Bachelor thesis

Artificial Intelligence

Radboud University

Forecasting greenhouse cucumber production using a combined LSTM-TCN deep neural network.

Author:
Yrsa Kleijkers
s1015940

First supervisor:
dr. S (Silke) Hemming
Wageningen University and Research - scientific research team Greenhouse Technology
silke.hemming@wur.nl

Second supervisor:
Prof. T.M. (Tom) Heskes
Computer Science and Artificial Intelligence - Radboud University
tom.heskes@ru.nl

Third supervisor:
AS (Anna) Petropoulou MSc
Wageningen Plant Research - Wageningen University and Research
anna.petropoulou@wur.nl

January 30, 2022

Abstract

A deep neural network (DNN) is proposed that combines a long short-term memory recurrent neural network (LSTM-RNN) with a temporal convolutional network (TCN) to forecast weekly greenhouse cucumber production. This DNN is expected to overcome the challenges that come along when forecasting greenhouse crop production, like biological unpredictability, behavioral delay in the crop's response and short and long-term relationships that exist between the crop and the different conditions in the greenhouse. The focus will be on using predictors that can be measured inside the greenhouse to forecast the weekly harvest value, with one or two weeks as input data. Due to limited amount of real-life data, synthetic data is used to optimize and train the LSTM-TCN DNN. This also means that this research mainly focuses on providing a proof of concept rather then creating a ready to use application. The performance of the LSTM-TCN forecasting, expressed by the mean square error (MSE), is compared to using only a LSTM-RNN DNN. The results show an decrease of 69% and 38% in MSE, although not significant, when using the combined LSTM-TCN DNN compared to using a standalone LSTM-RNN DNN for one and two weeks of input data respectively. The results of this Bachelor of Sciences (BSc) thesis research show the potential of using a LSTM-TCN DNN for forecasting greenhouse cucumber production.

Contents

Acronyms

Deep neural network (DNN)
Bachelor of Sciences (BSc)
Long Short-term Memory recurrent neural network (LSTM-RNN)
Long Short-term Memory recurrent neural network (LSTM)
Temporal convolutional neural network (TCN)
autoregressive integrated moving average (ARIMA)
Recurrent Neural Network model using a GRU cell (RNN-GRU)
Convolution Neural Network (CNN)
Random Forest (RF)
Support Vector Regression (SVR)
Mean square error (MSE)
Cross validation (CV)

Chapter 1

Introduction

The global food demand is increasing and at the same time resources for growing fresh food are becoming scarcer [17]. The greenhouse industry could play an important role in this challenge, as it allows high crop production with high efficiency in the use of resources like water [32]. Next to this, greenhouse production allows for year round production with less influences by outside climate conditions and can therefore be a secure producer of fresh food.

Wageningen University and Research organised an international competition on autonomous greenhouse management, with the aim to combine horticultural expertise with artificial intelligence to grow cucumbers with efficient resource use [22]. This competition led successfully to several algorithms that could automate greenhouse cucumber cultivation, with some even outperforming the manually-grown reference production. However, accurate information on expected crop production is often missing, as it is now heavily relying on human expertise. Being able to accurately predict greenhouse production allows for farming planning and management, while using the available resources optimally and while making financial related decisions, to meet timely market demands.

The main challenges in forecasting greenhouse crop production lie in the complexity of the crop, namely its biological unpredictability and its delay in behavior. For example, a heavy temperature increase for several hours can lead to plants being unable to recover, but also plants that are able to recover and therefore will still produce cucumbers. Where this point of being unable to recover per plant is, is still unknown and this makes it a difficult challenge. Next to this do there exist both long and short-term relationships between the variables that all need to be captured by the DNN. For example, the effect of increasing indoor greenhouse air temperature can become visible on the plants some hours later but effects of this increase can appear several days later as well. Another challenge is the multivariate time-series data, this data is generated containing variables that have dependencies within themselves and also between other variables over time. Next to this is often seasonality present that should be removed from the data to prevent having meaningful patterns hid. These challenges make it difficult to create an explicit model that can reflect these complex relationships between the variables and the crop production. Due to technological advances more sensors are present in the greenhouses that provide more data. Therefore new data-driven techniques like Artificial Intelligence and more specifically deep neural networks (DNN) are gaining a lot of attention, also because they can capture more complex patterns, like non-linear relationships.

This research gives a proof of concept in forecasting greenhouse cucumber production with a DNN that is optimized, trained and tested on synthetic data coming from the combined deterministic model INTKAM and KASPRO [3] [9]. The DNN consists of a combination of a long short-term memory - recurrent neural network (LSTM-RNN, LSTM in short) and a temporal convolutional network (TCN). The DNN is based on the DNN that was build by a recent research study by Gong, L. et al. where greenhouse tomato production was being predicted with promising results [19]. In the rest of this paper you can read about the following:

1. The phenomena 'future data leakage' and how it can be prevented when working with time-series data (chapter 2, section 2.1).
2. Background information on LSTM-RNNs and TCNs, what they are and how they work (chapter 2, sections 2.2 and 2.3).
3. Related work on using DNNs, TCNs and LSTMs in time-series forecasting in general and specifically in greenhouse crop production forecasting (chapter 3, sections 3.1, 3.2 and 3.3).
4. Hypotheses of this BSc thesis research (chapter 3, section 3.4).
5. Synthetic data description, pre-processing and data engineering that has been executed (chapter 4, sections 4.1, 4.2.1, 4.2.2 and 4.2.4).
6. Population informed nested cross validation and creation of the train, validation and test samples (chapter 4, section 4.2.3).
7. The network structure and hyper-parameter optimization (chapter 4, section 4.3).
8. Results (chapter 4, section 4.5).
9. Conclusion and future work (chaper 5).

Chapter 2

Preliminaries

2.1 Future data leakage in time series data forecasting and how it should be prevented

When working with time-series data, future data leakage needs to be prevented during the optimization, training and evaluation of the model. Future data leakage is happening when data that comes from the future, is used to forecast a value that happened earlier in time. When future data leakage is happening, the algorithm has picked up patterns from the future and uses these patterns for forecasting the value. This, however, could give misleading good results. In real life, it is namely not possible to use data from the future for forecasting as it has not happened yet. To prevent this, test data needs to come from later in time compared to the training and validation data. This makes sure that training is not performed with future data and is ensured that hyper-parameters are not optimised based on future data. Also, for the same reason as mentioned before, should the normalization factors be calculated using training data only. These normalization factors can then be used on to normalize all the data, in this way future data leakage is prevented.

2.2 Long Short-term Memory - Recurrent Neural Network

Recurrent neural networks have been proven to be effective when using it on non-linear time-series data. However, when the time-series data contains delays and gaps, as well as long distance dependencies, the problem of vanishing and exploding gradients occur. The LSTM-RNN, in short LSTM, solves this problem. Therefore, they are often used when a recurrent network is needed nowadays [37]. A LSTM network consists of the following elements: (See figure 2.1 for a visualisation)

1. Cell state: The key element of the LSTM. The cell state is regulated by the forget, input and output gates.

2. Forget gate: A Sigmoid layer that looks at the output of the previous LSTM unit and the current input to the LSTM unit to decide what values to forget/keep. 1 means keep everything and a 0 means forget everything.

3. Input gate: A Sigmoid layer that decides what values of the cell state to update.

4. Tanh layer: Creation of new candidate values for the cell state.

5. The current cell state: The output of the forget layer is multiplied with the previous cell state value. This now contains the information that is being kept. Towards this, the multiplication of the input gate layer, a tahn layer, is added (the new information) and this forms the current cell state.

6. Output gate: A Sigmoid layer that decides what to output from the current cell state. The cell state is put through a tahn function and this is multiplied together.

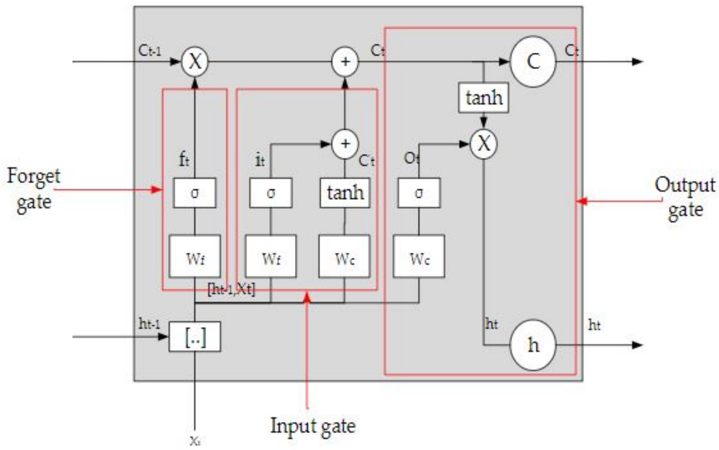

Figure 2.1: Visualization of a LSTM network [37]. Where the C_t is the current cell state, C_{t-1} is the previous cell state, σ represents a Sigmoid layer, W the weights, h_t the current output value and h_{t-1} the previous output.

