Imperial College London

MEng Individual Project Report

Imperial College London

Department of Computing

Investigating the Application of Machine Learning Models to Predict Resource Usage Trends within Distributed Systems

Author:
Raphael Leong

Supervisor:
Dr. Kin K. Leung

Second Marker:
Prof. William Knottenbelt

June 27, 2019

Abstract

With the rise in popularity of cloud-based distributed systems, extensive research has been done to further our understanding of these large-scale systems. The Google data center clusters are a prominent example of these systems and much of the research has been focused around the publicly available Google cluster trace dataset which outlines details of the thousands of machines' workloads and job/task scheduling managed within a Google cluster.

To gain better insight into the cluster trace, several researchers have built forecasting models to predict the resource usage of these machines within the cluster. Some of these approaches have used statistical stochastic models which requires extensive understanding of machine lifecycles and job scheduling. Machine learning approaches are relatively new within this space and we explore how effective this approach can be.

We first define possible features we can use for our forecasting model by analysing the relationships between the machine timeseries data. Ultimately, we show the strong temporal correlation which exists within each individual machine timeseries and use this autoregressive feature to train our model. We propose a neural network architecture for our forecasting model, which includes a 1D convolutional layer to learn a representation of the complex timeseries sequence, and a LSTM unit which have been proven effective to learn autoregressive data. We then evaluate our model using suitable metrics and attempt to draw comparisons to other models. With this project, we hope to push the use of machine learning methods within the field of distributed computing.

Acknowledgements

I would like to thank my supervisor Dr. Kin K. Leung for providing me with this opportunity and offering invaluable advice and suggestions throughout the project. They have been extremely enthusiastic and supportive throughout the course of this project. I would like to also express my gratitude towards Dr. William Knottenbelt for providing additional insight and advice at the beginning and how to effectively proceed with my project.

I would also like to thank my peers for being encouraging and motivating throughout the years and being able to provide helpful feedback towards the project.

Contents

Chapter 1

Introduction

With emergence of scalable systems and technologies, most of our devices and applications are now heavily governed and interconnected by cloud-based distributed systems. There exists a large amount of information and data that is being constantly shared by many machines within these networks. With this in mind, the DAIS-ITA research program in collaboration with my supervisor, Dr. Kin K. Leung, aims to be able to utilise these large sets of data and connections to further help build and develop a modern infrastructure can operate seamlessly across many distributed networks to effectively analyse real-time situations. [1]

In an effort to improve our understanding of distributed systems, we look into the resource usage trends that exist within these systems. One of the main aspects of these networked systems is being able to manage and optimise resource and workload distribution within a cluster of machines. Therefore, establishing a forecasting model which can predict the amount of resources a machine is going to utilise in the future can extremely beneficial.

Machine learning is a relatively new concept within the field of distributed computing. There have been several prior research papers, including a recent one published by Dr. Kin K. Leung [2], which have proposed different models in order to predict future resource occupancy but not many have explored the full capabilities of using artificial neural networks as a means to forecast the timeseries data. Therefore, the focus of the project will be to investigate the potential of applying machine learning models towards these datasets collected from distributed systems, in order to help better optimise resource and workload distribution in the future.

The main aim for this project is to investigate the possibility of building a machine learning model that can predict the resource usage for each machine within large cluster networks. As the relationship between a set of machines and their workload characteristics can be quite dynamic and complex, it will be difficult to fit the processed timeseries data towards a suitable machine learning model which can be used for accurate predictions. Therefore, we will have to initially perform extensive analysis to help characterize these machines within the large dataset.

Initially, we will analyse from a cluster data trace provided by Google [3], which provides information related to cluster usage within Google data centers. Due to the dataset being extremely large, containing data and information pertaining to 12,583 nodes over a time period of around 29 days. Not only would this be difficult to parse and analyse in detail, it will also be challenging to naturally find suitable features among all the units in the cluster network for our machine learning model.

To obtain a deeper insight into the cluster trace, we will take several characteristics from prior works as a baseline to start our findings. We will first generate machine timeseries sequences which describe the resource usage over a randomly sampled period of time within the trace. We then analyse the spatial correlation between combinations of machine timeseries datasets and identify possible features we can use for our forecasting model. We will also analyse the temporal correlation of individual machine timeseries datasets in order to identify whether linear regression problem can be autoregressive.

Once we have a satisfactory set of features we can train our model with, we will begin implementing a machine learning model that can effectively predict resource utilisation within an individual machine. To further optimise our model, we will tune the hyperparameters through trial and error, with the goal of minimising the loss function. We will then evaluate the results of our newly created model and determine whether it is able to accurately predict a machine's resource occupancy and availability in the future based on past we obtain from the cluster trace.

The main goals of the project is to contribute the following:

- Extensively analyse and provide extra insight into the Google cluster trace dataset
- Investigate whether there exists a strong spatial correlation between machines based on certain job/task features or machine characteristics
- Investigate whether there exists a strong temporal correlation within an individual machine's time series data
- Investigate the application of an appropriate machine learning model which can predict future CPU and memory usage within each machine timeseries

Chapter 2

Background

2.1 Distributed Network Systems

Distributed computing is a term which describes a set of network systems where the components are located on many different interconnected networked computers with the focus of sharing their resources and communicating their actions between each other in order to serve a common purpose for the distributed network system.

With the rise in popularity of technologies and applications that require more processing power, companies are starting to favour using distributed systems to operate their businesses as it allows them to easily scale their services and have more fault-tolerant systems due to the increased amount of nodes in the network system.

2.1.1 Google Data Center Clusters

This project will mainly focus on analyzing the trace dataset provided from Google Clusters which are publicly available. Google hosts several remote data centers which have many Google Clusters which hold racks of thousands of machines, interconnected by a high bandwidth network. These machines are managed by a common cluster-management system which help with scheduling and delegating jobs to the machines. Jobs consist of Linux programs which can be built from several processes and these are given to machines which have resources available to run. The main purposes of these clusters are to host Google's online application services and store large amounts of collected user data. [3][4]

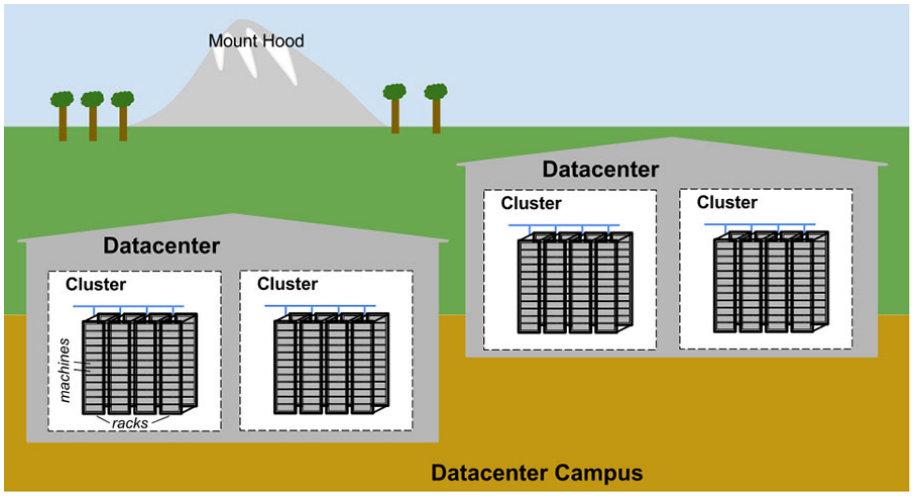


Figure 2.1: Diagram of Google data center taken from online documentation

2.2 Related Works

There are many articles and research papers within the field of distributed computing. We will be going through papers that solely focus on analysing features and characteristics of the Google cluster trace dataset in an effort to develop a better understanding of cloud-based distributed systems.

One of the key insights, we will try to take away from these papers are the potential features and characteristics [5] we can group machines by to further analyse potential spatially correlated relationships that we can use for our machine learning model. Charles Reiss[6] provides detailed analysis over many common characteristics that exist within the trace that we can use to classify our machines:

- **Resource Usage Stability** - when tasks run for a long period of time, their resulting resource usage is usually stable.
- **Short Job Duration** - significantly shorter jobs account for an extremely low percentage to the overall utilization.
- **Repeated Job Schedules** - some jobs are run repeatedly at a certain point in time with similar resource uptake.
- **Machine Attributes** - machines consist of a few different platforms, which classifies the build and specification of the machine

In a recent paper written by Dr. Kin K. Leung and Tiffany Tuor [2], they propose a complex dynamic algorithm to group machines in a cluster based on the transmission frequency along with their surrounding machines. Therefore, in our project, we plan to use these basic characteristics as a baseline to explore additional features and correlations that may potentially exist within the cluster trace dataset.

There have also been many approaches to building a forecasting model that can effectively predict future resource usage within machine timeseries. Several statistical models have been implemented such as one by Stefano Sebastio [7], who implemented a non-Markovian stochastic reward network model which uses the characterisation of a machine's workload lifecycle to create states which can predict what kind of jobs will run in the future. Dr. Kin K. Leung [2] also proposed using LSTM units for their forecasting model due to the autoregressive nature of the trace data. A common trend in these papers is the use of a non-Markovian property within their models, where the current state value is dependent on what has been observed in previous states. Therefore, when deciding which machine learning model to use for our forecasting model, we will naturally select one that remembers previous timesteps within the sequence data such as a recurrent neural network architecture.

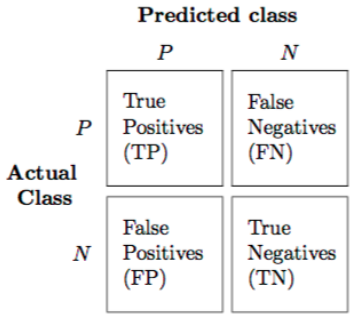


Figure 2.2: Confusion matrix example structure

2.3 Machine Learning Models

Machine Learning is a field of artificial intelligence which covers the application of algorithms that can provide systems with the ability to learn and train from set of data to later accurately build a suitable model to classify unseen data values. A supervised machine learning system can assist with characterizing complex and dynamic relationships between data points by simply inferring a function from a given set of labelled training data.

2.3.1 Linear Regression

Linear regression is an approach to a mathematical problem where the goal is to model a relationship between observed training data to output variables. To achieve this, the algorithm aims to infer a function that can map input variables to appropriate corresponding function outputs. A linear regression approach is typically used for non-classification models which will be useful for our forecasting model.

