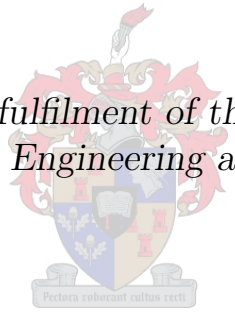
Predictive Models for Smart Vineyards

by
F.R. Lüttich



Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Engineering at Stellenbosch University

Supervisor: Prof. T.R. Niesler
Department of Electrical & Electronic Engineering

April 2019

Stellenbosch University https://scholar.sun.ac.za

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the owner of the copyright thereof (unless to the extent explicitly otherwise stated) and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

March 2019

Copyright © 2019 Stellenbosch University

All rights reserved

Stellenbosch University https://scholar.sun.ac.za

Abstract

We investigate the application of machine learning algorithms to the predictive analysis of environmental datasets compiled from two distinct vineyards. These datasets include the soil temperature at various depths and locations, the soil moisture content of the same locations and the bud-burst dates. Measurements were taken regularly over the space of four months for one vineyard and over twelve months for the other.

The prediction of the soil temperature from either ambient measurements or from satellite data, as well as the prediction of soil moisture content and the bud-burst dates were the primary objectives of our analysis. Linear regression, feedforward neural networks and recurrent neural networks were considered as algorithms. For the neural networks, several training strategies were considered.

It was found that neural networks outperform linear regression when predicting soil temperatures from ambient temperature and humidity, and also when predicting soil moisture content from ambient temperature, humidity and rainfall data. Although recurrent neural networks (LSTMs) were able to achieve even better results when the data was carefully prepared, these networks were sensitive to discontinuities present in the data due to faulty sensor measurements. Feedforward neural networks, on the other hand, were more robust to these errors. Since sensors placed in a vineyard are exposed and must remain unattended, this is an important aspect to consider. It was also found that soil temperatures could be predicted with a modest loss in accuracy from freely-available satellite land temperature measurements. Although cloud cover leads to sporadic non-availability of the measurements, they represent a very attractive alternative to locally installed weather sensors since they would no longer need to be installed or maintained.

For soil moisture content and bud-burst dates neural networks provided better predictions than a naïve guess. While this indicates potential for such models, these results must be re-examined using a larger dataset.

Although this thesis presents only preliminary results due to the lack and small size of suitable datasets, our results nevertheless clearly indicate the potential of machine learning techniques to assist viticulture.

Stellenbosch University https://scholar.sun.ac.za

Opsomming

In hierdie tesis ondersoek ons die toepassing van masjienleer algoritmes op die voorspellende ontledings van omgewings data stelle saamgestel uit lesings van twee verskillende blokke wingerde. Hierdie data stelle sluit lesings van die grond temperatuur op verskillende dieptes en areas, ondergrondse water inhoud en die "bud-burst" of bloeisel datums in. Data was versamel oor 'n tydperk van vier maande vir die een blok en oor twaalf maande vir die ander blok wingerd.

Die voorspelling van grond temperatuur, vanaf of die omgewings temperatuur, of vanaf satelliet data, asook van die grond vog inhoud en die bloeisel datums was die primêre doelwitte van ons ontledings. Lineêre regressie, vorentoe-voerende neurale netwerke (VVNNe) en wederkerende neurale netwerke (WNNe) was oorweeg as algoritmes. Vir die neurale netwerke was verskeie opleidings strategië oorweeg.

Dit was gevind dat neurale netwerke, lineêre regressie oortref met voorspelling van grond temperature vanaf omgewings temperature en humiditeit, asook met die voorspelling van grond vog inhoud vanaf omgewings temperatuur, humiditeit en reënval data. Alhoewel wederkerige neurale netwerke selfs beter resultate gelewer het wanneer die data stelle noukeurig voorberei was, was hierdie netwerke sensitief vir diskontinuïteite in die data as gevolg van foutiewe sensor lesings. Die VVNNe, aan die ander kant, was meer robuus. Aangesien sensors in wingerde blootgestel word aan die elemente, en hulle sonder toesig moet funksioneer vir uitgerekte periodes, is hierdie 'n belangrike aspek om te oorweeg in enige formulerings. Dit was ook gevind dat voorspellings rakende grond temperature, voorspel kon word met 'n minimale verlies aan akkuraatheid vanaf vrylik beskikbare satelliet land-oppervlak temperature. Alhoewel wolkbedekking lei tot sporadiese onderbreking van die lesings, bly dit 'n aantreklike alternatief tot lokale weer sensors, aangesien hulle nie op grondvlak geïnstalleer of onderhou hoef te word nie.

Grond vog lesings en bloeisel datums kon meer akkuraat voorspel word as 'n naïewe raaiskoot. Alhoewel hierdie bevindinge aandui dat hierdie bevindinge potensiaal inhou, moet hierdie resultate her-evalueer word met groter data stelle vir beter betroubaarheid.

Hierdie tesis verteenwoordig slegs voorlopige resultate, as gevolg van die gebrek aan groot genoeg en geskikte data stelle, maar steeds dui ons resultate duidelik die potensiaal van masjienleer tegnieke om wingerd-en-wynkunde beplannings by te staan in die ontwikkeling van meer betroubare resultate.

Stellenbosch University https://scholar.sun.ac.za

Acknowledgements

- Prof. Thomas Niesler, for the best supervision I could ever ask for. He went through tremendous effort to make time to share his knowledge and experience with me, even though he had many other responsibilities at the same time.
- Dr Tara Southey, from the agricultural department for not only supplying us with the data we used, but also helping us understand the data better, as well as helping us understand which results would benefit viticulture the most and why.
- My loving wife, Belinda, for all her love, support and understanding throughout this time. This thesis would not have been possible without her.
- My parents for all their support and prayers throughout the last two years.
- All my family and friends that helped me get through this time. Special thanks to my oldest friend, Corné Smith, for all the support and encouragement he offered me throughout the time.
- Everyone in the DSP lab for all the help and chats throughout the years. Special thanks to Raghav Menon for all the help in finding my feet with machine learning.
- Winetech for supporting this research financially.

Stellenbosch University https://scholar.sun.ac.za

Contents

Stellenbosch University https://scholar.sun.ac.za

Stellenbosch University https://scholar.sun.ac.za

Stellenbosch University https://scholar.sun.ac.za

List of Figures

Stellenbosch University https://scholar.sun.ac.za

Stellenbosch University https://scholar.sun.ac.za

List of Tables

Stellenbosch University https://scholar.sun.ac.za

List of Abbreviations

Adam	adaptive moment estimation
BP	back propagation
BPTT	back propagation through time
cm	centimeter
DoY	day of the year
FFNN	feedforward neural network
HoM	hour of measurement
HV	high-vigour
km	kilometer
LST	land-surface temperature
LSTM	long-short term memory
LV	low-vigour
m	meter
max	maximum
min	minimum
ML	machine learning
MLP	multilayer perceptron
mm	millimeter
MV	medium-vigour
NAG	Nesterov accelerated gradient
NN	neural network
°C	degrees Celsius
ReLU	rectified linear unit
RH	relative humidity
SGD	stochastic gradient descent
SOFM	self-organising feature map
Ssw.	Somerset West
Stb.	Stellenbosch
SVM	support vector machine
Temp.	temperature
VIs	vegetation indices
WSN	wireless sensor network

Stellenbosch University https://scholar.sun.ac.za

Chapter 1

Introduction

Agriculture is the process of cultivating land and breeding animals to provide humans with essential products, including food, fibre and medicine. The development of agriculture can be considered a breakthrough in human development that gave rise to sedentary human civilisation. It gave rise to food surpluses that enabled people to start living in cities. Without agriculture, humans would still be living a nomadic lifestyle that would greatly hinder technological development.

Modern technologies such us motorised transportation, pesticides and fertilizers have made it possible for agriculture to become increasingly efficient and produce ever greater crop yields. This has allowed cities to expand and technology to thrive. We have, however, reached a point where populations continue to increase even as the land area available for agriculture has become limited. This has given rise to an even greater need for agricultural efficiency and once again agriculture is depending on technology for further advancement.

One of the oldest branches of agriculture, viticulture, which is the cultivation and harvesting of grapes, and the field of oenology, which is the science and study of wine and wine making, have become increasingly dependant on technology. Oenology has evolved over the years to become a refined science where precise measurements and data are required for optimal results. It follows a yearly pattern, since the cultivation of grapevines and the growth of grapes are dependant on the season. This means that each iteration of experimentation takes a year to complete in full since the results of new experiments can only be observed after the harvest season. When compared to other sciences, where one is limited mostly by the

Stellenbosch University https://scholar.sun.ac.za

number of work-hours that can be spent, this slow pace necessitates oenology to extract as much data as possible from every harvest.

1.1 Manual environmental monitoring

At first, technology allowed the accurate measurement of environmental parameters such as soil temperature, humidity and moisture. This was a great advancement since now correlations could be made between certain environmental parameters and the harvesting and outcome of the grapes. For example, one study by Burgos et Al. in 2007 found that warmer soil temperatures decreased the days between bud-burst and flowering [6]. Before such measurements were available, the experience of the farmer had to be relied upon.

While such measurements are very useful, they also require a lot of time and effort to acquire. Firstly, the devices, while not too expensive on their own, become very expensive when required in large numbers to cover a farm or large vineyard. Initially, the measurements also had to be collected manually by physically visiting each measuring station, which could take up a lot of time. An associated limitation was that it was usually not possible to monitor the data on the go, but only after the data collection had been completed. This meant that should one of the devices become faulty, this might become apparent only after all the data had been collected.

1.2 Wireless Sensor Networks

Wireless Sensor Networks (WSNs) are a relatively new technology that enables multiple devices (such as the environmental monitoring devices) to be linked to each other using radio communication channels. All measurement devices could be connected in this way, either directly to the closest neighbours via chaining, or to a master node. This master node would usually have an interface allowing the operator to request the data from all measuring nodes without having to visit each individually [20]. Certain WSNs also provide real-time feedback on all the devices in the network, so that faulty devices can quickly be identified.

Stellenbosch University https://scholar.sun.ac.za

This technology, while still evolving, solves many of the problems associated with isolated measurement devices. The WSN must still be installed on the vineyard or farm, however. Even though WSNs have become cheaper and more energy efficient, they remain a significant financial investment due to the large number of sensors involved.

1.3 Machine Learning

Machine learning refers to a class of algorithms that learn to achieve a desired outcome (such as regression or classification) through a process called "training" in which the parameters of the algorithm are optimised based on example data. Machine learning has gained massive popularity recently due to the increasing availability of computational resources.

When considering the application of machine learning to viticulture, the idea is to use measured environmental and other data to learn patterns and use these to predict particular variables of interest to the farmer. It may even be possible to use the software models' predictions to replace environmental sensors completely. This would reduce the time and money needed to install and maintain monitoring systems.

According to Vazquez et al. the level of precision generally accepted as accurate for remote sensing land surface temperature estimations is between 1 and 2°C [38]. The accuracy of an average soil temperature sensor is around 0.4°C [1]. Hence, if a machine learning model could perform with similar accuracy, it could be considered accurate enough for practical use, reducing the need to install expensive hardware sensors. To develop such machine learning algorithms, large and detailed datasets are required for training.

[1]Based on temperature sensor No. 107 sold by Campbell Scientific® (source: http://www.campbellsci.com/soil-temperature).

Stellenbosch University https://scholar.sun.ac.za

1.4 Aims and scope of this thesis

This project aims to investigate ways in which machine learning techniques could be used to help advance modern agriculture, specifically viticulture, by decreasing the need for manual data collection and extraction. Specifically, this project will investigate how accurately certain environmental parameters can be predicted by using different machine learning models that were trained on either easily obtained data, or data received from typical sensors used in the viticulture industry.

First we will consider the classic method of linear regression for a relatively small dataset. The task we consider is the prediction of ambient temperature from soil measurements, soil measurements from other soil measurements, as well as predicting soil temperatures from ambient temperature measurements.

Next, a more complex and current model, the feedforward neural network, will be used to perform the same task. Here, we also investigate the performance of various pre-training techniques, as well as the addition of dynamic data as additional inputs. A brief investigation was also done into the performance of a model trained on one vineyard's data to predict soil temperatures of another vineyard.

After this, we explore the possibilities of predicting other variables like the moisture content in the soil, as well as the bud-burst dates of vines.

Finally, we consider the application of recurrent neural networks in the form of long-short term memory (RNNs with LSTM).

This study lays a foundation upon which further research can build. We will try to provide some insight into which aspects of the data the considered algorithms find most useful for prediction. This can help focus ongoing data collection efforts.

1.5 Overview of this work

This section describes the layout of this thesis by providing an outline of the contents of each chapter.

Stellenbosch University https://scholar.sun.ac.za

Chapter 2 presents a brief summary of the literature on machine learning applications in viticulture. It will be shown that most literature is focussed on computer vision and imaging techniques and not on the precessing of environmental sensor data.

Chapter 3 describes basics of the algorithms and methods used in this study. This includes introductions to linear regression (Section 3.1), neural networks (Section 3.2), as well as extentions to neural networks (Sections 3.2.5 - 3.2.10), and also recurrent neural networks with long-short term memory (Section 3.3).

Chapter 4 presents a summary of the two datasets used in this study. The first dataset concerns a vineyard in Stellenbosch and contains soil temperature measurements and ambient measurements obtained from sensors located locally in the vineyard taken over the space of one year. The second dataset concerns a vineyard in Somerset West and contains soil temperature measurements taken over a span of almost 4 years, although it turned out that not all this data was usable. The Somerset West data also contains ambient measurements from sensors located locally in the vineyard, six months of soil moisture content measurements, as well as the bud-burst dates for the last four years. This chapter also explains how satellite land-surface temperatures were obtained and converted for use in our experiment.

Chapters 5 and 6 describe the experiments performed on the Stellenbosch and Somerset West datasets respectively. These experiments include the prediction of soil temperatures from ambient temperature and humidity measurements, the prediction of soil temperature from freely-available satellite data, the prediction of soil moisture content from ambient temperature, humidity and rainfall measurements, and the prediction of bud-burst dates. Linear regression, feedforward neural networks and recurrent neural networks (LSTMs) are considered. Finally, Chapter 7 concludes the thesis and recommends avenues of future work.

Stellenbosch University https://scholar.sun.ac.za

Chapter 2

Machine Learning in Viticulture

With the rise of machine learning applications in various industries, some research has begun to consider the use of machine learning techniques in Viticulture. However, in comparison with other areas of application such as human language technology, the applications of machine learning to viticulture described in the literature are limited. Furthermore, most applications in viticulture focus on computer vision and image processing and not on sensor networks.

The five most relevant topics of research for applications of machine learning in viticulture are:

- Harvest yield estimation.
- Grape disease detection.
- Vineyard management and monitoring.
- Quality evaluation.
- Grape phenology.

The application of machine learning to these research topics will be discussed in the following sections.

2.1 Harvest Yield Estimation

Being able to estimate or forecast the yield is very important for the wine industry. Yield refers to the size of the harvest, typically given as a mass in tonnes. Traditional methods involve the manual and destructive sampling of the vine bushes, allowing

Stellenbosch University https://scholar.sun.ac.za

them to be inspected by hand to count and determine their weight, berry size, and berry numbers. However, these methods are destructive and time-consuming.

A study in 2011 by Nuske et al considered using an automated computer vision technique combined with a k-nearest neighbour classifier to detect and count green grape berries against a green leaf background [25].

A small vehicle fitted with a visible light camera would drive along the rows of the vineyard. They used several components to enable berry detection. First, they used the radial symmetry transform to identify berry locations. After that, a combination of colour and texture features followed by a k-nearest neighbour classifier was used to classify the detected points. Finally, false positive detections were removed for bunches which do do not have at least five berries in close proximity to each other. They managed to predict the yield to within 9.8% of the actual crop weight.

In 2012 the authors extended their work by utilising calibration data obtained from previous harvests and a small set of hand-picked samples [26]. By using this approach, they were able to improve their yield estimation accuracy by 4% and 3% for the harvest calibrated and hand-picked samples respectively.

Another study by Vincent Casser [7] addressed the problem of colour-based grape detection for in-field images. They performed experiments in four different situations: night time red berries, night time white berries, day time red berries and day time white berries, attempting to classify individual berries. By using feedforward neural networks (FFNNs) they were able to achieve an average classification accuracy of 93%. A comparison with a support vector machine (SVM) showed that the FFNN offered an advantage in terms of computation time.

A more recent study by Aquino et al. in 2017 [1], uses mathematical morphology and pixel classification for grapevine berry counting. Firstly, a set of berry candidates represented by connected components was extracted. Then, using key features of these components, six descriptors were calculated and used for false positive discrimination using a supervised approach. A low-cost smartphone camera was used to assemble a dataset of 152 images. Two different classifiers were

Stellenbosch University https://scholar.sun.ac.za

tested, a three-layer neural network and an optimised SVM. The neural network outperformed the SVM, yielding consistent recall and precision values of 0.9572 and 0.8705, respectively.

2.2 Vineyard Pruning and Monitoring

For studies relating to vineyard management, wireless sensor networks (WSNs) are the most popular tool since they are efficient and useful for monitoring. There have also been a few studies using computer vision and image processing to monitor the health of the vines as well as for pruning management. However, these will not be discussed here since our focus is on machine learning techniques.

A study by Perez et al. [29] considers grapevine bud detection under natural field conditions to aid in winter pruning. The scale-invariant feature transform (SIFT) was used to obtain low-level image features. Subsequently these were used in a bag-of-features approach to build the image descriptors for an SVM image classifier.

Classification was achieved by sliding a fixed size window over the original image and classifying each part as containing or not containing the target object. Test images were between 100 and 1600 pixels in size. Their results showed a classification recall greater that 0.89 in patches containing at least 60% of the original bud pixels, where the proportion of bud pixels in the patch is greater than 20%, and the bud is at least 100 pixels in diameter. For patches that hold more than 90% of the bud pixels, and these pixels represent between 20% and 30% of the patch (i.e. patches from three to five times larger buds), even better results were obtained.

2.3 Disease Detection

Early detection of diseases is a very important research area in viticulture. Diseases that are commonly found on grapes include downy mildew, powdery mildew, anthracnose, grey mold and black rot. These are all caused by fungi. Grown cell disease, on the other hand, is an example of a disease caused by bacteria. These diseases can cause massive problems for the grapes and economic losses for

Stellenbosch University https://scholar.sun.ac.za

the vineyard. For example downy mildew can taint the flavour of wine [35] and grey mold (botrytis) can decrease yield and wine quality [22].

There are several reasons why the detection of diseases is such a challenge for machines. One is that the grapes may be covered by a natural bloom that has similar visual characteristics to that of diseased berries. Another is that the symptoms of the disease may differ depending on the variety and the developmental stage of the grape. More than one disease may also be present at the same time, adding a further challenge to automatic detection. One of the most difficult problems faced, however, is that factors such as nutrient deficiencies, pesticides and weather can produce similar symptoms to those of diseases.

Various studies have addressed this problem and a lot of these focus on applications of computer vision. However, there are several studies that use machine learning to attempt to address the problem.

One study, by Meunkaewjinda et al. [21], proposed an automatic diagnosis system for grape leaf disease. First, in an attempt to remove background noise, grape leaf segmentation was performed. A self-organising feature map (SOFM) was used together with a neural network to recognise the colours of the grape leaf. A modified SOFM model with a genetic algorithm for optimization was used to perform the grape leaf disease segmentation. A SVM was then applied to classify the grape leaf disease. The model categorised the leaf images into three classes: either scab disease, rust disease or no disease. The system demonstrated the potential for automatic diagnosis of grapevine diseases.

Another study by Li et al. in 2011 [19] proposed an image recognition technique to identify and diagnose grape downy mildew and grape powdery mildew. First, pre-processed images were compressed using nearest neighbour interpolation. The k-means algorithm was then used to perform unsupervised segmentation of the disease images. After fifty shape, colour and texture features were extracted from the images, a SVM classifier was used to perform disease recognition. This achieved recognition rates of 90% and 93.33% for downy mildew and powdery mildew respectively.

Stellenbosch University https://scholar.sun.ac.za

After Indian vineyards suffered great losses from leaf diseases in the 1990s, Sanjeev et al. proposed a diagnosis and classification approach for grape leaf diseases using neural networks [31]. Grape leaf images with complex backgrounds were input to the system. Green pixels were isolated by thresholding, followed by noise removal using anisotropic diffusion. The grape leaf diseases were then segmented using k-means clustering. Best classification results were achieved using a feedforward neural network.

In a different study, Harshal et al. [39] used background removal segmentation, leaf texture analysis and pattern recognition to detect downy mildew and black rot. A unique fractal-based texture feature was used to characterise the leaf texture and a multiclass SVM was used to classify the extracted pattern. An accuracy of 96.6% was achieved.

The detection of black rot, downy mildew, powdery mildew, anthracnose, gray mold, and crown gall diseases was also considered by Shilpa et al. [13]. The Haar wavelet transform was used for feature extraction and a feedforward neural network for classification, leading to a classification accuracy of 93%.

2.4 Other machine learning applications in viticulture

One factor that is often used as an indicator for the optimal time for harvest is the grape seed maturity. Using traditional methods to identify maturity is often very time consuming and subjective, since it is often performed by human sensory and visual analysis. There have been various studies trying to improve this by application of image processing and machine learning techniques.

One such study was performed by Zuniga et al. in 2014 [42]. Their method was based on seed images and allowed the classification of three seed classes: mature, immature, and over-mature. The invariant colour model [10] was used for seed segmentation, to avoid shadows and highlights. Using the results of a previous study by Avila et al. [2] the c3 colour model [10] was chosen to transform the values of the pixels. Classification was achieved by three multilayer perceptrons (MLPs), one for each class. A recognition rate of 90% was achieved on the training set and 86% on the test set.

Stellenbosch University https://scholar.sun.ac.za

Romero et al. had considered the estimation of vineyard water status using multispectral imagery from an unmanned aerial vehhicle (UAV) platform and machine learning algorithms [30].

In this work, several vegetation indices (VIs) derived from aerial multispectral imagery were used to estimate the midday stem water potential (Ψ_{stem}) of grapevines. Machine learning algorithms were used to evaluate relationships between Ψ_{stem} and VIs. Simple regression models showed little to no correlation. However, application of artificial neural networks with VIs as inputs showed high correlation between the estimated and measured water potential. Correlations of R=0.8, 0.72 and 0.62 were obtained for the training, validation and test sets, respectively.

2.5 Conclusion

The fairly small set of studies described in this chapter demonstrates that there is great potential for the application of machine learning in viticulture. However, the literature study also shows that the work done so far is limited, and there is great potential to further consider the practical challenges of viticulture by the application of and analysis with machine learning algorithms. This thesis will consider specifically the modelling and prediction of soil temperatures, soil moisture content and bud-burst dates using machine learning techniques.

Stellenbosch University https://scholar.sun.ac.za

Chapter 3

Methods

One main objective of this study is to determine how accurately some measurements can be predicted from other measurements that are easier to obtain. This can be represented algebraically as

$$\underline{y} \approx \underline{\hat{y}} = h_\theta(\underline{x}) \tag{3.1}$$

Here $\underline{y}$ is a vector $[y_0 \ ... \ y_{p-1}]$ of P measurements we wish to approximate using the set of q measurements $\underline{x} = [x_0 \ ... \ x_{q-1}]$. The function h_θ will be used to accomplish this approximation and has r parameters $[\theta_0 \ ... \ \theta_{r-1}]$ which must be estimated from the data $\underline{x_0} \ ... \ \underline{x_{N-1}}$ so that the prediction $\underline{\hat{y}}$ most closely approximates the true values $\underline{y}$. Three forms of the function h_θ will be considered in this work: a linear model, a feed-forward neural network, and a recurrent neural network.

To estimate the parameters $\underline{\theta}$ of the model $h_\theta(\underline{x})$, it is common experimental procedure to split the available data $\underline{x}$ into a training, validation, and testing partition. The parameters $\underline{\theta}$ are estimated from the training partition, any additional "hyper-parameters" are optimised on the validation set, and final independent testing is performed on the test set.

Going into this study, we were not told specifically which tasks need to be considered or which experiments should be run. Instead, the objective was a fairly open-ended search for useful patterns in the data.