2.3 Temporal Convolutional Networks

Convolutional networks are known for their use in image feature extraction. Also in time-series data convolutional networks, temporal convolutional networks (TCNs), can be meaningful. The TCN uses causal convolutions and dilations to be able to automatically find the inherent short and long-term patterns in sequential data. In this way each TCN layer holds different short and longer term patterns that can exist in the time-series data. To be able to have a TCN working correctly for sequential data, the following constraints exists for TCNs:

1. The convolutional layers should be causal. By using causal layers the network uses only information from time steps that happened in the past. This prevents future data leakage as discussed in section 2.1.

2. The TCN's output should be of equal length to the TCN's input. This makes sure that the TCN can have multiple convolutional layers on top of each other. In this way every layer can look at a different receptive field, the amount of time-steps that is looked at, to learn the different short and long-term patterns of the time-series data.

To make sure that these constraints are satisfied and the TCN is in its correct form to be able to be used in a DNN, the TCN should consist of and consider the following:

1. The TCN has a 1D fully-convolutional network architecture. This network takes as input a 3-dimensional tensor and also outputs a 3-dimensional tensor. The input shape is (#samples, input_length, input_size) and the output shape is (#samples, input_length, output_size). Where input_length is equal to the amount of time-points in one sample. And the input_size is equal to the amount of features available at every time-point.

2. The receptive field of the TCN is important. It is namely desirable to have full history coverage. This means that the whole input sample size will be used for forecasting. The following parameters have an influence on the receptive field of the TCN:

    (a) Kernel Size. The kernel size is the amount of time-points that is looked at simultaneously by the convolutional layers. See figure 2.2 for an TCN with 4 layers and kernel size 2. This TCN has a receptive field of 4 time-points.

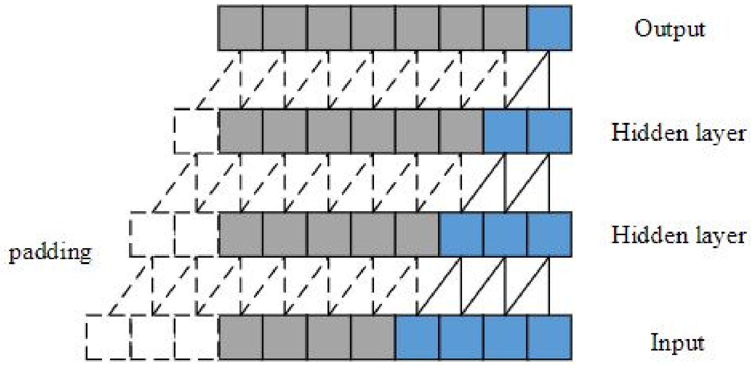

Figure 2.2: An example of a TCN with 4 convolutional layers with kernel size 2 [21].

(b) Dilation base. Dilation factors, dilated convolutions, are used to prevent explosion of the amount of layers when ensuring large enough receptive fields. Too many layers are undesirable because it will explode the number of parameters that have to be learned. Next to this, a lot of layers lead to degradation problems related to the gradient of the loss function. Dilation factors tell the filter how many time-points should be skipped in between the time-points used. Using dilated factors enable exponentially large receptive fields for the up following convolutional layers. Using exponential dilation factors reduce the complexity of the learning problem to a logarithmic scale instead of a linear scale. See figure 2.3 to see how dilation factors enable to have large receptive fields with a smaller amount of layers.

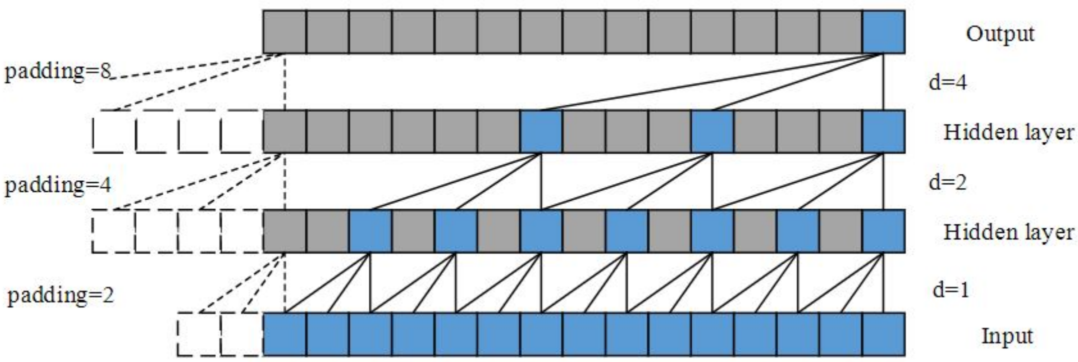

Figure 2.3: An example of a TCN with 4 convolutional layers, kernel size 3 and exponential dilation factors: 1,2,4 [21]. The receptive field is equal to 15 time-points.

(c) Number of convolutional layers. As stated in the previous two elements, the amount of layers needed to get full history coverage depends on the kernel size and the dilation factors. By having multiple layers, that all have a different dilation factor, it is possible that each layer can learn different kind of short/long-term dependencies that exist in the time-series data efficiently.

3. Zero-padding: Among the constraints of a TCN is that the input size should be equal to the output size. Therefore, zero-padding needs to be used. For example, with a kernel size of 3, two zeros need to be added to end up with the same output size.

Chapter 3

Related Work

3.1 Methods for forecasting time-series data

A lot of methods have been created to forecast time-series data as many real-world applications can be addressed as a forecasting time-series problem. Nowadays, these real-world applications mostly involve multivariate data, containing both long and short-term patterns. Explicit models and simple artificial neural networks are often not capable to capture such complex patterns in the data. DNNs are promising in addressing such forecasting problems, and are therefore, used more and more often.
Lai compares traditional and state-of-the-art baseline approaches like autoregressive integrated moving average (ARIMA) and Recurrent Neural Network model using a GRU cell (RNN-GRU) with the deep neural network LSTM [27]. Their LSTM used a combination of a Convolution Neural Network (CNN) and a Recurrent Neural Network to extract both short-term local dependencies between the variables and long-term patterns and trends of the time-series data. When working with complex mixtures of repetitive patterns, Lai found significant improvements in performance using a LSTM compared to the other models. The other models performed less on some of the aspects of these problems and therefore retrieved lower performance scores. For example, they identified that when scale fluctuations occur in the data the performance of the RNN-GRU drastically drops, while autoregression (AR) is robust to this. But the AR has a high computational cost and tended to overfit when forecasting with high dimensional time-series data, which was not happening for the RNN-GRU. They found empirical evidence for the success of LSTM models when both long and short-term dependencies really occur in the data. When these dependencies are not present, the performances are comparable to the other baseline models.
Inheritors of the LSTM have been built among them the Convolutional LSTM network [36]. These inheritors perform equally well on learning both the short and long-term patterns of time-series data. However, the LSTM

and its inheritors are all quite time demanding, because RNN models cannot be parallel, and they are non-robust for forecasting with periodic time-series data [16].
Other recent methods that have been created for forecasting time-series data are hybrid multi-scale approaches. Examples are Empirical Mode Decomposition (EMD), Ensemble EMD, multi-level wavelet decomposition network and Variational Mode Decomposition [11] [35] [34] [38]. These hybrid multi-scale approaches decompose the time-series data in the time-domain with every component having the original time-series data length. In this way, each component represents a different frequency within the data and can then be used to facilitate the forecasting. They are most useful in non-linear and non-stationary time series data. However, these methods are difficult to use in real-world applications, like greenhouse yield forecasting, as a decomposition value needs to be provided as a hyper-parameter that is difficult to estimate when working with complex multivariate time-series data.
Research has been done to find other approaches that do not face the aforementioned challenges as for both LSTMs and hybrid multi-scale approaches. TCNs have been proposed due to their prominent characteristics of casualness in the convolution architecture design and in the sequence length, as well as their overall efficiency.

3.2 Temporal Convolutional Networks used with time-series data

In their paper, Wan compared the performances of a TCN with a LSTM network in forecasting PM2.5 and electric power [16]. Their decision to use a TCN stems from the limitations that a LSTM brings along and the prominent characteristics of a TCN, as mentioned in the previous section. Next to this, is a TCN suitable for building a very deep and wide network. This can be done by using convenient combinations of residual blocks and convolutional components (see chapter 2, section 2.3 for an elaborate explanation of a TCN architecture and its components). A deep and wide network that is still efficient is very useful for learning complex relations in the data, but is at the same time good in generalisation. In their work, Wan decided to use a multi-headed TCN to receive more robust results. A multi-headed TCN creates sub-models for each of the variables in the data and combines the results of each sub-model. Those combined results are put though fully connected layers, to come to a final prediction. Results indicated that the TCN performed better and had a lower implementation complexity compared to the LSTM, especially for larger horizons.