2.3.2 Performance Measures

There are several metrics when evaluating a trained model's performance: [8][9]

- Confusion Matrix - typically used in classification problems, easily observe when system makes an error in classification of a data value. We can easily derive statistics such as accuracy, precision and F1-score.
- Root mean squared error (RMSE) - typically used to evaluate linear regression problems, this value highlights the difference between the observed and predicted values in which we can assess our model from.
$$\sqrt{\frac{1}{n}\Sigma_{i=1}^{n}(y_i - \hat{y}_i)^2}$$
- Mean absolute error (MAE) - also used to evaluate non-classification problems similarly to RMSE, but is used for learning datasets with more noise data due it being more robust to outliers.
$$\frac{1}{n}\Sigma_{i=1}^{n}|y_i - \hat{y}_i|$$

2.3.3 Machine Learning Challenges

Solving the regression involves breaking down several issues: [10]

- **Model selection** - comparing various models to find the simplest model that fits the dataset
- **Hyperparameter Optimisation** - selecting good features for modelling the data and finding suitable parameter values to optimise the model
- **Overfitting** - problem where the trained model overly fits the data but does not generalize for the testing data

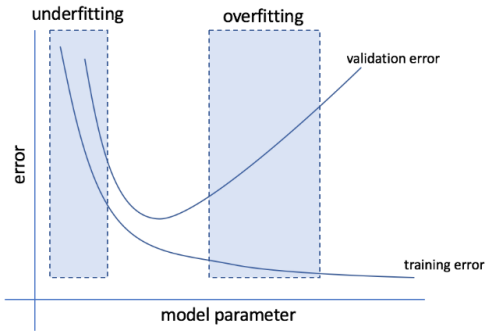


Figure 2.3: Overfitting example

Overfitting

Generally, in machine learning, the overfitting problem occurs when after training from a set of data, there is poor performance observed when comparing it with the testing data due to the generated model being overly complex for the dataset. This lack of generalization usually implies that the model being built has either too much bias or variance and as the complexity of the model increases, the more likely problem occurs. To solve this issue, we add regularization as we need our model to be more flexible to cover more situations, making it better for generalization.

Hyperparameter Optimisation

When implementing a machine learning model, many of them have hyperparameters which need to be set before training our model such as learning rate of the model or the number of hidden layers within the architecture. The process of optimising these parameter values requires constant training and testing of the model with different parameter values until we arrive at an optimal value which minimises the error function.

Cross Validation

Cross validation is a technique where we split the sampled data into equal segments and leave out one to use for validation whilst using the rest for training the prediction model. Once the model has been trained on the rest of the data, we test it with the validation chunk. Once this process has been done, it is repeated with another part of the data to be used as the testing set, until all combinations of the training data have been used. The final performance metric is averaged from all the rounds of cross validation. The method iterates through all combinations of the training data to reduce the amount of bias and variance that is caused instead of only choosing one part of the data at random to be used for validation. This process can be useful for evaluating our model and tuning its hyperparameters.

2.3.4 Deep Learning - Artificial Neural Networks

Deep learning is a branch of machine learning that is focused on algorithms surrounding the training of artificial neural networks. Neural networks have many layers which contain many perceptrons which are trained to classify a set of inputs with an activation function. To be able to classify complex sets of data values, samples are fed into the network which will go through many hidden layers of perceptrons in attempt to learn a set of output features. [11]

The advantages to using deep learning methods over traditional machine learning methods is that its performance heavily improves with extremely large amounts of data whereas other machine learning techniques have significantly worse performance as many of them have trouble computing large sets of data. This can prove useful for us when trying to apply these algorithms to time-series data which can usually be quite difficult to train due to the amount of computation required.

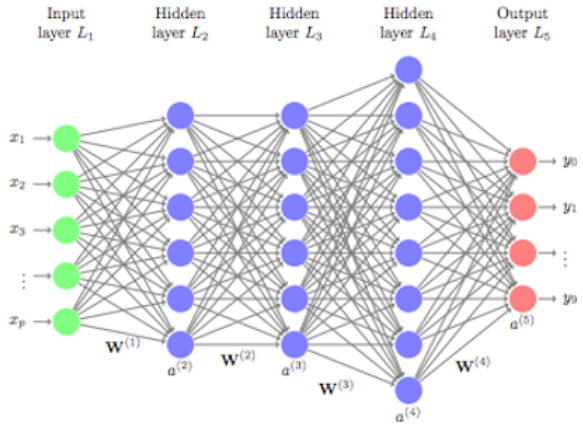


Figure 2.4: Example architecture of a multi-layer feed-forward neural network

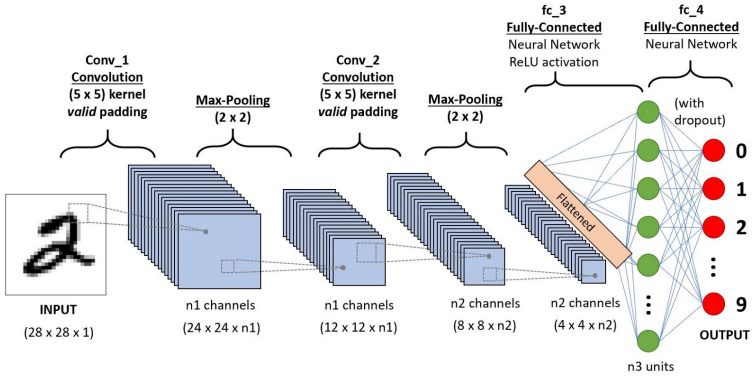


Figure 2.5: Example architecture of a multi-layer convolutional neural network

Recurrent Neural Networks and Long Short-Term Memory Networks

RNNs are a branch of neural networks which act similarly to a basic feed-forward neural network with the extra feature of being able to loop back between layers and remember past states. A common type of RNN is a Long short-term memory network (LSTM) which uses powerful algorithms to enable the memory of long-term dependencies. This allows us to process long sequences of data, where past observations are important for the model to train on and learn features from the input data. LSTMs are particularly useful for applications where the sequencing of the data is important such as text data or timeseries forecasting.

Convolutional Neural Networks

CNNs are another branch of neural networks which uses convolutional filters to usually process imaging data for the network. The input data required for CNNs are usually 2-dimensional and the architecture consists of a convolutional layer, a pooling layer and a fully-connected layer. When data is fed into the convolutional layer, a filter processes the 2D data, compressing it down to a new representation of the data which is then learnt by the output fully-connected layer. CNNs are extremely powerful in classifying imaging data and is heavily used within medical imaging. Alternatively, 1D CNNs can also be used for non-image data to tackle other challenging problems such as predicting complex sequence data.

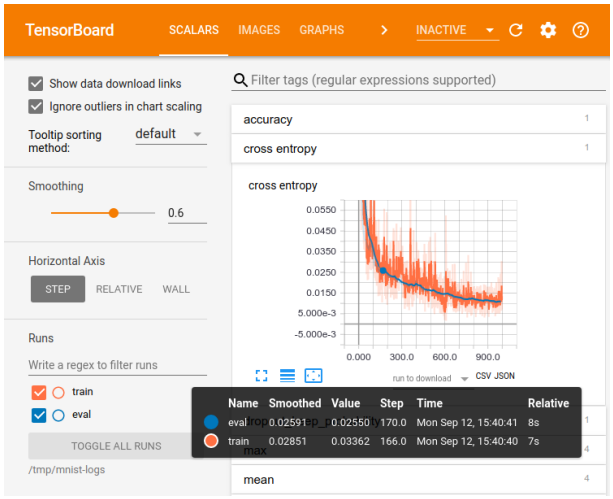


Figure 2.6: Example of Tensorboard UI

2.4 Machine Learning Frameworks

There are many existing machine learning libraries and frameworks that are available for building machine learning applications.

2.4.1 Pytorch

PyTorch is an open-source Python machine learning library used for many different applications. It is built from another machine learning framework, Torch, and is widely used within the machine learning community due to the simplicity of the Python language. It is a Lua based deep learning framework and is used heavily within the industry by companies such as Facebook, Twitter and Google. It has the ability to implement CUDA along with C/C++ libraries for processing and was made to scale up the production of building models and overall flexibility.

2.4.2 TensorFlow

Tensorflow is one of the most widely used deep learning frameworks being used by companies such as Twitter, IBM and Google due to its highly flexible system architecture. It has a few libraries that support languages such as Python, C++ and R. One of the main advantages it has over PyTorch is that it has effective data visualisation with the Tensorboard tool which can help evaluate and compare different training models.

2.4.3 Keras

Keras is a lightweight and minimalist deep learning framework designed for quick experimentation of deep learning models. It was designed to work with Tensorflow or Theano, providing a simplistic interface to allow for fast prototyping of neural networks which can then later be implemented in Tensorflow.

2.4.4 Microsoft Cognitive Toolkit (CNTK)

The Microsoft Cognitive Toolkit is an open source deep learning framework, popular for its great compatibility and ease of access. It supports language interfaces such as Python and C++. It is known to provide higher performances and scalability when operating parallelly on multiple machines compared to other frameworks such as Theano and Tensorflow. The toolkit supports both recurrent and convolutional neural networks which are widely used in applications such as image handling and speech recognition.

Chapter 3

Google Cluster Trace Dataset

For the purpose of this project, we will be primarily using the Google cluster trace [3] for analysis of distributed systems and later use the dataset to train a machine learning model which can predict future machine workload. The trace contains information and statistics related to the job processes and workload of each machine detailed by the cluster scheduler. These jobs have a wide variety of uses and applications but much of personal information is obfuscated for privacy reasons. [4] In this section, we try to break down and analyse the complexity behind the machine workloads and the cluster scheduling system.

3.1 Cluster Trace Overview

The trace is built from multiple different machine datasets which illustrate various parts of the cluster usage over a period of around 29 days. Specifically, we are interested analyzing the CPU usage and memory utilisation of each machine over a sample time period. Using this data, we investigate whether there exists a strong enough correlation within the relationship between the machine workload data to predict future trends in resource occupancy of individual machines in the cluster.

The timeseries dataset is stored in a CSV format, where each record within each table has a unique timestamp associated, indicating approximately how many microseconds have passed since the beginning of the trace. There exists a schema and documentation which outlines the details of the dataset structure.[4]

The following data tables along with their associated headers can be found within the directories of the trace dataset:

- `job_events` - *time, missing info, job ID, event type, user, scheduling class, job name, logical job name*
- `task_events` - *time, missing, job ID, task index, machine ID, event type, user, scheduling class, priority, CPU request, memory request, disk space request, different machines restriction*
- `task_constraints` - *time, job ID, task index, comparison operator, attribute name, attribute value*
- `task_usage` - *start time, end time, job ID, task index, machine ID, CPU rate, canonical memory usage, assigned memory usage, unmapped page cache, total page cache, maximum memory usage, disk I/O time, local disk space usage, maximum CPU rate, maximum disk IO time, cycles per instruction, sample portion, aggregation type, sampled CPU usage*
- `machine_events` - *time, machine ID, event type, platform ID, CPUs, Memory*
- `machine_attributes` - *time, machine ID, attribute name, attribute value, attribute deleted*

The `job_events` table describes the details and composition of each job at the specified timestamp. Each job is labelled with a unique job identifier and has other minor details. Each job is made up

of thousands of tasks and the each record highlights various specifications and parameters the task is running with at the time. The `task_events` and `task_constraints` tables give us information about the specification of each task such as the scheduling priority, resource request and possible constraints the task may have. More importantly, the `task_usage` table within the trace indicates what machine ID the task is being executed on and indicates the time period of the task being run. It also indicates how much CPU and memory is being used during the process.

