System failure prediction using log analysis

A Deep Learning approach to predict failure in a system using Recurrent Neural Network(LSTMs)

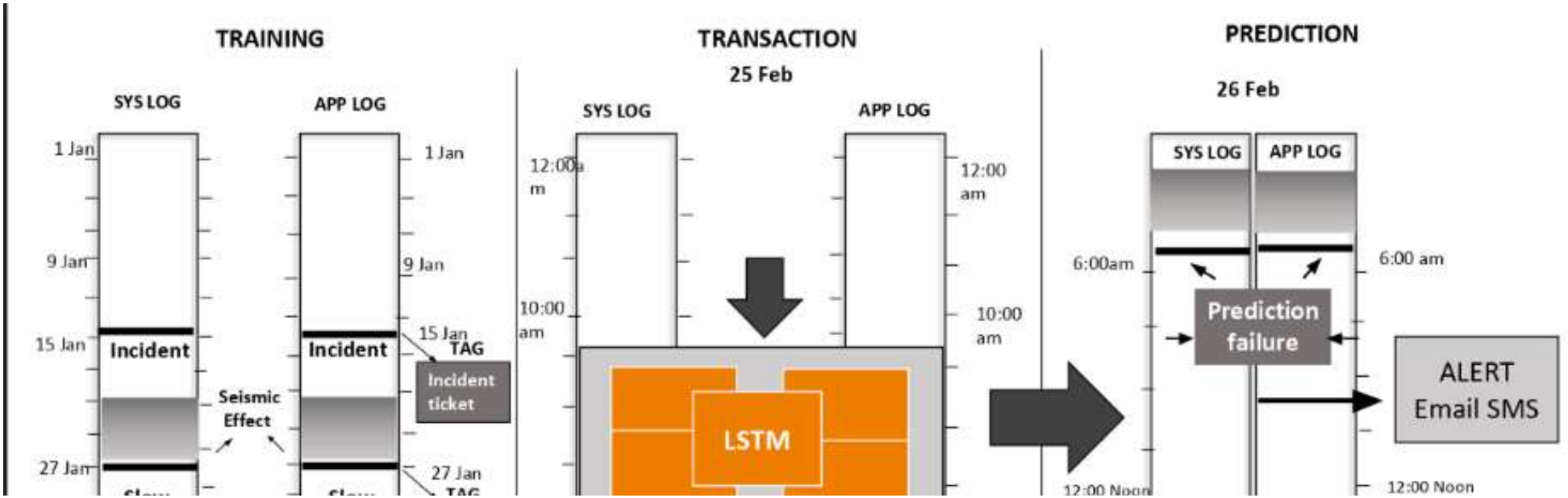
Animesh Dutta Follow
Dec 19, 2019 · 6 min read ★

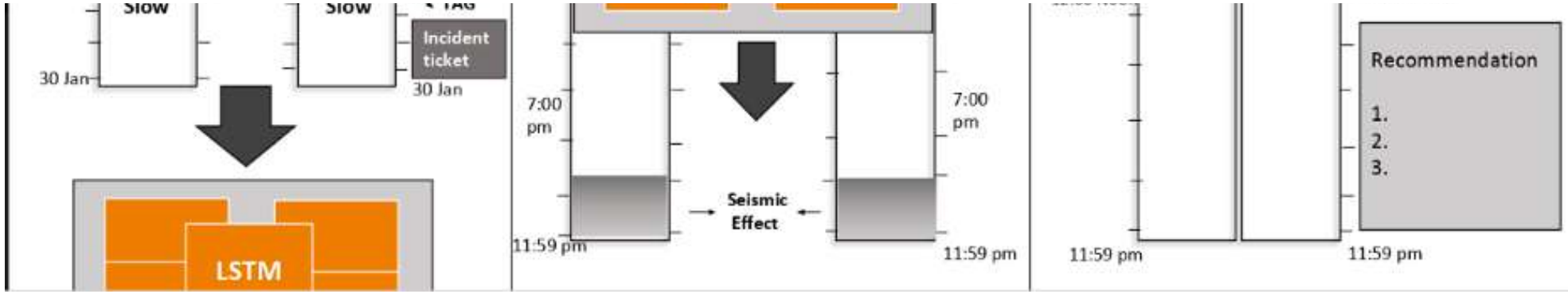
In modern days, system failure is a grave issue and needs to be dealt with. IT companies or various research organizations can be highly benefited if an accurate system failure prediction can be obtained. The adverse effect of computer failure is somewhat mitigated if a proper prediction is made beforehand. The usage of resources, applications and