When considering experiments to perform using the data, we had to consider

Stellenbosch University https://scholar.sun.ac.za

a few things, most notably what exact data we have and the information that it carries. We also later wanted to see if we could further improve on the first experiments that were performed using only the smaller dataset. Lastly, the results of the experiments should be of interest to the wine industry in some way.

Various different experiments were performed, namely:

1. Predicting ambient temperatures using soil temperatures.
2. Predicting soil temperatures using other soil temperatures.
3. Predicting soil temperatures using microclimate logger data.
4. Predicting soil temperatures using freely available data.
5. Predicting soil temperatures using a mixture of available data.
6. Predicting moisture levels of the soil using freely available data.
7. Predicting the bud-burst dates using soil temperatures.
8. Predicting soil temperatures given a series of previous soil temperatures.

The following chapters will describe the experiments that were performed and the results that were obtained. All experiments use the data described in Chapter 4. The data sets will be referred to as:

- The soil data (from Section 4.1).
- The mesoclimate data (from Section 4.2).
- The weather station data (from Section 4.3).
- The rainfall data (from the weather station data in Section 4.3).
- The moisture data (from Section 4.5).
- The satellite data (from Section 4.6).
- The bud-burst data (from Section 4.9).

Stellenbosch University https://scholar.sun.ac.za

3.1 Linear Regression

Linear regression was applied to determine whether some measurements could be inferred from others using a simple linear relationship.

The algorithm tries to predict values y, given inputs, x, by linear transformation with parameters θ as shown below [9]:

$$y \approx h_\theta(x) = \underline{\theta} \cdot \underline{\mathbf{x}} \tag{3.2}$$

Here, y is the true value that the model is trying to predict and $h_\theta(x)$ is the model's prediction given inputs x. The squared error cost function, $J(\theta)$ is minimised by iteratively adjusting the parameters θ. The cost function, $J(\theta)$ is as follows [4]:

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^i) - y^i)^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right] \tag{3.3}$$

Where the second term,

$$\lambda\sum_{j=1}^{n}\theta_j^2 \ ,$$

is the regularization factor [3], using λ (lambda) to determine how much regularization should be applied. The parameters θ are updated iteratively using the equations below:

$$\begin{aligned} &\textit{Repeat } \{ \\ &\theta_0 := \theta_0 - \alpha\left[\frac{1}{m}\sum_{i=0}^{m}(h_\theta(x^i) - y^i)x_0^i\right] \\ &\theta_j := \theta_j\left(1 - \alpha\frac{\lambda}{m}\right) - \alpha\left[\frac{1}{m}\sum_{i=0}^{m}(h_\theta(x^i) - y^i)x_j^i\right] \\ &\}, \textit{ until convergeance} \end{aligned} \tag{3.4}$$

This will be repeated until convergence or for a fixed number of iterations. In this formula, θ (theta) represents the parameters of the linear predictor. The first of these, θ_0, is known as the bias term and is not included in the regularization term. Furthermore, x_0 is defined to always equal 1 and hence has a separate update equation. The meta-parameter α is the learning rate and determines how harshly the parameters (θ) are updated after every step. The meta-parameter λ

Stellenbosch University https://scholar.sun.ac.za

(lambda) is the regularization parameter and in this case was set to 0.03 after some testing. The number of entries in the dataset is represented by the variable m. x and y respectively represent the input and target values in all sets. The superscripts i represents the specific data point of the set (set rows) and the subscript j represents the parameter number (set columns). The predicted values will be output by the function h_θ, which assumes the following form [3]:

$$h_\theta(x) = \underline{\theta} \cdot \underline{\mathbf{x}} = x_0.\theta_0 + x_1.\theta_1 + x_2.\theta_2 + ... + x_n.\theta_n \tag{3.5}$$

Where n is the number of input features. The input features are based on the measurements which the model uses to predict the true value of y. All data is normalised to have zero mean.

3.1.1 Cross-validation procedure

For all the linear regression experiments that follow, the same dataset and rotation method was used. The dataset was split into 10 parts. The first 8 parts (80%) were used as the training data. The next 10% was used as the validation set. The last 10% was used as the unseen test data on which the final results of the model were also calculated. The final error metric to be seen in the following experiments are based on the predictions that the model makes on this final 10% of the dataset.

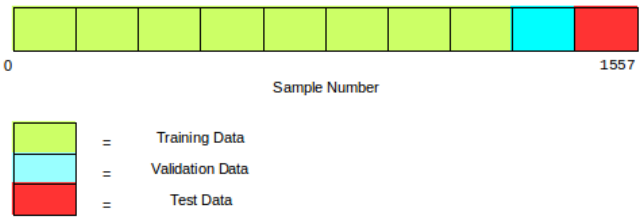


Figure 3.1: Linear regression dataset split.

Figure 3.1 graphically shows how the dataset is split up into its three parts. After training and testing the model with this dataset, an average error was obtained. The dataset was then shifted to the right by 10% and the training and testing procedure was performed again. After 10 such shifts we will have trained and tested on all the different parts of the dataset. The final error is the

Stellenbosch University https://scholar.sun.ac.za

average over all 10 of these runs. Figure 3.2 visually shows how the dataset rotates for the first two runs.

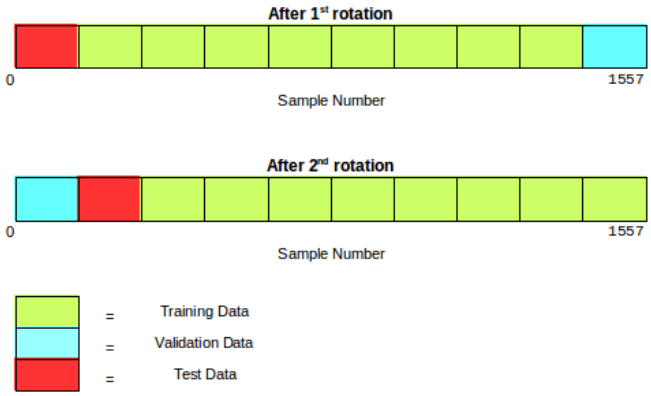


Figure 3.2: Linear regression dataset split after rotation.

3.1.2 Calculating the error score

To calculate the error score of a model, the predicted outputs were compared with the targets (true values). Each output was compared with its corresponding target value by taking the absolute value of the difference divided by the target, as shown in Equation 3.6.

$$error = \frac{\|target - output\|}{target} \tag{3.6}$$

This results in a set of error values (one per target) between 0 and 1 which is averaged to obtain the overall test-set error value. This can be repeated multiple times (building and training the model from scratch), and the average overall error used as the final error score representing the success of the model.

This measure of prediction accuracy is known as the mean absolute percentage error (MAPE) or also as the mean absolute percentage deviation (MAPD). This measure has two drawbacks, however. The first is that it is not appropriate when the dataset contains zero values, since this causes a division by zero. The second is that, for predictions that are too low, the error rate can never exceed 100%, whereas for predictions that are to high there is no upper limit. This can cause statistical models using MAPE to favour models that under-estimate over models that over-estimate. Fortunately, since the temperatures measured are all positive, the first issue is not encountered. Also, as will be seen, in this study the error

Stellenbosch University https://scholar.sun.ac.za

rates never come close to 100%. In fact, most errors will be in the region of 5%, which should avoid under-estimating models being favoured.

Taking in to account that results from our dataset will not suffer any negative consequences by using MAPE, MAPE was chosen because of how intuitive and easy to read and interpret the error rate is.

3.2 Neural Networks

Neural networks are the second type of model considered in this study. Neural networks are made up of multiple layers of linear weights, followed by non-linear activation functions allowing the model to learn more complex non-linear relationships than a simpler model like linear regression. The following section will give a brief introduction to neural networks, and describe the neural network learning-techniques that were used for experimentation.

3.2.1 A brief introduction to Multilayer Perceptrons (MLPs)

At the core of any neural network lies a small unit called the perceptron (the neurons of the network). A perceptron is a really basic unit that works in a similar way to a logic circuit component like a NAND gate for instance. We depict a perceptron in Figure 3.4.

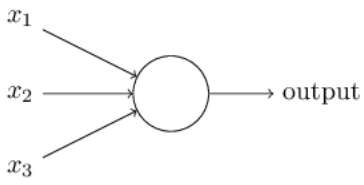


Figure 3.3: The basic perceptron.

In Figure 3.4 x_j, $(j\ in\ \{1, 2, 3\})$ represents the inputs, and the arrows represent the weights w_j for each input. The inputs are multiplied by the weights and added together to calculate the neurons activation, which is given by Equation 3.7.

Stellenbosch University https://scholar.sun.ac.za

$$z = \sum_j w_j x_j \tag{3.7}$$

The neuron is said to *fire* when the activation value passes a certain threshold. The neurons output will then either be 1 or 0 depending on if it has fired or not [9]. In algebraic terms:

$$\begin{aligned} output = 0 \quad &\text{if} \quad \sum_j w_j x_j < threshold \\ output = 1 \quad &\text{if} \quad \sum_j w_j x_j \geq threshold \end{aligned} \tag{3.8}$$

If we see x_j and w_j as two vectors of length j, we can simplify this equation by rewriting the sum term as a dot product, i.e. $\sum_j w_j x_j = w \cdot b$. Next, we can move the threshold to the left of the equality and replace it by what is called the bias (b), i.e. $threshold = -b$. One can think of the bias as an indication of how easy it is for the neuron to fire. The lower the bias, the more easily the neuron will activate. This allows us to rewrite Equation 3.8 as follows.

$$\begin{aligned} output = 0 \text{ if } w \cdot x + b < threshold \\ output = 1 \text{ if } w \cdot x + b \geq threshold \end{aligned} \tag{3.9}$$

A drawback of neurons such as these described above is that small changes in the weights can cause big changes in the output of the network since, if the weights change enough to push the activation of one neuron past the threshold and fires, this can cause other neurons to fire, causing a chain reaction. One way to avoid this is to include a continuous activation function in the neuron instead of a hard threshold. With this change, the output of the neuron is described by Equation 3.10 [4]:

Stellenbosch University https://scholar.sun.ac.za

$$\begin{aligned} output &= a(z) \ , \\ with \quad z &= w \cdot x + b \end{aligned} \tag{3.10}$$

Here, a is the activation function. There are many popular activation functions used in practice. One of these are called the sigmoid function which takes on the following form:

$$a(z) = \frac{1}{1 + e^{-z}} \tag{3.11}$$

The sigmoid causes the neuron to have a smooth output function in response to gradient changes in the input instead of the discontinuous step function it had before. The output of the neuron will now also only change by a small amount as a result of small changes to the weights, which is an important property when we wish to train the weights by gradient descent. This particular function also has a nice derivative which makes it even better to use as an activation function.

When multiple layers of these neurons are stacked together, they are referred to as *multilayer perceptrons* or MLPs [4]. Even though the neurons used are hardly ever perceptrons, but rather sigmoid neurons, this is still a widely used name. An MLP usually consists of an input layer, one or more hidden layers, and the output layer as can be seen in the figure below:

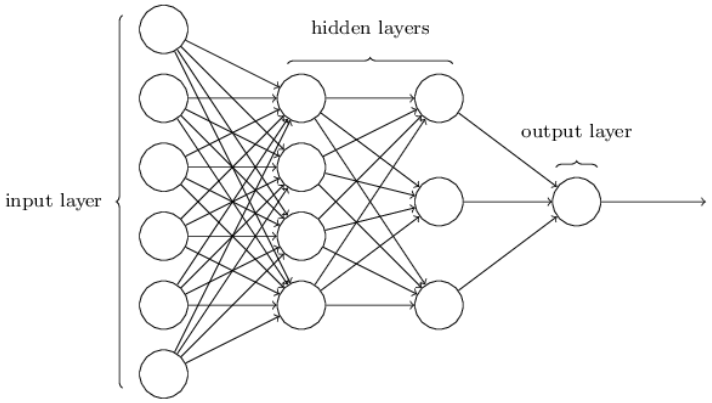


Figure 3.4: An example of an MLP, as used for classification or regression.

Stellenbosch University https://scholar.sun.ac.za

The size of the input layer will depend on what the network must learn from. If it must classify an image, for example, the inputs may be the raw pixel values. The hidden layers are any layers that are not the input or output layers, and the number of neurons in these layers depend on multiple factors and are fine-tuned along with a number of other hyper-parameters to find the best network configuration for the problem. The size of the output layer will depend on the type of problem. For classification purposes, the output layer often only consists of one neuron (outputting either a 1 or a 0). For regression problems, the size depends on the number of quantities that must be predicted.

3.2.2 Training the MLP

Before it can be used for classification or regression, a neural network must be trained. This can be achieved by iteratively adjusting the weights in a process called *Gradient Descent* towards the (hopefully global) optimum.

First a *forward pass* is performed. This consists of applying an input to the network, and calculating the outputs of each layer using Equation 3.10. This is repeated for each layer until the output is reached.

More formally, assume that all the weights and biases for each layer are contained in matrices W_j and b_j respectively. In this case j refers to the layer number, $j = 1$ for the input layer and $j = n$ for the output layer. Next, recall that the activations for all of the neurons in the $j'th$ layer can be calculated by multiplying the value of the neurons in layer $j-1$, adding the biases, and using that value in the activation function. For the case of the sigmoid neuron this is:

$$a_j(z) = \frac{1}{1 + e^{-z}} \tag{3.12}$$

and where a_j is an array containing the activations for all the neurons in layer j, and

$$z = w_{j-1} \cdot a_{j-1} + b_j \tag{3.13}$$

By doing this for every layer of the network, one at a time, the inputs are passed through the network until the output layer is reached. At this point all of

Stellenbosch University https://scholar.sun.ac.za

the neurons will have a computed activation value. There will be computed values for each output which can be compared with the target values, and the difference can be used to adjust the weights.

At this point it should be mentioned that the initial values for the weights and biases are usually randomised. There are different methods for weight initialization such as pre-training, and using different random distributions, but it should be known that the weights are deliberately initialised before the network is given any input values.

After completing the forward pass, the backward pass is executed. This is achieved by a process called *backpropagation*, or *backprop* which propagates the output error back through the nodes towards the input. The weights are subsequently updated using a technique called *gradient descent*. It should be noted that *backprop with gradient descent* is not the only technique that can be used to train a neural network, but it is a very popular one, and it is also the technique used for the neural network in this work.

3.2.3 Backpropagation with gradient descent

Using the notation that x denotes an input vector, we can then define $y(x)$ as the desired output of the model given the input x, or in other words, the target/ideal value for the model given the input x. The goal is now to define an algorithm that lets us find appropriate weights and biases so that the network can approximate $y(x)$ for every input value of x. To do this we will define a cost function [24]:

$$C(w,b) = \frac{1}{2n}\sum_{x} \|y(x) - a\|^2 \tag{3.14}$$

Here, w and b are the weights and biases respectively, n is the total number of training inputs and a is the output of the model given the input x. Of course a depends on w, b, and x, but has been omitted here for easy readability. This specific cost function is known as the *quadratic* cost function, or also sometimes the *mean squared error* (MSE).

By looking at this equation we can see that it will give us the squared error

Stellenbosch University https://scholar.sun.ac.za

between the model's attempt at the right answer and the actual right answer, summed over all different inputs x. To find the optimal weights and biases for our model to perform well, we must minimise this function with respect to the weights and biases. A cost $C(w, b) = 0$ is the minimum possible cost and indicates that all the input values x in the summation in Equation 3.14 are mapped perfectly to the correct output values.

Consider the minimisation of some cost function $C(v)$ where v can be any number of parameters $v = v_1, v_2, ...$ To find the minimum, we will find the local gradient of the function C at the current value of v and then move v by a small amount in the direction of the negative gradient, i.e. downhill. If we do this repeatedly, each time recomputing the gradient of the new value of v, we must eventually reach a minimum where the gradient is zero.

Assuming for illustration that v is two dimensional, $v = v_1, v_2$, and suppose we move a small amount δv_1 in the v_1 direction and δv_2 in the v_2 direction. This will cause C to change as follows [24]:

$$\Delta C \approx \frac{\partial C}{\partial v_1} \Delta v_1 + \frac{\partial C}{\partial v_2} \Delta v_2 \tag{3.15}$$

We now define Δv as a vector of changes in v, $\Delta v = (\Delta v_1, \Delta v_2)$, as well as the gradient of C to be the vector of partial derivatives, $\left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}\right)$. We'll denote this vector as ∇C, i.e.:

$$\nabla C = \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}\right) \tag{3.16}$$

Having made these definitions, we can now rewrite ΔC as:

$$\Delta C \approx \nabla C \cdot \Delta v \tag{3.17}$$

Using this equation we can deliberately choose Δv so that ΔC is negative. Suppose we choose

$$\Delta v = -\eta \nabla C \tag{3.18}$$

where η is a small positive constant (known as the learning rate). It follows that $\Delta C \approx -\eta \nabla C \cdot \nabla C = -\eta \|\nabla C\|^2$. Since $\|\nabla C\|^2 \geq 0$, it is guaranteed that C

Stellenbosch University https://scholar.sun.ac.za

will always decrease. Hence, we can use Equation 3.18 to compute a value for Δv and then update the parameters v by that amount [24]:

$$v \to v' = v - \eta \nabla C \tag{3.19}$$

We can use this update rule again and again until, hopefully, a global minimum is reached. Considering again our original problem, we see that v is a vector containing our weights and biases. Therefore we can use the update rule in Equation 3.19 to update the weights and biases by moving down the slope of the cost function $C(v)$ until we reach a minimum [24]. Thus we repeat:

$$w_k \to w'_k = w_k - \eta \frac{\partial C}{\partial w_k} \tag{3.20}$$

$$b_l \to b'_l = b_l - \eta \frac{\partial C}{\partial b_l} \tag{3.21}$$

until approximate convergence is achieved. In essence, this is how a neural network is trained. It must be noted that this is a very simple explanation and that a large body of literature is dedicated to the refinement of gradient descent for neural network training.

3.2.4 Dropout

Many complex patterns can be learned by a neural network with a large number of parameters and enough training time. There is however still a serious potential problem with these networks called overfitting.

Overfitting is what happens when a neural network learns the patterns of the training data well but fails to generalise to other, unseen, data. Overfitting leads to very promising results on the training data, but poor performance on unseen testing data. Dropout is a technique developed by Nitish Srivastava et al. to help address this problem. According to the authors [34]:

"In a standard neural network, the derivative received by each parameter tells it how it should change so the final loss function is reduced, given what all other units are doing. Therefore, units may change in a way that they fix up the mistakes of the other units. This may lead to complex co-adaptations. This in turn leads

Stellenbosch University https://scholar.sun.ac.za

to overfitting because these co-adaptations do not generalise to unseen data."

Dropout simply ignores the weights of a few randomly chosen neurons at every training iteration. The choice of these neurons is governed by a pre-selected dropout probability. This strategy causes the weights of other neurons to become unreliable, forcing every neuron to learn more robust features from the data, and not rely too heavily on other neurons to correct for its mistakes.

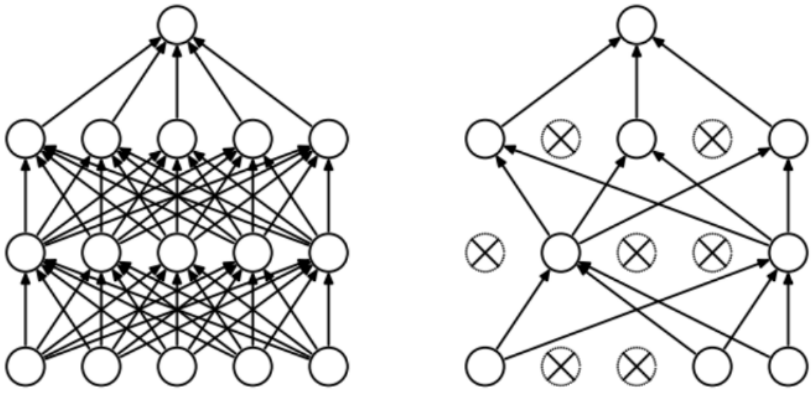


Figure 3.5: An illustration of dropout. Reproduced from: Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting", JMLR 2014 [34].

Dropout increases the number of training iterations needed for convergence. However, the training time per epoch is also reduced.

Dropout was applied when training most models used in this study. It will be made clear when it was not used for specific cases. In the machine learning frameworks used in this study, dropout is implemented as a separate neural network layer. Based on the dropout probability, this layer selects which units in the previous layer are set to zero, and which to pass through. This effectively removes the corresponding units preceding the dropout units from the network.

Stellenbosch University https://scholar.sun.ac.za

3.2.5 Momentum

When using gradient descent to update the network's parameters, the equation with which the weights update their values is shown in Section 3.2.3. We can rewrite this equation in a form that is a bit easier to read as follows:

$$\theta_{t+1} = \theta_t - \alpha \nabla J(\theta_t) \tag{3.22}$$

In Equation 3.22, θ_t represents all the parameters at the current time step, θ_{t+1} the updated parameters, α the learning rate and $\nabla J(\theta_t)$ the gradient of the loss function with respect to the model parameters θ_t.

A problem with this standard formulation of gradient descent is that the gradient of the loss function, $\nabla J(\theta)$, changes very quickly after each iteration. By keeping the learning rate, α, small, the model can still converge, but this may take a very long time. When α is too large, the training process can diverge.

To help overcome this problem, a technique called momentum was proposed. It is called momentum since it mimics the way in which a ball that rolls down a hill builds up momentum in the direction it is travelling. This momentum allows it to almost ignore minor bumps in the path and to keep going downhill. The equations for gradient descent with momentum can be written as,

$$\begin{aligned} v_{t+1} &= \mu v_t - \alpha \nabla J(\theta_t) \\ \theta_{t+1} &= \theta_t + v_{t+1} \end{aligned} \tag{3.23}$$

Here, we introduce a new hyperparameter μ, which is the momentum parameter. It simply dictates how much the direction of each weight update is influenced by the previous update direction. This reduces oscillations between updates during training.

3.2.6 Nesterov accelerated gradient

Momentum introduces a new problem, however. When the ball is rolling down the hill and reaches the bottom, its momentum is often quite high and since it is not self-aware it does not know to stop once the bottom of the hill (the minimum) has been reached. In some cases this causes the ball to overshoot the minimum

Stellenbosch University https://scholar.sun.ac.za

and continue rolling uphill, possibly causing it to miss the minimum completely. This problem was noticed by researcher Yurii Nesterov [23].

Nesterov accelerated gradient (NAG) is a way of giving the momentum term a little bit of foresight. We know that momentum term $v_{t+1} = \mu v_t$ will move the parameters θ. By computing $\theta - \mu v_t$ we will have a rough approximation of the next position of the parameters. This means we are effictively looking ahead by calculating the gradients with relation to the approximate future parameters instead of our current parameters.

$$\begin{aligned} v_{t+1} &= \mu v_t - \alpha \nabla J(\theta_t - \mu v_t) \\ \theta_{t+1} &= \theta_t + v_{t+1} \end{aligned} \tag{3.24}$$

The NAG update method was used in many of the experiments in this study, as it proved to work very well on our data.

3.2.7 AdaGrad

Adagrad was introduced by Duchi et al. in their 2011 paper "Adaptive subgradient methods for online learning and stochastic optimization" [8].

Adagrad allows the learning rate α to scale differently for every parameter at every time step based on the history of the gradients. This is done by simply dividing the current gradient in the update step by the sum of the previous gradients.

$$\begin{aligned} g_{t+1} &= g_t + \nabla J(\theta_t)^2 \\ \theta_{t+1} &= \theta_t - \frac{\alpha \nabla J(\theta_t)}{\sqrt{g_{t+1} + \epsilon}} \end{aligned} \tag{3.25}$$

The main benefit of AdaGrad is that it eliminates the need to manually tune α. The biggest disadvantage of AdaGrad is that the learning rate is always decreasing and decaying. This can cause the learning process to become extremely slow and even stop completely after extended training.

Stellenbosch University https://scholar.sun.ac.za

3.2.8 RMSProp

RMSProp was first introduced by Geoffrey Hinton in his undergraduate course at the University of Toronto, and even though it is well-known and implemented in many deep learning libraries, it was never released as a publication. Geoffrey Hinton himself asked that citations should be made to his undergraduate course slides [14].

