Data Center Resource Usage Forecasting with Convolutional Recurrent Neural Networks

Miika Malin and Jaakko Suutala

University of Oulu / ITEE / BISG, 9BISG, P.O. Box 4500, 90014
Oulu Finland (e-mail: {firstname.lastname}@oulu.fi)

Abstract: Energy efficiency, scalability, and reliability are increasingly important for sustainable data centers. In this paper, we focus on forecasting real-world resource usage using neural network time series models, specifically utilizing convolutional recurrent long short-term Memory (LSTM) and gated recurrent unit (GRU) architectures. In our analysis, we compare LSTM and GRU in terms of forecasting accuracy and computational complexity during model training. We demonstrate that recurrent neural networks are more accurate and robust compared to the traditional autoregressive integrated moving average (ARIMA) time series model in this complex forecasting problem. GRU achieved a 9% reduction and LSTM a 5% reduction in forecasting mean squared error (MSE) compared to ARIMA. Furthermore, the GRU architecture with a 1D convolution layer outperforms LSTM architecture in both forecast accuracy and training time. The proposed model can be effectively applied to load forecasting as part of a data center computing cluster. In this application, the proposed GRU architecture has 25% fewer trainable parameters in the recurrent layer than the commonly used LSTM.

Keywords: recurrent neural network, convolutional neural network, data center load forecasting, energy efficiency, sustainability, control optimization, monitoring

1. INTRODUCTION

Energy consumption reduction and resource optimization are becoming important in computational intensive data center environments aiming towards more sustainable and green systems (Bourhnane et al., 2020). World wide energy consumption of data centers has been estimated rose to 205 TWh in 2018 from 153 TWh in 2005. This means ∼1% of global total electricity usage (Masanet et al., 2020; Jones, 2018). The whole information and communications ecosystem causes more than 2% of emissions. This is on same level with aviation fuel emissions, and it is predicted to be even higher in future (Jones, 2018).

Optimizing energy consumption of hardware in data centers is critical, as servers and other IT equipment can typically take more than 40% of total energy-usage in data center (Shehabi et al., 2016). To be able to optimize the IT systems in proactive manner, intelligent and efficient resource usage forecast is one subject to be solved. Several different neural network architectures have been proposed for the task, and many of the solutions use recurrent neural network (RNN) based approaches (e.g in (Zhang et al., 2016; Janardhanan and Barrett, 2017; Ouhame et al., 2021; Yuan et al., 2024)) since it has been designed to be used with sequential problems such as time series forecasting. Also more traditional models such as autoregressive integrated moving average (ARIMA). ARIMA is a common tool used by statisticians in time series forecasting (Hewamalage, 2020), and has been used to tackle the problem, e.g. in (Kumar and Singh, 2020; Calheiros et al., 2015). In this work, we are going to compare two recurrent neural network architectures: long short-term memory (LSTM) and gated recurrent unit (GRU) together with traditional ARIMA model.

While several architectures leveraging recurrent neural networks have emerged to address load forecasting, they often overlook the critical attribute of efficiency. Notably, many of these architectures do not utilize convolutional or more efficient GRU recurrent layers. In a notable advancement, the architecture proposed by Ouhame et al. (2021) focused on optimizing the convolutional layer of the LSTM model for efficiency. Building upon this progress, our work introduces a novel approach by integrating both GRU and convolutional layers. This synergistic combination not only enhances forecast accuracy but also improves model efficiency, a critical attribute essential for real-world integration in data center scenarios.

Efficient and dependable resource usage forecasts are essential for managing dynamic, scalable clusters, enabling the adjustment of the number of powered-on physical machines as needed. Such forecasts hold the potential to significantly reduce energy consumption in data centers (Bayati, 2018). Additionally, resource usage forecasts find application in load balancing, particularly in virtual machine (VM)-based data centers where the allocation of physical machines for VMs can be modified. This capability facilitates the optimization of resource utilization and enhances the overall efficiency of data center operations (Shaw et al., 2017).

2. METHODS FOR TIME SERIES FORECASTING

In this section, we explore the time series forecasting methodologies employed in this study, with a focus on predicting data center resource usage efficiently. By leveraging advanced predictive techniques, our aim is to provide a robust and efficient framework for analyzing temporal data patterns and enhancing the accuracy of our forecasts. The forecasts can be used to optimize resource allocation and improving operational adaptability in data centers.

2.1 Autoregressive Integrated Moving Average model

The autoregressive integrated moving average model is a widely-used forecasting method that integrates autoregression, moving average, and differencing. In this study, we employ the ARIMA model as a baseline for forecasting due to its historical prevalence in data center resource usage prediction tasks and its ability to effectively model various types of time series data.

ARIMA model can be defined as

$$\begin{aligned} y_t^{'} =& \alpha + \phi_1 y_{t-1}^{'} + ... + \phi_p y_{t-p}^{'} \\ &+ \theta_1 \epsilon_{t-1} + ... + \theta_q \epsilon_{t-q} + \epsilon_t, \end{aligned} \tag{1}$$

where α is the optional intercept of the model. ϕ_i is the coefficient for the autoregressive part of the model, and θ_i is the coefficient for the moving average part of the model. $y_t^{'}$ is the value of the differenced time series at timestep t, and ϵ_i is the past forecast error (Hyndman and Athanasopoulos, 2019).