3.3 Deep Neural Networks that are used to forecast greenhouse harvest yield

DNNs have been largely applied in the field of greenhouse yield forecasting, as DNNs retrieve better performances compared to the traditional machine learning methods and empirical models.

In their research, Alhnaity compared performances of traditional machine learning algorithms like Support Vector Regression (SVR) and Random Forest (RF) with LSTM for greenhouse tomato yield prediction [1] [8] [5] [23]. The comparison of the methods was carried out using the Mean Square Error (MSE) scores as evaluation metric. They found that the MSE of the LSTM model was around 0.002 and the SVR and RF around 0.015 and 0.040 respectively, indicating a much better performance of the LSTM model.

Another recent and promising research for the greenhouse field is done by Gong [19]. Gong tested a DNN that consisted of a combination of a LSTM followed by a TCN part to predict greenhouse tomato production. Results showed that the LSTM-TCN DNN yielded improved results compared to a standalone LSTM or TCN. The DNN was trained using daily information of parameters such as CO2 concentration (ppm), temperature (°C), humidity deficit (g/kg), relative humidity (percentage) and radiation (W/m^2) from inside the greenhouse and also historical yield information (g/m^2). They found that the historical yield information was the most informing predictor for their forecasting. They decided to use one LSTM layer with 20 LSTM units and one TCN block with 250 filters. They trained the combined LSTM-TCN DNN on 3 data sets and had for all the three data sets the training MSE evolve to a value of zero. The combined LSTM-TCN DNN obtained mean test MSE scores of 0.11, 0.04 and 0.05 for the three data sets respectively.

3.4 Hypotheses

Because of the promising results of the combined LSTM-TCN network architecture on forecasting greenhouse tomato production created by Gong, a similar architecture will be implemented in this thesis to forecast greenhouse cucumber production [19]. Gong found that 'previous yield information' was the most important predictor for his model to forecast the tomato yield. However, in this research it is decided not to use this variable as one of the predictors. The decision was made considering that this information is not always available. For example, forecasting harvest in 5 weeks from now, is difficult without employing other models, as intermediate weekly yield values up to the fifth week are unknown. The other variables, like indoor air temperature, are controllable by greenhouses growers and are therefore easier to estimate for future situations. By predicting the harvest for a less complex crop like cucumbers and by the optimization of the hyper-parameters, it is expected that historical yield can be left out and still retrieve a low MSE when predicting the harvest using the combined LSTM-TCN DNN. This research focuses on the following hypotheses:

1. The combined LSTM-TCN deep neural network will retrieve a lower Mean Square Error compared to a LSTM deep neural network.

2. Model generated- synthetic data of variables: Air Temperature (ºC), Air temperature sum, CO2 Concentration (ppm), PAR (photosynthetic active radiation mol/m^2/s), PAR sum, Plant Load (fruits per m2) and number of days since planting is expected to contain the necessary information to forecast greenhouse cucumber production. The predictors were selected based on expert knowledge in the availability of the parameters in an actual as well as the available virtual greenhouse environment.

Chapter 4

Research

4.1 Data description

The synthetic data used for training and testing the DNN is generated using the combined INTKAM - KASPRO deterministic models [3] [9]. The different scenarios were created using different heating temperature trajectories, changing offsets on the heating trajectories, varying dosing capacities and set-points of CO2, changing patterns of fruits to be maintained on the plants, as well as, varying stem densities. For a precise description of the variables created by the simulation model see table 4.1. In total 320 scenarios were created. Every scenario has 3816 observations, which comes down to approximately 168 days of time-series data. The variable 'Harvest' is provided as daily data, the other variables are all hourly data. So in total the original synthetic data contained 1221118 observations for the 9 variables listed in table 4.1. The variable 'Harvest' contains the information to be predicted, the other variables can be used as predictors.

4.2 Pre-processing

Before the created synthetic data could be applied to a model, prepossessing had to be done. The upcoming subsections describe the pre-processing steps that have been executed on the original data:

1. Data engineering: Creation of additional variables (subsection 4.2.1).
2. Justification of the variable selection to be used as predictors and why certain variables are left out (subsection 4.2.2).
3. The cross validation technique that is used and how train, validation and test samples are created with the correct shape (subsection 4.2.3).
4. Normalization technique used (subsection 4.2.4).

Table 4.1: Description of the original variables

Name	Description	Unit	Min	Max	Mean	Variance
Scenario	Scenario number	Scenario Number	1	320	-	-
DaysAfterPlanting	The amount of days that have past since the start of the scenario	Days	0	159	-	-
Week	The week number in the year	Week Number	15	38	-	-
Date	The date	Date Number	44300.04	44459.0	-	-
Tair	Temperature of the air inside the greenhouse	°C	13.79	37.14	22.30	9.67
CO2	Carbon Dioxide Concentration	ppm	255.43	712.87	440.17	1732.55
PAR	Photosynthetic Active Radiation	μmol/m^2/s	0.0	1529.89	295.12	132387.35
PlantLoad	Number of fruit per squared meters	# fruits per m2	0.0	26.0	13.32	67.80
Harvest	The amount of cucumbers that have been harvested per day	kg/m2	0.0	$9.16 \cdot 10^{-1}$	$1.35 \cdot 10^{-2}$	$5.76 \cdot 10^{-3}$

4.2.1 Additional variables

Additional variables have been created, using only the values of the originally synthetic data, to improve forecasting. When forecasting greenhouse cucumber production, it is useful to have an indicator of where in the growth cycle the cucumber production is. This helps the model to forecast the cucumber production and prevents it from predicting yield at moments where this is not possible, like at the very beginning of a growth cycle. The variable DaysAfterPlanting already serves as an growth stage indicator. Next to this, the total amount of photosynthetic active radiation and temperature exposed to the cucumber plants, are used as they express past cucumber growing conditions [28] [22]. Therefore it is decided to include two additional variables, called Tair_total and PAR_total. These variables contain the summed amount of temperature and PAR from the beginning up to that time-point in the time-series data. They indicate what the conditions were during the growth cycle of the cucumbers in the greenhouse. In this way it is expected that they provide valuable information for the model to be used when forecasting the cucumber production.
Next to this is a variable created that contains a weekly variant of the variable 'Harvest', the variable to be predicted. In the original data this variable is given per day. Since the daily harvest is, next to the growing stage of the cucumber, also dependent on manual interventions as the moment and frequency of harvest, this variable is transformed to a weekly harvest variable. In this way the influence of these random factors is lowered. The variable gets the name SlideCumHarvest and contains at every time point in the scenario the total summed harvest of the past 7 days (a week). For the time points that do not have 7 days of data available before that time point the value 0 is given. This means that the variable that will be predicted is the total summed harvest of the past week at a specific time-point. See table 4.2 for a precise description of the added variables.

Table 4.2: Description of the additional variables

Name	**Description**	**Unit**	**Min**	**Max**	**Mean**	**Variance**
Tair_total	Total accumulative inside air temperature	°C	18.56	91935.79	42081.52	599967-667.60
PAR_total	Total accumulative PAR	ppm	0.0	1126662.08	575982.25	1084579-05855.29
SlideCum-Harvest	Summed harvest of the past 7 days	kg/m2	0.0	4.26	2.23	1.33

4.2.2 Variables that are used as predictors

The variables that have been chosen are both available by the combined INTKAM - KASPRO deterministic models and the real-life data set obtained during the Autonomous Greenhouse Challenge in 2019 [3] [9] [22]. Next to this is expert knowledge used to decide what variables most influence the greenhouse cucumber production. The variables that have been chosen to forecast the last week's cucumber harvest are the following (see Table 4.1 and 4.2 for their descriptions):

1. DaysAfterPlanting
2. Tair
3. CO2
4. PAR
5. PlantLoad
6. Tair_total
7. PAR_total

The information containing the previous harvest values will not be used despite the findings by Gong, which was that the previous yield information was the most valuable predictor for his model to predict the tomato yield [19]. As already stated in chapter 3, section 3.4, it is due to the often unavailability of such previous harvest values in real-life situations.

