IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 19 May 2023, accepted 20 June 2023, date of publication 27 June 2023, date of current version 30 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3289884

RESEARCH ARTICLE

Leveraging Sparse Auto-Encoding and Dynamic Learning Rate for Efficient Cloud Workloads Prediction

DALAL ALQAHTANI
Department of Electrical and Computer Engineering, The George Washington University, Washington, DC 20052, USA
College of Computer Science and Information Systems, Najran University, Najran 61441, Saudi Arabia
e-mail: Alqahtani_d@gwu.edu

This work was supported by Najran University, Saudi Arabia.

ABSTRACT Cloud computing provides simple on-demand access to a centralized shared pool of computing resources. Performance and efficient utilization of cloud computing resources requires accurate prediction of cloud workload. This is a challenging problem due to cloud workloads' dynamic nature, making it difficult to predict. Here we leverage deep learning which can with proper training provide accurate bases for the prediction of data center workload. Deep Learning (DL) models, however, are challenging to train. One challenge is the vast number of hyperparameters needed to define and tune. The performance of a neural network model can be significantly improved by optimizing these hyperparameters. We recognize two of the essential issues to predict data center workloads using deep learning efficiently. First is the high dimensionality which requires removing superfluous information via some form of dimension reduction. Secondly, is the learning rate. Small learning rates can make the time for training very excessive while long learning rates can miss optimal solutions. Our approach is therefore dual-pronged. First, we use Sparse Auto-Encoder (SAE) to retrieve the essential workloads representations from the original high-dimensional historical cloud workloads data. Secondly, we use Gated Recurrent Unit with a Step-Wise Scheduler for the Learning Decay (GRU-SWSLD). The proposed system is demonstrated with data from Google cluster workload traces to predict Central Processing Unit (CPU) usage using the data center's workload traces at several consecutive time steps. Our experimental results reveal that our proposed methodology provides a better tradeoff between accuracy and training time when compared with other models.

INDEX TERMS Cloud computing, deep learning, sparse auto-encoder, gated recurrent unit, learning rate.

I. INTRODUCTION

Cloud computing enables end users to access remotely allocated services from any location at any time over the internet [1]. Large cloud service providers have built large-scale data centers connected to the internet to market their resources as services. Many businesses found it to be more cost-efficient to shift their local services to the cloud [2]. End users adapt their applications to the cloud and make requests with varying Quality of Service (QoS) needs. These are called Service Level Agreements (SLAs) [3].To maximize profit, cloud providers want to receive as many new requests as possible. However, cloud providers must meet QoS criteria per the agreed-upon SLA with end-users. As such, there is a need for effective resource provisioning systems to accomplish this objective [3]. To satisfy user requests, the allocation of resources should address SLAs such as availability, dependability, response time limit, and cost cap [4]. The cloud resources needed to handle user requests depend on the incoming workloads. To meet this demand, there are two options: either long-term reservations or on-demand resource provisioning. If long-term reservations are used, there may be many unused cloud resources, while with on-demand provisioning, only a few requests can be handled at once. In addition, users often have unpredictable access to cloud resources and the workload can change rapidly [5].

The associate editor coordinating the review of this manuscript and approving it for publication was Sajid Ali.

When there is inconsistency in the amount of workload to be done, it can lead to problems of not having enough or having too many resources allocated to complete the tasks. This can result in both wasting time and resources. Over-provisioning happens when there are more tasks assigned to a particular resource than it can handle while under-provisioning happens when there are idle resources that are not being utilized efficiently [6]. Based on the evaluation report, the Google cluster recorded an average 20 % CPU utilization and 40 % memory utilization. Alibaba cloud data centers recorded an average CPU utilization of their entire cluster ranging from 5 % to 80 % percent with considerable variances during the day [7]. Predicting workloads accurately in data centers can be challenging due to the significant variability in task trends. Cloud data centers have high-dimensional workload traces with lots of information, including machine specifications. It's important to extract essential information from these traces for training models, but these cloud traces have noise and redundancy making it difficult to predict workloads accurately [8]. To address these challenges, we need to learn the patterns and relationships of workloads over time and create efficient and accurate prediction algorithms that can adapt to workload fluctuations. Additionally, we can process the cloud data center traces to extract necessary data while maintaining prediction accuracy.

Host load prediction has been a topic of interest for researchers for a long time due to the potential for financial benefits. Many previous studies have focused on traditional high-performance computing (HPC) or grid systems [9], [10], [11]. When analyzing the workloads of cloud and grid systems, it was found that the amount of fluctuation or noise in the cloud was much higher, up to 20 times more than in a grid system [12]. Therefore, forecasting the host load in a cloud is much more complex and difficult than in grids.

Currently, the most commonly used methods for predicting cloud workloads are autoregressive (AR) and traditional neural networks. [13], [14], [15], [16], [17]. However, these methods are not always effective for large-scale public cloud data centers, where the workload patterns are more complex and difficult to capture. Traditional neural networks, which generally consist of shallow networks with only 1 or 2 hidden layers, may be helpful for tasks that involve repetitive patterns, such as those found in small data centers or grids. But to achieve better accuracy, more advanced neural networks need to be applied to capture the correlation between complex workloads in data centers.

Deep learning approaches based on Recurrent Neural Networks (RNNs) have been effective in modeling cloud workloads due to their superior capacity to analyze sequential data [18]. RNNs are especially useful for forecasting random workloads, but classical RNNs are unable to effectively capture the dependency information of long-term sequences because of the vanishing gradient problem. The vanishing gradient problem occurs when an RNN is used to train long-sequence data, and the trained model's weights eventually fail because the network cannot be updated [19]. To address the vanishing gradient problem, variants of RNN, such as the Long Short-Term Memory Network (LSTM) [20] and Gated Recurrent Unit (GRU) [21], have been proposed. LSTM is more complex than GRU since it has more sets of gates and requires more memory, making it slower than GRU.

Nevertheless, training an RNN-based system on high-dimensional cloud workload traces can be time-consuming, so we need to use dimensionality reduction techniques like Principal Component Analysis (PCA) [22] and auto-encoder [18] to compress the data. PCA is primarily a linear transformation while auto-encoders can model complex nonlinear functions. Since cloud workload traces are nonlinear, auto-encoders are a better option. Of the five types of auto-encoders [30], Sparse Auto-Encoder (SAE) is the best for cloud workload traces due to its ability to provide a noise-resistant feature representation that selectively activates regions in the input data.

In the field of deep learning, optimizing the performance of a model is a critical task. The selection of hyperparameters, or parameters that are established before the training process starts, is one of the crucial elements that affect how well deep learning models perform. The learning rate is a hyperparameter that controls how big of a step the optimizer algorithm makes when training to reduce the error function. The learning rate is the most crucial hyperparameter, despite the fact that there are several to take into account [24]. The optimization procedure may be ineffective or perhaps fail entirely if the learning rate is improperly configured.

This paper's primary contributions are as follows:

- A Sparse Auto-Encoder neural network is constructed to learn the representation from cloud workloads traces and perform dimensional reduction by training the network to disregard signal noise. Along with the reduction side, a reconstructing side is learned, in which the Sparse Auto-Encoder attempts to build a representation as close to the raw data as possible from the reduced encoding. This enables Sparse Auto-Encoders to learn essential features in the data.
- A step-wise scheduler for the learning decay is integrated with the GRU model to improve prediction performance over multivariate intervals of time and different sliding windows.
- We apply the proposed methodology to actual cloud center workload traces from Google's cluster to verify its performance.

The remainder of our work is structured as follows: Section II introduces the fundamentals of Sparse Auto-Encoder (SA), stochastic gradient descent and the learning rate, and the Gated Recurrent Unit (GRU) model. Section III examines related work on cloud workloads prediction. The system model is described in Section IV, along with the proposed model. Section V examines the proposed methodologies through simulated studies using real-world cloud workload traces. Finally, Section VI draws this paper's conclusion.

II. PRELIMINARIES

In this section, we'll look at how to solve the problem of high dimensionality in cloud workloads prediction, extract essential representation from the CPU usage at different scales, and optimize the deep learning model. We will introduce the three techniques that will be used in the paper: Sparse Auto-Encoder (SA), stochastic gradient descent and the learning rate, and Gated Recurrent Unit (GRU).

A. SPARSE AUTO-ENCODER

The Sparse Auto-Encoder (SAE) network is an unsupervised learning algorithm that learns features from n unlabeled data $\{X^1, X^2, X^3 \ldots, X^n\}$ where each $X^i \in \mathbb{R}^{\mathbb{N}}$. The Sparse Auto-Encoder network is made up of three layers: the input layer, the hidden layer, and the output layer, and the dimensionality reduction in the SAE process consists of two steps: encoding and decoding. The encoding stage, as shown in equation (1), converts the input data x into the representations of the hidden units where $x = (x_1, x_2, x_3, \ldots, x_i)$ is the high-dimensional input data vector.

$$X = g(Wx + b) \tag{1}$$

The encoding stage, as shown in equation (2), maps the hidden units' representations into the reconstructed data $z = (\hat{x_1}, \hat{x_2}, \hat{x_3}, \hat{x_i})$ where $\hat{x_i}$ is the low-dimensional output of the ith sample in the dataset from the hidden layer.

$$\hat{X} = g(\hat{W}z + \hat{b}) \tag{2}$$

In both equations (1) and (2) g is the activation function for the hidden unit, W and $\hat{W}$ are the weight matrices, and b and $\hat{b}$ are the encoder and decoder bias vectors, respectively. The Sparse Auto-Encoder uses back-propagation to set the target values to be equal to the inputs $\hat{x^i} \approx x^i$.