ARIMA model uses hyperparameters p, d, q, and following conclusions about ARIMA model hyperparameters p and q can now be derived from the Equation 1: p can be seen as the order of the autoregressive part of the model, and q is the order of moving average part of the model. The hyperparameter d is the order of differencing in time series (Hyndman and Athanasopoulos, 2019).

2.2 Convolution on Time Series Data

Convolution can be seen as sliding a window over the data. Due to the success of convolutional neural networks (CNNs) in image and natural language processing, convolution layers has been applied to time series analysis as well.

In CNNs processing image data, convolution involves a two-dimensional filter sliding over the width and height of the image (Goodfellow et al., 2016). In contrast, for time series data, the only dimension to slide over is time, which requires the use of one-dimensional convolution.

Two dimensional convolution starting from point (i, j) can be defined as:

$$C(i,j) = \sum_{m=0}^{h} \sum_{n=0}^{w} I_{i+m,j+n} K_{m,n}, \tag{2}$$

where I is the input data, K is the kernel of the convolution with dimensions $h \times w$. Kernel contains weights w for the convolution. Here, h is the height, and w is the width of the convolution window. (Goodfellow et al., 2016)

Since the 1D convolution can only slide through one dimension, w is always same as number of features in input. Definition for 1D convolution can be derived from Equation 2:

$$C(i) = \sum_{m=0}^{h} \sum_{n=0}^{w} I_{i+m,n} K_{m,n}, \tag{3}$$

where i is the row of input where the convolution starts and w is the number of features in dataset. In time series context, w can be seen as a number of features recorded in each time point. Again, in time series context this means that kernel is slided over the time dimension.

Use of the 1D convolution reduces the computational complexity from $\sim O(N^2K^2)$ to $\sim O(NK)$ when comparing to 2D convolution, when input dimensions are $N \times N$ and kernel dimensions $K \times K$ (Kiranyaz et al., 2021).

2.3 Long Short-Term Memory

Long short-term memory is a recurrent neural network, which uses LSTM units and tries to solve vanishing gradient problem of recurrent neural networks (Hochreiter and Schmidhuber, 1997). This means that the architecture can find more efficiently long term dependencies from time series.

LSTM network with forget gate and biases consists of LSTM cells, where each cell has three gates:

- Forget gate

$$f_t = S(W_f x_t + U_f h_{t-1} + b_f). \tag{4}$$

- Input Gate

$$i_t = S(W_i x_t + U_i h_{t-1} + b_i). \tag{5}$$

- Output gate

$$o_t = S(W_o x_t + U_o h_{t-1} + b_o). \tag{6}$$

In Equations 4-6 S is an activation function. Often S is a sigmoid function, as proposed in the original version of LSTM (Hochreiter and Schmidhuber, 1997; Hewamalage, 2020; Dey and Salem, 2017). Now output / hidden state h_t of cell at timestep t can be defined as:

$$h_t = o_t \odot T(c_t), \tag{7}$$

where

$$\begin{aligned} c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\ \tilde{c}_t &= T(W_c x_t + U_c h_{t-1} + b_c). \end{aligned} \tag{8}$$

In Equations 7 and 8 T is an activation function, and often $tanh$ function is used as proposed in the original architecture (Hochreiter and Schmidhuber, 1997). In Equations 4 - 8 x_t is the input vector for LSTM cell at timestep t. W_x, U_x and b_x are weights and biases to be tuned in the training process. The operation symbol $\odot$ is an element-wise Hadamard product. (Hochreiter and Schmidhuber, 1997; Hewamalage, 2020; Dey and Salem, 2017)

From Equations 4-8 we get that LSTM has total of $4(n^2 + nm + n)$ optimizable parameters. Here n is the dimension of hidden state and m is the dimension of input vector. (Dey and Salem, 2017)

2.4 Gated Recurrent Unit

Gated recurrent unit has been motivated by LSTM unit, but it has a simpler design (Cho et al., 2014). Unlike LSTM units, GRUs do not have an output gate as shown in Equations 9-11. This more streamlined architecture provides efficiency in both training and forecasting tasks. Since it has fewer gates, it also has fewer weights to optimize, making the backpropagation through time faster. Additionally, using the trained model to yield forecasts is more efficient due to the reduced number of calculations required.

GRU network consists of multiple GRU cells, where each cell has two gates:

- Reset gate

$$r_t = S(W_r x_t + U_r h_{t-1} + b_r), \tag{9}$$

- Update gate

$$u_t = S(W_u x_t + U_u h_{t-1} + b_u). \tag{10}$$

By using these two gates the output / hidden state h_t at timestep t can be defined as:

$$\begin{aligned} h_t &= (1 - u_t) \odot h_{t-1} + u_t \odot \tilde{h}_t \\ \tilde{h}_t &= T(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h), \end{aligned} \tag{11}$$

where T is hyperbolic tangent and S is sigmoid activation function.

