

Aalto University  
School of Science  
Master's Programme in Information and Communications Technology - Innovation

Mohammad Omar Nasir

# **Supervised Learning in Lighting Control Systems**

## **Using Deep Learning for Predictive Modelling**

Master's Thesis  
Espoo, September 03, 2018

Supervisors: Professor Alexander Helmut Jung, Aalto University  
Advisor: Juslen Henri, D.Sc. (Tech.)  
Laura Sepponen, D.Sc. (Tech.)

Aalto University  
 School of Science

 Master's Programme in Information and Communications  
 Technology - Innovation

 ABSTRACT OF  
 MASTER'S THESIS

<b>Author:</b>	Mohammad Omar Nasir		
<b>Title:</b>	Supervised Learning in Lighting Control Systems Using Deep Learning for Predictive Modelling		
<b>Date:</b>	September 03, 2018	<b>Pages:</b>	80
<b>Major:</b>	Digital Media Technology	<b>Code:</b>	SCI3023
<b>Supervisors:</b>	Professor Alexander Helmut Jung		
<b>Advisor:</b>	Juslen Henri, D.Sc. (Tech.) Laura Sepponen, D.Sc. (Tech.)		
<p>The objective of the thesis is to develop Predictive Models for Lighting Control Systems. Lighting Systems typically employ various sensors to automatically control installed Luminaires. An example of such a sensor is the Passive Infrared (PIR), that detects human motion and subsequently changes the state of Luminaires. These sensors have pre-defined delay timer values which control the amount of activity in a lighting system. Luminaires are generally forced to stay on for a long period of time, to ensure that lights are not turned off when a room is occupied. However, using long delay timers also leads to excessive energy consumption. By developing predictive models, the system can anticipate human presence in advance, and tune light parameters to achieve the optimal balance. Using deep learning, it is shown that by framing the historical data of sensor outputs as time-series forecasting problem, it is possible to predict the output of a PIR sensor in advance, and use that information to develop lighting systems that can achieve higher energy savings than conventional solutions.</p>			
<b>Keywords:</b>	Machine Learning, Lighting Control, Predictive Modelling, Neural Networks, LSTMs		
<b>Language:</b>	English		

# Acknowledgements

I would like to extend my sincere gratitude to Henri Juslen for his overall guidance and valuable insights on Lighting control systems. I would also like to graciously acknowledge Laura Sepponen for her wholehearted mentorship throughout this study. I would like to thank Javad Nouri for his guidance on Machine Learning related challenges, and Abdullah Ibrahim for helping understand existing software infrastructure. I also extend my gratitude to all the wonderful colleagues at Helvar for their continued support, and for providing a fantastic work atmosphere. Finally, my deepest gratitude goes to my family and friends for their continuous love and support.

This research is dedicated to my late father, Dr. Nasir Mehmood.

# Contents

<b>1</b>	<b>Introduction</b>	<b>7</b>
1.1	Overview of Lighting Systems . . . . .	7
1.2	Lighting Control Techniques . . . . .	8
1.3	Benefits of Proactive Systems . . . . .	8
1.4	Outline of the Thesis . . . . .	9
<b>2</b>	<b>Background</b>	<b>11</b>
2.1	Drawbacks of Lighting Control Systems . . . . .	11
2.2	Predictive Models . . . . .	12
2.3	Machine Learning for Predictive Modelling . . . . .	13
2.3.1	Supervised Learning . . . . .	13
2.3.2	Unsupervised Learning . . . . .	14
2.3.3	Related Works . . . . .	14
<b>3</b>	<b>Data Analysis</b>	<b>16</b>
3.1	Quantitative Analysis for Time-Series . . . . .	16
3.2	Components of a Time-Series . . . . .	17
3.2.1	Principles of Decomposition . . . . .	18
3.2.2	Representing Data as Time-Series . . . . .	18
3.2.3	Decomposition Analysis . . . . .	19
3.3	Forecasting Framework . . . . .	21
3.3.1	Look Back Analysis . . . . .	22
3.3.1.1	Autocorrelation Function . . . . .	22
3.3.1.2	Partial Autocorrelation Function . . . . .	23
3.3.2	Feature Evaluation . . . . .	25
3.4	Supervised Learning for Time-Series . . . . .	26
3.4.1	Training and Validation Datasets . . . . .	27
3.4.2	Cross-Validation in Time-Series . . . . .	27
3.4.3	Uniformly Sampling Observations in Multiple Time-Series . . . . .	28
3.5	Class Sparsity . . . . .	29

3.5.1	Downsampling . . . . .	30
3.5.2	Frequency Distribution . . . . .	30
<b>4</b>	<b>Methodology</b>	<b>32</b>
4.1	Feed-Forward Network . . . . .	32
4.2	Training a Neural Network . . . . .	33
4.3	Optimization . . . . .	35
4.3.1	Activation Function . . . . .	35
4.3.2	Loss Function . . . . .	35
4.4	Recurrent Neural Network . . . . .	36
4.4.1	Vanilla RNNs . . . . .	37
4.4.2	LSTMs . . . . .	38
4.4.3	Bidirectional LSTMs . . . . .	39
4.4.4	Stateless vs Stateful LSTMs . . . . .	40
4.5	Statistical Evaluation Metrics . . . . .	40
4.5.1	Confusion Matrix . . . . .	41
4.5.2	Precision/Recall and F1-Score . . . . .	41
4.5.3	Matthews Correlation Coefficient . . . . .	42
4.6	Empirical Evaluation Metrics . . . . .	42
4.6.1	Energy Consumption . . . . .	42
4.7	Lighting Control . . . . .	43
<b>5</b>	<b>Implementation</b>	<b>44</b>
5.1	Data Pre-processing . . . . .	44
5.1.1	Sliding Window . . . . .	44
5.1.2	Walk-Forward Validation . . . . .	45
5.2	Network Design . . . . .	46
5.2.1	Feed Forward Network . . . . .	46
5.2.2	Vanilla LSTM . . . . .	46
5.2.3	Bidirectional LSTM . . . . .	48
5.3	Batch Training in LSTM . . . . .	49
5.4	Problem Definition . . . . .	50
5.5	Flow Chart . . . . .	51
5.6	Algorithm . . . . .	52
<b>6</b>	<b>Evaluation</b>	<b>55</b>
6.1	Network Dimensioning . . . . .	55
6.1.1	Feed Forward Network . . . . .	55
6.1.2	Vanilla LSTM . . . . .	57
6.2	Evaluation using Statistical Metrics . . . . .	59
6.3	Model Tuning . . . . .	60

6.3.1	Learning Rate, Batch Size and Dropout Regularization	60
6.3.2	Bi-directional LSTM . . . . .	62
6.3.3	State Management in LSTMs . . . . .	63
6.3.3.1	Stateless LSTMs . . . . .	63
6.3.3.2	Resetting States Manually . . . . .	64
6.4	Lookback Feature Selection . . . . .	65
6.5	Estimating Model Skill . . . . .	66
6.6	Controlling Luminaires . . . . .	68
6.7	Performance on Test Set . . . . .	70
<b>7</b>	<b>Discussion</b>	<b>72</b>
<b>8</b>	<b>Conclusions</b>	<b>74</b>
<b>A</b>	<b>Hyperparameter Tuning</b>	<b>78</b>

# Chapter 1

## Introduction

A Lighting system refers to the intentional deployment of lights for purposes of illumination. Lighting has existed since the inception of time itself, manifested through celestial bodies such as the Sun and the Moon. In its earliest form, the concept of artificial lighting originated when human ancestors burnt wood, leaves and other flora to create illumination. Over time, sophisticated methods such as burning of fossil fuels became representative of evolutionary progress. Vocational advancements coupled with societal progress gradually placed greater emphasis on indoor lighting, framed as a direct consequence of successive Industrial revolutions. The importance of lighting was further solidified as people increasingly began to work in office buildings. It was not until electricity achieved mass generation that lighting was eventually designated a fundamental necessity.

### 1.1 Overview of Lighting Systems

Today, lighting is ubiquitous. Such is the pervasiveness of modern lighting systems that it is required in every aspect of the modern life, be it inside a class room, workplace, or outdoors for construction projects. With the advent of IoT (Internet of Things) architecture and Big Data revolution, lighting systems have transitioned from simpler isolated units to incredibly complex network of lights and sensors, which often scale up to hundreds and thousands in numbers in a single installation. The industry itself has exhibited exponential growth curve in recent years [3], primarily due to expansive urbanization and a greater demand for energy efficiency. At its core, a typical Lighting control system has an array of devices that provide luminance, known as 'Luminaires'. The system engages various presence sensors that determine whether the room is occupied or not, and subsequently control

Luminaires. This is opposed to accepting input from switch based controls panels in more traditional variants [12, 19]. Advanced Lighting systems with networking capabilities harness wired lighting protocols such as DALI, or wireless technologies like ZigBee to develop messaging and control protocols [4]. Data collected from all connected devices in a lighting network is uploaded to a central server, which is used to provide meaningful insights such as device usage patterns, and fault maintenance.

## 1.2 Lighting Control Techniques

Early lighting control systems incorporated switch based systems, which required laborious manual intervention. Furthermore, these systems relied on responsible usage which would inadvertently lead to wasteful energy expenditure. Technological improvements led to the development of smart systems, which utilized complementary sensors to minimize human interaction. Examples include motion detection sensors that detect movement to control associated luminaires, thus reducing energy expenditure while continuing to ensure light is available during room occupancy [2, 7, 12].

Bakker *et al.* [7] categorized lighting in intelligence by differentiating between Reactive, Anticipatory and Proactive control systems. Reactive controlling is the most dominant technique in modern lighting systems, defined as either a switch-based control unit, or a presence sensor which is used to control lighting. Anticipatory systems use prior knowledge such as meeting schedules, which are retrieved from integrated office systems, or calendar events of an individual user. Prior knowledge allows a smart lighting system to preemptively control the luminaire. Proactive systems attempt to control lighting by predicting the expectation of occupancy. Most of the literature reviewed by Bakker *et al.* [7] consisted of reactive systems, with the exception of a single study on anticipatory systems. This establishes a promising research area for developing proactive systems in the context of lighting control.

## 1.3 Benefits of Proactive Systems

The motivation to develop smarter and efficient systems stems from the increasing demand for energy efficiency. The source of majority of the produced electricity around the world comes from non-renewable energy sources. Reactive lighting control systems have achieved higher energy savings as compared to switch-based lighting systems. Similarly, a system that is able to deter-



mine the expectation of occupancy in the future can execute lighting control commands. If this system can accurately predict when the room is going to be occupied, it is also possible to predict when the same room becomes unoccupied. The latter introduces a new set of possibilities for more efficient lighting control. For example, predicting non-occupancy can be used as a measure to determine the duration of occupancy. As long as the proactive system continues to predict occupancy, the lighting control system can ensure the availability of sufficient light levels. Once the output changes to non-occupancy predictions, the lighting control system can initiate changes in light levels. The advantage is instead of using fixed rule-based systems that switch light levels based on pre-configured timers, a proactive system can dynamically adapt to various occupancy situations which can result in greater efficiency and higher energy savings.

Occupancy situations can be repetitive in nature, for example fixed working routines of an individual, or be highly irregular, such as a celebratory event at the office. Both situations necessitate different lighting control strategies, in that the former requires the desk luminaire to consistently provide light until the individual leaves their desk, where as for the latter employees are generally clustered in one room, and a proactive system can quickly turn off lights in non-occupied areas. For the individual case, the system will continue to predict occupancy and ensure light remains available. For the latter, if the system were to use fixed timers, the luminaires will continue to remain on, all over the office, until the timer expires. On the contrary, the proactive system will observe a sudden drop in occupancy data from the sensors, predict non-occupancy, and ensure the system dims down light levels quickly. The objective of this research, therefore, is to explore the possibility of developing a proactive system using Machine Learning, which predicts the expectation of occupancy. Moreover, by finding the optimal model the predictions can be used to simulate light levels in order to evaluate the influence on energy savings, and determine if tangible benefits are achievable with this approach.

## 1.4 Outline of the Thesis

This thesis is divided into 7 chapters. Chapter 1 has provided an overview of lighting control systems, with emphasis on methodologies and techniques prevalent in modern systems. These include the DALI protocol, utilizing the IoT platforms, and characterization of intelligence within lighting systems. In Chapter 2 the drawbacks of the current PIR motion sensing technology are identified. It further establishes the need for improvements in exist-

ing systems through the use of Predictive Modelling and Machine Learning. Chapter 3 introduces the concept of quantitative time-series analysis and forecasting as an application of predictive modelling. The goal is to represent existing PIR data as a time-series to be able to draw meaningful insights. Using principles of the forecasting framework, the time-series is analyzed for feature engineering.

After analyzing the input for the Machine Learning model, Chapter 4 outlines the methodological choices involved in the choice of the model and their variants. It discusses the underlying algorithms of the chosen models, along with strategies to evaluate their predictive performance as a classification problem as well as in the context of lighting control. In Chapter 5, the implementation process from building the data pipeline to model training and evaluation is discussed. It describes the structure of the neural network designed for the problem at hand along with a formal definition of the Machine Learning problem, including various techniques for model tuning. Algorithms for training the customized model and how the predictions will be utilized for lighting control are presented.

Chapter 6 presents the findings of the study. It first establishes the choice of the network parameters based on empirical results. Subsequently, the chosen model is optimized using various strategies as discussed in Chapter 4 & 5. After determining optimal parameter values, model skill is computed using time-series cross validation. The performance of the lighting control algorithm, along with the energy savings, is evaluated on both the validation sets and the test set. The results are finally summarized by reporting the predictive performance and energy savings on the test set. Lastly, in the concluding Chapters 7 & 8, the findings from the thesis are summarized augmented with key insights.

## Chapter 2

# Background

The most commonly deployed sensor for lighting control systems is a PIR device. A PIR sensor detects presence by measuring changes in the heat signature of the target object. This produces a variation in the output voltage of the sensor, which is then digitally interpreted to generate a representative binary state for room occupancy [12]. For example, the reading of a PIR sensor will transition from room to body temperature if an occupant is present. Using absolute values is therefore not a reliable measure, as it is difficult to distinguish between static and mobile heat sources. Since the objective is to control Luminaires, static signatures produced by indoor heating devices can continuously trigger a PIR sensor, and consequently force the Luminaire to stay turned on. Therefore, to achieve efficient lighting control, it is more reasonable to use motion as an indicator of human presence.

### 2.1 Drawbacks of Lighting Control Systems

Lighting control systems augment the output of a PIR by applying delay timers. A Delay timer is designed to act as a safety net; i.e. to ensure that sudden movements such as a person exiting or re-entering a room does not needlessly trigger state transitions for lighting devices. However, the mechanism is not without its caveats. If the value of the delay timer is smaller than the duration for which the occupant was motionless, the PIR will incorrectly presume non-occupancy and force the Luminaire to switch off prematurely. This results in poor user experience, and the occupant has to simulate motion to turn the lights back on. A possible solution is to set the delay timers to inordinate values. This has the unintended consequence of increased energy consumption, where the luminaire does not turn off after the occupant has left the room. Therefore, traditional lighting control strategies

induce a clear trade-off between energy consumption and user experience [12].

## 2.2 Predictive Models

To mitigate the consequences of the trade-off between energy consumption and user experience, a system that learns and develops insights on the usage patterns of a sensor can effectively control both the delay timers and luminance levels in real-time, without necessitating hardware changes at a more fundamental level. Before discussing potential solutions, it is important to realize that type of indoor environment in which the PIR is installed is closely related to the observed movement patterns. Sensors installed in a meeting room tend to produce sporadic trends, whereas corridor sensors observe human flow consistently. Therefore, inclusion of prior knowledge such as meeting schedules and calendar events is limited in its utility, as an anticipatory system would be unable to generalize output predictions for various PIR installation environments. A proactive system trained on the collection of different PIR sensors would theoretically be able to learn and distinguish between individual patterns.

Predictive Modelling is defined as **a branch of data analytics that analyses the relationship between past events to predict the most likely outcome of the future**. Such modelling techniques attempt to discern historical patterns to produce an expectation of the future. It differs from Descriptive modelling, in that the latter is used to analyze and describe the trends that exist within the available dataset. The input to a predictive model is a representation of past events in a machine-readable format, whereas the output consists of forecast of events that are subsequently utilized for effective decision making [8].

These predicted values can be utilized to improve the existing lighting control system in the following ways:

1. The predictions can indicate periods of both activity and inactivity in the future, allowing the system to dynamically adjust time delay values to minimize energy consumption.
2. Recognize false-events, i.e. situations where small time delay values incorrectly determine inoccupancy.

## 2.3 Machine Learning for Predictive Modelling

From Tom M. Mitchell's definition of Machine Learning:

**“Machine Learning is a scientific field aiming to build computer systems that automatically improve with experience. A machine learns with respect to a particular task  $T$ , performance metric  $P$ , and type of experience  $E$ , if the system reliably improves its performance  $P$  at task  $T$ , following experience  $E$ .”** [18]

Machine Learning is an intersection between Computer Science and the field of Statistics. It aims to develop algorithms based on statistical models that determine and describe relationships between variables in the dataset. A Machine Learning model is trained iteratively on a processed dataset with appropriately constructed features. The objective of the training phase is to learn the relationship between a set of inputs and outputs. The inputs are usually referred to as independent variables or features, and the output is known as the dependent or response variable. The algorithm penalizes inaccuracies in the model, thus forcing it to improve accuracy with each iteration.

Two common tasks in Machine Learning are Classification and Regression. Classification is defined as a methodology to categorize classes in the dataset by qualifying their relationships. For example, a dataset that contains pictures of different animals can be modelled to understand their differences. The descriptive power of the machine learning model would therefore be evaluated on the identification accuracy for any unknown image. Regression is defined as a technique to quantify the relationship between dependent and independent variables [15]. A regression model is considered accurate if it can determine the true value of the response variable given new inputs. An example of regression is weather forecasting based on measurements such as temperature, pressure, etc.

To develop the modelling techniques described above, two fundamental approaches are explained below:

### 2.3.1 Supervised Learning

In Supervised learning, the Machine Learning model is explicitly given instructions about the expected output and available inputs. A set of input and output pairs are created in which the output is pre-labelled with a value. For example, in the animal classification problem, each image is marked with the name of the animal. The training process would then entail producing the correct label given a set of input images.

### 2.3.2 Unsupervised Learning

Unsupervised learning is performed by feeding input data to the machine learning model without explicitly defining the response variable. The goal of the model is to determine possible structures in the dataset [9]. An example is a model that differentiates between different customers based on their spending habits. It is not pertinent to classify customers into categories, rather the classification methodology is the objective of the training process. Unsupervised learning is inherently challenging, but necessitates less manual work in data processing.

