LSTM for Time-series Prediction of FECBench

Robert Canady
Institute for Software Integrated Systems
Vanderbilt University
Nashville, TN
robert.e.canady@vanderbilt.edu

***Abstract*—For this project in Deep Learning, I am doing a time-series prediction of application latency using Long short-term memory (LSTM) from the results obtained from FECBench. It is a multivariate problem with 13 input parameters, and the goal is to predict the value of the latency at time t+1, from the data of t. This project will hopefully be extended for use in real world resource management applications in future.**

Keywords—LSTM, FECBench, resource management

I. Introduction

With the rise of Cloud Computing and IoT, there is an increasing need for efficient resource management strategies. This is especially the case in multi-tenant cloud platforms where multiple consumers are sharing the same resource. It is very important to be able to detect and predict potential sources of performance bottlenecks. One common source of this comes from Performance Interference, which is where applications that are running on the same physical machine cause performance degradation because of their influence on each other.

Gathering statistics about applications is not always a simple task. As outlined in FECBench [1], there are many reasons why gathering the necessary statistics without consuming too many cloud resources is difficult such as: "having to deal with different virtualization techniques, heterogeneity and advances in hardware and operating systems, changing application mix with differing SLOs, dynamic workloads, and the tight coupling of the installed statistics collection strategies to hardware-specific, low-level statistics collection APIs."

Being able to not only detect but predict the performance interference could be very helpful when trying to meet SLOs of individual applications. There are some situations where it may be infeasible to collect the necessary statistics about an application. This is a situation where a pretrained model could be of use to predict resource usage statistics and help with resource provisioning. If you have already trained a model based on values obtained from applications ran on the same/similar machine, then it will be able to predict the statistics on that machine that you are not able to measure statistics on.

II. Problem Description

A. FECBench

FECBench is a framework created by Yogesh Barve of the DOC group at Vanderbilt University. It allows the user to collect various microarchitectural metrics as well as the metrics obtained from the *collectd* tool. It also can collect metrics from virtual machines (VMs) and Docker containers. FECBench has a benchmarking component that consists of a number of latency-sensitive, client-server target applications. Along with this, there are workload applications that can be used to cause performance interference on these target applications.

It would be very useful to be able to predict the latency of one of the applications from the previous timestep's data for various application areas such as cloud resource management. This is the problem that I am trying to solve.

B. Training Data

The training data from this project was gathered from FECBench after running six different benchmark applications with the performance interference capabilities. The six applications are:

- pmd: analyzes a set of Java classes for a range of source code platforms
- blackscholes: calculates the prices for a portfolio of European options analytically with the Black-scholes partial differential equation
- luindex: uses lucene to indexes a set of documents; the works of Shakespeare and the King James Bible
- dedup: compresses a data stream with a combination of global and local compression called "deduplication"
- freqmine: employs an array-based version of the Frequent Pattern-growth method for Frequent Itemset Mining
- fft (Fast Fourier Transform): algorithm that computes the discrete Fourier transform of a sequence

The metrics collected and used to predict the latency were: Benchmark ID, host L2 bandwidth, host L3 bandwidth, host L3 system bandwidth, host memory bandwidth, host context

switch, host CPU percentage, host disk IO, host disk IO time, host disk weighted IO time, host memory, host network, and host latency. These metrics are collected every five seconds and the latency is based on how many times the application has ran in those five seconds.

The training data for X will be all of those values at time t. Then the training y value will be the host latency at time t+1.

C. Learning Task, Experience, and Performance Measure

The learning task for this project is to learn how various metrics of a computer application with some inserted interference will affect the latency at the next time step.

The experience for this project will be split into two parts, input and output. The input will be the FECBench data collected for the six different applications with interference. The output data will be the predicted latency.

The performance measure I will use for this project will be mean squared error (MSE), root mean squared error (RMSE), graphing the loss function, and graphing the predicted latencies vs the actual latencies.

III. My Solution

A. RNN

Recurrent Neural Networks are a special type of neural network that is specifically designed for processing sequences of values. Some of their application areas are Natural language processing, machine translation, and speech recognition.

The equation for recurrent neural networks is supplied below:

$$s^{(t)} = f(s^{(t-1)}; \theta)$$

Where $s^{(t)}$ is the state of the system and it depends on the state of the system at time (t-1). This is what causes it to be recurrent. All predictions at time t, are based on all previous predictions and the information learned from them.

RNNs do have some drawbacks that need to be discussed. RNNs are very good with learning short-term dependencies. However, for any long-term dependencies, they do not perform very well. This is due to an issue with vanishing gradients [2].

B. LSTM

LSTM is a special form of RNN. LSTMs were originally introduced to solve the vanishing gradient problem that can sometimes occur in RNNs. Information flowing through an LSTM network does not cause the previous information to be completely changed like RNNs, but rather modifies it slightly and selectively remembers or forgets information [2].