All W_x are weight matrices, and all b_x are bias vectors, which constitute the learnable parameters of the model. Again $\odot$ is element-wise Hadamard product (Cho et al., 2014; Hewamalage, 2020; Dey and Salem, 2017).

From Equations 9-11 we get that GRU has total of $3(n^2 + nm + n)$ optimizable parameters. Again n is the dimension of hidden state and m is the dimension of input vector as in the LSTM. (Dey and Salem, 2017)

3. EXPERIMENTAL DESIGN

In this section, we present a detailed overview of the data, the architecture of the model built using the methodologies outlined in Section 2, and the comprehensive training process. These elements form the foundation for the CPU usage forecasting described in Section 4.

3.1 Dataset Description

The dataset contains resource usage traces of 1750 Virtual Machines from Bitbrains distributed data center (Bitbrains, 2013). The usage trace length is 1 month: from August 12, 2013 to September 11, 2013. The dataset has samples in 5 minute intervals. Bitbrains has customers from various industries, resulting diverse use cases and usage from one VM to another.

Table 1. Input variables from Bitbrains data

Name	Description	Unit
cpu_usage	Central processing unit usage	%
memory_usage	Memory usage	%
disk_read	Disk read throughput	KB/s
disk_write	Disk write throughput	KB/s
net_receive	Network received throughput	KB/s
net_transmit	Network transmitted throughput	KB/s

Dataset consists of two different sets. The first set contains data for 1250 VMs in fast storage area network (SAN) and second set contains data for 500 VMs from both SAN and slower network attached storage devices. Only the traces for VMs in fast storage area network were used in our experiments.

Many of the VMs had low or nearly static load throughout the trace period. This poses challenges for forecasting, as static CPU usage is trivial to forecast and would not accurately reflect the forecast accuracy. To address this issue, a subset of machines which had average CPU usage greater than 30% was selected. From this subset, five random machines were chosen for model training and forecasting to ensure a more representative evaluation of the methods. The five selected machines were identified by their ID numbers: 220, 242, 253, 269, and 283. All the features together with descriptions which were used in the forecasting models are defined in Table 1.

We also checked the correlations between all the variables from the SAN dataset, excluding the five machines used for forecasting to ensure no data leakage. These correlations are shown in Fig. 1. The target variable, CPU usage, had the strongest correlation with memory usage with Pearson's $r = 0.69$, while other input variables had relatively low correlation values of $r \leq 0.12$. There was moderate correlation between some of the input variables: Net transmit and net receive ($r = 0.51$), disk write and disk read ($r = 0.26$), and net receive and disk write ($r = 0.27$). These relationships are expected since network traffic tends to happen in both directions when communicating with the server, workloads can perform writes followed by subsequent reads (or vice versa), and in some applications, it makes sense to save the received data on disk.

Although CPU usage did not exhibit strong correlation with other input variables aside from memory usage, we chose to include all available features in the model. This decision was based on the understanding that correlations only measure linear relationship between variables and do not account for the time shifts essential in forecasting tasks (Hyndman and Athanasopoulos, 2019). Additionally, while there was some moderate correlation among a few input variables (e.g., net transmit and net receive, disk write and disk read), the level of multicollinearity was not substantial enough to warrant feature exclusion.

The rationale for incorporating all available features as input was to provide the model with diverse data, enabling it to capture different workload patterns more effectively. Consequently, we opted not to apply any feature selection techniques (such as principal component analysis) to the data. Instead, the correlation analysis served as an initial sanity check to ensure the validity of the data.

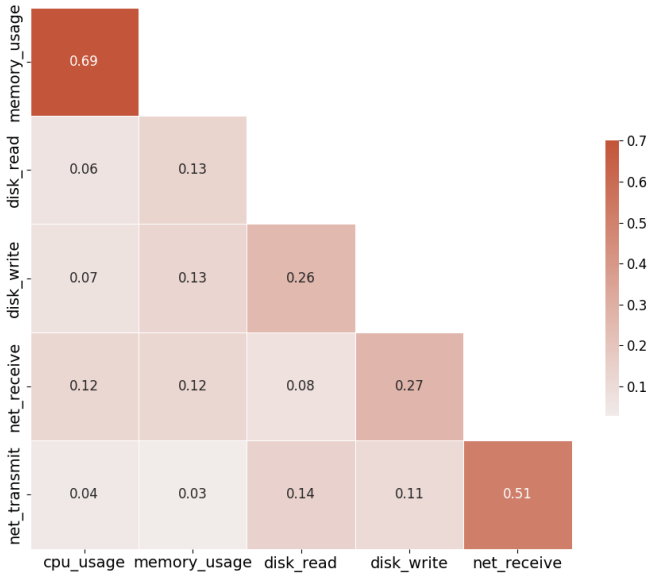

Fig. 1. Correlations between all input variables in the Bitbrains fast storage area network data. Machines used for forecasting have been excluded from the correlation analysis.

