CS

COC251

B612838

AIR POLLUTION FORCASTING USING DEEP LEARINNG

by

James Lovick

Supervisor: Qinggang Meng

Department of Computer Science
Loughborough University

April 2019

Abstract

Many densely urban areas across the world suffer from air pollution. Improving the quality of air around the world will involve inventive methods of forecasting air pollution, which could provide early warnings to areas with high population density. This project tests three neural architectures on stock market data to evaluate the performance on time series data. A sliding window approach is used to extract features and a window spacing is implemented to test how the performance changes when the distance between windows is increased in size. Evaluation of three different optimiser functions are used to see compare the performance between three architectures. A CNN-LSTM model is proposed to increase the performance of the standard LSTM neural network architectures. The CNN-LSTM is optimised using the simple, (μ,λ) and (μ+λ) evolutionary algorithms to evaluate which one performs best. The results show that the simple algorithm yielded better results than (μ,λ) and (μ+λ).

Contents

1 Introduction

1.1 Motivation

The quality of air across the world in major cities is linked to many aspects of human life such as health, economy and environment. Forecasting air pollution can help discover different measures needed to improve the quality of air around the world. Air pollution forecasting can also serve as an early warning system to prevent the levels becoming dangerously high, and in the long term protect the population. Each year an estimated 4.2 million premature deaths are linked to air pollution (WHO, 2018).
With diminishing air quality in cities such as Beijing, finding ways of more accurately predicting future pollution levels and finding pollutant correlations will allow countries to make policies to control the level of pollution better and quicker.
Deep learning methods are brilliant at finding complex patterns in large streams of time series data. With the cost of storage becoming more affordable, data sets have become far larger and the field of parallel computing has advanced significantly (Lipton, Berkowitz and Elkan, 2015). New machine learning frameworks, such as tensorflow, and pytorch developed by google and facebook respectively, have fuelled people to develop neural networks to predict time series data.

1.2 Project Aim

This project aims to compare different neural network architectures to find the strength of their predictive powers and then create a neural network that can process historical data to make accurate predictions on the levels of air pollution. Using the results from the neural network architectures, an analysis will be presented on the feasibility of using neural networks in the future for time series prediction.

1.2.1 Objectives

1 Research deep learning techniques and the background of air pollution forecasting.

2 Develop a combination of simple neural networks on a public time series dataset such as stock market prices.

3 Retrieve air pollution for pre-processing.

4 Modify Recurrent Neural Networks, RNN, for the air pollution data and train the software

5 Evaluate the results and apply deep learning optimisation techniques to gain better predictions for air pollution forecasting.

2 Literature Review

2.1 Air Pollution Forecasting

Bai *et al.* (2018) discusses different techniques of forecasting air pollution such as statistical models and Artificial Intelligence, AI, methods. Bai et al. concluded that AI performance was more optimal than that of statistical models but both cases show the importance of using Meteorological data to improve forecasting across all models.

Meteorological variables such as wind speed and ambient temperature can affect the dispersal of air pollution, so it is important to include these factors when making predictions about air pollution concentrations. Grivas and Chaloulakou (2006) Used a genetic algorithm to select which meteorological factors were used in their multi-layer perceptron model. The results were comparable to those which used all the input variables or slightly inferior when predicting the PM10 hourly concentrations, but the models benefitted with much shorter computational times.

2.2 Deep Learning

There has been an abundance of research in the field of deep learning and while researching Recurrent Neural Networks, RNN, was a monumental task, Lipton *et al.* (2015) understood this in their review of RNNs. The paper highlighted the increased accuracy of RNNs over earlier neural networks but had the problem of vanishing or exploding gradients. This was solved by Hochreiter and Schmidhuber (1997). They created the Long Short Term Memory, LSTM, cell that would replace the cells in the hidden layer of Normal RNNs, each memory cell contains a node with a recurrent edge of fixed weight, one to pass the gradient across many timesteps. This led to many more successful runs and would allow the network to learn much faster. Since then the LSTM has been optimised. Gers *et al.* (1999) introduced forget gates in the LSTM which allowed the LSTM to learn self-resets of memory contents that have become irrelevant solving the over saturation problem when given a continuous input stream of data. This has allowed LSTMs to achieve greater performance when dealing with the time series data that is used in air pollution forecasting.

Li *et al.* (2017) compared many different deep learning techniques for predicting air pollution concentrations. LSTM Neural networks were shown to outperform the other shallow models that were tested. They also found that large time lags permit an increased number in unrelated inputs which increased the models complexity.

As neural networks increase they generally have an exponential time complexity to train them. Huqqani *et al.* (2013) shows the performance of parallelizing a Backpropagation Neural Network on a GPU and a multithreaded CPU. They found that GPU based

parallelisation should be preferred if the input size and number of neurons is large. In contrast to this Ho *et al.* (2008) created a GPU-based Cellular Neural Network simulator that ran 8-17 times faster than the CPU-based CNN. This will allow for faster model training when using large data sets

2.3 Summary

Researching deep learning techniques has provided an insight into which methods are currently leading field. LSTMs are said to be the most accurate on large sets of time series data, coupling this with a genetic algorithm to select meteorological variables should help to decrease computation time while still withholding quality of predictions. Examination of genetic algorithms has also suggested that they could also be used to optimise hyperparameters of recurrent neural networks as the algorithms make no assumptions about how the parameters are used (Whitley, Starkweather and Bogart, 1990). Finally, model training times will be decreased greatly with the use of GPU parallelisation.

3 Design

3.1 Development

There are several programming languages that could be considered such as Java, Prolog and R, each with individual strengths and weaknesses. Java should be considered primarily as a choice for artificial intelligence which is deeply connected to search algorithms allowing development when optimising hyper-parameters. Prolog is a language commonly used in artificial intelligence as its logical notation is advantageous when used to create neural networks. In my experience, using Prolog, the syntax is compact but can become confusing when the program becomes larger and more complex. R is acknowledged to be one of the best languages used for manipulating statistical data, but the language can be complex as it was developed by statisticians to encompass their specific language. This project is written using Python because the syntaxis readable, and contains many machine learning libraries whilst also associated with a large community of programmers.

3.2 Data

Neural networks require large amounts of data for training and validation, so a suitable package was required to load the raw data before pre-processing. The Pandas package, used to load the data into the program, has several useful features for manipulating the data before it is fed into the neural networks. Functions such as “to_datetime” and “drop” permit proper indexing and selection of columns respectively, allowing pre-defined selection of the appropriate historical data. Scikit-learn was selected specifically for access to its extensive features, such as “MinMaxScalar”, which allowed efficient data analysis and manipulation for pre-processing data to reduce training times of artificial intelligence models.

3.2.1 Stock market data

For initial testing of different machine learning architectures, a public time series dataset was needed. The decision was taken to use the S&P 500 stock market index from the past 18 years as the data includes five different channels (open, close, high, low, volume) and has areas of small and large change. This provided a good benchmark on how well different neural network architectures can learn and how well they can predict while different conditions prevail.

3.2.2 Air pollution Data

For the final model, air pollution data provided by Berkeley Earth will be used. The dataset consists of eight features of air quality recorded in Beijing from 2010-2017. The features recorded include: air pressure, air temperature, cumulative rain hours, cumulative snow hours, dew point, wind direction, wind speed and 2.5 particulate matter. The response variable is the level of 2.5 particulate matter in the final model of the project. The data was gathered from several data sets and cleaned by Berkeley Earth before prior to this project.

3.3 Deep learning

In recent year and with on going development, neural networks have become more popular. Several deep learning libraries need to be considered when creating neural networks. Tensorflow is an obvious choice when choosing a machine learning library because there is a large community and resources for fast development of systems. The Tensorflow API allows for programs to be written in python. Tesnorflow also supports the Keras sequential API. Keras allows you to define your neural network layer by layer enabling you to develop architectures quickly and efficiently.
Three different neural network architectures were created using Keras and Tensorflow. This allowed a period of learning and time to discover the strength and weaknesses of each architecture before creating a final neural network to predict air pollution.

3.3.1 Feedforward Neural Network

One of the first neural network architectures created was the feedforward neural network. Its components were based on the neurons in the brain. A neuron in the brain works by firing an electrical signal only when signals from connecting neurons breach a threshold. The perceptron used in feedforward neural networks works in the same way.

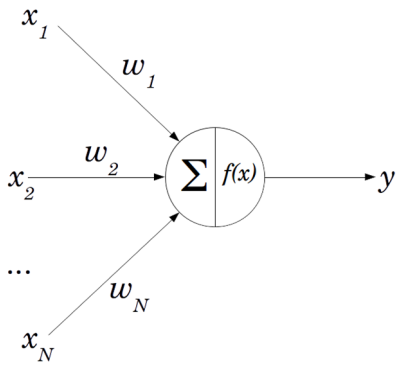

Figure 1 (Lucidarme, 2017)

Figure 1 shows the architecture of a perceptron. The perceptron takes the weighted sum of connected perceptron's, and given an activation function f(x), will output a number y. Feedforward neural networks will consist of an input layer, one or more hidden layers, and an output layer. The number of perceptron's in each layer will depend on the problem at hand. For time series prediction the output layer will have the same amount of perceptron's as the number of predicted steps you want the network to make.

3.3.2 Convolutional Neural Network

Convolutional Neural Networks, CNN, are classically used for image processing and classification. CNN's were inspired by how mammals perceive the visual world. The first CNN was created by LeCun *et al.* (1998), and was called LeNet5 which was able to classify hand-written digits. CNN's process data differently to regular Feedforward Neural Networks as the layers have 3 dimensions, height, width and depth and not all neurons will be connected to the next layer, but to a region of the next layer.