The difference between RMSProp and AdaGrad is that the g_t term is calculated using an exponentially decaying average instead of a sum.

$$g_{t+1} = \gamma g_t + (1 - \gamma)\nabla J(\theta_t)^2 \tag{3.26}$$

In Equation 3.26, g_t is referred to as the second order moment of $\nabla J(\theta)$. A first order moment, m_t can also be introduced.

$$m_{t+1} = \gamma m_t + (1 - \gamma)\nabla J(\theta_t) \tag{3.27}$$

We then also add momentum,

$$v_{t+1} = \mu v_t - \frac{\alpha \nabla J(\theta)}{\sqrt{g_{t+1} - m_{t+1}^2 + \epsilon}} \tag{3.28}$$

and then finally update the weights, θ, as before,

$$\theta_{t+1} = \theta_t + v_{t+1} \tag{3.29}$$

3.2.9 AdaDelta

AdaDelta [41] is an extension of AdaGrad which tries to remove the problem of the decaying learning rate. Instead of accumulating all past squared gradients, it restricts the window of accumulated past gradients to a fixed size. It also uses an exponentially decaying average of g_t. However, it does not use the traditional learning rate α as used in Equations 3.23 and 3.24 . Instead, it introduces x_t, the

Stellenbosch University https://scholar.sun.ac.za

second moment of v_t.

$$\begin{aligned} g_{t+1} &= \gamma g_t + (1-\gamma)\nabla J(\theta)^2 \\ x_{t+1} &= \gamma x_t + (1-\gamma)v_{t+1}^2 \\ v_{t+1} &= -\frac{\sqrt{x_t+\epsilon}\nabla J(\theta_t)}{\sqrt{g_{t+1}+\epsilon}} \\ \theta_{t+1} &= \theta_t + v_{t+1} \end{aligned} \tag{3.30}$$

3.2.10 Adam

Adam is a more recent update rule that was first purposed by D.P. Kingma and J.L. Ba in their 2014 paper "Adam: A Method for Stochastic Optimization" [18]. Adam is an abbreviation of Adaptive Moment Estimation. Like AdaDelta, Adam also computes adaptive learning rates for each parameter in addition to storing an exponentially decaying average of past squared gradients. It also stores an exponentially decaying average of past gradients, similar to momentum.

$$\begin{aligned} m_{t+1} &= \gamma_1 m_t + (1-\gamma_1)\nabla J(\theta_t) \\ g_{t+1} &= \gamma_2 g_t + (1-\gamma_2)\nabla J(\theta_t)^2 \\ \hat{m}_{t+1} &= \frac{m_{t+1}}{1-\gamma_1^{t+1}} \\ \hat{g}_{t+1} &= \frac{g_{t+1}}{1-\gamma_2^{t+1}} \\ \theta_{t+1} &= \theta_t - \frac{\alpha\hat{m}_{t+1}}{\sqrt{\hat{g}_{t+1}+\epsilon}} \end{aligned} \tag{3.31}$$

The values γ_1 and γ_2 are also commonly know as the beta1 ($\beta 1$) and beta2 ($\beta 2$) parameters. Typical values for $\beta 1$ and $\beta 2$ are 0.9 and 0.999 respectively.

In practice Adam performs favourably compared to other learning methods. It usually converges quickly and rectifies most of the problems exhibited by other learning methods such as the vanishing learning rate or slow convergence.

Adam was used often to train neural networks in this study.

Stellenbosch University https://scholar.sun.ac.za

3.3 Recurrent Neural Networks with Long-short Term Memory

One limitation of the feed-forward neural networks (FFNNs), that we have been using to this point is that they have no concept of memory other than the weights that they have learned. Their behaviour is solely dependent on the current input example. This means that these models can sometimes struggle to make good predictions for time-dependent data. Recurrent Neural Networks (RNNs) try to address this by taking as input not only the current example, but also what has been seen in the past.

3.3.1 Understanding RNNs

Like many machine learning algorithms, RNNs are not a new concept and were already being used in the 1980s. They have, however, only recently started to achieve convincing success, thanks to the growth in computational power, the availability of massive datasets, as well as the invention of Long Short-Term Memory (LSTM) in the late 1990s.

By having internal memory, RNNs are able to remember important observations that they have seen in the past to help them better predict what is coming next. Figure 3.6 presents a simple illustration of the difference between a FFNN and in a RNN.

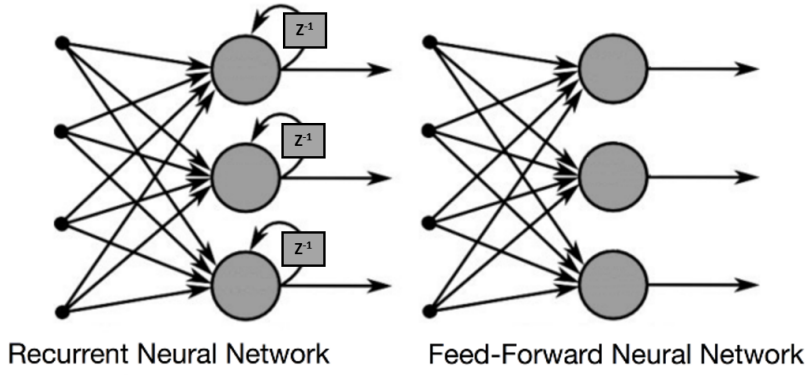


Figure 3.6: Information flow in a Recurrent Neural Network (RNN) and in a Feed-Forward Neural Network (FFNN).

The FFNN learns weights that link the inputs to the outputs. In contrast, the

Stellenbosch University https://scholar.sun.ac.za

RNN learns weights not only for the current input, but also for previous neuron outputs. In Figure 3.6, $z^{-}1$ indicates a one step delay. RNNs are also trained by gradient descent using an algorithm known as backpropagation through time.

Backpropagation through time (BPTT) can be understood as backpropagation applied to an "unrolled" RNN. An unrolled RNN is a way of visualising the processing of sequential inputs as a series of neural networks, as illustrated in Figure 3.7. Figure 3.7 shows an unrolled RNN for T consecutive input vectors $\underline{x}_0 ... \underline{x}_t$.

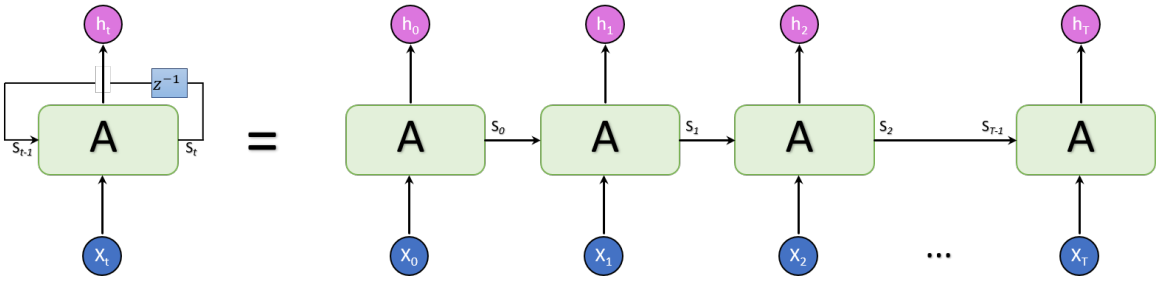


Figure 3.7: A RNN and it's unrolled form for an input sequence consisting of T consecutive input vectors $\underline{x}_0 ... \underline{x}_t$ [27].

In this section, the notation used will be the following: h_t refers to the output of the unfolded RNN at time t and h_{t-1} to the output from the previous time step. The parameter x_t is the input to the network at time step t. S_t is the concatenation of the values h_t and C_t which is the network output and the cell state at time step t, respectively. The cell state, C_t, will be explained in the following section.

In BPTT, the error is propagated backwards from the output at the last time step h_t to each input $\underline{x}_0 ... \underline{x}_t$. By unrolling the RNN, it becomes clear that the error at any given time step depends on the error at previous time steps. Once the error has been calculated for every time step, the weights can be updated. If there are a large number of timesteps, BPTT can be very computationally expensive.

The reason why RNNs were initially not successful is due to two problems encountered during BPTT training, namely vanishing gradients and exploding gradients. Exploding gradients occur when the model assigns an extremely high weight to one or more parameters, usually because of a long chain of multiplications during backpropagation. Fortunately, there is an easy way to deal with exploding

Stellenbosch University https://scholar.sun.ac.za

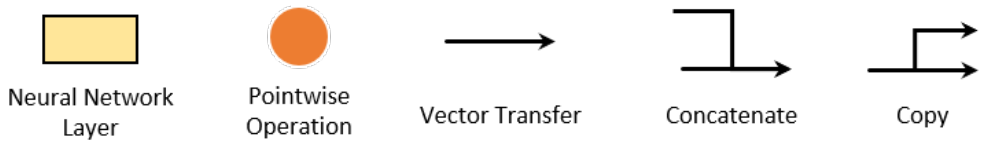
gradients - by clipping the gradients at some threshold [27].

Vanishing gradients occur when the value of a gradient becomes too small, causing the model to stop learning or to learn extremely slowly. This phenomenon is usually due to a long chain of multiplications of small gradients. This is a much harder problem to solve than exploding gradients since one cannot simply truncate the gradients when they get too small. LSTMs provide a solution to this problem.

3.3.2 Long Short-Term Memory

The Long-Short Term Memory (LSTM) recurrent neural network was proposed by Sepp Hochreiter and Juergen Schmidhuber in their 1997 paper titled "Long Short-Term Memory" [16]. The crucial difference between LSTMs and regular RNNs is their use of gated memory, that sidesteps the vanishing gradients problem and therefore allows them to remember important information for longer. This makes them much better at learning from data in which important events happen with long time delays in between.

In this section the following notation will be used for the graphical explanations:



Each line in these illustrations carries an entire vector, from the output of one node to the inputs of others. The orange circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Line merging denotes concatenation, while line forking denotes its content being copied and the copies going to different locations.

LSTMs are able to remember their inputs over a long timespan, because they save information using a process that is similar in some respects to computer memory, incorporating the notions of read, write and delete. The network will consider its input and assign an importance factor to the information. Based on this, it will decide whether it wants to store or delete information. The importance

Stellenbosch University https://scholar.sun.ac.za

factors are governed by weights that are trained like all the other weights. Over time, the network will learn to recognise whether information is important or not.

All RNNs are structured as a chain, with repeating modules. These modules have a very simple structure, like a single tanh layer, in standard RNNs. In Figure 3.8, a chunk of a neural network, A, looks at some input x_t and outputs a value h_t.

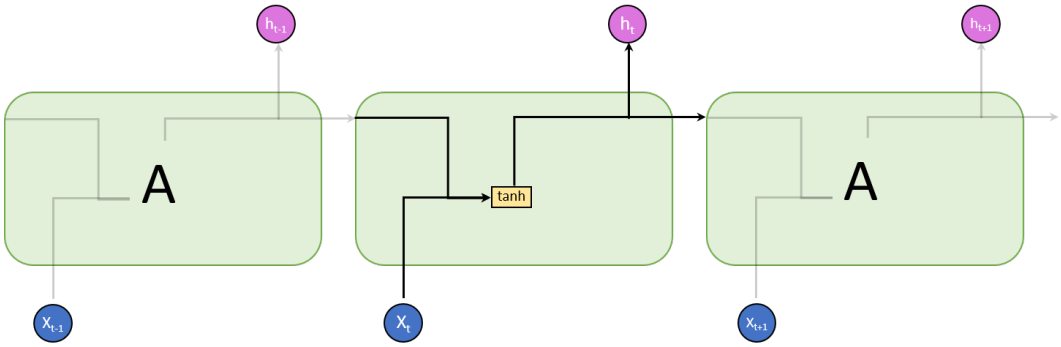


Figure 3.8: A single tanh unit in the repeating module of a standard RNN.

In a LSTM, we also see this chain-like arrangement, but the internal structure of the module is different from the standard RNN. Instead of the single (tanh) layer, LSTMs have four layers interacting in a very particular way [27].

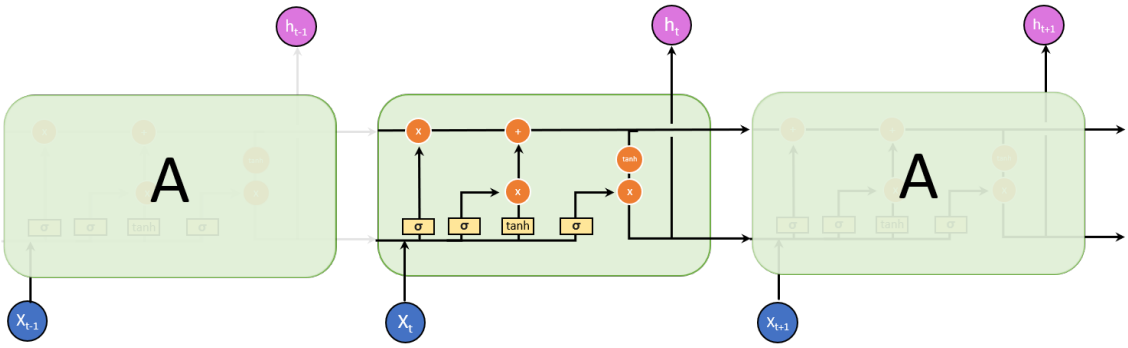


Figure 3.9: The four layers in the repeating module of a standard LSTM.

A key aspect of the LSTM is the conveyor-belt-like property called the cell state. It runs through the entire chain with only some minor external interactions along the way. In Figure 3.10 below, one can see C_t representing the cell state at timestep t.

Stellenbosch University https://scholar.sun.ac.za

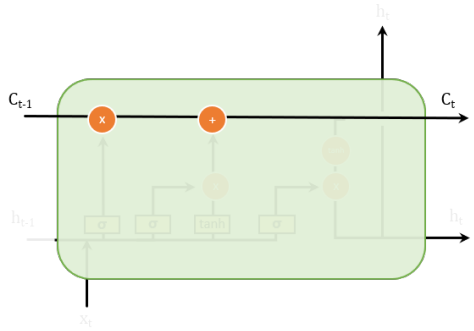


Figure 3.10: The LSTM cell state running from time step $t-1$ to t.

The process of removing or adding information to the cell state is carefully regulated by small sigmoid layers followed by a point-wise multiplication operation. These structures are called gates, as illustrated in Figure 3.11.

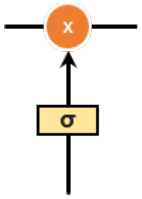


Figure 3.11: The LSTM gate unit consisting of a sigmoid layers as well as a point-wise multiplication operation.

Since the sigmoid unit always has an output between 0 and 1, the gate can control how much of each component of the cell state it should let through. A LSTM has three of these gates to control the cell state. These gates are called the input, output and forget gates respectively.

3.3.3 LSTM Walk Through

The first thing an LSTM cell does is decide what information from the previous cell state it will keep, and what it will discard. It bases this decision on the values of h_{t-1} as well as x_t, and provides a number, f_t, between 0 and 1 for each element of the cell state vector C_{t-1}. This constitutes the "forget gate layer".

In a real-world example, the LSTM might be trying to predict the next word in a sentence. In this case the cell state might include the gender of the current subject. When it sees a new subject, however, it has to forget about the previous one.

Stellenbosch University https://scholar.sun.ac.za

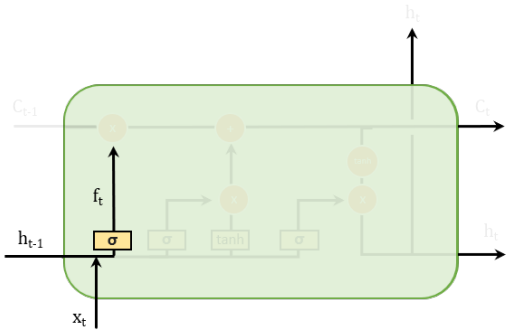


Figure 3.12: The LSTM forget gate consists of a sigmoid layer as well as a point-wise multiplication operation.

The equation that governs this behaviour is,

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t]) \tag{3.32}$$

where W_f are the weights of the forget gate and $[h_{t-1}, x_t]$ denotes the concatenation of vectors h_{t-1} and x_t [27].

Next, the LSTM cell must decide what new information is worth adding to the cell state. This requires two steps. First, an "input gate layer" decides which components i_t of the state vector to update with a sigmoid function. Next, a vector of new candidate values, $\tilde{C}_t$, that will be added to the state is generated.

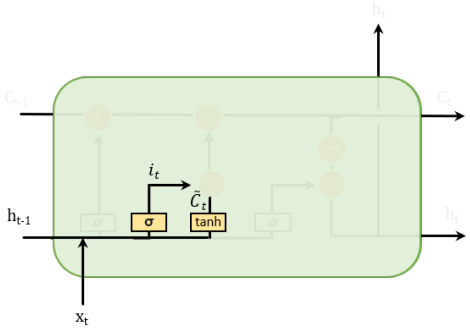


Figure 3.13: The second step of the LSTM, consists of a sigmoid input gate and a tanh to update the gated state vector components.

In mathematical terms,

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, x_t]) \\ \tilde{C}_t &= tanh(W_C \cdot [h_{t-1}, x_t]) \end{aligned} \tag{3.33}$$

Stellenbosch University https://scholar.sun.ac.za

with W_i and W_C representing the weights of the input gate and candidate update layer respectively, while i_t are the gating decisions.

After the components of the state vector that should be updated have been identified, the cell state, C_{t-1}, is updated to give the new cell state, C_t. To achieve this, the old cell state vector is multiplied element-wise by f_t to achieve forgetting of the selected elements. Then the cell state is updated by the addition of $i_t * \tilde{C}_t$. The result is the updated state vector [27].

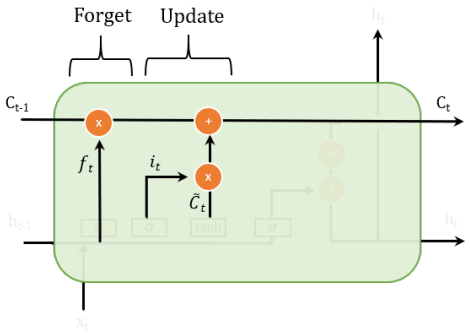


Figure 3.14: When the cell state of the LSTM is updated, forgetting is executed first, followed by updating with new candidate state values.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{3.34}$$

The final step is to determine the output from the state. First, the parts of the cell state that will affect the output are identified by a sigmoid layer. Next, the cell state C_t is passed through a tanh layer to scale the values between -1 and 1. Finally, these scaled values are multiplied element-wise with the sigmoid layer to determine the output h_t.

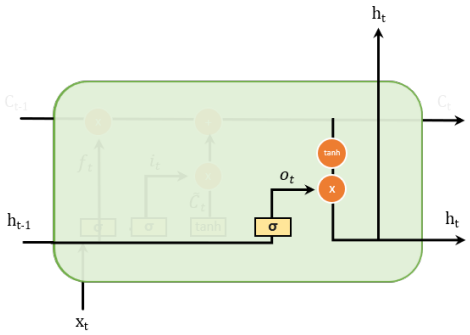


Figure 3.15: Determining the output of the LSTM unit from its state, previous output and current input.

Stellenbosch University https://scholar.sun.ac.za

$$
\begin{aligned}
o_t &= \sigma(W_o \cdot [h_{t-1}, x_t]) \\
h_t &= o_t * tanh(C_t)
\end{aligned}
\tag{3.35}
$$

This concludes our description of a single step of a regular LSTM unit. In practice, variants of the basic LSTM model are often used. However, these variants follow a similar logic.

3.4 Summary and conclusion

This chapter has presented a brief introduction to the methods that will be applied to our datasets. Linear regression, feedforward neural networks and recurrent neural networks - specifically long-short term memory (LSTM) - were considered. The way in which cross-validation will be used to train and evaluate these models was also described.

Stellenbosch University https://scholar.sun.ac.za

Chapter 4

Datasets

Two different sets of data were used in this work. One was available at the outset, while the second became available later.

Firstly, we were provided with a dataset that consisted of soil temperatures, ambient temperatures and humidity, as well as local weather station data. This dataset originated from a particular vineyard in Stellenbosch and will be referred to as the Stellenbosch data or "Stb. data".

Later, we were provided with a second dataset that consisted of soil temperatures, ambient temperatures, humidity and rainfall from a local weather station, as well as measurements of the plant phenology (growth stages), micro climate, water status, vegetative data, reproductive data, and physiological data. This dataset originated from a particular vineyard in Somerset West and will be referred to as the "Ssw. data".

Stellenbosch University https://scholar.sun.ac.za

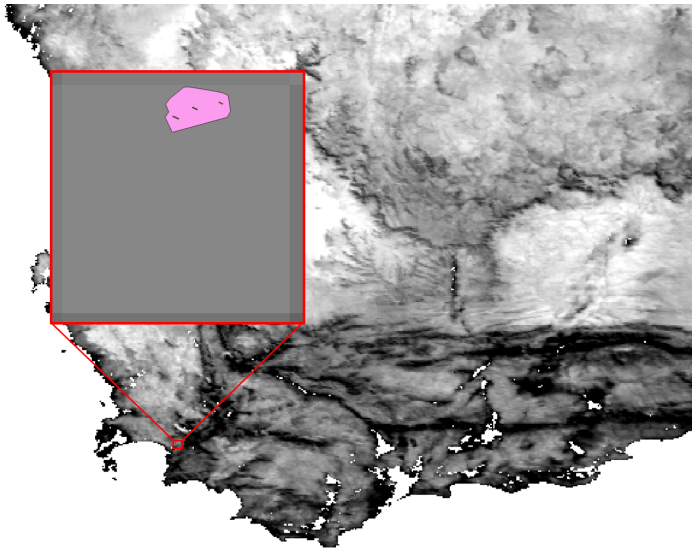


Figure 4.1: A map indicating the location of the Somerset West vineyard. The block in red is a zoomed in image showing the shape of the site. The area in red roughly equals 1 km^2.

Unfortunately, in both of the datasets, some data was incomplete or corrupt and hence only a portion was suitable for the application of machine learning algorithms. The data we took particular interest in are the soil temperatures, the weather station data, the ambient sensor data, the moisture data, and some dates termed the "bud-burst dates". We also retrieved some land surface temperature (LST) data from the MODIS satellite. These datasets will briefly be discussed in the following sections.

4.1 Soil Data

This consists of two different datasets obtained on different dates. The first consists of temperatures measured in the soil of a vineyard in Stellenbosch and will henceforth be referred to as the Stellenbosch soil data, or "Stb. soil". The second was obtained at a much later date and consists of temperatures measured in the soil of a vineyard in Somerset West. It will hereafter be referred to as the Somerset West soil data, or "Ssw. soil".

Most of the experiments we present first were done using the Stb. soil data and most of the experiments we present later were done on the Ssw. soil data,

Stellenbosch University https://scholar.sun.ac.za

since the Ssw. dataset is larger.

4.1.1 Stellenbosch (Stb.) soil data

This dataset contains temperature readings taken at four different depths within the soil and in different vine blocks. The measurements were taken within a single vineyard over an area consisting of 4 rows and 5 blocks. A block is a 9 m stretch consisting of six separate vine-plants. The two outer vines act as buffers while the inner four are used for measuring. The depths considered were 5 cm, 10 cm, 20 cm, and 40 cm. Each block was subject to a different mulch depth added on top of the soil. These treatments are labelled T1-T5, with T1 corresponding to the least amount of mulch and T5 the largest amount of mulch. These mulch depths are listed in Table 4.1

Treatment Label	**Mulch Depth**
T1	Control (no mulch)
T2	2 cm
T3	4 cm
T4	8 cm
T5	16 cm

Table 4.1: Treatment labels and their respective amount of mulch for the soil data.

The experimental layout is shown in Figure 4.2. Each square represents a single block, and is labelled with the treatment type.

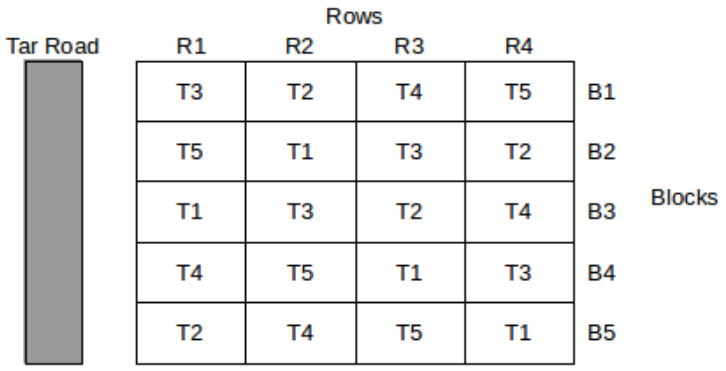


Figure 4.2: Experimental layout for the Stb. soil data.