other memory consuming processes can be limited if such a case is about to occur thus preventing the system breakdown.

Achieving accurate predictions with adequate time left is quite difficult. In this blog, I present a simple approach to detect the failure by parsing the log files quite in advance. We generate an early warning before a failure condition arises. To serve our purpose, we have used a Recurrent Neural Network, namely, Long Short-Term Memory. The approach in this paper uses a sliding window to fetch the desired results. The important factors taken into account are RAM, CPU and Hard Disk utilization.

The given figure shows the approach used:





Methodology

System Failure Prediction is essential in many applications like where a computer needs to perform high computations. Very high usage of hard disk or crash of RAM can prevent the applications being executed on HPC. High-Performance Computing is the use of parallel programming to run complex programs efficiently. The recovery of HPC can take long or even might not be possible at times. The use of time series forecasting has been common too but it didn‘t include the parameters that we are going to state next which can be beneficial.

Log files obtained from systems comprises of information on the status and memory consumption of a system. We know that three main utilizations of a computer are CPU, RAM, and hard disk utilization. These log files can provide us with timestamps and the exact utilization of resources on definite timestamps. We have taken into account the values at timestamps with the same interval. We had a data comprising of the log files of a system

generated in the past five days in .nmon format. The timestamp variation considered is fixed and has a fixed gap of 10 minutes.

We had .nmon files that were converted to .csv for proper study. We can also visualize the data using nmon visualizer tool. The parameters we are considering for CPU, RAM and Hard disk utilization are listed below:

CPU utilization:

User%: This states that the processor is spending x% of its time running user space processes. A userspace process is one that doesn‘t use kernel. Some common user-space process includes shells, compilers, databases and all programs related to desktop. If the processor isn‘t idle then usually the majority of CPU time is used to run user space processes.

Sys%: This states that the processor is spending x% of its time running system processes. System processes comprise of kernel space programs

Wait%: This states that the processor is spending x% of its time running kernel processes. All the processes and system resources are handled by the Linux kernel. The kernel performs tasks like running system processes managing hardware devices like a hard disk in kernel space.

Idle%: This shows what percentage of CPU is waiting for other processes to complete. At times, the processor might initiate a read/write operation and needs to wait for other processes to finish.

CPU utilization = User% + Sys% + Wait%

RAM utilization:

- MemTotal: Total usable memory
- MemFree: The amount of physical memory not used by the system
- Buffers: Memory in buffer cache, so relatively temporary storage for raw disk blocks
- Cached: Memory in the page cache (Diskcache and Shared Memory)
- MemUsed = MemTotal-MemFree-Buffers-Cached

% of used RAM at a particular timestamp = (MemUsed/MemTotal) * 100

Hard Disk utilization:

Hard disk utilization %= (Hard disk space used/ total hard disk space) *100

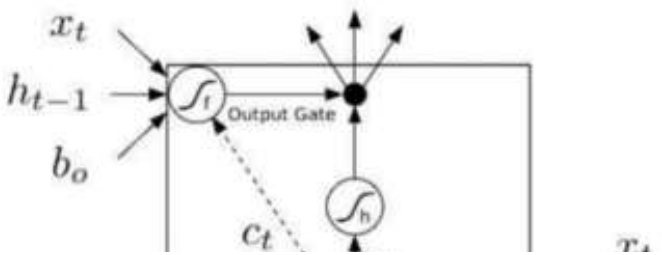
PCA:

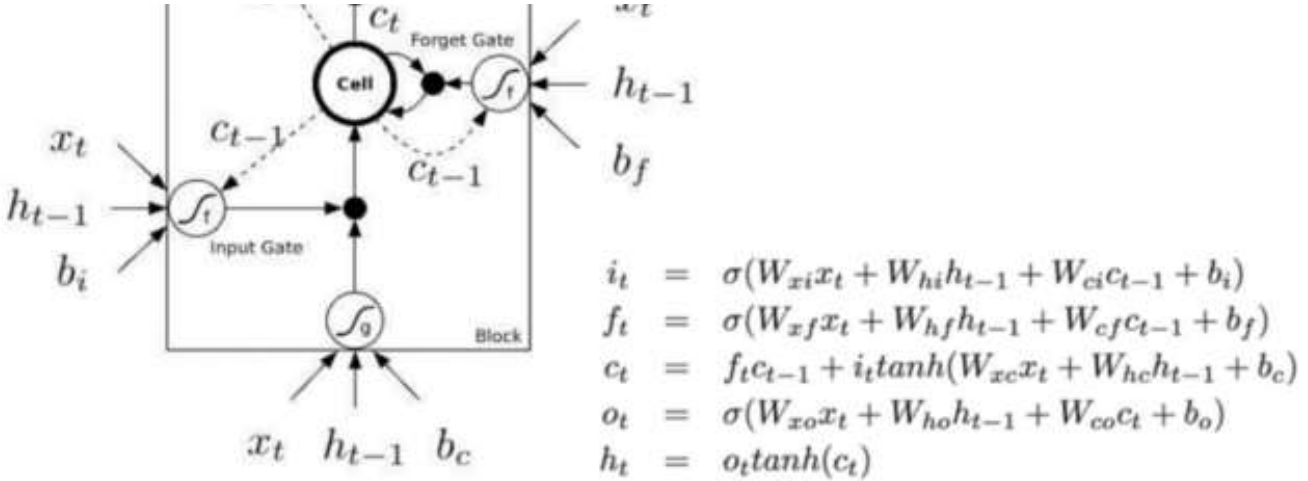
Now we have our values for CPU, RAM and hard disk utilization for timestamps. We apply PCA to get a single reduced value out of these 3 parameters. As we know PCA or Principal Component Analysis is a way to deal with highly correlated variables. We can get a single value for all the utilizations stated above. We can then apply univariate time series forecasting to predict a single value for the future timestamps.

Steps for applying PCA:

a) Standardized the data. b) Calculate the eigenvalues and eigenvectors from the covariance matrix. c) Sort the eigenvalues in decreasing order to rank corresponding eigenvectors. d) Select 1 eigenvector corresponding to the largest eigenvalue. This gave us the reduced parameter for each timestamp.

LSTM model:





LSTM architecture

The steps we followed:

i) A moving forward window of size 50, which means we used the first 50 data points as out input X to predict Y i.e. 51st data point. Using the window from 1st to 51st point next, we forecast for the 52nd point.

ii) A two-layered LSTM model combined with a dense output layer was used to make the prediction. iii) We used the following two approaches to predict the output —

a) We forecasted the value for each item in the test dataset.

b) We feed the forecast which was made previously back to the input window by moving it a step forward and make a forecast at the timestamp needed. We have a 3D input vector for the LSTM comprising of several samples, several timestamps, number of features. After saving the weights for the model trained, we plot the predicted values to visualize the trend in data. We have the model with a 2 layered LSTM and a dense output layer at the end as shown in table 3 below. We use dropout in between as it is a regularization technique that is used to prevent over-fitting while we train our LSTM model.