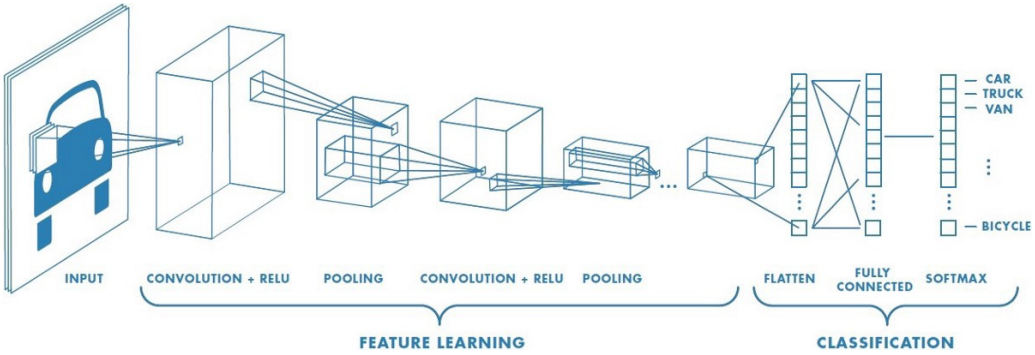

Figure 2 (Saha, 2018)

Figure 2 shows the architecture of a standard CNN. In image classification the convolutional layers run a kernel/filter over the input image, this is the way convolutional layers extract features such as edges from the image. The pooling layer comes directly after the convolutional layer which reduces the spatial size of the input. Finally, the data is flattened before being passed onto a set of fully connected layers to provide a classification of the image. In time series prediction the convolutional layer uses a one-dimensional kernel/filter to extract temporal features from the data. Tsantekidis *et al.* (2017) showed the strengths of using CNN's for time series forecasting over support vector machines.

3.3.3 Long Short-Term Memory

The recurrent cell used in Recurrent Neural Networks, RNN, is well suited for time series forecasting, in contrast to Feedforward perceptron's, the RNN cell has a connection to itself. Therefore, the network has two inputs, the present and the recent past. This gives the RNN

the advantage over other neural networks as it can learn patterns from the past that may affect future values. In time series prediction, knowing the recent past isn't enough. Long term dependencies need to be able to be created in order to be able to predict future values more accurately. Long Short Term Memory, LSTM, cells are a type of recurrent cell created by Hochreiter and Schmidhuber (1997)

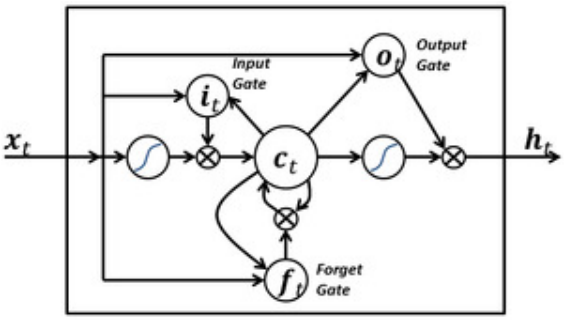

Figure 3 (Moawad, 2018)

Figure 2 shows the architecture of the LSTM cell. It consists of three different gates, input, output and forget gates. The forget gate decides which values it should forget based on the input x_t and the values in the cell state at c_{t-1}, this is stored in f_t. Next the input gate decides which values need to be updated in the cell state c_{t-1} stored in and stored in i_t. Now the cell state needs to be updated with the values that are going to be forgotten and the values that are added from the input transforming c_{t-1} into c_t. Finally, the output gate decides what values of the cell state are going to be output in h_t. The use of the forget gates gives LSTM cells an advantage over traditional RNN cells, as this can add or remove information from the cell state to create long term dependencies and remove dependencies no longer relevant to the current time step.

3.4 Hyperparameters

There are several parameters that need to be defined before training, some define how the data is pre-processed while others define how the network will optimise the weights between layers. Finding the most effective hyperparameter combination is difficult as the search space is so big. Many combinations of hyperparameters will be tested to a greater understanding of how the hyperparameters effect the performance of each neural network. During the data pre-processing phase, a sliding window approach will be used to extract features from the data. It is important to see how the different window sizes effect the performance and predictive powers of the networks.

An optimiser function is a function that changes the weights of the network during training to reduce the error of the predicted values. There are several different optimiser functions that can be used during training such as Adam and RMSProp. The Adam optimiser is commonly

used in time series prediction so it is important to test this against different optimisers to see how well it performs against other optimisers for this specific problem.

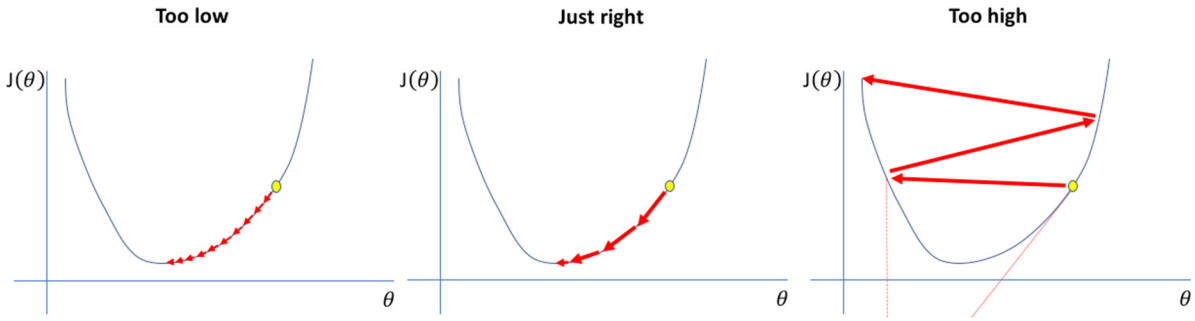

Figure 4 (Jordan, 2018)

Parameters such as number of epochs, batch size and learning rate effect neural networks in different ways and need to be chosen depending on the configuration of the other parameters. If the learning rate is too low, then the networks won't converge so the number of epochs will need to be increased to accommodate. If the learning rate is too high, then the weights of the networks will be changed too drastically, and the network will struggle to minimise the loss. Figure 4 shows a visualisation of how learning rate affects the convergence of the functions. The batch size hyperparameter tells the network when to update the weights of the network during each epoch. It has been observed in practice that when using a larger batch there is a significant degradation in the quality of the model, as measured by its ability to generalise (Shirish Keskar *et al.*, 2017).

3.4.1 Genetic Algorithm

Traditionally neural network architectures are defined ahead of time and optimised by trial and error. Unfortunately, more difficult problems generally require more hidden units in each layer which increases the search space for optimum architectures. Genetic Algorithms are based on the process of natural selection. The models with the best performance are selected for reproduction in the hope of the offspring yielding better performance. By selecting individuals that perform well, the search space is decreased. Genetic Algorithms work by specifying the population size, number of generations and the size of the genome for each individual. The parameters of each model are encoded into a binary array by assigning an index range for each parameter. This way the parameters can easily be decoded before being used to define the network architecture. After a model has been executed, a fitness score is returned. Running genetic algorithms increases the likelihood of high-performance solutions compared with traditional trial and error techniques. This project

will be using the Distributed Evolutionary Algorithms in Python (DEAP) to implement the genetic algorithm as this is a simple and straight forward package.

4 Plan

Based on the goals in section 1.2.1, a number of tasks were derived to split the project into manageable chunks, and a plan was created to help monitor the progress through the project. The literature review allowed evaluation of the current areas of deep learning that have been researched and where this project can provide insight in to the field of air pollution forecasting which also gave a starting point for many of the tasks throughout the project.

4.1 Gantt Chart

Figure 5 shows the Gantt chart created to show the time allocated to separate tasks in the project. Throughout the project, the Gantt chart was updated with new tasks and displayed which tasks that overran.

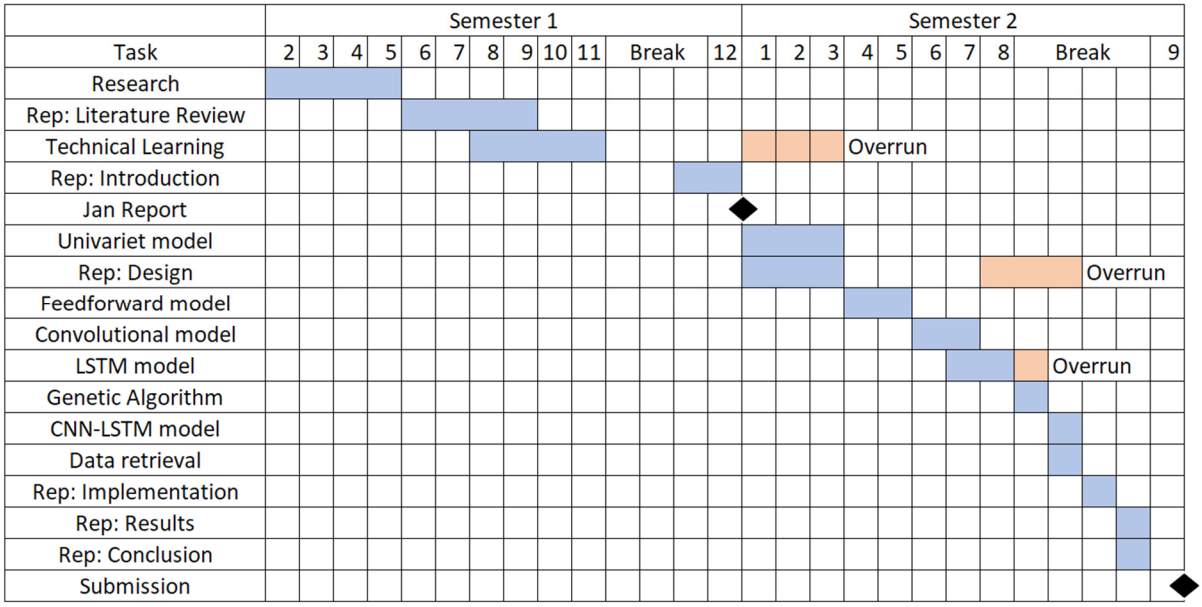

Figure 5

Task descriptions:

- Research: Preliminary research was very important at the beginning of the project. The importance of understanding what the fundamentals of neural networks were and how they worked.
- Literature Review: This was a summary of published accomplishments within the field of deep learning and indicated which direction the project could usefully examine over duration of the course.