3.2 Model Architecture

The model architecture is presented in Fig. 2. Input consists of 90 timesteps of history data from 6 features which are described in Table 1. The model's output is a forecast of CPU usage for the subsequent 6 timesteps, which corresponds to a 30-minute forecast given the 5-minute data collection intervals. The choice of a 30-minute forecasting horizon was driven by a balance between operational practicality and predictive accuracy. In data center management, it is crucial to have a sufficiently long forecasting period to enable proactive measures and resource adjustments. A 30-minute horizon provides adequate lead time to implement necessary actions such as load balancing or resource allocation. Simultaneously, this period is short enough to maintain a high level of forecast accuracy, which tends to degrade over longer horizons. CPU usage was selected as the resource to forecast because it is typically regarded as the most critical resource in a data center due to its limited availability and high demand (Zharikov et al., 2020).

1D Convolution layer was used for enhanced temporal representation of time series, thereby potentially enhancing forecast accuracy. The convolution layer had kernel with length of 6, and 35 filters. While employing a substantial amount of filters can potentially diminish model efficiency, it significantly aids in capturing diverse features inherent in the time series data. This is discussed more in Section 4.2. Since kernel with length of 6 was used with 35 filters and stride of 1 output of 1D convolution layer has dimensions (85,35). This output is fed into the recurrent layer.

The recurrent layer in our model employs either GRU or LSTM RNN units, both renowned for their ability to capture temporal dependencies effectively. In both implementations the dimension of hidden state was deliberately set to a relatively high value of 1024, and this is same as the final output dimension of recurrent layer. This choice ensures that a comprehensive comparison of efficiency between the two RNN architectures can be conducted accurately. The rationale behind this decision lies in the understanding that a higher hidden state dimension necessitates more calculations, thereby ensuring that the results obtained are representative and robust. GRU implementation has less parameters to train than LSTM as described in Sections 2.3 and 2.4. This should make the GRU architecture more efficient than LSTM and real training time comparison between these two recurrent layers is presented in Section 4.2.

The Dense layer within the architecture requires an equal number of neurons as the desired forecast length. As detailed at the outset of this section, the forecast is for 6 subsequent timesteps. This ensures that the output dimensionality aligns precisely with the target forecast length.

Since all the 6 features were fed into 1D convolutional layer with kernel length of 6 it implies that each kernel had $6 \times 6 = 36$ weights to be optimized. 35 filters which each had its own bias was used. This makes $36 \times 35 + 35 = 1295$ trainable parameters total in the 1D convolution layer.

The number of trainable parameters in recurrent layer varies depending on the specific recurrent architecture employed. As told in Section 2.3 the LSTM has $4(n^2 + nm + n)$ optimizable parameters. Again n is the dimension of hidden state and m is the dimension of input vector. In

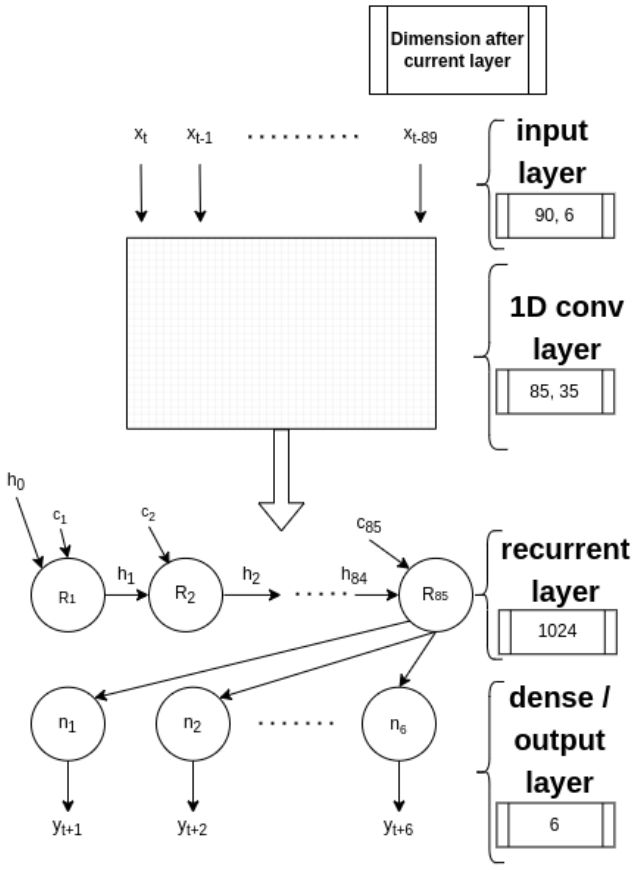

Fig. 2. Model architecture at timestep t. Dimension after each layer is shown on the right side.

this architecture this makes $4(1024^2 + 1024 \times 35 + 1024) = 4341760$ trainable parameters for the recurrent layer with LSTM. Since Tensorflow's CUDA implementation of GRU uses two bias terms in the reset gate, the total number of trainable parameters are $3(n^2 + nm + 2n)$, which in this architecture is $3(1024^2 + 1024 \times 35 + 2 \times 1024) = 3259392$ trainable parameters when using GRU on recurrent layer.