The trace also provides substantial information surrounding the machines within the cluster. The `machine_events` and `machine_attributes` illustrates details about which events are being run on individual machines at certain timestamps and which hashed attribute is being stored on each machine. The information we are interested in regarding these tables is the static values of platform ID, CPUs and Memory. There are 4 different platform IDs which relate to specific builds of the machines that are utilised within the cluster. Different platform IDs can have varying resource capacity usages which we will prove to be useful for workload analysis.

3.2 Analysis methods and tools

3.2.1 Preliminary Analysis

From an initial look at the trace, we are particularly interested in the `task_usage` table when trying to analyse machine CPU rates and memory usage within the cluster. As there are over ten thousand machine IDs within the trace, it is a challenge to find features and relations amongst all the machine workloads. Using the all the machine workloads from the entire trace with uncorrelated relationships will also result in ineffective training for our machine learning model later. Therefore, the goal is to find possible characteristics within the trace that can produce a cross correlation between a subset of machines.

Initially, we are interested in trying to find common trends within the data which can split the cluster into smaller groups defined by similar characteristics and features. The cluster data is formed from machines of many different generations and builds with varying performance and storage specifications which can lead to widely different resource usage and limits. As mentioned before in Section 2.2, other basic characteristics have been also been observed by researchers who have analyzed the trace dataset extensively and they have concluded to these traits have trends that we can potentially utilise for our forecasting model to accurately predict resource usage: [6]

- **Resource Usage Stability** - when tasks run for a long period of time, their resulting resource usage is usually stable.
- **Short Job Duration** - significantly shorter jobs account for an extremely low percentage to the overall utilization.
- **Repeated Job Schedules** - some jobs are run repeatedly at a certain point in time with similar resource uptake.
- **Machine Attributes** - machines consist of a few different platforms, which classifies the build and specification of the machine

Initially, we will use these characteristics as a baseline to start our findings. The goal of the project is to be able to identify a strong relationship between potential subsets of machines that we can use for our forecasting model later. Therefore, further analysis into the nature of the cluster trace dataset is required to find possible characteristics we can use to differentiate between subgroups of machines. We will start by investigating the spatial correlation between machine pairs and also analyse resource usage percentage within an indiviudal machine to investigate the level of temporal correlation present.

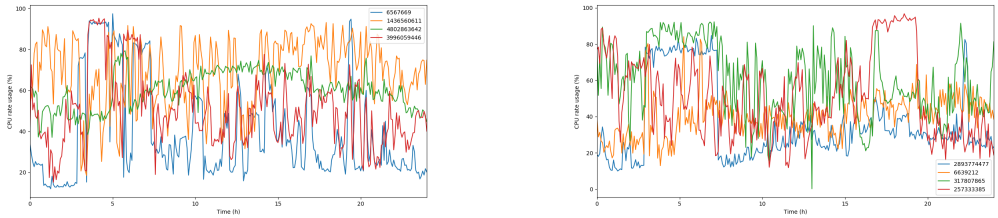


Figure 3.1: Visualising the percentage of CPU usage over a time period for 10 randomly chosen machines

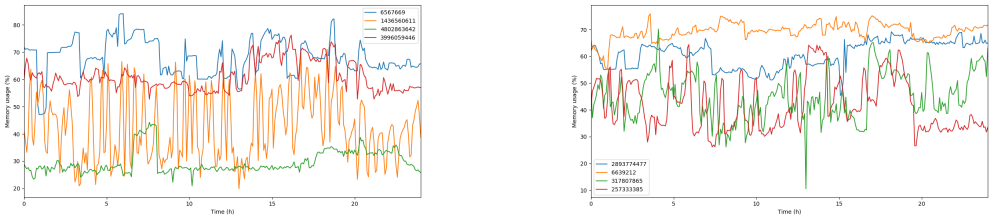


Figure 3.2: Visualising the percentage of memory utilisation over a time period for 10 randomly chosen machines

3.2.2 Python Analyzer Implementation

Since the trace is in stored in a compressed CSV format, it makes it straightforward to read and analyse as there are existing Python modules that can help with parsing the large amounts of data. Therefore, we design and develop a trace analyzer in Python, based on a design outlined in a detailed article [12] which provides details on how to efficiently break down and research the cluster trace efficiently.

For the most of this project, we will only be analyzing CPU usage trends within machine workloads. To effectively analyse the CPU usage of machines within the trace, the analyzer will randomly sample machine IDs from the cluster of machines as analysing over thousands of machines at once will make it hard to interpret and analyse the data. On top of this, the analyzer will also randomly select a starting timestamp to sample from, conducting an analysis over a section of the trace over a specified sample time period, usually using a period duration of 24 hours. We also store a machine index where we can keep track of each machine's unique ID number, platform ID and their CPU and memory capacities.

As there are a large amount of records within the sampling time period, the analyzer processes the trace data to obtain CPU and memory usage statistics for each machine by sampling the usage data at regular time intervals of 5 minutes. This enables us to calculate the percentage of resource capacity used at each sample moment of time and we can then use this as a new base for a univariate timeseries dataset. Using this, we can analyse trends such as the cross correlation between specific pairs of machine timeseries and the autocorrelation within an individual machine workload sequence. Later on, we will use the same generated timeseries dataset to train our forecasting model.

3.2.3 Initial Observations and Analysis

From a first glance, by simply observing the plain data corresponding to the resource usage of several randomly chosen machines in figures 3.1 and 3.2, we can conclude very little about the data due to the complexity of the scheduling process. At a surface level, we can identify that there are small trends where the fluctuations in the resource usage within a machine would correspond to another, which is most likely due to the way the jobs are scheduled to each machine. However, we cannot simply conclude the relationship from these graphs alone, this feature of the cluster trace would have to be further analysed by testing the cross-correlation between their datasets.

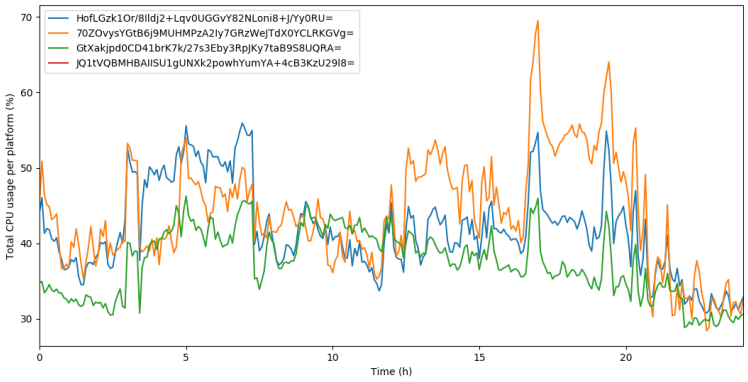


Figure 3.3: Graph showing the average percentage of CPU used for across all machines with a specific platform ID

3.2.4 Platform ID Trend Analysis

One of the key characteristics of the machine dataset we investigated was whether there was a relationship between machines that were of the same platform ID, which cagetorised machines based on their builds and specifications. The following table outlines the details regarding the 4 hashed platform ID that a machine can be a part of:

Platform ID Specifications			
Platform ID (hashed)	CPU cores available	Memory available	Amount of machines with platorm ID
`HofLGzk1Or/8Ildj2+Lqv0UGGvY82NLoni8+J/Yy0RU=`	0.5	0.2493 - 0.749	11632
`70ZOvysYGtB6j9MUHMPzA2Iy7GRzWeJTdX0YCLRKGVg=`	0.25	0.2498	123
`GtXakjpd0CD41brK7k/27s3Eby3RpJKy7taB9S8UQRA=`	1	1	796
`JQ1tVQBMHBAIISU1gUNXk2powhYumYA+4cB3KzU29l8=`	n/a	n/a	32

Figure 3.3 illustrates the overall workload schedule over a sample time period, across all the machines which have been sorted by platform ID. We can observe that depending on which platform ID the machine is assigned to, it will have slightly varying job schedules with different amounts of resource usage. Generally, the idea is that machines with higher capacities and performance specifications are reserved for a different subset of tasks. Therefore, we can use this characteristic to analyse if there exists any potential spatial correlation between specific machines of the same platform ID.

3.2.5 Spatial Correlation Analysis

To achieve a better understanding of the relationship between different machines workloads, we analyse the cross correlation coefficient between chosen machine pairs using Pearson's correlation coefficient.

$$r = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2(y_i - \overline{y})^2}}$$

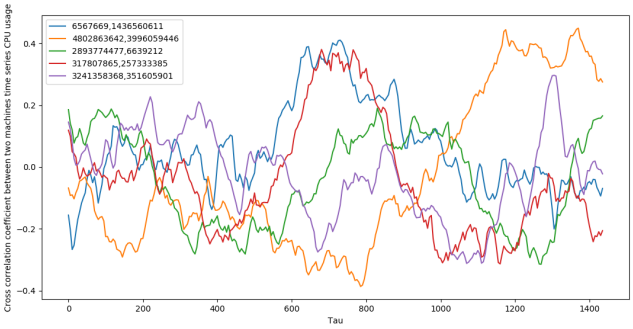
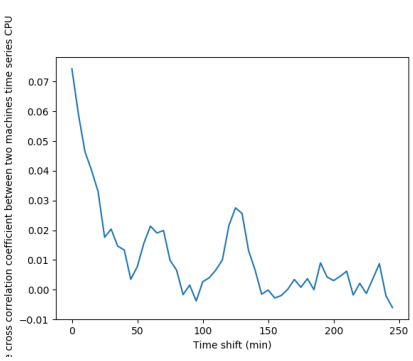
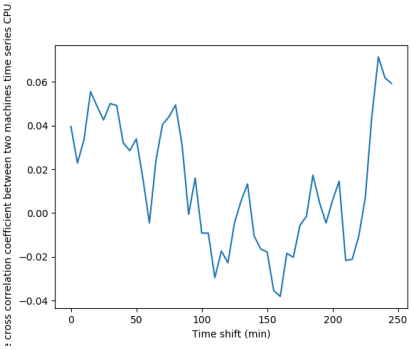


Figure 3.4: Visualising the cross correlation coefficient of 5 machine pairs over a scaling period of time-shift



(a) 500 machine pairs, randomly sampled



(b) 50 machine pairs, platform ID `GtXakjpd0CD41brK7k/27s3Eby3RpJKy7taB9S8UQRA=`

Figure 3.5: Visualising the average cross correlation coefficient between machine pairs

Initially, we start by selecting random machine pairs regardless of their characteristics and attempt to find if there exists a spatial correlation between randomly chosen machines. We also applied a time-shift scale to one of the machine sequence data within the pair to see if this change would affect the correlation in a significant manner. Figure 3.4 illustrates an example of the data we process. Furthermore, we use the cross correlation of randomly selected machine pairs as a baseline to compare against other machine pairs that have been grouped by certain features such as platform ID.