- Technical Learning: This included learning how to pre-process data for neural networks and learning the fundamentals of Tensorflow and python to create some deep learning models to give an idea how the later models could be implemented in the project.
- Introduction/January report: Identified the motivation behind the project and the aims that could be achieved within the time restraints. A short report was submitted to my supervisor detailing how the project was progressing along with a decision on which software would be used and a plan to show the timetable of proposed aims within the time available.
- Create and compare models: Three models of different architectures are created and compared to show the different predictive powers on stock market time series data.
- Optimised Model: This task follows on from the previous task. A new model is to be created based on the results of the three architectures tested.
- Genetic Algorithm: Implement a genetic algorithm to optimise the hyperparameters of the new model.
- Data retrieval: Retrieve air pollution data from Berkeley earth.
- Implementation: Describing how the different models were created and implemented, and the challenges that were evident during implementation.
- Results: Tests and performance for the models will be evaluated to show what was achieved.
- Conclusion/Submission: A final overview of what has been achieved, what worked well and improvements that can be implemented in the future.

4.2 Software Engineering

The decision to use a rapid prototyping software engineering methodology allowed time to learn the software and rapidly start building prototypes early in the development process. These prototypes allowed a fundamental understanding behind building neural networks and their behaviour under differing conditions.
A Github repository was used for version control, because with so many prototypes being created within this project it became important to keep track of the different versions. This would eliminate mistakes from the prototypes being included in the final model.

4.3 Software Testing

During implementation several tests are used to make sure the program function correctly. Several functions were created and unit tests were performed to evaluate functionality.

Integration tests took place afterwards to assess the combination of functions as they work together. Integration tests are particularly important when using a rapid prototyping software engineering methodology, as regularly functions are taken from past prototypes that have been superseded.

5 Implementation

This section explains how the models were implemented using the stock market data how implementation of the optimised model and the genetic algorithm took place. During implementation several decisions were made which aided the creation of the final model.

5.1 Environment

Section 3 describes the packages chosen to create the models. Python and Tensorflow were selected for deep learning implementation. Pandas, Numpy and Scikit-learn are used for loading and pre-processing of the data. The DEAP package and Bitstring module were used for the genetic algorithm and Matplotlib was used for displaying results from the models. These packages were installed into a virtual environment using Anaconda which provided an easy to use command line interface to install these packages quickly and securely.

Anaconda provides several Integrated Development Environments, IDE's, to help with development such as Spyder and Visual Studio Code, however the decision was taken to use Jupyter Notebook. Jupyter Notebook especially useful for developing deep learning models as the interface allows the code to be split into sections which can be run independently to the rest of the code. This feature was vital when unit testing the pre-processing functions as it allowed examination of each function to ensure functionality prior to running the whole file.

As explained in section 4, a Github repository was used to help with version control and also served as a reliable backup should issues arise.

5.2 Pre-processing Stock Market Data

The stock market data used for the first three deep learning architectures was the S&P 500 data starting from the year 2000. The data was stored locally in a .CSV file so it could be easily loaded into the program using a Pandas data frame. The data was indexed using the date time function and was split into training and testing groups. Since the stock market data only contained approximately 4000 rows of data, the decision was taken to use a train/testing ratio of 90/10 respectively.

The data was normalised to the range of the activation functions used, to allow the models to converge faster and give more accurate results.

Traditionally a sliding window is used to break the input data into more manageable chunks for neural networks to learn from. Given a sequence (1,2,3,4,5), a sliding window of length 3 would split the sequence into a set [(1,2,3),(2,3,4),(3,4,5)] giving the network the knowledge

to predict that the next element in the sequence after a window (1,2,3) has a high probability of being 4.

The sliding window technique was used but extracted features from each window to input into the neural networks. Mean, Median and Interquartile range features were extracted from each window.

Date	Open	Mean	Median	IQR
03/01/2000	1469.25	1431.5	1399.42	55.8
04/01/2000	1455.22			
05/01/2000	1399.42			
06/01/2000	1402.11			
07/01/2000	1403.45			
10/01/2000	1441.47			
11/01/2000	1457.6			

Figure 6

Figure 6 is a visualisation of feature extraction. From the 4 rows selected by the window in the open column, the mean, median and interquartile range was calculated and held in their own list variables. Then the window slides down the open column again and another 3 values are calculated and inserted into the respective variables. This process is repeated until the sliding window reaches the end of the sequence. The resulting three feature sequences are horizontally stacked together in a new variable and the process is repeated for the remaining columns in the stock market dataset. With five columns in the data set, three features extracted per column, the resulting data set will have fifteen columns. To test how the models perform when the temporal spacing becomes larger, a method was implemented to change the distance between each window. By reducing the overlap of the windows, the size of the input data reduces and could potentially reduce execution time while maintaining model performance.

Neural networks require input data to be in 3 dimensional format of (batch_size, time_steps, features). Traditionally the window size would be the number of time steps, since the windows have been summarised into features of length 1 per window, the number of timesteps is 1.

5.3 Preliminary Models

As described in section 3, the three models which will be tested are the Feedforward Neural Network, the Convolutional Neural Network and the Long Short-Term Memory Neural Network. All three of these architectures were designed for different purposes but have all shown potential in time series prediction. Each architecture will have multiple hyperparameters will be tested to find the strength of prediction.

The optimiser functions which will be tested for each architecture are Adam, Adadelta and RMSProp Optimisers. Two activation functions, Relu and Tanh, will be tested with each

optimiser function as well. The loss function will stay constant throughout testing.
The performance of different sliding window sizes and how the performance changes when the gap between each window increases will also be tested.

5.3.1 Optimiser Function

The optimiser function in neural networks is an extremely important part of the program. The role is to reduce the loss function by changing the weights of the neurons in relation to the predefined learning rate of the model. A well-trained model should have a smooth loss curve that levels out towards zero.

RMSProp Optimiser

RMSProp is an optimiser function proposed by Hinton et al. (2014) in Lecture 6e of his Coursera class. The motivation behind creating RMSProp was to fix the problem of diminishing learning rates that the Adagrad optimiser has. The formula for RMSProp is shown in figure 7.

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

Figure 7 (Ruder, 2017)

E[g] is the moving average of the squared gradients. g_t is the gradient of the cost function with respect to the weight. n is the learning rate and β is the moving average parameter.

Adadelta Optimiser

Adadelta is an optimiser developed at a similar time to RMSProp and is an extension from Adagrad. Like RMSProp, Adadelta was developed to fix the diminishing learning rates that Adagrad has. The method dynamically adapts over time using only first order information and has minimal computational overhead beyond vanilla stochastic gradient descent (Zeiler, 2012). The comparison of Adadelta and RMSProp will provide insight into these product as they were both developed at a similar time to address the same problem.

Adam optimiser

The Adaptive Moment Estimation (Adam optimiser) is used consistently in the field of time series prediction. It was created to gain the benefits of RMSProp and Adagrad and used for creating adaptive learning rates for each parameter. In addition to storing an exponentially

decaying average of past squared gradients v_t like Adadelta and RMSProp, Adam also keeps an exponentially decaying average of past gradients m_t, similar to momentum (Ruder, 2017)

5.3.2 Model Architectures

Due to the time restraints of this project, the choice of architectures for the preliminary models needed to be kept simple as to save time on training time whilst also being able to show the behaviour of the hyper-parameters under testing.

Feedforward Model

The Feedforward Neural Network architecture consists of two fully connected layers, the first of which has a hidden size of 1500 neurons and the second has 750 neurons. The decision was taken to only use 2 layers as Feedforward Neural Networks overfit quickly and using more layers would be impractical. To receive the output of the network a final fully connected layer with 1 neuron is used, this allowed one-step ahead prediction on the stock market data.

CNN Model

The Convolutional neural network architecture consists of two 1 dimensional Convolutional layers of size 256 and 128 respectively, using a kernel size of 2 and 1 respectively. After each convolutional layer there is a max-pooling with a pool size of 2 and 1 respectively. After the final max-pooling layer, the output is flattened before being input into a fully connected layer with a hidden size of 50 followed by a final fully connected layer of size 1 to give one-step ahead prediction. Since convolutional layers are used for extracting features, the feature extraction phase used in the Feedforward and LSTM model will not be necessary when testing the CNN.

LSTM Model

The Long Short-Term Memory neural network architecture consists of 1 LSTM layer with a hidden size of 1000 neurons followed by a dropout layer. Drop out is a technique used to reduce over fitting by randomly dropping a percentage of neurons and there connections during training to stop the neurons co-adapting (Srivastava *et al.*, 2014). This technique was used to prevent overfitting. Finally, a fully connected layer of 1 neuron was used to give a one-step ahead prediction.

5.4 Air Pollution Model

Following the preliminary models, a final model was proposed to provide optimised performance on the air pollution data. To achieve this the combination of a CNN and LSTM models were used to predict the levels of air pollution in the Berkeley Earth data set.

5.4.1 CNN-LSTM Architecture

When creating a hybrid neural network, there are two ways of executing such a task. Either create and define each network individually and feed the input from one to the other, or use time distributed layers to define the network in one function. For simplicity this project uses the latter as performance would not be affected.
The CNN consists of a 1-dimensional convolutional layer with kernel size of 1, a max-pooling layer of kernel size 2 and a flatten layer before feeding into the LSTM layers. Each layer of the CNN is wrapped in a time distributed layer wrapper. This allowed the CNN to be applied to every temporal slice in the input data before being fed to the LSTM layers.
The LSTM consists of 2 LSTM layers using Relu activation, both with dropout of 20%, a fully-connected layer of size 50 and a final fully-connected layer of size 1 to give a one-step ahead prediction.

