Enhanced Auto-scaling by using Dynamic scaling policy with Deep learning :LSTM

MSc Research Project
MSc in Cloud Computing

Mohith Srivathsa Ravikumaraiah

Student ID: 20218940

School of Computing
National College of Ireland

Supervisor: Jitendra Kumar Sharma

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Mohith Srivathsa Ravikumaraiah
Student ID:	20218940
Programme:	MSc in Cloud Computing
Year:	2021-22
Module:	MSc Research Project
Supervisor:	Jitendra Kumar Sharma
Submission Due Date:	15/08/2022
Project Title:	Enhanced Auto-scaling by using Dynamic scaling policy with Deep learning :LSTM
Word Count:	6060
Page Count:	17

Signature:	Mohith Srivathsa
Date:	15th August 2022

Enhanced Auto-scaling by using Dynamic scaling policy with Deep learning :LSTM

Mohith Srivathsa Ravikumaraiah
20218940
MSc in Cloud Computing

15th August 2022

Abstract

Cloud computing is becoming the main pillar of strength for most of the new technologies because of its ability to scale and change. Auto-scaling systems are being used to make this kind of flexibility on demand possible. When it comes to AWS auto scaling it has the EC2 autoscaling policies. For ensuring the exact number of available EC2 instances correctly, EC2 auto scaling groups are used. this Autoscaling groups will also help the application to handle the incoming loads. In this paper, an auto-scaling system with deep learning technology is proposed. AutoScaling best steps are chosen based on what needs to be done. In this paper, a deep learning method of enhancing auto-scaling groups is implemented. In the initial stage, different deep learning methods are applied to the dataset and the results are evaluated. From the result, LSTM found to be the most appropriate method when compared to the BI-LSTM and Attention BI-LSTM methods. So, the Auto scaling groups are triggered by using this method.

Contents

1 Introduction

Cloud computing lets you use datacenter resources like servers, applications, networks, and storage when you need them. Infrastructure as a service (IaaS) Cloud providers make sure that the quality of service is stable by using auto-scaling mechanisms that are based on triggers to handle variable workloads. However, it takes several minutes for scaling actions to take effect. When business-critical applications are put into the cloud, they are often set up with more resources than they need to avoid the auto-scaling mechanism, even though the auto-scaling decisions are often bad or come at the wrong time.[Camelion] Most auto scaling methods can be put into two groups: reactive and proactive. Reactive method always checks to see if certain metrics, like CPU usage, are above their threshold and if virtual machines need to be set up to meet the demand. The approaches which are Reactive uses metrics that have been seen recently to make decisions about planned scaling and the current state of the cluster. As part of the proactive methods, the web application's load is predicted and modelled, and machine learning techniques are used to find patterns in data from the past. These approaches are hard to tune and don't work well with workloads that change quickly and in unexpected ways. Because the workload changes from run to run, proactive approaches need to be updated all the time. This is especially important when trying to predict the future and see if the growing complexity of the model will be too much for the cluster's resources. Most organisations and companies even now prefer reactive methods because they work well and are easy to use. [1]

Research Question: Can a Dynamic scaling policy in AWS autoscaling group efficiently predict Auto-scaling in Public cloud using LSTM, Bi-LSTM, and Attention Bi-LSTM methods of Deep learning technique?

1. Autoscaling: Autoscaling is a cloud computing technique that is used to allocate computing resources in a way that is always changing. That automatically grows or shrinks based on how much traffic there is.

2. Deep learning is a class of machine learning that falls outside the scope of artificial intelligence. It is able to learn patterns that will inurn helps to make decisions and the processing of raw data in an effective and efficient manner.

3. Long short-term memory, also known as LSTM, is a subset of the recurrent neural network.

4. AWS Dynamic scaling policy: It is one of the 3 scaling policy in the Autoscaling group of EC2 Autoscaling in AWS Management console

Auto scaling combined with deep learning techniques was chosen as a research area because scalability is seen as one of the most important features of cloud services and infrastructure. Because customers changes so quickly, a data centre that was perfect one moment may never be usable the next. By changing the vm (Virtual Machines) on the fly, which IaaS auto scaling technology lets you do, until there is no more space on the hardware of the cloud. Auto scaling is used a lot now to find the right balance between cutting costs and improving the quality of very high-level cloud services.[2]

Auto scaling is a method for distributing services and setting them up in a way that changes as the workload does. The goal of auto-scaling is to avoid situations where there are too many or too few resources. It could waste resources that are used to run services, or it might not give the performance that was expected. [3] All of these problems are easy to solve with the help of modern deep learning technologies, which help auto scaling to be performed in very better way.

The time series dataset is taken as the input and the data is initially analyzed through the exploratory data analysis also the data is pre-processed. Three different deep learning models are considered in this paper namely LSTM, BI-LSTM, and Attention BI-LSTM. these 3 deep learning models are being compared for bringing out the best prediction method to be chosen for auto-scaling. Based on the experimental results, one of the deep learning methods is used for triggering the auto-scaling in the console. Initially, the launch template is configured in EC2. The application load balancer is attached to the auto scaling group and a new auto-scaling group is created for this purpose.

For the newly created instance, an SSH connection is established, and further, the files and all the dependencies are uploaded into the EBS volume. For forecasting the data the lstm.py file is used. By using the algorithm which is used in the program, it triggers the auto-scaling Group for performing the predictions when the CPU load is increased. Finally, based on the input which is given as the load the CPU utilization takes place which leads to effective auto-scaling. Cloud watch is used for monitoring all these activities on EC2 and auto-scaling group.

This document consists of seven different sections. Introduction has been given in the first section.

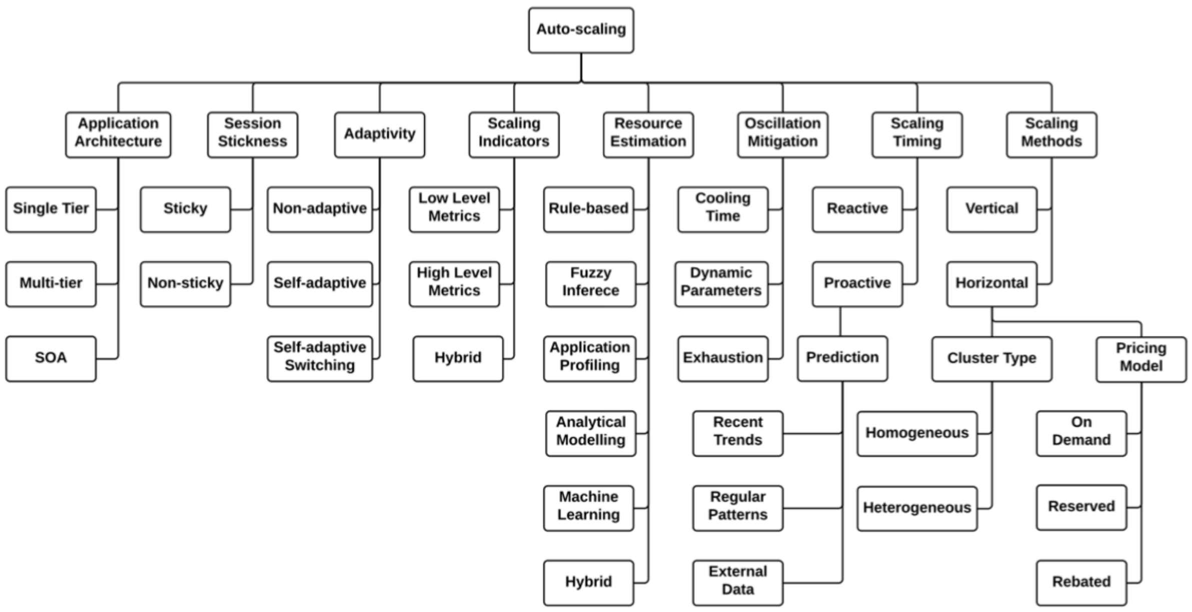


Figure 1: Proposed Architecture [4]

In second section detailed summary of all the literature work is present. The third and fourth section consists of the details regarding the methodology and design specifications. Implementation and evaluation of results are discussed in the fifth and sixth sections respectively. Finally, the brief summary of the entire research is concluded in the last section.

2 Literature Review

2.1 Related work

In [5], the author surveys auto scaling techniques and their development for cloud web apps. Auto scaling can be abstracted using MAPE (monitor-analyze-planning-execution). In the loop, each phase's key problems are identified, and auto scaler taxonomy and properties are discussed. A detailed taxonomy categorises problems and their solutions. The conclusion analyses all existing auto scaling techniques with their disadvantages and advantages to help future researchers. This paper begins the literature review and further we discusses auto scaling methodologies.