The dense layer of architecture has $n_{output}(n_{input} + 1)$ parameters to be optimized. In this architecture it means $6(1024 + 1) = 6150$ trainable parameters.

All these layers combined makes ∼3.27M trainable parameters in architecture with GRU units in recurrent layer and ∼4.35M trainable parameters with LSTM units.

3.3 Model Training Process

All models (GRU, LSTM, and ARIMA) were trained on data set consisting of the first 75% of data from each VM. Example of train and test split can be seen in Fig. 3.

In the training process of RNN architectures TensorFlow (Abadi et al., 2015) version 2.5 was used. 20% of the training data was further split for the validation of model and hyperparameter tuning during the training process. The hyperparameter tuning for the RNN architectures was conducted empirically through trial and error. We experimented with various combinations of key hyperparameters, including the number of layers, number of units per layer, and learning rate. The weights that yielded the highest forecast accuracy on the validation set were saved, and then later used to forecast the test in the model comparison in Section 4.1.

For the RNN architectures, min-max normalization was applied to scale all input variables to the [0, 1] range. Minimum and maximum values for each VM were calculated from the training set, and these same values were used to scale both training and test set.

The Adam optimizer combined with mean square error (MSE) loss function was used to update the weights of the RNN during the training process. Adam was chosen as optimizer because it is well suited for architectures with large amount of optimizable parameters (Kingma and Ba, 2014). The loss in both training and validation set was monitored through the training process. Example of the learning curve for the machine ID 283 can be seen in Fig. 4.

ARIMA model was optimized and trained using the pmdarima (Smith et al., 2017) Python package. ARIMA model parameters (p, d, q) were initially optimized using the algorithm specified in Hyndman and Khandakar (2008). Subsequently, the model was trained for each VM using its training data. The maximum depth for parameters were set to the default settings of the pmdarima package, specifically (5, 2, 5). The detailed ARIMA model parameters used for each VM are provided in Table 2.

For both recurrent neural networks and ARIMA model, a distinct model was trained for each machine. In Section 4.1, the forecast experiments and results of the trained models are evaluated using the held-out test set of future time series points, as illustrated in Fig. 3.

Table 2. ARIMA model parameters for each machine

Machine ID	Parameters		
	p	d	q
220	5	1	3
242	3	1	2
253	2	1	3
269	2	1	3
283	3	1	3

Source codes and demonstration for dataset selection, ARIMA model parameter optimization and fitting, as well as GRU architecture training and forecasting are publicly available (Malin, 2024).

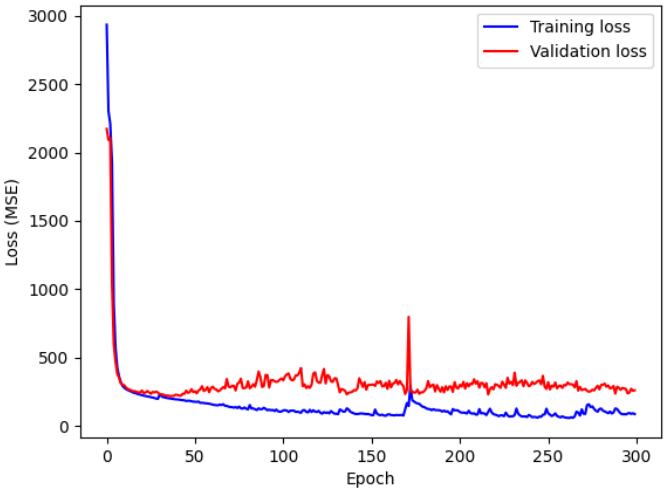

Fig. 4. Learning curve for machine ID 283.

4. RESULTS

In this section, we will evaluate the performance of the methods introduced in Section 2 using resource usage data from real-world scenarios and training proces introduced in Section 3. This analysis aims to demonstrate the efficiency and practical applicability of the forecasting techniques in a data center environment.

4.1 Forecast Accuracy

In our experiments, held out test set containing last 25% of data as described in Section 3.3 was forecasted. Root mean square error (RMSE), Mean Square Error and 95th percentile of the absolute error (AE) were calculated over the forecast results for the entire test set. The results for forecast accuracy are shown in Fig. 5 and Table 3.

Prior analysis of the data provided critical insights into the problem. During this process, we discovered that some CPU usage values slightly exceeded 100%. Based on this observation, we decided to clip the forecasts of all models within the [0, 105] interval to eliminate unreasonable forecasts.