5.4.2 Air Pollution Data

The air pollution data consists of 8 columns with data recorded hourly from 2010-17. The data set consists of two data sets, one recorded from 2010-14 and another from 2015-17. Prior to receiving the cleaned data some changes were made to the wind direction feature. The wind direction recorded between 2010-14 was recorded by giving a general direction of NE, SE, CV, NW, where CV is equivalent to SW, and between 2015-17 the wind direction was recorded in degrees. This could be due to hardware upgrades and being able to more accurately measure the direction. To compensate, the degrees were converted into the general representation used between 2010-14 and then represented each category of direction as a number from 1-4. This allowed the neural network to process the wind direction.
All columns were normalised between 0-1 for faster convergence during training. The data was separated into windowed timesteps in the same way as the stock market data was for the preliminary models but without extracting features. The CNN layers were used to extract the features of the time series data before being fed into the LSTM. This should help reduce the run time which is vital when using a genetic algorithm.

Time distributed layers allow the CNN to process all the data before feeding it on to the LSTM, this means that the data is required to be in a 4-dimensional input shape. Before being input into the model the data is reshaped to match the format of (batch_size, subseq, timesteps_per_subseq, channels) where window_size = subseq * timesteps_per_subseq. This allows the CNN to extract the features and then the LSTM will the output back together before the final fully connected layer will give an output.

5.4.3 Genetic Algorithm

As described in section 3, the genetic algorithm is implemented using the DEAP to optimise the layers of the neural network. Each layer is represented by a pre-specified number of bits in an array and are decoded before training is executed. A gene length of 25 bits are used for the three parameters. 5 bits are used to define the size of the CNN, giving it a maximum size of 31 and 20 bits are used to define the 2 LSTM layers giving them a maximum size of 1023. This was important as making the search space of the genetic algorithm too large would require large execution times to find an optimal solution.
The algorithm has a starting population size of 10 and will run for 5 generations. The execution time for genetic algorithms change considerably depending on the generation and population number. Unfortunately, because it was not practical to have both parameters large, a decision was made to have a high number of generations or a high population count. Vrajitoru, (2000) tested this in relation to information retrieval from a document collection and concluded that a higher population had significantly higher chance of extracting information. For this reason, the population is double the number of generations.
There are four key evolutionary tools used in a genetic algorithm: mate, mutate, select and evaluate. Each tool is registered before execution and they define how the population will be changed between generations.
For mating I used one-point crossover. Figure 8 gives a visualisation of one-point crossover.

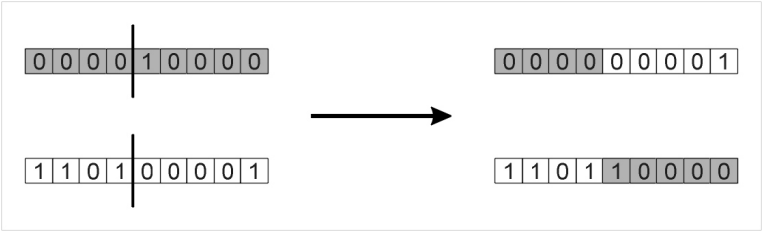

Figure 8 (Eiben and Smith, 2015)

In one-point crossover the two parent genomes are split at the same index in the array. Two children are output from the function with their genomes made up of a combination of the parents using the crossover point as a reference.

The mutation tool implemented was the flip bit function. This function is ideal for this problem as the genome is represented as a binary array. The function takes an individual and mutates the genome by flipping the bits based on a predefined probability, thus ensuring some variety in the population.
For selecting individuals for the next generation the tournament tool is used. This tool selects a predefined number of the population to participate in a “tournament” where the individual with the best fitness attribute in the tournament is selected for the next generation. This way the best individual has a high chance to make it to the final population.
Finally the evaluation tool to evaluate a single individual in the population. This is where the algorithm calls the function that builds and runs the model, taking the current individuals’ genome as input and returns the fitness score of the individual to the genetic algorithm. The RMSE of the predicted graph for each individual is calculated as the fitness score. In order for the genetic algorithm to contrast between individuals, the RMSE is calculated after the predictions have been un-normalised to penalise poor performance.

5.4.4 Evolutionary Algorithms

The DEAP library offers several algorithms used for evaluating and evolving the population. The three main algorithms used in the DEAP package are simple, (μ,λ) and (μ+λ). (μ,λ) represents that an evolutionary algorithm will generate λ offspring from μ parents and selects the μ best individuals from only from λ offspring. Similarly (μ+λ) represents that an evolutionary algorithm will generate λ offspring from μ parents and selects the μ best individuals from the μ+λ individuals in total (Fogel, David B, 2009). Probabilistic measures for crossover and mutation are also given. This differs from the simple algorithm as there is no definition to how many individuals will be passed on to the next generation, only probabilistic measures for crossover and mutation are given. All three algorithms will be tested to compare thier performance using the tools specified in section 5.4.3.

5.5 Testing Methodology

Preliminary Models

To evaluate the three models, (Feedforward, CNN, LSTM) performance metrics of the models will be recorded for a series of window sizes and spaces between each window. Performance metrics will be recorded when testing how the model performs using three different optimiser functions and two different activation functions. Because the stock market data is approximately 4000 rows long, window spacing of 1, 2 and 4 will be tested. Any

higher and the input data is reduced to below 1000 rows. The window sizes that will be testing are 1, 2, 3, 8, 9, 10, 15, 16, 17, 25, 26, 27, 45, 46 and 47. This should provide insight into how the performance changes as the window size is increased. Using the optimal configuration of window size and spacing parameters, the three optimiser functions to be tested are: Adam, Adadelta and RMSProp, with two activation functions: Relu and Tanh. This should provide information to which optimiser function is optimal using each activation function. The performance metrics to be recorded are, Mean Absolute Error, MAE, and Root Mean Squared Error, RMSE. MAE measures the magnitude of error in the predictions made compared with the ground truth values and is expressed in the formula below.

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n} |y_j - \hat{y}_j|$$

RMSE is similar to MAE in that they are both used to measure the average magnitude of error, but by taking the square root RMSE has a high weight toward large errors. When evaluating the fitness of individuals, penalising large errors will allow for a greater contrast between the fit and unfit individuals in the population. This allows the genetic algorithm to converge on an optimised solution. The RMSE formula is expressed below.

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(yj - \hat{y}_j)}$$

Air Pollution Model

Testing the evolutionary algorithm to optimise the CNN-LSTM for air pollution forecasting, the crossover, mutation and selection functions were kept constant along with the probabilistic values for crossover and mutation. The simple evolutionary algorithm will be executed with a starting population of ten and will run for six generations. For (μ,λ) and (μ+λ) evolutionary algorithms, a starting population μ of size 10 and λ of size 10 will be used. Because of time constraints it is impractical to larger population sizes as these algorithms will run for over 10 hours optimising the CNN-LSTM model on the air pollution data set.

6 Results

In this section, results collected from the preliminary tests and the genetic algorithms will be displayed giving a summary of the graphs and tables provided.

6.1 Preliminary models

6.1.1 Feedforward Neural Network

Window Size and Spacing Results

When testing the window size and window spacing parameters, the Feedforward model ran for 15 epochs using a batch size of 32. For optimisation an Adam optimiser was used with a learning rate of 0.0001

Table 1: Window spacing 1

Window size	Mean Absolute Error	RMSE
1	0.02613	0.03539774
2	0.02601	0.034554305
3	0.02698	0.035665109
8	0.02947	0.038144462
9	0.02984	0.037107951
10	0.03005	0.039496835
15	0.03402	0.042941821
16	0.0385	0.050408333
17	0.0401	0.052469038
25	0.05451	0.069584481
26	0.05046	0.065199693
27	0.04815	0.062960305
45	0.08917	0.102810505
46	0.09535	0.109316056
47	0.09049	0.104019229

The table above shows the results from the feed forward neural network using a window spacing of 1. From these results, the best result with the lowest RMSE metric has a window size of 2. Generally, as the window increases with size so does the MAE and the RMSE.

Tables 2: Window Spacing 2

Window size	Mean Absolute Error	RMSE
1	0.03894	0.051439285
2	0.03877	0.050921508
3	0.03932	0.051516987
8	0.03873	0.050783856
9	0.04519	0.057680153
10	0.03923	0.051312766
15	0.04154	0.053516353
16	0.03523	0.046669048
17	0.03834	0.050059964
25	0.04117	0.053009433
26	0.039	0.050556899
27	0.04295	0.054881691
45	0.03506	0.045639895
46	0.03608	0.046647615
47	0.03848	0.049416596

When testing window spacing 2, the learning rate was decreased to 0.00001 as the models were overfitting and the predictions were becoming extremely noisy.

The table above shows the results of using a window spacing of 2. Interestingly the model preferred a window size much larger when the spacing was increased to 2 as the best window size is 45. There trend is limited when increasing the window size as the RMSE hovers around 0.05 except for window size of 9 that has an RMSE of 0.57 and window size 16 which has RMSE score of 0.047.

Table 3: Window Spacing 4

Window size	Mean Absolute Error	RMSE
1	0.05841	0.072904047
2	0.0602	0.07515983
3	0.05934	0.074013512
8	0.05841	0.072443081
9	0.06079	0.075013332
10	0.06927	0.084255564
15	0.05977	0.073661387
16	0.06726	0.081774079
17	0.06377	0.077833155
25	0.07578	0.090343788
26	0.06416	0.078051265
27	0.05746	0.070710678
45	0.05971	0.073986485
46	0.06578	0.080467385
47	0.06272	0.077226938

The table above shows the results of using a window spacing of 4. The best RMSE score

was achieved using a window size of 27. However, no trend is present when the window spacing is increased this far for the Feedforward model.