Each cell of an LSTM network contains three different gates which are called forget, input and output gates that each serve different purposes to learning. The forget gate is responsible for removing information that is no longer needed for learning. The input gate is responsible for adding information to the cell state. Lastly, the output gate is responsible for deciding what information is useful and should be sent as an output. Also, as can be seen in figure 1, there are two states being sent to the next cell, the hidden state or the output of this cell and the cell state [2].

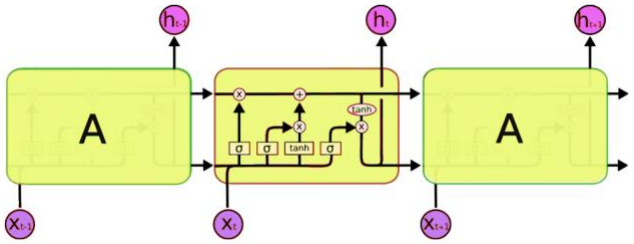

Figure 1. Architecture of an LSTM network

C. Program Description

In this section I will describe my program I wrote to complete the project:

Preprocessing

For preprocessing, I initially had to load the data in, which was done using pandas. Since I had six files, I concatenated the six CSV files together to have a total of 1800 rows of data. I then extracted all of the relevant columns from the files, there were 35 columns in the files but only 13 of them were relevant to predicting the latency.

The next step in the process was label encoding and normalizing the data. The benchmark ID needed to be label encoded to be of any use. Next, every other value was normalized to be between 0 and 1. After this, I called a function called *helper()* that returns all of the values along with those values shifted for one timestep, so it returned var(t) and var(t+1). I did not need all of the values for time (t+1), so I dropped all of the extra columns except for the one that represented latency.

The last step of preprocessing was to split the data into train_X, train_y, test_X, test_y. The ratio for the train-test split I used was 0.9, which gave 1620 training rows and 180 testing rows. I then reshaped the X vectors to have the right shape for LSTM by inserting a value for timesteps. The final shape for train_X, train_y, test_X, and test_y were (1620, 1, 13), (1620,), (180, 1, 13), and (180,) respectively.

Model

The function createLSTM() calls the model I created that is sequential and has 2 LSTM layers, one fully-connected layer, and then an output layer. The first LSTM layer has 300 units and a dropout ratio of 0.5. The second LSTM has 200 units and a dropout ratio of 0.5. The fully-connected layer is dense with an activation function of relu and 500 units. Lastly, the output layer is dense as well. It is then compiled using mean square error as the loss and Adamax or Nadam as the optimizer.

Running the Model

To run the model, I first call the function createLSTM() described in the previous step. Now that the model is created,

I fit the model using the train_X and train_y data. I run this for 80 epochs, with a batch size of 72, and validation data of test_X and test_y.

IV. Evaluation

To evaluate model, I first plot the loss function for the training data and the validation data. The next piece of information I receive is the final mean squared error value and root mean squared error value for the training. The last thing I use to evaluate the model is a graph of the predicted latencies versus the actual latencies for the test data. I have included the graphs for loss function and Predicted vs actual latency with the Nadam and Adamax optimizers for comparison. These two optimizers gave the best results but ultimately, Adamax seemed to work the best of the two.

	Nadam	Adamax
MSE_train	0.00424444	0.00513333
RMSE_train	0.04137407	0.0479
MSE_val	0.00101493	0.002
RMSE_val	0.02656667	0.03773333