In the raw data, temperatures are listed for each row, block, and depth, while

Stellenbosch University https://scholar.sun.ac.za

the treatment type is inferred from Figure 4.2. In total there are therefore 80 columns (4 rows, 5 blocks, 4 depths) for each line of raw data, and each line corresponds to a specific time and date.

The soil data stretches from 10:20 AM on 2016/09/28 to 09:32 AM on 2017/02/28. Measurements were taken of irregularly-spaced intervals of time, with roughly an hour (or a little more) between measurements. There are missing measurements throughout the period. In particular, there is a relatively large gap in the data from 2016/12/07 to 2016/12/19 where there are no entries. There are also 160 measurements indicated as "NA" scattered throughout the data which indicates either missing or corrupted data-points.

Finaly, there is a large block of data, stretching from 2016/10/12 to 2016/10/24, in which all entries for row 2 (413 rows * 20 columns = 8260 entries in total) are given as -127. This most likely indicates a faulty sensor and should therefore also be regarded as missing data and be disregarded for calculations.

4.1.2 Somerset West (Ssw.) soil data

Just like the Stellenbosch soil data set, this dataset contains temperature readings taken within the soil. In this dataset soil temperatures were measured at three distinct locations on a hill, namely low-vigour (top of the hill, least amount of moisture), medium-vigour (middle of the hill, medium moisture), and high-vigour (bottom of the hill, highest moisture). Unfortunately, data was only available for the low-vigour (LV) and medium-vigour (MV) locations. In each case, temperatures were measured at three different depths: 0-15 cm, 15-30 cm, and 30-60 cm. The measurements were made at 15 minute intervals.

Data collection took place from 2012-11-26 to 2016-12-06. However, there are many gaps in the data between 2014 and 2016. Moreover, the measurements taken at the 0-15 cm depth as well as the 30-60 cm depth were seen to become faulty by early April 2014.

Stellenbosch University https://scholar.sun.ac.za

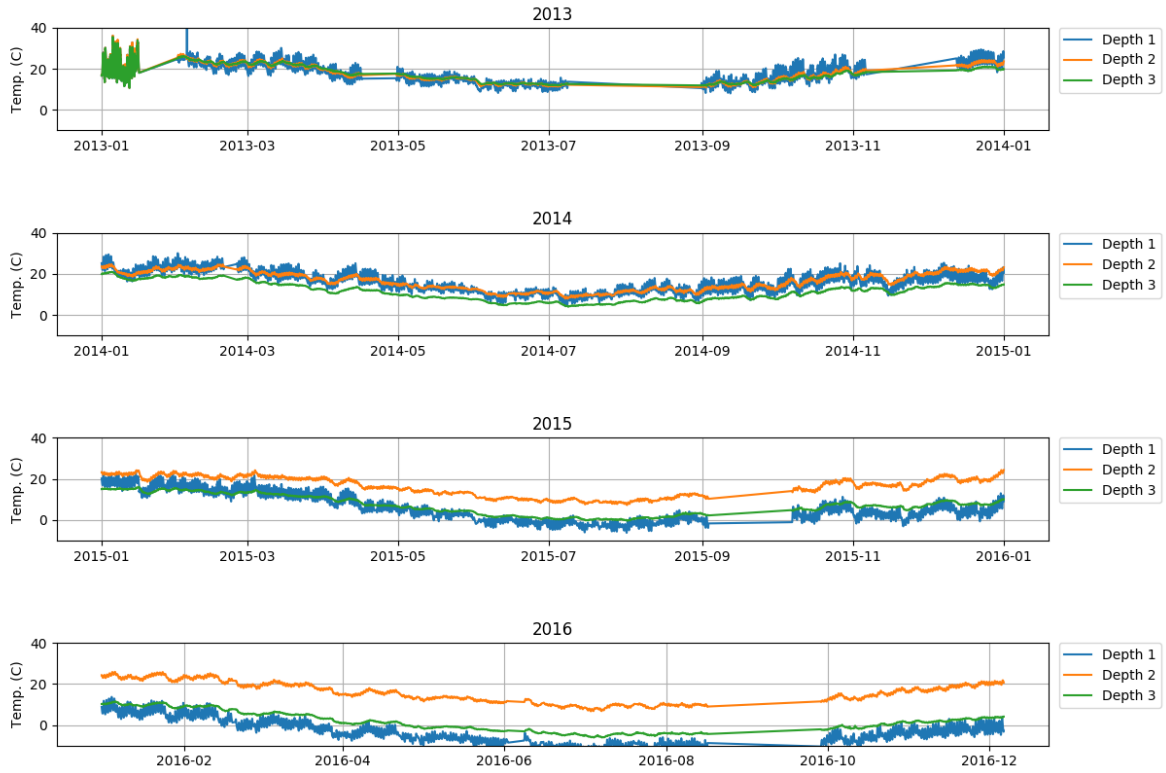


Figure 4.3: An extract of the soil temperatures at three different depths (0-15 cm, 15-30 cm, 30-60 cm) for the medium vigour Somerset West vineyard. It is evident that by April 2014 the measurements for the 0-15 cm and the 30-60 cm depths begin to diverge from the measurements at the 15-30 cm depth, indicating a fault in the measurement system.

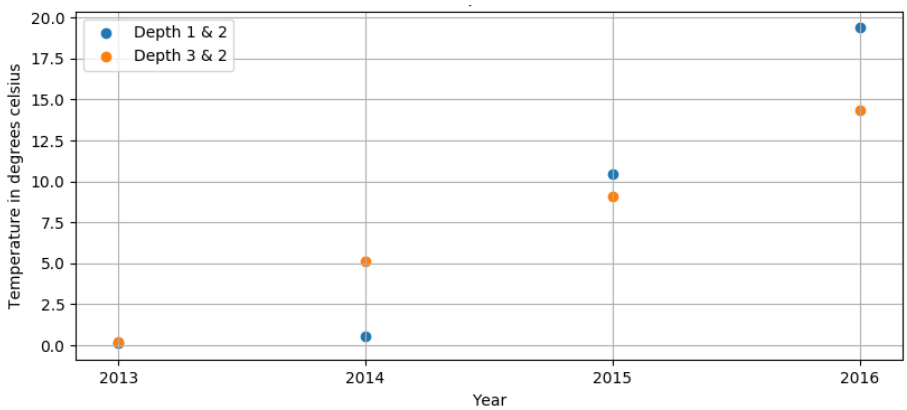


Figure 4.4: The average difference in temperature measurements at different depths for the medium vigour Ssw. soil data. Taking depth 2 (15-30 cm) as a baseline, we can see that in 2013 all three measurements show approximately the same temperature. From 2014 onward, however, average measured temperatures at depths 1 and 3 start to differ greatly from depth 2, indicating a fault.

Stellenbosch University https://scholar.sun.ac.za

Hence, it was decided to only use the first 40000 entries of the dataset. This corresponds to a little more than one year's worth of data (2012-11-26 to 2014-02-12). This is still substantially more than is available in the Stellenbosch soil dataset. In particular, now we have data for all four seasons of the year and an algorithm can learn from a wider spread of data.

4.2 Mesoclimate Data

The mesoclimate datasets contain temperature and humidity measurements taken by sensors placed in the canopy of the vineyard. This means that the mesoclimate data is geographically co-located with the soil temperature data. The readings were taken at 30 minute intervals and seem to be complete (i.e. there are no gaps in the data).

4.2.1 Stellenbosch mesoclimate data

The Stellenbosch mesoclimate data stretches from 12:30 PM on 2015/09/18 to 9 AM on 2017/03/08. Measurements are provided every 30 minutes. There are no missing entries except for a large block from 2016/10/28 to 2016/12/07. This dataset will henceforth be referred to as the Stellenbosch mesoclimate data, or "Stb. meso".

4.2.2 Somerset West mesoclimate data

The Somerset West mesoclimate data spans from 2012-11-26 to 2016-04-04. Although there are still several gaps in the data, no faults in the measurements were suspected. The data is relatively complete up to 2014-02-12, meaning that there is a good degree of overlap with the soil temperature data. Figure 4.5 illustrates this.

Stellenbosch University https://scholar.sun.ac.za

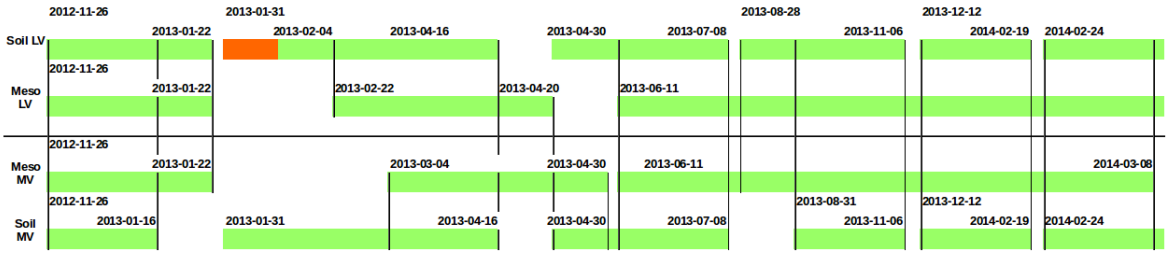


Figure 4.5: A timeline indicating the overlap between the available low-vigour (LV) and medium-vigour (MV) soil and the mesoclimate data for the Somerset West vineyard. The data needs to overlap for it to be useful for training a machine learning model. We see that the data overlaps relatively well, especially the LV data. The orange block indicate some faulty soil measurements.

One small fault was however detected with the mesoclimate-sensors. Sometimes the moisture builds up between the vines and cannot escape, causing the humidity measurement to reach 100%. This is incorrect, since relative humidity in the air is very unlikely to reach 100%. To address this problem, the humidity readings from the mesoclimate sensors were compared with those of the closest local weather station (weather station data discussion to follow in the next section). If the measurement was either greater than 95% relative humidity (RH) or it differed from the weather station reading by more than 30% RH, the weather station reading was used to replace that entry.

4.3 Weather station data

In addition to the measurements taken in the vineyard itself, data was also available from weather stations located on both farms on which the vineyards are located. These weather stations measured the air temperature, humidity, wind direction (0-360) and rainfall.

The weather station data from the Stellenbosch farm stretches from 2012/09/16 to 2016/12/07. The data was measured every hour and there are few missing entries.

The weather station data from the Somerset West farm stretches from 2007-01-01 to 2014/06/25. Measurements were taken every hour and there are almost no missing entries. This means we have access to weather station data across the

Stellenbosch University https://scholar.sun.ac.za

whole range of usable soil temperatures for the Somerset West data.

4.4 Stellenbosch Data overlap

The measurements in the three different Stb. datasets all span different periods of time, with some overlap as shown in Figure 4.6. These overlapping portions will be the most useful.

Because the Stb. weather station data ends at 2016/12/07 and because of the missing mesoclimate data, there is only one month of data where all three datasets overlap. This is from 2016/09/28 to 2016/10/28.

If we consider only the soil data and the mesoclimate data, the overlap is larger and stretches from 2016/09/28 to 2016/10/28 and from 2016/12/19 to 2017/02/28 (a total of 3 months and 11 days). Since the mesoclimate data closely corresponds to the weather station data, it may be defensible to omit the latter.

However, not all the data in this 11 month and 3 day period is usable, since we have not yet taken into account corrupt soil data entries indicated as "NA" or "-127". After removing these entries, and also picking out the mesoclimate entries that are closest to each soil data entry (so that each set has the same number of entries) we are left with 1557 usable sets of measurements. This means that 1557 time-aligned (same time/date) soil temperatures are known for every row/block/depth combination as well as the air temperature and humidity at the same time and dates are known.

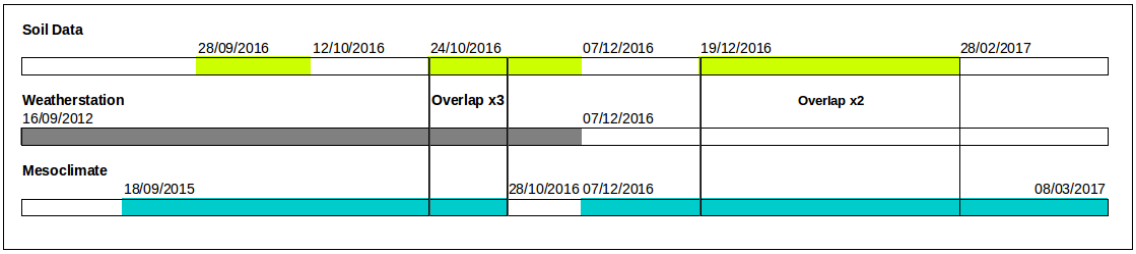


Figure 4.6: Visual guide to the data overlap for the Stellenbosch vineyard.

Stellenbosch University https://scholar.sun.ac.za

4.5 Soil moisture data

Soil moisture measurements were gathered using a neutron probe. Neutron probes are able to take very accurate readings of soil moisture content. The probe consists of a nuclear unit, serving as a neutron source as well as a detector, a housing to contain the electronic receptors, as well as a shield to safely transport the radioactive device.

The nuclear unit is lowered down an aluminium access tube where the neutron source will start scattering fast neutrons. These neutrons get deflected by hydrogen, which is most commonly found in water particles underground, and are slowed down. The source then detects and counts the returning slow neutrons. The number of returning slow neutrons counted is directly related to the moisture content in the soil [5].

Our dataset only consists of a few weekly measurements of the soil moisture content. In our dataset, these values are typically in the range of 4000 to 8000 counts [32] with an average around 6600 counts.

These measurements are made at three different depths: 30 cm, 60 cm, and 90 cm. The data-set stretches from 2013-02-21 to 2014-12-01, however, the dataset only contains 214 entries in total since the moisture was only measured once a week for approximately three months in the start of the year and then again for another three months at the end of the year. For example, in 2014 there are six measurements made between January and March at the three depths and in low-, medium-, and high-vigour blocks. That gives $6 * 3 * 3 = 54$ entries for those three months and then another 78 entries at the end of the year between August and December.

4.6 Satellite LST data

In 1999, NASA launched a satellite named Terra-1. This satellite was specifically designed to monitor change in the nature of the earth's ecosystem at a global level. Terra collects data about the earth using five different sensors integrated into the satellite. These five sensors observe the oceans, land surface, atmosphere, snow

Stellenbosch University https://scholar.sun.ac.za

and ice and energy budget of the earth [37].

The Terra satellite images the entire surface temperature of the earth on a daily basis using Moderate-Resolution Imaging Spectroradiometer (MODIS) sensors. The MODIS sensor has a good balance between temporal and spatial resolution (daily measurements at a $1km^2$ resolution). This, together with the fact that the data is freely available, makes it a very useful data source for land surface temperatures. Terra operates on a descending orbit, meaning that the satellite travels from north to south over the Earth's surface. The Terra satellite has a mean equatorial crossing time at 10:30 am.

Land surface temperature (LST) combines the results of surface-atmosphere interactions and energy fluctuations between the atmosphere and the ground, indicating how hot the surface of the earth would feel to the touch averaged over $1km^2$ tiles [40].

A major constraint of the MODIS LST measurements is that they are only available in clear sky conditions, since the visible and thermal infra-red spectral ranges cannot penetrate clouds [33]. Land surface temperatures in cloud-covered areas are simply not available since temperatures at the top of the clouds are then measured instead.

The satellite captures data at our location of interest twice a day: once late in the morning, and once again at night. The data is downloaded as ".HDF" files, which are then converted to normal ".tif" files using an open source application called the MODIS re-projection tool (MRT). Each pixel of the file represents a $1km^2$ tile of the earth's surface and allows the temperature to be determined using Equation 4.1 below. This converts each pixel of the image to a temperature in degrees Celsius. A short python script was written to process all downloaded MODIS files and extract the temperature for the pixel covering our desired location.

$$Temp.\ in\ celsius = Pixel\ value * 0.02 - 273.15 \tag{4.1}$$

The data we needed would stretch over the same span of time as the soil data, which is from 2012-11-26 to 2014-02-12. A day-time and a night-time measurement was available within every 24 hour period. Unfortunately, many data points are

Stellenbosch University https://scholar.sun.ac.za

missing because of the cloud contamination discussed earlier.

4.7 Graphical representation

Selected portions of the Stellenbosch data were plotted as 2D graphs in order to develop an intuition for the trends and patterns in the measurements. Two types of plots were made. One takes a specific row/block combination and plots the temperature measurements at all four soil depths and the ambient temperature (from the mesoclimate data) against time. Figure 4.7 shows one such plot.

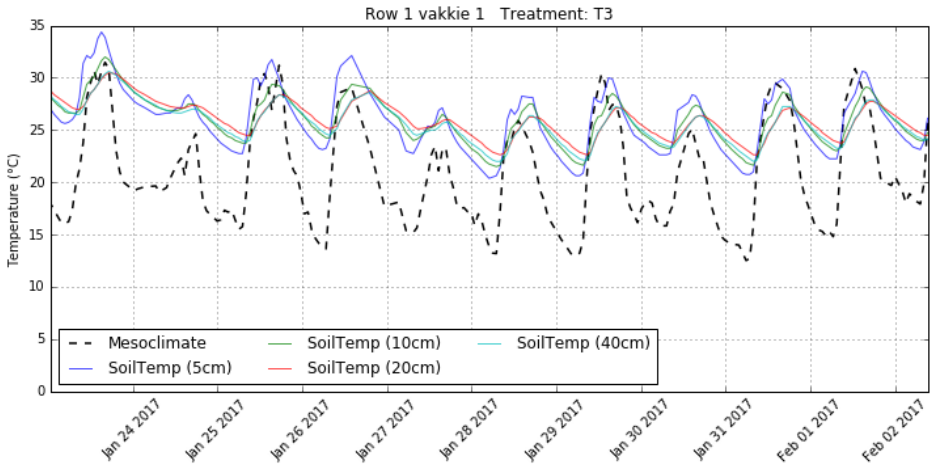


Figure 4.7: Temperature at various depths and ambient temperature vs time (200 data points).

Figure 4.7 shows only 200 data points to make it easier to see the trends in the data. Similar behaviour is seen over other time periods.

The second type of plot considers a specific row and depth and plots the temperature measurements for different treatment types and the ambient temperature against time. Figure 4.8 shows such a plot.

Stellenbosch University https://scholar.sun.ac.za

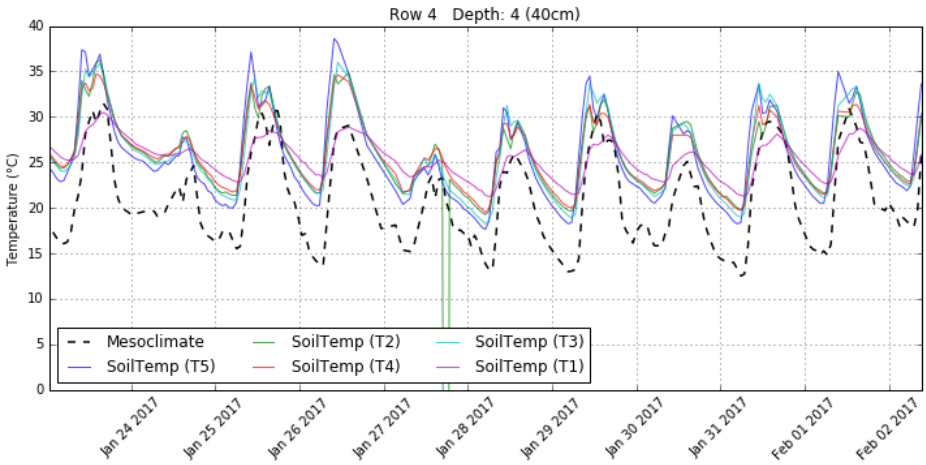


Figure 4.8: Temperature for row 1 and depth 4 for different treatment types as described in Table 4.1 versus time (200 points).

The various maxima and minima on the graphs correspond to the daily highest and lowest temperatures as observed during the day and night respectively. The sudden dip on Jan 27 in Figure 4.8 is an example of a corrupt data point (-127) as discussed earlier.

It can also be observed from Figure 4.7 that the deeper in the soil we measure, the less the temperature varies. This is to be expected since the soil should stabilise temperatures better than air.

4.8 Mean squared error calculation

It is hard to determine the correspondence between different measurements from visual inspections of the data alone. A more objective approach is to calculate the mean squared error (MSE). As a first analysis, we calculated the MSE between the ambient temperature and the temperatures measured for each different combination of depth and treatment type for the Stellenbosch soil data. The results are shown in Table 4.2.

This table shows the mean squared error between the ambient temperature and every depth/treatment type combination, normalised by the smallest value to make it easier to interpret. The higher the MSE, the less the soil temperature follows the ambient temperature. Hence the highest values indicate the most

Stellenbosch University https://scholar.sun.ac.za

	Mulch Depth (treatment type)				
Sensor Depth	**0 cm**	**2 cm**	**4 cm**	**8 cm**	**16 cm**
5 cm	1.098	1.15	1.117	1.07	1
10 cm	1.39	1.328	1.19	1.41	1.23
20 cm	1.52	1.64	1.537	1.7	1.54
40 cm	1.61	1.785	1.696	1.76	1.77

Table 4.2: Mean squared error (MSE) calculated over time between the ambient air temperature and the soil temperature.

stable soil temperature.

We see that the deeper the sensor is located, the higher the MSE and the more resistant the soil temperature is to a change in the ambient temperature. This makes sense since soil should retain its temperature better than the air outside.

The results are however surprising when considering the mulch depth, which we expected would dampen the temperature changes in the soil below. However, it seems that the mulch depth has almost no effect on the change of temperature since no trend is observable for the MSE at different mulch depths. This could be due to a variation in the true amount of mulch due to the practical difficulty of ensuring a uniform depth. However, it may also simply indicate that a mulch covering does not substantially affect soil temperature.

4.9 Bud-burst dates

One final set of data we received for the Somerset West vineyard that was of interest as a machine learning application are the bud-burst or bud-break dates.

Every winter, when the temperatures start to drop, the grapevines go into a dormant (hibernation-like) state during which they drop their leaves and start to undergo a number of processes in preparation for the cold temperatures of the winter. Vines set themselves up with the biological equivalent of 'anti-freeze' to ensure they survive the winter [17].

Once the temperatures have risen sufficiently in spring time, the vines emerge from their dormancy. They will start to grow small buds and the buds will then

Stellenbosch University https://scholar.sun.ac.za

give rise to new leaves and flowers. The bud-burst dates recorded are the dates when these initial buds start to appear. These dates are useful since they indicate that the growth stage of the vine has begun and it becomes possible to estimate how long it will be until the vines reach their next important stages of growth.

Since bud-burst happens only once per year, this dataset consists of only 15 entries. The entries consist of one date per year per vigour level (Low, medium, or high) for 2012 to 2016. Since we do not have corresponding data for the high-vigour soil/ambient, only 10 of those entries are usable.

4.10 Conclusion

This chapter has described the datasets that will be used for experimentation. One, smaller, dataset is available for a vineyard in Stellenbosch, and one, somewhat larger, dataset is available for a vineyard in Somerset West. In both cases soil temperatures at various depths, ambient temperatures and ambient humidity are available. In addition some soil moisture content measurements and bud-burst dates are available for the Somerset West dataset.

Stellenbosch University https://scholar.sun.ac.za

Chapter 5

Experiments with Stb. Dataset

This chapter describes experiments performed using the Stellenbosch vineyard dataset. Experiments using the Somerset West dataset will be described in the following chapter.

5.1 Linear Regression

Linear regression was applied to determine whether some measurements in the dataset could be inferred from others.

Three sets of independent experiments were performed. Firstly the model was provided with the soil temperature data as input and asked to predict the ambient temperature. Secondly, the model was given certain parts of the soil data and asked to predict other parts of the soil data. Lastly, and arguably most importantly, the model was given ambient measurements and asked to predict the soil temperatures.