The forecast results for each VM are presented in Table 3. Error metrics were calculated from the forecast of future CPU usage in the held out test set. For every error metric, the GRU architecture achieved the best average forecast

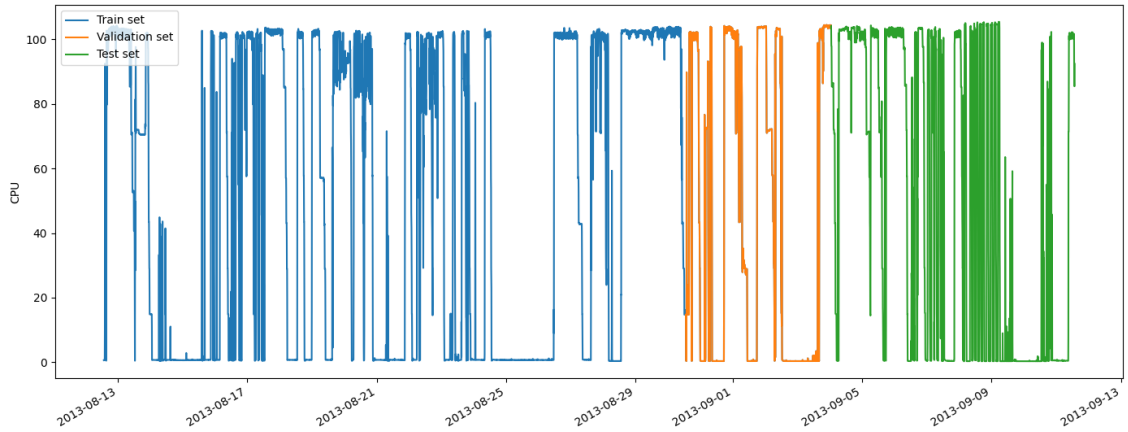

Fig. 3. Data set split example with CPU data from machine ID 253. 75% of the data was used in the training process. Last 20% of the training data was still used as a validation set to monitor metrics during training. Last 25 % of the whole dataset was held out for test set as a comparison between LSTM, GRU, and ARIMA.

Table 3. Forecasting errors for the test set of each machine. The model with the best accuracy (lowest error metric) is bolded, and the model with the lowest accuracy (highest error metric) is italicized. The mean and standard deviation are highlighted in the same manner

Machine	**RMSE**			**MSE**			**95th AE percentile**		
ID	**GRU**	***LSTM***	***ARIMA***	**GRU**	**LSTM**	**ARIMA**	**GRU**	**LSTM**	**ARIMA**
220	**28.72**	29.09	*30.80*	**825.05**	846.21	*948.46*	**66.38**	71.95	*84.16*
242	**25.49**	26.74	*26.81*	**650.11**	714.90	*718.69*	**62.75**	65.16	*71.29*
253	**23.14**	23.42	*24.92*	**535.55**	548.49	*621.13*	64.84	**62.63**	*71.60*
269	*20.04*	***19.37***	*19.78*	401.71	***375.11***	*391.31*	48.21	***44.48***	*47.13*
283	**20.06**	*21.00*	20.34	**402.47**	*441.01*	413.75	**48.94**	49.14	*53.30*
Mean	**23.49**	23.92	*24.53*	**562.98**	585.14	*618.67*	**58.22**	58.67	*65.50*
Std	**3.71**	4.00	*4.60*	**179.44**	194.44	*230.46*	**8.90**	11.47	*15.04*

performance across all machines, with an RMSE of 23.49, an MSE of 562.98 and a 95th percentile AE of 58.22. Conversely, the ARIMA model had the worst average error metrics, with an RMSE of 24.53, an MSE of 618.67 and a 95th percentile AE of 65.50. The LSTM architecture's performance fell in between these two, with average error metrics of 23.92, 585.14, and 58.67, respectively. Both the LSTM and the GRU architectures outperformed the baseline ARIMA model in forecast accuracy for every VM and across all metrics, as shown in Table 3. Additionally, when comparing the medians of the error metrics, the RNN architectures outperform the baseline ARIMA model. This can be seen from the boxplots of the error metrics in Fig. 5.

Considering all the calculated forecast error metrics in Table 3, the GRU architecture provided the lowest error in 11 out of 15 cases. In the remaining four cases where the GRU did not achieve the lowest forecasting error, the best-performing model was still an RNN architecture, specifically the LSTM. This highlights the superior performance of RNN-based models in our forecasting task.

4.2 Computational Efficiency

Both RNN architectures (GRU and LSTM) were trained using the NVIDIA Tesla V100-SXM2-32GB in the supercomputer Puhti at CSC – IT Center for Science. Models with architectures as described in Section 3.2 were trained for 100 epochs.

Both RNN models were trained on the same data set from machine id 357. 80% of the data was allocated for the training process, while the remaining 20% was reserved for calculating validation metrics. Although the results gathered during validation were not applied in this setting, the validation step was performed to accurately simulate the real training process of neural networks.

The total training time for the architecture with GRU units in the recurrent layer was 380.27 seconds, whereas for the architecture with LSTM units in the recurrent layer, it took 461.38 seconds. This observation supports the theory presented in Section 2.4 that GRU should be more efficient to train than LSTM.