Immediately, we can tell from figure 3.5a that there is a significantly weak cross correlation between many of the machine pairs with an average correlation coefficient of approximately **0.07** at zero time-shift.

We mostly expected that there would be a weak spatial correlation between machine pairs that were randomly selected out of the entire cluster. We can now use this as a baseline to compare against when attempting analyse the spatial correlation within machines that have been grouped together by a certain feature. Therefore, we also analysed selected sample machine pairs of the same platform ID but as the results show in Figure 3.5b, the average correlation coefficient is approximately **0.04** at zero time-shift. These results don't differ by a significant margin when compared to the results obtained from random sampled machine pairs.

The low amount of cross correlation between machine sequence data indicates that there is a lack of overall spatial correlation between machines within the cluster. There may be other alternative features which can improve the results of this analysis but the challenge is being able to identify a suitable characteristics that we can use to group specific machines by.

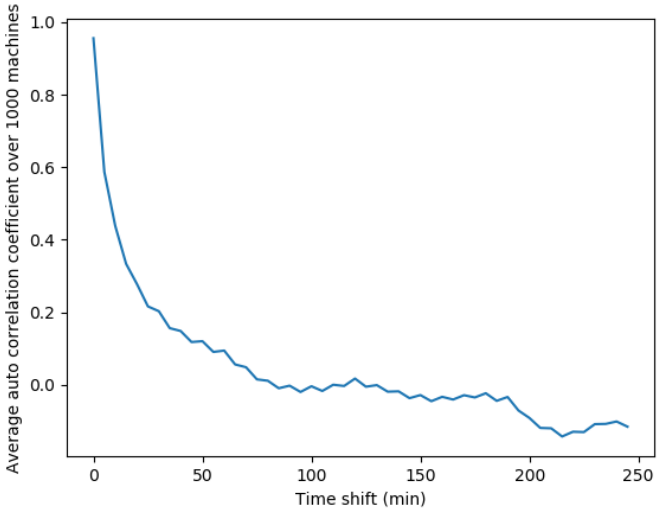


Figure 3.6: Graph showing the average auto correlation coefficient over 1000 machines

3.2.6 Temporal Correlation Analysis

Another analytical approach that we took in order to investigate the possible characteristics is to analyse if there exists any temporal correlation within an individual machine's time-series. If repeated job schedules and resource usage stability can be used as characteristics to identify relationships within the timeseries data, there would exist a relatively strong autocorrelation within the data as we would be able to observe a common pattern which would resemble a job schedule. We start by analyzing random individual machine sequences and investigating the autocorrelation within the data, where we calculate the correlation coefficient between the original data and its time-shifted variant.

Figure 3.6 displays the findings of our temporal correlation analysis and it shows a relatively strong autocorrelation that exists within machines regardless of platform builds. This indicates that with each machine workload schedule, especially during the first hour, there exists a significant temporal correlation. This is most likely due to a common job schedule pattern that the machines are assigned to, which results in a relatively stable autoregressive data suitable for a forecasting model.

3.2.7 Discussion

In conclusion, through the analysis of machine timeseries data of CPU usage, there was an evident low amount of spatial correlation between machines. Even when attempting to group machines by platform ID, it still resulted in no clear cross correlation between the machine pair timeseries data. There is an innate difficulty when it comes to attempting to group these machines by certain particular characteristics due to the complex structure of the cluster and its job scheduling patterns. In the future, if there was a better understanding of the underlying features within the cluster trace, it can result in finding an improved spatial relationship found between subsets of machines.

Alternatively, the analysis showed that there was a strong amount of temporal correlation within individual machine workloads regardless of other machine characteristics such as platform ID. This indicates that the individual machine workload timeseries data can be potentially used for training a forecasting model for each individual machine timeseries without requiring the context data of the other surrounding machines. Therefore, we will use this autoregressive feature as a basis for our network architecture to create a foreceasting model to predict future resource usage within an individual machine workload's lifecycle.

Chapter 4

Recurrent Neural Network Model

From the conclusion of the trace analysis, we propose a linear regression approach to create our forecasting model. We create a recurrent neural network as LSTM units are able to model timeseries data effectively due to it being able to retain an internal hidden state which is particularly useful for learning autoregressive data.

Initially, we will build and train our model based on one machine workload timeseries based on a randomly sampled machine ID. After selecting our network architecture and optimising the model's hyperparameters, we will use the same model parameters to train on different workload sequences from other machines and evaluate the performance of our forecasting model.

4.1 Data preprocessing

4.1.1 Dataset Division

With access to a large amount of samples from the cluster trace, we can choose any random sample starting timestamp to process the trace data from. We will initially train our model on the processed timeseries data of machine ID `350588109` which has the common platform ID of `HofLGzk1Or/8Ildj2+LqvOUGGvY82NLoni8+J/YyORU=`. We will sample two different starting timestamps for a period of 24 hours. Our training and validation dataset was taken and split from one section of the cluster trace whilst the testing dataset was taken from a completely different section of the trace.

The following describes how we will split and use the machine timeseries dataset:

- **Training dataset (timestamp 1574603581118, sample duration of 12 hours)** - this dataset is used to train the RNN model by minimising the error given by the loss function after predicting CPU usage values on the training set.
- **Validation dataset (timestamp 1617803581118, sample duration of 2 hours)** - this dataset is used for tuning the hyperparameters of the model. When we train the model using the training data on different parameter values, we evaluate the combination of parameter values chosen by testing it on the validation set.
- **Testing dataset (timestamp 1370398689445, sample duration of 24 hours)** - after we have fully optimised the model's hyperparameters, we evaluate the forecasting model's predictive performance on the test dataset, providing us a metric on how well the model performs on unseen data. The test dataset is solely used for evaluating the model's performance.

4.1.2 Timeseries Data Feature representation

Before we can model the timeseries data, we have to prepare the dataset from the cluster trace to enable the LSTM to learn an input sequence of observations and map it to an output value. Therefore, we have to modify the univariate timeseries data, that we have collected and processed from the trace using the Python analyzer, in a format which can be used by the LSTM cell. [13]

For the neural network to be able to learn a timeseries sequence, we need to feed input data where each sample is a sequence of observed values of a certain timestep length. This sequence will be our set of ordered features for the model which will then predict an output CPU usage value from the input sequence. We do this by modifying the processed sequence from the Python analyzer to have dimensions `(no_of_samples x input_sequence_length)` where the input sequence length is the number of timesteps the LSTM will learn from to map to an output value.

For example, if we were to have a sequence dataset `[23, 55, 81, 93, 89, 82, 63, 47, 28, 22]` and wanted to take 5 samples for the LSTM to learn 3 input timesteps per sample, we would modify the input data into the following format:

$$\begin{bmatrix} 23 & 55 & 81 \\ 55 & 81 & 93 \\ 81 & 93 & 89 \\ 93 & 89 & 82 \\ 89 & 82 & 63 \end{bmatrix}$$

where the output test data would have feature values `[93, 89, 82, 63, 47]` which we can use to evaluate against the loss function.

4.2 Performance Metrics

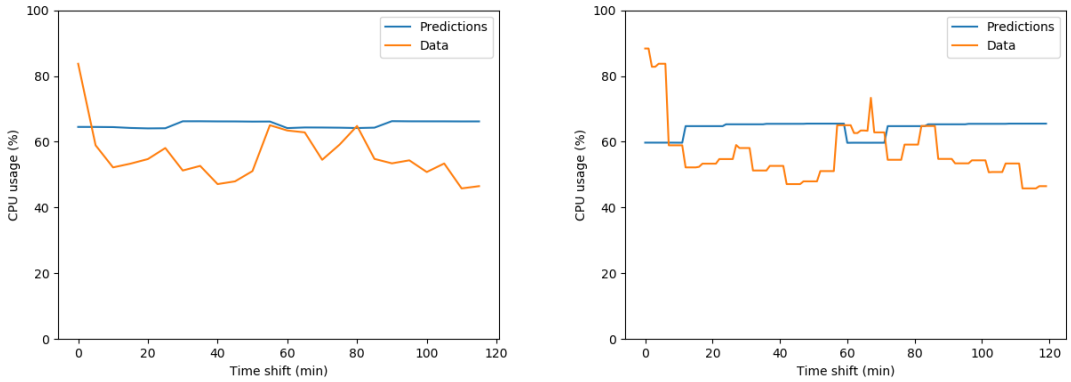
To be able to select the appropriate hyperparameters when building our neural network, we require suitable metrics to evaluate the performance of our machine learning model. Typically for linear regression, there are two appropriate loss functions for this approach:

- Mean Square Error (L2 Loss) :=
$$\frac{1}{n}\Sigma_{i=1}^{n}(y_i - \hat{y}_i)^2$$
- Mean Absolute Error (L1 Loss) :=
$$\frac{1}{n}\Sigma_{i=1}^{n}|y_i - \hat{y}_i|$$

In this case, we will be using the MSE loss function due to the complex nature of the timeseries sequence and being able to produce a more stable solution compared to MAE. The aim when training the model is to minimise the loss function when predicting future CPU usage percentage values on the training set. We then test our model hyperparameters by evaluating the model predictions against the validation set, which has not been seen by the forecasting model.

4.3 Hyperparameter Optimisation and Network Architecture

The model architecture is comprised of a 1D convolutional neural network (CNN) which feeds data into a LSTM network. The 1D CNN consists of a 1D convolutional layer followed by a 1D max pooling layer. The output representation of the timeseries data generated by the 1D CNN is then fed into the LSTM cell to keep a store of the state at every step. The model also uses a linear activation function for the output layer which is suitable for this regression problem.