Optimiser and Activation Function Results

The best results from the window spacing and size tests yielded a window size of 2 and a spacing of 1. This configuration will be used to test the optimiser and activation functions. To get the maximum potential out of each optimiser, the learning rate had to be changed to prevent over and underfitting. Adam and Adadelta performed best using a learning rate of 0.001 while RMSProp required a learning rate of 0.000001. Adadelta was prone to underfitting and required a larger learning rate whereas RMSProp was prone to overfitting and required a much lower learning rate.

Table 4: Optimisation Results

Optimiser	Activation	Mean Absolute Error	RMSE
Adam	Relu	0.02601	0.034554305
Adam	Tanh	0.06314	0.085445889
Adadelta	Relu	0.0518	0.06074537
Adadelta	Tanh	0.06746	0.088605869
RMSProp	Relu	0.03481	0.04365776
RMSProp	Tanh	0.05409	0.07330075

The table above shows the results using the three optimiser functions. In every case Tanh activation was the worst scoring better in combination with RMSProp. Relu activation scored best overall with the Adam optimiser. Adadelta has a higher RMSE than RMSProp in both cases.

Prediction Performance

Using the overall best configuration, the Feedforward model was trained and used to predict the test data. Figure 9 visualises how the model performed. The model performs quite well capturing the trend of the data but introduces peaks when the change between time steps is small and is unable predict large peaks very accurately.

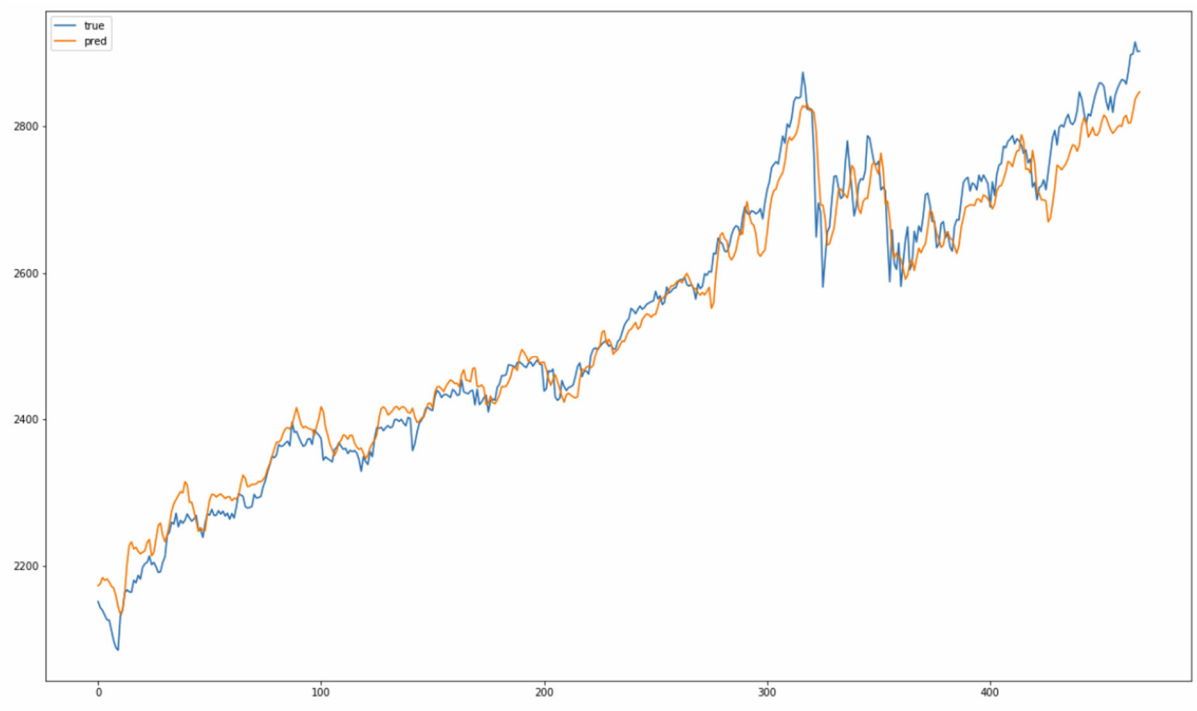

Figure 9

6.1.2 Convolutional Neural Network

Window Size and Spacing Results

When testing the window size and window spacing parameters, the CNN model ran for 15 epochs using a batch size of 32. For optimisation an Adam optimiser was used with a learning rate of 0.0001.

When training the CNN, a kernel size of 2 was used to extract features, this limitted testing window sizes of 1 and 2 which were not possible as the kernel had to move over the window to extract information. In consideration of this, window sizes of 4 and 5 were evaluated instead.

Table 5: Window Spacing 1

Window size	Mean Absolute Error	RMSE
3	0.03095	0.039899875
4	0.02585	0.036400549
5	0.02612	0.037603191
8	0.03022	0.042813549
9	0.02969	0.039560081
10	0.03349	0.044643029
15	0.0323	0.043714986
16	0.03703	0.048373546
17	0.03036	0.040595566
25	0.03323	0.046043458
26	0.0349	0.048414874
27	0.03522	0.047381431
45	0.05765	0.078409183
46	0.04094	0.061741396
47	0.05709	0.07551821

The table above shows the results collected using window spacing 1. The best performance was yielded when using a window size of 4. Generally, the larger the window size, the greater RMSE becomes. Interestingly, there isn't much change between (15,16,17) and (25,26,27). This could indicate that there is a threshold where the performance drop levels out before decreasing rapidly when the window size exceeds 45.

Table 6: Window Spacing 2

Window size	Mean Absolute Error	RMSE
3	0.07687	0.087281155
4	0.07734	0.087721149
5	0.06515	0.075053314
8	0.07035	0.080901174
9	0.06514	0.076243032
10	0.06759	0.077498387
15	0.05512	0.06556676
16	0.05822	0.06942622
17	0.6665	0.076713754
25	0.06508	0.076765878
26	0.05889	0.068330081
27	0.05788	0.077736735
45	0.05015	0.064660653
46	0.05851	0.071260087
47	0.05748	0.070469852

While using a window spacing of 2, the performance has greatly diminished with RMSE scores doubling with a low window size compared with window spacing 1. Interestingly, larger window sizes seem unaffected by the larger window spacing. This will be because the

ratio between window spacing and window size is much larger for smaller windows meaning that spatial temporal information will be lost.

Table 7: Window Spacing 4

Window size	Mean Absolute Error	RMSE
3	0.08054	0.090796476
4	0.08518	0.096555683
5	0.08483	0.096083297
8	0.0667	0.078044859
9	0.07956	0.089654894
10	0.05909	0.072958893
15	0.04794	0.067223508
16	0.2247	0.235053185
17	0.03916	0.04936598
25	0.04089	0.052592775
26	0.2361	0.245987805
27	0.04636	0.066895441
45	0.0557	0.069404611
46	0.2713	0.280695565
47	0.06512	0.080541915

It was clear from the results of window spacing 2 that the CNN does not perform well when the window spacing is increased. Increasing window spacing even further has produced even worse results than previously. In some cases, the model is clearly struggling to learn the features of the data with window sizes of 16, 26 and 46 having an RMSE metric spiking to around 0.25. This could mean the model is overfitting and the loss is spiking high every other epoch.

Optimiser and Activation Function Results

When testing the performance of the optimiser and activation functions, a window size of 4 and a window spacing of 1 is used. These are the dominant values when pre-processing the data. To get the best performance from each optimiser function a learning rate of 0.0001 was used for Adam and RMSProp. Adadelta required a learning rate of 0.01 and required the model train for 50 epochs instead of 15. This was because using a learning rate of 0.1 the model was overfitting, but with 0.01 the model was underfitting.

Table 8: Optimiser Results

Optimiser	Activation	Mean Absolute Error	RMSE
Adam	Relu	0.02585	0.036400549
Adam	Tanh	0.1313	0.162265215
Adadelta	Relu	0.0832	0.096010416
Adadelta	Tanh	0.1353	0.159122594
RMSProp	Relu	0.06999	0.08014986
RMSProp	Tanh	0.2049	0.241143111

The results show that for every optimiser Relu activation performs better than Tanh. Adam optimiser performs the best by quite a margin using Relu. Interestingly, Adadelta performed better than the other two optimiser when using Tanh but worst when using Relu.

Prediction Performance

Using the optimised configuration of hyper-parameters, Figure 10 shows the prediction graph created after the CNN was trained. The graph shows that the CNN is good at extracting features from the data but the accuracy of the predictions were very low. The model overestimates changes when the change between timesteps is low demonstrated at approximately timestep 50, and underestimates changes when the change between time steps is high demonstrated at approximately timestep 310