Based on the above discussion, Machine learning for Predictive Modelling in the context of lighting control is best described as a Classification problem. The response variable is the binary occupancy state of the sensor. Conversely, the input variables are observations of the past states of the sensor. In addition, the model has to be explicitly supervised about the relationship it needs to determine between the observations and response variable. *Therefore, the Machine Learning algorithm should be trained on past trends that encapsulate Experience  $E$ , given Performance Metric  $P$  which represents how well the model predicts the future for the given Classification Task  $T$ .*

The objective of this study can now be summarized as the implementation of predictive modelling algorithms, which determine probabilities for the events in the future. It is formally defined as:

**What is the probability that an event will occur, for each of the next  $N$  time steps?**

### 2.3.3 Related Works

There has been a growing interest in Building Automation (BA) systems in recent years. Traditional BA systems are generally reactive in nature. An example is the Heating and Ventilation Air Control (HVAC) subsystem, which regulates airflow and indoor temperature to maintain user comfort. It employs sensors that measure CO<sub>2</sub> levels and temperature values that are further used to identify degrading environmental conditions. These act as triggers for parametric changes in HVAC systems to achieve optimum comfort levels.

Lighting control is another component of a BA system. Both HVAC and Lighting control systems are analogous in terms of the defined objective and expected benefits. Using reactive techniques to reduce energy consumption and increase user comfort have proven to be effective and reliable. Therefore, interest in researching methods to improve current system efficiency has been steadily increasing. Techniques that predict occupancy in advance are

beneficial in regulating air flow and temperature levels, as there is a direct relationship between the number of occupants and CO<sub>2</sub> levels. It is useful to review occupancy prediction research, as the underlying concept is equally applicable to lighting control systems.

Ryu et. al. [21] adopted a two-step approach in developing a prediction model for HVAC systems. The first step involved constructing a decision tree to accurately determine the occupancy at current time step, using input from a variety of sensors including temperature, humidity, CO<sub>2</sub>, etc. This information is subsequently used in a Hidden Markov Model to successfully predict occupancy in the future. Similarly, Adamopoulou et. al. [1] also propose a context-aware predictive modelling system, which builds an occupancy model based on historical data, and utilizes the model to engage in short-term and long-term forecasts.

Qolomany et. Al. [20] compared LSTMs and ARIMA models on time-series data sampled from Wi-Fi access points in a commercial building. The Wi-Fi data was used to identify the number of people in a room based on their device addresses. This was pre-processed appropriately to serve as input to the LSTM network. The research cites significant improvements over linear ARIMA models in terms of forecasting accuracy, and serves as a basis for the work reported in this study.

Artificial Neural networks are increasing in popularity, and have proven to be universal function approximators. They have also been used in time-series prediction problems. Khashei et. al. [16] proposed an ensemble of neural networks and ARIMA models in improving prediction accuracy. Cron et. al. [6] performed a comparative study of various neural networks used in NN3 Time-series Prediction Competition. Their findings suggest that Neural networks are able to perform competitively with statistical models. Given the nature of the promising results, it is feasible to run a neural network on time-series data without requiring detailed statistical analysis on the data itself.

## Chapter 3

# Data Analysis

Before choosing a Machine Learning algorithm, it is important to understand and describe the dataset. Extracting meaningful information from the dataset, and removing spurious or confounding variables improves the accuracy of the model. It also aids in developing a more holistic understanding of the problem. The objective of this section is to explain quantitative forecasting principles in time-series, and to apply statistical analysis tools to discern significant features which exist in the dataset.

### 3.1 Quantitative Analysis for Time-Series

A Time-Series is a sequence of observations over time. If the amount of observations is sufficient, Quantitative analysis becomes an effective tool that does not require the practitioner to have extensive domain knowledge. It can be used to extrapolate historical observations by dissecting collected information, or to explain the relationship between various aspects of the time-series. The former is convenient as it does not require the practitioner to completely understand the complex relationships that are assumed to exist within the time-series; rather it deals with the quantities holistically.

For any time-series, predicting the continuation of historical patterns in the future allows for improved decision making. Often real-world time-series have stochastic components that are difficult to predict. The stochasticity of real-world phenomenon does not imply that the time-series is entirely random in nature, rather certain aspects of the observations exist that tend to exhibit similar trends. This knowledge can be exploited to develop statistical models, which predict the future state of time-series based on historical data. This is defined as **Forecasting** [17]. Forecasting time-series is a scientific endeavour with well-defined steps given below:



1. Problem Definition
2. Information Collection
3. Data Analysis
4. Model Fitting
5. Model evaluation

Previous sections have dealt with defining the need for forecasting in context of lighting control. The next step is information collection. The data from PIR sensors requires conversion into a representative time-series for feature analysis. Any significant features that exist in the time-series are used to develop statistical models which fit the given data, and are used to forecast the future. Various measures are employed to determine the quality and accuracy of the forecast.

## 3.2 Components of a Time-Series

A time-series can be represented as a combination of trend, seasonality, cyclicity, and residuals patterns [17].

**Trend:** A trend is a general systematic pattern that exists for the duration of the series, and is characterized by a long-term increase or decrease in its value.

**Seasonality:** Seasonal patterns are periodic in nature, and have a constant length.

**Cyclicity:** Cyclic patterns differ from seasonal patterns as they do not have a fixed length. A cyclic pattern is expected to repeat; however, the duration of the pattern can be irregular.

**Residuals:** It is the irreducible error that remains in the time-series once the effects of both trend and seasonality have been removed. This is stochastic in nature and cannot be reliably predicted.

The size of the window represents the assumption of seasonal recurrence. For example, a window of 60 minutes assumes that the time-series exhibits similar pattern every hour. On the other hand, a window size of 1440 minutes will attempt to model the daily seasonality of a PIR sensor. Therefore, variations in window sizes produce different interpretations of the dataset.

### 3.2.1 Principles of Decomposition

Classical decomposition techniques apply linear additive models to break a time-series into its components, defined in (3.1).

$$\mathbf{F}(t) = T_t + S_t + R_t \quad (3.1)$$

The first step is to remove any trend and cyclic components of the series. This is accomplished by taking the moving average of the samples chronologically. By assuming that adjacent samples in the time domain have similar values, moving average smoothers random variations in the series. The averaged series is known as the trend-cycle component. This is then subtracted from the original data to obtain seasonal and residual components [17].

The seasonal component is assumed to have a constant length before extracting it from the de-trended series. The  $i$ th data point in each seasonal window is averaged across all observations. The seasonal component is therefore the concatenation of these averaged values as a recurring sequence. Finally, by subtracting both trend and seasonal components from the actual data, the remainder is the residual of the series.

### 3.2.2 Representing Data as Time-Series

The output of a PIR sensor is to be converted into a time-series. Whenever a sensor detects movement, an event is recorded in the database with its timestamp. Since the PIR can only transition between two states, the observations correspond to the state of occupancy. This is denoted as a Boolean **True** value. Furthermore, the state is considered **False** in the absence of an event. Hence the output of PIR sensor is a sequence of states corresponding to binary values (1 or 0) across the time dimension.

Let  $\mathbf{P}$  be the set of all PIR sensors:

$$\mathbf{P} = \{Sensor1, Sensor2, \dots, Sensor5\} \quad (3.2)$$

The output for a single sensor is then defined as:

$$\mathbf{P}_i = \{e(t) \mid t \in \mathbf{T}\} \quad (3.3)$$

Where  $\mathbf{e}$  is a **True** event that occurs at time  $t$ ;  $\mathbf{T}$  is the Index Set containing all recorded timestamps in chronological order for sensor  $\mathbf{P}_i$ . Figure 3.1 shows the output of an exemplary PIR sensor for 1 day.

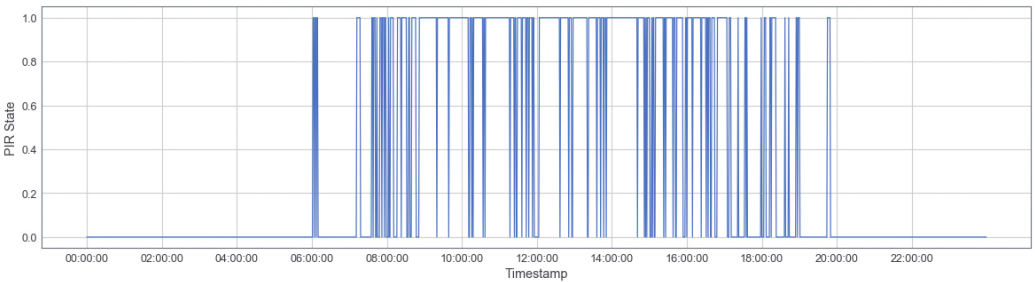


Figure 3.1: PIR sensor output for 1 day.

3.2.3 Decomposition Analysis

Plotting the time-series reveals trends, seasonality or similar features of the data. Figure 3.2 shows trend-decomposition of a PIR sensor over a period of 3 days using a window size of 60 minutes. The moving average plot shows a recurring pattern of employee activity for each day, which corresponds to office working hours.

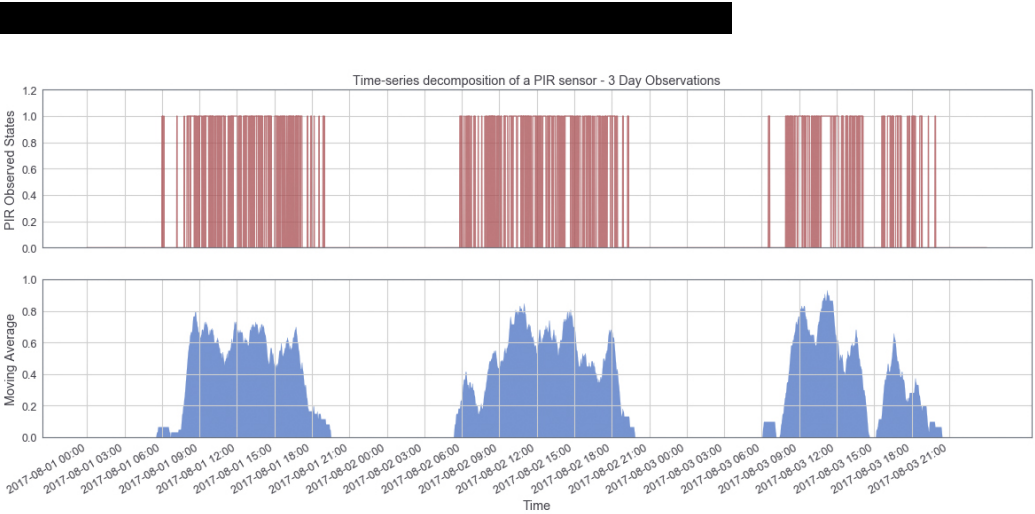


Figure 3.2: Decomposed output of a high activity sensor, using a 60 minute window over a period of 3 days.

Figure 3.3 contains plots for the decomposition of a PIR sensor over a period of 30 days. Each x-axis marker indicates the start of a day. There is minimal observed activity on weekends. This can be characterized as weekend seasonality. Additionally, the PIR output is consistent across weekdays. The only exception is 25<sup>th</sup> Aug, Friday, where the activity patterns are highly sporadic.

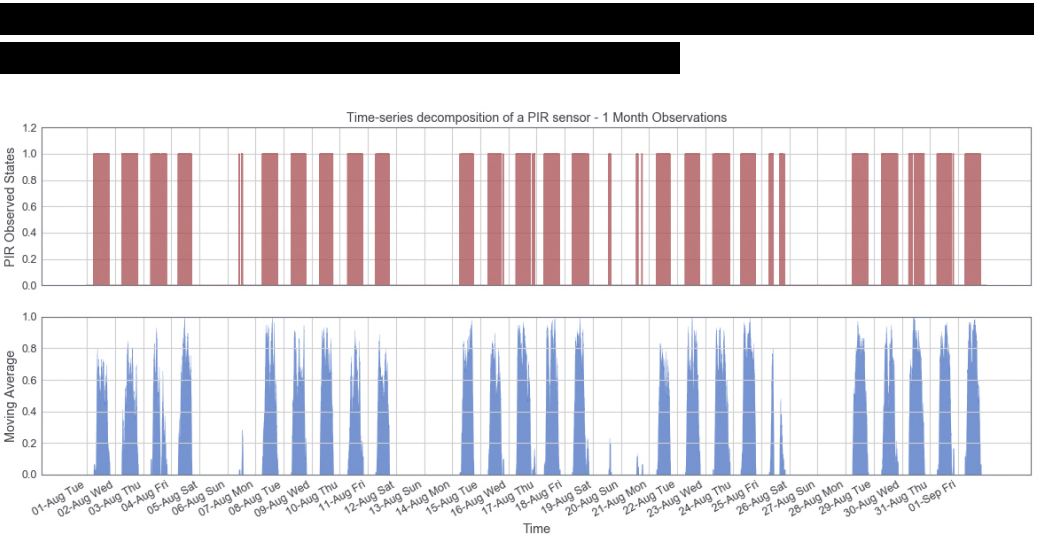


Figure 3.3: Decomposed output of a high activity sensor, using a 60 minute window over a period of 1 month.

Figure 3.4 shows the moving average plot over a period of 1 year. The chosen window size is 1440 minutes, which corresponds to 1 day. Ignoring the missing data from April, the only discernible pattern is the decrease in activity during the month of June, and at the end of December.



Figure 3.4: Decomposed output of a high activity sensor, using a 1440 minute window over a period of 1 year.

To analyse in detail the composition of the PIR output from one day, figure 3.5 shows the trend, seasonal and residual components of the time-series. The moving average plot follows the working hours schedule. The seasonal window size was set to 60, which assumes the patterns recur every 60 minutes. Different values were also experimented with, however, there are

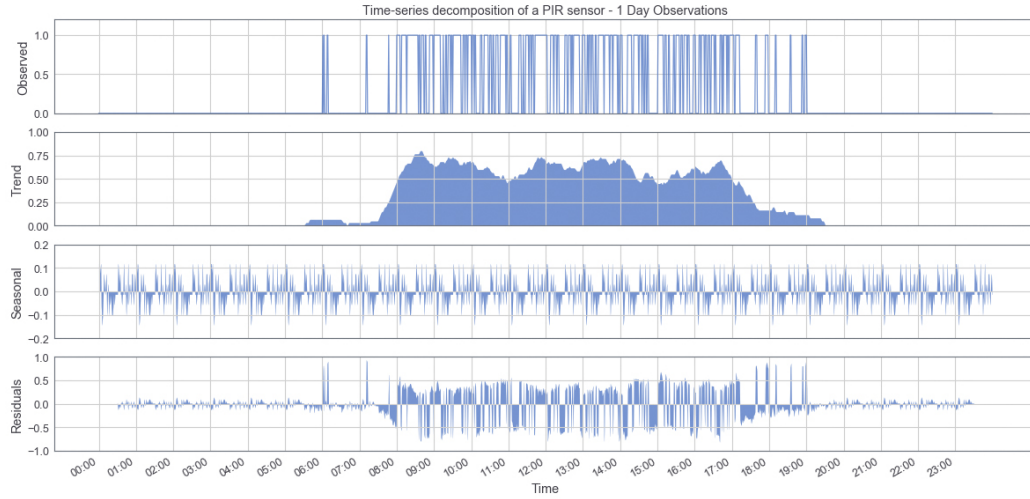


Figure 3.5: Decomposed output of a high activity sensor, using a 60 minute window for 1 day.

always significant residuals in the decomposed series which suggests that the time-series comprises of non-linear components. A larger window size gives a smoother trend, but the complexity of the seasonal component increases to compensate for high residuals.

### 3.3 Forecasting Framework

A central concept in time-series forecasting is the **assumption of continuity**. It establishes the relevance of past observations that are expected to persist in the future [17]. Forecasting is not possible if the time-series is entirely stochastic in nature. This is known as a **random walk**. Random walk time-series are difficult to predict, and the strategy to reproduce current value as a forecast usually results in the best accuracy. For continuous time-series, the extent of influence of historical data on the quality of the forecast is a critical investigative factor. As an example, for a forecasting model with 1 day granularity, the influence of samples from 20 years ago will be less relevant if the forecast is strongly correlated with recent observations.

To develop a forecasting model, various aspects of the forecasting framework are explained below.

1. **Forecasting Origin:** It is the last known sample from the dataset, beyond which the prediction exists.

2. **Forecasting Horizon/Look Ahead:** It is the number of samples that the model is expected to predict into the future.
3. **Look Back:** The window containing required past observations to predict the future.

Larger horizons reduce the quality of the forecast, as the uncertainty, or prediction error, increases [6]. Moreover, a larger look back window can increase computational complexity at the expense of negligible gains in forecast accuracy. Conversely, a smaller look back window encodes less information which increases the forecast variance.

Once the size and position of these components are fixed, an update technique is implemented, which sequentially feeds samples from the time-series to the model. The chosen method is called **Rolling Forecast Origin Recalibration**. A rolling forecast origin implies that the look back window is progressively shifted forward in time by adding a new observation. This updates the observations in forecast horizon as well. Using the updated look back and look ahead windows, the model weights are recalibrated to maintain assumption of continuity. The goal of such an approach is to develop a model that continues to learn and generalize various patterns within the time-series.

Lastly, the method to update look back window and forecasting horizon is called **Fixed-Size Rolling Window**. The update process prunes the oldest observation to ensure the window size remains constant [24].

### 3.3.1 Look Back Analysis

Two time-series analytical tools, Autocorrelation Function (ACF), and the Partial Autocorrelation Function (PACF), will be used to approximate the size of look back window. It is worthwhile to remember that both ACF and PACF only compute linear correlations, whereas real-world datasets can contain non-linear dependencies.

#### 3.3.1.1 Autocorrelation Function

Autocorrelation is defined as the coefficient of correlation between two samples in a time-series. These samples can be separated by several time-steps which defines the **lag order  $k$** . A lag  $k$  autocorrelation between two samples  $x_i$  and  $x_{i+k}$ , is calculated by subtracting the overall mean of the series from each pair of observation separated by  $k$  time-steps. Each difference is multiplied together and then summed over the entire time-series to produce the covariance at lag  $k$  for all samples, normalized by mean squared difference for the whole series.

$$\text{ACF} = \frac{\sum_{i=1}^{N-k} (x_i - \bar{x})(x_{i+k} - \bar{x})}{\sum_{i=1}^N (x_i - \bar{x})^2} \quad (3.4)$$

A function which computes Autocorrelation at different orders of lag is known as ACF. The output of an ACF is a vector of correlation coefficients between pairs of  $x_t$  and  $x_{t-k}$  where  $k$  is the lag order. The coefficient of correlation,  $r$ , at each lag determines persistence, or reversals. A coefficient of 1 indicates that similar values follow in persistent patterns, whereas -1 indicates that opposite values follow in reversal patterns. A value of 0 indicates that there is no correlation between the time samples, and the output is similar to white noise.