At first, the author [6] used an unsupervised learning method to look at the logs of a web application to find space partitions of a uniform resource identifier that were based on the size of the document and how long it took to respond. After that, the distribution of request arrival rate is worked out for each URI (Uniform Resource Identifier) application. Then, the pattern of probabilistic workload is made by using the URI distributions (PWP). In their proposed framework, they called the application's workload a "Probabilistic Workload Pattern." This helps to describe the application's workload and predict its future workload accurately.

This paper [7] solves auto scaling with hybrid method called Chameleon, a combination of proactive methods and reactive fallback. This method uses automated, time-series-based, on-demand forecasting to predict load intensity. Chameleon is benchmarked against proactive and reactive methods of auto scaling.They improved auto scaling's accuracy and timeliness. In the evaluation section, they compared their hybrid auto scaling chameleon model with user-oriented benchmark metrics. Experiments were run on private CloudStack, public AWS EC2, and OpenNebula, which uses the DAS-4 supercomputer based on IaaS.They measured with 5 real-world workloads. This CPU-intensive scenario involves enterprise Java.

This paper [8] uses machine learning plus proactive autoscaling for VNF auto scaling. Here, the author describes a dynamic traffic change. The proposed ML classifier can learn VNF scaling decisions from historic information. It manages network behaviour like seasonal+spatial traffic load for scaling decisions. ML auto scaling focuses on service quality and savings the cost. This paper's system converted the problem of Supervised ML classification into VFN auto scaling. With real network load traces, ML and ISP are compared. Machine learning was used to autoscale VNF. This study suggests exploring and operational cost and cost model in power usage, service deterioration, operating processes, and VNF management solutions.

The author demonstrated a fuzzy logic-based auto scaling approach for web applications [1]. Auto scaling was proposed to reduce resource consumption and improve SLAs. Real-life Wikipedia traces were used to evaluate AWS cloud performance. Using their reactive auto scaler, they reduced SLA violations and cloud resource utilization. Auto scaling dynamically adjusts threshold based on cluster load and cluster size. Their model doesn't require historical data or a long training period like machine learning does. Anyone can set up fuzzy linguistic rules using SLA metrics.More research is needed to determine the correlation between the proposed metrics for designing auto scalers. Auto scaling can help predict workload and save money.

In this paper, both the elasticity solutions and the classic solutions were looked at critically, and an overview of containerization, which is a lightweight virtualization technology [9], was also given. This paper gives a clear analysis of the ways in which elasticity is used in containers. The elasticity is divided up into seven groups: configuration, scope, mode, purpose, provider, method, and architecture.Elasticity, its concepts, and terms similar to scalability and efficiency are all talked about. Based on different proposals and analyses of cloud elasticity, the proposed classification has most of the features and sections.

This paper is a longer version of "Chameleon." In it, the problems with current auto scalers have been pointed out [10]. The model can be used to scale applications for multiple services in a coordinated way. They also test the Chamulteon auto-scaling model, which is focused on measurements in four sets with different environments, different resource needs, and traces of the real world. Overall, endosed elasticities metrics and user, Chamulteon's auto-scaling performance was the best.They make a comparison this auto scalar to different auto scalers and find that their model works best. There might be a need for a separate cost function for this.

Author proposes reinforcement learning (RL) solutions for container-based apps [11]. Controlling vertical and horizontal elasticity. This project aims to adapt to fluctuating workloads. The proposed RL solutions use model-based, Q learning, and dyna-Q to understand system dynamics. The proposed policies have been integrated through elastic docker and have self-adoption in docker swarm. Experiments on prototypes and simulations showed the effectiveness and flexibility of RL-based model policies.The design was a more general and flexible solution, and RL also suggested policies for controlling applications which is based on the elasticity of the container. The results of the evaluation show that solutions which are based on RL are useful and can be changed easily.

Intelligent decision-making engine based on artificial-neural-networks and metaheuristics to save user time and resources. Author [[12] proposes a software framework for solving complex real-world optimization problems. Open stack deployment results show that the proposed model of decision support engine could save optimization time and cost significantly. Cloud extension for large-scale optimization-based Generic algorithm framework. They distributed expensive fitness evolutions using IaaS instead of HPC. WoBinGO prototype is operationally capable, but its decision-making engine is lab-validated.

The PASCAL service is introduced [3] In this paper. It is proposed as part of the architecture of generic distribution . PASCAL stands for "Proactive Auto SCALing." This method combines the proactive approach with the forecast of incoming work to figure out how much provisioning is needed. Scale-in and scale-out operations are planned based on how each application's strategy works. A strategy was made for stream processing systems that automatically scale in a distributed way. Auto scaling is proactive for distributed data stores such as Apache Cassandra, and it concentrates on selecting when scaling actions run depending on how long it takes to turn storage nodes on and off so that configuration is ready when it's needed.

In this paper [13], the research on autoscaling methods which already did exist comprehensively and it can fit with the heterogeneous nature of applications for hybrid environments is discussed. In this paper, a method for choosing future workloads that is both efficient and predictive is provided. The the author goes on to say that because of the lack of productivity of cloud applications of workloads that are entirely different each time, the provisioning of automatically scheduled resources has to work effectively. They also planned to expand their research to the analysis of heterogeneous applications and the identification of variations with the help of deep learning models in workloads in order to provide solutions that are efficient for dynamic workloads.

For dealing with the cloud environment's fluctuating nature when it comes to resource requirements, author [14] used Scaling methods that are reactive. on demand Resource provisioning is an effective method for dealing with fluctuations in workload.To get around this problem, intelligent resource provisioning mechanisms allocate only the resources that are absolutely necessary based on their knowledge of the dynamic environment in which they operate. RLPAS is the algorithm that was proposed for this project (reinforcement-learning-based-proactive- auto-scaler). The RLPAS approach is used in a virtualized resource cloud environment.

In this work, the author [15] proposes auto scaling method of predictive to detect bursts when they occure in online for fulfilling fixed response time requirements. They are using the realistic bursty workloads to benchmark application performance and compare it to their auto scaling model. This benchmarking shows that the average response time decrement is violated just under 70% of the time. Auto scaling automatically detects workload spikes. they are using the burst detection to control application resources and speed up SLO responses. Their auto scaling approach boosts app performance during usage spikes. Using multiobjective optimization to minimise response-time and SLO costs.

Learning-based autoscaling framework that makes real-time vertical (resource pool) and horizontal (replicas) adjustments. The author [16] uses proactive auto scaling and an LSTM framework to reduce cloud-native application latency (long-short-term-memory). An algorithm prototype is shown. With learning-based LSTM forecast models, their framework can achieve golds in QoS bite states metrics of forecasting systems, such as load or request. Cloud application frameworks include control loops and self-healing. Using deep learning and machine learning to track data and cloud-native application adaptability is difficult. Proactive deep learning improves auto scaling.

This paper [17] presents a containerized method for vertical auto scaling machine learning services. Machine learning improves a phase discovery method based on Conditional-Restricted-Boltzmann-Machines to adapt to different workloads (CRBM). Using cyclical knowledge. When running certain workloads, containers frequently undergo phase transitions. Consequently, machine learning services may be used differently to manage resources. MLP to predict container-based workload faces. During prediction phases, predictive vertical auto scaling resizes containers dynamically. Machine learning workloads are used to test proactive auto scaling on containers. It maintains a consistent number of resizing operations, like the best reactive method.

In this paper[18] . Fluctuations in cloud user demand can increase energy consumption, reduce service quality and performance, and waste resources. The proposed framework solves this problem by consolidating workloads onto fewer energy-efficient physical machines based on application resource needs. This paper makes three contributions: First, an Online Multi-Resource Feed-forward Neural Network (OMFNN) predicts future application resource needs. The proposed framework is evaluated and compared to different scenarios using real-world workload traces from the Google cluster benchmark dataset.

This paper analyses the most effective auto scaling techniques for hybrid applications. In order to maximise resource utilisation, reduce costs, and meet service level agreements (SLAs), it is crucial to identify the optimal solutions for heterogeneous applications during auto-scaling. This paper explains auto scalers' scaling methods and strategies. A comprehensive categorization of auto-scaling techniques is provided. Unsupervised deep learning models can be used to examine time series prediction, they added. Deep learning can predict an application's future workload. Deep learning can help predict evolving workloads[19].