The Sparse Auto-Encoder will still find interesting patterns in the data even if there are many hidden units if we apply a sparsity constraint on them. In the Sparse Auto-Encoder, the ${a_j}^2$ indicates that hidden unit j is activated. ${a_j}^2(x_i)$ expresses the hidden unit's activation when the network gets a particular input x. Equation (3) is to estimate each hidden neuron's average activation value. m is donated for the number of samples in the dataset, (x_i) is donated for the input vector for the i^{th} sample in the dataset.

$$\hat{p_j} = \frac{1}{m}\sum_{i=1}^{m}\lceil {a_j}^2(x_i)\rceil \tag{3}$$

$\hat{p_j}$ is enforced by the constraint p, where p is a parameter for sparsity, usually a small value near zero. We can use equation (4) to calculate the sparsity cost:

$$\sum_{j=1}^{s^2} p\log\frac{p}{\hat{p_j}} + (1-p)\log\frac{1-p}{1-\hat{p_j}} \tag{4}$$

s^2 is the number of neurons in the hidden layer, and the index (j) is summing over the hidden units in the Sparse Auto-Encoder. (2) is based on KL divergence [26] and can also be written as:

$$\sum_{j=1}^{s^2} KL(p \parallel \hat{p_j}) = \sum_{j=1}^{s^2} p\log\frac{p}{\hat{p_j}} + (1-p)\log\frac{1-p}{1-\hat{p_j}} \tag{5}$$

KL divergence is a standard function for measuring how different two distributions are, and the function has the property that $KL(p \parallel \hat{p_j}) = 0$ if $\hat{p_j} = p$, and otherwise it increases monotonically as $\hat{p_j}$ diverges from p. Thus, minimizing this penalty term has the effect of causing $\hat{p_j}$ to be close to p [50]. The overall cost function is now:

$$J_{sparse}(W, b) = J(W, b) + \beta\sum_{j=1}^{s^2} KL(p \parallel \hat{p_j}) \tag{6}$$

The term β controls the weight of the sparsity penalty term. The term $\hat{p_j}$ depends on W, b also because it is the average activation of hidden unit j, and the activation of a hidden unit depends on the parameters W, b.

B. STOCHASTIC GRADIENT DESCENT AND THE LEARNING RATE

The Stochastic Gradient Descent (SGD) algorithm is used to train deep-learning neural networks. Stochastic gradient descent is an optimization process that calculates the goal function, which is the sum of the loss functions for each example in the training dataset. Given a training dataset of n examples, we assume that $f_i(X)$ is the loss function with respect to the training example of index i, where X is the parameter vector [29]. Then we arrive at the objective function.

$$f_x = \frac{1}{n}\sum_{i=1}^{n} f_i(X) \tag{7}$$

The gradient of the objective function at X is computed as

$$\nabla f_x = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(X) \tag{8}$$

Stochastic gradient descent (SGD) reduces computational costs at each iteration. At each iteration of stochastic gradient descent, we uniformly sample an index $i \in 1, \ldots, n$ for data examples at random, and compute the gradient [29] $\nabla f_i(X)$ to update X

$$X \leftarrow X - \eta\nabla f_i(X) \tag{9}$$

where η is the learning rate. The learning rate is a hyperparameter that controls the speed at which the model learns. In particular, it regulates the amount of error that the weights of the trained model are updated with each time they are updated, such as at the end of each batch of training examples.

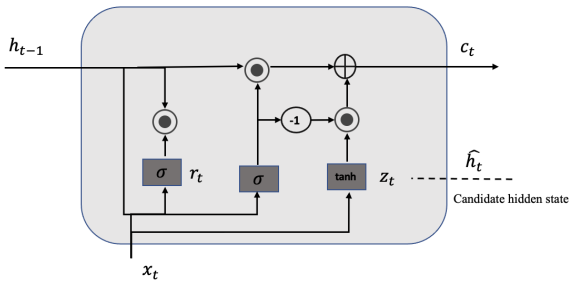


FIGURE 1. Basic structure of memory cell in GRU.

C. GATED RECURRENT UNIT (GRU)

Recurrent Neural Networks (RNNs) are commonly utilized for processing sequential data due to their ability to capture temporal relationships. Traditional RNNs, on the other hand, suffer from vanishing and exploding gradient issues, which restrict their usefulness in learning long-term dependencies.

The Gated Recurrent Unit (GRU) is a form of RNN that tackles these issues through the use of gating mechanisms. GRU introduces two major gates: r_t is a reset gate, and z_t is an update gate. These gates enable the model to selectively reset or update information within the hidden state, allowing for effective information flow and memory management.

Figure (1) depicts the single-cell structure of GRU. The following represents the calculation of the GRU cell:

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r) \tag{10}$$

The reset gate r_t plays a crucial role in determining what information from the previous hidden state h_{t-1} should be forgotten or reset when calculating the candidate hidden state $\hat{h}_t$. To compute r_t, it takes into account the current input x_t and the previous hidden state h_{t-1}. These two are combined through a weighted sum $W_{xr}x_t + W_{hr}h_{t-1}$, and a bias term b_r is added. This combined value then undergoes a sigmoid function σ, which transforms the values to a range between 0 and 1. By applying the r_t, the model can determine which specific aspects of the previous hidden state need to be focused on or disregarded during the current step of the computation.

$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z) \tag{11}$$

The update gate z_t controls how much old memory information h_{t-1} should be merged with the candidate hidden state $\hat{h}_t$ to generate the current hidden state h_t. It is computed similarly to the reset gate by taking the current input x_t and the prior memory content h_{t-1}, merging them through a weighted sum $W_{xz}x_t + W_{hz}h_{t-1}$, and adding a bias term b_z. The result is then run through a sigmoid function σ. The sigmoid function output determines the proportion of past memory material to keep, with values near 1 suggesting more retention and values near 0 indicating lesser retention.

$$\hat{h}_t = \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_n) \tag{12}$$

The candidate hidden state $\hat{h}_t$, indicates the information that might possibly be added to the current hidden state h_t. It is calculated by integrating the current input x_t with a modified version of the previous hidden state h_{t-1}. The change is accomplished by element-wise multiplication $\odot$ of the reset gate r_t and the previous hidden state h_{t-1}. The current input and the modified previously hidden state are merged using a weighted sum $W_{hh}(r_t \odot h_{t-1})$, and a bias term b_n is added. The output is then processed via the hyperbolic tangent function tanh to verify it is within the range of -1 to 1.

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \hat{h}_t \tag{13}$$

The hidden state h_t, indicates the recurrent neural network's current state. It is calculated by utilizing the update gate z_t to combine the previous hidden state h_{t-1} and the candidate hidden state $\hat{h}_t$. The element-wise multiplication $\odot$ and addition operations are used to accomplish this. The proportions of the previous hidden state and the candidate hidden state are determined by the update gate. If the update gate z_t is close to 1, it signifies that the model wants to retain most of the previous hidden state. Conversely, if z_t is close to 0, the model emphasizes the candidate's hidden state. The resulting hidden state h_t combines previous relevant information with new information supplied by the candidate hidden state $\hat{h}_t$.

III. RELATED WORK

Workload prediction in cloud services has gotten much attention in the research community, and many scholars have contributed significantly to solving this issue. This section will review traditional regression models, classic machine learning methods, and classic deep learning methods.

A. TRADITIONAL REGRESSION METHODS

Traditional regression or statical methods consist primarily of Moving Average (MA), Autoregressive (AR), AutoRegressive Integrated Moving Average (ARIMA), and Autoregressive Moving Average (ARMA). Hu et al. [32] overcome the cloud workload prediction problem to attain elasticity in cloud computing. For workload prediction, they employ some classic time series analytic approaches and a Kalman filter model. Moving Average (MA), Auto Regression (AR), and Auto Regression Integrated Moving Average (ARIMA) models are among the mathematical models used in their time series approach. Calherios et al. [33] also use ARIMA for workload prediction and evaluate their results regarding resource utilization improvements and efficiency. Saripalli et al. [34] developed a two-step method that included workload tracing (LT) and load forecasting (LF). The proposed cubic-spline LT outperforms other linear LTs based on the Moving Average in modeling the high variance of the loads. Kumar et al. [36] attempted to lower the cost of electricity by conducting a better workloads forecast. They utilized CPU, RAM, and network trace data gathered from Wikimedia Grid to evaluate the prediction

performance of the ARIMA, seasonal integrated ARMA, and fractionally integrated ARMA with the singular spectrum analysis method. They demonstrated that while the ARFIMA model suffers from lengthy calculations when the input size rises, increasing the input size may produce better forecasting outcomes. Liao et al. [38] predict VM CPU usage using traditional time series prediction methods such as autoregressive moving average, moving average, and auto-regressive as an ensemble approach.

These regression-based approaches have demonstrated their suitability for workload prediction. However, most of these approaches are only appropriate for workloads with clear trends, such as those found in grid or HPC systems, which have lower variance than large-scale cloud data centers. The high variance of modern cloud workloads makes these approaches challenging to depict the relationships between various components. While regression models are simple and based on linear workloads correlations [35], they cannot capture nonlinear variations in could workloads. As a result, more advanced learning methods, such as machine learning (ML) and deep learning (DL)-based techniques, have already been examined to capture the variations of cloud workloads successfully.