4.2.3 Cross validation and creation of the training, validation and test samples

Performing normal cross validation (CV) for training and evaluating the model when working with time-series data is not allowed due to phenomena of future data leakage, as described in section 2.1 [31]. The variation, population informed nested cross validation, prevents future data leakage and is therefore used for evaluating and training the models of this research [7]. This variation comprises the following:

1. Assumption: The different scenarios are independent from each other.
2. The scenarios are split into train scenarios, which are fully used for training samples, and the combined, validate and test scenarios. The validate/test scenarios are used as follows: randomly a starting position is chosen, to take 1512 (9 weeks) of validation samples of. All the data after these validation samples, data from later in time, can be used as test data. The test data should come later in time than

the validation data to prevent future data leakage, see section 2.1 for an explanation. The starting time-point positions of the validation data are chosen randomly to retrieve validation samples that are representative of the whole time-series data and not for example just the beginning. This also means that the amount of test samples is not fixed before staring the optimization, training and evaluation of the models.

3. The cross validation is done by varying the input sample sizes. In the end the error will be calculated by averaging the obtained errors. The sample sizes used are 1 and 2 weeks of data, to predict the last week's harvest.

The split between training, validation and test should be around the ratio 50:25:25 [30]. However due to computational reasons the ratio between training, validation and test is chosen to be around 78:14:8. This ratio is variable since the amount of test data is not constant due to the random starting positions of the validation data as stated earlier. Using this ratio, the amount of validation data is lowered. In this way the computational cost of hyper-parameter optimization is reduced, as this is carried out using only the validation data.

4.2.4 Normalization

The data is normalized and scaled between 0 and 1 using the MinMaxScaler normalization formula. Because future data leakage needs to be prevented, the normalization factors are decided using training data only. The MinMaxScaler formula is:

$$X_{std} = \frac{(X - X.min(axis = 0))}{(X.max(axis = 0) - X.min(axis = 0))} \tag{4.1}$$

$$X_{scaled} = X_{std}(max - min) + min \tag{4.2}$$

X_{std} will be normalized to a range between 0 and 1. For every feature the maximum and minimum value is calculated, X.max(axis=0) and X.min(axis=0) respectively [37] [13].

4.3 The structure of the network and hyper-parameter optimization

The created DNN is structured as follows:

1. LSTM-RNN layer: Long Short-Term Memory - Recurrent Neural Network. For explanation on how a LSTM-RNN network works see section 2.2.

2. TCN part: Temporal Convolutional Network. For explanation on how a TCN network works see section 2.3.

3. Dense layer with a 1D dimensionality of the output space [6]. This means that each neuron in the dense layer receives input from all neurons of its previous layer, this makes it fully connected.

The output of the LSTM-RNN states ([..., h_{t-1}, h_t, h_{t+1}, ...]) are the input of the TCN layer and the Dense layer will in the end use the features extracted by the TCN to predict the last week's harvest.
For the hyper-parameter optimization the technique RandomSearch is used [4]. For both the folds, 1 and 2 week(s) of input, 15 combinations are evaluated and the best result, expressed in the lowest MSE, is used to train and test the model. The following hyper-parameters, with the possible value options, are optimized:

1. Model:
   (a) epochs: [10, 20, 40, 60]
2. LSTM-RNN:
   (a) number of LSTM units: [5, 15, 25, 30]
3. TCN:
   (a) Kernel size: [2, 4, 6, 8]
   (b) Use of layer normalization: [True, False]
   (c) Dropout rate: [0.0, 0.25, 0.5]
   (d) Number of filters: [40, 50, 64, 75, 80]

The hyper-parameters 'Use of layer normalization' (normalize the sub outputs between the different convolutional layers) and 'Dropout rate' (do not update a certain percentage of the weights) are there for generalization. The uses of the other hyper-parameter can be found in chapter 2. Each model uses the Adam optimizer and uses as score function the MSE [25].

4.4 Comparison with LSTM-RNN network

The performance of the combined LSTM-TCN DNN is compared with using only a LSTM DNN. The same actions are performed as describe above. However, optimization is done on only those hyper-parameters that are involved for the LSTM model, which are the epochs and the LSTM-units.

4.5 Results

In this section the most important results are described. For more elaborate information on these results refer to appendix A.

4.5.1 LSTM-TCN model with input size of 1 week - results

Using one week of samples as the input for the LSTM-TCN DNN, The hyper-parameter optimization retrieved a lowest MSE of $4.850 \cdot 10^{-4}$ using the hyper-parameter values:

1. use_layer_norm=True
2. LSTM units=25
3. nb_filters=75
4. kernel_size=2
5. dropout_rate=0.25
6. epochs=40

After training the network, using the optimized hyper-parameter values, the average training MSE is equal to $5.656 \cdot 10^{-3}$ and the training MSE after completion of the training is $3.906 \cdot 10^{-5}$. The average test MSE is equal to $1.957 \cdot 10^{-4}$ and the test MSE after completion training is $2.549 \cdot 10^{-5}$. See appendix A table A.1 for the MSE scores for training and testing per epoch. The figure 4.1 shows that the MSE stabilizes.

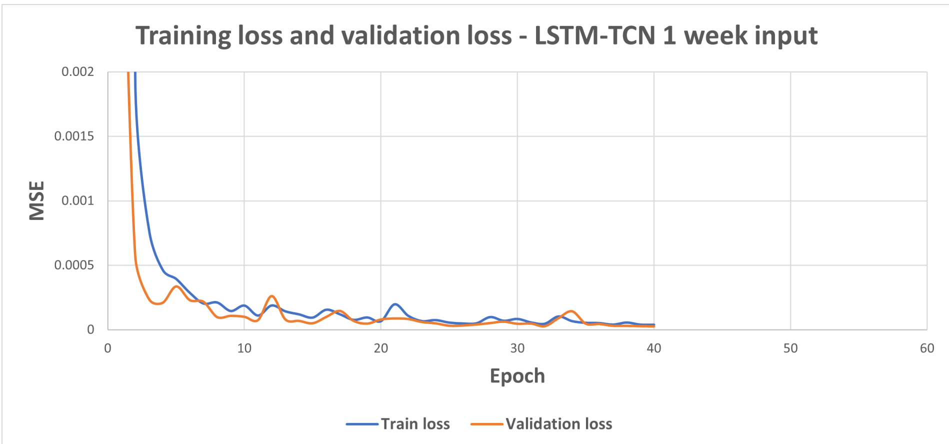

Figure 4.1: Training and Validation loss of the LSTM-TCN model input size 1 week trained using optimized hyper-parameters: use_layer_norm=True, LSTM units=25, nb_filters=75, kernel_size=2, dropout_rate=0.25 and epochs=40

4.5.2 LSTM-TCN model with input size of 2 weeks - results

Using 2 weeks of samples as input to the LSTM-TCN DNN, the hyper-parameter optimization retrieved a lowest MSE of $1.453 \cdot 10^{-3}$ using the hyper-parameter values:

1. use_layer_norm=True
2. LSTM units=5
3. nb_filters=40
4. kernel_size=8
5. dropout_rate=0.5
6. epochs=60

After training the network using these hyper-parameter values, the average training MSE is equal to $2.407 \cdot 10^{-3}$ and the training MSE after completion of the training is $5.050 \cdot 10^{-4}$. The average test MSE is equal to $7.966 \cdot 10^{-4}$ and the test MSE after full training is $1.990 \cdot 10^{-4}$. See appendix A, table A.3 for the MSE for training and testing per epoch. Figure 4.2 shows that, with one strong deviating MSE at epoch 38, the MSE is able to stabilise.

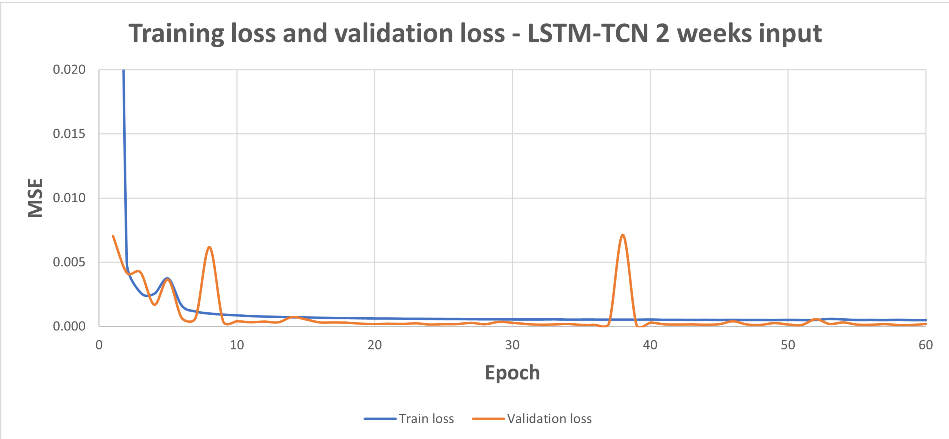

Figure 4.2: Training and Validation loss of the LSTM-TCN model input size 2 weeks trained using optimized hyper-parameters: use_layer_norm=True, LSTM units=5, nb_filters=40, kernel_size=8, dropout_rate=0.5 and epochs=60

4.5.3 LSTM model with input size of 1 week - results

Using 1 week of samples as input of the LSTM, the hyper-parameter optimization retrieved a lowest MSE of $3.090 \cdot 10^{-4}$ using the hyper-parameter values:

1. LSTM units=30
2. epochs=60

After training the network using these hyper-parameter values the average training MSE is equal to $4.895 \cdot 10^{-4}$, with the training MSE after completion of the training is $7.316 \cdot 10^{-5}$. The average test MSE is equal to $3.177 \cdot 10^{-4}$ and the test MSE after training is $7.039 \cdot 10^{-5}$. See appendix A table A.5 for the MSE for training and testing per epoch. Figure 4.3 shows that the LSTM model is not able to stabilize the MSE within the 60 epochs.