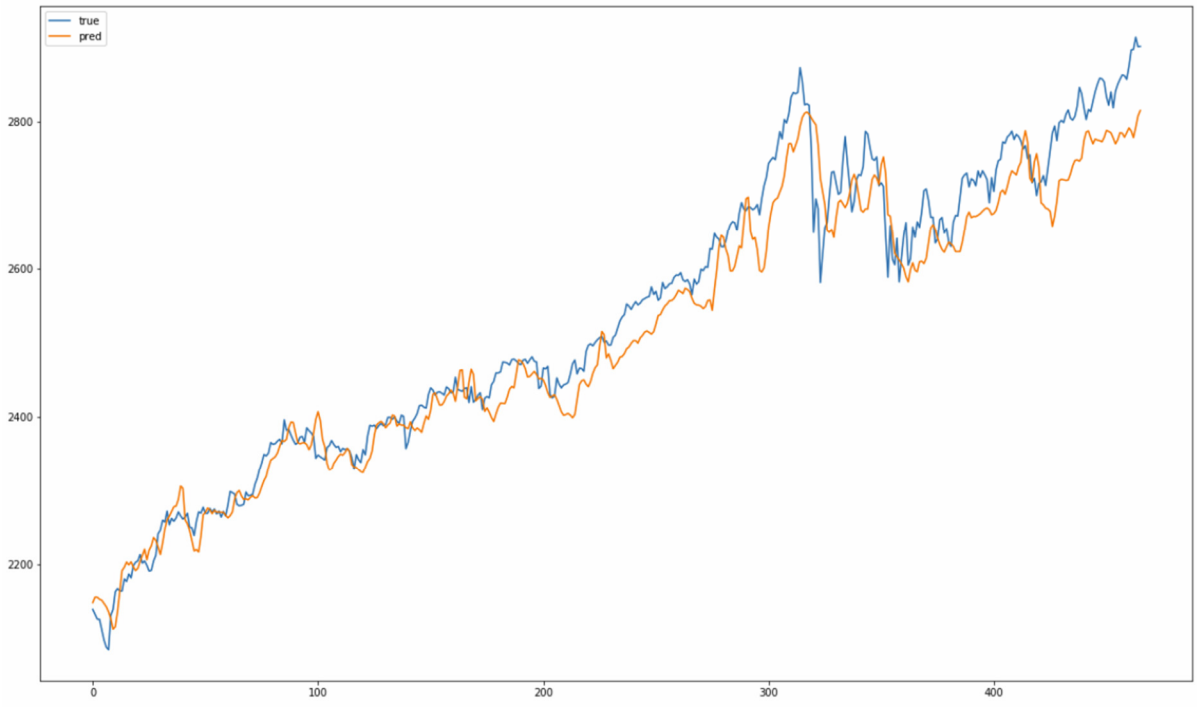

Figure 10

6.1.3 LSTM Neural Network

Window Size and Spacing Results

When testing the window size and window spacing parameters, the CNN model will run for 15 epochs using a batch size of 32. For optimisation an Adam optimiser will be used with a learning rate of 0.0001.

Table 9: Window Spacing 1

Window size	Mean Absolute Error	RMSE
1	0.02635	0.036345564
2	0.02711	0.037336309
3	0.02557	0.03468429
8	0.0256	0.034161382
9	0.02575	0.034322005
10	0.02638	0.034669872
15	0.02612	0.034205263
16	0.02625	0.034307434
17	0.02584	0.034205263
25	0.02775	0.036249138
26	0.02806	0.03671512
27	0.02778	0.036701499
45	0.03172	0.040024992
46	0.03113	0.039268308
47	0.03248	0.040607881

The results collected above show the performance of the LSTM with window spacing 1. The performance of the LSTM is not affected greatly when the window size changes between 3 and 17. However, as the window size increases after point 17 there is a decrease in performance. Table 9 shows indicates the best performing window size is 8.

Table 10: Window Spacing 2

Window size	Mean Absolute Error	RMSE
1	0.04167	0.055208695
2	0.03859	0.051332251
3	0.03915	0.052220686
8	0.03756	0.050318983
9	0.03781	0.050596443
10	0.03687	0.049487372
15	0.03566	0.047780749
16	0.03523	0.047381431
17	0.03505	0.047159304
25	0.03312	0.044474712
26	0.03277	0.043817805
27	0.0328	0.043817805
45	0.03233	0.041916584
46	0.03241	0.041833001
47	0.03237	0.041964271

When the window spacing is increased to 2, the performance of the smaller window sizes decreases but recovers when the window size reaches 47. This is a similar characteristic to the CNN. When the window spacing increases, temporal information is lost. This will affect the efficiency for LSTM's as they rely on temporal information to make long term dependencies.

Table:11 Window Spacing 4

Window size	Mean Absolute Error	RMSE
1	0.06858	0.08357033
2	0.06871	0.083767535
3	0.07253	0.087664132
8	0.07735	0.092016303
9	0.07633	0.09087904
10	0.07675	0.091350972
15	0.07935	0.094138196
16	0.07915	0.093584187
17	0.07678	0.091098847
25	0.07601	0.089983332
26	0.07648	0.09064767
27	0.07281	0.086873471
45	0.08113	0.096228894
46	0.08326	0.098397154
47	0.08173	0.09698969

Finally, when the window spacing reaches a size of 4, so much temporal information is lost that not even the larger window sizes can recover performance. Unlike the CNN, none of the tests returned any spikes in the RMSE indicating that LSTMs are more robust against overfitting.

Optimiser and Activation Function Results

When testing the optimisers and activation functions, a window size of 8 and window spacing of 1 is used. These values gave the best performance in the previous test.
To gain the best performance from each optimiser function a learning rate of 0.0001 was used for Adam and RMSProp. Adadelta required a learning rate of 0.01 and required the model train for 50 epochs instead of 15. This was because using a learning rate of 0.1 the model was overfitting, but with 0.01 the model was underfitting.

Table 12: Optimiser results

Optimiser	Activation	Mean Absolute Error	RMSE
Adam	Relu	0.0256	0.034161382
Adam	Tanh	0.05444	0.071154761
Adadelta	Relu	0.1512	0.157765649
Adadelta	Tanh	0.05175	0.06860758
RMSProp	Relu	0.02471	0.033075671
RMSProp	Tanh	0.004314	0.221855809

The best performing optimiser was RMSProp using Relu activation. In every case Tanh performed worse apart from Adadelta, where Relu gave the worst performance. RMSProp gave particularly bad results when using Tanh.

Prediction Performance

Using the best configuration of hyper-parameters, the model was trained to plot the graph of predictions. Figure 11 shows the graph of predictions created by the LSTM model. The graph gives a good prediction line following the trend of the ground truth values. Compared to the Feedforward Neural Network and the CNN, the predicted line is very smooth with little noise. This is facilitates the reduction in the number of false peaks which the CNN was especially susceptible to creating. However the LSTM fails to predict noisy data shown at timestep 350, the predicted values are unable to follow such a noisy line. This could be due to the feature extraction phase applied to the data before training.

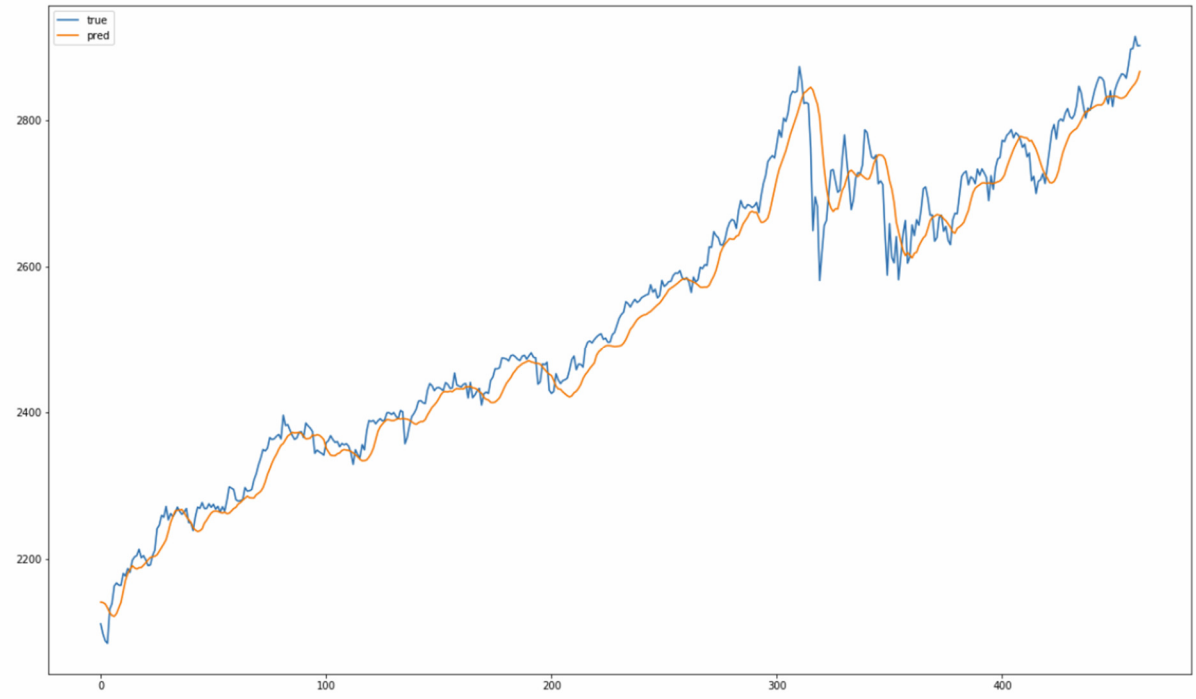

Figure 11

6.2 Genetic Algorithms

Following the results of the preliminary models a CNN-LSTM hybrid model is proposed. When testing both the CNN and LSTM separately, the results show that the LSTM suffers from the feature extraction phase prior to training while the CNN can produce acceptable results by extracting features on the raw data during training. Therefore, a final model is proposed to improve upon the performance of the models. The architecture of this model is detailed in section 5.4.1. The model will use an Adam optimiser and Relu activation function. Even though the LSTM performed best with an RMSProp optimiser, Adam will be used as the CNN had decreased performance with RMSProp while the performance of the LSTM using Adam is still competitive. In order to get the best performance from the CNN a window size of 4 will be used when optimising the CNN-LSTM. Using this architecture, three evolutionary algorithms in the DEAP package will be tested to see how each one performs when optimising the neural network layers for the air pollution data set.

Simple Evolutionary Algorithm

The simple evolutionary algorithm ran for 6 generations with a starting population of 10. Statistics were recorded after each generation and are displayed in the table below.