B. CLASSIC MACHINE LEARNING METHODS

Singh et al. [37] introduced a resource provisioning approach that relies on heterogeneous workloads' Quality of Service (QoS) requirements. The proposed solution allocated cloud resources to serve highly variable workloads by clustering, analyzing, and classifying workloads based on common patterns. Furthermore, the workload evaluation was carried out using a K-Means-based clustering method with weights assigned to quality characteristics in each cloud computing workload via QoS requirements. Their simulation results showed that the proposed solution effectively reduces the execution time and cost of cloud workloads while meeting the QoS requirements of customers. Rahmanian et al. [43] developed an ensemble cloud resource utilization predictive model based on the learning automata theory. The algorithm used two techniques for cloud resource forecasting: single window and multiple windows. Ghobaei-Arani et al. [44] developed a Markov decision process-based hybrid method. The Markov decision process is established when there are finite states and transitions between them. Bi et al. [45] introduced a workloads prediction approach that combines the Savitzky-Golay filter and wavelet decomposition with stochastic configuration networks. Liu et al. [46] proposed an adaptive workloads prediction strategy based on workloads classification, in which multiple prediction models can be allocated to the various workloads classifications.

Machine learning has limitations in its effectiveness. While conventional machine learning and classical artificial neural network techniques can extract non-linear characteristics of workloads without relying on restrictive assumptions, they often fall short in their ability to make accurate predictions. The primary reason for this is that these methods rely heavily on heuristic algorithms, which may require workloads with apparent regularity or trends to predict precisely. However, in real-world scenarios, workloads are often complex and unpredictable, and may not exhibit such clear trends or patterns. This lack of predictability can make it difficult for machine learning algorithms to accurately forecast future workloads.

C. CLASSIC DEEP LEARNING METHODS

Several efforts have been made to use deep learning approaches to predict data center resource use. As an example, Janardhanan et al. [40] demonstrated the weaknesses of traditional ARIMA models in predicting nonlinear patterns of extremely variant cloud workloads and propose the LSTM model to predict the cloud workloads of Google cluster machines. Compared to ARIMA, the memory cells that store temporal dependencies over long periods help LSTM make better forecasts. Duggan et al. [39] used RNN to forecast the cloud workloads of virtual machines on a host in a CloudSim-simulated environment. Back Propagation Through Time (BPTT) is used by RNN to optimize network weights. The authors find that RNN has better learning and higher convergence speeds than Back backpropagation Neural Networks, Random Walk Forecasting, and MA. Yang et al. [23] predicted host load on Google clusters using auto-encoder features as input to Echo State Networks (ESN). The auto-encoder represents the essential CPU load patterns with reduced dimensionality. However, the choice of initial weights and extracted features may limit the learned model's capacity to generalize. Gao et al. [42] use bidirectional long short-term memory to predict task failure, saving resource waste from failure recoveries while maintaining service availability and reliability. The model was validated on Google workloads traces and outperformed Hidden Semi-Markov Models, SVR, RNN, and LSTM regarding accuracy.

Shi et al. [25] and Kong et al. [26] demonstrated that RNNs could not effectively ensure excellent long-term prediction performance. This is due to the classic RNN's inability to solve the gradient vanishing problem during training. As the gap between the information and the predicted value increases, the RNN will lose its capacity to connect and interpret relevant data.

To overcome these open problems, we first develop a Sparse Auto-Encoder to improve learning efficiency in response to the high dimensionality of the workloads data. To better deal with the high variance of workloads patterns and increase the performance of the DL models for prediction, the step-wise scheduler for the learning decay is then utilized to optimize the GRU for attaining highly accurate forecasts of cloud workloads traces.

IV. INTEGRATION OF DEEP LEARNING FOR CLOUD WORKLOADS PREDICTION

A. SYSTEM MODEL DESIGNING

This section presents our system model for addressing the challenge of predicting highly dynamic cloud workloads.

Predicting cloud workload accurately is a difficult task due to the possibility of significant changes occurring in a short time span. An inaccurate workload prediction can result in unnecessary costs or violations of Service Level Agreements (SLAs). To address this challenge and improve the performance of deep learning models, we aim to develop a system model that accurately predicts future workloads while being efficient for service providers to use.

To optimize resource utilization, our proposed model combines workload forecasting with auto-scaling techniques to adjust the number of active or idle servers. However, traditional regression models or machine learning techniques are impractical for accurate prediction due to the close coupling of cloud workloads with time series. Therefore, we present a workload forecasting model for cloud data centers that reduce the differences between expected and actual workloads by considering the high dimensionality of workloads and improving the prediction of an optimizing deep learning model over a multivariate time. Our proposed system model, named GRU-SWSLD, consists of four components, as depicted in Figure (2). By leveraging this system model, service providers can make accurate workload predictions and optimize resource utilization, thereby reducing unnecessary costs and complying with SLAs.

To begin, the Real Time Resource Usage Monitor system in the cloud center records VM workloads and stores them in a repository in real-time. The historical workloads used in this research are taken from this monitoring system. The proposed forecasting model utilizes historical workloads data from the cloud data center as inputs. The workload traces are first preprocessed and compressed before being fed into the prediction processor module. The main function of this processor module is to convert the generated unsupervised multivariate time series workloads traces into supervised data that can be used as inputs in the predictor model. The processor module used to handle cloud workloads traces involves several steps, including:

- Extracting the historical workload's CPU usage.
- Normalizing data.
- Creating sliding windows and grouping data into multivariate time series.
- Utilizing Sparse Auto-Encoder (SAE) to reduce the dimensions of the data.

After pre-processing the workload traces, the output data is fed into the predictor model to generate the prediction using our proposed GRU-SWSLD model.

Finally, the prediction model's output is utilized to predict future workloads by the forecaster model. The predicted workloads are sent to the cloud resource manager, who will use the forecasts to identify the best resource provisioning techniques for autoscaling VMs in a cloud data center.

B. PROCESSOR MODEL

In cloud computing, the data associated with workloads comprises various measurements that reflect the system's operational condition. These measurements typically include CPU usage, memory utilization, RAM, bandwidth, disk space, disk I/O, and other relevant metrics. Of these, CPU usage is considered the most significant and restricted resource in computer systems and is often the primary bottleneck in cloud platforms [47]. This importance is reflected in the industry, where companies such as Google [47] and Amazon [49] view CPU utilization as a critical factor in enhancing appropriate provisioning in cloud data centers. Proper allocation and resource management decisions can be made at any moment by ensuring that the system has adequate CPU resources. In this paper, from the historical workloads traces, we are going to extract the CPU usage, which is donated as $\overrightarrow{X} = (x_1, x_2, x_3, \ldots, x_n)$ and $n \in \mathbb{R}$ and x_n is the CPU usage at time n.

One of the critical data preprocessing steps in deep learning is the normalization of historical workload traces. This step is particularly important due to the highly dynamic nature of the cloud data center environment, which leads to

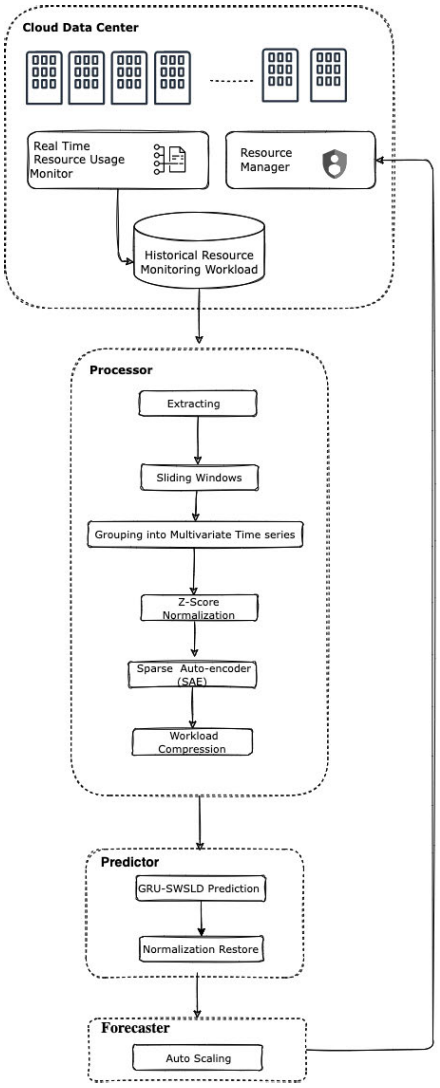


FIGURE 2. The architecture of the proposed GRU-SWSLD model.

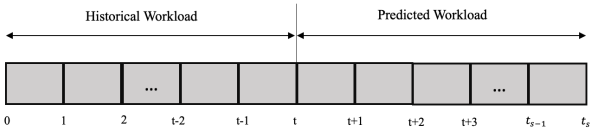


FIGURE 3. The scenario of formatting a predicting workloads traces.

significant variations in CPU usage over time. Additionally, normalization helps to speed up the convergence of deep learning algorithms by addressing the issue of differing feature ranges within the dataset. Without normalization, numerically larger distance values would be produced when algorithms compute distances among features.

To address these issues, we utilized Z-score normalization, which is one of the most commonly used normalization approaches in deep learning. Z-score normalization involves transforming the data so that it has a mean of zero and a standard deviation of one. This normalization technique effectively scales the data to a standard range and enables the deep learning algorithms to converge more quickly. Z-score normalization is expressed by the following equations:

$$Z = \frac{x - \mu(\overrightarrow{X})}{\sigma} \tag{14}$$

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \overrightarrow{X})^2}{N}} \tag{15}$$

where in equation (14), x is the original value, the mean value of $\overrightarrow{X}$ is represented as $\mu(\overrightarrow{X})$, and the standard deviation is represented in equation (15) where x_i is a data point and the number of data points in the workloads traces represented as N.

Next step, we preprocessed the CPU usage in the multivariate time series window form. The historical workloads represented as $X_t = (x_1, x_2, x_3, \ldots, x_t)$ the CPU usage record value at time t is x_t which is a sequence of data points sorted sequentially with fixed time intervals. $\hat{y}_t$ symbolize the forecasts for a prediction interval of s consecutive time steps $(x_{t+1}, x_{t+2}, x_{t+3}, \ldots, x_s)$. The scenario of formatting and predicting workloads traces are presented in Figure (3).

