' algorithms. Lv *et al.* [28] design the RNNprop model, in which the algorithms, Adam [15] and RMSprop [16], are embedded, and Wichrowska *et al.* [29] propose a hierarchical RNN architecture to better scale to larger problems and transfer to new tasks. Although these RNNs with various architectures haves been proposed, LSTM networks are still widely used for designing 'learn to optimize' algorithms because of its simplicity and robust performance. For instance, authors in [17] apply the LSTM networks to learn to map a given gradient to an update step size for variables and demonstrate that the trained LSTM network can outperform conventional gradient-based optimization algorithms on various unconstrained optimization problems. Moreover, LSTM networks are also used to solve other types of optimization problems, such as the Bayesian swarm optimization problem [30], the black-box optimization problems [31] and the minimax optimization problems [32]–[34].

However, none of the aforementioned research considers using LSTM networks to solve constrained optimization problems. It is important to note that recurrent neural networks used in [35]–[38] for solving optimization problems are a specific type of neural networks, which uses special analog components, such as integrators and amplifiers. In contrast, our proposed method specifically uses LSTM networks to capture the temporal evolution of the iterative steps in searching optimal solutions, thus making our work totally different from the existing techniques.

VII. Conclusion

In this paper, we have proposed the CLSTMs to solve nonconvex, constrained optimization problems. In developing the solution technique, a projection function is introduced to resolve an issue of keeping the Lagrangian multiplier positive and avoid numerical difficulties. In addition, we have analyzed the impact of the projection function on the optimal solutions and the relationship between minimizing the loss functions and solving the original constrained optimization problem with special parameter settings. Furthermore, we have formulated a resource-allocation problem and applied the new CLSTMs to solve it by using the practical datasets from Alibaba. Extensive experiments have been conducted to validate and study the performance of the proposed CLSTMs. By considering 1,000 parameter scenarios of the resource-allocation problem with convex utility functions, our numerical results have shown that (1) the trained CLSTMs can find the near-optimal and stable solutions in few iterations (e.g., achieving 98% mean relative accuracy after 67 iterations), (2) the proposed approach is very robust to parameter changes as the trained CLSTMs can produce excellent solutions for problems with system parameters drawn from distributions different from those used in the training process, and (3) the proposed criteria for selecting the projection functions to ensure non-negative Lagrangian multipler(s) can help the CLSTMs obtain optimal solutions. Moreover, we consider another 1,000 scenarios with non-convex utility functions and the numerical results validate the trained CLSTMs can also generate the near optimal and stable solutions with less iterations (e.g., achieving 97% mean relative accuracy after 68 iterations) and CPU times compared with the conventional gradient-based methods. Moreover, the solutions to non-convex optimization problems obtained from the CLSTM achieve 90% or better of the true optimum after 11 iterations, which corresponds to only 19% and 17% of the number of iterations required by the GDM and GD methods, respectively, to produce the same results. The CLSTM performance also represents a reduction of CPU time consumption by 33% and 36% when compared with the GDM and GD, respectively.

In the future, we plan to explore extension of the proposed method to the mixed integer nonlinear programming (MINLP) problems. This is challenging because of the following two reasons. Firstly, although it is possible to apply the CLSTMs to solve MINLP problems if gradients can be properly estimated, it is unclear that how close the generated solutions will be to the optimum. Secondly, because the CLSTMs may converge to non-integer solutions, further processing in terms of discretization or rounding to integers may be needed. Furthermore, the inseparable variables will also need to be considered. Since the dependence among these variables is invisible to the CLSTMs, the CLSTMs may not be able to generate appropriate update step sizes for these variables, and thus it may require additional structures and training procedures to help the CLSTMs learn the dependences among these variables.

References

[1] Z. Chen, K. K. Leung, S. Wang, L. Tassiulas and K. Chan, 'Robust Solutions to Constrained Optimization Problems by LSTM Networks,' MILCOM 2021 - 2021 IEEE Military Communications Conference (MILCOM), pp. 503-508, 2021.