In [20], they propose a deep learning method for centralised and decentralised settings that can scale vertically and horizontally in multidomain networks. Using a real-world operator traffic dataset for testing, validating, and training, they model auto-scaling as a timeseriesforecasting problem with multi-step and one-step predictions of the future. Over a commercial data set from a network operator, they compare the performance of various deep-learning models and explore the merits and drawbacks of Federated learning versus centralised data analysis. They also compared native reactive and predictive auto scaling. Their prototype verifies the simulated results, as tested.

In this Paper [21]. First, they introduced an auto scaling model that can identify scaled microservices with the highest resource demand. Their generic model reduces Kubernetes HPA response time by 20%. Reinforcement learning improved auto scaling's value identification. The authors validated and trained reinforcement learning agents to horizontally autoscale microservices based on resource consumption. Because of microservices' varying resource demands and usage, auto scaling should be better understood. They used horizontal auto scaling and container-level vertical scaling. They compared vertical and horizontal auto scaling based on response time.

Time-series-forecasting, an automatic scaling technique, was used in this case to predict the workload resources. For auto-scaling in multivariate environments , BI-LSTM was proposed [22]. The proposed auto-scaling framework can be used in open stack, Q burnets, and other container and virtual machine environments. Using workload datasets that incorporate Materna and bitbrains data centre trace logs. In addition, they evaluates the performance of the proposed framework in this paper. The author also proposed a hybrid approach to auto scaling, combining proactive and reactive approaches, in order to improve future work with auto scaling.

2.2 Summary

From all the literature work which has been done on this topic, it is observed that most of the authors have been using reactive and proactive auto-scaling. Nowadays, the deep learning method is introduced to reactive and proactive auto-scaling to get more accurate results. Using complex algorithms for doing simple auto-scaling is quite time-consuming and not efficient. So, in this paper, an auto-scaling method using a deep learning technique is proposed in which the auto-scaling is triggered from a program that is by using CLI. The dataset is trained with the appropriate deep learning algorithm. then a deep learning algorithm is selected to trigger the auto-scaling group by establishing an SSH connection. In this way, the research is different from all the previous works which have been done before on auto-scaling. Later on, in the sections, a detailed explanation of implementation can be found.

3 Methodology

In this part of the article, we will talk about the various approaches that have been taken to develop the system which is proposed. Datasets, exploratory data analysis, data preprocessing, comparison of deep learning models, Different deep learning methods of LSTM and services used in AWS will be discussed. For the purpose of gaining a deeper comprehension of the concepts, an in-depth analysis of each technique will be carried out.

3.1 Datasets

The dataset used is Dataset.csv and DatasetForecast.csv which contains day, week, hour, minute, request, maxReq20minutes, minReq20minutes, meanReq20minutes of columns, and a total of 8064 rows of data and forcastshift.csv which contains forecast data in which both of these data set highly unpredictable.[23][24]

3.1.1 Data pre-processing

The wide variety of algorithms in machine learning including the neural network use the inputs which are in constant size and this is one of the main requirement. in order to create a number of input and output pairs in the form of vectors. For making a forecast first it is need to be given call the information

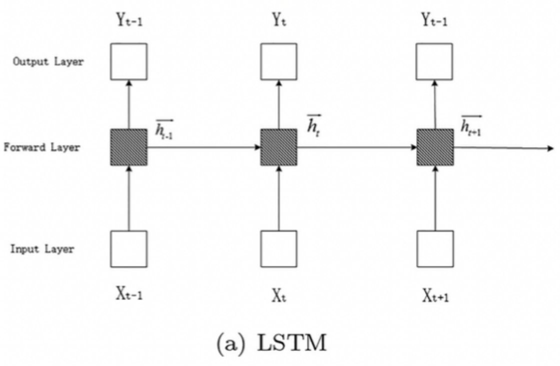


Figure 2: LSTM unit [4]

which is needed by the non-recurrent neural networks.Here, the advantage is any data which is present outside the window has no impact on the present forecast. Normalization for the selected time series data. the data is being pre-processed and ready to proceed with the further process.[25]

3.1.2 Exploratory Data Analysis

In this project, by performing exploratory data analysis for investigating and analyzing the data set and main-characteristics of data set are summarized with the methods of the data visualization. The method of EDA helps in manipulating the data sources and how best we can do it to get all the possible answers we might need. with the help of this we can discover patterns, check assumptions, test hypothesis or we can spot anomalies. once the process of EDA is completed and all the insights are being drawn, we will get the ability of performing very high sophisticated modeling or data analysis, or this data can be used to perform machine learning and deep learning.

The libraries I have used for performing EDA are Seaborn, Matplotlib and Pandas.

3.2 Deep Learning

Deep learning, also called DNN (deep-neural-network), is a type of machine learning AI that is able to mimic how the human brain learns patterns to make decisions and process raw data in a way that is similar to how the human brain works. FFNN (feed-forward-neural-network), RNN, CNN (conventional-neural-network) are three important parts of DNN (recurrent-neural-network). There is only one pass of data from the input layer to the output layer, so there is no need for a state memory. The simplest example of FFNN is the MLP (multi-layer Perceptron). CNN is the regularised version of MLP. RNN networks use state memory as well as cycles to process input sequences. In RNN, weights are shared across time, which makes it easy for damn dope to show patterns and process data in a way that makes sense. LSTM and GRU are both parts of RNN (Grated-recurrent-units). [20]

3.3 LSTM

LSTM A neural network with a hidden state that is activated by previous states is used after completing of data preprocessing. It will dynamically use contextual information and handle variable-length sequences. This is an RNN structure. Figure shows LSTM's four main components. The input gate, I controls the size of new memory decided to add to a memory block. The forget gate, f, determines how much memory to forget. Output gate (o) Output gate modulates memory output. Cell-activation vector Represented as c, it has two components: partially-forgotten-previous-memory (ct-1) and new memory modulated c t.*. Figure shows the LSTM mathematical model and the data processing of the hidden state ht for input xt. [26]

$$i_t = \sigma(W_{xi}x_t) + W_{hi}h_{t-1} + b_i \tag{1}$$

$$f_t = \sigma(W_{xf}x_t) + W_{hf}h_{t-1} + b_f \tag{2}$$

$$O_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \tag{3}$$

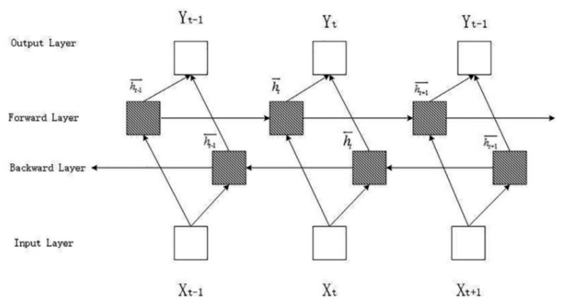


Figure 3: Bi-LSTM [4]

$$C'_t = tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{4}$$

$$C_t = f_t \bigotimes C_{t-1} + i_t \bigotimes C'_t \tag{5}$$

$$h_t = O_t \bigotimes tanh(C_t) \tag{6}$$

where it, value of I, similarly ft, represent f, ct and ot represent c and o, at moment t.
W= self updating weights(Hidden layer).
b = bias vector
(.)= sigmoid
tanh(.) = (function)hyperbolic tangent.
output range(hidden layer) = [0,1].
$\bigotimes = elementwise - multiplication$

3.4 Bi-LSTM

LSTM networks can exploit historical context. Without future context, problem meaning is unclear. For this, we use biLSTM (bidirectional-LSTM), which can succeed and proceed contexts by mixing forward and backward hidden layers (figure). Forward-Passover and backward-Passover are filtered in the identical wait 2 forward-passes plus

backward-passes of the regular network, each time stages through LSTM should reveal both backward plus forward hidden states. with time-based biLSTM back propagation training. [26]

3.5 Attention Bi-LSTM

Fusion of the BI-LSTM layer with the attention mechanism is used to find the correlation between the attention mechanism's final state and its intermediate states. This mechanism is used to keep information safe based on the problem of information redundancy and this is needed to solve it as much as possible. For the predicted output, the parameters of each layer of attention are given a weight. The Bi-LSTM model of attention is shown in the picture above. [4]

3.6 DEEP LEARNING RESULTS

As discussed, the deep learning methods which are used in this project are LSTM, BI-LSTM, and ATTENTION-BILSTM. All these three are being compared by using the different entities like MSE, RMSE, R – SQUARE, MAE, and MAPE. When all these values for each of the deep learning models are compared, it is clear that LSTM is the best suited for this project to perform the enhanced auto-scaling. The mean squared error in LSTM is much lesser when compared to the other two models of deep learning. when it comes to root mean square error it is the same. The detailed readings can be found in figure 5.