Where t_s is the number of predicted consecutive time steps and $s >= 1$, the final step in the prepossessing is applying the Sparse Auto-Encoder (SAE), as the historical workloads are high in dimension. Many researchers [51], [52], [53], [54] show the efficiency of using SAE in reducing the dimensions of the high and variant data. In our proposed model and as shown in Algorithm 1, we used the historical CPU Usage workloads to input the Sparse Auto-Encoder. The SAE attempts to learn an estimate of the identity function during the encoding stage $X = g(Wx + b)$ and decoding stage $\hat{X} = g(\hat{W}z + \hat{b})$ so that the output $\hat{x}$ is close to the input x. We used for the activation function g the sigmoid function as shown in equation (16):

$$g(x) = \frac{1}{1 + e^{-x}} \tag{16}$$

The reconstructing cost function between the input x and the decoded input $\hat{x}$ is represented as in equation (17) to calculate the average sum square errors for all N input data.

$$E = \frac{1}{N}\sum_{i=1}^{N} ||x_i + \hat{x}_i||^2 \tag{17}$$

We add a sparsity constraint to the hidden layer to force the Sparse Auto-Encoder to learn the number of variables required to encode the model's input data. The sparsity penalization determines the underlying dimensionality of the low-dimensional data $\hat{x}$ even if the hidden layer has a fixed number of units s^2, so the average activation of the hidden unit $\hat{p}_j$ is calculated as:

$$\hat{p}_j = \frac{1}{m}\sum_{i=1}^{m} \lceil a_j{}^2(x_i) \rceil \tag{18}$$

And to calculate the sparsity cost, we use the formula in equation (19):

$$(\sum_{j=1}^{s^2} KL(p \parallel \hat{p}_j) = \sum_{j=1}^{s^2} p \log \frac{p}{\hat{p}_j} + (1-p) \log \frac{1-p}{1-\hat{p}_j} \tag{19}$$

where p is a sparsity constraint parameter that ranges from 0 to 1.

As a result, the overall cost function is to compute the weights and bias elements by minimizing the objective function J (W, b), which has the following form:

$$J(W, b) = \frac{1}{2n}\sum_{j=1}^{n} ||x_i + \hat{x}_i||^2 + \frac{\lambda}{2}(\sum_{l=1}^{2}\sum_{i=1}^{sl} + \sum_{j=1}^{sl+1}(W_{ji}^{(l)})^2 + \beta \sum_{j=1}^{s^2} KL(p \parallel \hat{p}_j) \tag{20}$$

The first term in the cost function ensures that each input x_i is well reconstructed by minimizing the sum average squares error. The second term is a regularization term used to fine-tune the weights between the hidden and output units in order to enhance performance and prediction while preventing overfitting, sl is the number of units in layer l, and the third term ensures that the activations are sparse, where it indicates that the majority of the activations are zero.

C. THE PREDICTOR MODEL

When training a neural network, specifying the learning rate hyperparameter is crucial. The learning rate parameter adjusts the number of weight updates required to minimize the network's loss function. Setting the learning rate too low slows down the training process due to minimal changes in network weights. Conversely, setting the learning rate too high can cause divergent behavior in the loss function,

Algorithm 1 Sparse Auto-Encoder For Dimensionality Reduction

Input: Historical workloads $\overrightarrow{X} = (x_1, x_2, x_3, \ldots, x_n)$
Initialization: number of hidden units s^2, number of epoch, weight matrices $W, \hat{W}$, bias vector $b, \hat{b}$, and activation function ${a_j}^2$
for each training epoch n= 1,2,3,...n **do**
  for each hidden units j= 1,2,3.., s^2 **do**
    Encoder $X = sigmoid(Wx + b)$
    Decode $\hat{X} = sigmoid(\hat{W}z + \hat{b})$
    Compute the average activation of hidden units: $\hat{p_j} = \frac{1}{m}\sum_{i=1}^{m}\lceil {a_j}^2(x_i)\rceil$
    Calculate the sparsity cost: $\sum_{j=1}^{s^2} KL(p \parallel \hat{p_j}) = \sum_{j=1}^{s^2} p \log \frac{p}{\hat{p_j}} + (1-p)\log\frac{1-p}{1-p_j}$
  end for
  Calculate the overall cost function: $J_{sparse}(W, b) = J(W, b) + \beta \sum_{j=1}^{s^2} KL(p \parallel \hat{p_j})$
  Backward propagation to update weight parameters
end for
Output: The compressed cloud workloads traces $X_c = (x_{c1}, x_{c2}, x_{c3}, \ldots, x_{cn})$

as depicted in Figure (4). Previous research [55], [56], [57], [58], [59] has demonstrated the effectiveness of various learning rate schedulers in enhancing machine and deep learning model accuracy. This study proposes the adoption of the step-wise decay scheduler for the GRU-SWSLD model, which decreases the learning rate by a factor at specific intervals or epochs. The step-wise decay scheduler is computed using the following formula:

For each (T/T_{drop}):

$$LR = LR0 * \frac{\lambda}{\lambda factor} \tag{21}$$

where LR is the new learning rate, LR0 is the initial learning rate, λ factor is the amount the learning rate decay should change at each drop, T is the iteration/epoch number, and T_{drop} is the frequency with which the rate should be dropped (step size).

We integrated the step-wise learning decay scheduler with the Gated Recurrent Unit (GRU) model, as shown in Algorithm 2. We used the Mean Square Error to measure the performance of our proposed model for cloud workloads prediction as follows:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x_i})^2 \tag{22}$$

The complexity of Algorithm 2 in terms of big O notation is determined by many factors, including the size of the input sequence, the number of hidden units in the network, and the size of the input and output layers. The computational complexity of complete iterations of forward and backward passes through a GRU network can be

Algorithm 2 Integration of Gated Recurrent Unit (GRU) and Step-Wise Scheduler for the Learning Decay (SWSLD)

Input: The compressed cloud workloads traces $X_c = (x_{c1}, x_{c2}, x_{c3}, \ldots, x_{cn})$
Initialization: weight W, Bias b, time sequence t, number of iteration/epoch T, initial learning rate $RL0$, λfactor, iteration/epochs drop T_{drop}
for each training epoch n= 1,2,3,...T **do**
  for each epochs drop T_{drop} apply the SWSLD **do**
    $LR = LR0 * \frac{\lambda}{\lambda factor}$
    calculate the update gate z_t at time t: $z_t = sigmoid(W_{xz}x_{ct} + W_{hz}h_{t-1} + b_z)$
    calculate the rest gate r_t at time t: $r_t = sigmoid(W_{xr}x_{ct} + W_{hr}h_{t-1} + b_r)$
    from the rest gate r_t, start a new memory content $\hat{h}_t$: $\hat{h}_t = \tanh(W_{xh}x_{ct} + W_{hh}(r_t \odot h_{t-1}) + b_n)$
    Using the updated gate z_t, compute the h_t vector to determine what to hold in the current h_t and prior memory h_{t-1} states: $h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \hat{h}_t$
  end for
end for

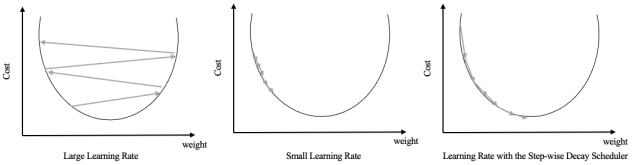


FIGURE 4. Different learning rate parameters.

expressed as $O(\sum_{n=1}^{T} NH^2 + NHD)$. The first term in the equation represents the computational complexity associated with the hidden state computations. It is directly influenced by the length of the input sequence N and the square of the number of hidden units H^2. With each iteration n out of the total T iterations, the computational complexity increases quadratically as the number of hidden units grows. The second term in the equation represents the computational complexity associated with the output layer computations in the GRU network. It is influenced by the length of the input sequence N, the number of hidden units H, and the size of the output layers D. This term accounts for the computations involved in generating the output based on the hidden state. As N, H, or D increases, the complexity grows linearly, suggesting that larger input sequences, more hidden units, or larger output layer sizes contribute to increased computational demands.

Finally, the forecaster module estimates new values for resource provision in cloud data centers using the predictor model output.

V. EXPERIMENTS

This section of the paper outlines our experimental setup and the real-world cloud workload traces used in our study. We evaluate the effectiveness of the GRU-SWSLD model for

predicting workload in cloud data centers and compare its performance against that of other models.

A. REAL-WORLD CLOUD WORKLOADS TRACES

Our experimental evaluation employs real cloud workloads obtained from Google. The dataset was publicly released by Google in May 2011 [47]. The dataset presents a diverse range of computing platforms and workloads that were performed on them, as it covers a span of 29 days and comprises 40 million and 670,000 jobs, respectively. The trace encompasses various cells, where each cell represents a collection of machines administered by a common cluster management system. Each job within the trace is composed of one or multiple tasks, each of which may comprise several processes targeted at executing on a single machine.

The following five tables comprise the trace data: the Task Constraints Table, the Task Resource Usage Table, the Task Events Table, the Jobs Events Table, and the Task Attributes Table. We will use a Task Resource Usage Table that contains 500 files for our experiment. Each file contains a summary of resource usage, and it records 20 features such as start and finish timestamps, task indexes, mean CPU usage rate, memory usage information, the total amount of allocated memory, cache memory usage, disk I/O times, sampling rates, cycles, and memory accesses per instruction (MAI) [47]. In our experiment, we will use the mean CPU usage rate.