### 3.3.1.2 Partial Autocorrelation Function

Partial Autocorrelation is calculated by removing the lagged correlation that can propagate between the samples, to determine their unconditional correlation. Consider the case where the correlation coefficient is significant between two samples of lag order 2,  $x_t$  and  $x_{t-2}$ . A PACF provides insights whether the coefficient is conditional on ACF propagation through the intermediate sample  $x_{t-1}$ , or the two samples are unconditionally correlated. An example is a PIR sensor that detects movement at exactly 08:00 AM every morning. Two samples from consecutive days would have a high correlation coefficient, that can be reliably estimated irrespective of the conditional correlation of the events within the 24-hour period between the data points. In practice, however, it is extremely unlikely to observe the exact pattern every day, owing to several extraneous variables such as employee behaviour, office events, etc.

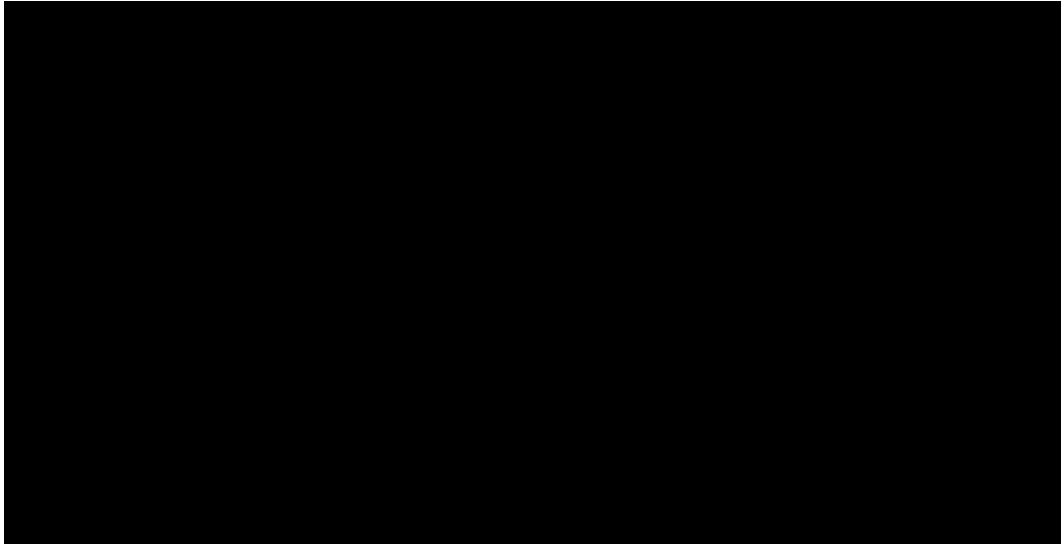


Figure 3.6: Sample ACF and PACF plot for a high activity sensor

Using the Python Library statsmodels [22], a sample ACF and PACF plot for a high activity sensor is shown in figure 3.6.

Each sample in the ACF plot indicates the autocorrelation coefficient, plotted against the 95% significance limits. If a value at **lag k** exceeds the 95% confidence interval, it is considered as a significant coefficient. In other words, the current value of the time series is correlated with the previous value at **lag k**.



To determine the optimal size of the look back window, the maximum significant coefficient should be considered from the ACF. Figure 3.7 shows a histogram of all the significant lag values, computed over the training set for each sensor in 1 day.



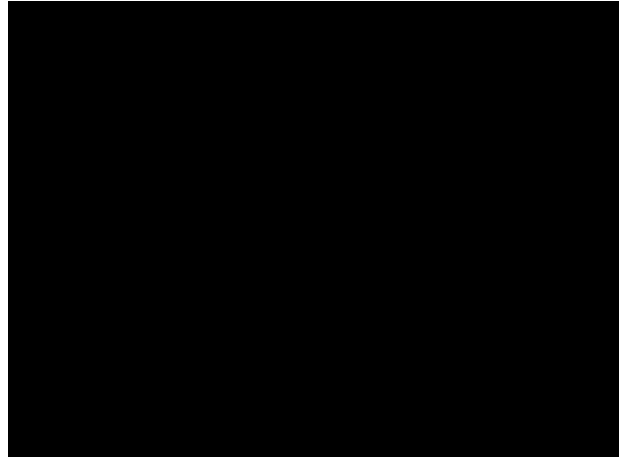


Figure 3.7: Histogram of Significant Lag Orders

A small number of samples lie close to lag 0, which is autocorrelation of a time-series with itself and is always 1. Lag 1 correlation indicates that the PIR state at the previous time step is the only required information to predict the next value. Any attempt to linearly model these sensor outputs would result in a naive forecast, in which the prediction is a replica of the value at the last time step. However, using small lag order limits the amount of information available for the machine learning model to make predictions, whereas an increase in lag order should improve accuracy at the cost of computation time. [REDACTED]

[REDACTED] Larger sequences can provide additional contextual information for forecasting, however, it would require greater training time. These inferences will be empirically determined in the Evaluation section.

### 3.3.2 Feature Evaluation

Using data from the month of August for the selected sensors, a useful method to identify significant features of the dataset is aggregate event count for different granularities. Figure 3.8 shows the boxplot of number of events for each minute of the hour. [REDACTED]



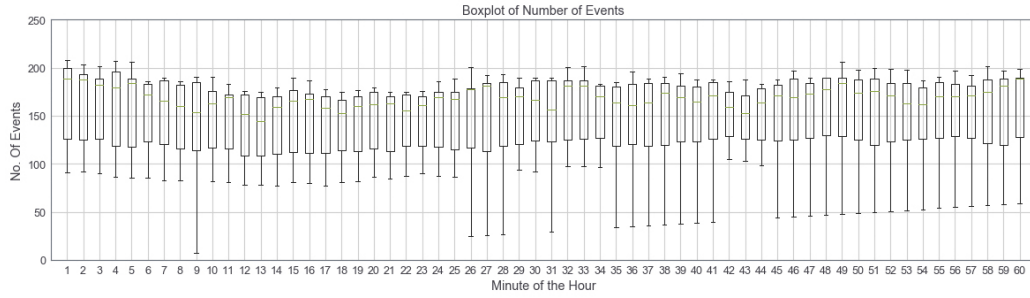


Figure 3.8: Total number of events in 1 hour

Similarly, figure 3.9 visualizes the boxplot of number of events for each hour of the day. This is symptomatic of the office working hours, and follows a general trend. [REDACTED]

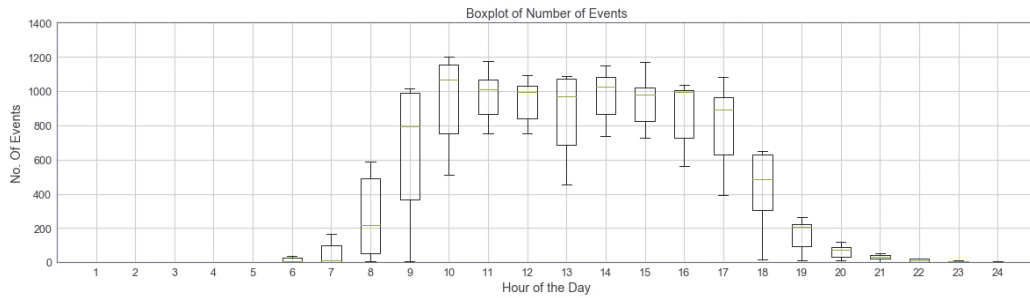


Figure 3.9: Total number of events in 1 Day

### 3.4 Supervised Learning for Time-Series

In Section 2.2, the predictive modelling problem was characterized as supervised learning, which requires a pair of input and output sequences to be fed to the algorithm. The next sections discuss strategies to convert raw data into an algorithm compatible format, as useful input features. The available dataset for this study was collected from an office. It contains the output state of 6 PIR sensors, which are installed in corridors, working areas, and meeting rooms.

### 3.4.1 Training and Validation Datasets

It is the requirement in a Machine Learning problem to split the dataset into 3 parts: Training, Validation and Test data. The model is initially trained to interpret and learn the encoded information in the Training dataset. Because of bias-variance trade-off, the model must retain generalizability on the unseen Validation set. This describes the predictive power of the trained model. A model which exhaustively learns patterns in the training dataset becomes sensitive to trivial changes. The goal is to learn a generic representation of the available information, which can be used to describe similar datasets with high accuracy.

The choice of validation set is important in context of time-series. Generally individual data points are assumed to be independently distributed, and allow the training and validation sets to be constructed using random sampling from the dataset. This violates the assumption of continuity for time-series as validation samples can precede training samples in time. To maintain autocorrelation between samples, time-series datasets need to be chronologically preserved. Consequently, two techniques are employed: a) Out-of-sample forecasting, which ensures that the trained model cannot peek at the validation samples, and b) Last-block validation which fixes the last samples from the dataset for validation [17, 24]. This ensures that forecasting is always performed on future samples.

### 3.4.2 Cross-Validation in Time-Series

A Machine learning model trained on fixed training and validation sets cannot be guaranteed to perform on other datasets with similar accuracy. It is advantageous to train the model multiple times on different subsets of the data, and report the average skill of the model. Generally, practitioners use k-Folds Cross validation approach to construct multiple training and validation subsets. This technique ensures that each data point is part of the validation set at least once. However, it does not guarantee the temporal order of samples and is prone to breaking the assumption of continuity.

Instead of using cross-validation, the approach adopted in this study is to roll through the complete dataset. For example, the dataset is selected to accommodate approximately  $k$  subsets. The first subset is then split into training and validation sets using last-block division. The second training set is constructed by including the first validation set at its beginning, and adding new observations from  $k_2$  subset. The corresponding validation set again forms the last-block of observations from  $k_2$  subset. This process is continued until the complete dataset is divided into chains of training and

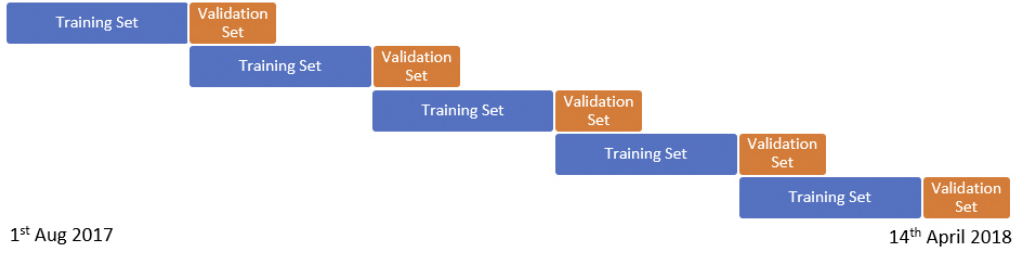


Figure 3.10: Rolling Cross-Validation Technique

validation sets, where the validation set never precedes its corresponding training set.

The available dataset is between 1<sup>st</sup> Aug 2017 and 30<sup>th</sup> April 2018, which excludes missing samples from February. Rolling training and validation subsets of the data are created, with the ratio between each set at **40:10** days. It is visualized in Figure 3.10. This mimics a real-world scenario in which the data becomes sequentially available. The analysis in subsequent sections is performed under the assumption that only the first training data set is available. Moreover, the test dataset consists of 15 days preceded by the last validation set, from 14<sup>th</sup> April to 30<sup>th</sup> April.

### 3.4.3 Uniformly Sampling Observations in Multiple Time-Series

Typically, each PIR sensor has a maximum distance, height and a beam angle that defines its coverage area. The goal of the installation of a sensor is to maximize the intended coverage area. Understandably, the detected patterns should vary depending on the installation and utilization patterns of the target area. For example, a PIR sensor whose objective is to monitor movement in working areas will produce dense waveforms. This pattern will be different to a sensor that is installed in the office kitchen, which typically observes irregular motion throughout the day. Moreover, the coverage area of a sensor is proportional to the observed activity as well. Therefore, different sensors record events at different timestamps.

The objective here is to produce predictions for a forecasting horizon of  $N$  minutes. This assumes that the observations from all the sensors are equally spaced. However, a PIR event trigger entirely depends on behaviour of the occupant. Therefore, it is imperative to define the input of the PIR as a uniformly sampled time-series for consistency.

Re-writing the PIR output eq. (3.3):

$$\mathbf{P} = \{e(t) \mid e \in \{1, 0\} \text{ and } t \in T\} \quad (3.5)$$

Where  $\mathbf{T}$  is the Index Set representing fixed timestamps. A PIR sensor will have a value of either 1 or 0 at timestep  $t$ , that is sampled using the events since the last timestep. If an event occurred between  $t - 1$  and  $t$ ,  $\mathbf{x}_t$  will have a value of 1.

### 3.5 Class Sparsity

The timestamp resolution of the recorded events is per second, which implies that the minimum sampling frequency cannot be less than 1 second. Let the Index Set be defined as the set containing all possible timestamps, sampled at a frequency of 1 Hz, for a specific day. The sampled output of a PIR sensor can be represented as follows:

$$P_{i(s)} = \{e(t) \mid t \in T_s\} \quad (3.6)$$

Where  $T_s = \{00 : 00 : 00, 00 : 00 : 01, 00 : 00 : 02 \dots 23 : 59 : 59\}$

The length of the index set is 86,400, which is the total number of seconds in 1 day.  $\mathbf{P}_{i(s)}$  is True if an event occurred at the corresponding timestamp, otherwise the value is False. To understand the distribution of events in the time-series, figure 3.11 shows the histogram of the count of number of events in a single day for all sensors, computed over the whole dataset.

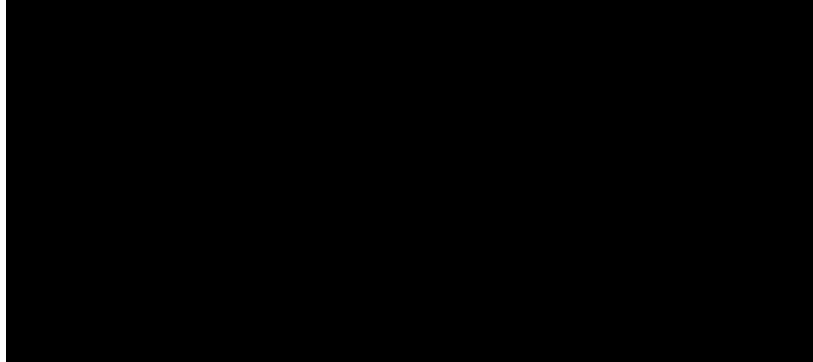


Figure 3.11: Distribution of Number of Events in 1 Day

$$\text{Class Ratio} = \frac{\text{Average Number of Class 1 Samples}}{\text{Total Samples}} = 0.53\%$$

This approach creates an obvious imbalance in the dataset. Since the output of a PIR sensor is binary in nature, data imbalance can be measured in terms of the ratio between the two classes, and is called *Data Sparsity*. Sparsity makes it difficult to learn any meaningful representations of the dataset. Approximately 99.47% percent of the samples for a PIR belong to class 0, and the Machine Learning algorithm will likely learn to predict class 0 for every time-step.

### 3.5.1 Downsampling

From a practical perspective, executing lighting control every second would require a large amount of bandwidth over the network. Moreover, unnecessary switching can potentially reduce the life of a luminaire, which is avoidable by increasing the sampling frequency. Using a value of 1/60 Hz, the system will take a decision about the state of a luminaire every minute. This is achieved by effectively down sampling the data using maximum as the aggregate function. Using an index set with a resolution of 1 minute, the output of a PIR sensor can be described as:

$$P_{i(m)} = \{e(t) \mid t \in T_m\} \quad (3.7)$$

Where  $T_m = \{0, 1, 2 \dots 1440\}$ .

The output of the sensor is True, if there was an event during the last 60 seconds. Consequently, re-computing the histogram and the class ratio gives the following value:

$$\text{Class Ratio} = \frac{\text{Average Number of Class 1 Samples}}{\text{Total Samples}} = \frac{463}{1440} = 32.15\%$$

### 3.5.2 Frequency Distribution

Since the source of the data is an office environment, it is logical to assume that majority of the recorded events occur during the working hours. From figure 3.1, it is evident that majority of the samples in the earlier and later part of a day are False events. To determine if the behaviour of the waveform is consistent across the dataset, the optimal window within which most of the activity for a PIR is computed. Figure 3.12 (a) below shows the aggregated event count of all available sensors. To determine the upper and lower limits

within which 99% data samples are contained, the normalized cumulative frequency distribution is shown in figure 3.12 (b).

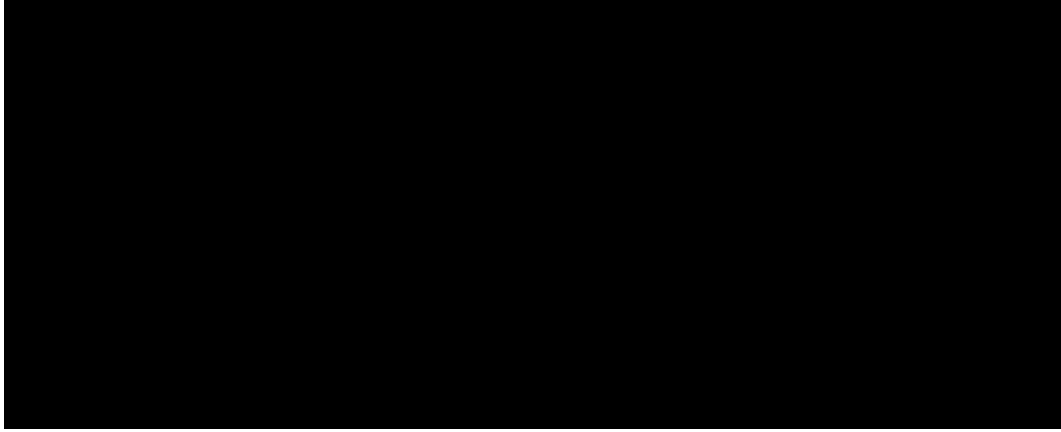


Figure 3.12: (a) Distribution of Events (b) Cumulative Frequency Distribution

The distribution of the event density confirms that the behaviour of the sensors is consistent across the whole dataset, as most class 0 samples occur at the head or tail of the data. These can be removed from the dataset to further improve class ratio. [REDACTED]

[REDACTED] Counting the number of events that occur within this window, the corresponding value of class ratio is calculated as:

$$\text{Class Ratio} = \frac{\text{Average Number of Class 1 Samples}}{\text{Total Samples}} = 52.00\%$$

## Chapter 4

# Methodology

The previous sections gave insights on suitable representations of a sensor output. Once the PIR output has been converted into a time-series, the next step is to build a Machine Learning model that takes a pair of input and output sequences, and learns to predict a sequence given historical data.