Based on these experimental results, LSTM method in deep learning is considered for the auto-scaling.

The graph in the figure 6 represents the comparison between the LSTM, BI-LSTM and ATTENTION BI-LSTM like mean square error, root mean square error and mean absolute error.

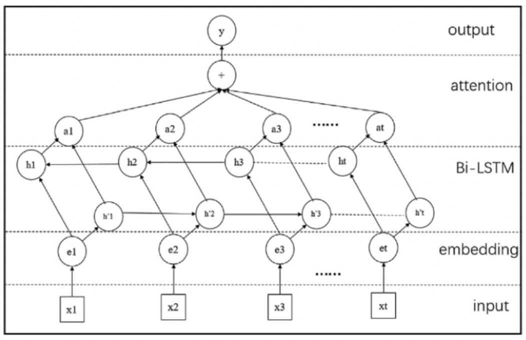


Figure 4: Attention Bi-LSTM [4]

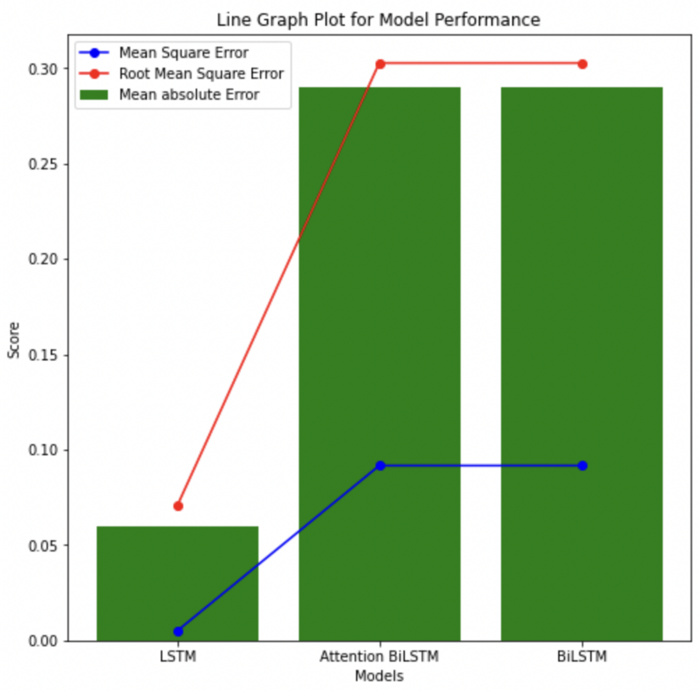
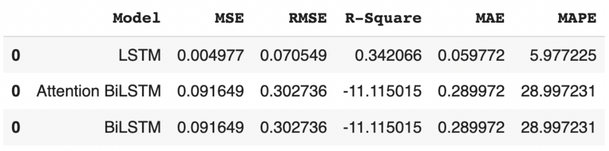


Figure 5: LSTM vs Bi-LSTM vs Attention Bi-LSTM [4]



	Model	MSE	RMSE	R-Square	MAE	MAPE
0	LSTM	0.004977	0.070549	0.342066	0.059772	5.977225
0	Attention BiLSTM	0.091649	0.302736	-11.115015	0.289972	28.997231
0	BiLSTM	0.091649	0.302736	-11.115015	0.289972	28.997231

Figure 6: LSTM vs Bi-LSTM vs Attention Bi-LSTM [4]

3.7 AWS Autoscaling group

Amazon EC2-Auto-Scaling ensures you have enough instances to handle application load. Auto-Scaling-groups contain EC2 instances. It Amazon-EC2-Auto-Scaling guarantees that never has fewer instances than you specify in Auto-Scaling-Group. It Amazon-EC2-Auto-Scaling ensures your group never exceeds the instances count you specify as maximum. It ensures that the specified capacity when you create it or later in your group. It can also launch/terminate instances as per the scaling policies.[27]

3.8 CloudWatch Metrics

Metrics measure system performance. Many services automatically provide resource metrics for diffrent AWS services including EC2 Auto-scaling. You can always enable the detailed monitoring for Amazon EC2-instances. Amazon CloudWatch can search, graph, and alarm on all your account's metrics (both AWS resource and application metrics). You can view up to the minute and historical metric data for 15 months. For analysing Autoscaling metrics I have used this AWS service.

4 Design Specification

This section lays out the details of how we planned and carried out our proposed project.

In the beginning, the dataset is chosen for the purpose of performing exploratory data analysis and data preprocessing. After that, I have given that data to three different deep learning models, such as LSTM, Bi-LSTM, and Attention Bi-LSTM, and evaluated which of these three models is superior for carrying out auto scaling and prediction of model in terms of actual data and predicted data. Finally, I came to the conclusion that the LSTM model was the most effective of the three. 7

After that, I attached the application load balancer to the autoscaling group that I'm going to create. I also configured a launch template in AWS EC2.
The autoscaling group was created with a desired capacity of 1, a minimum capacity of 1, and a maximum capacity of 3.
And I have also created the dynamic tracking policy in the auto scaling group. in the dynamic tracking policy I have chosen target tracking policy. If the CPU utilisation exceeds the threshold of 80 percent, the target tracking policy instructs the autoscaling group to allocate more resources to instances in accordance with the maximum capacity if those resources are required and also it will terminate the allocated instances when the CPU Usage become less. In this case, the desired capacity will always be dynamic because, as the load increases, so does the desired capacity; conversely, if the load decreases, so will the desired capacity will be decreases.

After that, the EC2 instance will be launched by my autoscaling group in accordance with the minimum capacity, which in my case is 1. And all of my data and dependencies are saved in the EBS volume that is attached to this specific EC2 instance, I have made sure that the termination protection is set to the true state for this particular instance.

After that, I used SSH to establish a connection to the specific instance. After which I created a new directory and uploaded all of the files and dependencies that will be kept in an EBS volume.

I have imported all of the dependencies that are necessary to run the LSTM.py Python file that I have created. This file will forecast the Load, and it will also increase the CPU load when it is performing the predictions, which will, in turn, increase the CPU utilisation. The target tracking policy will then trigger the autoscaling feature, which will add and remove instances based on the current level of CPU utilisation, with a minimum threshold of 80 percent.

The dashboard of cloud watch metrics enables monitoring, viewing, and reviewing of all these logs simultaneously. Also for better understanding ,

I have uploaded all of my files and dependencies to Google Drive and connected through Google Colab in order to have a better viewing experience of graphs and a greater understanding of the various