The model summary:

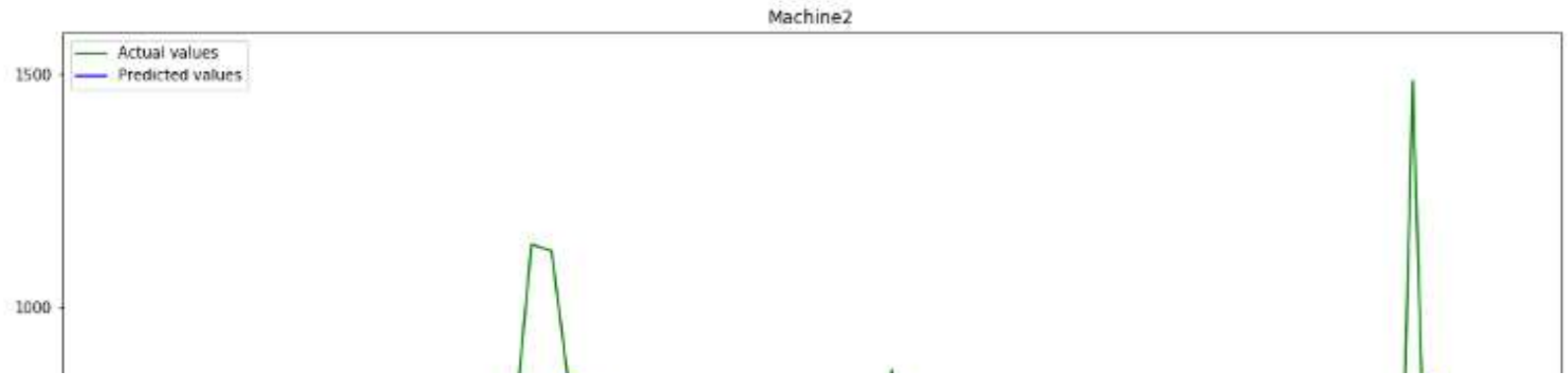
Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 50, 50)	10400
dropout_1 (Dropout)	(None, 50, 50)	0
lstm_2 (LSTM)	(None, 256)	314368
dropout_2 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 1)	257
activation_1 (Activation)	(None, 1)	0