## Neural Networks

An Artificial Neural Network (ANN) is an algorithm inspired by the biology of a human brain. An ANN simulates a brain synapse by connecting various neurons together to induce collective learning. Information is fed to an ANN, which flows through the network in a graph based structure. Each neuron and its synapse is responsible for performing mathematical operations on its input, and to subsequently transfer the output to the next neuron. The error from the final output of the network is evaluated and is used to update the state of each neuron to improve network accuracy [9].

### 4.1 Feed-Forward Network

The basic form of an ANN is known as the Feed-forward Network (FF). The operation of this network is restricted in terms of information flow; data can only flow in the forward direction [9]. The structure of a single neuron network is shown in figure 4.1.

At its core, a hidden layer constitutes a neuron, defined in terms of a Weight vector and a bias term. For multi-dimensional inputs, each feature in the sample is multiplied with the corresponding scalar weight value. The outputs are summed up and added to a bias value. This is fed to an activation



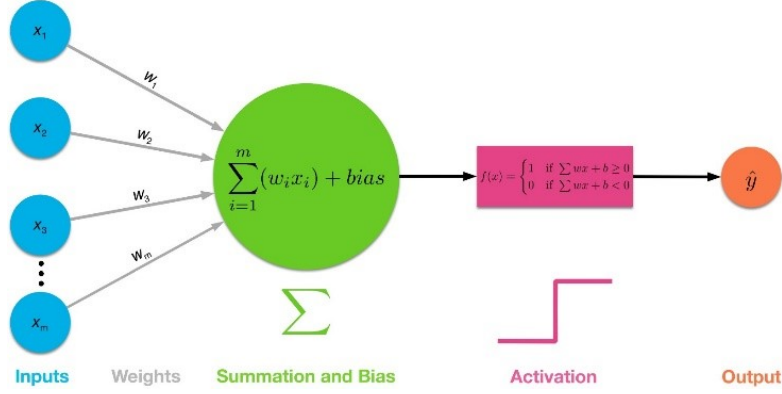


Figure 4.1: Single Neuron Feed Forward Network

function which provides the final output of the network. This operation is known as the forward pass of the network, summarized below:

Let  $\mathbf{x}$ ,  $\mathbf{h}$ , and  $\hat{\mathbf{y}}$  denote the input, hidden, and output layers of the network. The operations of the network are defined as:

$$h(\mathbf{x}) = \mathbf{w}\mathbf{x} + b \quad (4.1)$$

$$\hat{\mathbf{y}} = \text{act}(h(\mathbf{x})) \quad (4.2)$$

The hidden layers of a Feed forward network have connections to each neuron in the input as well as the output layer. This type of hidden layer is called **fully connected** layer. Moreover, a network is defined in terms of its width and depth. The depth of the network corresponds to the number of layers in the network, whereas the width of the network is the number of neurons in each layer.

## 4.2 Training a Neural Network

In training phase, the goal of the ML model is to find the optimal model parameter which produces the least error. An error function of an ML model is defined as:

$$\mathbf{L}(\theta) = L(\hat{\mathbf{y}}, y_{\text{true}}) \quad (4.3)$$

Gradient descent is a common optimization technique for ML models [9]. In real-world applications, the error function can exhibit both convexity and non-convexity, as illustrated in figure 4.2.

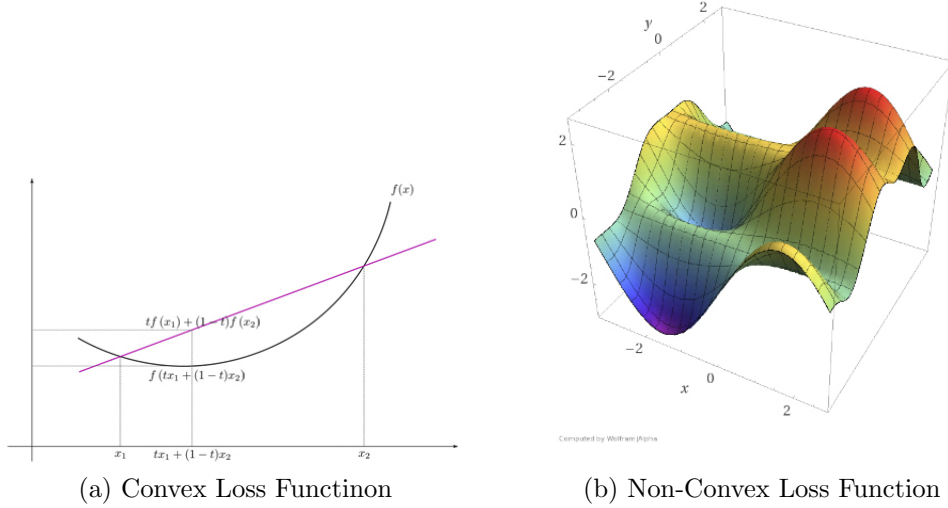


Figure 4.2: Finding global optima in Gradient Descent

$\mathbf{F}(\mathbf{x})$  represents the value of the convex error or loss function in figure 4.2 (a). Similarly, figure 4.2 (b) represents a highly non-convex error function. Since the initial value of the neurons is random, the computed loss can lie anywhere on the graph. Since the global minimum error exists as a valley in the loss function output, the optimization algorithm is responsible for determining the direction to the minima. This is achieved by taking the derivative of the error function. The derivative supplies the gradient and the direction to the minimum error. It is then used to update the weight vector and bias term for each neuron in the network. This technique to update the network variables is known as **backpropagation** [9].

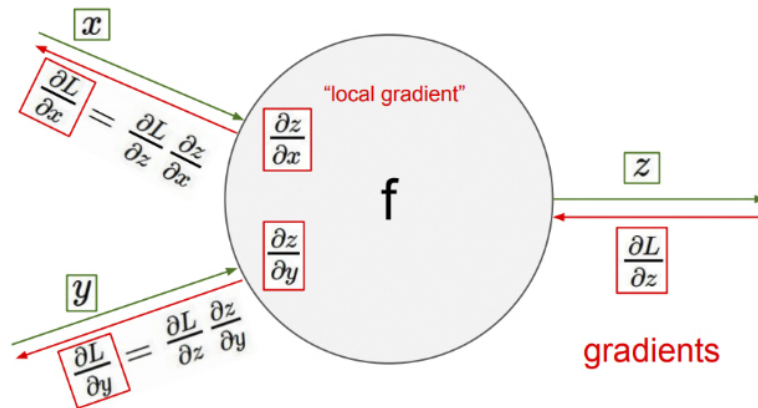


Figure 4.3: Backpropagation in Neural Networks

Neural networks have multiple hidden layers. The accumulated error is a function of all the neurons in hidden layers, which are updated after a single forward pass. The gradient of the loss function is calculated w.r.t. to each trainable variable, using chain-rule differentiation.

## 4.3 Optimization

An implementation of a neural network can considerably vary depending on the choice of optimization techniques. These components of the network influence the quality of the output, and must be determined carefully.

### 4.3.1 Activation Function

The activation function of a neural network applies a transfer function to each neuron in the network. There are two factors to consider when selecting an activation function; the type of problem and assumptions about the nature of the output. For example, a regression model which predicts temperature restricts the final output to the observed limits of the temperature. Similarly, a classification model should predict the probability of the output of belonging to  $N$  classes, in the form of an  $N$  vector of probabilities. Moreover, activation functions are either linear or non-linear, implying the transformation applied to the output of a neuron represents an assumption about linearity or non-linearity within the data. Most real-world datasets are best explained through non-linearities, however, linear functions are simpler and efficient to compute. As discussed in the previous section, the gradient of the loss function is computed through the chain-rule, using the output from each layer. Therefore, the activation function must be differentiable [9].

A common example of a linear function is the straight line:

$$F(x) = x \quad (4.4)$$

Popular non-linear activation functions include the sigmoid and tanh:

$$F(x) = \frac{1}{(1 + e^{-x})} \quad (4.5)$$

$$F(x) = \tanh(x) \quad (4.6)$$

### 4.3.2 Loss Function

After selecting a Machine Learning model, the next step is to choose an appropriate loss function representative of problem definition and objective.

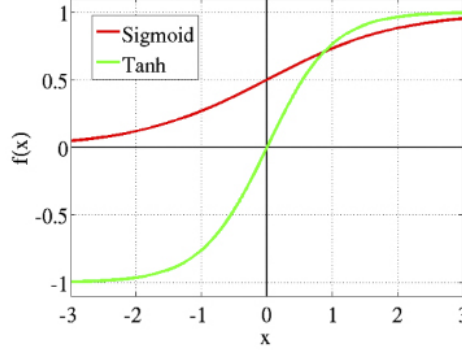


Figure 4.4: Sigmoid and Tanh Outputs

It was established in Section 3.2.2 that the input to the model will be a time-series, and the sequence is binary in nature. Therefore, the predictions from the network will be a sequence of binary values. Given the true values of the PIR for each predicted time-step, the chosen loss function should compare the disparity with the predictions and output an error value. The larger the deviation from true labels, the greater the error. Conversely, the error should be minimum when the network produces accurate predictions. For this purpose, the chosen loss function is **binary cross-entropy** [9]. Binary cross entropy measures the dissimilarity between the predicted probability and the ground truth by determining the amount of extra bits required to encode the predicted labels.

$$L(\theta) = \frac{1}{N} \sum_{n=1}^N H(p_n, q_n) = -\frac{1}{N} \sum_{n=1}^N \left[ y_n \log \hat{y}_n + (1 - y_n) \log(1 - \hat{y}_n) \right] \quad (4.7)$$

Here  $p$  and  $q$  denote a vector of probabilities and ground truth at each time step. To calculate the cross entropy for a sequence containing  $N$  samples, the dissimilarity between each sample and its corresponding ground truth is first summed, and later averaged to produce a single value. This is the total loss of the network.

## 4.4 Recurrent Neural Network

A neuron in a Feed-forward network is associated with a fixed length weight vector, since the input has a pre-determined size. In context of sequences that have interdependent samples, the input is truncated to a specific value

and then fed to the network. The FF network will minimize the loss function based on the input data, however, it cannot determine long-term dependencies in the network. For example, the network is updated after every  $x_t$ , but it is theoretically possible to achieve lower loss if the network acquires memory that retains knowledge of  $x_{t_1}$ , under the assumption that  $x_t$  and  $x_{t_1}$  are strongly dependent. One solution is to feed  $x_t$  and  $x_{t-1}$  together, but this scales the size of the input layer unnecessarily as the dependence assumption grows larger.

#### 4.4.1 Vanilla RNNs

A more robust approach is to use Recurrent Neural Networks (RNNs). An RNN treats the input data as a connected sequence consisting of conditionally dependent samples [13]. It computes the output based on temporal inputs,  $x_t, x_{t-1} \dots x_{t-n}$ , shown in Figure 4.5. Here  $n$  specifies the amount of past information maintained by the network, and indicates assumption of correlation between successive data samples. The forward pass of an RNN is similar to FF, with one critical difference. It combines the output of the hidden layer  $h_{t-1}$  when computing the hidden layer output,  $h_t$ . These outputs are called **states**, illustrated with the blue connection.

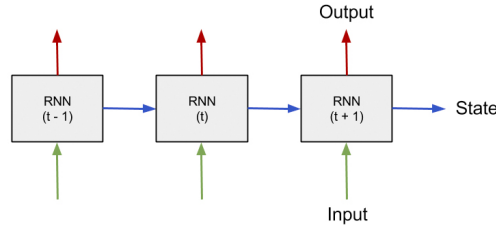


Figure 4.5: Vanilla RNN Architecture

Recurrent neural networks perform back-propagation after unrolling the network for the specified sequence length,  $n$  [25]. Figure 4.5 shows an unrolled RNN as an extended FF network. By treating each state of an RNN as a connected FF network, it is possible to apply backpropagation across the whole input sequence. The parameter  $n$  specifies the extent to which the gradient computations are performed over the past inputs. If the total length of the input data is  $N$ ,  $n$  is always smaller than  $N$ . However, using  $n=N$  requires massive computational resources to propagate the error to the initial data sample. A more reasonable approach is to propagate the error to a small value of  $n$ , which improves computational performance at the expense

of memory capacity, as the network is unable to learn very long-term dependencies. This is known as **truncated backpropagation through time** or **TBPTT** [26].

RNNs outperform FFNs when dealing with time-series or sequential data [13]. However, computing gradient updates is essentially a multiplicative operation, and long input sequences result in the gradient reducing to a miniscule value, or exploding to very large values. RNN outputs lead to exploding and vanishing gradients because of the requirement of the network to assign equal priority to each sample [14].

#### 4.4.2 LSTMs

The introduction of **LSTM (Long Short-Term Memory)** networks solved the exploding and vanishing gradient problem. By selectively utilizing or discarding certain input samples, an LSTM ensures gradient updates remain feasible. In RNNs, the size of TBTT controls the theoretical limit on the memory of the network, and realistic gradient updates. This induces a trade-off where smaller sequences avoid exploding or vanishing gradients, at the expense of long term memory. LSTMs are therefore able to learn much larger sequences without accumulating unreasonable values of gradients [14].

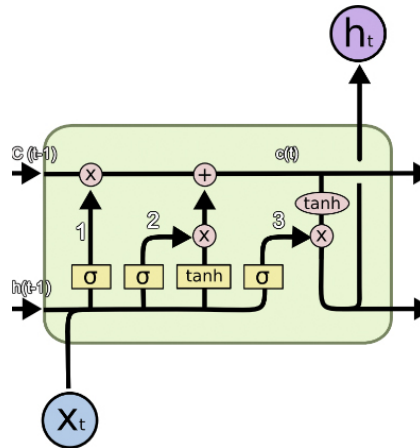


Figure 4.6: LSTM Gates

The structure of an LSTM is shown in figure 4.6. An LSTM cell implements 3 gates, forget, output and input [14]. It maintains two separate states, known as cell state and hidden state. The hidden state is the final output of the LSTM cell whereas the cell state is combined with various outputs from each gate at every time step.

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \quad (4.8)$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \quad (4.9)$$

$$c_t = f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \quad (4.10)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_{t-1} + b_o) \quad (4.11)$$

$$h_t = o_t \tanh(c_t) \quad (4.12)$$

**W** represents the weight matrix for input to each gate, which are the inputs, cell state, and hidden state. The Input gate applies a sigmoid function (4.8) to the input at current time-step, to decide which values to use. The sigmoid scales the values between 0 and 1, where a value of 0 indicates the ignored input. The input is also passed through a tanh function, which scales the input values between -1 and 1. Both outputs are multiplied together (4.10), where the function of the sigmoid output is to either select or drop the tanh output. This is interpreted as assigning a priority value to each input between 0 and 1.

The forget gate is straightforward (4.9). It applies a sigmoid to the inputs, and produces a priority value indicating the previous cell state to keep. Once the values from input and forget gate have been computed, they are combined to determine the update to the cell state (4.10). This is the only time the cell state will be updated, and this cell state will be used in the next time step.

The value of the output gate is determined by passing the input through a sigmoid function (4.11). The updated cell state is passed through tanh function, after which it is multiplied with the output of the output gate. This determines the hidden state (4.12). The updated cell state decides the values to keep and discard, from both the input and past memory. It is combined with the results from the output gate, to be output as the hidden state of the cell.

### 4.4.3 Bidirectional LSTMs

The inputs to a Vanilla LSTMs are fed sequentially. This implies that the LSTM cell preserves more information from the recent past, as opposed to the oldest sample in the input sequence. At the time of the prediction, large number of sequential computations across the time-series can diminish the importance of earlier samples.

Bidirectional LSTMs tackle the issue by utilizing two stacked LSTM cells. Each cell accepts the input sequence in opposite order. The first LSTM accepts inputs in the range  $(x_{t-n}, \dots, x_{t-1}, x_t)$ , whereas the second LSTM

accepts inputs between  $(x_t, x_{t-1} \dots x_{t-n})$ . The benefit is at the time of prediction, the LSTM cell has information from both the oldest sample  $x_{t-n}$  as well as the most recent one  $x_t$ , thereby providing more information context to improve prediction accuracy [26].

#### 4.4.4 Stateless vs Stateful LSTMs

An important hyperparameter when tuning LSTMs is the decision on when to reset the hidden state of an LSTM cell. A cell accumulates information about inputs using various gates, which are re-initialized after each input sequence. This is known as **Stateless LSTM**. The goal of a Stateless LSTM is to learn relationships within an input sequence, as opposed to learning long-term dependencies that may exist between multiple sequences.

A Stateful LSTM does not reset the hidden state after processing each input sequence. Instead it initializes the hidden state for the next input sequence using the state from the last input. This allows a Stateful LSTM to learn long-term dependencies. Based on earlier data analysis, an approximation for the optimal look back window is 124 minutes. A Stateful LSTM will continue to model the relationship between input sequences of length 124, whereas a Stateless LSTM will reset the hidden state after each sequence.

Graves *et al.* [11] demonstrated noticeable improvements by resetting the RNN state after an arbitrarily large sequence, which was of several orders of magnitude greater than the TBTT sequence size. The model was able to perform controlled gradient updates on shorter sequences, while retaining context from past sequences.

### 4.5 Statistical Evaluation Metrics

A loss function in a neural network is used to find the optimal ML model that minimizes overall error. To determine the predictive power of the network, a loss function does not encapsulate all aspects of the quality of the predictions. The simplest metric is **Binary Accuracy**, defined as the number of bit changes required to convert a binary vector to another. This is illustrated by an example below:

$$Y = [1, 1, 0, 1], \hat{Y} = [1, 0, 0, 1]$$

There is only 1 bit change required to convert the predictions to the ground-truth, therefore the accuracy of the model is 75%.

In binary classification problems with imbalanced class ratios, it is helpful to understand the number of times a model predicts each class. For example,



a ML model can achieve 90% accuracy on a dataset with 90% class 0 samples, by predicting class 0 all the time. In context of lighting control, this has drastic implications. Therefore, to accurately describe model predictions, four classification tests, **Precision**, **Recall**, **F1-Score**, and **Matthews Correlation Coefficient** will be used.

### 4.5.1 Confusion Matrix

A confusion matrix categorizes the predictions into four segments, as described in Table 4.1.

Result Type	Description
True Positive (TP)	Both ground-truth and the prediction belong to Class 1.
True Negative (TN)	Both ground-truth and the prediction belong to Class 0.
False Positive (FP)	The predicted Class is 1, whereas the ground-truth is Class 0.
False Negative (FN)	The predicted Class is 0, whereas the ground-truth is Class 1.