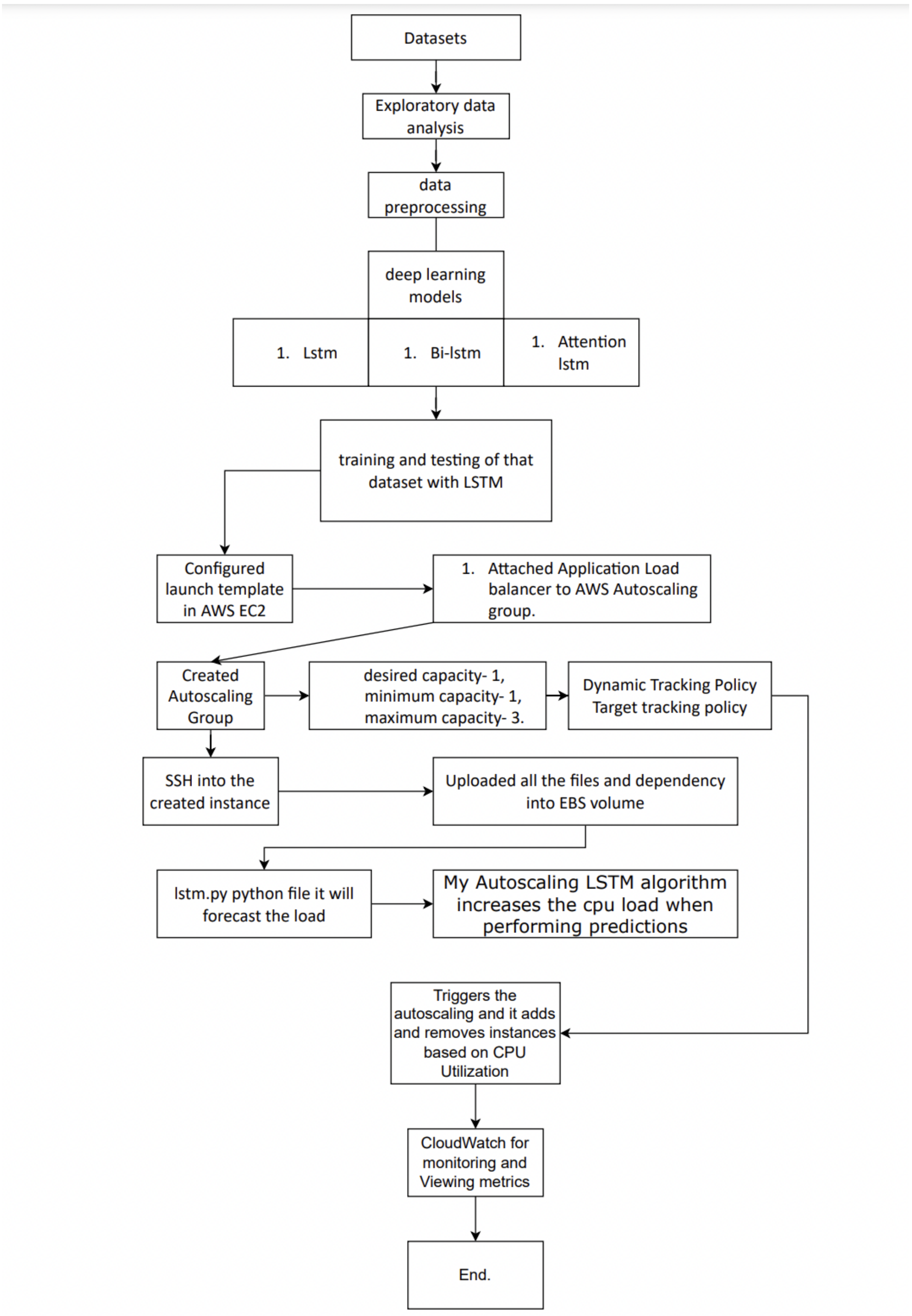

Datasets
Exploratory data analysis
data preprocessing
deep learning models
1. Lstm
1. Bi-lstm
1. Attention lstm
training and testing of that dataset with LSTM
Configured launch template in AWS EC2
1. Attached Application Load balancer to AWS Autoscaling group.
Created Autoscaling Group
desired capacity- 1, minimum capacity- 1, maximum capacity- 3.
Dynamic Tracking Policy Target tracking policy
SSH into the created instance
Uploaded all the files and dependency into EBS volume
lstm.py python file it will forecast the load
My Autoscaling LSTM algorithm increases the cpu load when performing predictions
Triggers the autoscaling and it adds and removes instances based on CPU Utilization
CloudWatch for monitoring and Viewing metrics
End.

Figure 7: Proposed Architecture [4]

metrics. This was done in order to have a better understanding of graphs and matrices.

4.1 LSTM-Time series forecasting

LSTM for Time-Series-Forecasting: The LSTM model, the data from input layer goes through three LSTM hidden-layers(48 hidden nodes per layer). This means that every input sub sequence (48 time) scales can be processed by making use of ReLU as activation-function in all of the nodes. After the output of a final LSTM hidden-layer, the input sequence summary is interpreted by two dense hidden-layers (48 and 24 nodes) , in both. The backpropagation-parameters are like the FFNN model, and the goal is to reach the local minimum to reduce the prediction error (lines 7-10).

Algorithm 1 TSF(Time-Series-Forecasting) LSTM

```
 1: h1, h2, h3 ← LSTM layer, h4, h5 ← dense layer;
 2: Dropout P robablity ← 0.5;
 3: d1,d2,d3,d4 ← 48, d5 ← 24;
 4: A1, A2, A3, A4, A5 ← ReLU;
 5: O←adam,L←huberloss,lr←0.01,E←200;
 6: train set ← base station 1+base station 2+base station 3;
 7: model ← def ine(h1 ...h5 , d1 ...d5 , A1 ...A5 );
 8: for i=1,iE,i=i+1 do
 9:     L(i)←compile(model,train set,metrics);
10:     new weights(i)←weight update(L(i),O,lr);
11:     update(model,new weights(i))
12: Return model;
```

5 Implementation

Python was used throughout the entire process of building the proposed system.

I used different Python libraries to simulate the system's internal parts. These include math for mathematical tasks, numpy for arrays, pandas for data analysis and machine learning tasks, sklearn module which incorporates a number of different loss, score, and utility functions in order to evaluate the effectiveness of classification, matplotlib- From Python we can create static, interactive and animated visualisations with the help of Matplotlib, which is a comprehensive library, keras as deep learning API, tensorflow as an open-source machine learning library, seaborn as a Python matplotlib library, statsmodels as a Python module that estimates statistical models, conducts statistical tests, and explores statistical data.

I was able to integrate my model with the AWS, and now my model is capable of predicting the values after being trained and tested on the data. In addition, the CPU load is increased while it is making data predictions. Therefore, if the CPU utilisation reaches the threshold of 80 percent, the target tracking policy of AWS auto scaling will allocate resources (launch m2.medium). Then the cloudwatch metrics will display all of the activity logs and these logs can be used for better understanding.

The following are the specifications of the machine that will run the developed system:

Operating System: Ubuntu, 22.04 LTS, amd64.

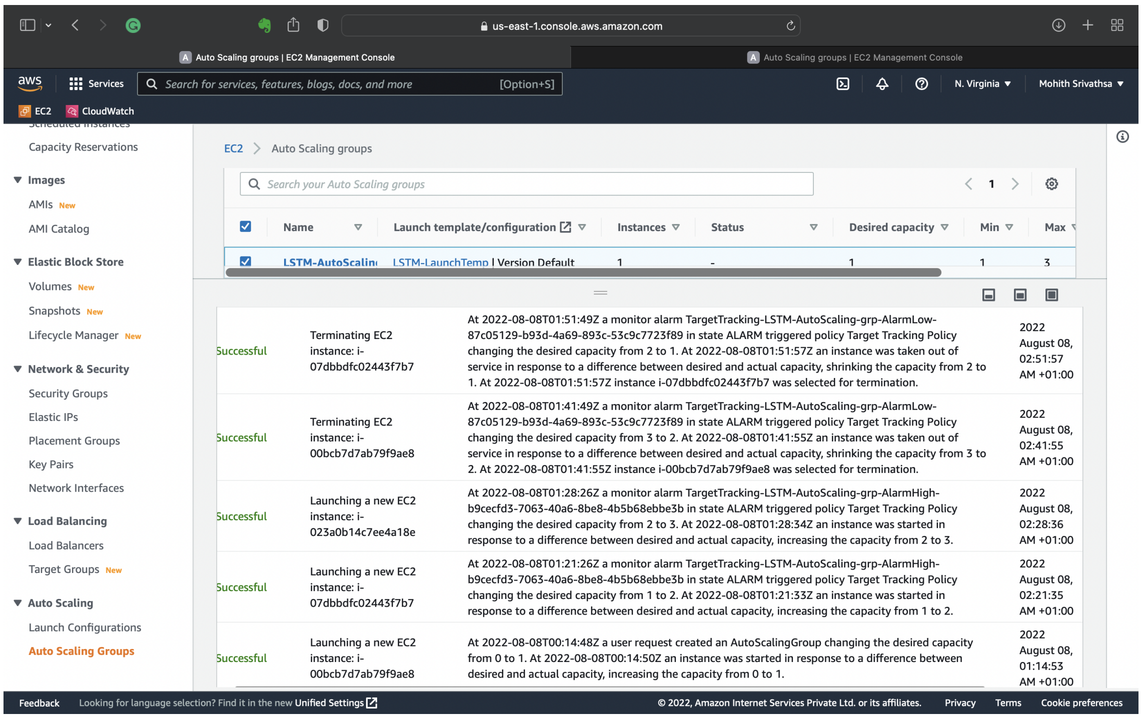
Instance type: t2.medium

Number of vCPUs: 2.

RAM: 4GB

Language : Python

Libraries : pandas, matplotlib, sklearn, math, numpy, matplotlib, keras, tensorflow, seaborn, statsmodels.