5.1.1 First experiment

In the first experiment the soil temperature data, as presented in Section 4.1.1 was used as training material, and the model was configured to predict the ambient temperature of the corresponding time. Hence the model has 80 input variables, corresponding to the 4 rows, 5 blocks and 4 depths of measurements, and one output. On average, the model is able to estimate the ambient temperature to within **1.24 °C (8.14%)** of the true temperature. Figure 5.1 illustrates the typical

Stellenbosch University https://scholar.sun.ac.za

temperatures predicted by the model, with the error shown by shading. We see that the model tends to under-estimate the temperatures at most of the peaks and troughs.

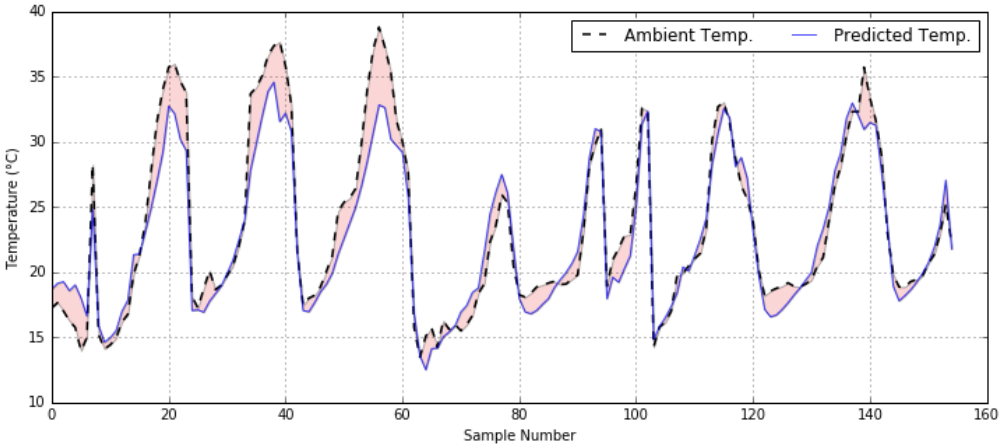


Figure 5.1: True and estimated ambient temperatures using linear regression and soil temperatures as input.

5.1.2 Second experiment

In the second experiment the model was given a certain subset of the soil data and asked to predict a different, non-overlapping, subset of the soil data. For instance, given the data for rows 1 and 4 at depths 1 and 4, predict the temperatures for row 3 at depths 1 and 4. A linear regression model will then be calculated for every output variable. For instance in the example mentioned above, a model will be determined for each of the 5 blocks in the two depths it is trying to predict. So 10 different models will be calculated in total (5x2).

Different configurations were tested in Experiment 2. It was usually possible to predict temperatures to within 3.5% of their true value. It was found that smaller errors could be achieved when more training data were available.

The addition of date and time as inputs was also evaluated. By including the month, day, hour, and hour squared value of each measurement, four new input variables were added. The inclusion of these parameters barely resulted in an improvement in accuracy, however. The improvement was greatest when the training data were few.

Stellenbosch University https://scholar.sun.ac.za

Table 5.1 shows a summary of some examples of the results achieved with this experiment. The table includes results from the model before and after the additional training inputs were added.

Table 5.1: Example results from experiment 2

Example Number	**1**	**2**	**3**
Rows input	1,4	2	2
Blocks input	1-5	1-5	1-5
Depths input	1,4	4	4
Total inputs	20	5	5
Predicted Rows	3	3	3
Predicted Blocks	1-5	1-5	1-5
Predicted Depths	1,4	1,4	4
Total predicted variables	10	10	5
Error with dates (°C)	0.59°C	0.4°C	0.54°C
Error with dates (%)	2.58 %	1.77 %	2.42 %
Error without dates (°C)	0.49°C	0.39°C	0.56°C
Error without dates (%)	2.17 %	1.76 %	2.55 %

One can see from the table that these extra input parameters rarely help the model to perform better, and even when it does perform better, the performance gain is very small.

5.1.3 Third experiment

For the final experiment, the model was configured to predict the temperatures in the different parts of the soil when given ambient temperature and humidity as well as time and date as inputs.

This however yielded poor results, achieving an average error rate of up to **2.7°C** or **12.29%** depending on the point at which the temperature was being

Stellenbosch University https://scholar.sun.ac.za

predicted. It was established that one of the reasons the model was performing so poorly was the format of the first two training parameters (the day of year and hour parameters). Due to the discontinuous cyclic nature of these two parameters, they can not have a linear relationship with the temperature. For example, the time experiences a discontinuity for 23H59 to 00H00 at midnight every day. This made it hard for the model to train on these parameters since it would struggle to find a linear relationship between these parameters and the temperature.

To address this the time and date inputs were incorporated in a different way. The hour of the measurement as well as the day of the year the measurement was made were first extracted from the time and date. These were then used to construct two features each as follows:

$$\begin{aligned} x[1] &= cos\left(\frac{dayOfYear * 2\pi}{365}\right) \\ x[2] &= sin\left(\frac{dayOfYear * 2\pi}{365}\right) \\ x[3] &= cos\left(\frac{hour * 2\pi}{24}\right) \\ x[4] &= sin\left(\frac{hour * 2\pi}{24}\right) \end{aligned} \tag{5.1}$$

Here $x[1]$ and $x[2]$ represent the day, and $x[3]$ and $x[4]$ the hour. Equation 5.1 maps the hour and day onto a unit-circle, thereby removing the discontinuity at 24H00/00H00 and 365/1 respectively. Specifically, 23H00 and 0H00 are close together in this representation, while they are not when the raw time of day is used. This causes the model to better understand the inputs leading to better prediction results.

The ambient temperature and humidity were added as inputs 5 and 6 ($x[5]$ and $x[6]$) to yield a six-dimensional input vector used to predict the soil temperatures using linear regression.

The model did quite well using this set of features. Figure 5.2 illustrates the prediction results for this experiment.

Stellenbosch University https://scholar.sun.ac.za

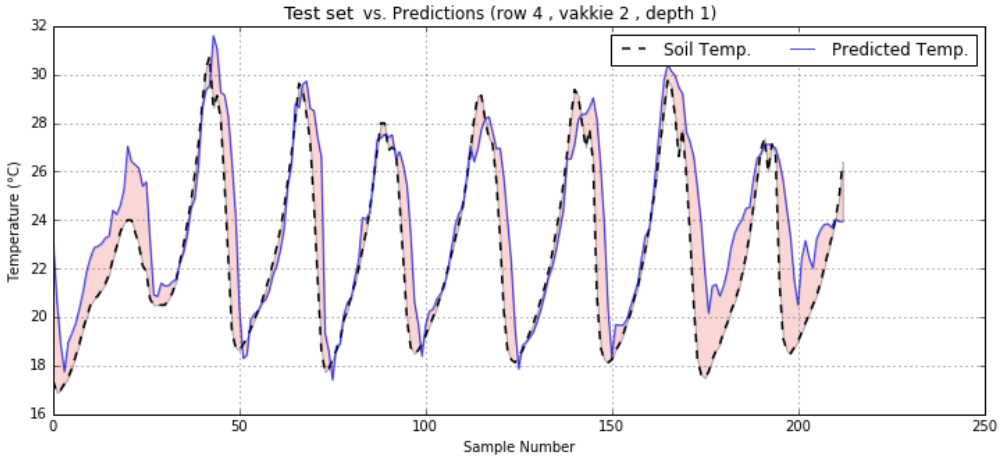


Figure 5.2: True and estimated soil temperatures at depth 1 for row 4, block 2 when using polar transformed features for hour and day inputs.

The figure above shows the model attempting to predict soil temperatures at row 4, block 2, depth 1. The error for this specific case is 1.57 °C on average (or 7.85%). The overall average error over all soil depths is **1.26 °C** (or **5.56%**) with a standard deviation of 0.13 °C between the averages.

The log value of the ambient temperature was also considered as an additional feature, but did not lead to a further reduction in error.

5.2 Neural Networks

Linear regression is a simple model that provides a good baseline. However, it can model only linear dependencies between the inputs and outputs. Therefore we next considered a more complex model that can learn non-linear relationships in the data. Neural networks were chosen for this purpose. The neural network experiments will try to solve the same task addressed by the linear regression model in Section 5.1.3: predicting the soil temperatures given the ambient temperature. This was discovered to be the most useful application for these datasets with practical application in mind.

Unless stated otherwise, the neural networks described in the following were coded in Python 2.7 using a library called Lasagne, that in turn builds upon another library named Theano.

Stellenbosch University https://scholar.sun.ac.za

Theano is a library that makes setting up and solving symbolic expressions much easier by allowing one to define variables, and then link these to a formula which only requires input values later when called. This makes it easy to set up all the dependent equations used by a neural network.

Lasagne allows the easy implementation of specific neural network structures by defining all the different layers of the network and then stacking them together. Once the initial network is set up, it is easy to revisit the different layers and fine-tune them without disrupting the rest of the network.

A small section dedicated to the basics of Theano and Lasagne is presented in Appendix A.

5.2.1 The first model

Since there are so many different ways to construct and fine-tune neural networks, it was decided to start with the simplest network structure and to use this to attempt to match the linear regression model. This was done by simply using one input and one output layer, and linear neurons. This structure is equivalent to the linear regression model and it was verified that this model achieved similar results when given the same inputs as in Section 5.1.3, when predicting the soil temperatures from the ambient temperature.

After this confirmation, the model complexity was increased slightly by adding two hidden layers and a dropout layer [34] for each of the hidden layers as a form of regularization. This meant that the model will be a standard feed-forward network trained with backpropagation and stochastic gradient descent (SGD).

Many configurations were tested experimentally. It was found that a network with two hidden layers delivered best results on our data. Fewer layers resulted in disappointing results, which might be due to an inability to learn non-linear patterns. More than two hidden layers also gave bad predictions, which is suspected to be the result of overfitting to the data.

Stellenbosch University https://scholar.sun.ac.za

The meta-parameters that were optimised in this way include:

- Number of hidden layers
- Number of neurons in each layer
- Number of dropout layers as well as their dropout-probability
- The method of weight initialization
- Number of training epochs
- The mini-batch size

Due to this large number of meta-parameters, it was a lengthy process to optimise the network structure setup even for our relatively small dataset. The network that led to the best validation set performance was the following:

- Two hidden layers, for a total of four (dense) layers including the input and output layers.
- 7 Neurons for the 7 input variables (same variables as for the linear regression model), 150 neurons in the first hidden layer, 100 in the second hidden layer, and 80 neurons in the output layers for the 80 different temperature predictions.
- Two dropout layers (one for each of the hidden layers) with a 50% dropout probability.
- The weights were initialised using Xavier initialization [11].
- The squared error function was used since this is a regression problem.
- The model will learn using momentum with *Nesterov's Accelerated Gradient* [36] using a learning rate (ϵ) of 0.01 and a momentum (μ) of 0.75.
- 500 training epochs.
- A batch size of 100 when training, and 210 (the size of the full validation and test set) for validation and testing.

Stellenbosch University https://scholar.sun.ac.za

The first results

Using the error score defined in Section 3.1.2, this model obtained an average error of **5.4%**, which is slightly better than the linear regression model. Over all of the 80 different measurement points (4 rows x 5 blocks x 4 depths) the best prediction achieved an average of 3% error and the worst an average error of 10% (averaged over the 210 input-samples). An example of the temperatures versus measurement number of the predicted values versus the target values for the best and worst measurement-points respectively is shown in the two graphs below, with the error shaded in red.

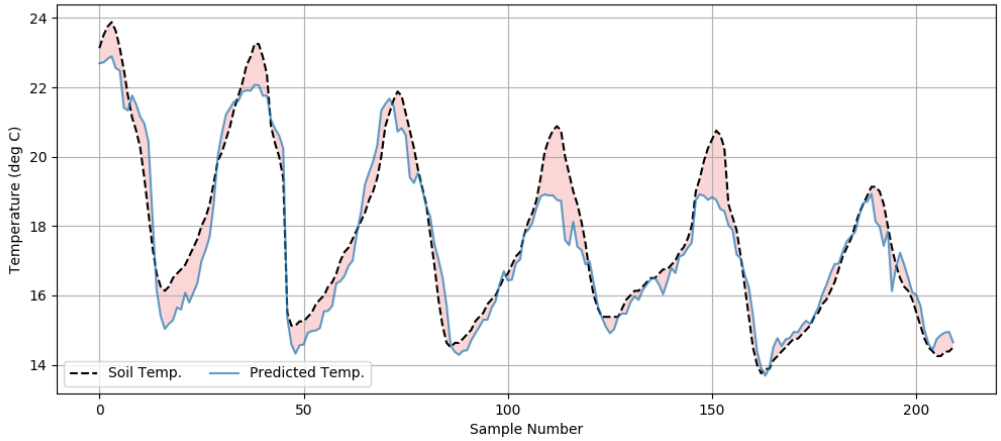


Figure 5.3: Best prediction vs. target value plot (temperature vs. sample number) for the first experiment. The error is indicated by shading.

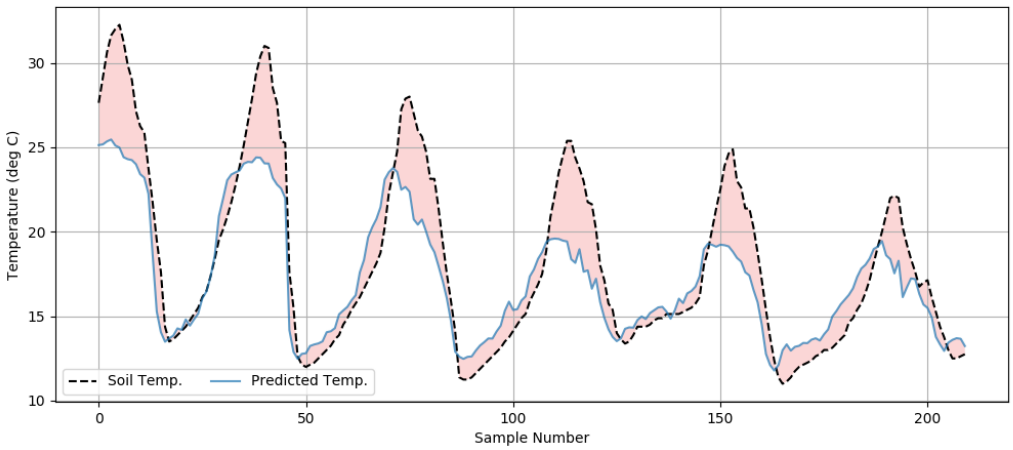


Figure 5.4: Worst prediction vs. target value plot (temperature vs. sample number) for the first experiment. The error is indicated by shading.

Stellenbosch University https://scholar.sun.ac.za

Inspection of the graphs indicates that most errors occur at the maximum temperatures of each day, where the prediction is too low. It must also be noted that no two iterations of network training will provide the exact same results since the weights are randomly initialised. However, performance was seen to remain highly consistent, and an average error of 6.5% (on a single training iteration) was the highest error seen so far.

5.2.2 Auto encoder pre-training

For the second neural network model, an auto encoder was used to pre-train the weights of the network. These pre-trained weights will then be used as the initial weight values. The final weights were obtained by training from this initialisation as described in Section 5.2.1, only with fewer training epochs.

An auto encoder is a neural network that is trained to re-create its input at its output. Auto encoders are often shallow networks consisting of only one hidden layer. Shallow auto encoders may however be stacked together to form deep auto encoders. The network can be seen as consisting of two parts: an encoder function $h = f(x)$ and a decoder that produces a reconstruction $r = g(h)$. This architecture is shown in Figure 5.5 below.

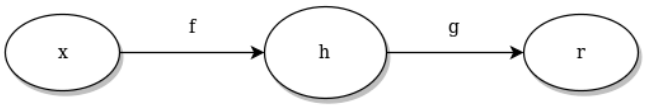


Figure 5.5: General structure of an auto encoder, mapping input x to an output (reconstruction) r through an internal representation h. The encoder part of the network is represented by f, while the decoder part is represented by g.

If the structure of the auto-encoder allows it to simply learn the mapping $g(f(x)) = x$, then it is not especially useful. Instead, auto-encoders are restricted in ways that do not allow them to simply copy the input to the output. This can be done by implementing a "bottleneck" layer in the middle of the network that has fewer neurons than the input layer. By implementing this restriction, the network is forced to pick the most important features from the input data to help

Stellenbosch University https://scholar.sun.ac.za

in the reconstruction process.

In our experiment, the auto-encoder has an input layer with seven neurons (for the seven input parameters), followed by a hidden layer with 150 neurons (which is the same size as the first hidden layer in the model we are trying to train), then a bottleneck layer with only 5 neurons. After the bottleneck layer, a further two layers of 150 and 7 neurons respectively complete the auto-encoder. What this network must try to do it to learn weights that will give an output to resemble the input as closely as possible.

When this network succeeds in reconstructing its inputs at the output, the weights running from the input layer to the first hidden layer can be used to initialise the weights of the final model (W1 in Figure 5.6). Then this input layer is used to propagate the inputs through the first layer and obtain activations for the 150 neurons in the first layer. Next, a second shallow network is constructed using these 150 activations as the input, a single hidden layer with 100 neurons (same as the final model), and then an output layer again with 150 neurons. The second shallow network is now trained to reconstruct its 150 inputs at its outputs. The weights of this second auto-encoder now form the second set of pre-trained weights for the final model (W2 in Figure 5.6). This process continues until pre-trained weights for all of the layers have been obtained. In our case, the process was repeated a total of 5 times to obtain the final set of pre-trained weights running from the second hidden layer to the output layer in the final model (W3 in Figure 5.6). Figure 5.6 shows the three auto-encoders that were used to obtain these pre-trained weights. The numbers at the bottom of the rectangles represent the number of neurons in each layer. The arrows labelled W1, W2, and W3 respectively indicate the sets of weights that were extracted and used to initialise the weights of the final model.

Stellenbosch University https://scholar.sun.ac.za

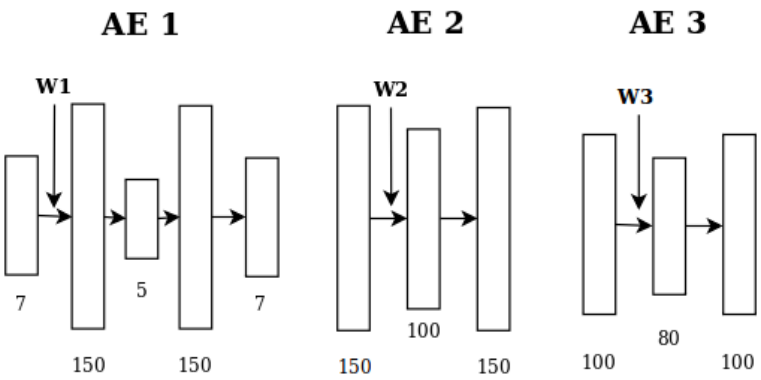


Figure 5.6: The three auto encoder networks, labelled AE1, AE2, and AE3 respectively, used to pre-train the weights for the final network. W1, W2, and W3 indicate the specific sets of weights used in the final model for the layers 1, 2, and 3 respectively. The rectangles represent a layer of the neural net with the numbers at the bottom indicating the number of neurons in that layer.

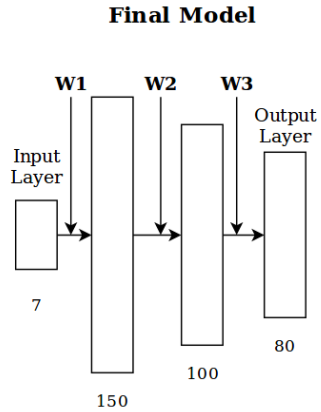


Figure 5.7: The final neural network using the weights W1, W2 and W3, pre-trained by the three auto-encoders. W1, W2 and W3 respectively indicate the weights that were obtained from the three auto-encoders as shown in Figure 5.6. The rectangles represent a layer of the neural net with the numbers at the bottom indicating the number of neurons in that layer.

Note that the architecture of the final model shown in Figure 5.7 is the same as the first model observed in Section 5.2.1. The only difference is that this model uses pre-trained weights learned from the auto-encoders as shown in Figure 5.6.

Using these pre-trained weights to initialise the weights of the network, we trained a new network in the same way described in Section 5.2.1. This final training must occur over fewer training epochs, otherwise the model may overfit once again.

Stellenbosch University https://scholar.sun.ac.za

Auto encoder results

Auto encoder pre-training unfortunately did not provide substantially better results. The network managed to achieve an average error rate of **5.37%** when using auto encoders pre training as described above.

The performance achieved by neural networks initialised using auto encoder weights was not substantially better than that achieved without this pre-training strategy. It therefore seems that the auto encoders are not learning patterns from the data that were not already picked up during conventional training.

5.3 Scikit-learn Neural Network

For the third model, a different neural network library, Scikit-learn, was chosen. This library is quick and easy to set up and to experiment with.

The "scikit-learn.neural_network" package can be used to configure either a classification or a regression type network. In this case we used the regression model. The network architecture was set to be the same as the Lasagne model, i.e. 7 neurons in, 150 neuron first hidden layer, 100 neuron second hidden layer, and 80 output neurons. This model also uses rectified linear neurons (or "ReLu") and learns using the "Adam" update rule [18] with a learning rate of 0.001. At the time of writing, the scikit-learn has no way of adding dropout layers to the model. This is different to the Lasagne models which can include dropout layers. The model was trained for 500 training epochs which is the same as the lasagne model.

After giving the scikit-learn model the input data, it achieved very similar results to the Lasagne model. Note that the parameters stated above were found to produce the best results after trying many different parameter combinations. The models trained with scikit-learn did exhibit more variance in terms of their performance, with errors ranging from as low as 5.2% to as high as 6%. The average overall performance was **5.4%**, however, which is the same as that achieved by the networks trained using Lasagne.

Stellenbosch University https://scholar.sun.ac.za

5.4 Pre-trained Neural Network

In this section we will use the weights learned using scikit-learn to initialise the weights of a Lasagne model. The Lasagne model is then trained on the input data for very few epochs to find the final adjustments to the weights.

At first, this did not really help, and in fact caused the error rate to increase a bit. After some investigation, however, some favourable results were obtained.

First it was decided to also use the Adam weight update method in Lasagne instead of Nesterov Momentum, to match the update method used in scikit-learn. Next, the dropout layers were removed from the Lasagne network to match the scikit-learn network architecture more closely. We could now expect the network to always learn the same weights after a certain number of training epochs since all random factors have been removed from the training procedure. Finally, the number of training epochs used in Lasagne was reduced to ten, thereby limiting this step to fine-tuning of the weights. So it would seem that the network only needed to make some small final adjustments to the weights.

By using this procedure, the model achieved a **4.84%** average prediction error on the data. This is much better than was achieved with any of the previous models. Since all randomness has been removed from the network development, the results are exactly reproducible.

5.5 Shuffling the data

For all of the previous tests, the data was divided into 80% training, 10% validation and 10% testing data. These sets represented blocks of data points that were consecutive in time. It was decided to see if the model would still be able to fit the data if the data sets are shuffled.

To implement this, the data points were shuffled before being divided into 80%-10%-10% training, validation, and test partitions as before. This actually seemed to improve the overall results of the model since it was now forced to learn different, better, patterns in the data.

Stellenbosch University https://scholar.sun.ac.za

When running the first model with shuffling, but all other parameters kept the same, except training for 1000 epochs instead of 500, it achieved an average error of **4.696%** over 100 runs. This is a reduction of 0.7% from the 5.4% achieved without shuffling. The figure below shows the results by plotting the targeted values together with the predictions. Note that the data appears messy because, due to the shuffling, consecutive points on the graph are no longer consecutive in time.

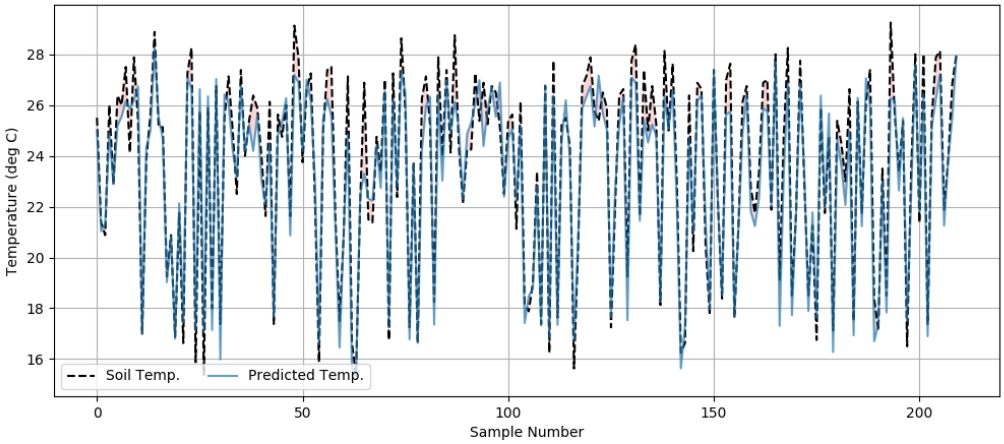


Figure 5.8: Predictions plotted with target values after shuffling the data (temperature vs. sample number).