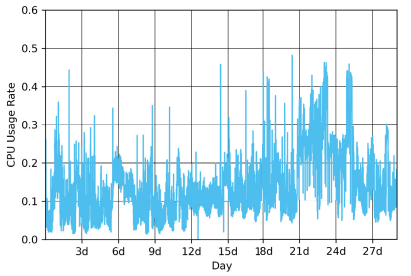
B. CLOUD WORKLOADS TRACES PREPROCESSING

In this research study, the input feature and the goal output for our model were selected to be CPU usage. To collect the necessary data, we randomly extracted information on the CPU usage of 1877 machines over a period of 29 days from Google Traces. This involved extracting task resource usage from 500 task usage files to obtain machine-specific CPU usage values. To aid in the convergence of gradient descent for model learning and optimization, we normalized the CPU usage by a zero mean and a standard deviation of one.

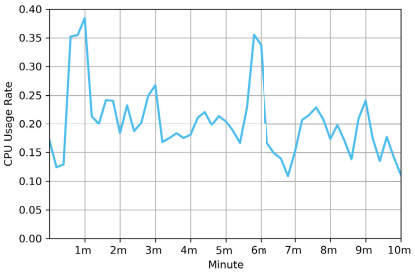
Additionally, to train and validate our proposed model, we divided the Google workload traces into 20 days for training, 6 days for validation, and 3 days for testing. Figure (5) provides a graphical representation of the daily and hourly variations in workload for a particular machine based on the Google Cloud data center workload traces.

C. EXPERIMENTAL RESULTS

We conducted experiments for the proposed approach using Python version 3.9.12 and Tensorflow version 2.10.0. The machine specification was Dell PowerEdge C8220X, Dual 6-Core 2.1GHz Intel Xeon E5-2620v2 CPUs, with Dual NVIDIA K20 GPU accelerator card. To compress real-world cloud workload traces, we first determined the optimal number of hidden units for the Sparse Auto-Encoder (SAE). To this end, we utilized Google workload traces to assess the SAE with various hidden units and compared their cost functions and compression results. The cost function



(a) Workloads Traces Per-Day



(b) Workloads Traces Per-Minute

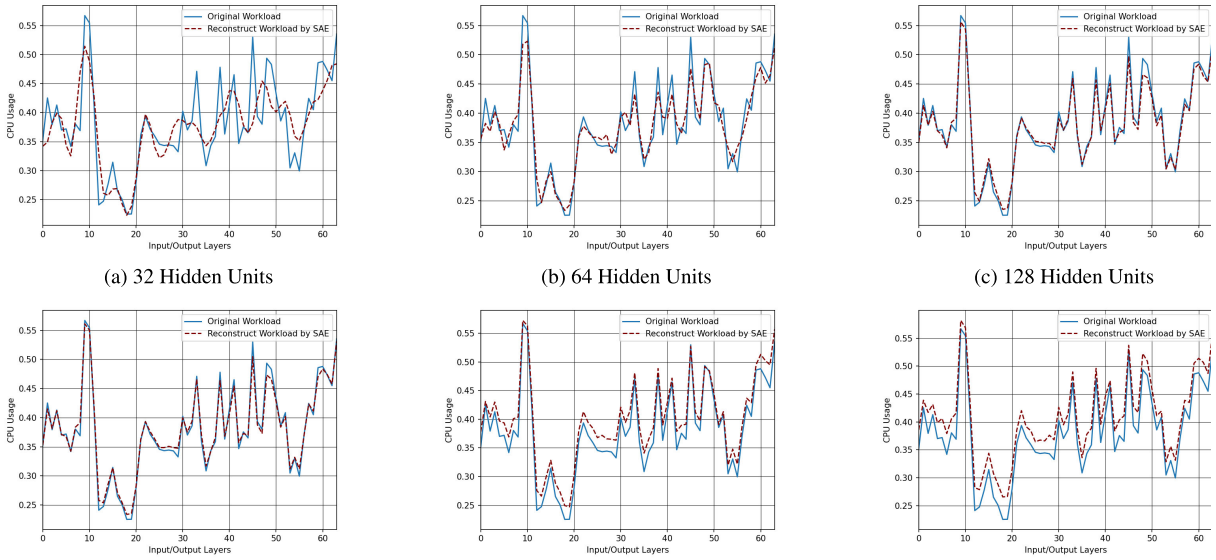
FIGURE 5. Google datacenter workloads traces for one machine.

was used to evaluate the model's accuracy in determining the relationship between the input and output concerning the hidden units. Our findings, presented in Table 1 and Figure (6) (a-f), demonstrate that the cost function is high when the number of hidden units is low, ranging from 32 to 64. However, when the number of hidden units is increased from 128 to 1024, the cost function values become relatively close, indicating that the models with these hidden units are capable of learning the important representations from actual cloud workload traces.

Furthermore, we evaluated the SAE with different input/output layers (32, 64, and 128) and a fixed number of 256 hidden units. Our results, presented in Table (2) and Figur (7) (a-c), show that the choice of input/output layers size can significantly impact the performance of the SAE. Specifically, the SAE trained with an input/output layers size of 32 achieved the lowest cost function of 0.907732, indicating that it was the most effective in reconstructing the input data. Conversely, the SAE trained with an input/output layers size of 128 achieved the highest cost function of 3.348493, indicating that it was the least effective in reconstructing the input data.

In our study, we utilized the Sparse Auto-Encoder with 256 hidden units and 32 input/output layers to compress the real-world cloud workload traces.

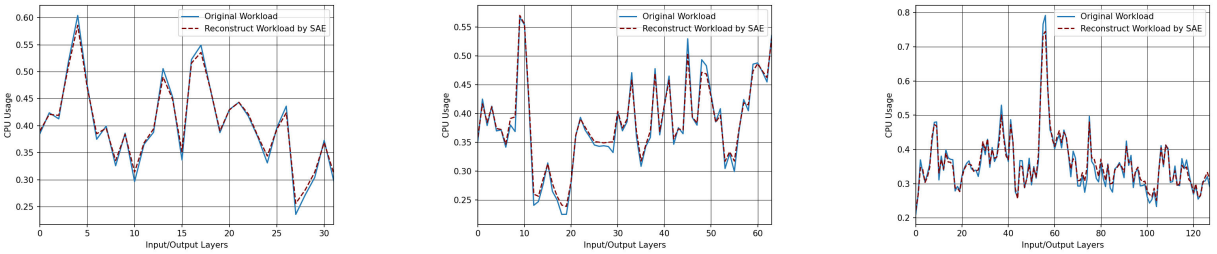
In this study, we presented an evaluation of our proposed model GRU-SWSLD for predicting workloads using compressed historical data and a combination of Gated Recurrent Units (GRU) and step-wise scheduling for learning decay. The effectiveness of our approach is compared with other deep learning methods for workload prediction such as Recurrent Neural Networks [59], Gated Recurrent Units [30], and Long Short-Term Memory [60] with varying learning



(a) 32 Hidden Units (b) 64 Hidden Units (c) 128 Hidden Units

(d) 256 Hidden Units (e) 512 Hidden Units (f) 1024 Hidden Units

FIGURE 6. Evaluation of the SAE for data center workloads traces compression with different hidden units.



(a) 32 Input/Output Layers (b) 64 Input/Output Layers (c) 128 Input/Output Layers

FIGURE 7. Evaluation of the SAE with 256 hidden units and different input/output layers.

TABLE 1. Comparing the selection of hidden units in SAE and cost functions.

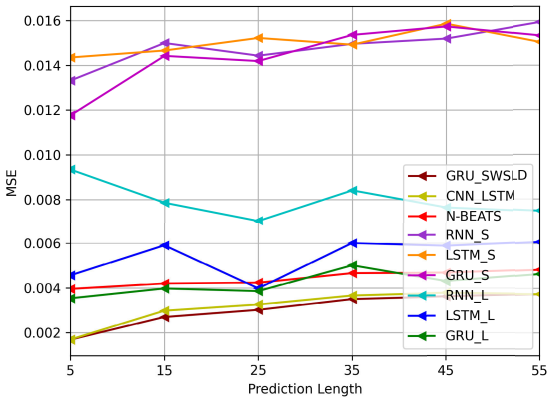
Hidden Unites	Cost Function
32	2.668244
64	1.954733
128	1.578198
256	1.553476
512	1.564936
1024	1.579853

TABLE 2. Comparing the selection of input/output layers in SAE and cost functions with 256 hidden unit.

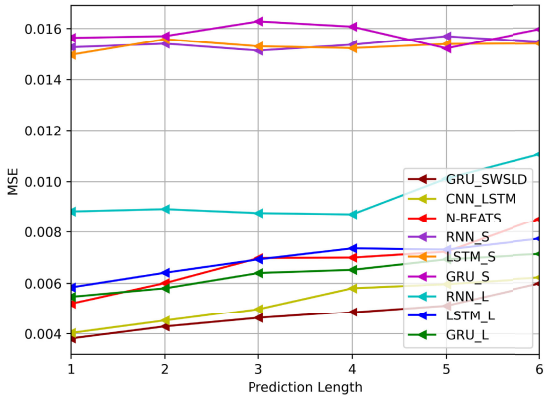
Input/Output Layers	Cost Function
32	0.907732
64	1.553476
128	3.348493

rates. Furthermore, we compared GRU-SWSLD to state-of-the-art deep learning models such as Hybrid Dilated CNN-LSTM [61] and N-BEATS [23]. We evaluate the prediction accuracy of various techniques using the Mean Squared Error (MSE) metric and training time, with different forecasting lengths. A historical sliding window of 24-time steps is set for the experiment. To assess the multivariate time series prediction, sliding windows of m = 1, 3, 5, 7, 9, and 11 are used, corresponding to sequential CPU usage values of 5, 15, 25, 35, 45, and 55 minutes. For hours multivariate time series predictions, sliding windows of h = 12, 24, 36, 48, 60, and 72 are employed, corresponding to sequential CPU usage values of 1, 2, 3, 4, 5, and 6 hours. The hyperparameters used to train the real-world workload traces for the GRU-SWSLD model are presented in Table (3).