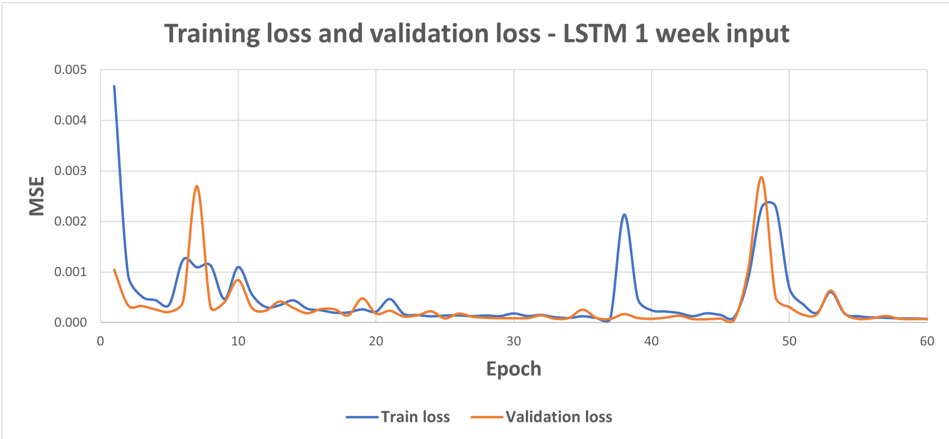

Figure 4.3: Training and Validation loss of the LSTM model input size 1 week trained using optimized hyper-parameters: LSTM units=30 and epochs=60

4.5.4 LSTM model with input size of 2 weeks - Search LSTM 2 results

Using 2 week of samples as input of the LSTM, the hyper-parameter optimization retrieved a lowest MSE of $3.177 \cdot 10^{-3}$ using the hyper-parameter values:

1. LSTM units=30
2. epochs=10

After training the network using these hyper-parameter values the average training MSE is equal to $1.159 \cdot 10^{-3}$ with the training MSE after completion training being $4.093 \cdot 10^{-4}$. The average test MSE is equal to $9.446 \cdot 10^{-4}$ and the test MSE after training is $3.220 \cdot 10^{-4}$. See appendix A table A.7 for the MSE for training and testing per epoch. Figure 4.4 shows that the LSTM model is not able to stabilize the MSE within the 10 epochs.

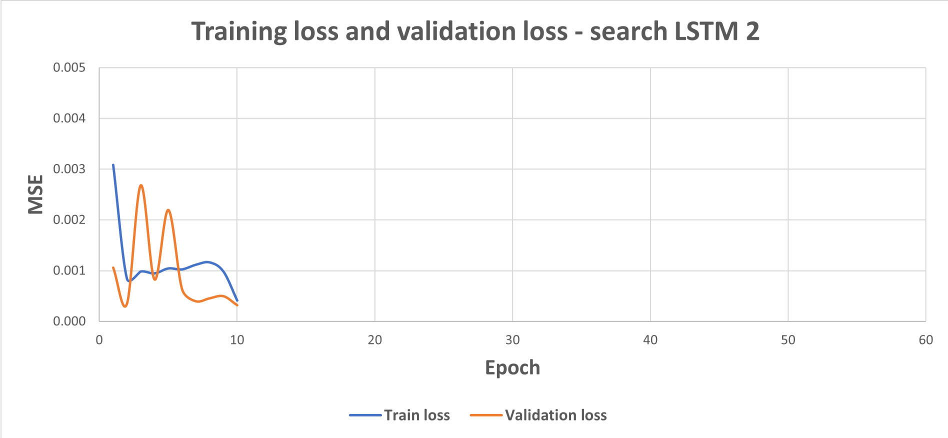

Figure 4.4: Training and Validation loss of the LSTM model input size 2 weeks trained using optimized hyper-parameters: LSTM units=30 and epochs=10

4.5.5 Comparison results LSTM-TCN and LSTM model

A T-test on the average training MSEs (mean MSE over all the epochs) showed a significant difference in performance from each other with a p-value of 0.045. The p-value of 0.726 on the average test MSEs shows no significant difference in performance. The average MSEs for the LSTM are overall lower compared to the LSTM-TCN DNN (see table 4.3).
The MSEs of the training and testing after completing the training (MSE retrieved after the last epoch) are overall lower for the LSTM-TCN DNN model. However, the results are not significant with p-values 0.915 and 0.633 for the training and testing respectively (see table 4.4).

Model	**Input Size**	**Train MSE**	**Test MSE**
LSTM-TCN	1 week	2.823e-3	7.203e-4
LSTM-TCN	2 weeks	2.407e-3	7.966e-4
LSTM	1 week	4.895e-4	3.177e-4
LSTM	2 weeks	1.159e-3	9.446e-4

Table 4.3: The average MSE scores, for the models LSTM-TCN and LSTM with respectively 1 and 2 weeks of input samples.

Model	**Input Size**	**Train MSE**	**Test MSE**
LSTM-TCN	1 week	4.659e-5	2.167e-5
LSTM-TCN	2 weeks	5.040e-4	1.990e-4
LSTM	1 week	7.316e-5	7.039e-5
LSTM	2 weeks	4.093e-4	3.220e-4

Table 4.4: The MSE scores after training the models with their optimized hyper-parameter values, for the models LSTM-TCN and LSTM with respectively 1 and 2 weeks of input samples.

Chapter 5

Conclusions

The average MSEs (MSEs over all epochs averaged) show a better performance in forecasting the last week's greenhouse cucumber harvest for the LSTM model compared to the combined LSTM-TCN model, with a significant difference for the average training MSE. The fact that the LSTM model performs better on average over all epochs, is because the MSEs are a lot higher in the beginning for the combined LSTM-TCN model, around 0.1/0.2 to 0.003/0.005 for the LSTM-TCN and LSTM model respectively (see tables A.1, A.3, A.5 and A.7). Because of this, the average of the LSTM-TCN model ends up at a higher error.
Comparing the evaluation scores (the test MSEs) of the fully trained models, better results are obtained by the combined LSTM-TCN model compared to the standalone LSTM model. The combined model achieved a decrease of 69% and 38% in the test MSE for one week and two weeks of input data respectively, compared to the LSTM model, however, these results are not significant different.
Looking at the training of the models, the LSTM-TCN model is able to stabilize its training and testing MSE over the epochs for both one and two weeks of input data, while the LSTM model is not able to do this. This could indicate that the LSTM model needs different values for the hyperparameters or that the LSTM model is not able to capture the complex relationships that are present in the synthetic greenhouse cucumber data to predict the last week's cucumber harvest.
For the LSTM-TCN model with 2 weeks of input data, it can be observed that after stabilization of the test MSE, a relative bigger test MSE occurs at epoch 38. This could be caused by having the cucumber production being forecasted less accurate for all the testing samples or by only a few of the testing samples.
This all shows that even though the results are not significant, the first and second hypothesis can be accepted and that there is potential for the proposed LSTM-TCN DNN to capture the challenge of greenhouse cucumber

production forecasting.

5.1 future work

For future work the following is suggested:

1. Use all the created synthetic data from the deterministic model KASPRO and INTKAM [9] [3]. At the moment due to time constraints, only 100 out of the 320 available scenarios were used to optimize and train the DNNs, using all the available scenarios will provide more robust results.

2. This research only used the input sizes one and two weeks of data to predict the last week's greenhouse cucumber harvest. Also evaluating the input sizes three, four and five weeks of data will provide useful information. The average cucumber growth cycle is around 17 days, therefore it is expected that, when using more weeks as input data, the DNN will only use the last weeks to predict the last week's harvest [22]. This, since the first weeks are containing information about a different cucumber growth cycle and therefore a different harvest. In this way insight is given on whether the DNN really learned the problem or is randomly using the values to come to a prediction value.

3. Given the related work and results of the optimization and learning of the LSTM-TCN DNN and the standalone LSTM DNN, it is advised to extend the options at the hyper-parameter optimization with higher numbers for the number of epochs, LSTM units and TCN filters.