Table 13: Simple evolution statistics

Gen	Num_evals	Average	Standard Deviation	Min	Max
0	10	23.0373	8.56087	17.0284	47.4544
1	6	19.8708	4.52226	16.6595	32.2598
2	9	19.342	3.66622	16.4469	29.5762
3	6	20.2567	6.75721	16.4469	40.2061
4	9	19.8056	2.77348	16.4005	25.9706
5	10	117.617	294.136	15.9775	1000

The results show that the algorithm was successful with optimising the hyperparameters across each generation. While the optimal solution was constant across generation 2 and 3, the algorithm successfully selected the optimal individual for the next generation, therefore retaining the most optimal solution across to the next generation. In the last generation the maximum fitness is 1000, because one individual in the population had an invalid genome. To penalise invalid genomes, an “if” clause was implemented to give any individual that contained a parameter equal to zero a fitness score of 1000.

The optimal neural network with an RMSE of 15.9775 returned was:

Conv Layer	20
LSTM Layer 1	843
LSTM Layer 2	74

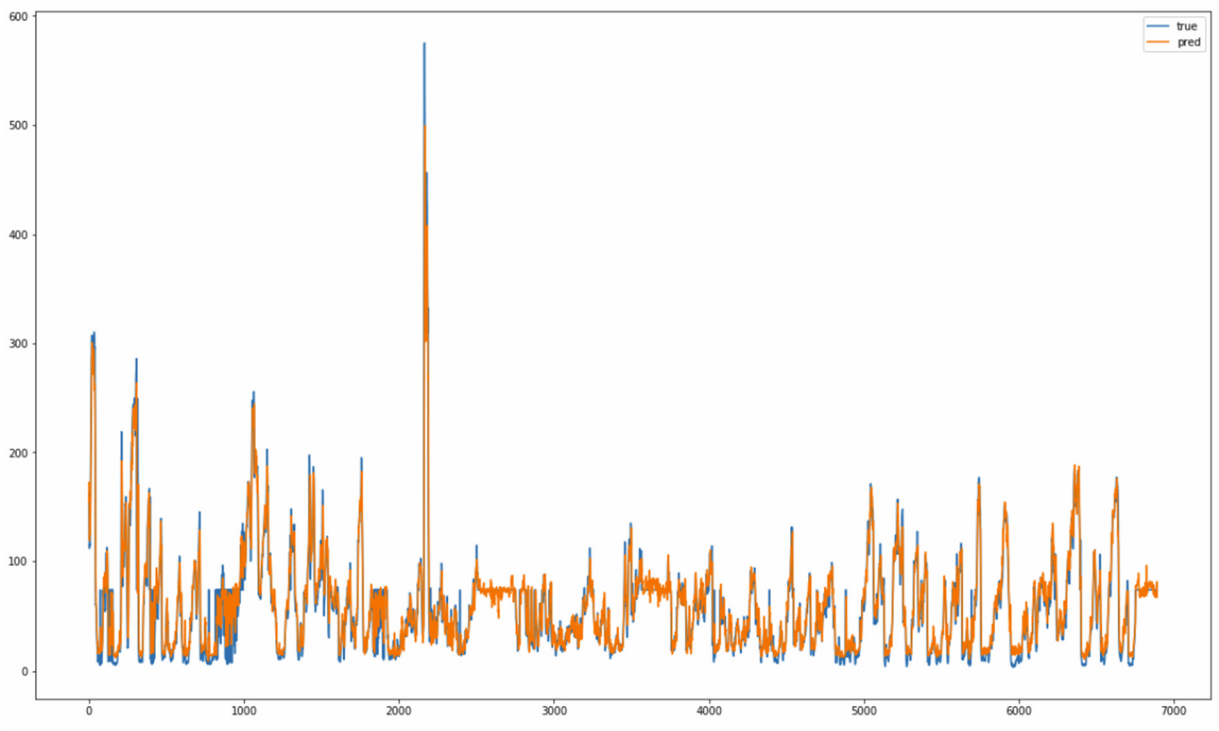

Figure 12

Figure 12 shows the prediction graph of the optimised neural network returned from the Simple Evolutionary Algorithm. The graph shows that the optimised model performs well and manages to capture the shape of the peaks. While the model doesn't capture the full peak, the predicted values manage to follow the noise of the ground truth values. However, the models accuracy struggles at approximately timestep 900.

(μ,λ) Evolutionary Algorithm

The (μ,λ) algorithm ran for 6 generations with a starting population of μ = 10 and λ = 10. The table below contains the statistics of each generation created by the algorithm.

Table 14: (μ,λ) statistics

Gen	Num_evals	Average	Standard Deviation	Min	Max
0	10	24.4829	12.4379	16.4068	60.7141
1	7	17.8129	1.17192	16.1709	20.1645
2	8	16.9806	1.26391	16.1709	19.5415
3	6	18.0814	1.72364	16.1709	22.3403
4	6	16.9029	1.1369	16.1709	19.6707
5	6	16.5858	0.975952	16.1709	19.4701

The results show that the algorithm was successful in in optimising the population to yield an optimal model with RMSE of 16.1709. While the algorithm was unable to find a further optimised solution after generation 1, it was successful in retaining the optimised solution in the population to the end. The results do show that the average RMSE did decrease which

implies optimisations were occurring between generations, however no further optimisation was found.

The optimal neural network with an RMSE of 16.1709 returned was:

Conv Layer	14
LSTM Layer 1	240
LSTM Layer 2	993

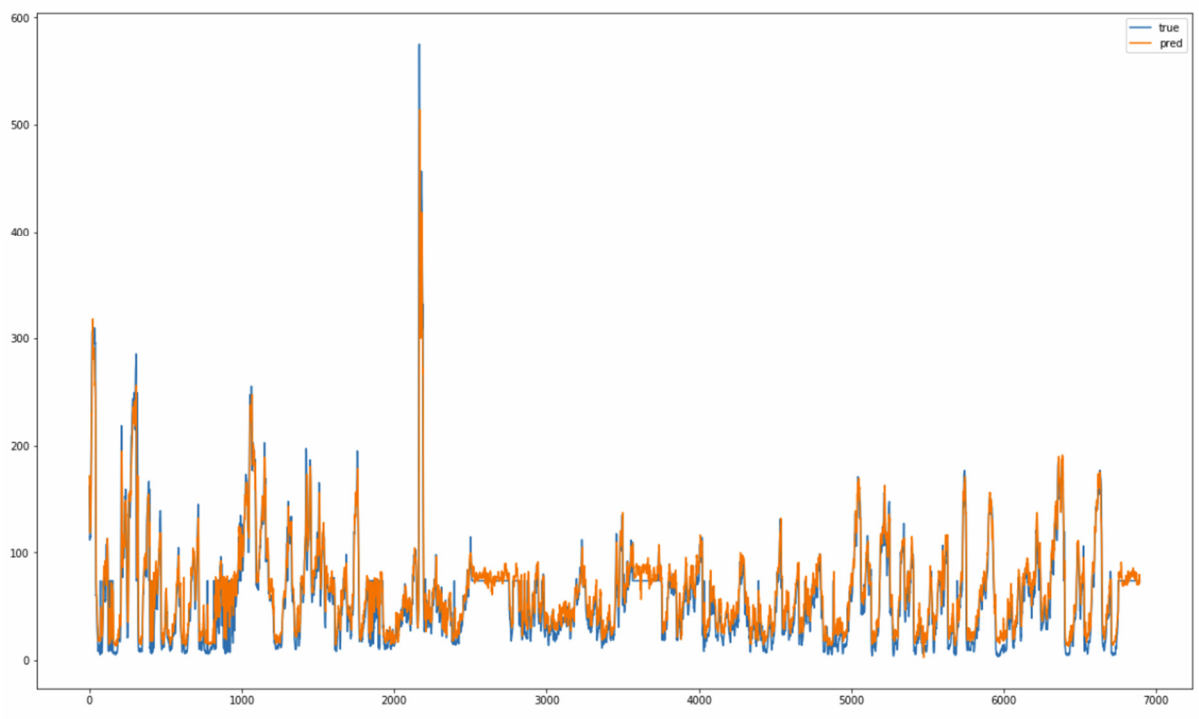

Figure 13

Figure 13 shows the prediction graph of the optimised model returned by (μ,λ) evolutionary algorithm. The model manages to follow the air pollution ground truth values well.

(μ+λ) Evolutionary Algorithm

The (μ+λ) algorithm ran for 6 generations with a starting population of μ = 10 and λ = 10. The table below contains the statistics of each generation created by the algorithm.

Table 15: (μ+λ) statistics

Gen	Num_evals	Average	Standard Deviation	Min	Max
0	10	19.8049	2.11773	17.7382	24.1272
1	7	17.794	0.448106	17.2562	18.7636
2	8	17.7698	0.504892	17.2447	18.7636
3	8	17.5266	0.426628	17.0583	18.3193
4	8	17.1871	0.216723	16.6263	17.505
5	4	17.2447	3.55E-15	17.2447	17.2447

The results indicate that the algorithm did manage to optimise the hyperparameters, however the optimal solution returned was not the optimal solution evaluated. The algorithm managed to optimise the hyperparameters until the last generation where the entire population turned into the optimal solution from generation 2. This is troubling behaviour as the optimal solution was lost and all the returned population was the same.
The optimal neural network with an RMSE of 17.2447 returned was:

Conv Layer	14
LSTM Layer 1	145
LSTM Layer 2	495

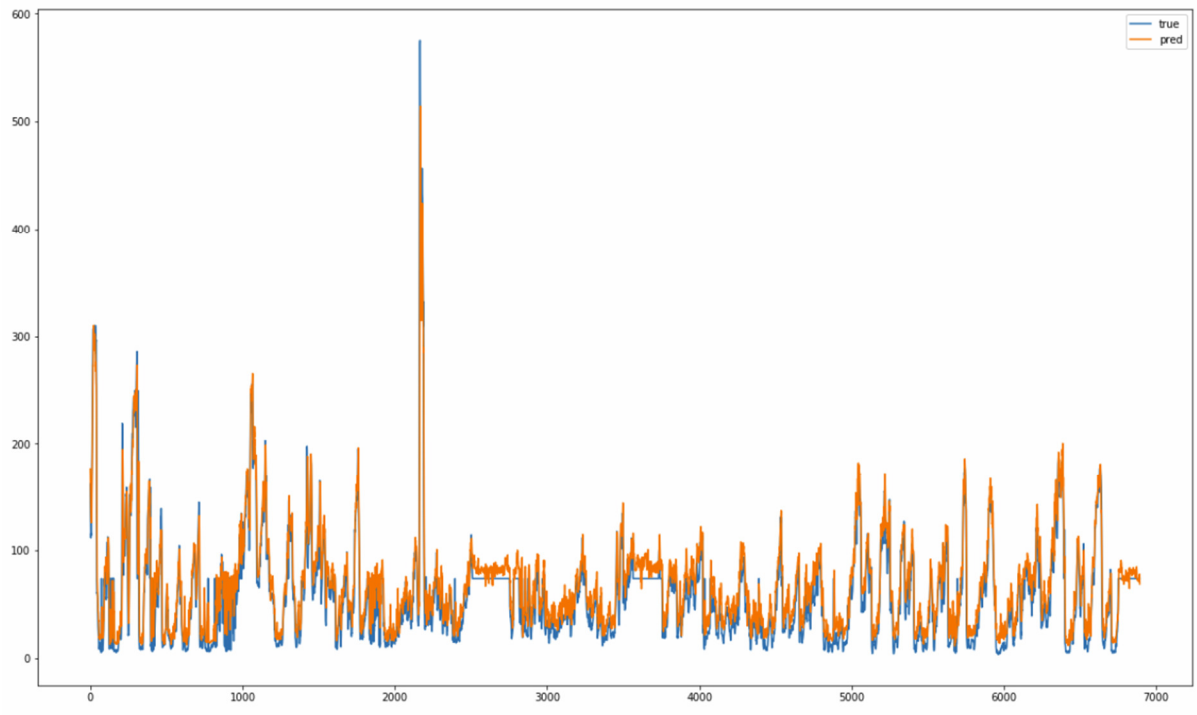

Figure 14

Figure 14 shows the prediction graph of the optimised solution returned from the (μ+λ) evolutionary algorithm. The model performs well and manages to capture the path of the ground truth values of the air pollution data

7 Conclusion

To conclude, the results from the preliminary models indicate that neural networks require the sliding windows to be relatively small to allow a good performance. Neural networks also require the sliding windows to contain a large overlap. However, it was vital to test whether increasing the window spacing could reduce training times while maintaining performance. The results indicate that the larger window spacing rapidly decreases performance. Therefore, larger window spaces is not viable solution when attempting to decrease the training times of genetic algorithms.

In all three of the preliminary models the Relu activation was superior to the Tanh activation function and in the cases of the CNN and Feedforward Neural Network, the Adam optimiser was superior to Adagrad and RMSProp. The LSTM yielded better results using RMSProp over Adam optimiser, but the performance of both optimisers was generally similar.

The Feedforward Neural Network produced a good prediction graph but accuracy suffered when the change between timesteps was low as the model attempted to predict peaks and troughs that was not present in the ground truth values. The CNN yielded a similar prediction graph to the Feedforward model, but accuracy suffered more when the change between timesteps was also small. This resulted in the CNN introducing “noise” which was not present in the ground truth values. The LSTM produced the smoothest prediction graph. Whilst the predictions were not overly accurate, there is some benefit to a smooth prediction line as the model will catch the overall trend of the data.

The results indicate that the Simple Evolutionary algorithm manages to optimise the hyperparameters at each generation, returning an optimised model that could predict the levels of the 2.5 particulate matter well. Whilst (μ,λ) evolutionary algorithm did not manage to further the optimisation every generation, the algorithm did retain the optimised solution through 5 generations. This optimised model was comparable to that of the simple evolutionary algorithm while containing different hyperparameters. However, the (μ+λ) evolutionary algorithm yielded poor results as it was unable to retain the optimal solution to the end generation and returned a population that was identical i.e no variety. Better results from the genetic algorithm would require increasing the population and number of generations, this would allow for more of the search space to be explored but would increase the amount of time to search.

The CNN-LSTM architecture worked well in predicting the levels of 2.5 particulate matter, using a CNN layer to extract features before passing them to the LSTM saved execution time as the features do not need to be calculated while pre-processing the data and creates a more accurate model.

Further research in this area could include implementing multi step forecasting for air pollution to create predictions for 24 hours as opposed to 1 hour. Using deep learning and genetic algorithms for air pollution forecasting, needs to continue as the results show the potential genetic algorithms have for optimisation. More complex models will be needed to produce multi step forecasting, however this will increase the search space for optimisation. For introducing multistep predictions more meteorological features need to be collected for training the networks.

8 References

Bai, L. *et al.* (2018) ‘Air Pollution Forecasts: An Overview’, *International Journal of Environmental Research and Public Health*. Multidisciplinary Digital Publishing Institute, 15(4), p. 780. doi: 10.3390/ijerph15040780.

Eiben, A. E. and Smith, J. E. (2015) *Natural Computing Series Introduction to Evolutionary Computing*. Available at: www.springer.com/series/ (Accessed: 28 April 2019).

Fogel, David B, T. B. (2009) ‘Evolutionary Computation 1 Basic’, *Comprehensive Chemometrics*, pp. ix–x. doi: 10.1016/B978-044452701-1.09002-5.

Gers, F. A., Urgen Schmidhuber, J. J. and Cummins, F. (1999) *Learning to Forget: Continual Prediction with LSTM*, *IEE*. Available at: http://www.idsia.ch/http://www.idsia.ch/ (Accessed: 8 January 2019).

Grivas, G. and Chaloulakou, A. (2006) ‘Artificial neural network models for prediction of PM10 hourly concentrations, in the Greater Area of Athens, Greece’, *Atmospheric Environment*. Pergamon, 40(7), pp. 1216–1229. doi: 10.1016/J.ATMOSENV.2005.10.036.

Hinton, G., Srivastava, N. and Swersky, K. (2014) *Neural Networks for Machine Learning Lecture 6a Overview of mini---batch gradient descent*. Available at: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf (Accessed: 25 April 2019).

Ho, T.-Y., Lam, P.-M. and Leung, C.-S. (2008) ‘Parallelization of cellular neural networks on GPU’, *Pattern Recognition*. Pergamon, 41(8), pp. 2684–2692. doi: 10.1016/J.PATCOG.2008.01.018.

Hochreiter, S. and Schmidhuber, J. (1997) ‘Long Short-Term Memory’, *Neural Computation*, 9(8), pp. 1735–1780. doi: 10.1162/neco.1997.9.8.1735.

Huqqani, A. A. *et al.* (2013) ‘Multicore and GPU Parallelization of Neural Networks for Face Recognition’, *Procedia Computer Science*, 18, pp. 349–358. doi: 10.1016/j.procs.2013.05.198.

Jordan, J. (2018) *Setting the learning rate of your neural network.* Available at: https://www.jeremyjordan.me/nn-learning-rate/ (Accessed: 20 April 2019).

LeCun, Y. *et al.* (1998) ‘Gradient-based learning applied to document recognition (unread, the MNIST reference)’, *Proceedings of the IEEE*, 86(11), pp. 2278–2324.

Li, X. *et al.* (2017) ‘Long short-term memory neural network for air pollutant concentration predictions: Method development and evaluation’, *Environmental Pollution*. Elsevier, 231, pp. 997–1004. doi: 10.1016/J.ENVPOL.2017.08.114.

Lipton, Z. C., Berkowitz, J. and Elkan, C. (2015) *A Critical Review of Recurrent Neural Networks for Sequence Learning*. Available at: https://arxiv.org/pdf/1506.00019.pdf (Accessed: 7 January 2019).

Lucidarme, P. (2017) *Simplest perceptron update rules demonstration • Robotics, Teaching & Learning*. Available at: https://www.lucidarme.me/simplest-perceptron-update-rules-demonstration/ (Accessed: 17 April 2019).

Moawad, A. (2018) *The magic of LSTM neural networks – DataThings – Medium*. Available at: https://medium.com/datathings/the-magic-of-lstm-neural-networks-6775e8b540cd (Accessed: 18 April 2019).

Ruder, S. (2017) *An overview of gradient descent optimization algorithms **. Available at: http://caffe.berkeleyvision.org/tutorial/solver.html (Accessed: 25 April 2019).

Saha, S. (2018) *A Comprehensive Guide to Convolutional Neural Networks—the ELI5 way*. Available at: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (Accessed: 22 April 2019).

Shirish Keskar, N. *et al.* (2017) *ON LARGE-BATCH TRAINING FOR DEEP LEARNING: GENERALIZATION GAP AND SHARP MINIMA*. Available at: https://arxiv.org/pdf/1609.04836.pdf (Accessed: 20 April 2019).

Srivastava, N. *et al.* (2014) *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, *Journal of Machine Learning Research*. Available at: http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf (Accessed: 26 April 2019).

Tsantekidis, A. *et al.* (2017) *Forecasting Stock Prices from the Limit Order Book using Convolutional Neural Networks*. Available at: http://users.auth.gr/passalis/assets/pdf/confs/2017_CBI_CNNLOB.pdf (Accessed: 22 April 2019).
Vrajitoru, D. (2000) *Large Population or Many Generations for Genetic Algorithms? Implications in Information Retrieval*. Available at: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=ACD0CF825F8F882F8246D3D6980D1937?doi=10.1.1.21.1147&rep=rep1&type=pdf (Accessed: 28 April 2019).
Whitley, D., Starkweather, T. and Bogart, C. (1990) 'Genetic algorithms and neural networks : optimizing connections and connectivity', 14, pp. 347–361.
Zeiler, M. D. (2012) *ADADELTA: AN ADAPTIVE LEARNING RATE METHOD*. Available at: https://arxiv.org/pdf/1212.5701.pdf (Accessed: 25 April 2019).