To initiate the training process, the initial learning rate (LR0) is set to 1, the initial decay set to.1, the λ factor is



(a) Minutes



(b) Hours

FIGURE 8. Comparing the accuracy of various forecasting methods using the data center workloads traces and various prediction lengths.

set to 10, the number of epochs for the data center workload traces is set to 60, and the iteration/epochs drop T_{drop} is set to 20. To improve the performance of the GRU-SWSLD model, we incorporated a step-wise scheduler that adjusted the learning rate over the course of training. Specifically:

- For the first 20 iterations/epochs while training the GRU-SWSLD model, the new learning rate (LR) will be.01
- For 21-40 iterations/epochs while training the GRU-SWSLD model, the new learning rate (LR) will be.001
- For 41-60 iterations/epochs while training the GRU-SWSLD model, the new learning rate (LR) will be 0.0001

We conducted a comparative analysis of various neural network models for time series prediction. Specifically, we evaluated the performance of the GRU-SWSLD model in comparison with other models, including Recurrent Neural Network with a Small learning rate (RNN_S), Recurrent Neural Network with a Large learning rate (RNN_L), Long Short Term Memory with a Small learning rate (LSTM_S), Long Short Term Memory with Large learning rate (LSTM_L), Gated Recurrent Unit with Small learning rate(GRU_S), Gated Recurrent Unit with Large learning rate(GRU_L), a hybrid model of dilated Convolutional Neural Network and Long Short Term Memory (CNN_LSTM), and Neural Basis Expansion Analysis for Time Series (N-BEATS). the RNN-based models were configured with a small learning rate of 0.0001 and a large learning rate of 1.

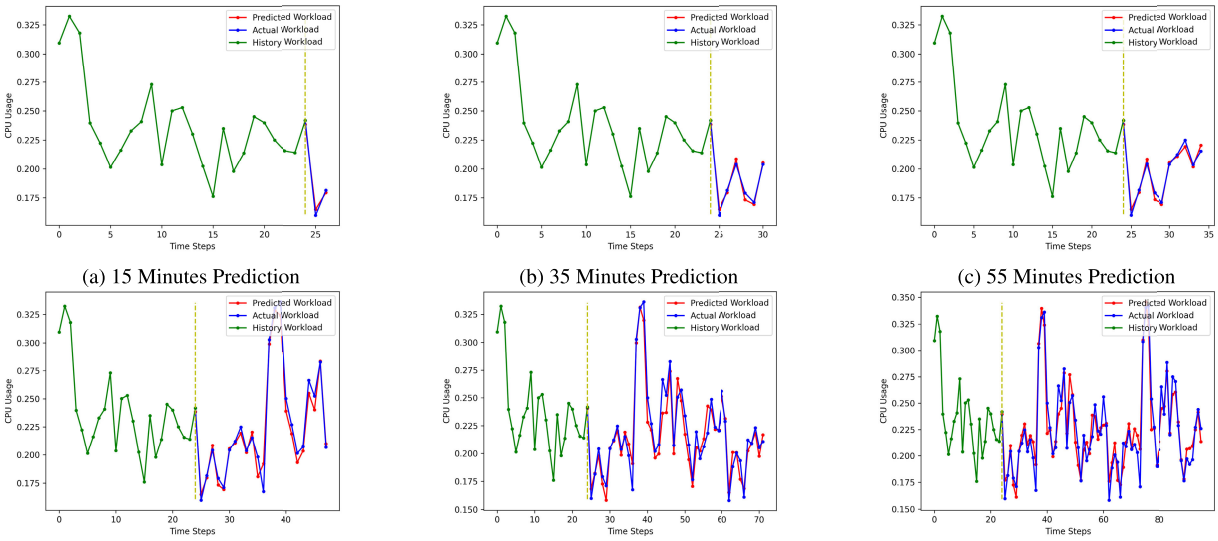
TABLE 3. Hyperparameters of proposed GRU-SWSLD model.

Hyper Parameters	Values
History Window Length	24
Number of Training Epochs	60
Batch Size	128
Initial Learning Rate	1
λ	.1
iteration/epochs drop T_{drop}	20
$\lambda factor$	10

Figure (8) illustrates the Mean Square Error (MSE) of various techniques with respect to different forecasting lengths. The results demonstrate a general increase in the MSE for all of the considered approaches as the prediction length increases. Notably, the GRU-SWSLD model outperforms other Recurrent Neural Networks (RNN)-based methods that utilize small or large learning rates to achieve accurate predictions. This can be attributed to the effectiveness of the Sparse Auto-Encoder in reconstructing low-dimensional features from the original workload traces, as well as the integration of the GRU and a step-wise learning decay scheduler in the model.

In the context of recurrent neural networks (RNNs), such as RNN_S, LSTM_S, and GRU_S, the application of small learning rates can result in convergence taking a significant amount of time, or in some cases, failing to converge entirely. The use of small learning rates increases the risk of becoming trapped in a local minimum of the loss function, instead of finding the global minimum. This is due to the optimization algorithm being unable to make updates significant enough to escape the local minimum. Conversely, the use of high learning rates for RNN_L, LSTM_L, and GRU_L can cause the prediction algorithm to overshoot the minimum of the loss function and ultimately result in a suboptimal solution.

The GRU-SWSLD model demonstrated superior performance when compared to the dilated CNN-LSTM model. The latter model may introduce information loss in the feature maps as a result of the dilation rate which regulates the separation between filter weights. The combination of the dilated CNN with LSTM has the potential to compromise the level of detail retained in the input features, thereby overlooking important information and leading to a loss of data. Ultimately, this may impede the LSTM's capacity to learn valuable representations and result in a decline in predictive accuracy. Additionally, it should be noted that N-BEATS is not well-suited to managing high-dimensional data, including



(a) 15 Minutes Prediction (b) 35 Minutes Prediction (c) 55 Minutes Prediction

(d) 2 Hours Prediction (e) 4 Hours Prediction (f) 6 Hours Prediction

FIGURE 9. Demonstration of GRU-SWSLD's accuracy on google workloads traces by varying the length of prediction intervals.

TABLE 4. MSE Comparison with data center workloads traces of various forecasting methods with different levels of prediction lengths.

Models	5m	15m	25m	35m	45m	55m	1h	2h	3h	4h	5h	6h
GRU_SWSLD	0.001693	0.002701	0.003021	0.003501	0.003619	0.003729	0.003818	0.004287	0.004627	0.004872	0.005132	0.005966
CNN_LSTM	0.001701	0.002982	0.003258	0.003661	0.003781	0.003724	0.004031	0.004521	0.004996	0.005810	0.005970	0.006204
N_BEATS	0.003954	0.004198	0.004235	0.004661	0.004681	0.004824	0.005214	0.006013	0.006982	0.007001	0.007231	0.008536
RNN_S	0.013318	0.014996	0.01443	0.014959	0.015194	0.015927	0.015267	0.01541	0.015136	0.015371	0.015699	0.015467
LSTM_S	0.01435	0.014661	0.015223	0.014923	0.01586	0.015036	0.014975	0.015582	0.015305	0.015234	0.015399	0.015417
GRU_S	0.011763	0.014421	0.014189	0.015359	0.015735	0.015329	0.015639	0.015706	0.016285	0.016075	0.015226	0.015958
RNN_L	0.009343	0.007854	0.007031	0.00841	0.007642	0.007465	0.008837	0.008928	0.008771	0.008722	0.010124	0.011086
LSTM_L	0.004563	0.005901	0.003982	0.006004	0.005883	0.006068	0.005846	0.006414	0.006932	0.007359	0.00731	0.007729
GRU_L	0.003537	0.003973	0.003857	0.005011	0.0043	0.004631	0.00548	0.005809	0.006404	0.006526	0.006932	0.007131

data center workload traces, given its assumption that the time series data exhibit a periodic pattern.

The findings presented in Table (4) demonstrate that the GRU-SWSLD approach is superior to other methods for capturing long-term memory relationships in high-dimensional cloud data center workloads. This superiority is achieved through the use of a step-wise scheduler for learning decay, which enables the model to avoid a fixed learning rate. The step-wise scheduler adjusts the learning rate at predetermined intervals throughout the training process, allowing for larger parameter updates during the initial stages of training when the loss function is likely to be high. Subsequently, smaller parameter updates are made as the loss function approaches a minimum, enabling the model to converge more rapidly to an optimal solution while avoiding overshooting or becoming trapped in a suboptimal local minimum.