4. Make the connection between this synthetic trained model and using the model on real-life data. Check how well the model still performs on real-life data. One method for addressing the potential reality gap is by performing domain randomization on the synthetic data [15]. Parameters of the deterministic models are randomized in non-realistic ways to force the DNN to learn the features that are essential in forecasting greenhouse production. Another method is to train the DNN with a mixture of real-life and synthetic data [18].

5. It might take some time before enough real-life data is available, to be able to train models for forecasting greenhouse crop production on real-life data directly. Therefore, research on how to make the greenhouse crop data originating from deterministic models, better reflecting the data coming from real-life greenhouse production situation, could be beneficial. This to facilitate the transition of the synthetic created data-driven models towards performing well on the real-life greenhouse data.

6. To be able to make an application that can be used in real-life greenhouse cucumber production forecasting, more real-life data is needed to evaluate the created data-driven models. Therefore future research should focus on obtaining more real-life greenhouse cucumber data.

5.2 limitations

For this studies, as mentioned in section 4.1, synthetic data is used. For results reflecting the performance of the model in the real world, data from actual greenhouse cucumber production should be used. Next to this, due to the limited amount of time and the computational capacity available for this research, not all synthetic data has been used for creating the models. The decision was made to only use 100 randomly chosen scenarios, out of the 320 scenarios available. It is therefore possible that the data used for the hyper-parameter optimization and training is not reflecting the whole spectrum of this greenhouse cucumber production forecasting challenge.
The additional variables Tair_total and PAR_total do not contain the total summed temperature/PAR up to that point from the beginning of a cucumber growth cycle but the beginning of the very first planting of the plants. Therefore, once cucumbers are harvested from the plants and a new cucumber growth cycle starts, these variables do not reset to zero but continue to sum the temperature and PAR values. This might reduce the effectiveness of these additional variables for providing information on the conditions during the cucumber growth cycle.
Lastly, due to time constraints, rerunning the combined LSTM-TCN DNN with 2 weeks of input data to investigate the error fluctuation for the test MSE at epoch 38 could not be performed. Therefore it is not possible to indicate whether this fluctuation is accidental and what could have caused it.

Bibliography

[1] Alhnaity, B., et al., *Using deep learning to predict plant growth and yield in greenhouse environments.*, 2019, International Symposium on Advanced Technologies and Management for Innovative Greenhouses: GreenSys2019 1296.

[2] Virtanen, P., et al. and SciPy 1.0 Contributors, *SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python*, Nature Methods **17** (2020), 261–272.

[3] E. Arcelis L.F.M., Elings, A., de Visser, P.H.B. Heuvelink, *Simulating growth and development of tomato crop.*, 2008, Acta Hortic. 2009, 821, 101–110.

[4] Bergstra, J. Bengio, Y.., *Random search for hyper-parameter optimization.*, 2012, Journal of machine learning research 13.2 (2012).

[5] L. Breiman, *Random forests.*, 2001, Machine Learning, vol. 45, no. 1, pp. 5-32.

[6] Francois Chollet et al., *Keras*, 2015.

[7] C Cochrane, *Medium: time series nested cross-validation*, 2018, https://towardsdatascience.com/time-series-nested-cross-validation-76adba623eb9.

[8] V. Cortes, C. Vapnik, *Support-vector networks.*, 1995, Machine Learning, vol. 20, no. 3, pp. 273-297.

[9] H.F. De Zwart, *Analyzing energy-saving potentials in greenhouse cultivation using a simulation model*, 1996, Ph.D. Thesis, Wageningen University, Wageningen, The Netherlands.

[10] Abadi, M., et al., *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015, Software available from tensorflow.org.

[11] Huang, Norden E., et al., *The empirical mode decomposition and the hilbert spectrum for nonlinear and non-stationary time series analysis.*,

1998, Proceedings of the Royal Society of London. Series A: mathematical, physical and engineering sciences 454.1971 (1998): 903-995.

[12] Pedregosa, F., et al., *Scikit-learn: Machine learning in python*, Journal of machine learning research **12** (2011), no. Oct, 2825–2830.

[13] Pedregosa, F., et al., *Scikit-learn: Machine learning in Python*, Journal of Machine Learning Research **12** (2011), 2825–2830.

[14] Pezoa, F., et al., *Foundations of json schema*, Proceedings of the 25th International Conference on World Wide Web, International World Wide Web Conferences Steering Committee, 2016, pp. 263–273.

[15] Tremblay, J., et al., *Training deep networks with synthetic data: Bridging the reality gap by domain randomization.*, 2018, Proceedings of the IEEE conference on computer vision and pattern recognition workshops.

[16] Wan, R., et al., *Multivariate temporal convolutional network: A deep neural networks approach for multivariate time series forecasting.*, 2019, Electronics 8.8 (2019): 876.

[17] FAO., *The state of food security and nutrition in the world—building climate resilience for food security and nutrition*, 2018, Food and Agriculture Organization of the United Nations (FAO): Rome, Italy, 2018; ISBN 978-92-5-130571-3.

[18] Rameshbabu K. Imran M. Roth H. Gamal, O., *Bridging the reality gap: Investigation of deep convolution neural networks ability to learn from a combination of real and synthetic data.*, 2020, In ICINCO (pp. 446-454).

[19] Gong, L., et al., *Deep learning based prediction on greenhouse crop yield combined tcn and rnn.*, 2021, Sensors 21.13 (2021): 4537.

[20] Millman K.J. van der Walt S.J. et al. Harris, C.R., *Array programming with NumPy*, Nature **585** (2020), no. 7825, 357–362.

[21] J. He, Y.. Zhao, *Temporal convolutional networks for anomalydetection in time series.*, 2019, J. Phys.: Conf. Ser.1213 042050.

[22] Hemming, S., et al., *Remote control of greenhouse vegetable production with artificial intelligence—greenhouse climate, irrigation, and crop production.*, 2019, Sensors 19.8 (2019): 1807.

[23] J. Hochreiter, S. Schmidhuber, *Long short-term memory.*, 1997, Neural computation, vol. 9, no. 8, pp. 1735-1780.

[24] J. D. Hunter, *Matplotlib: A 2d graphics environment*, Computing in Science & Engineering **9** (2007), no. 3, 90–95.

[25] Ba J Kingma D P, *Adam: A method for stochastic optimization[j].*, 2014, Computer Science, 2014.

[26] Kluyver, T.,et al., *Jupyter notebooks – a publishing format for reproducible computational workflows*, Positioning and Power in Academic Publishing: Players, Agents and Agendas (F. Loizides and B. Schmidt, eds.), IOS Press, 2016, pp. 87 – 90.

[27] Chang W. C. Yang Y. Liu H. Lai, G., *Modeling long-and short-term temporal patterns with deep neural networks.*, 2018, In The 41st International ACM SIGIR Conference on Research Development in Information Retrieval (pp. 95-104).

[28] Yuan Q. Liu, T. and Y. Wang, *Hierarchical optimization control based on crop growth model for greenhouse light environment.*, 2021, Computers and Electronics in Agriculture 180 (2021): 105854.

[29] Wes McKinney et al., *Data structures for statistical computing in python*, Proceedings of the 9th Python in Science Conference, vol. 445, Austin, TX, 2010, pp. 51–56.

[30] G. Myrianthous, *Medium: training vs testing vs validation sets*, 2021, https://towardsdatascience.com/training-vs-testing-vs-validation-sets-a44bed52a0e1.

[31] Tang L. Refaeilzadeh, P. and H.. Liu, *Cross-validation.*, 2009, Encyclopedia of database systems 5 (2009): 532-538.

[32] C. Stanghellini, *Horticultural production in greenhouses: Efficient use of water.*, 2014, Acta Hortic. 2014, 1034, 25–32.

[33] G. Van Rossum, *The python library reference, release 3.8.2*, 2020.

[34] Wang, J.,et al., *Multilevel wavelet decomposition network for interpretable time series analysis.*, 2018, roceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery Data Mining. 2018.

[35] Z. Wu and Norden E. Huang., *Ensemble empirical mode decomposition: a noise-assisted data analysis method.*, 2009, Advances in adaptive data analysis 1.01 (2009): 1-41.

[36] Xingjian, S. H. I.,et al., *Convolutional lstm network: A machine learning approach for precipitation nowcasting.*, 2015, Advances in neural information processing systems. 2015.

[37] Zhang, T.,et al., *Research on gas concentration prediction models based on lstm multidimensional time series.*, 2018, Energies 12.1 (2019): 161.

[38] Dragomiretskiy, K., Zosso, D., *Variational mode decomposition.*, 2013, IEEE transactions on signal processing 62.3 (2013): 531-544.