Table 4.1: Confusion Matrix for Binary Classification

In lighting control, a True Positive correctly represents occupancy prediction. This will switch on the Luminaire when an occupant is present in the room. Conversely, a TN is a correct prediction of inoccupancy. This switches off the Luminaire when the room is empty.

However, a FP incorrectly predicts occupancy, which will lead to more energy consumption by turning on the Luminaire during state of inoccupancy. Moreover, a FN will switch off the Luminaire in an occupied room, resulting in decreased user experience. From a ML perspective, it is imperative that the model strives to maximise TP and TN, and minimizes FP and FN.

### 4.5.2 Precision/Recall and F1-Score

Precision and Recall are two statistical tests that incorporate some elements of the confusion matrix. Precision is defined as **the number of accurate instances among the predicted instances**, and is given below:

$$\text{Precision} = \frac{TP}{TP + FP} \quad (4.13)$$

Recall is defined as **the ratio of accurate instances from the ground-truth if all the desired instances were retrieved**:

$$\text{Recall} = \frac{TP}{TP + FP} \quad (4.14)$$

The F1-Score is the harmonic mean of both Precision and Recall, defined below:

$$\text{F1-Score} = \frac{2 * (\text{Precision} * \text{Recall})}{\text{Precision} + \text{Recall}} \quad (4.15)$$

### 4.5.3 Matthews Correlation Coefficient

F1-Score does not include True Negatives in its formula. A predictive lighting control systems must accurately predict TNs as well as TPs to minimize the effects of the trade-off between energy consumption and user experience. For this purpose, another statistical test known as **Matthews Correlation Coefficient** (MCC) will be used. It is interpreted as a correlation coefficient between observed and predicted binary classifications, and uses all elements of the confusion matrix.

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \quad (4.16)$$

An MCC outputs a value close to 0, if the model predictions are random guesses. It achieves a value of 1 when the model does not produce FPs and FNs, which is the desirable outcome.

## 4.6 Empirical Evaluation Metrics

From a Machine Learning perspective, having well-defined statistical evaluation metrics describe the predictive power of the model. In applied Machine Learning, the context of the application determines the usefulness of the model. For example, to control light levels and delay timers, it is imperative to evaluate the quality of the model by defining custom metrics which differentiate between predictions with high accuracy, and predictions that also produce efficient lighting control.

### 4.6.1 Energy Consumption

Based on the PIR values in the dataset, the corresponding light levels can be obtained from the database. Each PIR is used to control a specific Luminaire

with a light level on a scale of 0-255. Existing infrastructure is built using domain knowledge and pre-conceived notions about the appropriate values of delay timers. PIR predictions recommend both a light level and delay timer for the Luminaires. The predicted waveform for the test Luminaires is then compared against the actual waveform to determine the difference in the area under the graph. This corresponds to energy savings.

## 4.7 Lighting Control

Controlling light levels through PIR predictions requires empirical evaluation. If the strategy is aggressive, it is more likely to switch off Luminaires which can potentially decrease user experience. A more relaxed strategy is prone to increased energy consumption. As the final output of the network is the probability of a true PIR event at each minute, it is possible to manually control this trade-off using a decision threshold.

[REDACTED]

Once a threshold value is implemented, [REDACTED] need to be determined. This is controlled through a parameter defined as **timeout window**.

[REDACTED]

[REDACTED] This is a similar trade-off between user experience and energy consumption.

For a suitable value of timeout window, user experience can be adversely impacted if the brightness of the Luminaire is instantaneously dimmed to zero. To achieve realistic lighting control, fade timers are used that gradually decrease the light level.

[REDACTED]

## Chapter 5

# Implementation

This section discusses the implementation details of the Machine Learning model explained in the previous chapters, and associated challenges.

### 5.1 Data Pre-processing

Once the raw PIR data has been converted into machine-readable time-series format, it is ready to be fed to the model. Consequently, the dataset needs to be processed into pairs of input and labelled outputs for supervised learning. Each pair consists of an input and output time-series, whose lengths are denoted by look back and look ahead windows. Let  $\mathbf{X}$  and  $\mathbf{Y}$  denote the complete set of data sample pairs, chronologically preserved to maintain temporal relationships. Modifying eq. 3.5 the equation for a single PIR:

$$\mathbf{P} = \{e(t) \mid e \in \{1, 0\} \text{ and } t \in T\} \quad (5.1)$$

Where  $T$  is the index set:  $\mathbf{T} = \{0, 1, 2 \dots N - 1\}$  and  $N$  is the total number of samples in the dataset sampled at 1 minute frequency.

#### 5.1.1 Sliding Window

The dataset is first chained together to form a one-dimensional time-series. This sequence is then split into **Feature** and **Label** space, using look back and look ahead windows. Each subsequence is shifted by 1 sample, to create a Rolling window on the dataset. These are commonly used to transform time-series for inputs to a ML algorithms. Let  $\mathcal{X}$  and  $\mathcal{Y}$  represent the feature and label space respectively:


$$(5.2)$$

$$\begin{aligned} & \text{[Redacted]} \\ & \text{[Redacted]} \\ & \text{[Redacted]} \\ & \text{[Redacted]} \end{aligned} \tag{5.4}$$

At  $\mathbf{t}$ , the input and output to the model is look back and look ahead PIR outputs respectively, where the outputs are preceded by input sequences chronologically. By feeding a continuous time-series, the goal of the model is then to learn possible relationships between the two binary distributions, operating under the assumption that only a finite number of variations of the distributions exist. The trained model can then predict the expected output of the PIR sensor in the form of a time-series, based on the input sequence.

### 5.1.2 Walk-Forward Validation

After predicting look ahead number of samples, the algorithm executes lighting control and awaits further data. [REDACTED]

Therefore, the validation data set is pre-processed in the following way:

$$\begin{aligned} & \left( \frac{\partial}{\partial t} + \nabla_{\vec{v}} \right) \left( \frac{\partial}{\partial t} + \nabla_{\vec{v}} \right) f \\ &= \left( \frac{\partial}{\partial t} + \nabla_{\vec{v}} \right) \left( \frac{\partial}{\partial t} + \nabla_{\vec{v}} \right) f \end{aligned} \quad (5.6)$$

If the length of the sub-sequences for either of the data samples is shorter than their respective window sizes, the sequence is padded with zeros to ensure uniformity.

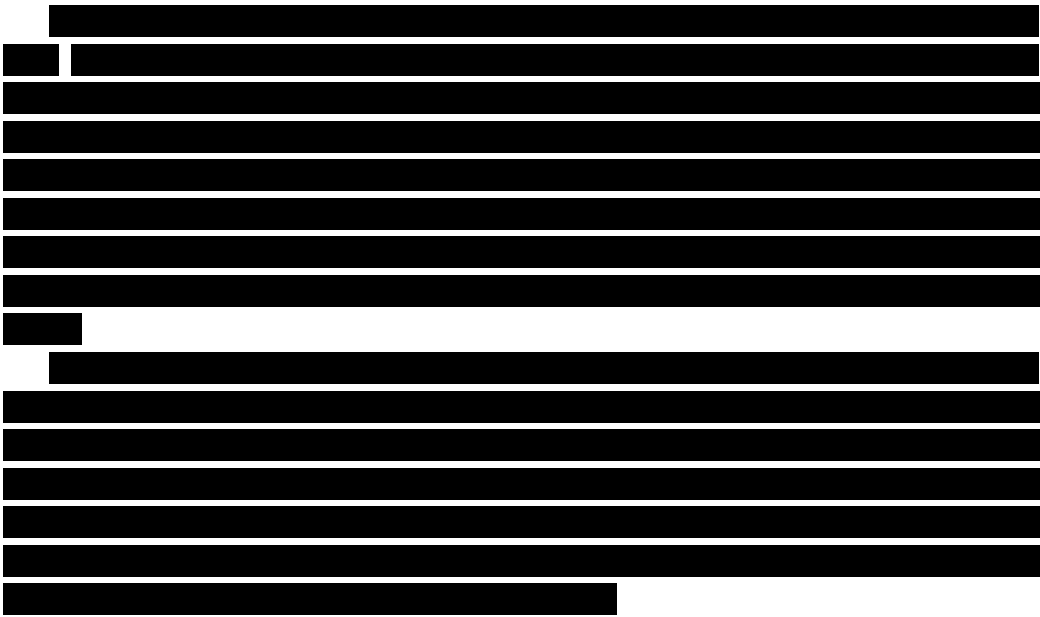
After understanding the underlying functionality of the building blocks of a neural network, it is important to determine the optimal configuration of the connections between various components. Two main types of neural networks are implemented with a brief discussion on each in the following sections.

Designing a Feed Forward network is relatively simpler. The time-series in each data sample is

██████████	██████████	██████████
██████████	██████████	██
██████████████████	██████████	██████████████████ ███████████████████
██████████	██████████	██

The input to an LSTM cell is three dimensional. The first dimension corresponds to the index to the data set. The second dimension is the length of the individual sequence, and the last dimension corresponds to the number of input features. Similarly, the output is also three dimensional with the first two dimensions having the same values. The third output dimension is the number of response variables.

- [REDACTED] [REDACTED] [REDACTED]  
[REDACTED]
- [REDACTED] [REDACTED] [REDACTED] [REDACTED]  
[REDACTED]



- Output: [redacted] [redacted] [redacted]

[redacted]  
The model is able to compare, time-step wise each prediction with its ground truth, and evaluate the loss using binary cross-entropy.

Input		Hidden		Output	
[redacted]		[redacted]		[redacted]	
[redacted]		[redacted]		[redacted]	
[redacted]		[redacted]		[redacted]	
[redacted]		[redacted]		[redacted]	
[redacted]		[redacted]		[redacted]	

Table 5.2: Proposed LSTM Architecture



Figure 5.1: Proposed LSTM Flow Diagram

5.2.3 Bidirectional LSTM

The implementation of the Bidirectional LSTM follows the same architecture, with the difference in the functionality of an individual LSTM cell.



Moreover, the dimensions of the network become enlarged due to the availability of another sequence. The merge operation can be performed in multiple ways; the most common approach is to concatenate both sequences along the feature dimension.

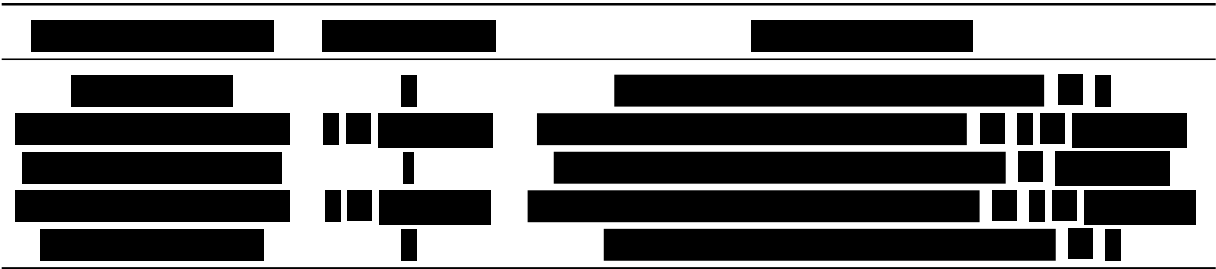


Table 5.3: Proposed Bidirectional LSTM Architecture



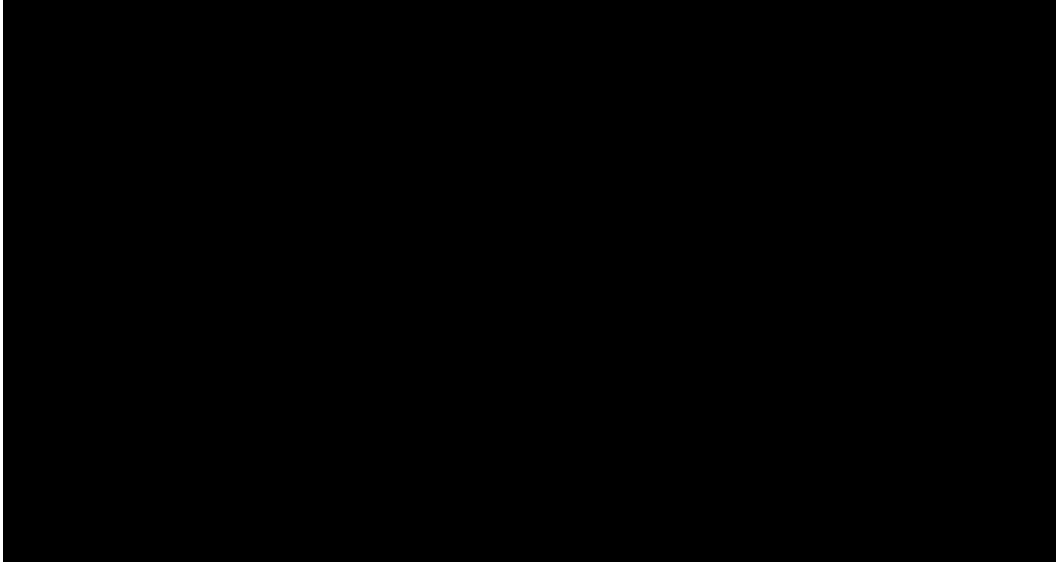


Figure 5.2: Proposed Bidirectional LSTM Flow Diagram

### 5.3 Batch Training in LSTM

Mini-Batch training is a technique used in stochastic gradient descent. Instead of using the complete data set as a single batch for a full gradient update, random sampling from the dataset is performed to select a mini-batch of the data. On expectation, the global minimum loss achieved from mini-batch gradient descent is equivalent to batch gradient descent [9].

Random sampling does not preserve the order of the dataset. For time-series datasets, the sequences must be fed chronologically. Therefore, it is imperative to select mini-batches for input to an LSTM by preserving their sequences. For example, if the mini-batch size is 32, the first 32 subsequences are selected as batch 1, the next 32 subsequences constitute batch 2, and so on. One complete iteration over the whole dataset is known as 1 epoch.

The output of the loss function is a single value for each subsequence, as the cross-entropy loss is a summation operation over the discrepancy between individual predictions and their associated ground truth. Since each mini-batch contains multiple sequences, the loss is usually averaged over all the sequences to be used in gradient descent in deep learning frameworks. It can be assumed that a mini-batch size consisting of a single sequence is logical in the context of time-series, as the network should learn short-term dependencies within 1 sequence, and long-term dependencies between sequences. Using mini-batches of multiple sequences would otherwise average the tem-

poral relationships between the sequences. However, the variance introduced by a batch size of 1 is large, and as a result, the model is unable to learn any meaningful interpretation of the time-series.

Feeding a mini-batch to the LSTM, containing sequences of look back length, produces a similarly sized mini-batch of predictions containing sequences of look ahead length. In a walk-forward validation system, data from the look ahead window is not yet available. The validation model can only accept mini-batch of size 1, which is equivalent to feeding 1 input sequence at a time. This is realized by initiating two neural networks with different dimensions. The first network is trained on the dataset using mini-batches larger than 1. The network weights are then copied to the second network which is used to evaluate the validation dataset.

## 5.4 Problem Definition

Both the feature and label space have been sampled from the input space consisting of all possible PIR events, as defined in equations (5.1), (5.2) and (5.3). To formally define the Machine Learning problem, let  $\mathcal{H}$  denote the hypothesis space consisting of the set of mappings which produce the PIR predictions,  $\hat{\mathbf{y}}$ , based on historical input  $\mathbf{x}$ . It is represented as:

$$\mathcal{H} : \mathcal{X} \rightarrow \mathcal{Y} \quad (5.7)$$

Where  $\mathbf{x}^{(i)} \in \mathcal{X}$  and  $\mathbf{y}^{(j)} \in \mathcal{Y}$  represent vectors in the feature and label space.

Each predictor mapping  $\mathbf{h}$  in the hypothesis space  $\mathcal{H}$  is associated with a collection of weight matrices  $\mathbf{W}$ . Because the architecture of the network contains [REDACTED] the predictor map can be broken down into separate functions. Let  $f$  denote the mapping of  $\mathbf{x}$ , using weight matrix  $\mathbf{W}^{(1)}$  into a hidden representation via the [REDACTED]:

$$\mathbf{h}^{(1)} = f(\mathbf{W}^{(1)}, \mathbf{x}) \quad (5.8)$$

Here  $\mathbf{h}^{(1)}$  represents the hidden state, [REDACTED]  
[REDACTED] consumes a separate weight matrix  $\mathbf{W}^{(2)}$ .

$$\mathbf{h}^{(2)} = f'(\mathbf{W}^{(2)}, \mathbf{h}^{(1)}) \quad (5.9)$$

Lastly, the output from the [REDACTED]  $\mathbf{h}^{(2)}$  is used as an input to the [REDACTED], represented by  $f''$ , to produce predictions  $\hat{\mathbf{y}}$ .

$$\hat{\mathbf{y}} = f''(\mathbf{W}^{(3)}, \mathbf{h}^{(2)}) \quad (5.10)$$

Therefore, a general mapping predictor  $\mathbf{h}$  can be re-written as follows:

$$\hat{\mathbf{y}} = f''(f'(f(\mathbf{x}))) \quad (5.11)$$

Which is equivalent to:

$$\hat{\mathbf{y}} = \mathbf{h}((\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{W}^{(3)}), \mathbf{x}) \quad (5.12)$$

or,

$$\hat{\mathbf{y}} = \mathbf{h}(\mathbf{W}, \mathbf{x}) \quad (5.13)$$

To determine the optimum predictor  $\mathbf{h}$ , the gradient descent algorithm will use Binary cross-entropy as the loss function, discussed in Section 4.3.2. The loss function can be written as follows:

$$\mathcal{J}((\mathbf{x}, \mathbf{y}); \hat{\mathbf{y}}) = \frac{1}{N} \sum_{n=1}^N L(\hat{y}_n, y_n) \quad (5.14)$$

Therefore, the optimal  $\mathbf{h}$  is characterized by the values of the weight matrix  $\mathbf{W}$  which minimize the binary cross-entropy loss between labels  $\mathbf{y}$  and predictions  $\hat{\mathbf{y}}$ .

## 5.5 Flow Chart

To summarize, the complete process for training and evaluation is shown in Figure 5.3. Data from PIR sensors is reformatted for supervised learning, and is split into training, validation and test sets. Each network type is fed with the training data and its performance is evaluated using validation data set. The best model for each network type is chosen, which is further evaluated using Statistical metrics. The selected model is evaluated on test data set, where the predictions are used to control Luminaires. The projected light levels and delay timer values are empirically evaluated against real-world settings.