When using the shuffled data in conjuction with the pre-trained weights obtained in Section 5.4 the model achieved an average error of **4.2%**, the best result so far.

5.6 Input Parameter Weights

To get a more quantitative indication of which inputs are most important for predicting the output, different combinations of the inputs were omitted and the effect on the predicted outputs evaluated. In each case, new neural networks were trained. The hyper parameters, such as the number of neurons and number of hidden layers were kept the same in order to allow direct comparison. Note that whenever the ambient temperature was given as input, the log of that temperature was also given. The table below shows the results the model achieved with different combinations of input parameters:

Stellenbosch University https://scholar.sun.ac.za

Test No.	Input parameters	No. of inputs	Average error
1	naïve guess	1	23.5%
2	Temperature	2	11.58%
3	Temperature + Humidity	3	10.96%
4	Temperature + Hour	4	9.58%
5	Temperature + Hour + DoY	6	4.9%
6	Temperature + DoY	4	5.27%
7	Hour	2	16.537%
8	Hour + DoY	4	5.56%
9	DoY	2	7.93%
10	All inputs	7	4.69%

Table 5.2: Prediction accuracy with different combinations of input parameters. DoY denotes the day of the year (1-365). Both DoY and the hour parameter are presented as sine and cosine components as described in Section 5.1.3. The naïve guess corresponds to predicting the soil temperature to be the same as the ambient temperature.

Looking at these results, some interesting conclusions can be drawn. It can be seen once again that the humidity input has only a very small effect on the prediction results. One very interesting result is that the model values the DoY parameter much more than the hour of measurement. When given in combination with the temperature, Test 5 (with DoY) makes an average error of only 5.27% compared to Test 3 (with hour) that makes an average error of 9.58%. Even more surprising is that when the model is given only the DoY, it is still able to make rather good predictions, achieving an error of 7.93%. This is the lowest error that the model achieves for a single input parameter (see Test 2 and Test 7 for a comparison). This means that the Day-of-the-Year parameter is definitely the input parameter that carries the most weight in this prediction by quite some margin.

5.7 Including dynamic information in the training data

Up to now, we have tried to estimate the output (soil temperature) given inputs measured at the same time. In the following, we will extend the inputs of the network to include also past measurements. This was done in two ways.

Stellenbosch University https://scholar.sun.ac.za

5.7.1 Incorporating previous measurements

For the first test, five extra features were added. These five features correspond to five previous ambient temperature measurements, made 30, 60, 90, 120 and 150 minutes earlier. Over 100 runs, this reduced the average error to **4.436%** from the previous best of 4.696%. This corresponds to an improvement of 0.25%.

When given only the previous two measurements, an average error of **4.483%** was achieved.

5.7.2 Incorporating previous minimum/maximum temperatures

In this experiment, two features were added to the 7 used in previous experiments, making up a total of 9 features for this training set (and 9 input neurons). These two features were the minimum and the maximum temperature measured over the previous 24 hours. This means that these features have a lot of repetition. For instance, every measurement made on the 13th of April would contain the same two values for the minimum and maximum temperature measured from April 12th.

Once again, this improved the results of the model, and helped to bring down the average error to **4.02%**. This is the best results achieved so far, even outperforming the model with the pre-trained weights. The table below shows a summary of the results:

Test Number	Model	No. of inputs	Average error
1	No dynamic information	7	4.70%
2	+ Previous 2 measurements	9	4.48%
3	+ Previous 5 measurements	12	4.44%
4	+ Min/Max temp. of previous day	9	4.02%

Table 5.3: Comparison of regression performance achieved when incorporating dynamic information in various ways.

Stellenbosch University https://scholar.sun.ac.za

5.8 Summary and conclusion

This section serves as a short summary of the results obtained with the different neural network models for the Stellenbosch dataset. All networks predict the soil temperature at different places and depths while learning from the date/time of measurement as well as the ambient temperature and humidity (unless stated otherwise).

Test No.	Model	Section	Avg. error
1	Naïve guess	5.2.1	23.5 %
2	Linear Regression	5.1.3	5.56%
3	First NN model (chronological data)	5.2.1	5.40%
4	Pre-trained /w Auto-encoders	5.2.2	6.5%
5	Scikit-learn Neural Net	5.3	5.40%
6	Pre-trained with Scikit-learn model	5.4	4.84%
7	Shuffled data (standard model)	5.5	4.70%
8	Shuffled /w pre-trained weights from test 4	5.5	4.2%
9	Given ambient temperature only	5.6	11.58%
10	Given Hour of measurement only	5.6	16.54%
11	Given Day of year only	5.6	7.93%
12	With previous 2 measurements	5.7	4.48%
13	With previous 5 measurements	5.7	4.44%
14	With previous day's min/max	5.7	4.02%

Table 5.4: Comparison of the results of all the different neural network models considered for the Stellenbosch dataset.

In conclusion, all regressors considered are able to achieve a much better estimate of the soil temperature than the naïve guess, which assumes the soil temperature to correspond to the co-located ambient temperature. The most important information required by the regressor, besides the ambient temperature, are the day of the year and the time of day, both represented as sine and cosine components. Finally, including the maximum and minimum ambient temperatures over the last 24 hours is an effective way of including dynamic information, and substantially improves performance.

Stellenbosch University https://scholar.sun.ac.za

Chapter 6

Experiments with Ssw. Data

After performing the experiments, described in the previous chapter, a second dataset became available. This data originates from a farm in Somerset West and has been described in Chapter 4. This dataset contains many more entries and spans more seasons than the Stb. data. This chapter describes the results of experiments performed with this dataset.

6.1 Predicting soil temperatures using microclimate logger data

This first experiment using the Ssw. dataset was the same experiment carried out with the Stb. data in Section 5.2.1. In this case, however, more data is available, stretching over a full year instead of just a few months.

Because of the data now extending over all four seasons, we were initially not sure if the model would in fact be able to make better predictions, since the longer timespan may also increase the difficulty of the prediction task. This was one of the objectives of this experiment - to see if a neural network model could still make reasonable predictions now that the dataset covers all four seasons.

A second objective was to see how well a neural network model could predict the soil temperatures at a certain location after being trained on data obtained at a different location. This will indicate how well the model is able to generalise to unseen locations from a specific training dataset.

Stellenbosch University https://scholar.sun.ac.za

Again, the neural network model was implemented with Lasagne. It uses three fully connected hidden layers with 150, 150 and 80 neurons (150-150-80), along with two dropout layers for the last two hidden layers. After many iterations of experimentation, the model consistently achieved lower error rates on the validation set when dropout was applied in this way. The output layer consists of a single neuron, indicating the soil temperature at a certain time, date, and depth. The input layer has eight units, corresponding to the following inputs (to be explained shortly):

1. Day of the year - cos
2. Day of the year - sin
3. Hour of measurement - cos
4. Hour of measurement - sin
5. Ambient Temperature (Celsius)
6. Relative Humidity (%)
7. Depth in soil (12.5 cm, 25 cm or 37.5 cm)
8. Vigour level (0,1 or 2)

The day of the year and hour of measurement inputs are split up into sine and cosine components, as explained in Section 5.1.3. This helps the model make more sense of the circular nature of these variables. The ambient temperature is from the Ssw. mesoclimate dataset and contains very local temperature readings. The humidity is also from the mesoclimate dataset, except when the humidity reading was identified as corrupt, in which case the local weather station measurement for that time was used instead (as explained in Section 4.2.2). The depth can be either 12.5, 25 or 37,5 cm. These are not exact depths, but fall in the range of depths (0-15 cm, 15-30 cm, 30-60 cm). The vigour levels can be either low, medium or high. For this model we coded these as 0, 1, and 2 respectively.

Since adding dynamic information, in the form of minimum and maximum temperatures of the previous day, worked so well on the Stb. dataset models, the effect of adding this information for the Ssw. dataset was also considered.

Stellenbosch University https://scholar.sun.ac.za

The dataset of 40000 entries was split into subsets using the ratio 80-10-10, with 80 % as the training set, 10 % as the validation set, and the final 10 % as the test set. All values in the training set were normalised to lie in the range between -1 and 1. The training data was shuffled during training.

Mini-batch gradient descent with Nesterov momentum, a learning rate of 0,001 and a momentum of 0,9 were used throughout. The mini-batch size was 256 (32000 / 125) and training continued for 500 epochs.

To calculate the prediction error rate made by the model, the following formula was used:

$$Error = \frac{|target - output|}{target} * 100\% \tag{6.1}$$

For example if the model gave an output of 15 °C, but the correct value was 10 °C, the error would be 50 %. The average error over all test samples was used to assess model performance.

Results

Several (50) models were trained, and the average error over all these models were taken. Error rates were compared to a naïve guess, which simply assumes that the soil temperature is the same as the ambient temperature, to see if the model is learning from the data. This naïve guess model achieved an average error of 21,71 %.

The average error over the 50 different models was **4,94 %** with a standard deviation of 0.18 %. This translates to a 1,02 °C error on average. When the minimum and maximum temperatures from the previous day were added as additional inputs, the models achieved an average error of **4.453 %** with a standard deviation of 0.27 %. This translates to an average error of 1.00 °C.

This shows that a neural network can learn valuable patterns in the temperature data and make reasonably accurate predictions. However, when the same model was tested on the Stb. soil temperature dataset, as is described in Section 4.1.1, it achieved an average error of **18,56 %**. This shows that the model unfortunately

Stellenbosch University https://scholar.sun.ac.za

cannot predict the soil temperatures well if it is trained on data from a different location. It might be possible to train a model that can generalise better, but it will need different input variables so that the model can understand the difference in the location, for example, soil composition and distance to the ocean might be useful variables.

6.2 Predicting soil temperature using freely available data

As a next experiment, we considered whether it is possible to predict soil temperature at various depths using only freely available data. As freely-available sources of data we considered local weather stations and satellite surface temperature data.

First, we used the same model architecture as in Section 6.1: a neural network with three fully connected hidden layers containing 150-150-80 neurons respectively, and with a dropout layer added after each of the last two hidden layers. The dropout layers have a dropout probability of 50 %. The output layer still consists of a single unit for the predicted soil temperature. There are seven inputs, namely:

1. Day of the year - cos
2. Day of the year - sin
3. Hour of measurement - cos
4. Hour of measurement - sin
5. Depth in soil (12.5 cm, 25 cm or 37.5 cm)
6. Satellite daily temperature (Measured by day, °C)
7. Satellite daily temperature (Measured by night, °C)

As a second experiment, we additionally included rainfall data measured by the local weather station (see Section 4.3) in the form of an exponential moving average, weighted over the last 100 samples. All other hyperparameters remain the same.

Stellenbosch University https://scholar.sun.ac.za

Results

The results were calculated in the exact same manner as the earlier experiment of Section 5.2.1. Recall that performance should be compared to the naïve guess which achieves an error of 21,71 %.

After training 50 different models and taking the average of their performance, the average error was **5,57 %** with a standard deviation of 0.76 %. This is only slightly worse than the results of Section 6.1, which used locally-measured ambient temperatures as well as a knowledge of the vigour level. This error rate corresponds to an average error of 1.13 °C.

Experiment 2, with the added rainfall measurement as input, yielded an average error of **5.33 %** with a standard deviation of 0.77 %. This shows that adding rainfall information does in fact help the model, but only slightly.

The vigour level was also added as input while experimenting with input features, however, this barely resulted in any change to the performance of the models. Vigour level is also not always freely available in all cases, which is the focus point of this experiment.

6.3 Predicting soil temperatures using a mixture of available data

In this section, we consider how well the model performs when provided with all the data we have. This means that it was trained using all the training data used in Section 6.1, as well as the day and night satellite temperatures, as well as the rainfall data introduced in Section 6.2.

The neural network architecture remained unchanged except that in this case there were 11 inputs. These inputs are:

1. Day of the year - cos
2. Day of the year - sin
3. Hour of measurement - cos

Stellenbosch University https://scholar.sun.ac.za

4. Hour of measurement - sin
5. Ambient temperature (°C)
6. Relative humidity (%)
7. Depth in soil (12.5 cm, 25 cm, or 37.5 cm)
8. Vigour level (0,1 or 2)
9. Satellite daily temperature (measured by day, °C)
10. Satellite daily temperature (measured by night, °C)
11. Hourly rainfall as a moving average (mm)

All remaining hyper parameters were unchanged.

Results

The results were calculated in the same manner as the previous experiments, and again the results should be compared with a naïve guess which achieves an error of 21,71 %.

50 neural networks were trained using these inputs, in each case initialising from a different set of random weights. On average, the networks achieved an error of **4,34 %** with a standard deviation of 0.36 %. This translates to an average error of 0.93 °C.

This is an improvement over the previous experiments and indicates that the local measurements and the wide-scale satellite and rainfall measurements are to some extent complimentary. However, the improvement might not be considered large enough to justify the costs and effort of obtaining all this additional data.

Figure 6.1 shows a comparison of the temperatures predicted by the neural network and the actual measured soil temperature. One can see that the model follows the trend very well, but underestimates the maxima en overestimates the minima, keeping the error average low by making conservative predictions.

Stellenbosch University https://scholar.sun.ac.za

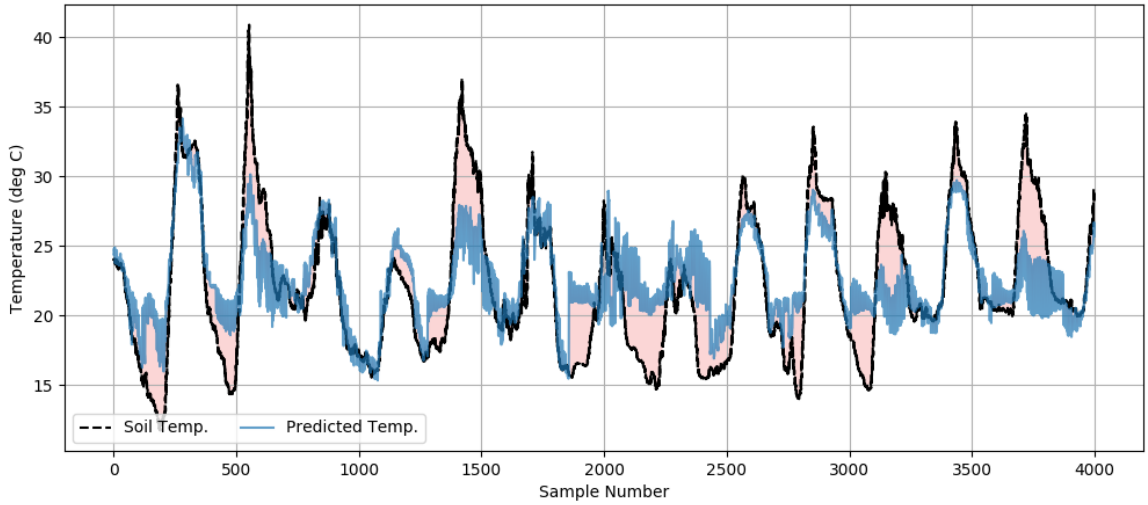


Figure 6.1: A graphical comparison of the temperatures predicted by the neural network (blue) and the measured soil temperature (dotted black).

6.4 Predicting soil moisture levels using freely available data

In this section we will attempt to predict something other than soil temperatures. We will predict the moisture content in the soil. If this can be predicted accurately enough, the moisture sensors would not be required, thereby greatly simplifying vineyard monitoring.

The moisture content in the soil is measured by water-counts as explained in Section 4.5. These are integer values ranging (in our dataset) from 4587 to 9662 with an average of 6165. The higher the count, the more water content in the soil.

The architecture of the neural network is similar to that used before. Three fully connected hidden layers were used again, this time containing 80, 50, 50 neurons respectively and only one dropout layer after the input layer. After multiple iterations of evaluation using different architectures, it consistently led to better results on the validation set when using dropout in this manner. The *Adam* update method [18] was used, and training continued for 7000 epochs. This architecture, as well as the hyperparameters, were selected after evaluating the model on the validation set. The input layer consists of 17 inputs as follows:

Stellenbosch University https://scholar.sun.ac.za

1. Day of the year - cos
2. Day of the year - sin
3. Vigour level (0,1,2)
4. Depth in soil (12.5 cm, 25 cm, 37.5 cm)
5. Satellite land-surface temperature (LST) (current day, °C)
6. Satellite land-surface temperature (previous night, °C)

7 - 12. Average weekly day-time and night-time LST, for each of the last 3 weeks

13. Humidity (current day, %)

14 - 16. Average weekly humidity (for each of the last 3 weeks)

17. Total rainfall within the last 30 days

Results

The error rate was calculated in the same manner as it was for the temperature predictions in Sections 6.1 - 6.3. Model performance should also again be compared with a naïve guess. In this case the naïve guess was simply the average moisture level over the entire training dataset (6165). This naïve guess achieves an error of 11,6 %.

After training 50 models from random initialisations, an average error of **8.01%** was achieved on the test set with a standard deviation of 1.75 %. The model made errors between 380 and 800 with an average of 510 counts (from the test set score).

This result demonstrates that the model is learning something from the data, and scores better than a naïve guess. However, the model prediction is not much better than a naïve guess.

It must be kept in mind that the model was trained with very little data. It is possible that better performance can be achieved for a larger training set.

Stellenbosch University https://scholar.sun.ac.za

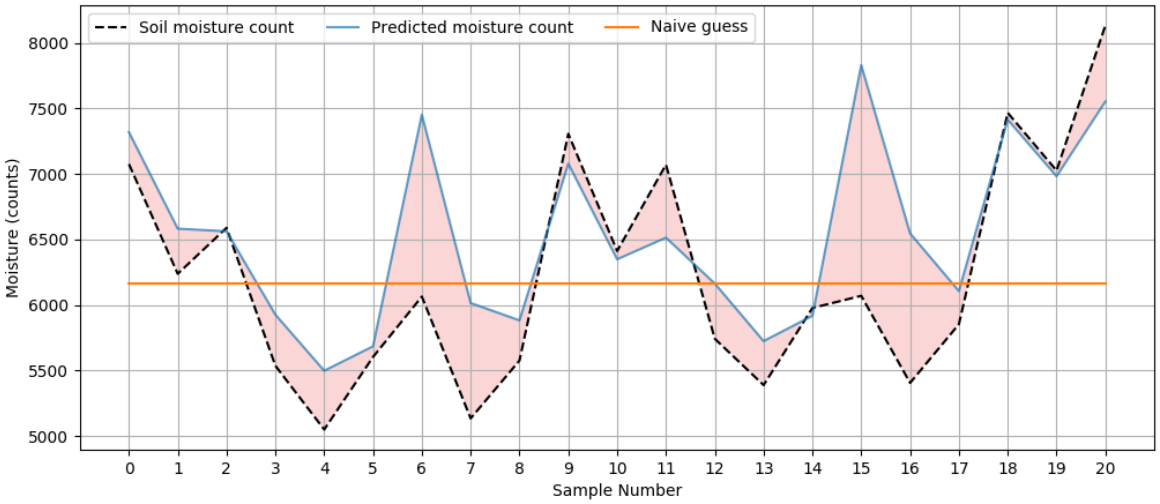


Figure 6.2: A graphical comparison of the soil moisture content predicted by the neural network (blue) and the measured moisture counts (dotted black).

6.5 Predicting bud-burst dates

As mentioned in Section 4.9, the bud-burst date is an important date in the growth stage of the grape vine. According to the data available to us, bud-burst seems to happen in the same month every year (September). Knowing if it is more likely to happen at the beginning of the month or the end could help the viticulturists prepare for an earlier-than-expected harvest date.

Since the bud-burst only occurs once per year, we had very little data to work with. In total, we had eight bud-burst dates over the span of four years, one date every year for the medium vigour vine-block and one every year for the low-vigour block.

Due to the small amount of data available, leaving-one-out cross-validation was employed. The models were trained on seven of the eight data points and then used to predict the eighth. This was repeated eight times, each time leaving out a different year and block. The final result was the average of these eight experiments.

The models were trained with 17 inputs, which included mostly data from the three months preceding the bud-burst month of September (June, July, and

Stellenbosch University https://scholar.sun.ac.za

August). This was done because, after many test runs, it was found that the model performed best when only considering the previous 3 months' data as input. The input parameters are the following:

1 - 3. Average monthly humidity (June, July and Aug.)

4 - 6. Average monthly air temperature (June, July and Aug.)

7 - 9. Average monthly rainfall (June, July and Aug.)

10. Average air temperature for the whole year up to September

11. Accumulated rainfall for the whole year

12. Moving average: air temperature (50 days)

13. Moving average: humidity (50 days)

14. Moving average: rainfall (50 days)

15. Moving average: soil temperature (50 days)

16 - 17. Binary columns to indicate low-vigour or medium-vigour

The models were set up to predict the day in September on which bud-burst will happen. In other words, if the models predicts "5", it is predicting that bud-burst will happen on the 5th of September.

Three different models were tested, namely linear regression, second order polynomial regression, and a neural network. 100 neural network models were trained, each from randomly initialised weights, and applied for each of the 8 experiments. The overall result was taken as the average of those 100 repetitions.

The models were all built and trained using the Scikit-learn library. All the input parameters were normalised before training. The neural network was built with four hidden layers of sizes 120-100-100-80, and uses the Adam training method.

Results

The models were trained on seven data-points and used to predict the eighth point. This was repeated for all eight available bud-burst dates. The score was then average over all eight runs.

Stellenbosch University https://scholar.sun.ac.za

As before, the model was compared with a naïve guess. This time, the naïve guess was the bud-burst date of the previous year. When using this naïve guess, an average error of **8.75 days** is made, with a standard deviation of 3.38 days. The best naïve guess was only off by 1 day, but worst by 16 days.

The linear regression model improved on the naïve guess, making an average error of **6.21 days** with a standard deviation of 5.02 days. The model's best guess was off by 1.13 days, and the worst by 15.46 days.

The second-order polynomial regression model performed best of all, making an average error of **4.15 days** with a standard deviation of 4.29 days. The polynomial model's best guess was off by 0.9 days, and its worst by 10 days.

The neural network also performed well, falling just short of the polynomial regression model, with an average error of **4.79 days** and a standard deviation of 4.02 days. In this case the best guess was off by 0.09 days, and the worst by 9.78 days.

Model	**Avg error**	**Max error**	**Min error**	σ
naïve guess	8.75 days	16 days	1.00 day	5.38 days
Linear regression	6.21 days	15.46 days	1.13 days	5.02
Polynomial regression	**4.15 days**	10.00 days	0.9 days	4.29 days
Neural network	4.95 days	**9.78 days**	**0.09 days**	**4.02 days**

Table 6.1: Comparison of the results of the bud-burst date prediction with different models.

The neural network and the polynomial regression model did quite well, but with the polynomial regression model making less than half the average error than a naïve guess, as well as having a much shorter training time, would be the model of choice. Even though the model does quite well on average, it is not perfect, and with a maximum error of 10 days might not be seen as very trustworthy yet. Out of the eight runs, the linear regression model was closer to the correct bud-burst day than the naïve guess 6/8 times. The polynomial regression model was closer 5/8 times and the NN model was closer 4/8 times. This makes a small case in the preference of the linear regression model, since it was closer to the true

Stellenbosch University https://scholar.sun.ac.za

bud-burst day more than the other models even though the average error is higher.

Further improvements will be needed to make this application practical in a real world scenario. However, it should be borne in mind that in this case our training set was particularly small. More training data should bring down the error rates.

6.6 Recurrent Neural Networks with Long Short-Term Memory

Since the introduction of long short-term memory (LSTM) recurrent neural networks, some outstanding results have been achieved on problems that require learning from sequential data, especially on data with long term dependencies. Since our data is mostly sequential, it might be expected that LSTMs should perform well.