Further improvements in efficiency could be achieved by further optimizing the convolution layer of the architecture. Methods such as max pooling in the convolution

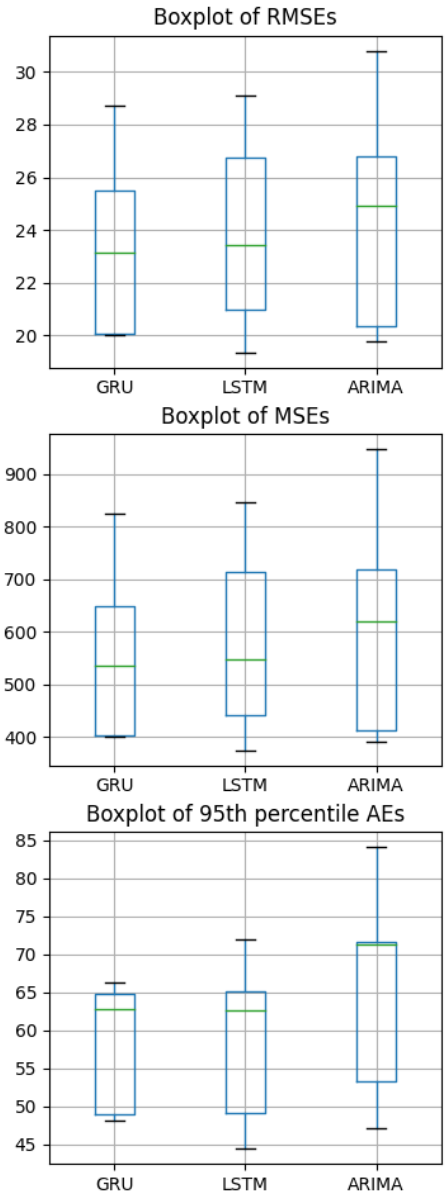

Fig. 5. Boxplot of all error metrics for the test set forecasts across different model architectures. The boxplot data holds results from all five machines used in the experiments.

layer can significantly reduce the dimensionality between the input and recurrent layers, thereby decreasing the number of weights to be tuned in the recurrent layer. However, despite initial optimization efforts, experimentation revealed that additional dimensionality reduction techniques resulted in a noticeable decrease in forecast accuracy. Therefore, while acknowledging the potential for further optimization, we decided not to pursue these techniques in this study.

Employing computationally efficient models, such as GRU, is crucial in data center load prediction. One key objective in this setting is to reduce the overall energy consumption of the computing cluster. Given that the forecasting models themselves are integral components of the system, using efficient models helps minimize their computational overhead, thereby contributing to energy savings.

4.3 Combining Accuracy and Efficiency

Table 4.3 presents a summary of the accuracy and efficiency results. The GRU model achieved a 9% reduction in forecast error, as measured by MSE, compared to the baseline ARIMA model. Additionally, the GRU model outperformed the LSTM model in terms of accuracy, despite having 25% fewer trainable parameters.

Table 4. Summary of all results. Average MSE and computational complexity is compared to the baseline ARIMA model. Rank takes both accuracy and computational complexity into account

Architecture	**Average MSE**	**Computational Complexity**	**Rank**
ARIMA	618.67 (Baseline)	Low	3.
LSTM	585.14 (-5.42%)	High	2.
GRU	562.98 (-9.00%)	Medium	1.

5. CONCLUSIONS

We have shown practical and accurate approach for forecasting IT resource usage in data centers using recurrent neural networks. In our experiments, both the LSTM and GRU architectures outperformed the baseline ARIMA model in terms of forecast accuracy.

The architecture with GRU units in the recurrent layer provided the best forecasting accuracy, as evidenced by the lowest mean and median error metrics. The LSTM units in the recurrent layer achieved the second-best forecast accuracy based on mean error metrics for the test set.

Considering that the GRU architecture not only delivered superior forecasting accuracy but also demonstrated efficient training times, we propose the use of GRU units in the recurrent layer of neural network architectures for resource usage forecasting. Reducing computational complexity in data center operations has significant practical implications. Improved forecasting accuracy and efficiency can enhance resource management, reduce energy consumption, allow dynamic scalability, and improve overall operational efficiency. These advancements can lead to cost savings and a lower environmental footprint, which are critical considerations in the field of sustainable computing.

This study provides valuable insights into forecasting data center load using RNN architectures. However, it is important to note that this work is based on the dataset from a single data center. Although the Bitbrains dataset has diverse use cases, this may limit the generalizability of the results. Future research should consider testing the models on datasets from multiple data centers to validate and potentially extend the applicability of the findings.

All the source codes for this work are publicly available (Malin, 2024), ensuring that all results can be reproduced. This will also enable future improvements to be built upon this base convolution and RNN architecture.

ACKNOWLEDGEMENTS

The work for this paper was done as part of ArctiqDC project, with financial support from the European Regional Development Fund via the Interreg Nord program. Authors would also like to thank IT Center for Science (CSC) for providing the computing resources.

REFERENCES

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. URL `https://www.tensorflow.org/`. [Online; accessed 28.5.2024].

Bayati, L.M. (2018). Power management policy for heterogeneous data center based on histogram and discrete-time mdp. *Electronic Notes in Theoretical Computer Science*, 337, 5–22. doi:10.1016/j.entcs.2018.03.031.