Appendix A

Appendix

A.1 LSTM-TCN Model results input size 1 week

A.1.1 Results per epoch for the model training and testing

Epoch	Train MSE	Test MSE	Epoch	Train MSE	Test MSE
1	0.21896282	0.00369524	21	0.00019764	0.00008719
2	0.00199291	0.00057480	22	0.00010832	0.00008304
3	0.00078444	0.00023928	23	0.00006742	0.00005925
4	0.00046736	0.00020779	24	0.00007511	0.00004928
5	0.00039454	0.00033560	25	0.00005573	0.00003079
6	0.00028669	0.00022920	26	0.00004893	0.00003245
7	0.00020347	0.00021666	27	0.00005050	0.00004051
8	0.00021195	0.00009811	28	0.00009845	0.00005133
9	0.00014550	0.00010752	29	0.00006999	0.00006229
10	0.00018770	0.00009989	30	0.00008404	0.00004556
11	0.00011044	0.00007459	31	0.00005695	0.00004749
12	0.00018858	0.00026095	32	0.00004686	0.00002710
13	0.00014313	0.00007944	33	0.00010350	0.00008884
14	0.00011969	0.00006839	34	0.00006701	0.00014295
15	0.00009454	0.00004998	35	0.00005421	0.00004633
16	0.00015570	0.00009932	36	0.00005183	0.00004414
17	0.00012009	0.00014543	37	0.00004025	0.00003017
18	0.00007694	0.00006788	38	0.00005612	0.00002943
19	0.00009564	0.00004722	39	0.00003998	0.00002719
20	0.00006669	0.00007923	40	0.00003906	0.00002549

Table A.1: Input size 1 week LSTM-TCN DNN Training and Test MSE

A.1.2 Optimization results

There are many combinations possible (1920 combinations). During the optimization only 15 combinations have been considered, therefore it is not possible to investigate the influence of each hyper-parameter individually. During the optimization the performance was expressed by the MSE. Looking at the values in table A.2 the following observations can be done: (Take into account that all these statements are not checked for correctness, this can only be done when all possible combinations have been tried, this is however due to computational reasons and time constraints not done in this BSc thesis research.)

1. A too high dropout rate (0.5) lowers the performance.

2. The best three results all used 40 epochs. Too little epochs reduces the performances.

3. Looking at the lines with MSE scores $6.330{\cdot}10^{-4}$ and $2.904{\cdot}10^{-3}$, it can be seen that all hyper-parameters are the same only the number of epochs and dropout rate differs between these two. It shows that when the number of epochs is too little and the dropout rate is too high the performance drops.

4. When 50 TCN filters are used, the performance is most often lower compared to using the other amounts of TCN filters.

5. A higher amount of LSTM units provides most of the times better performances.

MSE	**LSTM Units**	**Layer Norm**	**TCN filters**	**Kernel**	**Epoch**	**Dropout**
0.000485	25	TRUE	75	2	40	0.25
0.000492	30	TRUE	40	8	40	0.00
0.000509	30	FALSE	40	6	40	0.25
0.000540	30	FALSE	40	2	60	0.00
0.000570	25	FALSE	64	6	20	0.25
0.000605	5	FALSE	64	8	20	0.00
0.000633	30	TRUE	75	4	60	0.00
0.000725	15	TRUE	80	4	20	0.25
0.000812	5	TRUE	80	8	60	0.50
0.001150	15	FALSE	50	8	10	0.00
0.001392	15	TRUE	50	2	40	0.50
0.001427	15	TRUE	50	2	20	0.50
0.001986	5	FALSE	50	2	20	0.00
0.002904	30	TRUE	75	4	10	0.50
0.002968	15	FALSE	50	2	20	0.00

Table A.2: This table shows the MSE scores from best to worst for the 15 hyper-parameter value combinations evaluated during the random optimization for the LSTM-TCN DNN using an input size of 1 week.

A.2 LSTM-TCN Model results input size 2 weeks

A.2.1 Results per epoch for the model training and testing

Epoch	Train MSE	Test MSE	Epoch	Train MSE	Test MSE
1	0.09581646	0.00705840	31	0.00055591	0.00018927
2	0.00502188	0.00419593	32	0.00055619	0.00013672
3	0.00265191	0.00421872	33	0.00056874	0.00015605
4	0.00256173	0.00169853	34	0.00054335	0.00019555
5	0.00374896	0.00365256	35	0.00054585	0.00012128
6	0.00162693	0.00065014	36	0.00054618	0.00012675
7	0.00117390	0.00068372	37	0.00054050	0.00024764
8	0.00101954	0.00619129	38	0.00054580	0.00714613
9	0.00093432	0.00039019	39	0.00053802	0.00010795
10	0.00087895	0.00041075	40	0.00055307	0.00030107
11	0.00081912	0.00033026	41	0.00052763	0.00016896
12	0.00078122	0.00038106	42	0.00053023	0.00014741
13	0.00075831	0.00032581	43	0.00052125	0.00016176
14	0.00072388	0.00072799	44	0.00052767	0.00013460
15	0.00071615	0.00055259	45	0.00051677	0.00017735
16	0.00068974	0.00031951	46	0.00052461	0.00040583
17	0.00066987	0.00031491	47	0.00051498	0.00015775
18	0.00066537	0.00029445	48	0.00051806	0.00012125
19	0.00064909	0.00021834	49	0.00050851	0.00026558
20	0.00063327	0.00019036	50	0.00052552	0.00015336
21	0.00063102	0.00021147	51	0.00050729	0.00012535
22	0.00061401	0.00019449	52	0.00051541	0.00057035
23	0.00061254	0.00024348	53	0.00059849	0.00019619
24	0.00060232	0.00015053	54	0.00055959	0.00031409
25	0.00059462	0.00017837	55	0.00051486	0.00014628
26	0.00059140	0.00018862	56	0.00052071	0.00013118
27	0.00057591	0.00028137	57	0.00050597	0.00018249
28	0.00057148	0.00017488	58	0.00053048	0.00011316
29	0.00056857	0.00036102	59	0.00050241	0.00011654
30	0.00055954	0.00028851	60	0.00050396	0.00019902

Table A.3: Input size 2 weeks LSTM-TCN DNN Training and Test MSE

A.2.2 Optimization results

There are many combinations possible (1920 combinations). During the optimization only 15 combinations have been considered, therefore it is not possible to investigate the influence of each hyper-parameter individually. During the optimization the performance was expressed by the MSE. Looking at the values in table A.4 the following observations can be done: (Take into account that all these statements are not checked for correctness, this can only be done when all possible combinations have been tried, this is however due to computational reasons and time constraints not done in this BSc thesis research.)

1. The number of epochs needs to be high enough otherwise the performance lowers.
2. A higher amount of LSTM units increases the chance of a worse performance.
3. Overall, a bigger kernel size provides higher performances.

Score	**LSTM Units**	**Layer norm**	**TCN filters**	**Kernel**	**Epoch**	**Dropout**
0.001453	5	TRUE	40	8	60	0.50
0.001773	5	TRUE	64	2	60	0.50
0.002006	5	FALSE	80	6	40	0.25
0.002021	25	TRUE	64	4	40	0.50
0.002244	25	FALSE	80	6	20	0.00
0.002541	30	TRUE	75	8	20	0.00
0.003293	5	TRUE	75	4	40	0.25
0.003517	25	TRUE	75	6	10	0.25
0.004458	30	TRUE	64	4	20	0.50
0.006443	25	TRUE	64	2	20	0.00
0.006575	30	TRUE	75	2	20	0.00
0.009831	25	TRUE	50	6	10	0.50
0.015686	30	FALSE	64	2	10	0.50
0.529112	25	FALSE	50	2	20	0.00
4.197965	30	FALSE	64	2	20	0.25

Table A.4: This table shows the MSE scores from best to worst for the 15 hyper-parameter value combinations evaluated during the random optimization for the LSTM-TCN DNN using an input size of 2 weeks

A.3 LSTM Model results input size 1 week

A.3.1 Results per epoch for the model training and testing

Epoch	Train MSE	Test MSE
1	0.00467868	0.00104511
2	0.00095156	0.00034651
3	0.00051671	0.00032915
4	0.00044285	0.00026044
5	0.00035669	0.00021378
6	0.00124230	0.00042287
7	0.00109381	0.00269936
8	0.00112814	0.00030088
9	0.00046467	0.00040480
10	0.00109922	0.00084369
11	0.00055445	0.00029047
12	0.00030679	0.00023825
13	0.00034282	0.00041861
14	0.00043966	0.00029202
15	0.00027520	0.00018678
16	0.00024340	0.00026827
17	0.00019440	0.00026434
18	0.00020109	0.00014487
19	0.00026142	0.00047941
20	0.00021420	0.00017784
21	0.00046533	0.00023504
22	0.00017218	0.00012350
23	0.00014899	0.00014334
24	0.00012276	0.00022343
25	0.00013855	0.00008400
26	0.00013931	0.00017887
27	0.00012688	0.00011565
28	0.00014057	0.00009580
29	0.00012806	0.00008761
30	0.00017995	0.00008764