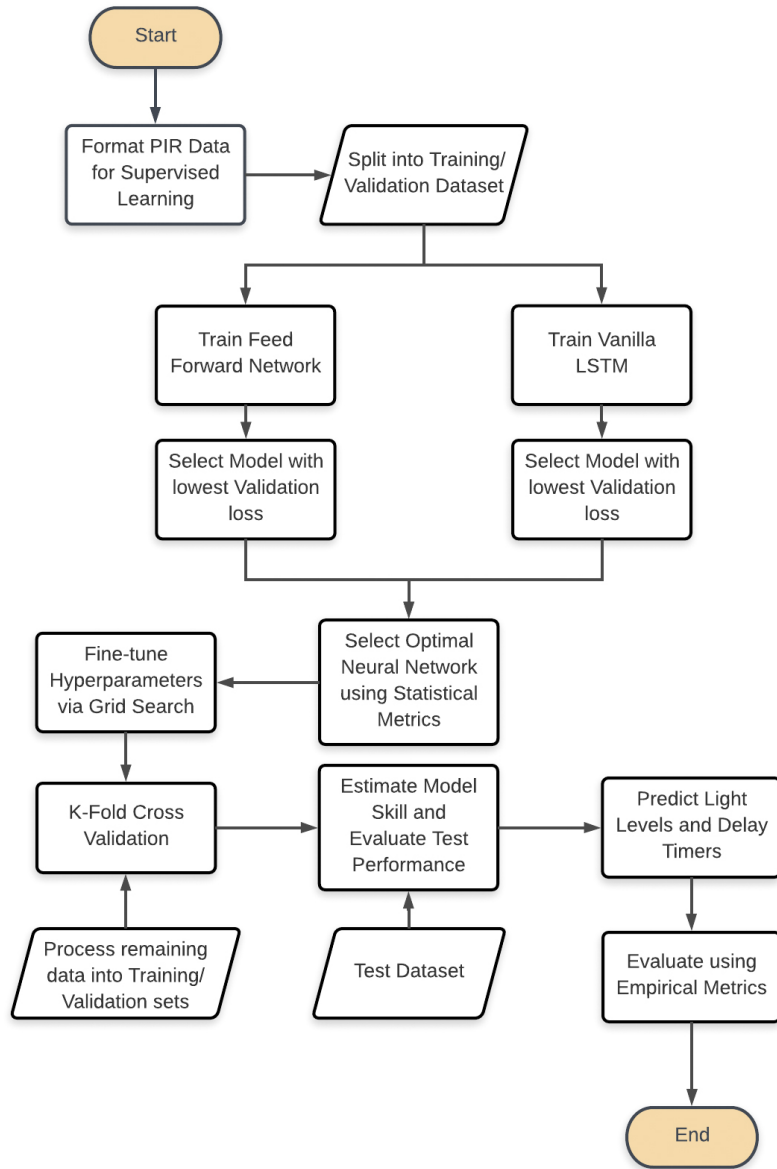


Figure 5.3: Complete Flow Chart

## 5.6 Algorithm

Algorithm for Vanilla LSTM/Bidirectional LSTM training is given below. Using sliding window technique, the data set is initially transformed into input and output sequences. These sequences are grouped together into

mini-batches, while preserving chronology of the time-series. [REDACTED]  
[REDACTED] [REDACTED] [REDACTED] [REDACTED]  
[REDACTED] [REDACTED]  
[REDACTED]

[illegible]

After the predictions have been generated by the neural network, they are transformed into equivalent light levels. As stated earlier, a trade-off exists between energy consumption and delay timers. For a baseline evaluation, the optimization of delay timers in real-time is performed according to a rule-based system.

[REDACTED]

---

**Algorithm 2** Procedure Generate Light Level

---

[REDACTED]

---

## Chapter 6

# Evaluation

Before evaluating the effect of different features in terms of the length of the look back window, the optimal dimension of the neural network is empirically determined. The optimal size of a neural network is ascertained using an exhaustive search of possible network variations, detailed in the next section. Each network variation is run for fixed number of epochs, after which the iteration with the lowest validation loss is recorded. This iteration represents the lowest achievable loss of that network. Finally, all the variations are compared based on individual loss values, to select the optimal network dimension.

### 6.1 Network Dimensioning

The various hyperparameters that are tuneable include the depth, width, activation function, batch size, regularization and learning rate. For each of the proposed network types, an initial grid search is executed to determine the best depth and width of the network. The look back window is ■■■, and the look ahead window is set at ■■■ minutes for all the network variations. The batch size is set at ■■■.

#### 6.1.1 Feed Forward Network

The Feed Forward network is evaluated using various values for depth and width, give in Table 6.1. Each hidden layer uses ■■■ as activation function. The output layer uses ■■■■ as the activation function. The network was trained using ■■■ optimization algorithm with the default learning rate of ■■■.

Width								
Depth	Train	Valid	Train	Valid	Train	Valid	Train	Valid
	0.426	<b>0.439</b>	0.426	0.444	0.424	0.443	0.454	<b>0.438</b>
	0.418	0.447	0.451	0.451	0.419	0.45	0.402	0.461
	0.421	0.458	0.414	0.463	0.419	0.467	0.402	0.46

Table 6.1: Training Validation Loss for Feed Forward Network

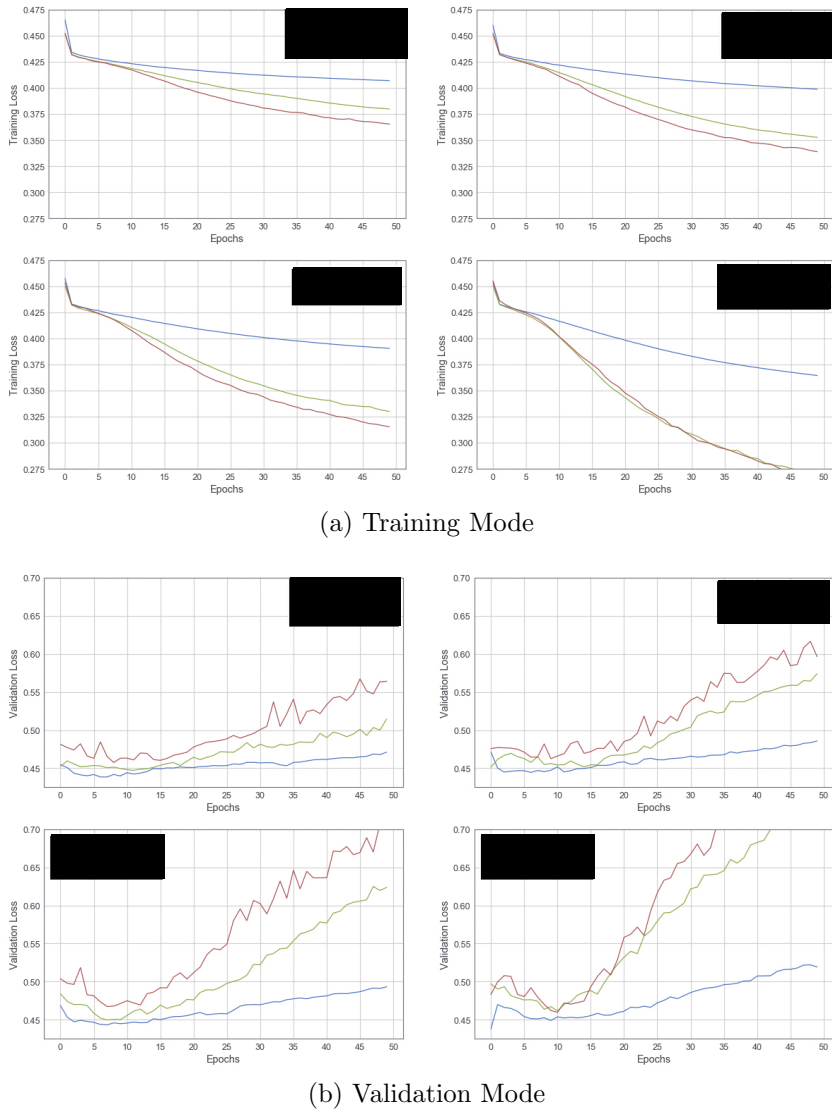


Figure 6.1: Visualizing Loss Function for Feed Forward Network



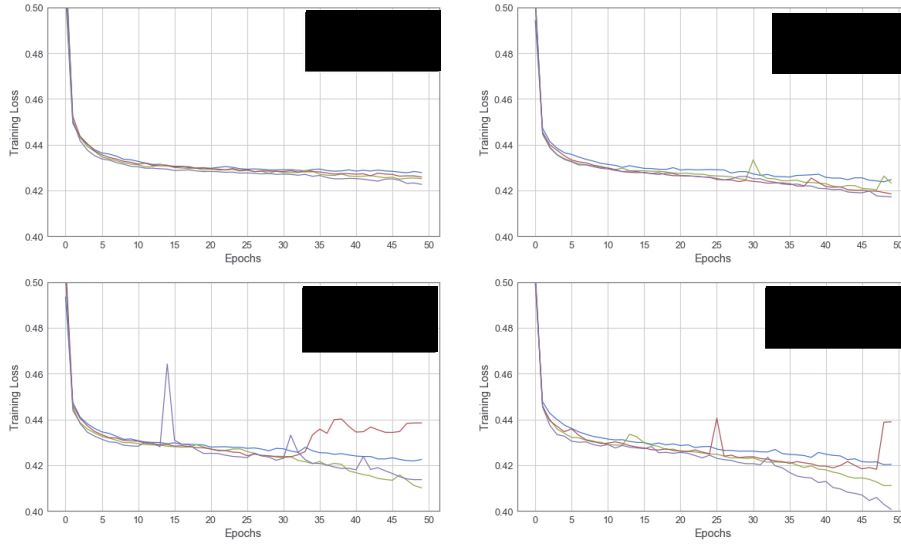
The results show that the model with the lowest validation loss is a `nn.Sequential` Feed forward network. Moreover, a `nn.LSTM` network with a width of `128` provides approximately the same validation loss, with a marginal difference of 0.22%. A significantly lower training accuracy is achieved with a `nn.LSTM` depth network, however, higher capacity models tend to overfit the training dataset. Using the above results, the selected model is a `nn.Sequential` because of similar performance with less computational requirements.

6.1.2 Vanilla LSTM

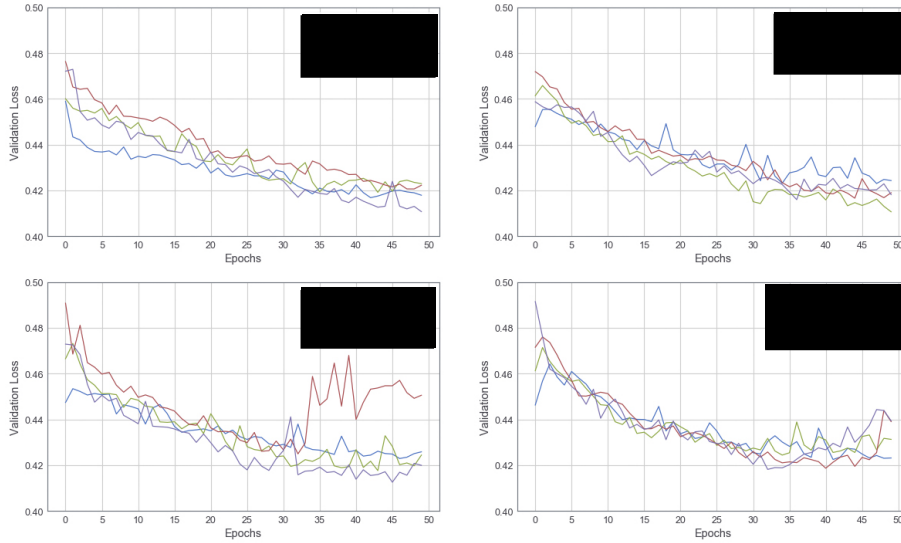
The Vanilla LSTM is implemented using `nn.LSTM`. Similarly, the width of the network refers to the number of neurons in each LSTM cell. These cells use `nn.ReLU` as the default activation function. The output layer is a distributed fully connected layer with `nn.ReLU` as the activation function. The network is again trained with `nn.Adam` algorithm, with the default learning rate of `0.001`. Moreover, a default dropout value of `0.5` was used.

Width	Width	Train	Valid	Train	Valid	Train	Valid	Train	Valid
128	128	0.429	0.417	0.424	0.423	0.423	0.423	0.424	0.422
128	128	0.426	0.419	0.423	<b>0.411</b>	0.414	0.418	0.421	0.424
128	128	0.426	0.421	0.42	0.417	0.425	0.425	0.42	0.419
128	128	0.423	<b>0.411</b>	0.423	0.416	0.416	0.413	0.423	0.418

Table 6.2: Training Validation Loss for Vanilla LSTM Grid Search



(a) Training Mode



(b) Validation Mode

Figure 6.2: Visualizing Training and Validation Loss for Vanilla LSTM

LSTMs exhibit similar trade-off between model capacity and validation loss. Deeper models exhibit lower training and higher validation loss. Based on the results, two models achieve the same validation loss in 50 epochs, [redacted] neurons, and [redacted] neurons. Because of the difference between training times, the [redacted] model is considered optimal as it provides the same performance in less computation time.

## 6.2 Evaluation using Statistical Metrics

Based on the results of the grid search on network design, the best performing models for each type of neural network are selected. These are [REDACTED] Feed Forward Network, and [REDACTED] LSTM. To differentiate between the two models, it is necessary to involve other metrics to make the decision more robust. Using the statistical metrics defined in Section 4.5, both models are evaluated on the validation dataset and the results are summarized in Table 6.3.

Network	Classification Accuracy	Precision	Recall	F1-Score	MCC
FFN	80.8%	0.761	0.817	0.788	0.615
LSTM	80.9%	0.772	0.797	0.785	0.614

Table 6.3: Metric values at threshold 0.5

It is evident that there is negligible difference between the models, however, it is important to realize that binary classification metrics require prediction probabilities to be rounded off to the nearest class. The default threshold is 0.5, i.e. a probability of at least 0.5 is considered as Class 1, or a True event. Values lower than 0.5 are classified as Class 0. In other words, the optimal value for the classification metrics might vary for different thresholds. Figure 6.3 shows the variation of each metric against different threshold values.

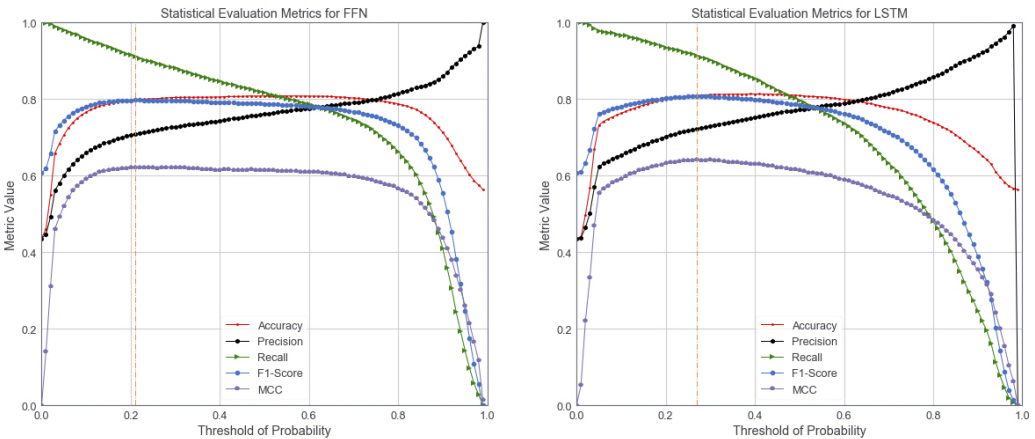


Figure 6.3: Statistical Metrics

There is trade-off between Precision and Recall, but a higher Recall is beneficial as it reduces the number of false-off events. Based on the two graphs, the peak F1-Score and MCC value occurs at a threshold value of 0.27 for LSTM, and 0.21 for FFN, summarized in Table 6.4. LSTM outperforms FFN in each metric at optimal threshold value. Moreover, it is evident that the threshold favours higher values of Recall in the Precision-Recall trade-off.





Network	Classification Accuracy	Precision	Recall	F1-Score	MCC
<b>FFN (t=0.21)</b>	79.8%	0.709	0.911	0.798	0.622
<b>LSTM (t=0.27)</b>	81.0%	0.723	0.914	<b>0.807</b>	<b>0.641</b>



Table 6.4: Metric values at optimal thresholds

## 6.3 Model Tuning

Based on the results from the previous section, the selected network is LSTM. Previously, the LSTM was trained using default values for the hyperparameters. Since it is computationally infeasible to run a holistic grid search on all possible combinations of different parameters, it is desirable to vary each parameter individually, while fixing the rest of the parameters. This does not provide definitive parameter values, rather it is used to identify optimal values. This section executes a grid search on the chosen network type to determine if further gains are achievable.

### 6.3.1 Learning Rate, Batch Size and Dropout Regularization

The three chosen hyperparameters are Learning rate, Batch size and Dropout regularization. The evaluated values of Learning rate are , , for batch sizes  and dropout of . The results for Learning rate are visualized in Fig A.1 and tabulated in A.1. The results for batch size are visualized in Fig A.2 and tabulated in A.2. The results for dropout are visualized in Fig A.3 and tabulated in A.3.

Very small learning rate predictably demonstrates poor learning capabilities on both the training and validation set. Conversely, a large learning rate exhibits wildly fluctuating descent curve. Both  are arguably similar, therefore the chosen learning rate is  based on marginally improved validation dataset performance.

The difference in validation set performance is negligible for `128` and `256` batch size. A batch size of `128` appears to perform larger updates during gradient descent in the initial epochs, and achieves an overall lower loss. A batch size of `256` achieves the lowest loss on training dataset, but performs poorly on validation data. Moreover, the final few epochs of the validation curve demonstrate that the loss values for larger batch sizes is plateauing, and they eventually reach approximately the same minimum loss. There is significant improvement in training time as well for larger batch sizes. Therefore, a batch size of `128` is selected.

The regularization trade-off between model capacity and validation performance is evident here. `0.5` dropout, `0.5`, shows consistently high training loss but manages to perform well on the validation dataset, after about 20 epochs. No dropout produces the lowest training loss but performs poorly on validation set. Moreover, apart from a dropout of `0.5`, all other loss curves demonstrably increase in the final epochs. This indicates that learning is still possible with a dropout of `0.5`, hence it is chosen as the optimal value.

The LSTM network was re-trained using the combination of optimal hyperparameter values, for 100 epochs. The combination with the best result is given in Table 6.5. The lowest achievable validation loss remains at 0.411.