Next, the training times of various techniques for workload traces forecasting were examined and compared, as illustrated in Figure (10). The investigated methods include RNN_S, RNN_L, LSTM_S, LSTM_L, GRU_S, GRU_L, SAE-GRU-SWSLD, dilated CNN-LSTM, and N-BEATS. The results indicate that RNN_S and RNN_L are the quickest and require the least training time, but their simple structure makes it challenging for them to manage long-term dependencies. LSTM_S and LSTM_L, on the other hand, can store and access long-term dependencies, but their training

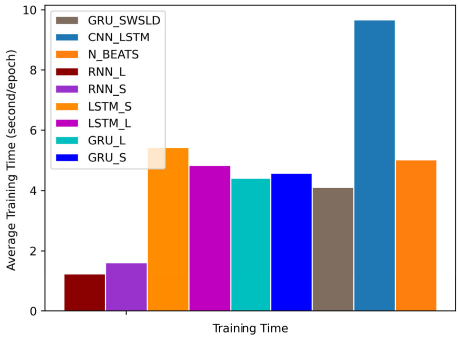


FIGURE 10. Comparing the training time of various forecasting methods using the data center workloads traces.

time is longer due to the complexity of their memory cells and three gates. GRU_S and GRU_L, which have fewer gates and simpler structures than LSTMs, require less training time than LSTM approaches but more than RNNs. By integrating the GRU model with a step-wise scheduler for learning decay, the GRU-SWSLD model achieved even less training time than GRU_S and GRU_L when combined with the SAE method. Finally, dilated CNN-LSTM and N-BEATS, which have complex architectures and a large number of parameters that need to be learned, require a longer training time than the other methods. In comparison to other prediction methods,

the GRU-SWSLD model has been found to offer a better balance between training time and prediction accuracy.

Furthermore, Figure (9) illustrates the variance between the predicted values generated by the GRU-SWSLD model and the actual values on the Google Cloud workload traces, across various forecast lengths, both in minutes and hours. These results highlight the model's efficacy in handling high-dimensional data center workload traces, across a range of prediction lengths.

Given the importance of efficient resource utilization and workload management in the cloud, the findings mentioned above thus show how well-suited the GRU-SWSLD is for handling dynamic and high dimensional data center workload traces with a range of prediction lengths.

VI. CONCLUSION

This study highlights the importance of accurate workload forecasting in cloud data centers prior to implementing auto-scaling. Our proposed approach, which incorporates a Sparse Auto-Encoder and a Gated Recurrent Unit with a step-wise learning decay scheduler, offers a novel solution for predicting cloud workload traces with a better tradeoff between prediction accuracy and training time. The results demonstrate that our proposed GRU-SWSLD model outperforms other RNNs-based models with small and large learning rates as well as advanced deep learning algorithms in terms of convergence, optimization, and handling of high-dimensional data of cloud workload traces with varying patterns. The step-wise learning decay scheduler also offers fine-grained control of the learning rate, allowing the model to quickly adapt to changes in workload patterns. In future work, we plan to explore reinforcement learning and further enhance deep learning models to cater to the complex requirements of cloud computing, fog computing, and IoT environments. Such endeavors will pave the way for more efficient, reliable, and scalable cloud systems, enabling better resource allocation, energy efficiency, and cost savings for cloud service providers and end-users alike.

ACKNOWLEDGMENT

The author would like to thank Tarek A. El-Ghazawi for his support throughout her Ph.D. studies.

REFERENCES

[1] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: State-of-the-art and research challenges," *J. Internet Services Appl.*, vol. 1, no. 1, pp. 7–18, Apr. 2010, doi: 10.1007/s13174-010-0007-6.

[2] S. A. Bello, L. O. Oyedele, O. O. Akinade, M. Bilal, J. M. Davila Delgado, L. A. Akanbi, A. O. Ajayi, and H. A. Owolabi, "Cloud computing in construction industry: Use cases, benefits and challenges," *Autom. Construction*, vol. 122, Feb. 2021, Art. no. 103441, doi: 10.1016/j.autcon.2020.103441.

[3] A. Shahidinejad, M. Ghobaei-Arani, and M. Masdari, "Resource provisioning using workload clustering in cloud computing environment: A hybrid approach," *Cluster Comput.*, vol. 24, no. 1, pp. 319–342, Apr. 2020, doi: 10.1007/s10586-020-03107-0.

[4] F. Chen, A. Lu, H. Wu, R. Dou, and X. Wang, "Optimal strategies on pricing and resource allocation for cloud services with service guarantees," *Comput. Ind. Eng.*, vol. 165, Mar. 2022, Art. no. 107957, doi: 10.1016/j.cie.2022.107957.

[5] Z. Wang, M. M. Hayat, N. Ghani, and K. B. Shaban, "Optimizing cloud-service performance: Efficient resource provisioning via optimal workload allocation," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 6, pp. 1689–1702, Jun. 2017, doi: 10.1109/TPDS.2016.2628370.

[6] P. T. Endo, M. Rodrigues, G. E. Gonçalves, J. Kelner, D. H. Sadok, and C. Curescu, "High availability in clouds: Systematic review and research challenges," *J. Cloud Comput.*, vol. 5, no. 1, Oct. 2016, doi: 10.1186/s13677-016-0066-8.

[7] J. Guo, Z. Chang, S. Wang, H. Ding, Y. Feng, L. Mao, and Y. Bao, "Who limits the resource efficiency of my datacenter," in *Proc. Int. Symp. Quality Service*, Jun. 2019, pp. 1–10, doi: 10.1145/3326285.3329074.

[8] M. H. U. Rehman, C. S. Liew, A. Abbas, P. P. Jayaraman, T. Y. Wah, and S. U. Khan, "Big data reduction methods: A survey," *Data Sci. Eng.*, vol. 1, no. 4, pp. 265–284, Dec. 2016, doi: 10.1007/s41019-016-0022-0.

[9] T. V. T. Duy, Y. Sato, and Y. Inoguchi, "Improving accuracy of host load predictions on computational grids by artificial neural networks," *Int. J. Parallel, Emergent Distrib. Syst.*, vol. 26, no. 4, pp. 275–290, Aug. 2011, doi: 10.1080/17445760.2010.481786.

[10] Y. Li and Z. Lan, "A survey of load balancing in grid computing," in *Proc. Int. Conf. Comput. Inf. Sci.*, 2004, pp. 280–285, doi: 10.1007/978-3-540-30497-5_44.

[11] Y. Wu, Y. Yuan, G. Yang, and W. Zheng, "Load prediction using hybrid model for computational grid," in *Proc. 8th IEEE/ACM Int. Conf. Grid Comput.*, Sep. 2007, pp. 235–242, doi: 10.1109/grid.2007.4354138.

[12] S. Di, D. Kondo, and W. Cirne, "Characterization and comparison of cloud versus grid workloads," in *Proc. IEEE Int. Conf. Cluster Comput.*, Sep. 2012, pp. 230–238, doi: 10.1109/cluster.2012.35.

[13] Z. Amekraz and M. Y. Hadi, "An adaptive workload prediction strategy for non-Gaussian cloud service using ARMA model with higher order statistics," in *Proc. IEEE 11th Int. Conf. Cloud Comput. (CLOUD)*, Jul. 2018, pp. 646–651, doi: 10.1109/cloud.2018.00089.

[14] T. Islam and D. Manivannan, "Predicting application failure in cloud: A machine learning approach," in *Proc. IEEE Int. Conf. Cognit. Comput. (ICCC)*, Jun. 2017, pp. 24–31, doi: 10.1109/ieee.iccc.2017.11.

[15] J. Qiu, Q. Wu, G. Ding, Y. Xu, and S. Feng, "A survey of machine learning for big data processing," *EURASIP J. Adv. Signal Process.*, vol. 2016, no. 1, p. 67, May 2016, doi: 10.1186/s13634-016-0355-x.

[16] P. Singh, P. Gupta, and K. Jyoti, "TASM: Technocrat ARIMA and SVR model for workload prediction of web applications in cloud," *Cluster Comput.*, vol. 22, no. 2, pp. 619–633, Nov. 2018, doi: 10.1007/s10586-018-2868-6.

[17] M. Toukir Imam, S. Faisal Miskhat, R. M. Rahman, and M. A. Amin, "Neural network and regression based processor load prediction for efficient scaling of grid and cloud resources," in *Proc. 14th Int. Conf. Comput. Inf. Technol. (ICCIT)*, Dec. 2011, pp. 333–338, doi: 10.1109/iccitechn.2011.6164809.

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[19] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *Int. J. Uncertainty, Fuzziness Knowl.-Based Syst.*, vol. 6, no. 2, pp. 107–116, Apr. 1998, doi: 10.1142/s0218488598000094.

[20] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[21] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014, *arXiv:1412.3555*.

[22] I. Jolliffe, "Principal component analysis," in *International Encyclopedia of Statistical Science*. 2011, pp. 1094–1096. [Online]. Available: https://link.springer.com/referenceworkentry/10.1007/978-3-642-04898-2_455, doi: 10.1007/978-3-642-04898-2_455.

[23] E. Patel and D. S. Kushwaha, "A hybrid CNN-LSTM model for predicting server load in cloud computing," *J. Supercomput.*, vol. 78, no. 8, pp. 1–30, Jan. 2022, doi: 10.1007/s11227-021-04234-0.

[24] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," 2012, *arXiv:1206.5533*.

[25] H. Shi, M. Xu, and R. Li, "Deep learning for household load forecasting—A novel pooling deep RNN," *IEEE Trans. Smart Grid*, vol. 9, no. 5, pp. 5271–5280, Sep. 2018, doi: 10.1109/tsg.2017.2686012.

[26] S. Kullback and R. A. Leibler, "On information and sufficiency," *Ann. Math. Statist.*, vol. 22, no. 1, pp. 79–86, Mar. 1951, doi: 10.1214/aoms/1177729694.

[27] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 157–166, Mar. 1994, doi: 10.1109/72.279181.

[28] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with LSTM recurrent networks," *J. Mach. Learn. Res.*, vol. 3, no. 1, pp. 115–143, 2003, doi: https://doi.org/10.1162/153244303768966139.

[29] J. Quinn, J. Mceachen, M. Fullan, M. Gardner, and M. Drummy, *Dive Into Deep Learning: Tools for Engagement*. Thousand Oaks, CA, USA: Corwin, A Sage Company, 2020.

[30] J. Zhai, S. Zhang, J. Chen, and Q. He, "Autoencoder and its various variants," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Oct. 2018, pp. 415–419, doi: 10.1109/smc.2018.00080.