Figure 2. MSE and RMSE of Nadam and Adamax Optimizer

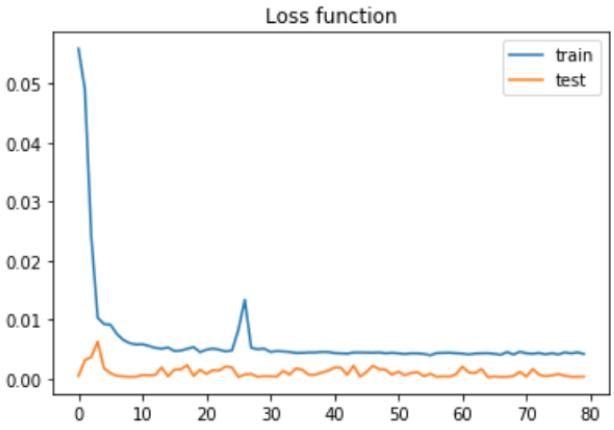

Figure 3. Loss Function w/ Nadam optimizer

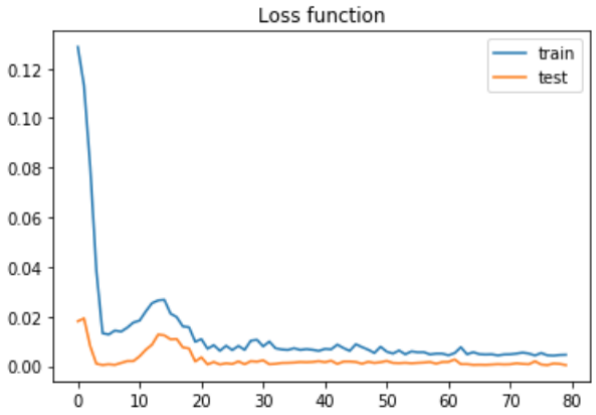

Figure 4. Loss Function w/ Adamax optimizer

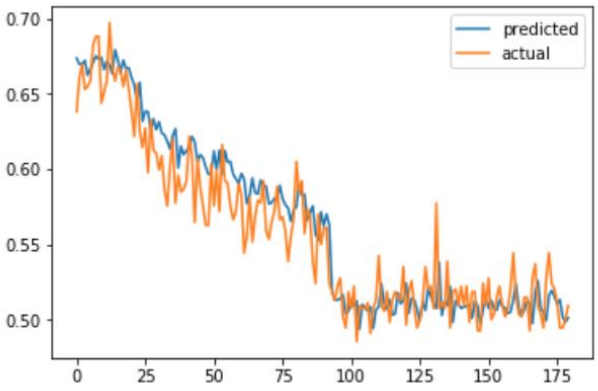

Figure 5. Predicted vs actual with Nadam optimizer

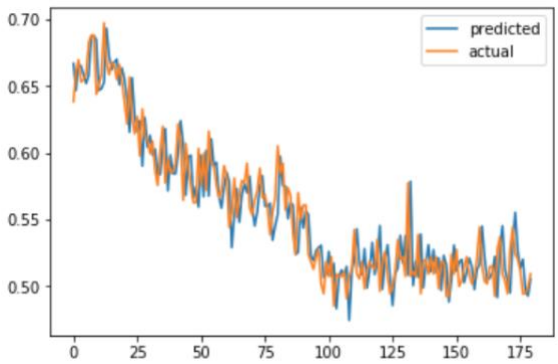

Figure 6. Predicted vs. actual with Adamax optimizer

Although the MSE value for Nadam ended up being somewhat lower, Adamax seemed to predict the values much better than Nadam.

V. Related Works

From what I have read, I have not been able to find any works that used LSTM networks to predict resource usage in a similar way that I have. However, there are works using other types of models to predict resource usage in cloud datacenters.

In one work [3], they used rescaled range analysis method to perform fractional differencing and capture long range dependencies in cloud workloads. In this method, the Hurst parameter is calculated and also calculates long range dependencies in time series. They also used an online approach for time series prediction models. They explore online adaptive models such as Gradient Descent and the Levenberg Marquardt Algorithm (LMA). For their dataset, they use the Google Cluster trace of about 12500 machines and provides runtime information of different jobs over a 29-day period. This cluster has a variety of different heterogeneous machines.

They evaluated their predictions with RMSE and got much lower values with their online models and in particular got the best results with LMA. I assume they get much better values for RMSE because they have a much larger dataset.

Another work [4], presents an approach called Automatic Proactive Resource Allocation (APRA), that proactively allocates resources by using Support Vector Machines (SVMs) for time series prediction to predict resource usage. SVMs are a form of machine learning that categorizes data by using a plane to separate the data. Since the resource prediction for one resource depends on the past resource usages of the same application on different VMs, cross correlation needs to be considered. This is done by applying the SVM algorithm to model multiple time series simultaneously.

They evaluated their work with RMSE, MAE and MAPE. Their RMSE values were an order of magnitude higher than the values obtained in my project. I assume this is because it is a much earlier work from 2014.

VI. Future Work

This work I think could be extended to be even better with further exploration of architectures. One issue was the lack of data so I would like to gather larger datasets from Yogesh. In the future I would also like to apply this technique in practice to see how it could help with resource management. I would potentially like to submit a paper about this technique.

VII. Conclusion

In this work, I used LSTM for time series prediction of the data obtained from FECBench. I evaluated my work with MSE, RMSE, loss, and graphing the predicted vs actual values. I tested multiple architectures and different regularizations and optimizations. I found the architecture used in my approach to work best with dropout at both LSTM layers and the fully connected layer and using the Adamax optimizer at the output layer. I have also compared my approach to two other time series prediction approaches and found that it outperforms one approach and underperforms the other approach. With more data however, I think the approach taken in this paper could be very helpful in real world resource allocation scenarios.

Acknowledgment

I would like to thank Prof Koutsoukos for all of his help throughout the semester and for all of his teachings about Deep Learning. I really feel like I learned a lot from this class that will be relevant to my future research.

References

[1] Y. D. Barve, S. Shekhar, A. D. Chokra, S. Khare, A. Bhattacharjee and A. Gokhale, "Poster: FECBench: An Extensible Framework for Pinpointing Sources of Performance Interference in the Cloud-Edge Resource Spectrum," Symposium on Edge Computing, 2018.

[2] P. Srivastava, "Essentials of Deep Learning: Introduction of Long Short Term Memory," *Analytics Vidhya*, 2017

[3] S. Gupta and D. A. Dinesh, "Online Adaptation Models for Resource Usage Prediction in Cloud Network," Twenty-third National Conference on Communications, 2017.

[4] D. Minarolli and B. Freisleben, "Cross-Correlation Prediction of Resrouce Demand for Virtual Machine Resource Allocation in Clouds," Sixth International Conference on Computational Intelligence, Communication Systems and Networks, 2014.