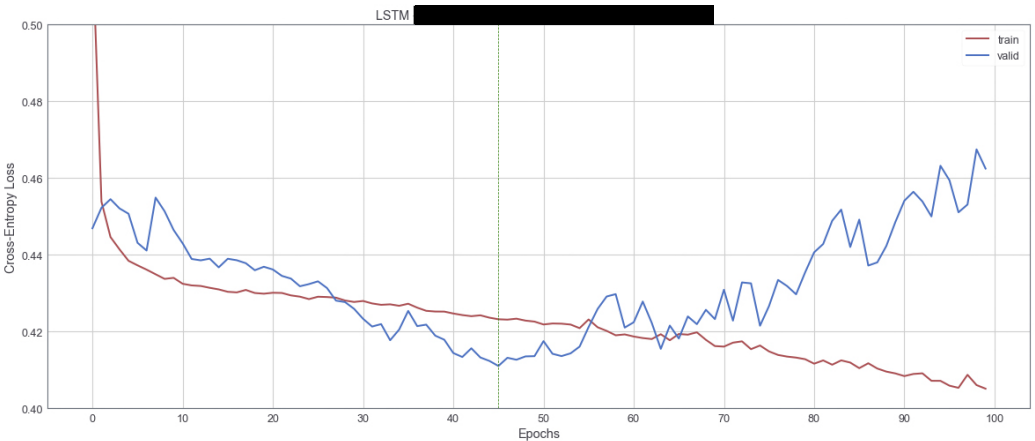


Figure 6.4: Optimal LSTM Loss Function

Train Loss	Valid Loss
0.423	0.411

Table 6.5: Loss Values for Optimal LSTM Model

### 6.3.2 Bi-directional LSTM

Bi-directional LSTMs outperform vanilla LSTMs in a variety of tasks. Using the optimal network parameters from the previous sections, the model is re-implemented using Bi-directional LSTM cells instead, to determine if further improvement is possible. The results of the training and validation loss, combined with statistical metrics are given in Figure 6.5 and Table 6.6.

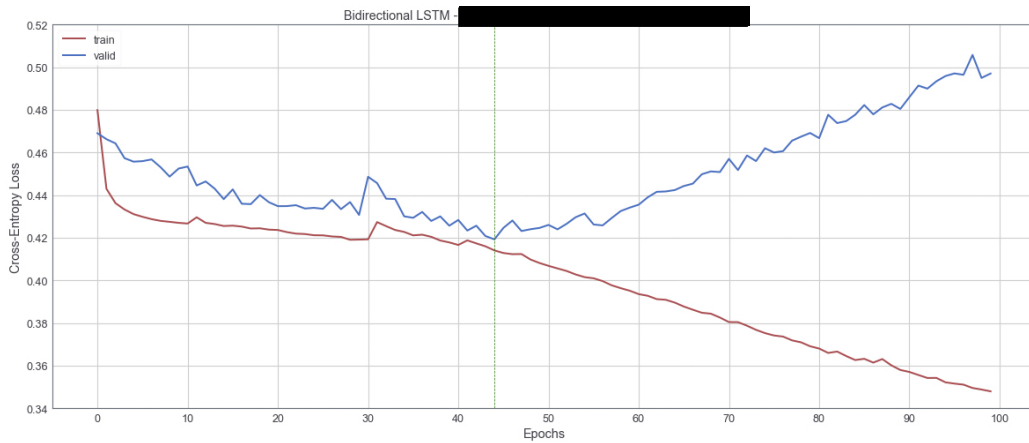


Figure 6.5: Bi-directional LSTM Loss Function

Train Loss	Valid Loss
0.414	0.419

Table 6.6: Loss Values for Bi-directional LSTM

Bi-directional LSTM performs slightly worse on the validation dataset while achieving a much lower training loss for the same number of epochs. This is because a Bi-directional LSTM cell has twice the model capacity. Moreover, bi-directional LSTMs perform well on language translation models where the context of the whole sentence dictates the next word. In autoregressive time-series problems, the oldest information might not be as relevant as the most recent sensor activity. This explains the similar performance of the bi-directional model.

With appropriate regularization, a Bi-directional LSTM would be able to achieve comparable performance, however, given the much higher computational requirements and training time, the trade-off between performance gains and computational requirements is infeasible.

### 6.3.3 State Management in LSTMs

As discussed in Section 4.4.4, the hidden state of an LSTM from the last time step is consumed to produce the next output. By iteratively updating and consuming the hidden state, an LSTM cell can retain information about the past occurrences of events for a continuous time-series sequence. If the hidden state is never reset, the last output of the cell would be conditionally dependent on the hidden state from the earliest time-steps. This dependence will become insignificant, contingent on the increasing distance between the two data points. In addition, the contribution from the earliest time steps will be infinitesimally small in gradient updates. This section deals with the effects of manually resetting the internal state of an LSTM cell.

#### 6.3.3.1 Stateless LSTMs

For all the previous experiments, the hidden state was forced to persist across all samples. A variant of an LSTM cell, which resets the state every time a sample has been processed, is implemented. It does not imply that the hidden state does not persist across consecutive time-steps, rather it is only retained for a single subsequence i.e. [REDACTED]. Once the network trains on 1 sample of size *look back*, the state is manually reset. Conceptually, the LSTM cell should only learn patterns within a subsequence whilst ignoring inter-sequence dependencies.

The results for a Stateless LSTM cell are given below:

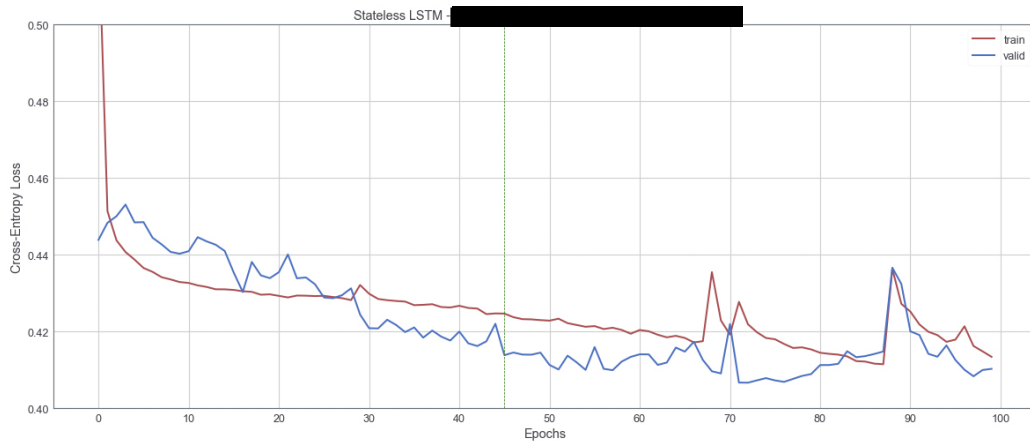


Figure 6.6: Stateless LSTM Loss Function

Train Loss	Valid Loss
0.421	0.406

Table 6.7: Loss Values for Stateless LSTM

The Stateless LSTM delivers improved performance on the validation set. Assuming the optimal look back window is [REDACTED] based on empirical ACF and PACF plots, coercing the LSTM cell to retain memory across longer time-steps should produce negligible gains. This has now been empirically verified, as the model achieves a lower loss value when it is trained to treat each subsequence independently. Furthermore, the performance has slightly improved as well, possibly due to the network learning spurious correlations with samples beyond [REDACTED] which eventually decreases model skill.

### 6.3.3.2 Resetting States Manually

The distribution of all the PIR sensors was shown to be concentrated within working hours in Section 3.5.2. To generalize, the expected pattern of a PIR sensor in a day can be used to provide bounds on the memory of an LSTM cell. This is achievable by feeding input sequences to an LSTM cell and allowing the cell to retain long-term memory up till the end of the [REDACTED] period. The hidden state is persisted across [REDACTED] after which it is reset. The results are shown in Figure 6.7. The performance is similar to a stateful LSTM, and does not provide any significant benefits.

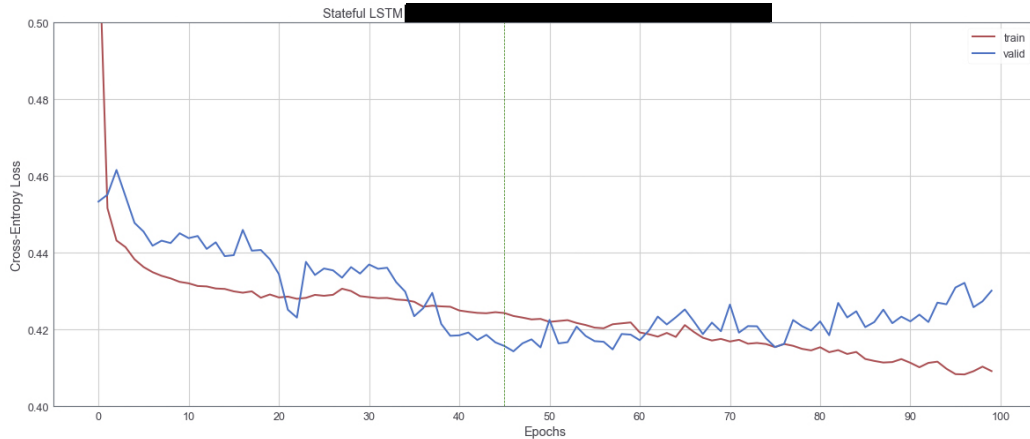


Figure 6.7: Managed State LSTM Loss Function



Train Loss	Valid Loss
0.424	0.414

Table 6.8: Loss Values for Managed State LSTM

## 6.4 Lookback Feature Selection

Various methodologies and model parameters were tested in the previous sections. Based on the results, a stateless LSTM is selected in the proposed Encoder Decoder network framework as the optimal strategy for the given machine learning problem. This section deals with evaluating the effect of the look back window on forecasting accuracy. The chosen look back windows are ██████████ minutes. Each instance is evaluated using both cross entropy loss, and classification accuracy.

LookBack	Cross-Entropy Loss		Accuracy	
	Train	Valid	Train	Valid
<span style="background-color: black; color: black;">██</span>	0.422	0.413	80.9%	81.2%
<span style="background-color: black; color: black;">███</span>	0.422	<b>0.409</b>	81.0%	<b>81.4%</b>
<span style="background-color: black; color: black;">████</span>	0.424	0.411	80.9%	81.3%
<span style="background-color: black; color: black;">█████</span>	0.422	<b>0.408</b>	81.0%	<b>81.5%</b>

Table 6.9: Lowest Values of Loss Function for Look Back Windows

Window sizes higher than ███ minutes exhibit erratic fluctuations after the network has been training for more than 50 epochs. The loss value for lookback of ███ for stateless LSTM was 0.406, similar to the value attained by lookback window of ███ minutes. The slight variations in loss values can be explained by considering random weight initializations in the network. Moreover, the accuracy is negligible across the look back windows.

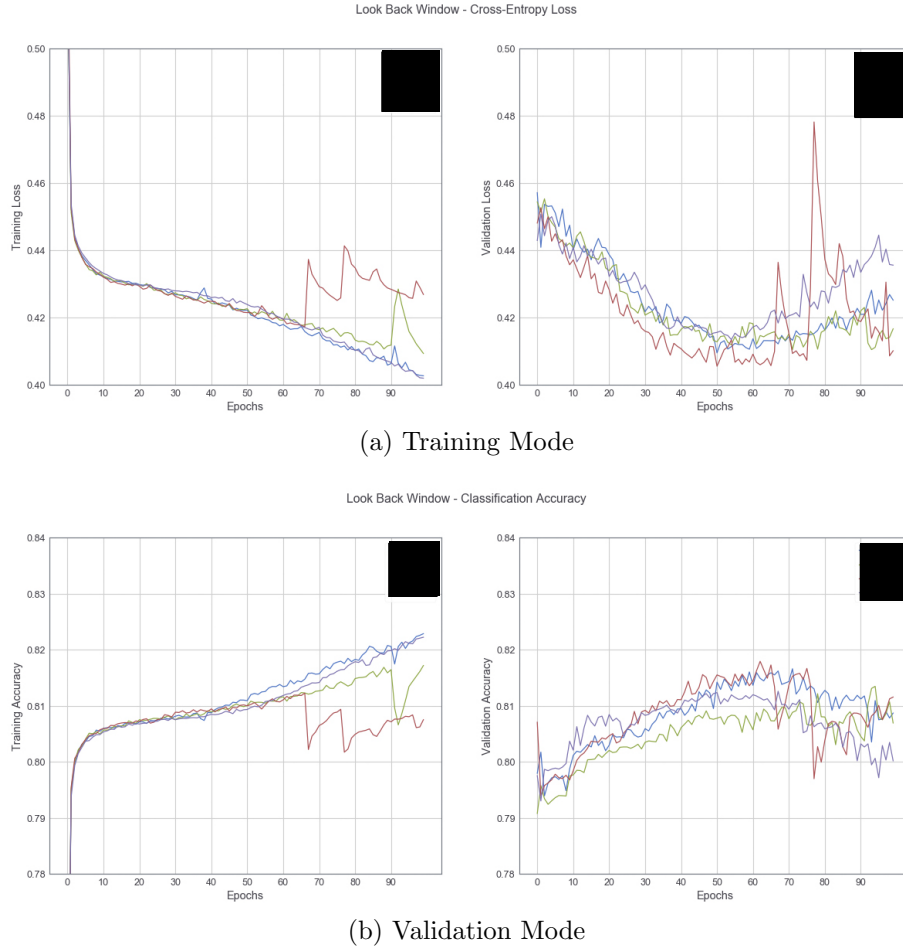


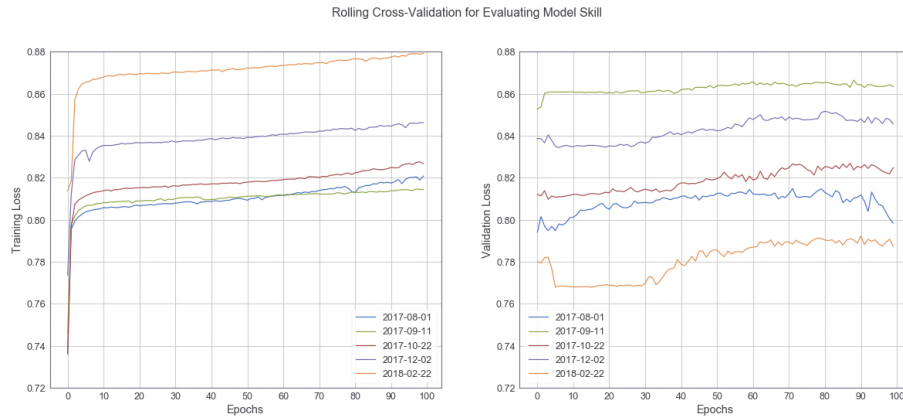
Figure 6.8: Loss Function for Different values of Look Back Window

## 6.5 Estimating Model Skill

Using rolling cross-validation, the model is re-trained one-by-one on the 5 subsets of the dataset. For each subset, last-block validation technique is used to fix the relative placements of training and validation set chronologically. Figure 6.9 shows the loss function plots for each dataset. Table 6.10 lists summary statistics, which are used to describe the expected performance of the deep learning model.



(a) Training Mode



(b) Validation Mode

Figure 6.9: Loss Function for Rolling Cross-Validation

Rolling CV	Mean	Std. Dev.	Std. Error	Min	Max
Cross-Entropy Loss	0.381	0.058	0.026	0.305	0.459
Accuracy	81.20%	3.30%	1.50%	76.80%	85.30%

Table 6.10: Descriptive Statistics for Model Skill

In forecasting problems, the baseline is often called the **naive forecast**, which is a replication of the samples from the look back window. Each model with the lowest validation from 5-fold cross validation is now compared against a 10-step naive forecast, for both cross-entropy loss and classification accuracy. The forecast horizon consists of samples from  $\mathbf{x}_{t-1}$  to  $\mathbf{x}_{t-10}$ . The results are summarized in Table 6.11.

Validation Sets					
Cross-Entropy Loss	1	2	3	4	5
Naive Forecast	3.779	2.684	3.489	3.073	4.138
Trained Model	0.407	0.305	0.382	0.353	0.459
% Reduction	89.23%	88.64%	89.05%	88.51%	88.91%
Classification Accuracy	1	2	3	4	5
Naive Forecast	76.40%	83.30%	78.20%	80.80%	74.20%
Trained Model	81.50%	86.40%	82.50%	84.90%	79.10%
% Increase	6.26%	3.59%	5.21%	4.83%	6.19%

Table 6.11: Loss and Accuracy Values for Naive Forecast

It is evident that the trained model produces a higher classification accuracy at a lower cross-entropy loss. If the time-series were completely random, the model would have produced a naive forecast. The results validate the hypothesis that the series is not completely random, rather there are repeating patterns and non-linearities that are captured by the LSTM model.

## 6.6 Controlling Luminaires

For each validation set, the luminaires that are associated with each sensor are first identified. The light levels for each luminaire are then obtained corresponding to the dates in the dataset. Rolling predictions are fed to the lighting control algorithm to generate recommendations for each luminaire. These recommendations are compared with the actual light levels by calculating the area under both curves. The light levels are generated using ■ timers, with delay timers set at ■ minutes. The recommended light level is set at ■ of the current luminaire setting. This is desirable instead of using absolute values, because the luminaires are also controlled through other sensors such as ambient light. Figure 6.10 shows the output of a sensor and its associated luminaire for 1 day, whereas Table 6.12 summarizes the results of computing the area under the curve for all validation data sets. Due to non-availability of detailed PIR control messaging, it is not possible to evaluate the impact on user-experience with the validation dataset.

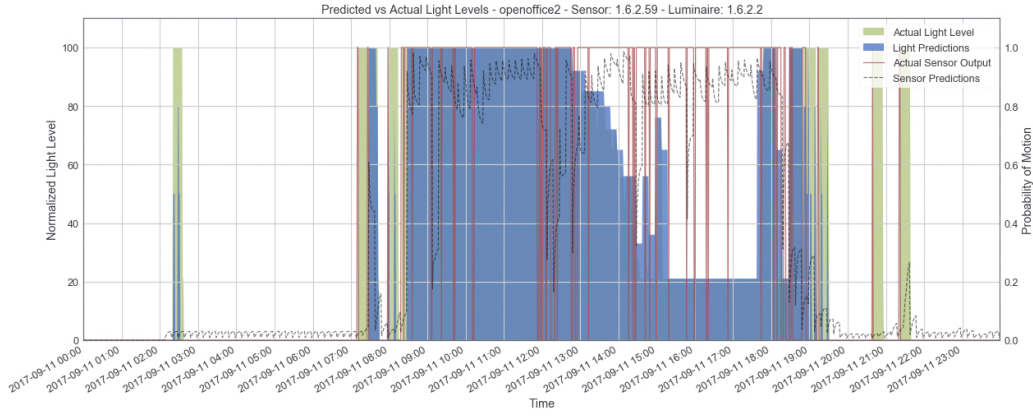


Figure 6.10: Sample output of the Lighting Control Algorithm

The green plot shows the actual light level, whereas the blue plot is the recommended light level. The drop in light levels after 13:00 follow ambient light sensor recommendations. To account for such unexpected variations, the algorithm [REDACTED] to determine the next recommendation. Moreover, the effects of large delay timers are evident at the 19:00 mark. The luminaires continue to remain on, even though the sensor registers little activity. This makes it possible to incur energy savings by predictably turning off the luminaire. However, a false-negative event occurs at 21:00 when a stochastic PIR trigger takes place exhibiting the limitations of such a system.