Name	Launch template/configuration	Instances	Status	Desired capacity	Min	Max
LSTM-AutoScalin	LSTM-LaunchTemp | Version Default	1	-	1	1	3

Successful	Terminating EC2 instance: i-07dbbdfc02443f7b7	At 2022-08-08T01:51:49Z a monitor alarm TargetTracking-LSTM-AutoScaling-grp-AlarmLow-87c05129-b93d-4a69-893c-53c9c7723f89 in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 2 to 1. At 2022-08-08T01:51:57Z an instance was taken out of service in response to a difference between desired and actual capacity, shrinking the capacity from 2 to 1. At 2022-08-08T01:51:57Z instance i-07dbbdfc02443f7b7 was selected for termination.	2022 August 08, 02:51:57 AM +01:00
Successful	Terminating EC2 instance: i-00bcb7d7ab79f9ae8	At 2022-08-08T01:41:49Z a monitor alarm TargetTracking-LSTM-AutoScaling-grp-AlarmLow-87c05129-b93d-4a69-893c-53c9c7723f89 in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 3 to 2. At 2022-08-08T01:41:55Z an instance was taken out of service in response to a difference between desired and actual capacity, shrinking the capacity from 3 to 2. At 2022-08-08T01:41:55Z instance i-00bcb7d7ab79f9ae8 was selected for termination.	2022 August 08, 02:41:55 AM +01:00
Successful	Launching a new EC2 instance: i-023a0b14c7ee4a18e	At 2022-08-08T01:28:26Z a monitor alarm TargetTracking-LSTM-AutoScaling-grp-AlarmHigh-b9cecfd3-7063-40a6-8be8-4b5b68ebbe3b in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 2 to 3. At 2022-08-08T01:28:34Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 2 to 3.	2022 August 08, 02:28:36 AM +01:00
Successful	Launching a new EC2 instance: i-07dbbdfc02443f7b7	At 2022-08-08T01:21:26Z a monitor alarm TargetTracking-LSTM-AutoScaling-grp-AlarmHigh-b9cecfd3-7063-40a6-8be8-4b5b68ebbe3b in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 1 to 2. At 2022-08-08T01:21:33Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 1 to 2.	2022 August 08, 02:21:35 AM +01:00
Successful	Launching a new EC2 instance: i-00bcb7d7ab79f9ae8	At 2022-08-08T00:14:48Z a user request created an AutoScalingGroup changing the desired capacity from 0 to 1. At 2022-08-08T00:14:50Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 0 to 1.	2022 August 08, 01:14:53 AM +01:00

Figure 8: The Activity log of AWS Autoscaling group

6 Results and Discussion

In the above figure we can see the complete logs recorded in the autoscaling groups of EC2 dashboard. Firstly, When I create this autoscaling group with minimum capacity of 1, desired capacity of 1, and maximum capacity of 3. It will launch the EC2 instance as per the minimum capacity mentioned while creating autoscaling group.

Firstly, 1st log can be seen in the bottom of the figure. it says autoscaling group will change the desired capacity to 1 from 0. So this instance will run until we change the minimum capacity to 0, or in the other hand, if the instance terminated by any chance, then autoscaling group will launch another instance in place of the unhealthy instance as per the launch configuration.

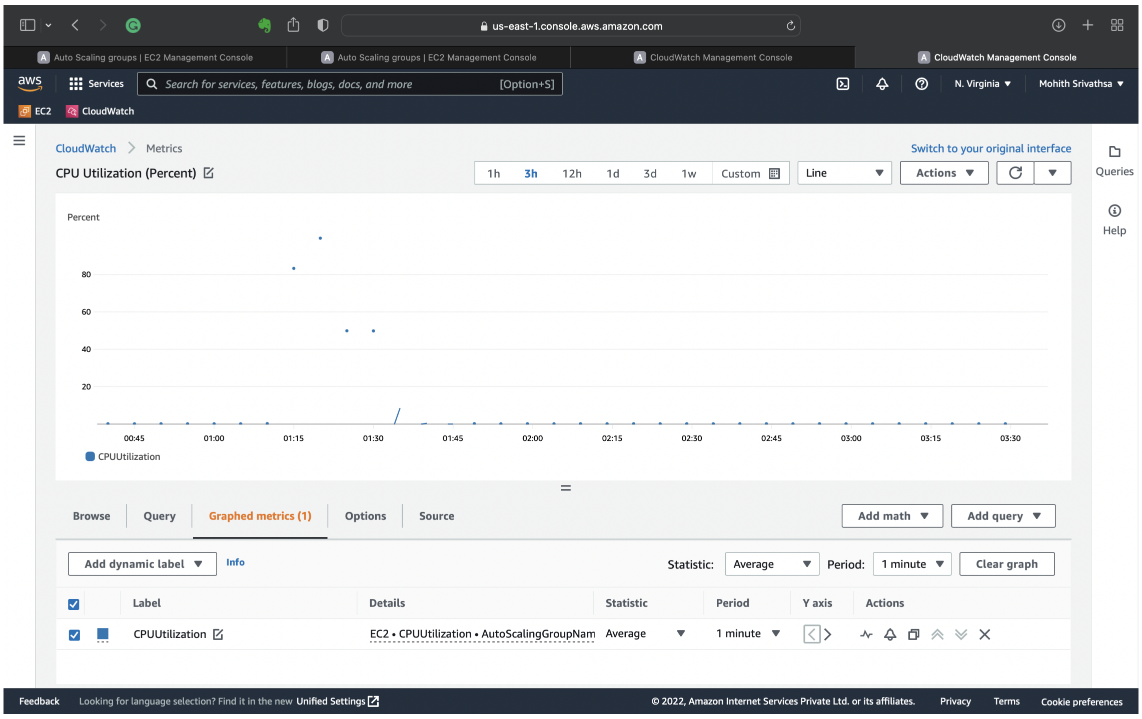
Secondly, I have done SSH into my created EC2 instance. and I have stored all the files and installed all the dependencies required to run my lstm.py file in the EBS volume. so when I run my Python file, it will give output of predicted values and it will increase the CPU load while doing predictions. this will trigger the dynamic scaling policy of autoscaling group. in that I have created the target tracking policy and here when the CPU load crosses the threshold of 80% it will change the desired capacity(1 to 2) and trigger the autoscaling group. in turn, autoscaling group will scale according to the target tracking policy(this will add/remove instances as specified in the auto scaling group's launch template).

Thirdly, the target tracking policy will change the desired capacity from 2 to 3. Then autoscaling group will launch the third EC2 instance because the alarm is still high.

Fourthly, after all the predictions done by LSTM the CPU utilization will become less than 80%, so the target tracking policy of autoscaling group will change the desired capacity from 3 to 2. so this will terminate the created EC2 instance for matching the desired capacity.

Lastly, the CPU utilization will be less than 80% again so, the target tracking policy of autoscaling group will modify the desired capacity from 2 to 1. again this will terminate the 2nd created EC2 instance as per the desired capacity.8

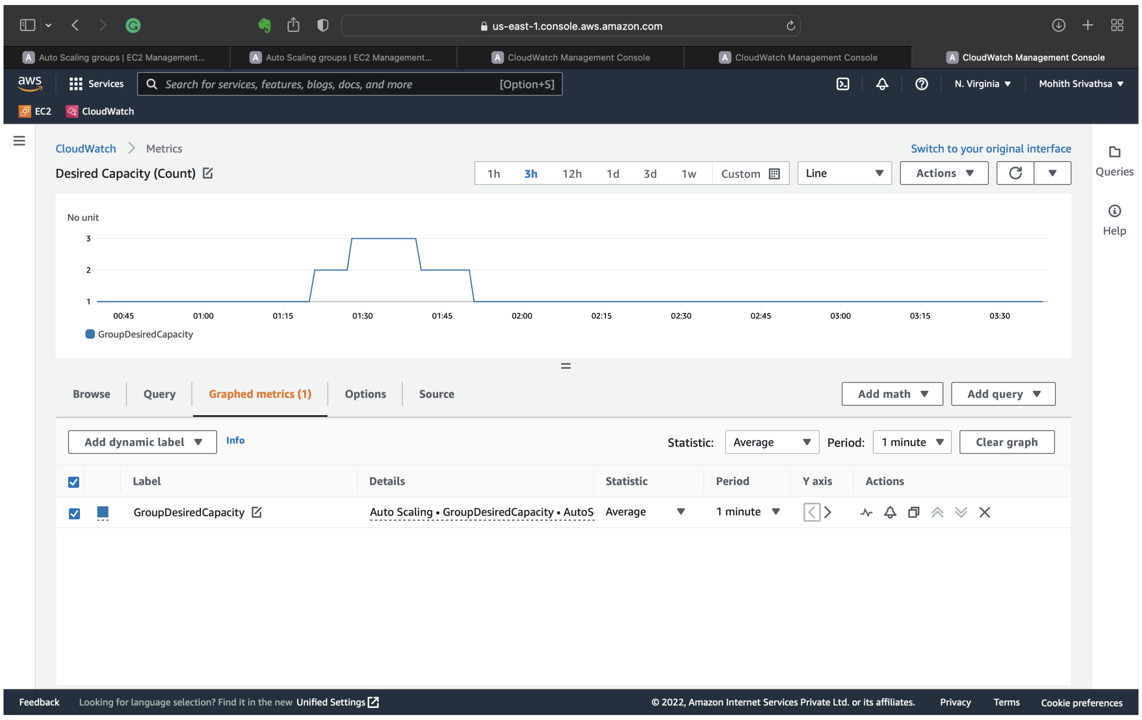
In the above figure9, it shows the cloud watch metrics of CPU utilization which is linked to my