(a) 5 min. sampling rate, MSE = 152.76

(b) 1 min. sampling rate, MSE = 142.07

Figure 4.1: Comparing the performance of the basic RNN model with a 1 minute and 5 minute sampling rate on the machine timeseries dataset `(ID: 350588109)`

The following table outlines the hyperparameters that were chosen for the RNN model, specifically tuned and tested with one individual machine timeseries set `(ID: 350588109)`:

Hyperparameters for RNN	
Parameter	Value
Number of epochs	500
Loss function	Mean squared error (MSE)
Batch size	60
Input sequence length (timesteps)	5
Sampling rate	1 minute
Number of hidden LSTM layers	2
Number of neurons in hidden layers	h1: 128, h2: 64
1D convolutional kernel size	4
1D MaxPool kernel size	2
Output activation function	Linear
Optimizer	Adam
Learning rate	0.00059
L2 regularization penalty (weight decay)	0.4

4.3.1 Increased sampling rate

One of the immediate problems that was observed when training the model was that our initial sampling rate of 5 minute intervals provided too few samples for the RNN model to train on. When processing a sample period of 1 day, the sampling rate of 5 minutes only provides 289 samples to use for the training and validation datasets. Therefore, we have decided to train the model with a new set of processed data with a sampling rate of 1 minute, which provides a total of 1441 samples over the same sampled time period.

Figure 4.1 compares the predictive performance of a basic RNN model trained with datasets that were processed by two different sampling rates. There is an improvement in MSE with the higher sampling rate as the model can attempt to learn more features of the timeseries. We can observe that the RNN training on the low sampling rate training set fails to learn any notable features due to the lower number of samples which can be observed and learned. Therefore, based on our findings, we have chosen to use a sampling rate of 1 minute for our training and validation datasets.

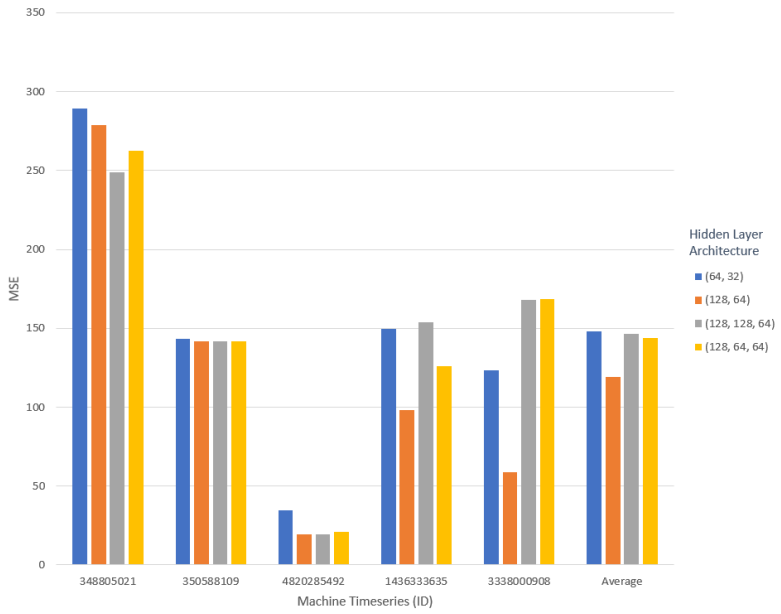


Figure 4.2: Evaluating LSTM architecture over different machine workload timeseries

4.3.2 LSTM Hidden Layers

There were several LSTM hidden layer architectures that we decided to train and test with as different number of layers and neurons per layers can affect how the RNN model can learn and predict future sequences. During our initial testing of the possible LSTM architectures, we had difficulty comparing the performance on the validation set of our initial machine timeseries. This was due to the negligible difference in MSE over the different architecture for this specific set of timeseries data.

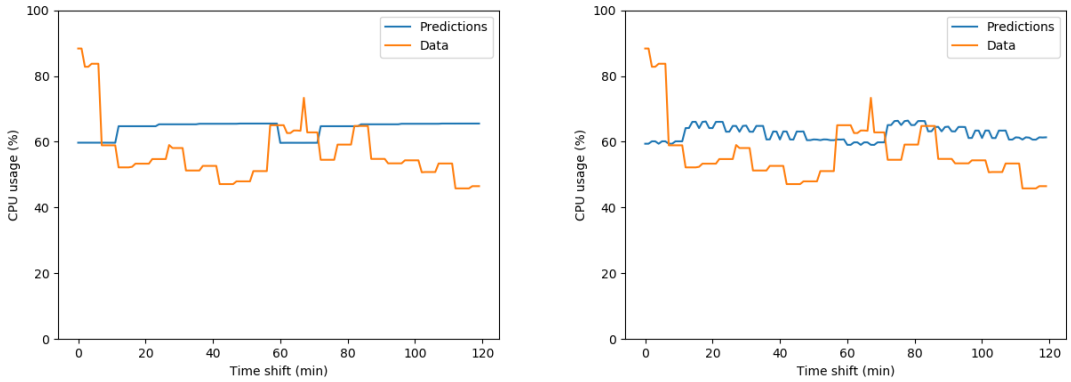
Therefore, we decided to train our model on other machine timeseries to further evaluate the different LSTM architectures. We sampled from the same timeperiod for the training and validation sets and kept the other hyperparameter values the same such as learning rate and L2 weight penalty. This ensures the training and validation remain consistent despite the model potentially overfitting on the different timeseries sets.

Comparing MSE values for different LSTM architectures				
LSTM architecture (col.) Machine ID (row)	(64, 32)	(128, 64)	(128, 128, 64)	(128, 64, 64)
350588109	143.163	141.965	141.673	141.701
348805021	289.493	278.778	248.639	262.673
4820285492	34.659	19.203	19.199	20.984
1436333635	149.488	97.981	153.993	126.080
3338000908	123.556	58.838	168.168	168.725
Average	**148.07**	**119.35**	**146.33**	**144.03**

These evaluation results (Figure 4.2) provide extra insight into the performance of the different architecture combinations. Increasing the number of neurons per layer allows for the LSTM allows for the model to learn more complex features much more effectively. Furthermore, having additional hidden layers doesn't appear to result in much improvement across the other machine timeseries sets whilst also increasing the time it takes for one forward pass.

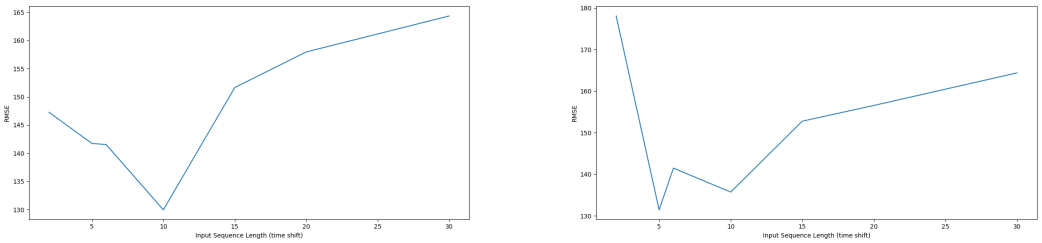
4.3.3 1D Convolutional Layer

Due to the complex nature of the timeseries data, we can implement a 1D convolutional neural network (CNN) which can learn an output representation of the input features from the timeseries data before feeding this representation into the LSTM unit which can effectively learn the CNN representation and further train the autoregressive model.



(a) Without 1D conv. layer, MSE = 142.088 (b) With 1D conv. layer, MSE = 119.557

Figure 4.3: Comparing the performance of the RNN model with the addition of 1D convolutional layer with machine timeseries set ID: `350588109`



(a) Without 1D conv. layer (b) With 1D conv. layer

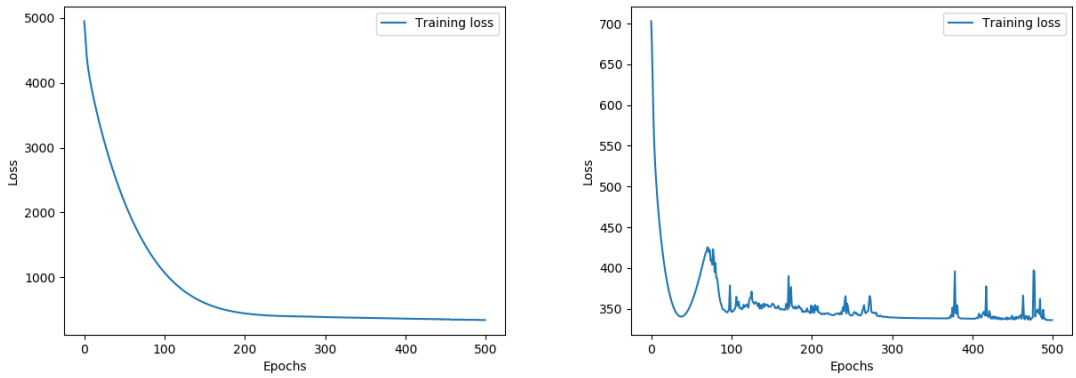
Figure 4.4: Evaluating different input sequence lengths with different architectures

Figure 4.3 shows the improvement in predictive performance when implementing a 1D convolutional layer with a significantly lower MSE value. With the new implementation of the 1D CNN layer, it allows the forecasting model to be able to generalize new timeseries data much more effectively as it will be able to recognise and fit more complex features within the a machine workload sequence.

4.3.4 Input sequence length (timesteps)

As mentioned before, changing the number of input timestep features we feed into the LSTM allows us to vary the amount of timesteps the model learns before predicting the next value. Throughout the process of training and tuning the hyperparameters of the model, we constantly tested and modified this parameter value due to the varying performance results obtained from testing different network architectures, particularly when implementing the 1D CNN layer.

Since we opted on using a batch size of 60, we could only use timestep values of `[2, 5, 6, 10, 15, 20, 30]` to fit the dimensions of the input data tensor. Figure 4.4 highlights that the optimal input sequence length is either 5 timesteps with the inclusion of the convolutional layer, or 10 timesteps without it. This maybe due to the model requiring less timestep features to train effectively when the input data has been processed by the CNN beforehand. As we have discussed before, creating a convolutional LSTM model would be able to predict complex timeseries data much more effectively compared to a standard LSTM model. Therefore, we will use an input sequence length of 5 for the RNN model to improve the overall predictive performance.



(a) Machine ID: `350588109`

(b) Machine ID: `348805021`

Figure 4.5: Training loss over different machine IDs with optimal parameters

4.3.5 Additional Hyperparameter adjustment

When tuning other model hyperparameters such as learning rate and the penalty for L2 regularization, we optimise these values primarily based on the training loss graph to ensure the model doesn't overfit the training data and learn the input features quickly and effectively. We finalised our optimal hyperparameter values based on our initial machine timeseries set (ID: `350588109`).