Bitbrains (2013). Gwa-t-12 bitbrains dataset. URL `http://gwa.ewi.tudelft.nl/datasets/gwa-t-12-bitbrains`. [Online; accessed 24.5.2024].

Bourhnane, S., Abid, M.R., Lghoul, R., Zine-Dine, K., Elkamoun, N., and Benhaddou, D. (2020). Towards green data centers. In J.L. Afonso, V. Monteiro, and J.G. Pinto (eds.), *Sustainable Energy for Smart Cities*, 291–307. Springer International Publishing, Cham. doi:10.1007/978-3-030-45694-8_23.

Calheiros, R.N., Masoumi, E., Ranjan, R., and Buyya, R. (2015). Workload prediction using arima model and its impact on cloud applications' qos. *IEEE Transactions on Cloud Computing*, 3(4), 449–458. doi:10.1109/TCC.2014.2350475.

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1724–1734. Association for Computational Linguistics, Doha, Qatar. doi:10.3115/v1/D14-1179. URL `https://aclanthology.org/D14-1179`.

Dey, R. and Salem, F. (2017). Gate-variants of gated recurrent unit (gru) neural networks. 1597–1600. doi:10.1109/MWSCAS.2017.8053243.

Goodfellow, I.J., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press, Cambridge, MA, USA. URL `http://www.deeplearningbook.org`.

Hewamalage, H. (2020). Recurrent neural networks for time series forecasting: Current status and future directions. *International Journal of Forecasting*, 37, 388–427. doi:10.1016/j.ijforecast.2020.06.008.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9, 1735–1780. doi:10.1162/neco.1997.9.8.1735.

Hyndman, R. and Athanasopoulos, G. (2019). Forecasting: principles and practice, 3rd edition. URL `https://otexts.com/fpp3/`. OTexts: Melbourne, Australia. OTexts.com/fpp3. [Online; accessed 13.08.2024].

Hyndman, R. and Khandakar, Y. (2008). Automatic time series forecasting: The forecast package for r. *Journal of Statistical Software*, 26, 1–22. doi:10.18637/jss.v027.i03.

Janardhanan, D. and Barrett, E. (2017). Cpu workload forecasting of machines in data centers using lstm recurrent neural networks and arima models. *2017 12th International Conference for Internet Technology and Secured Transactions (ICITST)*, 55–60. doi:10.23919/ICITST.2017.8356346.

Jones, N. (2018). How to stop data centres from gobbling up the world's electricity. *Nature*, 561, 163–166. doi:10.1038/d41586-018-06610-y.

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

Kiranyaz, S., Avci, O., Abdeljaber, O., Ince, T., Gabbouj, M., and Inman, D.J. (2021). 1d convolutional neural networks and applications: A survey. *Mechanical Systems and Signal Processing*, 151, 107398. doi:10.1016/j.ymssp.2020.107398.

Kumar, J. and Singh, A.K. (2020). Cloud datacenter workload estimation using error preventive time series forecasting models. *Cluster Computing*, 23(2), 1363–1379. doi:10.1007/s10586-019-03003-2. `doi:10.1007/s10586-019-03003-2`.

Malin, M. (2024). Demo for arima and gru model training. URL `https://github.com/miikamal/dc-rnn`. [Online; accessed 16.08.2024].

Masanet, E., Shehabi, A., Lei, N., Smith, S., and Koomey, J. (2020). Recalibrating global data center energy-use estimates. *Science*, 367(6481), 984–986. doi:10.1126/science.aba3758.

Ouhame, S., Hadi, Y., and Ulah, A. (2021). An efficient forecasting approach for resource utilization in cloud data center using cnn-lstm model. *Neural Computing and Applications*, 33. doi:10.1007/s00521-021-05770-9.

Shaw, S.B., Kumar, C., and Singh, A.K. (2017). Use of time-series based forecasting technique for balancing load and reducing consumption of energy in a cloud data center. In *2017 International Conference on Intelligent Computing and Control (I2C2)*, 1–6. doi:10.1109/I2C2.2017.8321782.

Shehabi, A. et al. (2016). United states data center energy usage report. no. LBNL-1005775. Technical report, Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States).

Smith, T.G. et al. (2017). pmdarima: Arima estimators for Python. URL `http://www.alkaline-ml.com/pmdarima`. [Online; accessed 27.5.2024].

Yuan, H., Bi, J., Li, S., Zhang, J., and Zhou, M. (2024). An improved lstm-based prediction approach for resources and workload in large-scale data centers. *IEEE Internet of Things Journal*, 1–1. doi:10.1109/JIOT.2024.3383512. Early Access.

Zhang, W., Li, B., Zhao, D., Gong, F., and Lu, Q. (2016). Workload prediction for cloud cluster using a recurrent neural network. 104–109. doi:10.1109/IIKI.2016.39.

Zharikov, E., Telenyk, S., and Bidyuk, P. (2020). Adaptive workload forecasting in cloud data centers. *Journal of Grid Computing*, 18. doi:10.1007/s10723-019-09501-2.