Statistic: Average Period: 1 minute

Label	Details	Statistic	Period	Y axis	Actions
CPUUtilization	EC2 • CPUUtilization • AutoScalingGroupNam	Average	1 minute		

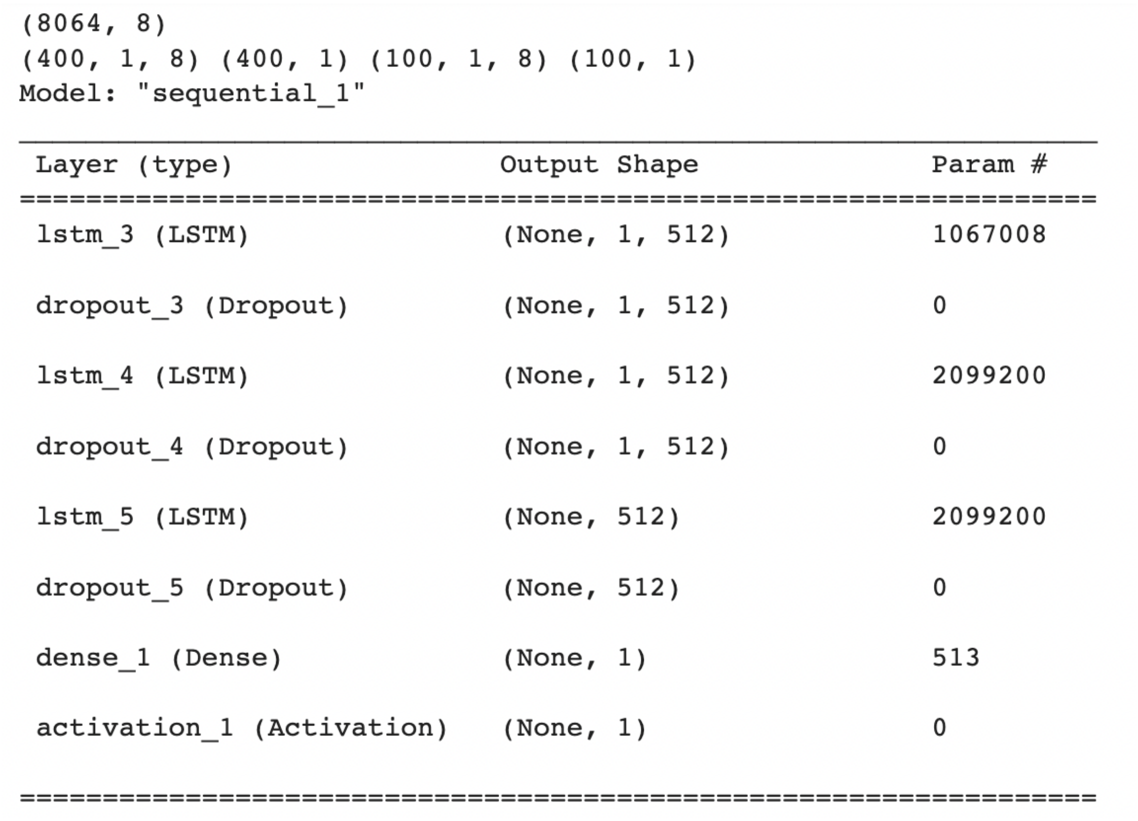
Figure 9: CPU utilization graph in cloudwatch metrics



Statistic: Average Period: 1 minute

Label	Details	Statistic	Period	Y axis	Actions
GroupDesiredCapacity	Auto Scaling • GroupDesiredCapacity • AutoS	Average	1 minute		

Figure 10: Desired capacity count



```
(8064, 8)
(400, 1, 8) (400, 1) (100, 1, 8) (100, 1)
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_3 (LSTM)               (None, 1, 512)            1067008

 dropout_3 (Dropout)         (None, 1, 512)            0

 lstm_4 (LSTM)               (None, 1, 512)            2099200

 dropout_4 (Dropout)         (None, 1, 512)            0

 lstm_5 (LSTM)               (None, 512)               2099200

 dropout_5 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 1)                 513

 activation_1 (Activation)   (None, 1)                 0

=================================================================
```

Figure 11: LSTM Sequential model output

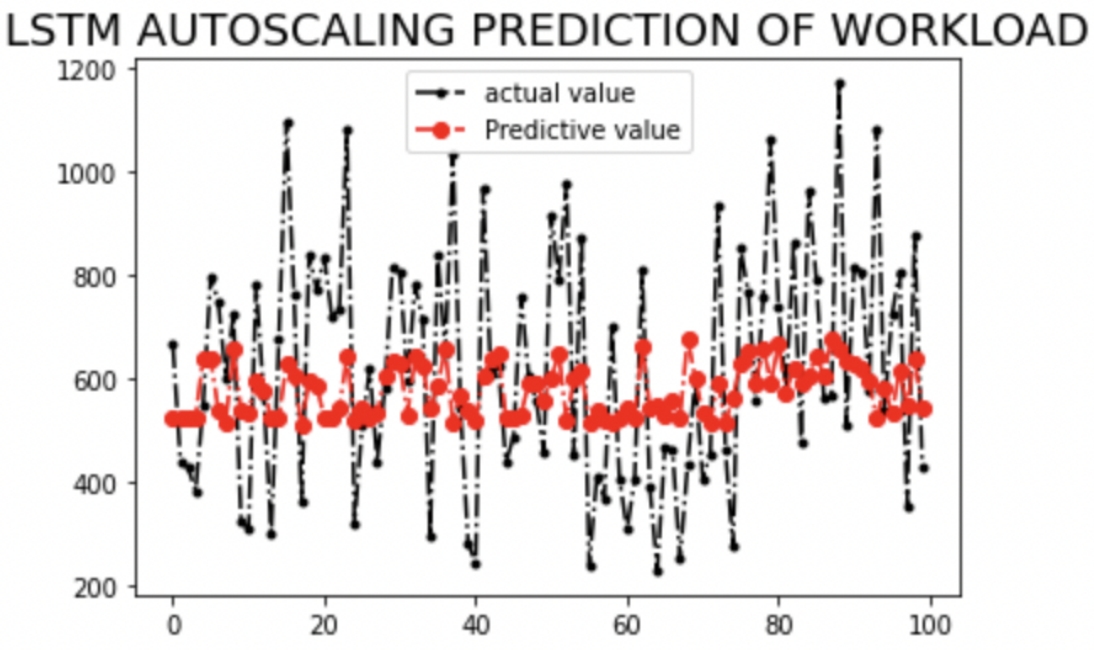


Figure 12: LSTM Actual value vs Predicted Value

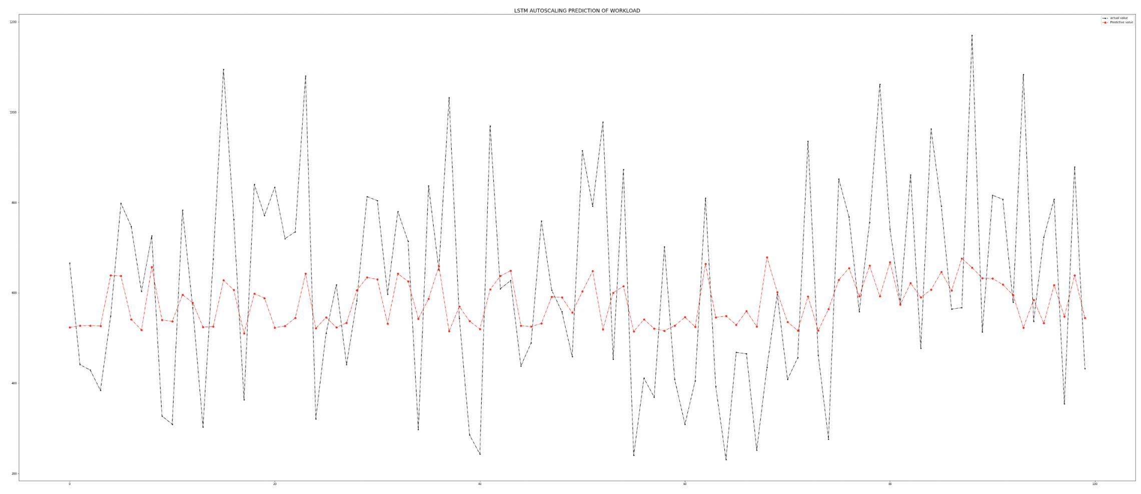


Figure 13: LSTM Actual value vs Predicted Value- more expanded for clear graph

autoscaling group. here we can clearly see that the CPU utilization at-low, at-high and again at-low states.