[2] H. Halabian, 'Distributed Resource Allocation Optimization in 5G Virtualized Networks', IEEE Journal on Selected Areas in Communications, vol. 37, no. 3, pp. 627–642, 2019.

[3] Y. Xu, H. Sun, and Y. Ye, 'Distributed Resource Allocation for SWIPT-Based Cognitive Ad-Hoc Networks', IEEE Transactions on Cognitive Communications and Networking, vol. 7, no. 4, pp. 1320–1332, 2021.

[4] J. Zhao, Q. Li, Y. Gong, and K. Zhang, 'Computation Offloading and Resource Allocation For Cloud Assisted Mobile Edge Computing in Vehicular Networks', IEEE Transactions on Vehicular Technology, vol. 68, no. 8, pp. 7944–7956, 2019.

[5] J. Du, C. Jiang, A. Benslimane, S. Guo, and Y. Ren, 'SDN-Based Resource Allocation in Edge and Cloud Computing Systems: An Evolutionary Stackelberg Differential Game Approach', IEEE/ACM Transactions on Networking, pp. 1–16, 2022.

[6] Q. Qin, K. Poularakis, G. Iosifidis, and L. Tassiulas, 'SDN Controller Placement at the Edge: Optimizing Delay and Overheads', in IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, 2018, pp. 684–692.

[7] Y. Xie et al., 'Virtualized Network Function Forwarding Graph Placing in SDN and NFV-Enabled IoT Networks: A Graph Neural Network Assisted Deep Reinforcement Learning Method', IEEE Transactions on Network and Service Management, vol. 19, no. 1, pp. 524–537, 2022, doi: 10.1109/TNSM.2021.3123460.

[8] H. Sun, et al, 'Learning to Optimize: Training Deep Neural Networks for Interference Management,' IEEE Trans. on Signal Processing, vol. 66, no. 20, pp. 5438-5453, 2018.

[9] F. Fioretto, T. W.K. Mak, and P. Van Hentenryck, '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.

[10] Z. Zhang and M. Tao, 'Learning Based Branch-and-Bound for Non-Convex Complex Modulus Constrained Problems with Applications in Wireless Communications', IEEE Transactions on Wireless Communications, 2021.

[11] Z. Gao, et al. 'Learning to Optimize on SPD Manifolds.' Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 7700-7709, 2020.

[12] C. Sun, D.-K. Kim, and J. P. How, 'FISAR: Forward Invariant Safe Reinforcement Learning with a Deep Neural Network-Based Optimizer', in 2021 IEEE International Conference on Robotics and Automation (ICRA), pp. 10617–10624, 2021.

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

[14] S. Boyd, S. P. Boyd, and L. Vandenberghe, Convex optimization. Cambridge university press, 2004.

[15] D. P. Kingma and J. L. Ba, 'Adam: A Method for Stochastic Optimization,' ICLR (Poster). 2015.

[16] T. Tieleman and G. Hinton, 'Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude'. COURSERA: Neural Networks for Machine Learning, 4(2), 2012.

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

[18] G. Tychogiorgos, A. Gkelias and K.K. Leung, 'A Non-Convex Distributed Optimization Framework and its Application to Wireless Ad-hoc Networks,' IEEE Trans. on Wireless Communications, Vol. 12, pp. 4286 – 4296, September 2013.

[19] D. T. Ngo, C. Tellambura and H. H. Nguyen, 'Resource Allocation for OFDM-Based Cognitive Radio Multicast Networks,' Proceeding of 2009 IEEE Wireless Communications and Networking Conference, pp. 1-6, 2009.

[20] A. Ribeiro and G. B. Giannakis, 'Separation Principles in Wireless Networking,' in IEEE Trans. on Information Theory, vol. 56, no. 9, pp. 4488-4505, Sept. 2010.