One of the problems that appeared from this is that when the model tries to learn other new workload sequences from different machines, the optimal value for parameters such as learning rate are significantly less effective, with signs of the model overfitting the timeseries data (Figure 4.5). If we were to optimise the individual training of all the possible machine timeseries, we would have to find new optimal hyperparameters which differ from the initial optimal parameters we selected. Regardless, we still opted to use these hyperparameters for the final RNN model due to the performance of the model still being relatively effective despite the slight loss in training.

Chapter 5

Implementation

5.1 Design Overview

Initially, when designing the implementation of our forecasting model, we have to be able to generate the timeseries datasets before we are able to start building and training our model. As mentioned before, we obtain our machine CPU usage timeseries data from processing sections of the cluster trace through the use of the Python analyzer, outlined in Section 3.2.2. The analyzer is able to parse the large amounts of CSV data from the trace (approx. 41 GB) and is optimised to efficiently sample task resource usage within the specified time period at regular 1 minute intervals. After collecting and processing the data samples, it is stored in a CSV format for later use when training the model.

After generating the dataset, we have to preprocess the format of the input in order to feed it into the LSTM model as discussed in Section 4.1.2. The modified dataset is then split into training and testing datasets as described in Section 4.1.1, and the forecasting model is trained individually on a specific machine timeseries dataset (initially starting with machine ID `350588109`). We then evaluate the forecasting model's predictions by testing it against either the validation set if we are tuning the model's hyperparameters as outlined throughout Section 4.3, or evaluating them against the testing dataset if we are assessing the optimised model's predictive performance. The results of the evaluation based on different periods of the dataset are then processed and outputted.

5.2 Technologies and Frameworks

All parts of the trace analyzer components were implemented using the `Python` programming language. Python has several useful packages when performing data analysis on large datasets. In

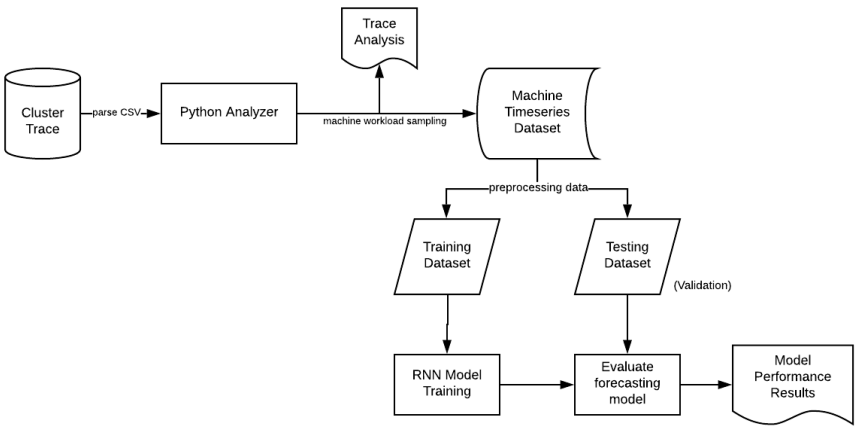


Figure 5.1: Data flow diagram of the RNN model and Python Analyzer

particular, the `Numpy` and `Pandas` packages have libraries that offer good utility tools and interfaces to efficiently parse and manipulate large datasets to conduct thorough analysis.

When implementing our forecasting model, we used the `PyTorch` framework to implement the RNN model. PyTorch contains many libraries which provide utility functions and classes to help construct and train our neural network. There also exists a large amount of documentation and online resources to assist us with any unfamiliarity we may have during the project.

5.3 Limitations

There were several performance issues when trying to generate the machine datasets for the forecasting model. Despite only having to generating the datasets once before storing them for consistent use, processing the relevant sections of the trace can take up to 1-2 hours to sample 24-hour period for a few machine workloads. This amount of processing time significantly increases when we scale up the number of sampled machine workloads during the same time period. However, once the datasets have been generated, we no longer need to reproduce the datasets and if necessary modifying the CSV data is a relatively quick process.

Another bottleneck for our system is that the forecasting model has to learn from a relatively large sample size with many features to perform effective predictions which can take a long time to process and train. In addition to this, we have to run the training of our model with the same parameter values separately for different machine workload sequences to be able to individually predict their future timeseries. Therefore, if we wanted to evaluate the predictive performance of over 1000 different machine timeseries, it would take an extremely long amount of time before we processed all the prediction results of each machine. For now, the current architecture is sufficient in predicting individual machines timeseries data.

Chapter 6

Evaluation

6.1 Evaluation method and metrics

As mentioned before, the time period of the cluster trace is extremely large which allows us sample many different periods of time to train our model on. After training and improving the neural network model using the training and validation sets, our goal for evaluation is to observe how the model performs against unseen data.

Therefore, we will evaluate the success of the trained model by testing the forecasting model on the unseen testing dataset as described in Section 4.1.1. Our tests will also include 9 different machine timeseries, equally sampled from 3 different platform IDs to provide further insight into the cluster trace. We will use the follow metrics when conducting the evaluation of our model:

- Root mean squared error (RMSE) :=

$$\sqrt{\frac{1}{n}\Sigma_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

- Mean absolute error (MAE) :=

$$\frac{1}{n}\Sigma_{i=1}^{n}|y_i - \hat{y}_i|$$

The testing set contains samples from a time period with a starting timestamp of `1370398689445` and a duration of 24 hours. The dataset not only consists of data which is unseen by the forecasting model but is also sampled from a starting point which is significantly different from our training and validation datasets (around 2 weeks difference), which provides us a more representative quantification of our model's performance. With this, we can effectively conduct an unbiased evaluation of our forecasting model.

6.2 Results

The following table shows the results of the model's predictive performance across 9 different machine timeseries datasets, measure using RMSE and MAE. Figure 6.1 also helps visualise the table data more effectively, highlighting the difference in performance across different platforms. Further visualisation of the model's performance can be found in Appendix A.

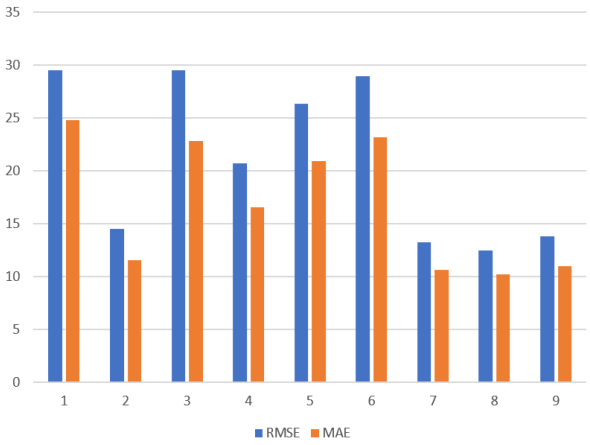


Figure 6.1: Visualising evaluation results for each machine timeseries

Evaluation results of sampled machine time series				
Index ref.	Machine ID	Platform ID	RMSE	MAE
1	348805021	HofLGzk1Or/8Ildj2+LqvOUG GvY82NLoni8+J/YyORU=	29.501	24.763
2	350588109	HofLGzk1Or/8Ildj2+LqvOUG GvY82NLoni8+J/YyORU=	14.480	11.506
3	1436333635	HofLGzk1Or/8Ildj2+LqvOUG GvY82NLoni8+J/YyORU=	29.501	22.770
4	3338000908	7OZOvysYGtB6j9MUHMPzA2Iy 7GRzWeJTdXOYCLRKGVg=	20.697	16.527
5	1390835522	7OZOvysYGtB6j9MUHMPzA2Iy 7GRzWeJTdXOYCLRKGVg=	26.306	20.914
6	1391018274	7OZOvysYGtB6j9MUHMPzA2Iy 7GRzWeJTdXOYCLRKGVg=	28.891	23.176
7	4820285492	GtXakjpdOCD41brK7k/27s3E by3RpJKy7taB9S8UQRA=	13.195	10.63
8	5015788232	GtXakjpdOCD41brK7k/27s3E by3RpJKy7taB9S8UQRA=	12.428	10.187
9	4874102959	GtXakjpdOCD41brK7k/27s3E by3RpJKy7taB9S8UQRA=	13.797	10.937

From analysing the results, we can immediately observe that there is a significant less error in predicting machine workload sequences that belong to platform ID `GtXakjpdOCD41brK7k/27s3Eb y3RpJKy7taB9S8UQRA=`. The following table displays the average RMSE and MAE values across each platform ID and the differences between them. This further concludes our findings of the model performance over the various timeseries data.

Additional evaluation metrics based on platform ID			
Platform ID	Avg. RMSE	Avg. MAE	Difference
HofLGzk1Or/8Ildj2+LqvOUG GvY82NLoni8+J/YyORU=	24.49	19.68	4.81
7OZOvysYGtB6j9MUHMPzA2Iy 7GRzWeJTdXOYCLRKGVg=	25.30	20.21	5.09
GtXakjpdOCD41brK7k/27s3E by3RpJKy7taB9S8UQRA=	13.14	10.58	2.56
Overall	**20.98**	**16.82**	**4.16**

6.3 Discussion

Comparing our results with current forecasting models, we had trouble finding other notable results of existing models within this space as most of them are designed to predict different aspects of the cluster trace. Therefore, we will only assess our results based on our observations and analysis.

Initially, we can immediately observe that machines belonging to the platform ID, `GtXakjpd0C D41brK7k/27s3Eby3RpJKy7taB9S8UQRA=`, perform significantly better on average compared to the other platforms. This is possibly due to these machines having the highest resource capacities and are solely reserved for demanding jobs/tasks. There also exists a slightly lower number of these machine builds that are active within the cluster which would further imply that they would have similar job schedules that are predictable and fairly fault-tolerant, resulting in less outliers impacting our performance.

The reason we use both RMSE and MAE metrics to evaluate the model is to observe how the forecasting model performance would be affected by outliers within the timeseries. As the MAE value is less sensitive to large error values compared to RMSE, we can use the difference between these values the assess the impact of outliers on the model's predictions.

We can observe these occurrences in the results with machine timeseries index 3 and 6 (figures A.4 and A.7 in Appendix A), where the difference between RMSE and MAE is around 5.7 - 6.7 which are significantly higher compared to the value of 2.6 from the performance of machine timeseries index 7 (Figure A.3). From this, we can observe that the actual data points within sequence heavily fluctuate, making it harder for the model to adjust their predictions correctly. These large unsuspecting changes in data may be caused by random noise data within the timeseries or due to the cause of machine failure within the cluster.

6.3.1 Model Limitations and Improvements

There are several possible aspects of the model we can change to improve the predictive performance across each machine timeseries. One of the main issues that occurred when tuning our model hyperparameters was that when applying the model to various other machine timeseries, we would need to train them with different parameter values to avoid problems such as overfitting.