Desired capacity count graph10: we can see in the above graph the increase and decrease of desired capacity in the auto scaling group from 1 to 2, 2 to 3, 3 to 2 and 2 to 1.

The figure11 showes the LSTM Sequential model output with created dropouts and dense activation layer.

The figure shows the total predictions/input sample.12

It is the same graph as above but expanded little bit for better understanding. 13

7 CONCLUSION

Deep learning methods often impact the technologies in cloud computing. One of the usecase of deep learning is to use it while predicting the load for auto-scaling. In this paper, for evaluation purpose, the performance metrics used is the CPU Utilization. When the CPU load is increased, the instance will be created and when the CPU load decreases it will gradually delete the instances according to the loads. When the dataset is processed through the appropriate method of deep learning. LSTM is found to be more accurate for training the dataset and after performing the comparison between different methods based on the error factor which gives the accuracy, it is found that LSTM has minimum values for all different error entities. So, LSTM is used for triggering the auto-scaling. Later, this LSTM is configured by setting up an SSH connection.

In the future, more deep learning methods can be implemented to the AWS through SDK and have most use of the prediction methods in terms of less response time and accuracy. This research work can be carried out by applying the most advanced technologies such as machine learning, deep learning and also the Artificial intelligence with the AWS services which are upgrading day by day and permitting to perform more SDK operations to get efficient output.

References

[1] B. Liu, R. Buyya, and A. Nadjaran Toosi, "A fuzzy-based auto-scaler for web applications in cloud computing environments," in *International Conference on Service-Oriented Computing*, pp. 797–811, Springer, 2018. CORE2021 Rank: C.

[2] V. Podolskiy, A. Jindal, and M. Gerndt, "Iaas reactive autoscaling performance challenges," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pp. 954–957, IEEE, 2018. CORE2021 Rank: B.

[3] F. Lombardi, A. Muti, L. Aniello, R. Baldoni, S. Bonomi, and L. Querzoni, "Pascal: An architecture for proactive auto-scaling of distributed services," *Future Generation Computer Systems*, vol. 98, pp. 342–361, 2019. JCR Impact Factor 2020: 7.187.

[4] M. Yan, X. Liang, Z. Lu, J. Wu, and W. Zhang, "Hansel: adaptive horizontal scaling of microservices using bi-lstm," *Applied Soft Computing*, vol. 105, p. 107216, 2021. JCR Impact Factor 2021: 6.725.

[5] C. Qu, R. N. Calheiros, and R. Buyya, "Auto-scaling web applications in clouds: A taxonomy and survey," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 1–33, 2018. JCR Impact Factor 2020: 10.282.

[6] W. Iqbal, A. Erradi, and A. Mahmood, "Dynamic workload patterns prediction for proactive auto-scaling of web applications," *Journal of Network and Computer Applications*, vol. 124, pp. 94–107, 2018. JCR Impact Factor 2020: 6.281.

[7] A. Bauer, N. Herbst, S. Spinner, A. Ali-Eldin, and S. Kounev, "Chameleon: A hybrid, proactive auto-scaling mechanism on a level-playing field," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 4, pp. 800–813, 2018. JCR Impact Factor 2021: 4.531.

[8] S. Rahman, T. Ahmed, M. Huynh, M. Tornatore, and B. Mukherjee, "Auto-scaling vnfs using machine learning to improve qos and reduce cost," in *2018 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, 2018. CORE2018 Rank: B.

[9] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle, "Elasticity in cloud computing: state of the art and research challenges," *IEEE Transactions on Services Computing*, vol. 11, no. 2, pp. 430–447, 2017. JCR Impact Factor 2021: 8.216.

[10] A. Bauer, V. Lesch, L. Versluis, A. Ilyushkin, N. Herbst, and S. Kounev, "Chamulteon: Coordinated auto-scaling of micro-services," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pp. 2015–2025, IEEE, 2019. CORE2021 Rank: A.

[11] F. Rossi, M. Nardelli, and V. Cardellini, "Horizontal and vertical scaling of container-based applications using reinforcement learning," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pp. 329–338, IEEE, 2019. CORE2021 Rank: B.

[12] V. Simic, B. Stojanovic, and M. Ivanovic, "Optimizing the performance of optimization in the cloud environment–an intelligent auto-scaling approach," *Future Generation Computer Systems*, vol. 101, pp. 909–920, 2019. JCR Impact Factor 2020: 7.187.

[13] E. Radhika and G. S. Sadasivam, "A review on prediction based autoscaling techniques for heterogeneous applications in cloud environment," *Materials Today: Proceedings*, vol. 45, pp. 2793–2800, 2021. JCR Impact Factor 2021: 31.041.

[14] J. Bibal Benifa and D. Dejey, "Rlpas: Reinforcement learning-based proactive auto-scaler for resource provisioning in cloud environment," *Mobile Networks and Applications*, vol. 24, no. 4, pp. 1348–1363, 2019. JCR Impact Factor 2021: 3.426.

[15] F. Tahir, M. Abdullah, F. Bukhari, K. M. Almustafa, and W. Iqbal, "Online workload burst detection for efficient predictive autoscaling of applications," *IEEE Access*, vol. 8, pp. 73730–73745, 2020. JCR Impact Factor 2020: 3.367.

[16] N. Marie-Magdelaine and T. Ahmed, "Proactive autoscaling for cloud-native applications using machine learning," in *GLOBECOM 2020-2020 IEEE Global Communications Conference*, pp. 1–7, IEEE, 2020. CORE2021 Rank: B.

[17] D. Buchaca, J. L. Berral, C. Wang, and A. Youssef, “Proactive container auto-scaling for cloud native machine learning services,” in *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*, pp. 475–479, IEEE, 2020. CORE2021 Rank: B.

[18] D. Saxena and A. K. Singh, “A proactive autoscaling and energy-efficient vm allocation framework using online multi-resource neural network for cloud data center,” *Neurocomputing*, vol. 426, pp. 248–264, 2021. JCR Impact Factor 2020: 5.719.

[19] P. Singh, P. Gupta, K. Jyoti, and A. Nayyar, “Research on auto-scaling of web applications in cloud: survey, trends and future directions,” *Scalable Computing: Practice and Experience*, vol. 20, no. 2, pp. 399–432, 2019.

[20] T. Subramanya and R. Riggio, “Centralized and federated learning for predictive vnf autoscaling in multi-domain 5g networks and beyond,” *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 63–78, 2021. JCR Impact Factor 2021: 4.195.

[21] A. A. Khaleq and I. Ra, “Intelligent autoscaling of microservices in the cloud for real-time applications,” *IEEE Access*, vol. 9, pp. 35464–35476, 2021. JCR Impact Factor 2020: 3.367.

[22] N.-M. Dang-Quang and M. Yoo, “An efficient multivariate autoscaling framework using bi-lstm for cloud computing,” *Applied Sciences*, vol. 12, no. 7, p. 3523, 2022. JCR Impact Factor 2020: 2.679.

[23] “Dataset 1 url link.” `https://raw.githubusercontent.com/Michael2397/MultivariateForecasting/master/requestMultiFeatures.csv`.

[24] “Dataset 2 url link.” `https://raw.githubusercontent.com/Michael2397/MultivariateForecasting/master/requestMultiFeaturesShift.csv`.

[25] S. Smyl and K. Kuber, “Data preprocessing and augmentation for multiple short time series forecasting with recurrent neural networks,” in *36th international symposium on forecasting*, 2016. JCR Impact Factor 2020: 3.367.

[26] G. Liu and J. Guo, “Bidirectional lstm with attention mechanism and convolutional layer for text classification,” *Neurocomputing*, vol. 337, pp. 325–338, 2019. JCR Impact Factor 2020: 5.719.

[27] “Dynamic scaling for Amazon EC2 Auto Scaling.” `https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-scale-based-on-demand.html`.