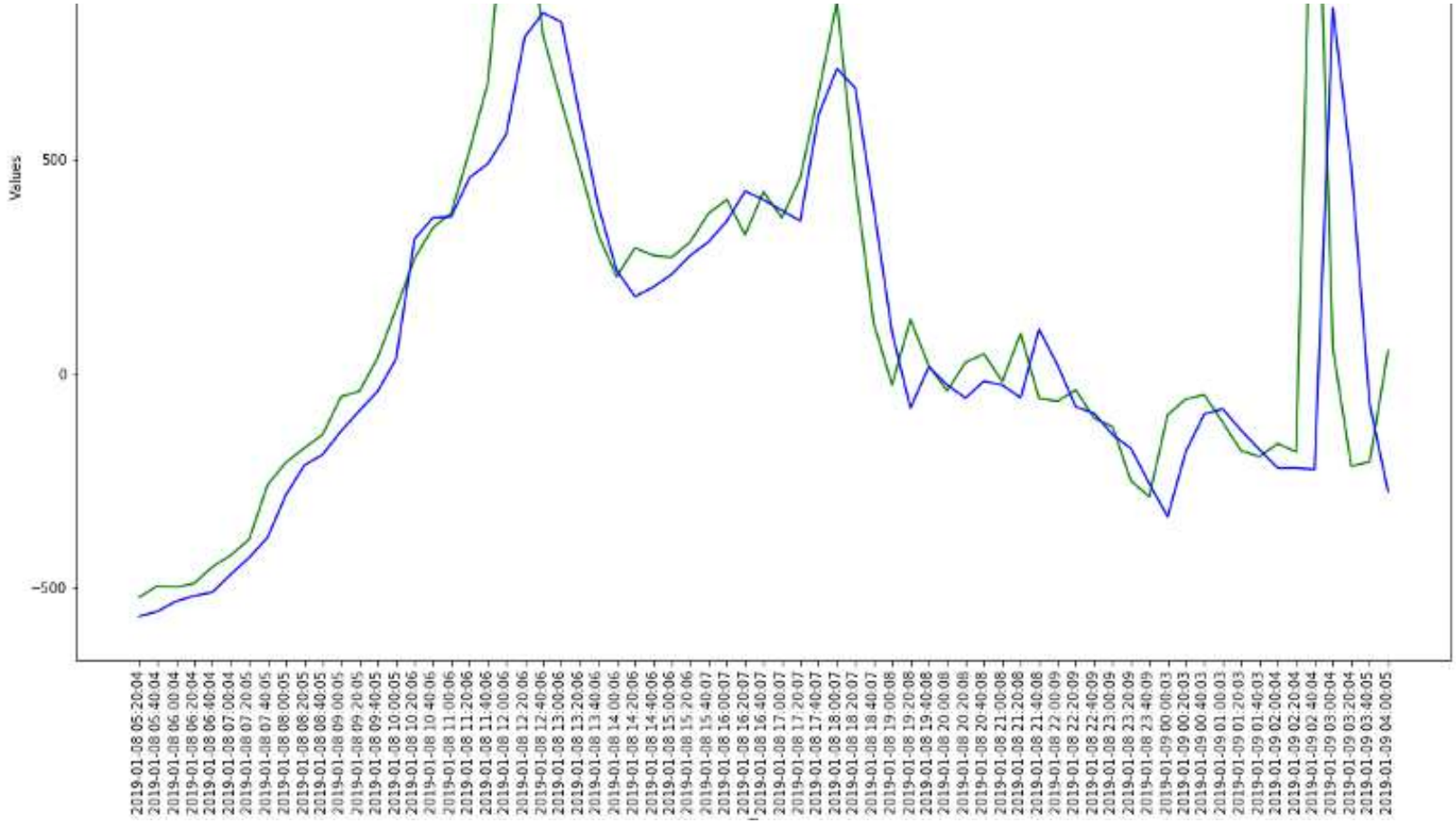
Total params: 325,025
Trainable params: 325,025
Non-trainable params: 0

Model summary for LSTM

Results:

The figure below displays the graphical view of predicted values and actual values to find the trend. Despite the less amount of training data, we got the desired variations in output. Our model was able to capture the trend. With more training data, more accurate results can be predicted. The green line represents the actual values while the blue line represents the predicted values. On X-axis, we have the dates with a timestamp gap of 10 minutes. The Y-axis is the PCA reduced value which we earlier obtained. The saved weights help us to predict the reduced values for future timestamps. Having the timestamps and PCA reduced value for failure cases (can be noted for some cases where actual system failure took place), we can correctly classify whether the predicted value falls in failure class or not. We used the Logistic regression classification algorithm to serve our purpose





This blog will help users to prevent system failure as it can send an alert mail or SMS to the person before the actual failure takes place. The user can then limit the number of running processes on the system by terminating redundant processes. Using LSTMs for time series forecasting helped us to get the reduced values for future timestamps. Then that value can be used to classify for Normal or Failure class using the Logistic Regression Classification model.

. . .

Conclusion:

> *This project was done as a part of an internship at Celebal Technologies, Jaipur.*

This approach will help users to prevent system failure as it can send an alert mail or SMS to the person before the actual failure takes place. The user can then limit the number of running processes on the system by terminating redundant processes. Using LSTMs for time series forecasting helped us to get the reduced values for future timestamps. Then that value was classified as Normal or Failure class using the Logistic Regression Classification model. This work can be extended if we can add some more features to the existing list. We have just considered CPU, RAM and Hard Disk utilization for our research. Parameters such as Input/output data transfers i.e. rate at which data is written on the hard disk, time for which a particular process is running can also be taken into consideration to obtain better accuracy. This blog post made as part of the Intel® Student Ambassador competition. I'm grateful to be bestowed with the opportunity to be a Student Ambassador for Artificial Intelligence at Intel.

Machine Learning Deep Learning Recurrent Neural Network System Failure Prediction

Intel Sponsored Student