[21] S. Nazemi, K.K. Leung and A. Swami, 'Distributed Optimisation Framework for In-network Data Processing,' IEEE/ACM Trans. on Networking, Vol. 27, No. 6, pp. 2432-2443, Dec. 2019.

[22] B. T. Polyak, 'Some methods of speeding up the convergence of iteration methods', Ussr computational mathematics and mathematical physics, vol. 4, no. 5, pp. 1–17, 1964.

[23] E. Ghadimi, H. R. Feyzmahdavian, and M. Johansson, 'Global convergence of the heavy-ball method for convex optimization', in 2015 European control conference (ECC), 2015, pp. 310–315.

[24] L. Liu, Y. Cheng, L. Cai, S. Zhou, and Z. Niu, 'Deep learning based optimization in wireless network', in 2017 IEEE international conference on communications (ICC), 2017, pp. 1–6.

[25] K. Li and M. Jitendra, 'Learning to optimize,' Proceedings of 5th International Conference on Learning Representations (ICLR), 2017.

[26] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, 'Learning Combinatorial Optimization Algorithms over Graphs', in Advances in Neural Information Processing Systems, vol. 30, 2017.

[27] K. Yonekura and H. Hattori, 'Framework for design optimization using deep reinforcement learning', Struct Multidisc Optim, vol. 60, no. 4, pp. 1709–1713, Oct. 2019.

[28] K. Lv, S. Jiang, and J. Li, 'Learning gradient descent: Better generalization and longer horizons', in International Conference on Machine Learning, pp. 2247–2255, 2017.

[29] O. Wichrowska et al., 'Learned Optimizers that Scale and Generalize', in Proceedings of the 34th International Conference on Machine Learning, vol. 70, pp. 3751–3760, Aug. 2017.

[30] Y. Cao, T. Chen, Z. Wang, and Y. Shen, 'Learning to optimize in swarms,' Advances in Neural Information Processing Systems, pp. 15018–15028, 2019.

[31] Y. Chen et al., 'Learning to Learn without Gradient Descent by Gradient Descent', in Proceedings of the 34th International Conference on Machine Learning, vol. 70, pp. 748–756, Aug. 2017.

[32] Y. Xiong, and H. Cho-Jui, 'Improved Adversarial Training via Learned Optimizer,' European Conference on Computer Vision, pp. 85-100. Springer, Cham, 2020.

[33] H. Jiang, Z. Chen, Y. Shi, B. Dai, and T. Zhao, 'Learning to Defend by Learning to Attack', in Proceedings of The 24th International Conference on Artificial Intelligence and Statistics, vol. 130, pp. 577–585, Apr. 2021.

[34] J. Shen, X. Chen, H. Heaton, T. Chen, J. Liu, W. Yin, and Z. Wang, 'Learning a minimax optimizer: A pilot study,' Proceeding of 9th ICLR, 2021.

[35] X.-B. Liang and J. Wang, 'A recurrent neural network for nonlinear optimization with a continuously differentiable objective function and bound constraints', IEEE Transactions on Neural Networks, vol. 11, no. 6, pp. 1251–1262, 2000.

[36] Y. Xia and J. Wang, 'A general projection neural network for solving monotone variational inequalities and related optimization problems', IEEE Transactions on Neural Networks, vol. 15, no. 2, pp. 318–328, 2004.

[37] M. Mestari, M. Benzirar, N. Saber, and M. Khouil, 'Solving nonlinear equality constrained multiobjective optimization problems using neural networks', IEEE Transactions on Neural Networks and Learning Systems, vol. 26, no. 10, pp. 2500–2520, 2015.

[38] Y. Xia, J. Wang, Z. Lu, and L. Huang, 'Two Recurrent Neural Networks With Reduced Model Complexity for Constrained l_1-Norm Optimization', IEEE Transactions on Neural Networks and Learning Systems, pp. 1–13, 2022.