[31] A.-F. Antonescu and T. Braun, "Simulation of SLA-based VM-scaling algorithms for cloud-distributed applications," *Future Gener. Comput. Syst.*, vol. 54, pp. 260–273, Jan. 2016, doi: 10.1016/j.future.2015.01.015.

[32] Y. Hu, B. Deng, F. Peng, and D. Wang, "Workload prediction for cloud computing elasticity mechanism," in *Proc. IEEE Int. Conf. Cloud Comput. Big Data Anal. (ICCCBDA)*, Jul. 2016, pp. 244–249. Accessed: Apr. 5, 2021. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7529565

[33] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, "Workload prediction using ARIMA model and its impact on cloud applications' QoS," *IEEE Trans. Cloud Comput.*, vol. 3, no. 4, pp. 449–458, Oct. 2015, doi: 10.1109/tcc.2014.2350475.

[34] P. Saripalli, G. V. R. Kiran, R. R. Shankar, H. Narware, and N. Bindal, "Load prediction and hot spot detection models for autonomic cloud computing," in *Proc. 4th IEEE Int. Conf. Utility Cloud Comput.*, Dec. 2011, pp. 397–402, doi: 10.1109/ucc.2011.66.

[35] M. Amiri and L. Mohammad-Khanli, "Survey on prediction models of applications for resources provisioning in cloud," *J. Netw. Comput. Appl.*, vol. 82, pp. 93–113, Mar. 2017, doi: 10.1016/j.jnca.2017.01.016.

[36] A. S. Kumar and S. Mazumdar, "Forecasting HPC workload using ARMA models and SSA," in *Proc. Int. Conf. Inf. Technol. (ICIT)*, Dec. 2016, pp. 294–297, doi: 10.1109/icit.2016.065.

[37] S. Singh and I. Chana, "Q-aware: Quality of service based cloud resource provisioning," *Comput. Electr. Eng.*, vol. 47, pp. 138–160, Oct. 2015, doi: 10.1016/j.compeleceng.2015.02.003.

[38] S. Liao, H. Zhang, G. Shu, and J. Li, "Adaptive resource prediction in the cloud using linear stacking model," in *Proc. 5th Int. Conf. Adv. Cloud Big Data (CBD)*, Aug. 2017, pp. 33–38, doi: 10.1109/cbd.2017.14.

[39] M. Duggan, K. Mason, J. Duggan, E. Howley, and E. Barrett, "Predicting host CPU utilization in cloud computing using recurrent neural networks," in *Proc. 12th Int. Conf. Internet Technol. Secured Trans. (ICITST)*, Dec. 2017, pp. 67–72, doi: 10.23919/icitst.2017.8356348.

[40] D. Janardhanan and E. Barrett, "CPU workload forecasting of machines in data centers using LSTM recurrent neural networks and ARIMA models," in *Proc. 12th Int. Conf. Internet Technol. Secured Trans. (ICITST)*, Dec. 2017, pp. 55–60, doi: 10.23919/icitst.2017.8356346.

[41] Q. Yang, Y. Zhou, Y. Yu, J. Yuan, X. Xing, and S. Du, "Multi-step-ahead host load prediction using autoencoder and echo state networks in cloud computing," *J. Supercomput.*, vol. 71, no. 8, pp. 3037–3053, Apr. 2015, doi: 10.1007/s11227-015-1426-8.

[42] J. Gao, H. Wang, and H. Shen, "Task failure prediction in cloud data centers using deep learning," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2019, pp. 1111–1116, doi: 10.1109/bigdata47090.2019.9006011.

[43] A. A. Rahmanian, M. Ghobaei-Arani, and S. Tofighya, "A learning automata-based ensemble resource usage prediction algorithm for cloud computing environment," *Future Gener. Comput. Syst.*, vol. 79, pp. 54–71, Feb. 2018, doi: 10.1016/j.future.2017.09.049.

[44] M. Ghobaei-Arani, S. Jabbehdari, and M. A. Pourmina, "An autonomic resource provisioning approach for service-based cloud applications: A hybrid approach," *Future Gener. Comput. Syst.*, vol. 78, pp. 191–210, Jan. 2018, doi: 10.1016/j.future.2017.02.022.

[45] J. Bi, H. Yuan, and M. Zhou, "Temporal prediction of multiapplication consolidated workloads in distributed clouds," *IEEE Trans. Autom. Sci. Eng.*, vol. 16, no. 4, pp. 1763–1773, Oct. 2019, doi: 10.1109/tase.2019.2895801.

[46] T. Mikolov, S. Kombrink, L. Burget, J. Cernocký, and S. Khudanpur, "Extensions of recurrent neural network language model," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2011, pp. 5528–5531. [Online]. Available: https://ieeexplore.ieee.org/document/5947611

[47] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: Format+ schema," Google, Menlo Park, CA, USA, White Paper Version 2.1, 2011, pp. 1–14. [Online]. Available: https://drive.google.com/file/d/0B5g07T_gRDg9Z0lsSTEtTWtpOW8/view?resourcekey=0-cozD56gA4fUDdrkHnLJSrQ

[48] J. Guo, Z. Chang, S. Wang, H. Ding, Y. Feng, L. Mao, and Y. Bao, "Who limits the resource efficiency of my datacenter: An analysis of Alibaba datacenter traces," in *Proc. Int. Symp. Quality Service*, Jun. 2019, pp. 1–10, doi: 10.1145/3326285.3329074.

[49] H. Liu, "A measurement study of server utilization in public clouds," in *Proc. IEEE 9th Int. Conf. Dependable, Autonomic Secure Comput.*, Dec. 2011, pp. 435–442, doi: 10.1109/dasc.2011.87.

[50] N. Andrew, "CS294A Lecture notes: Sparse autoencoder," Stanford Univ., Stanford, CA, USA, Lect. Notes, 2011, pp. 1–19.

[51] J. Bi, Z. Chen, H. Yuan, J. Zhang, and M. Zhou, "Self-adaptive teaching-learning-based optimizer with improved RBF and sparse autoencoder for high-dimensional problems," *Inf. Sci.*, vol. 630, pp. 463–481, Oct. 2022.

[52] F. Huang, J. Zhang, C. Zhou, Y. Wang, J. Huang, and L. Zhu, "A deep learning algorithm using a fully connected sparse autoencoder neural network for landslide susceptibility prediction," *Landslides*, vol. 17, no. 1, pp. 217–229, Sep. 2019, doi: 10.1007/s10346-019-01274-9.

[53] X. Han, Y. Liu, Z. Zhang, X. Lü, and Y. Li, "Sparse auto-encoder combined with kernel for network attack detection," *Comput. Commun.*, vol. 173, pp. 14–20, May 2021, doi: 10.1016/j.comcom.2021.03.004.

[54] Q. Hernandez, A. Badías, D. González, F. Chinesta, and E. Cueto, "Deep learning of thermodynamics-aware reduced-order models from data," *Comput. Methods Appl. Mech. Eng.*, vol. 379, Jun. 2021, Art. no. 113763, doi: 10.1016/j.cma.2021.113763.

[55] R. Ge, S. Kakade, R. Kidambi, and P. Netrapall, "The step decay schedule: A near optimal, geometrically decaying learning rate procedure for least squares," *Proc. 33rd Int. Conf. Neural Inf. Process. Syst.*, vol. 1341, 2019, pp. 14977–14988.

[56] Z. Li and S. Arora, "An exponential learning rate schedule for deep learning," 2019, *arXiv:1910.07454*.

[57] Z. Xie, I. Sato, and M. Sugiyama, "Understanding and scheduling weight decay," in *Proc. ICLR Conf.*, Nov. 2021, pp. 1–23.

[58] H. Yang, J. Liu, H. Sun, and H. Zhang, "PACL: Piecewise arc cotangent decay learning rate for deep neural network training," *IEEE Access*, vol. 8, pp. 112805–112813, 2020, doi: 10.1109/access.2020.3002884.

[59] A. Khodamoradi, K. Denolf, K. Vissers, and R. C. Kastner, "ASLR: An adaptive scheduler for learning rate," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2021, pp. 1–8, doi: 10.1109/IJCNN52387.2021.9534014.

[60] Y. Guo and W. Yao, "Applying gated recurrent units approaches for workload prediction," in *Proc. IEEE/IFIP Netw. Operations Manage. Symp.*, Apr. 2018, pp. 1–6.

[61] B. Song, Y. Yu, Y. Zhou, Z. Wang, and S. Du, "Host load prediction with long short-term memory in cloud computing," *J. Supercomput.*, vol. 74, no. 12, pp. 6554–6568, Apr. 2017, doi: 10.1007/s11227-017-2044-4.

[62] J. Bedi and Y. S. Patel, "STOWP: A light-weight deep residual network integrated windowing strategy for storage workload prediction in cloud systems," *Eng. Appl. Artif. Intell.*, vol. 115, Oct. 2022, Art. no. 105303, doi: 10.1016/j.engappai.2022.105303.

DALAL ALQAHTANI received the Bachelor of Science degree in computer science from Imam Abdulrahman Bin Faisal University, in 2009, and the Master of Science degree in computer science from The George Washington University, in 2015, where she is currently pursuing the Ph.D. degree with the Electrical and Computer Engineering Department. In addition to her academic pursuits, she is also an part-time IT Technical Specialist with the Division of Information Technology, The George Washington University. Since 2009, she has been a Faculty Member with the Computer Science and Information Systems Department, Najran University. Her research interests include high-performance computing, cloud computing, the Internet of Things, artificial intelligence, fog computing, reinforcement learning, and computer security. She is a member of Women in CyberSecurity.