LSTMs were trained to perform the same predictive tasks as the feedforward neural networks presented in Chapters 5.2.1 and 6.1: predicting soil temperature from ambient temperature. These experiments were split into two sub-experiments: predicting the soil temperature given a series of previous soil temperatures, and predicting soil temperatures given a series of previous ambient temperatures.

6.6.1 LSTM Experiments

Instead of building a LSTM network from scratch, we used an existing, working, LSTM and modified it to suit our data. Our experiments are therefore based on a Keras-based LSTM application designed to predict Google stock prices, and available on Github [28].

The unmodified model takes the closing stock price over 60 sequential days as input and attempts to predict the closing price on the 61st day. Training is performed by moving sequentially through the training set, each time considering the current price as target and the most recent 60 as input.

The model was built using four LSTM layers of 50 neurons each. Each of

Stellenbosch University https://scholar.sun.ac.za

these four layers was followed by a dropout layer with a 20% dropout probability to reduce overfitting. The four LSTM layers were followed by a 50 neuron dense layer, and then a single neuron as the output layer. Both the dense layer and the output neuron used rectified linear units (ReLU) activations.

After confirming that the model did indeed perform well on the Google stock price data, it was tested on synthetic data that was designed to have similar characteristics to the soil temperature measurements. This dataset was generated by summing a sine and a cosine component with random patterns of noise. An extract of actual soil temperatures from our data is shown in Figure 6.3. This extract illustrates the observed characteristics of the soil temperature data that inspired the function we used to generate synthetic data.

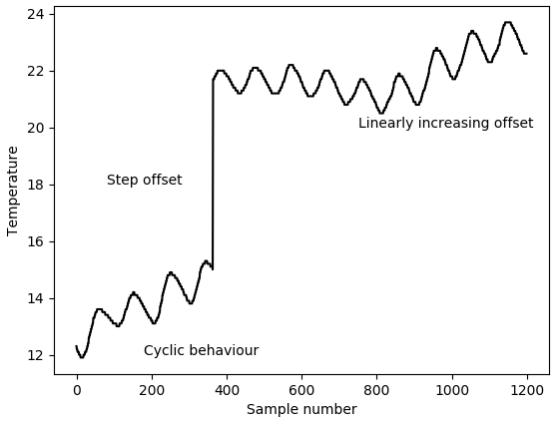


Figure 6.3: An extract of the soil temperature dataset that served as inspiration for Equation 6.2.

Figure 6.3 shows the cyclic nature of the daily temperatures, but also various signal components present in the data. One such component is the linearly increasing trend that the cyclic temperatures follow. Another component is the large discontinuities that are a result of missing or faulty measurements in the data. We also assume that there is a small element of Gaussian noise present in each measurement. Due to missing measurements, not every day contains the same number of measurements. This causes the period of the of oscillations to vary slightly. All of these attributes together form the basis for the design of Equation 6.2.

$$y = 2cos\left(\frac{2\pi f_1 t}{r(t)}\right) + 2sin\left(2\pi f_2 t + k\right)) + K_0(t) + K_1(t) + w(t) \qquad (6.2)$$

Stellenbosch University https://scholar.sun.ac.za

The function, $r(t)$ linearly changes the period of the cosine component over a random, but limited, number of samples. For example, it might scale the period by a factor of 2 over the space of 250 samples and then scale it by a factor of 0.7 over the next 400 samples. After that it might not scale it at all for the next 300 samples and so on. This mimics the varying number of measurements available for each day.

The components, $K_0(t)$ and $K_1(t)$ represent the step offset and the linearly increasing trend respectively, as illustrated in Figure 6.3. Finally, $w(t)$ represents additive Guassian noise.

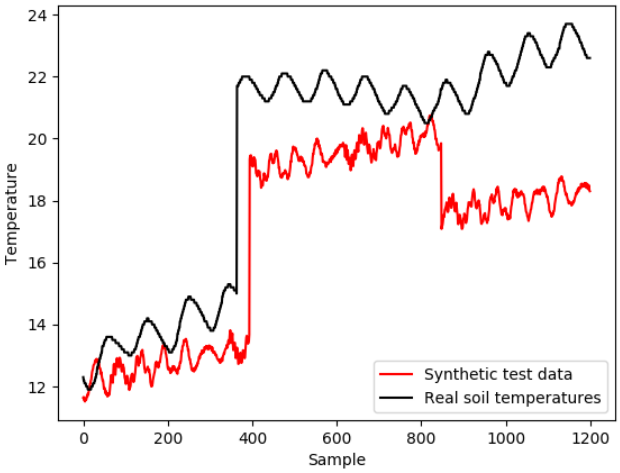


Figure 6.4: A randomly generated wave in red, and an extract from the real soil temperature data in black.

Figure 6.4 shows a small portion of the soil temperature dataset, and a portion of the synthetic data. This figure serves as an illustration to motivate the design of Equation 6.2. The cosine component models the daily temperature fluctuations, while the two piecewise-offset functions model the discontinuous jumps as well as the slowly increasing average observed in the soil temperature. The discontinuous jumps observed is a result of missing data. Hence, Equation 6.2 allows us to generate synthetic data with characteristics similar to those of the measured soil temperatures.

The LSTM model was trained to do one-step-ahead prediction on the synthetic data, using the Nesterov momentum optimiser. It was trained for 10 epochs at a time and consistently managed to fit the data well as illustrated in Figures 6.5 to 6.7 where the model is able to follow the trend of the held-out test data closely.

Stellenbosch University https://scholar.sun.ac.za

Figure 6.5 shows the LSTM predictions on synthetic data that strongly resembles the real soil temperature measurements, while the illustrations in Figure 6.7 show the LSTM predictions on various other randomised waveforms.

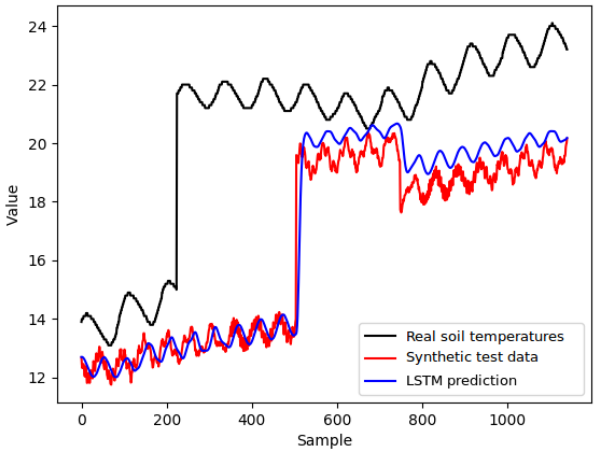
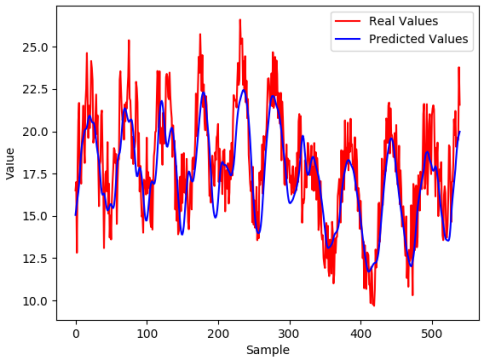
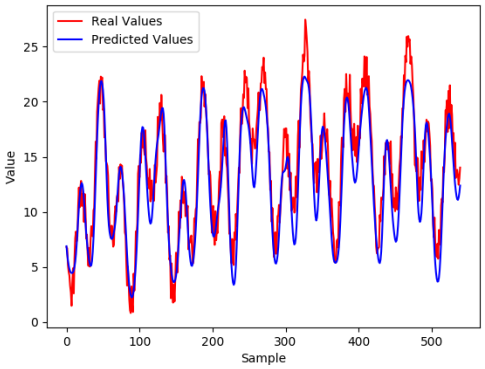


Figure 6.5: A randomly generated wave in red, with the LSTM predictions in blue and an extract from the real soil temperature data in black.





Stellenbosch University https://scholar.sun.ac.za

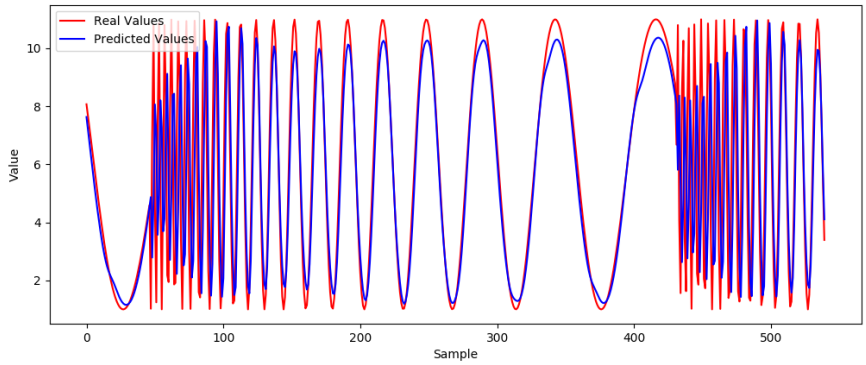


Figure 6.7: Portions of three randomly generated sine waves using Equation 6.2, shown in red, as well as the LSTM prediction in blue. This shows that the LSTM follows the randomly generated wave well.

6.6.2 One-step-ahead prediction of temperatures

The tests with the synthetic data in the previous section established that the LSTM is able to model data that is, at least to the eye, similar to the measured soil temperatures. Next, experiments were performed on the real soil temperature data. Again, the model was configured to do one-step-ahead prediction receiving the most recent 60 soil temperatures as input and predicting the next soil temperature. In each of the following experiments, a total of 100 models was trained, each model was initialised with random weights, and the average error across all 100 models reported.

When trained on the soil temperatures, the LSTM was not able to make good one-step-ahead predictions. Investigation into the behaviour of the networks indicated that the discontinuities in the data (as seen in Figure 6.4) were contributing to this poor performance. The data was therefore subsequently filtered to exclude these discontinuities so that every training example, consisting of 60 consecutive temperatures, was continuous. Thus, the discontinuities were removed from the data to which the LSTM was applied.

Stellenbosch University https://scholar.sun.ac.za

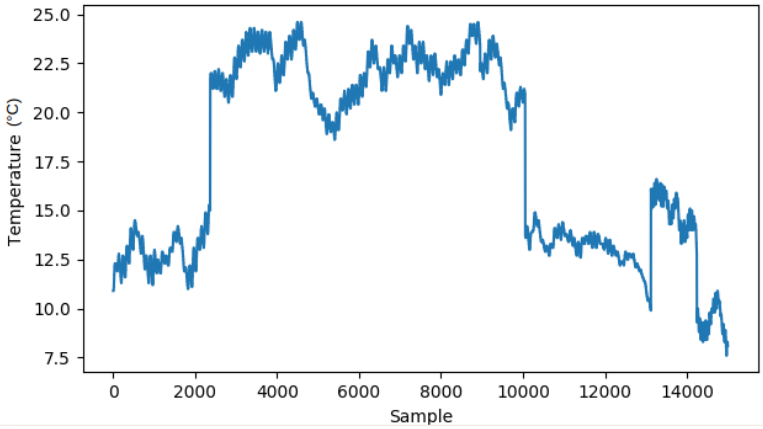


Figure 6.8: Full dataset of soil temperatures as used in the first LSTM experiment.

When considering the full set of soil temperatures, as shown in Figure 6.8, one can see big temperature jumps at several points. These discontinuities were removed from the data by considering only samples 2400 - 8600 for training and samples 8600 - 9600 for testing, as illustrated in Figure 6.9.

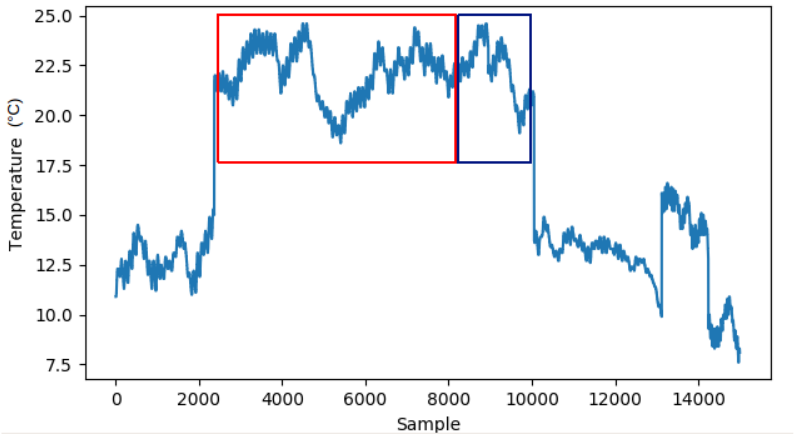


Figure 6.9: Soil temperature dataset as used for the first LSTM experiment, with the discontinuity-free training set indicated by the red block and the discontinuity-free test set by the blue block.

10% of the training data was kept aside as a validation set. The model was then trained using early stopping (training stopped after 5 epochs had passed without an increase in the validation score). After training and testing the LSTM on this smaller set of data, a much better result of **3.18%** over 100 runs was achieved. This demonstrates that the LSTM can predict continuous temperature data quite well. Figure 6.10 illustrates the test-set temperatures together with the predicted

Stellenbosch University https://scholar.sun.ac.za

values.

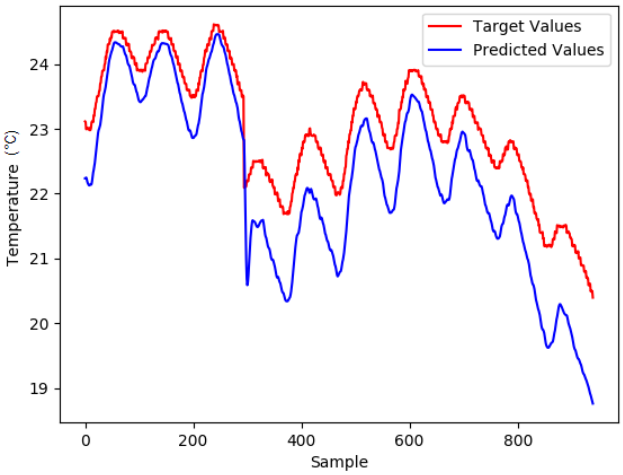


Figure 6.10: The test-set soil temperatures plotted in red and the one-step-ahead LSTM temperature predictions plotted in blue once discontinuities have been avoided by using the training and testing subset shown in Figure 6.9.

6.6.3 Prediction soil temperatures from ambient temperatures

Finally, we wanted to determine how well the LSTM would perform if given ambient temperatures as input, while still required to predict the soil temperatures. This experiment is directly comparable to that which was performed with the FFNNs in Sections 5.2.1 and 6.1 with the difference that the smaller discontinuity-free sub-sets shown in Figure 6.9 were used.

An average error of **4.6%** over 100 models with a standard deviation of 0.16% was achieved. The resulting soil temperature predictions for this experiment are shown in Figure 6.11 .

Stellenbosch University https://scholar.sun.ac.za

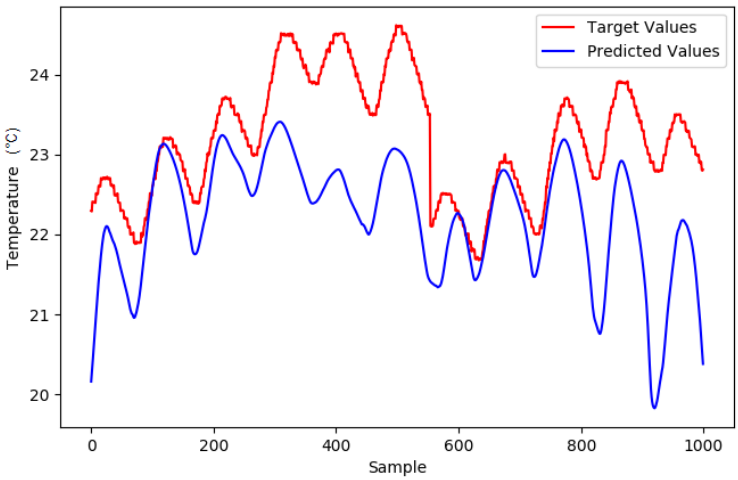


Figure 6.11: The test-set soil temperatures plotted in red and the one-step-ahead LSTM temperature predictions in blue when using ambient temperatures as input and using the training and testing subsets shown in Figure 6.9.

When relative humidity measurements were included as additional inputs, an average error of **4.35 %** with a standard deviation of 0.18% over 100 models was achieved. This result is comparable to the FFNN experiment described in Section 6.1, where the model achieved an average error of 4.94 %. However, when the FFNN model described in Section 6.1 is trained and tested on the same subset of data used by the LSTM, it achieved an average error of only 6.32 % with a standard deviation of 0.28 %, and an average error of 6.03 % with a standard deviation of 0.22 % when the previous day's minimum and maximum temperatures were added as additional inputs. hence, when tested under matching conditions, the LSTM is able to outperform the FFNN.

6.7 Summary of Ssw. dataset results

Table 6.2 summarises the results of the experiments performed with the Somerset West dataset, as described in this chapter.

In this table, the ambient temperature and humidity measurements are referred to as "Ambient", and it is specifically indicated when only one of the two is used as input. The satellite land-surface temperatures, as described in Section 4.6, are referred to as "Sat. LSTs". Also recall that the experiments in Sections 6.6.2 and 6.6.3 (test numbers 8 - 12) only use a subset of the full soil temperature dataset, as explained in Section 6.6.3.

Stellenbosch University https://scholar.sun.ac.za

We can conclude that it is possible to predict soil temperatures, soil moisture and bud-burst dates with accuracies better than a naïve guess. For the soil temperatures, best predictions were provided by the recurrent neural networks (LSTMs). However, these networks were also shown to be particularly sensitive to errors in the data caused by faulty sensors. The feedforward networks provided performance that was almost as good, while not exhibiting the same sensitivity.

The accuracies achieved for soil moisture content and bud-burst ddates were not as good as those achieved for the soil temperatures. However, in both cases (and especially for the bud-burst dates) the datasets were also much smaller. It is possible that substantially better performance might be achieved if more data were available.

Stellenbosch University https://scholar.sun.ac.za

Test no.	Model	Section	Targets	Inputs	Avg. error
1	naïve Guess	6.1	Soil Temp.	Ambient	21.71 %
2	FFNN	6.1	Soil Temp.	Ambient	4.94 %
3	FFNN	6.1	Soil Temp.	Ambient + Min/Max[1]	**4.45 %**
4	FFNN	6.1	Stb. dataset soil temp.	Ambient	**18.56 %**
5	FFNN	6.2	Soil Temp.	Sat. LSTs	5.67 %
6	FFNN	6.2	Soil Temp.	Sat. LSTs + Rainfall	**5.52 %**
7	FFNN	6.3	Soil Temp.	All data from Ssw.	5.4 %
8	LSTM	6.6.2	Soil Temp.	Soil Temp.	3.18 %
9	LSTM	6.6.3	Soil Temp.	Ambient Temp. only	4.6 %
10	LSTM	6.6.3	Soil Temp.	Ambient	**4.35 %**
11	FFNN	6.6.3	LSTM Soil Temps.	LSTM Ambient	6.32 %
12	FFNN	6.6.3	LSTM Soil Temps.	LSTM Ambient + Min/Max	6.03 %
13	naïve Guess	6.4	Moisture Content	Avg. of training data	9.42 %
14	FFNN	6.4	Moisture Content	Sat. LSTs + Rainfall	**8.28 %**
15	naïve Guess	6.5	Bud-burst dates	Prev. Date	8.75 days
16	Linear Regression	6.5	Bud-burst dates	Ambient + Rainfall + Soil Temps	6.21 days
17	Polynomial Regression	6.5	Bud-burst dates	Ambient + Rainfall + Soil Temps	**4.15 days**
18	FFNN	6.5	Bud-burst dates	Ambient + Rainfall + Soil Temps	4.95 days

Table 6.2: Summary of the results of the experiments performed using the Somerset West dataset.

[1]Minimum and Maximum temperature over the last 24 hours.

Stellenbosch University https://scholar.sun.ac.za

Chapter 7

Summary, conclusions and future work

In this chapter, we will summarise the most important results presented in this thesis, and draw conclusions based on these results. We will also identify avenues along which the work could be extended in the future.

The experiments presented in this thesis can largely be grouped as follows:

1. Predicting soil temperatures.
2. Predicting moisture content.
3. Predicting the bud burst date.

Most effort was spent on predicting soil temperatures, since that is where the most data was available.

7.1 Soil temperature prediction

A main focus of this thesis was the prediction of soil temperatures at various depths in a vineyard. Various configurations of inputs and model structures were considered in an attempt to gain some insight into which inputs are important and which models work best. The experiments were split into two groups, one performed on the smaller Stellenbosch dataset, and one performed on the larger Somerset West dataset.

Stellenbosch University https://scholar.sun.ac.za

7.1.1 Stellenbosch dataset

The Stellenbosch dataset was the first to become available during the course of this project, and therefore the subject of the first experiments. The aim was to predict the temperatures measured at different depths in the soil from the other information available in the dataset, such as local ambient temperature, local humidity, as well as the depth of measurement.

When using linear regression, soil temperatures could be predicted with an average error of **5.56%**. This was a surprisingly good result for such a simple model. A feedforward neural network, with the same inputs, was able to achieve the only slightly better average error of **5.4%**.

Next, pre-training strategies were considered. Auto-encoder pre-training afforded only marginal gains in performance, achieving an average error of **5.37%**. However, these models are much more complicated to set up.

A different style of pre-training was also tested. This was achieved by first training a neural network using Scikit-learn to predict the soil temperatures. Then, the weights obtained by that model were used to initialise a feedforward neural network model implemented in Lasagne. Feedforward neural networks trained in this way achieved an average error of **4.84%** when predicting soil temperatures.

A very insightful discovery was made by shuffling the training data instead of presenting it chronologically during training. This brought the average error down from **5.4%** to **4.69%**. When shuffling the training data as well as pre-training, the average error was reduced even further to **4.2%**.

Up to this point, all models were being trained to predict soil temperatures from contemporaneous ambient temperatures and humidities. The next step was to provide the model with some dynamic information in the form of previous measurements. Best results were obtained when the model was provided with the minimum and maximum temperature measurements of the previous day as inputs during training. This brought the average error down to **4.02%**. This was also the best result achieved for the Stellenbosch data.

Stellenbosch University https://scholar.sun.ac.za

7.1.2 Somerset West dataset

The Somerset West (Ssw.) dataset, which became available mid-way through the project, was considerably larger than the Stellenbosch dataset. In particular, the data spanned an entire year, and therefore provided scope to encapsulate the information of all four seasons. Hence the results obtained for this dataset might be viewed as more representative of a real-world practical application of our models.

In all cases, the performance of soil temperature prediction models for the Ssw. data were compared with a naïve guess, which is the result achieved when the soil temperature is taken to be the same as the ambient temperature. This naïve guess achieves an average error of **21.71%**.

A neural network trained on contemporaneous input was able to achieve an average error of **4.94%**. When the network was given the previous day's minimum and maximum temperature as additional inputs, the average error reduced to **4.45%**. This not only shows that a feed-forward neural network can make a much more accurate prediction than a simple guess, but also that it can achieve results that are accurate enough to be considered as a replacement for locally-installed soil temperature sensors.

It was however also shown that a model trained on the Ssw. data performed poorly when required to predict the soil temperatures of the Stb. site, with an average error of **18.56%**. This shows that models trained on data obtained from one particular location do not generalise well to another location. Further work is needed to determine how this mismatch can be minimised.

Up to this point all experiments focussed on estimating the soil temperature at a particular depth from ambient, but nevertheless local, temperature and humidity measurements. Such local measurements require sensors and associated communication infrastructure to be installed and maintained in the vineyard, and therefore are still expensive to obtain.

A next set of experiments therefore considered whether soil temperatures can be estimated on the basis of freely available satellite data, thereby eliminating

Stellenbosch University https://scholar.sun.ac.za

the need for locally placed sensors. This includes freely available satellite land surface temperatures, as well as hourly rainfall data collected from a local weather station. This model managed to achieve an average error of **5.67%** when using only satellite data and **5.52%** when considering also the rainfall data. The sparsity of the rainfall data might lead to the model assigning a very low importance to the rainfall input, leading to very small improvements in the results. This must be compared with the 4.94% error that is achieved when using temperature and humidity measured locally in the vineyard. Since no locally-measured data is used at all by this model, but local soil temperatures are expected from it, this good performance was a surprise and one could argue that it is the most important result of this study.