A suggestion to optimise our model performance is to implement hyperparameter optimisation algorithms such as Bayesian Optimisation which can automatically process the optimal learning rate and L2 weight penalties. This would obviously take much longer to train each model against a new individual machine timeseries but it would serve as a slight improvement to the training strategy.

An alternative suggestion is to implement an 'early stopping' mechanism when training our model, which allows us to retain the same parameter values such as learning rate whilst also eliminating the overfitting problem to a slight extent. There would still be an issue with training the model extremely fast and stopping it early but without learning any of its underlying features.

Finally, we could also train the model with collective machine timeseries sets from multiple machine IDs rather than individually. These would be specific machines that have been grouped together due to a certain relationship or characteristic. This can potentially improve the model's performance whilst also being able to train over a large amount of machines efficiently. However, as discussed earlier, there is a lot of difficulty in characterising and grouping the machines into subsets without further analysing the complex nature of the cluster trace.

Chapter 7

Conclusion

7.1 Overview

The initial goals of the the project were to:

- Extensively analyse and provide extra insight into the Google cluster trace dataset
- Investigate whether there exists a strong spatial correlation between machines based on certain job/task features or machine characteristics
- Investigate whether there exists a strong temporal correlation within an individual machine's time series data
- Investigate the application of an appropriate machine learning model which can predict future CPU and memory usage within each machine timeseries

Throughout this project, we have conducted extensive research and analysis into the cluster trace dataset. Through the use of our built Python analyzer, we managed to explore the structure of the trace and analyse sampled periods of time within the trace data. With the assistance of this tool, we managed to observe several machine workload trends. Specifically, we have mostly focused our efforts into investigating trends with percentage of CPU usage per machine.

We also attemped to find suitable features and characteristics we can create subgroups of machines from. Furthermore, we analyse the extent of spatial correlation that exists between machines within the cluster. We started by analysing whether we can group machines by platform ID and analysed if there existed a strong relationship within their timeseries sets. We also analysed the temporal correlation within individual machine timeseries to investigate whether there exists a predictable job schedule which can be related to trends in resource usage. Ultimately, we concluded that the cluster trace had overall weak spatial correlation on a surface level and much stronger temporal correlation, which we used as a base to investigate the application of a machine learning model.

After concluding the analysis of the cluster trace dataset, we implemented a recurrent neural network model which used a 1D convolutional layer applied on top of a LSTM cell. We developed effecient methods of generating and preprocessing data samples to train and test our model with. After extensive tuning and optimisation of our model architecture and hyperparameters, we tested our model on unseen data to be able to predict future CPU usage trends of an individual machine within the cluster and evaluated the performance using suitable metrics.

In conclusion, the project managed provide additional insight into the structure and features of the cluster trace whilst also proposing a new effective machine learning model, which utilises a CNN layer and LSTM unit, to predict future resource usage to a reliable degree. Furthermore, we can further explore the potential of apply machine learning models to other aspects of distributed systems.

7.2 Future work

7.2.1 Recurrent neural network improvements

As discussed in Section 6.3.1, there are several improvements we can make to our current model that have not been fully realised and tested yet such as implementing a Bayesian Optimisation algorithm to tune additional parameter values or utilising 'early stopping' when training the forecasting model.

7.2.2 Forecasting model for subgroups of machine timeseries datasets

Our current forecasting model is effective for predicting on individual machine workload timeseries data but the problem arises when we want to predict resource usage within an entire cluster of machines. As mentioned in Section 5.3, if we were to currently predict an entire cluster distributed system's resource usage activity, it would be too inefficient and ineffective to train one forecasting model individually for each of the 12,583 machine timeseries datasets. A future improvement would involve further analysis of the cluster trace machine features and characteristics which we can use to define potential subgroups of machines. We can then use their collective timeseries dataset for training an improved forecasting model.

7.2.3 Alternative Machine Learning approaches

The project mostly focused on applying linear regression and deep learning methods to create a forecasting model. Potentially better results could have been achieved if we were to investigate and apply other machine learning techniques such as using support vector machines. We naturally favoured using artificial neural networks as it extremely common to use these models to solve a linear regression problem.

7.2.4 Additional datasets

Despite the cluster trace being the primary focus of the project, a future goal would be to train and test this forecasting model architecture on more timeseries datasets collected from other large-scale distributed systems.

7.2.5 Prediction of other cluster trace statistics

With this forecasting model highlighting the potential of machine learning approaches within this space, a future goal would be to create a forecasting model that can predict other aspects of the cluster data such as the amount of resource requested by a machine, which machine a job/task scheduled is allocated to, and the likelihood for machine failure.

Bibliography

[1] DAIS-ITA. Dais-ita project aims. http://www.commsp.ee.ic.ac.uk/~wiser/dais-ita/aim.html.

[2] Kin K. Leung Bong Jun Ko Tiffany Tuor, Shiqiang Wang. Online collection and forecasting of resource utilization in large-scale distributed systems. 2019.

[3] Joseph Hellerstein Charles Reiss, John Wilkes. Google cluster-data: Clusterdata2011_2. https://github.com/google/cluster-data.

[4] Joseph Hellerstein Charles Reiss, John Wilkes. Google cluster-usage traces: format + schema. https://drive.google.com/file/d/0B5g07T_gRDg9Z0lsSTEtTWtpOW8/view.

[5] Shuchi Sethi Mansaf Alam, Kashish Ara Shakil. Analysis and clustering of workload in google cluster trace based on resource usage. 2016.

[6] Gregory R. Ganger Rand H. Katz Michael A. Kozuch Charles Reiss, Alexey Tumanov. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. 2012.

[7] Javier Alonso Stefano Sebastio, Kishor S. Trivedi. Characterizing machines lifecycle in google data centers, performance evaluation (2018). 2018.

[8] Mohammed Sunasra. Performance metrics for classification problems in machine learning. https://medium.com/greyatom/performance-metrics-for-classification-problems-in-machine-learning-part-i-b085d432082b.

[9] Aditya Mishra. Metrics to evaluate your machine learning algorithm. https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234.

[10] Ankita Jain Ruchika Malhotra. Software effort prediction using statistical and machine learning methods. 2011.

[11] Ivan Vasilev. A deep learning tutorial: From perceptrons to deep networks. https://www.toptal.com/machine-learning/an-introduction-to-deep-learning-from-perceptrons-to-deep-networks.

[12] Mark Stillwell. First steps exploring the google cluster dataset with ipython. http://blog.stillwell.me/blog/2013/07/15/first-steps-exploring-the-google-cluster-dataset-with-ipython/.

[13] Jason Brownlee. How to develop lstm models for time series forecasting. https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/.

[14] Anup Bhande. What is underfitting and overfitting in machine learning and how to deal with it. https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76.

[15] Georgios Drakos. Cross-validation. https://towardsdatascience.com/cross-validation-70289113a072.