The sensor is installed at the working desks, and predictably observes consistent activity. Therefore, the energy savings are comparatively lower with a mean value of 14%.

Model Number	Sensor					
	1	2	3	4	5	6
1	16.39%	63.84%	15.38%	8.87%	6.09%	24.83%
2	22.80%	58.32%	14.16%	7.17%	9.58%	9.93%
3	6.86%	46.74%	10.41%	8.17%	4.76%	10.67%
4	-	56.43%	13.16%	10.08%	9.70%	13.31%
5	22.94%	64.70%	15.00%	17.65%	23.58%	30.74%

Table 6.12: Energy Savings for Selected Sensors for all Validation sets

Mean	Std. Dev.	Min	Max
21.46%	18.27%	4.76%	64.70%

Table 6.13: Descriptive Statistics for Energy Savings

The mean energy saving is 26.20%, with a standard deviation of 18.57%. This is best explained by analyzing the patterns of individual sensors, as some sensors observe high activity compared to others. For these sensors, the algorithm will not be able to generate significant energy savings because consistent activity ensure that the luminaire rarely switches off. Conversely, for areas with sporadic activity, such as meeting rooms, larger delay timers ensure the light stays on even after the occupant has left the room, which provides the opportunity to incur higher savings once non-occupancy has been predicted.

## 6.7 Performance on Test Set

The last step is to evaluate the performance of the trained models on the test dataset. According to the assumption of continuity, and the relation between forecast horizon and forecast error, models trained on data from August should perform adversely on test data, as it has been selected from the end of the dataset i.e. April. To verify the hypothesis, optimal models from each of the 5-folds are evaluated against the test dataset through both statistical and empirical metrics.

Model Number	Test Loss	Accuracy	Precision	Recall	F1-Score	MCC
1	0.457	79.1%	0.743	0.905	0.816	0.594
2	0.459	78.9%	0.739	0.908	0.815	0.591
3	0.456	79.3%	0.747	0.902	0.817	0.597
4	0.458	79.5%	0.764	0.869	0.813	0.595
5	0.462	78.7%	0.739	0.905	0.814	0.588

Table 6.14: Test set Performance

Based on the results in Table 6.14, the lowest validation loss occurs for model number 3, trained on the December dataset. Moreover, the worst performing model was trained on the last validation dataset from April. This implies that the values obtained after evaluating the earliest and latest

trained models on the test set are similar, and the model does not need to be re-trained once new sensor data has been collected.

Model Number	Sensor					
	1	2	3	4	5	6
1	34.85%	18.08%	19.15%	75.76%	19.67%	17.25%
2	31.92%	17.17%	17.99%	75.23%	18.76%	17.08%
3	35.16%	18.03%	19.00%	74.37%	18.97%	18.74%
4	38.50%	21.97%	22.29%	77.08%	22.65%	21.70%
5	31.79%	16.41%	16.49%	73.72%	17.72%	17.51%

Table 6.15: Energy Savings on Test set

Mean	Std. Dev.	Min	Max
30.83%	21.10%	16.41%	77.08%

Table 6.16: Descriptive Statistics for Energy Savings on Test Set

The behaviour of the various trained models on prediction quality is consistent across each sensor. For example, the minimum and maximum energy savings for Sensor 2 have a difference of approximately 5%. Furthermore, the mean energy savings are 30.8%, with the overall energy saving for different sensors varying from approximately 16% to 77%.

## Chapter 7

# Discussion

The established research in building automation systems is focused on predicting the number of occupants in a room; or the resolution of forecast horizon is large enough to reduce the stochasticity of human behaviour, as discussed in Section 2.3.3. It is possible to predict number of occupants for the next 30 minutes, and develop a robust predictive model for HVAC systems. The most significant difference between both systems is that HVAC is less chaotic and requires more time to effectively control indoor climate, whereas lighting is highly sensitive to human motion patterns. Therefore, it is insufficient to simply develop a model that generates an overall probability of occupancy for the complete forecast horizon; rather, the expected deviations in the pattern of occupancy over the horizon are significantly more intuitive in the context of lighting control decisions.

Traditional time-series forecasting techniques require detailed statistical analysis of the underlying data, in order to develop robust prediction models. Neural networks, however, do not require a thorough analysis on time-series data, and have demonstrated comparable performance. These models suffer from parameter tuning issues, require huge amounts of data, and are generally difficult to train. On the other hand, neural networks can more effectively capture non-linearities in the data. By augmenting traditional statistical analysis tools for time-series with the modelling capabilities of neural networks, it is possible to use recurrent networks for time-series forecasting problems.

Time-series in existing research is highly seasonal. For example, popular time-series climate datasets can be de-trended to produce accurate forecasts using simpler models. In contrast, the highly stochastic data analyzed in this study is dependent on human behaviour in office environments, which reduces the forecast accuracy in comparison. Even with the daily seasonality, lighting control requires a higher forecasting resolution that restricts the usage of



techniques such as time-series decomposition through moving averages. By de-trending the series, a simpler waveform is obtained that represents motion during office hours. The neural network will predictably produce higher forecast accuracy on the simpler de-trended series, however, the results will not be useful for fine-grained lighting control.

The results of the study indicate that neural networks, such as LSTMs, are able to learn from sensor data as the series is not a random walk. This is inferred from the comparison between naive forecasting and model output. An incorrectly tuned time-series model suffer from oversimplification that tends to produce a replication of recent observations in the forecasting horizon. Similar effects are observed for random walk time-series. Nevertheless, it has been shown that a properly tuned LSTM model is able to model the stochasticity of data; and is more accurate than a naive forecast even if it is unable to achieve very high forecast accuracy. Moreover, the quality of the forecast largely depends on the model predicting the continuation of occupancy as opposed to the start of occupancy.

it has been demonstrated that increased efficiency in lighting control leads to significant reduction in energy consumption.

Ideally, the algorithm should optimize light levels by determining

This is important as sensor patterns vary between installations, and the non-occupancy decision window should vary depending on the type of sensor. For purposes of the study, the implemented algorithm uses an oversimplified assumption that light levels should be reduced

Nonetheless, energy savings have shown to be significant. This establishes that detailed optimization algorithms can increase energy savings. Lastly, more data is required to determine the impact of predictions on user experience. The trade-off between energy consumption and user experience places a theoretical bound on the maximum attainable savings from the lighting control algorithm. Therefore, it is recommended to frame this as an optimization problem requiring further research.

## Chapter 8

# Conclusions

Neural networks are powerful tools that are able to model complex functions and have the potential to solve real-world problems. One such problem relates to optimizing building automation systems, particularly indoor lighting control. Efficient systems are highly desirable and have been the subject of a multitude of research activity. More importantly, energy efficiency is an important aspect of the global drive to reduce carbon footprints and minimize the impact of electricity production.

The objective of the study was to develop a predictive model with the ability to forecast indoor occupancy. Traditional lighting control systems react to occupancy changes and are prone to inefficient energy consumption patterns. By developing machine learning models that anticipate the duration of occupancy based on historical data, it has been demonstrated that significant energy savings are realizable. Intelligent systems that consume anticipatory knowledge can make real-time decisions, which can be translated into efficient building automation systems.

It can be stated with confidence that the stated methodology and its outcomes reinforce the notion that decision making based on time-series forecasting has demonstrable potential. Furthermore, continued research is encouraged to improve upon the accuracy and quality of the results.

# Bibliography

- [1] ADAMOPOULOU, A. A., TRYFERIDIS, A. M., AND TZOVARAS, D. K. A context-aware method for building occupancy prediction. *Energy and Buildings* 110 (2016), 229–244.
- [2] BAKKER, C. D., VOORT, T. V. D., AND ROSEMAN, A. The Energy Saving Potential of Occupancy-Based Lighting Control Strategies in Open-Plan Offices: The Influence of Occupancy Patterns. *Energies* 11, 1 (2017), 2.
- [3] BAUMGARTNER, T., WUNDERLICH, F., JAUNICH, A., SATO, T., BUNDY, G., GRIESSMANN, N., KOWALSKI, J., BURGHARDT, S., AND HANEBRINK, J. Lighting the way: Perspectives on the global lighting market. *McKinsey & Company* (2012), 1–58.
- [4] BELLIDO-OUTEIRINO, F. J., FLORES-ARIAS, J. M., DOMINGO-PEREZ, F., GIL-DE-CASTRO, A., AND MORENO-MUNOZ, A. Building lighting automation through the integration of DALI with wireless sensor networks. *IEEE Transactions on Consumer Electronics* 58, 1 (2012), 47–52.
- [5] CHO, K., VAN MERRIENBOER, B., GULCEHRE, C., BAHDANAU, D., BOUGARES, F., SCHWENK, H., AND BENGIO, Y. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation.
- [6] CRONE, S. F., HIBON, M., AND NIKOLOPOULOS, K. Advances in forecasting with neural networks? Empirical evidence from the NN3 competition on time series prediction. *International Journal of Forecasting* 27, 3 (2011), 635–660.
- [7] DE BAKKER, C., ARIES, M., KORT, H., AND ROSEMAN, A. Occupancy-based lighting control in open-plan office spaces: A state-of-the-art review. *Building and Environment* 112 (2017), 308–321.

- [8] FAYYAD, U., PIATETSKY-SHAPIO, G., AND SMYTH, P. From Data Mining to Knowledge Discovery in Databases. *AI Magazine* 17, 3 (1996), 37.
- [9] GOODFELLOW, I., BENGIO, Y., AND COURVILLE, A. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [10] GRAVES, A. Generating Sequences With Recurrent Neural Networks. 1–43.
- [11] GRAVES, A., AND SCHMIDHUBER, J. Framewise phoneme classification with bidirectional LSTM networks. *Proceedings of the International Joint Conference on Neural Networks* 4 (2005), 2047–2052.
- [12] HAQ, M. A. U., HASSAN, M. Y., ABDULLAH, H., RAHMAN, H. A., ABDULLAH, M. P., HUSSIN, F., AND SAID, D. M. A review on lighting control technologies in commercial buildings, their performance and affecting factors. *Renewable and Sustainable Energy Reviews* 33 (2014), 268–279.
- [13] HO, S., XIE, M., AND GOH, T. A comparative study of neural network and Box-Jenkins ARIMA modeling in time series prediction. *Computers & Industrial Engineering* 42, 2-4 (2002), 371–375.
- [14] HOCHREITER, S., AND SCHMIDHUBER, J. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780.
- [15] JAMES, G., WITTEN, D., HASTIE, T., AND TIBSHIRANI, R. *An Introduction to Statistical Learning*, vol. 103 of *Springer Texts in Statistics*. Springer New York, New York, NY, oct 2013.
- [16] KHASHEI, M., AND BIJARI, M. An artificial neural network (p, d, q) model for timeseries forecasting. *Expert Systems with Applications* 37, 1 (2010), 479–489.
- [17] MAKRIDAKIS, S. G., WHEELWRIGHT, S. C., AND HYNDMAN, R. J. Forecasting: Methods and Applications. *Journal of Forecasting* (1998), 1.
- [18] MITCHELL, T. M. *Machine Learning*. McGraw-Hill, Inc. New York, NY, USA ©1997, 1997.
- [19] PANDHARIPANDE, A., AND CAICEDO, D. Smart indoor lighting systems with luminaire-based sensing: A review of lighting control approaches. *Energy and Buildings* 104 (2015), 369–377.

- [20] QOLOMANY, B., AL-FUQAHA, A., BENHADDOU, D., AND GUPTA, A. Role of Deep LSTM Neural Networks and Wi-Fi Networks in Support of Occupancy Prediction in Smart Buildings. *Proceedings - 2017 IEEE 19th Intl Conference on High Performance Computing and Communications, HPCC 2017, 2017 IEEE 15th Intl Conference on Smart City, SmartCity 2017 and 2017 IEEE 3rd Intl Conference on Data Science and Systems, DSS 2017 2018-Janua*, SmartCity (2018), 50–57.
- [21] RYU, S. H., AND MOON, H. J. Development of an occupancy prediction model using indoor environmental data based on machine learning techniques. *Building and Environment* 107 (2016), 1–9.
- [22] SEABOLD, S., AND PERKTOLD, J. Statsmodels: Econometric and Statistical Modeling with Python. *Proc of the 9th Python in Science Conf., Scipy* (2010), 57–61.
- [23] SUTSKEVER, I., VINYALS, O., AND LE, Q. V. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems (NIPS)* (2014), 3104–3112.
- [24] TASHMAN, L. J. Out-of-sample tests of forecasting accuracy: An analysis and review. *International Journal of Forecasting* 16, 4 (2000), 437–450.
- [25] WERBOS, P. J. Backpropagation Through Time: What It Does and How to Do It. *Proceedings of the IEEE* 78, 10 (1990), 1550–1560.
- [26] WILLIAMS, R. J., AND PENG, J. An Efficient Gradient-Based Algorithm for On-Line Training of Recurrent Network Trajectories. *Neural Computation* 2, 4 (1990), 490–501.

# Appendix A

## Hyperparameter Tuning

This appendix includes training and validation loss functions visualizations and summary results, for an LSTM model.

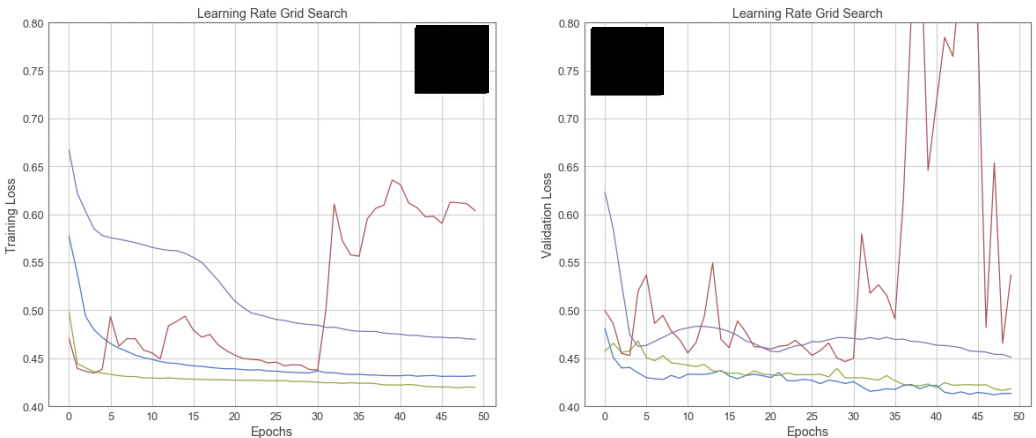
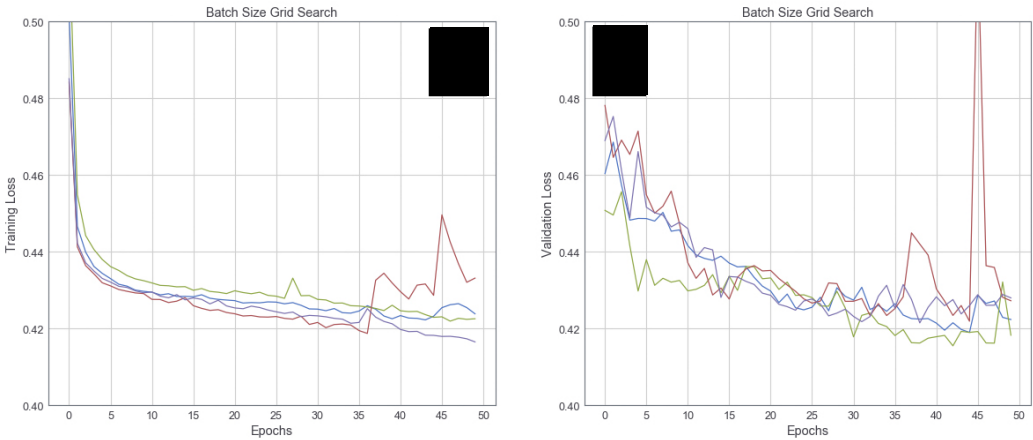


Figure A.1: Visualizing Loss Functions for Learning Rates

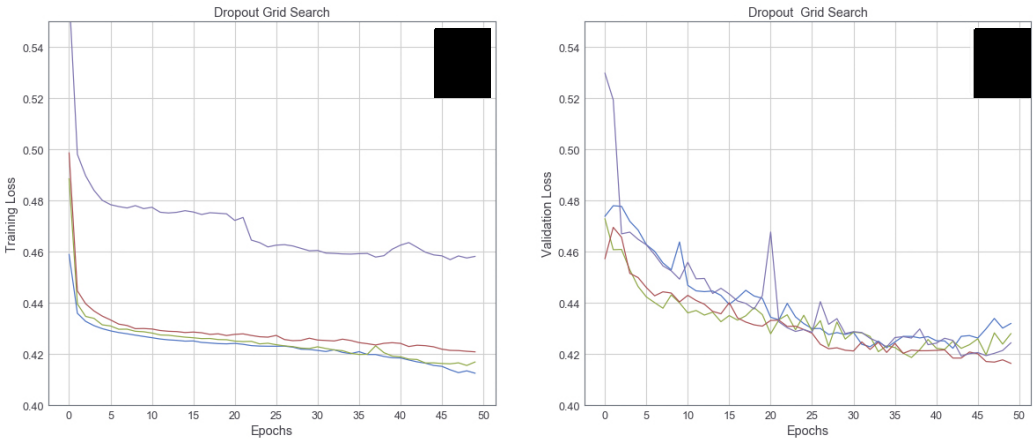
Learning Rate	Train Loss	Valid Loss
	0.438	0.447
	0.420	0.417
	<b>0.431</b>	<b>0.412</b>
	0.470	0.451

Table A.1: Results for Learning Rate evaluation



Batch Size	Training Time (hrs)	Train Loss	Valid Loss
████	7:31	0.429	0.422
██	4:47	0.422	0.421
████	3:35	0.423	0.419
████	3:05	<b>0.424</b>	<b>0.415</b>

Table A.2: Results for Batch Size evaluation



Dropout	Train Loss	Valid Loss
■	0.417	0.422
■	0.423	0.419
■	0.421	<b>0.416</b>
■	0.460	0.419

Table A.3: Results for Dropouts