If more effort were invested into collecting freely available data, the model might achieve even better results. For example, other sources of satellite data could be considered. It might then be possible to train a model that is accurate enough to completely replace ground sensors and the associated measurement network in the vineyard. This could potentially save costs.

The neural network model was also trained using both local and satellite data. This model managed to achieve an average error of **4.5%**. This is a very good result, showing that the accuracy of the model continues to increase when given additional information.

With respect to the soil temperature prediction experiments, two key results can be re-emphasised. Firstly, when using freely available data from satellite and local weather stations, it is possible to achieve performance that is almost as good as that achieved using local measurements. Secondly, it was established that currently, the presented models are not good at generalising to geographical locations different from those they were trained on.

7.2 Moisture content prediction

A second focus of this thesis was the prediction of soil moisture content.

As a naïve guess, the average moisture content in the training dataset was

Stellenbosch University https://scholar.sun.ac.za

used. This naïve guess achieved an average error of **9.42%**.

For this task, the best neural network model achieved an error that was only slightly better at **8.28%**.

For the soil moisture experiments, the training set was extremely limited and it is likely that these results would improve if more data was available.

7.3 Bud-burst date prediction

A final focus of this study was an attempt to predict the bud-burst date, which is the date at which the vineyard buds leave their dormant state for the first time after winter. Three model architectures were tested, namely, linear regression, second order polynomial regression, and a feedforward neural network. The training data was again extremely limited (even more so than with the moisture data).

Using the previous year's bud-burst date as a naïve guess, an average error of **8.75 days** was achieved. The linear regression model achieved an average error of **6.21 days**, with its worst guess off by 15.46 days. The polynomial regression model achieved an average error of **4.15 days**, with its worst guess off by 10 days. Finally, the neural network model achieved an average error of **4.95 days**, with its worst guess off by 9.78 days.

Although the training data was extremely limited, these experiments indicate that the models are indeed learning to predict the bud-burst dates from the data.

To take this experiment further, more data needs to be collected. It may very-well be possible to train a model to make reasonable bud-burst predictions, but not with only 8 training examples. Although the polynomial model achieved the best results here, the neural network might achieve better results when more training examples are available.

Stellenbosch University https://scholar.sun.ac.za

7.4 LSTMs

LSTM recurrent neural networks were set up to perform one-step-ahead prediction of soil temperatures. Each training example consisted of 60 consecutive soil temperature measurements, and the model was asked to predict the 61st measurement. At first, the model exhibited poor performance, achieving an average error of just **19.88 %** over 100 runs. After removing discontinuities due to faulty measurements from the data, however, performance improved dramatically to an average error of **3.18 %**.

Finally, the model was trained using 60 consecutive ambient temperatures as input to predict the soil temperature at the 61st time step. This experiment is the most closely comparable with the previous experiments performed using feedforward neural networks. Once discontinuities had again been removed from the data, an average error of **4.6 %** was achieved. When the humidity was provided as an additional input, performance improved to an average error of **4.35 %**.

These results show that there is definitely potential in using LSTMs for soil temperature prediction. These models achieved better results than the feedforward neural networks considered in this study and did so with a relatively small dataset (7000 points). It would seem, however, that these models are much more sensitive to discontinuities in the data. This means that data collection and preparation would need to be much more precise if recurrent neural networks are used. The model will lose a lot of accuracy whenever sensors stop working properly and cause discontinuities in the data.

7.5 Future work

This study has shown that the machine learning algorithms used here have great potential in predicting environmental variables. Although the data was not collected specifically for machine learning, the algorithms were able to learn patterns from it. There are various different avenues for pursuing further research using the techniques described in this study. Some of the avenues one can pursue to achieve better results, or build upon this research are listed below:

Stellenbosch University https://scholar.sun.ac.za

- Additional input information can be provided to the neural networks. This study was limited to the data that was available. There could be other measurable environmental variables that might increase neural network model performance. For example, soil properties such as the soil composition, distance to the ocean, or even geographical coordinates should there be enough data from multiple study sites. These type of properties might also increase these models' capacity to generalise across different vineyards.

- Data collection quality can be improved. When considering the raw of data we obtained for this study, it seemed extensive. However, after removing faulty measurements to prepare the data for machine learning application, a much smaller dataset remained. Furthermore, discarding faulty measurements left gaps in the data and these gaps can cause problems for machine learning models. This was especially true in the case of the LSTM models, where the discontinuities almost completely prevented the model from fitting to the data. The corruption of the data was in most cases due to sensors becoming faulty. These faulty measurements are sometimes difficult to detect. By reducing the incidence of such corrupt datapoints, the scope of application of machine learning methods will be improved.

- Data collection quantity can be increased. This point is related to the previous one. Machine learning models generally benefit from more training data. This is particularly true for the soil moisture and bud-burst date prediction experiments. Even though the results are better than a naïve guess, the dataset is too small to take these results as representative of the real-world problem. More data is needed to train and validate the methods.

- More variations of LSTM models can be considered. The LSTM models showed great potential in this study, even though the dataset was quite small. As briefly mentioned in Section 3.3, there are many variations of LSTM networks and perhaps some of these would be better suited to our problem.

- Machine learning models are good at finding abstract patterns in data that we humans cannot always see. With the right input data, models could potentially learn to predict more abstract or high-level properties. Such properties could include things such as measurable taste properties (e.g.

Stellenbosch University https://scholar.sun.ac.za

acidity), general wine quality, or even the yield size. The datasets required for this research are however not currently available.

These are just a few potential applications directly applying machine learning on viticulture data. With continued cooperation with viticulturists, machine learning could bring great benefits to the industry.

Stellenbosch University https://scholar.sun.ac.za

Bibliography

[1] A. Aquino, M. P. Diago, B. Millán, and J. Tardáguila, "A new methodology for estimating the grapevine-berry number per cluster using image analysis," *Biosystems Engineering*, vol. 156, pp. 80–95, 2017.

[2] F. Avila, M. Mora, C. Fredes, and P. Gonzalez, "Shadow detection in complex images using neural networks: application to wine grape seed segmentation," in *International Conference on Adaptive and Natural Computing Algorithms*. Springer, 2013, pp. 495–503.

[3] C. M. Bishop, *Pattern Recognition*, M. Jordan, J. Kleinberg, and B. Schölkopf, Eds. Springer, 2006.

[4] C. M. Bishop *et al.*, *Neural networks for pattern recognition*. Oxford university press, 1995.

[5] M. Bridge, "Neutron probes and soil moisture," https://www.naturalresources.sa.gov.au/samurraydarlingbasin/publications/neutron-probes-and-soil-moisture, August 2015, online; accessed August 2018.

[6] S. Burgos, N. Dakhel, and M. Docourt, "Le comportement thermique des sols : caractérisation et influence sur la vigne." *Global warming, which potential impacts on the vineyards?*, March 2007.

[7] V. Casser, "Using Feedforward Neural Networks for Color Based Grape Detection in Field Images," 2016, online; Accessed November 13, 2018.

[8] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

Stellenbosch University https://scholar.sun.ac.za

[9] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*, 2nd ed. New York: Wiley, 2001.

[10] T. Gevers and A. W. Smeulders, "Color-based object recognition," *Pattern Recognition*, vol. 32, no. 3, pp. 453–464, 1999.

[11] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010.

[12] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011.

[13] S. G. Gujjar and S. Angadi, "Plant Clinic - A Mobile App for Grape Plant Disease Recognition and Remedies," *IJSRD - International Journal for Scientific Research & Development*, vol. 3, no. 11, 2016.

[14] G. Hinton, "Neural Networks for Machine Learning," https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf/, online; accessed 05-Feb-2019.

[15] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, July 2012.

[16] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[17] A. W. R. Institute, "Bud dormancy and budburst," https://www.awri.com.au/wp-content/uploads/1_phenology_bud_dormancy_and_budburst.pdf, online; accessed August 2018.

[18] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[19] G. Li, Z. Ma, and H. Wang, "Image recognition of grape downy mildew and grape powdery mildew based on support vector machine," in *The proceesings of the International Conference on Computer and Computing Technologies in Agriculture*, 2011.

Stellenbosch University https://scholar.sun.ac.za

[20] A. Matese, S. Di Gennaro, A. Zaldei, L. Genesio, and F. Vaccari, "A wireless sensor network for precision viticulture: The NAV system," *Computers and Electronics in Agriculture*, no. 69, pp. 51–58, 2009.

[21] A. Meunkaewjinda, P. Kumsawat, K. Attakitmongcol, and A. Srikaew, "Grape leaf disease detection from color imagery using hybrid intelligent system," in *5th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, 2008.

[22] N. Nair and G. Hill, "Bunch rot of grapes caused by Botrytis cinerea," *Plant Diseases of International Importance*, vol. 3, pp. 147–169, 1992.

[23] Y. Nesterov *et al.*, "Gradient methods for minimizing composite objective function," http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.332.931&rep=rep1&type=pdf, July 2007, available online; Accessed Jan 2019.

[24] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015.

[25] S. Nuske, S. Achar, T. Bates, S. Narasimhan, and S. Singh, "Yield estimation in vineyards by visual grape detection," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2011*. IEEE, 2011, pp. 2352–2358.

[26] S. Nuske, K. Gupta, S. Narasimhan, and S. Singh, "Modeling and calibrating visual yield estimates in vineyards," in *Field and Service Robotics*. Springer, 2014, pp. 343–356.

[27] C. Olah, "Understanding LSTM Networks," http://colah.github.io/posts/2015-08-Understanding-LSTMs, August 2015, online; accessed July 2018.

[28] Parasgr7, "Recurrent Neural Network (LSTM) with Keras Framework," https://github.com/Parasgr7/Google-Stock-Price-Prediction.

[29] D. S. Pérez, F. Bromberg, and C. A. Diaz, "Image classification for detection of winter grapevine buds in natural conditions using scale-invariant features transform, bag of features and support vector machines," *Computers and Electronics in Agriculture*, vol. 135, pp. 81–95, 2017.

Stellenbosch University https://scholar.sun.ac.za

[30] M. Romero, Y. Luo, B. Su, and S. Fuentes, "Vineyard water status estimation using multispectral imagery from an UAV platform and machine learning algorithms for irrigation scheduling management," *Computers and Electronics in Agriculture*, vol. 147, pp. 109–117, 2018.

[31] S. S. Sannakki, V. S. Rajpurohit, V. Nargund, and P. Kulkarni, "Diagnosis and classification of grape leaf diseases using neural networks," in *Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*. IEEE, 2013, pp. 1–5.

[32] S. Shafian and S. J. Maas, "Index of soil moisture using raw Landsat image digital count data in Texas high plains," *Remote Sensing*, vol. 7, no. 3, pp. 2352–2372, 2015.

[33] T. Southey, "Integrating climate and satellite remote sensing to assess the reaction of Vitis vinifera L. cv. Cabernet Sauvignon to a changing environment," Ph.D. dissertation, Stellenbosch University, March 2017.

[34] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research*, vol. 15, no. 1, 2014.

[35] B. E. Stummer, I. L. Francis, T. Zanker, K. A. Lattey, and E. S. Scott, "Effects of powdery mildew on the sensory properties and composition of Chardonnay juice and wine when grape sugar ripeness is standardised," *Australian Journal of Grape and Wine Research*, vol. 11, no. 1, pp. 66–76, 2005.

[36] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International Conference on Machine Learning*, 2013.

[37] K. Thome, "Terra, The EOS flagship," http://terra.nasa.gov, NASA, online; accessed July 2018.

[38] D. P. Vazquez, F. O. Reyes, and L. A. Arboledas, "A comparative study of algorithms for estimating land surface temperature from AVHRR data," *Remote Sensing of Environment*, vol. 62, no. 3, pp. 215–222, 1997.

Stellenbosch University https://scholar.sun.ac.za

[39] H. Waghmare, R. Kokare, and Y. Dandawate, "Detection and classification of diseases of Grape plant using opposite colour Local Binary Pattern feature and machine learning for automated Decision Support System," in *3rd International Conference on Signal Processing and Integrated Networks (SPIN), 2016*. IEEE, 2016, pp. 513–518.

[40] Z. Wan, "MODIS land-surface temperature algorithm theoretical basis document (LST ATBD)," *Institute for Computational Earth System Science, Santa Barbara*, vol. 75, 1999.

[41] M. D. Zeiler, "ADADELTA: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[42] A. Zuñiga, M. Mora, M. Oyarce, and C. Fredes, "Grape maturity estimation based on seed images and neural networks," *Engineering Applications of Artificial Intelligence*, vol. 35, pp. 95–104, 2014.

Stellenbosch University https://scholar.sun.ac.za

Appendix A

A brief intro to Theano and Lasagne

This section will give a very brief introduction to Theano and Lasagne. These are basic software tools used extensively in this thesis.

A.1 Theano

The best short description of Theano comes from the documentation of Theano itself:

"*Theano is a Python library that lets you to define, optimise, and evaluate mathematical expressions, especially ones with multi-dimensional arrays (numpy.ndarray). Using Theano it is possible to attain speeds rivaling hand-crafted C implementations for problems involving large amounts of data. It can also surpass C on a CPU by many orders of magnitude by taking advantage of recent GPUs.*"

Theano is not a programming language itself, since you still code in Python, but like a programming language you will have to define variable names and types, build expressions describing how to use these variables together, and compile these expressions to use them for computation. The next part will show three examples of different things Theano can be used for.

First, we will start with setting up a basic algebraic expression.

Stellenbosch University https://scholar.sun.ac.za

```
import theano
from theano import tensor

# declare two symbolic floating-point scalars
a = tensor.dscalar()
b = tensor.dscalar()

# create a simple expression
c = a + b

# convert the expression into a callable object that takes (a,b)
# as input and computes a value for c
f = theano.function([a,b], c)

# bind 1.5 to 'a', 2.5 to 'b', and evaluate 'c'
x = f(1.5, 2.5)
x == 4.0
> True
```

In this example the variable type "dscalar" was used, but there are ofcourse many different variable types one can use including different matrix-type variables.

The next example will show the reader how to use Theano to compute and evaluate the logistic (sigmoid) function given as:

$$s(x) = \frac{1}{1 + e^{-x}} \tag{A.1}$$

The goal is to apply the function to each individual element of a matrix of doubles. The following code excerpt shows this can be achieved:

```
import theano
import theano.tensor as T

# define a variable as a matrix of doubles
x = T.dmatrix()

# define the expression for the function
s = 1 / (1 + T.exp(-x))
```

Stellenbosch University https://scholar.sun.ac.za

```
logistic = theano.function([x], s)
logistic([[0,1], [-1, -2]])

# this yields a 2x2 array:
>>>array([[ 0.5       ,  0.73105858],
       [ 0.26894142 , 0.11920292]])
```

In the previous example x is defined as a "dmatrix('x')". The string "x" added to the definition simply adds a tag or name to the variable x which can be used later on for debugging purposes.

Theano also allowsone to define a function with an internal state using something called shared variables. For instance, consider an accumulator. At the beginning, the state is initialised to zero. Then, on each function call, the state is incremented by the functions argument.

Let us define the accumulator function. It adds its argument to the internal state, and returns the old state value.

```
from theano import shared

# create a shared variable called 'state' and an integer scalar
# named 'inc'
state = shared(0)
inc = T.iscalar('inc')

# define the accumulator function taking 'inc' as input, returning
# 'state' as output, as well as updating the variable state with
# the value of state+inc when called
accumulator = function([inc], state, updates=[(state, state+inc)])
```

This code introduces a few new concepts. The shared function constructs so-called shared variables. These variables' value may be shared between multiple functions. The other new thing in this code is the "updates" parameter of function. It must be supplied with a list of pairs of the form (shared-variable, new expression). It can also be a dictionary whose keys are shared variables and values are the new expressions. Either way, it means "whenever this function runs, it will replace the

Stellenbosch University https://scholar.sun.ac.za

value of each shared variable with the result of the corresponding expression." In the code above, our accumulator replaces the state's value with the sum of the state and the increment amount.

Using the code from above:

```
>>> print(state.get_value())
0
>>> accumulator(1)
array(0)
>>> print(state.get_value())
1
>>> accumulator(300)
array(1)
>>> print(state.get_value())
301
```

The last example will show how one goes about computing the derivative of a simple function, $f(x) = x^2$.

```
import theano
from theano import tensor as T

x = T.dscalar()
y = x ** 2

gy = T.grad(y, x)

f = theano.function([x], gy)

>>> f(4)  # yields:
array(8.0)
```

As seen from the few examples above, Theano can be very helpful when working with symbolic expressions and their derivatives. Since this is what you are constantly working with when constructing neural networks, one can see why you would use Theano as a base to build upon.

Stellenbosch University https://scholar.sun.ac.za

A.2 Lasagne

The next library used in these experiments is called Lasagne. This library is strongly dependent on Theano since it constantly uses its symbolic variable expressions. In this example, we will build a small neural network to try and classify the digits given by the MNIST dataset. This network will be similar to the smallest network used by Geoffrey Hinton in his 2012 paper, "*Improving neural networks by preventing co-adaptation of feature detectors*" [15]. Only certain extracts of the code will be shown in order to convey the idea behind Lasagne. For a more complete tutorial on Lasagne, visit the Lasagne documentation website at `lasagne.readthedocs.io`.

When using Lasagne it is important to note that you will have to import Theano as well as numpy alongside Lasagne. While Lasagne is built on top of Theano, it is meant as a supplement to it and not a replacement. You will still use small parts of Theano code alongside your Lasagne code.

```
import numpy as np
import theano
import theano.tensor as T

import lasagne
```

Lasagne helps you to define an arbitrarily structured neural network by stacking multiple layers together. The first layer will always be an input layer. The neural network must have a function in which the outline of the network is constructed. Here that function is called "build_mlp()". Other hyper-parameters, as well as the training process are accessed in another function. The following begins the definition of the "build_mlp()" function and defines the input layer of the network.

```
def build_mlp(input_var):
    l_in = lasagne.layers.InputLayer(shape=(None, 1, 28, 28),
                                     input_var=input_var)
```

The numbers in the shape tuple represents, in order: (batchsize, channels, rows,

Stellenbosch University https://scholar.sun.ac.za

columns). With the batchsize set to None the network will accept input data of any size. There is only one channel since these are black and white images, and the 28 rows and columns represent the pixel size of the the MNIST characters (28x28).

Before adding the first hidden layer, we'll apply 20% dropout to the input data. Dropout is a process in which neurons in a layer are randomly eliminated with a certain probability. This technique was first used by Geoffrey Hinton in the same 2012 paper [15] in an attempt to prevent neurons from becoming dependent on other neurons in the same hidden layer and relying on them to learn certain features in the data. To add a dropout layer, one simply uses the following:

```
l_in_drop = lasagne.layers.DropoutLayer(l_in, p=0.2)
```

It takes only two parameters. The first is to link the layer to the correct previous layer - in this case the input layer. The second parameter, p, is the dropout probability.

Next, we add the first (fully connected) hidden layer known in Lasagne as a dense layer. Then we add a dropout layer to the first hidden layer, followed by another hidden layer, as well as a dropout layer for the second hidden layer. Finally, we will add the output layer. A brief explanation will follow after the code is shown.

```
# 1st hidden layer with 800 neurons and rectified linear non-linearity
l_hid1 = lasagne.layers.DenseLayer(
      l_in_drop, num_units=800,
      nonlinearity=lasagne.nonlinearities.rectify,
      W=lasagne.init.GlorotUniform())

# Dropout layer with dropout probability of 0.5
l_hid1_drop = lasagne.layers.DropoutLayer(l_hid1, p=0.5)

# 2nd hidden layer with 800 neurons, ReLU non-linearity
l_hid2 = lasagne.layers.DenseLayer(
  l_hid1_drop, num_units=800,
```

Stellenbosch University https://scholar.sun.ac.za

```
    nonlinearity=lasagne.nonlinearities.rectify)

# Dropout layer
l_hid2_drop = lasagne.layers.DropoutLayer(l_hid2, p=0.5)

# Output layer with 10 neurons (outputs) and softmax non-linearity
l_out = lasagne.layers.DenseLayer(
    l_hid2_drop, num_units=10,
    nonlinearity=lasagne.nonlinearities.softmax)

    return l_out
```

Notice that every layer is defined with a parameter that links it to the previous layer. Then it is only necessary to return the output layer and lasagne can still access the whole network. The num_units parameter defines the number of neurons in the layer. The nonlinearity parameter will determine the activation function that the network will apply to the values of that layer, in this case we use the Rectified Linear Unit or ReLU as coined by Glorot et. al. in 2011 [12]. The last parameter, W, is the weight initialization method. This method initialises the values of the weights before the model starts training. The exact values of the weights are usually randomised values spread over a certain distribution depending on which initialization method was used. Lasagne has many options built in, but "GlorotUniform" is the default and therefore gets omitted after the first layer since it's not needed unless a different initialization is desired.

We can now define the *main()* function. It begins like this:

```
# Load the dataset
X_train, y_train, X_val, y_val, X_test, y_test = load_dataset()
# Prepare Theano variables for inputs and targets
input_var = T.dmatrix('inputs')
target_var = T.dmatrix('targets')
# Create neural network model
network = build_mlp(input_var)
```

The first line loads the data into numpy arrays. This function is not part of Lasagne and must be provided yourself. It then defines symbolic theano variables

Stellenbosch University https://scholar.sun.ac.za

for the input and target variables. They are not tied to any data yet and their dimensionality will depend on what you link it with later. We then call the *build_mlp()* defined earlier to create a variable for the network.

The next step is to define the loss and update expressions to be minimised in training.

```
# Generate Theano expression for network output given input variable linked to
# the network's input layer.
prediction = lasagne.layers.get_output(network)
# Define Theano expression for the categorical cross-entropy loss between networ
# output and targets.
loss = lasagne.objectives.categorical_crossentropy(prediction, target_var)
# Take the mean of the loss to get scalar value
loss = loss.mean()

#  Collect all Theano SharedVariable instances making up the trainable parameter
params = lasagne.layers.get_all_params(nework, trainable=True)
# Generate an update expression for each parameter
updates = lasagne.updates.nesterov_momentum(
        loss, params, learing_rate=0.01, momentum=0.9)
```

Here, we get the network's current prediction by simply propagating the input through the network. We then measure the loss by in this case using the cross-entropy function to measure the loss since we are dealing with a classification problem. We then transform the loss into a scalar value by taking it's mean. We then create a theano variable, params, which contains all the weights and biases for the network. The next line will then cause the network to update these parameters by using a certain update function (nesterov-momentum in this case).

After all of these Theano expressions are defined we can compile a function to performing a training step:

```
train_fn = theano.function([input_var, target_var], loss, updates=updates)
```

This tells Theano to generate and compile a function taking two inputs (a mini-batch of images and a vector of corresponding targets) and returning a single output (the training loss). Additionally, each time it is invoked, it applies all

Stellenbosch University https://scholar.sun.ac.za

parameter updates in the updates dictionary (as shown in the Theano shared variable explanation in section A.1), thus performing a gradient descent step with Nesterov momentum. For validation, we compile a second function:

```
val_fn = theano.function([input_var, target_var], [test_loss, test_acc])
```

This one also takes a mini-batch of images and targets, then returns the (deterministic) loss and classification accuracy, not performing any updates.

Finally, we can write the training loop:

```
for epoch in range(num_epochs):
    # In each epoch, we do a full pass over the training data:
    train_err = 0
    train_batches = 0
    start_time = time.time()
    for batch in iterate_minibatches(X_train, y_train, 500, shuffle=True):
        inputs, targets = batch
        train_err += train_fn(inputs, targets)
        train_batches += 1

    # And a full pass over the validation data:
    val_err = 0
    val_acc = 0
    val_batches = 0
    for batch in iterate_minibatches(X_val, y_val, 500, shuffle=False):
        inputs, targets = batch
        err, acc = val_fn(inputs, targets)
        val_err += err
        val_acc += acc
        val_batches += 1
```

At the very end, we re-use the "val_fn()" function to compute the loss and accuracy on the test set, finishing the script.