Appendix A

Evaluation of forecasting model - Predictions vs Data

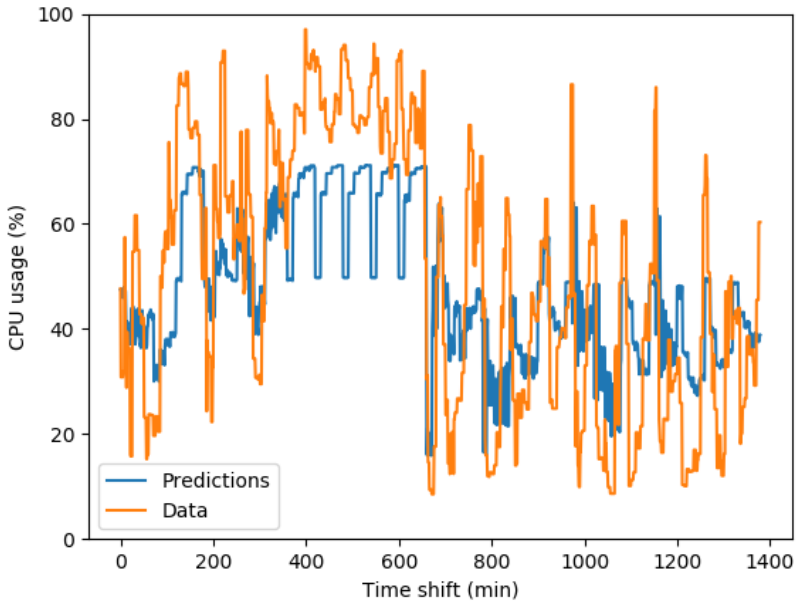


Figure A.1: Machine ID: 348805021

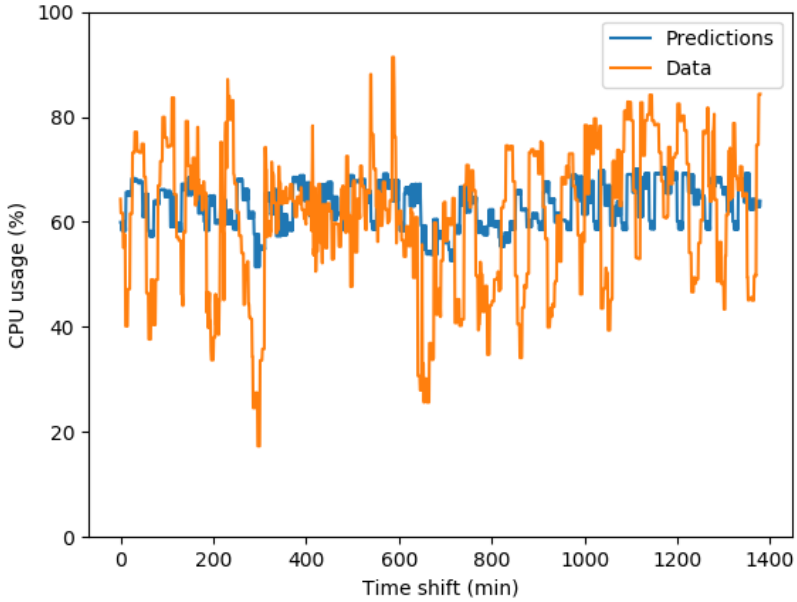


Figure A.2: Machine ID: 350588109

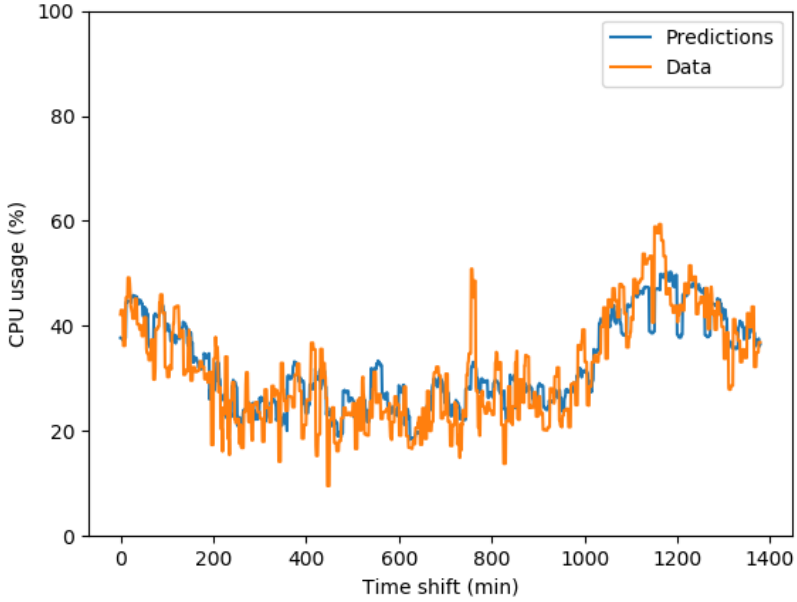


Figure A.3: Machine ID: 4820285492

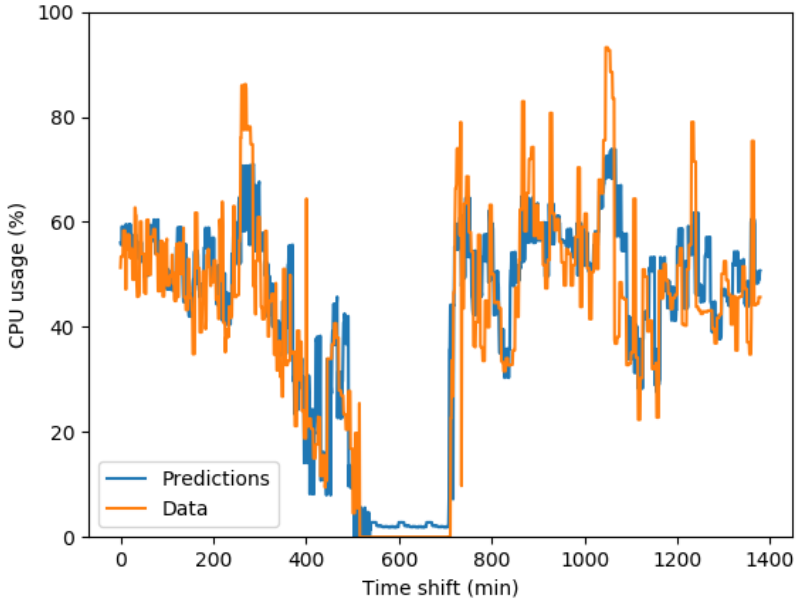


Figure A.4: Machine ID: 1436333635

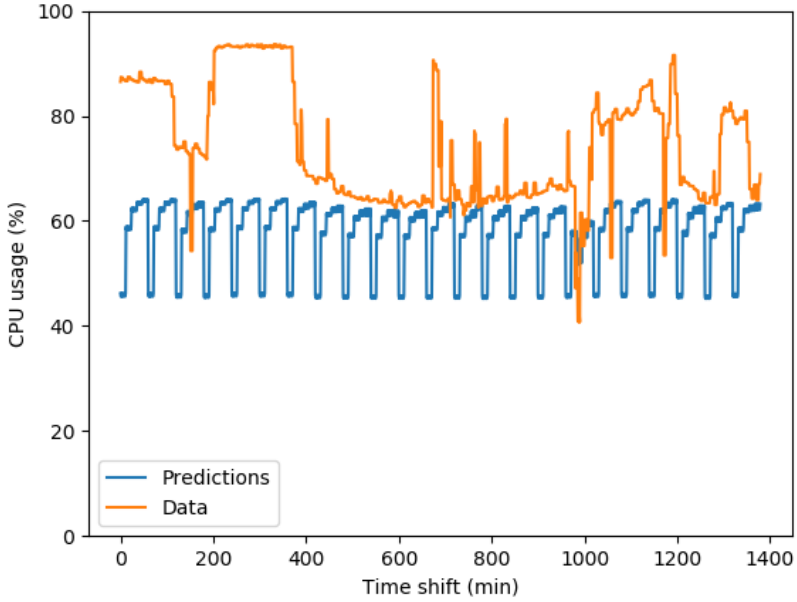


Figure A.5: Machine ID: 3338000908

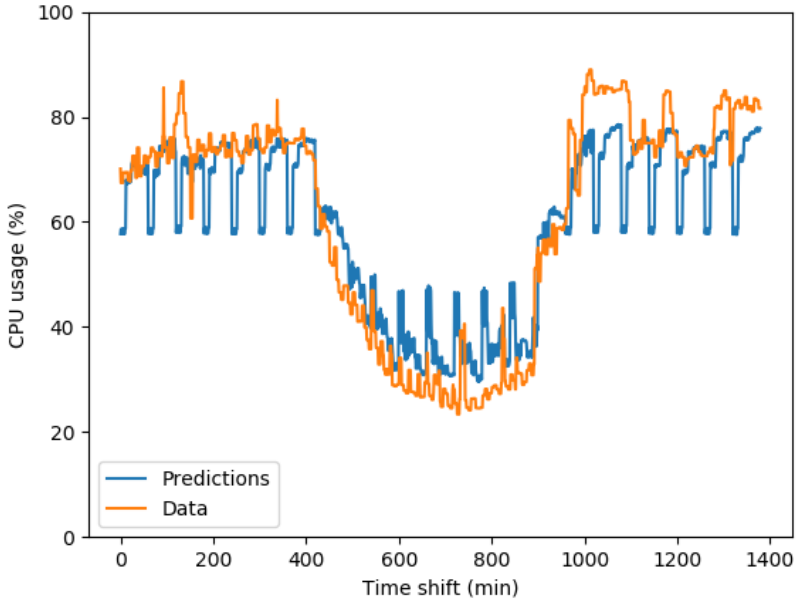


Figure A.6: Machine ID: 1390835522

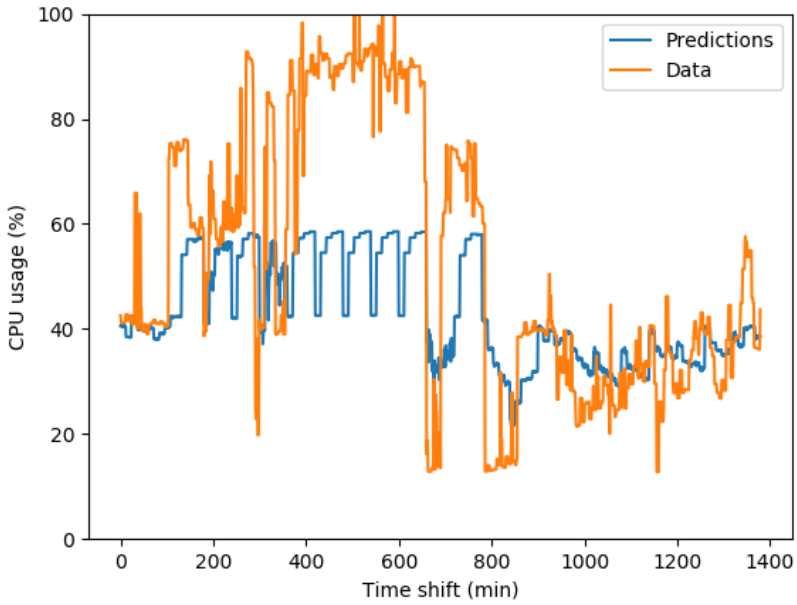


Figure A.7: Machine ID: 1391018274

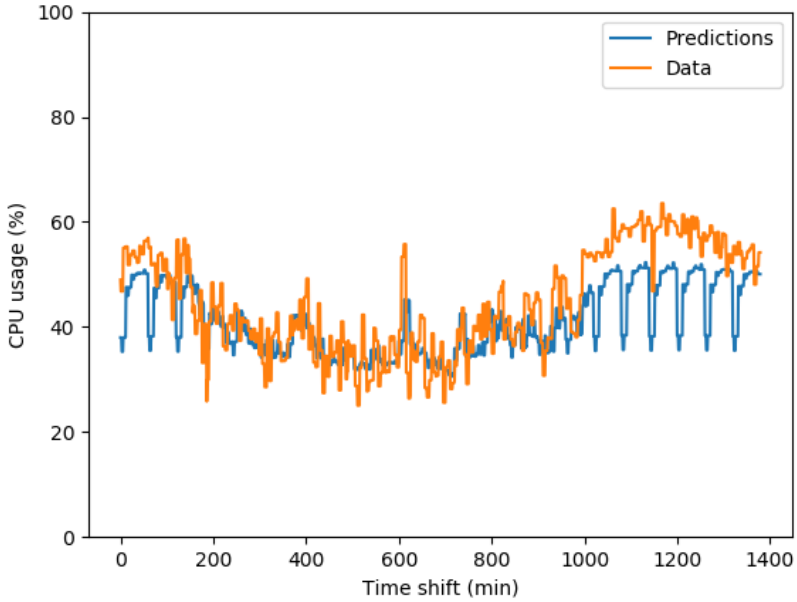


Figure A.8: Machine ID: 5015788232

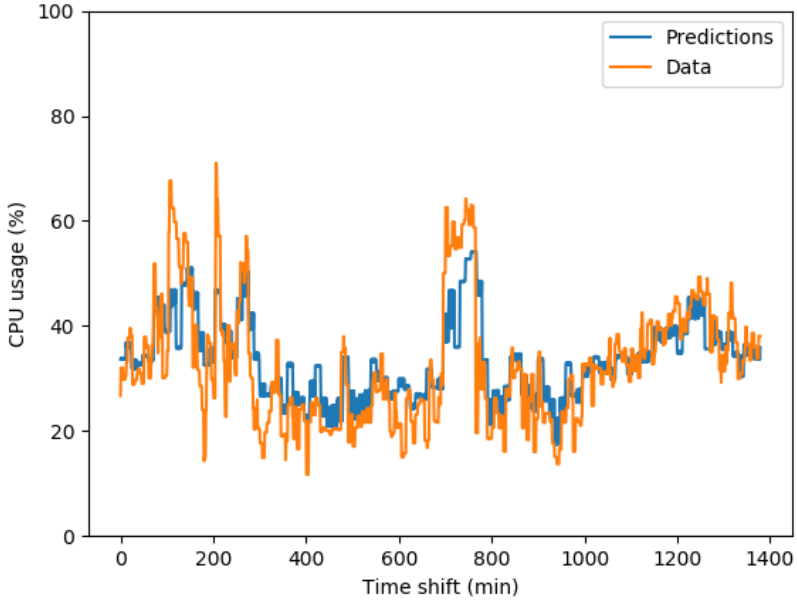


Figure A.9: Machine ID: 4878102959