Epoch	Train MSE	Test MSE
31	0.00012868	0.00008951
32	0.00015103	0.00014270
33	0.00010536	0.00007716
34	0.00009057	0.00009223
35	0.00012488	0.00025895
36	0.00009051	0.00009239
37	0.00007821	0.00007291
38	0.00213391	0.00017008
39	0.00046973	0.00009222
40	0.00024318	0.00007659
41	0.00021875	9.9406E-05
42	0.00018815	0.00013613
43	0.00012533	7.0610E-05
44	0.00018393	6.7055E-05
45	0.00014970	8.0453E-05
46	0.00010740	6.0474E-05
47	0.00086058	0.00104024
48	0.00228408	0.00287432
49	0.00229383	0.00050547
50	0.00068257	0.00030888
51	0.00036000	0.00015578
52	0.00019371	0.00016559
53	0.00060457	0.00063327
54	0.00017683	0.00018428
55	0.00012495	7.5070E-05
56	0.00010067	8.6808E-05
57	9.3475E-05	0.00013274
58	8.1461E-05	7.5769E-05
59	8.0727E-05	7.0271E-05
60	7.3158E-05	7.0394E-05

Table A.5: Input size 1 week LSTM DNN Training and Test MSE

A.3.2 optimization results

Only 2 parameters (instead of 6 for the LSTM-TCN) were optimized, therefore almost all possible combinations have been evaluated. See table A.6 for the optimization scores per combination of amount of LSTM units and epochs. Next to this contains figure A.1 the scores per amount of LSTM units. Figure A.2 shows the scores per amount of epochs. The following observations can be done:

1. In general gives a lower amount of epochs a higher MSE.
2. In general gives a lower amount of LSTM units a higher MSE.
3. For most of the LSTM unit options, 60 epochs gives the lowest MSE.
4. For most of the epoch options applies, the higher the amount of LSTM units the lower the MSE.

Score	**LSTM Units**	**Epochs**
0.000309	30	60
0.000334	30	40
0.000385	15	60
0.000397	25	60
0.000443	30	20
0.000538	25	20
0.000548	15	40
0.000832	25	40
0.000971	30	10
0.001033	5	40
0.001118	5	60
0.001140	25	10
0.001586	15	10
0.001587	5	20
0.002351	5	10

Table A.6: This table shows the MSE scores from best to worst for the 15 hyper-parameter value combinations evaluated during the random optimization for the LSTM DNN using an input size of 1 week

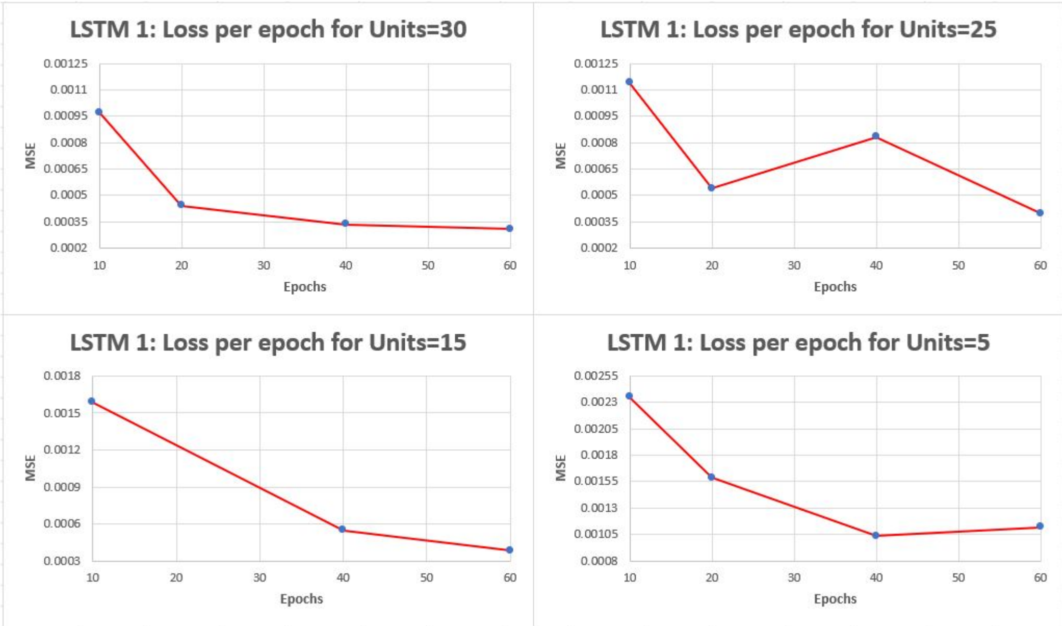

Figure A.1: MSE scores per amount of epochs for each amount of LSTM units separately, input size 1 week.

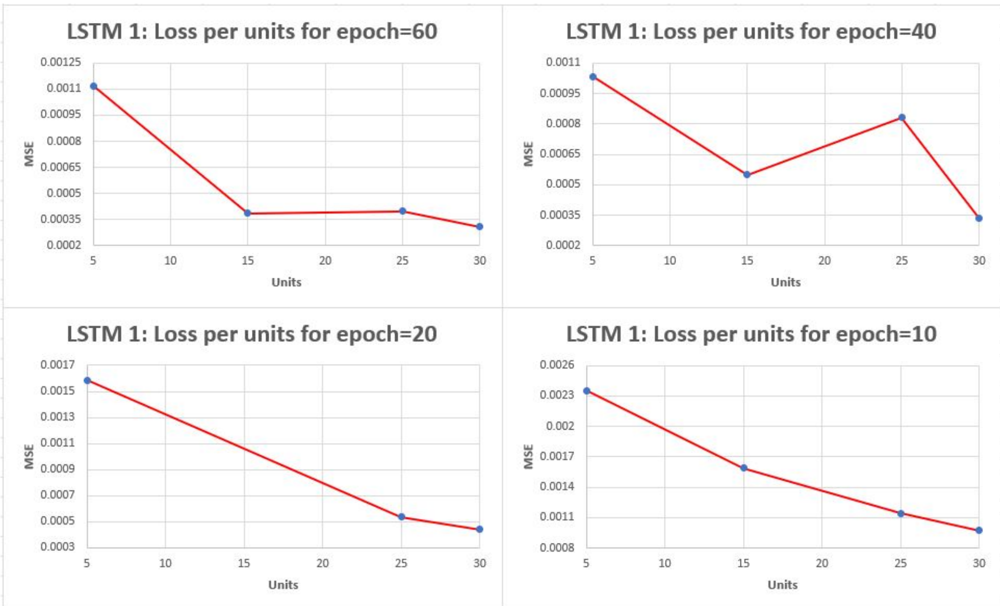

Figure A.2: MSE scores per amount of LSTM units for each amount of epochs separately, input size 1 week.

A.4 LSTM Model results input size 2 weeks

A.4.1 Results per epoch for the model training and testing

Epoch	Train MSE	Test MSE
1	0.00308641	0.00106294
2	0.00083728	0.00035414
3	0.00098293	0.00268440
4	0.00094502	0.00082947
5	0.00104264	0.00219900
6	0.00102459	0.00063495
7	0.00111592	0.00040008
8	0.00116248	0.00045873
9	0.00098127	0.00050049
10	0.00040930	0.00032195

Table A.7: Input size 2 weeks LSTM DNN Training and Test MSE

A.4.2 optimization results

Only 2 parameters (instead of 6 for the LSTM-TCN) were optimized, therefore almost all possible combinations have been evaluated. See table A.8 for the optimization scores per combination of amount of LSTM units and epochs. Next to this contains figure A.3 the scores per amount of LSTM units. Figure A.4 shows the scores per amount of epochs. The following observations can be done:

1. In general, the MSE is lower when more LSTM units are used.
2. For the higher amounts of LSTM units (30 and 25) the lowest MSE is obtained when using the lowest amount of epochs.
3. For most of the epoch options, the highest amount of LSTM units give the lowest MSE.

Score	**LSTM Units**	**Epochs**
0.003177	30	10
0.003293	15	40
0.003470	30	60
0.003479	30	20
0.003776	15	60
0.003866	5	60
0.004148	25	10
0.004179	25	40
0.004230	5	20
0.004234	30	40
0.004320	15	10
0.004411	25	60
0.004461	5	40
0.005618	5	10
0.006802	15	20

Table A.8: This table shows the MSE scores from best to worst for the 15 hyper-parameter value combinations evaluated during the random optimization for the LSTM DNN using an input size of 2 weeks

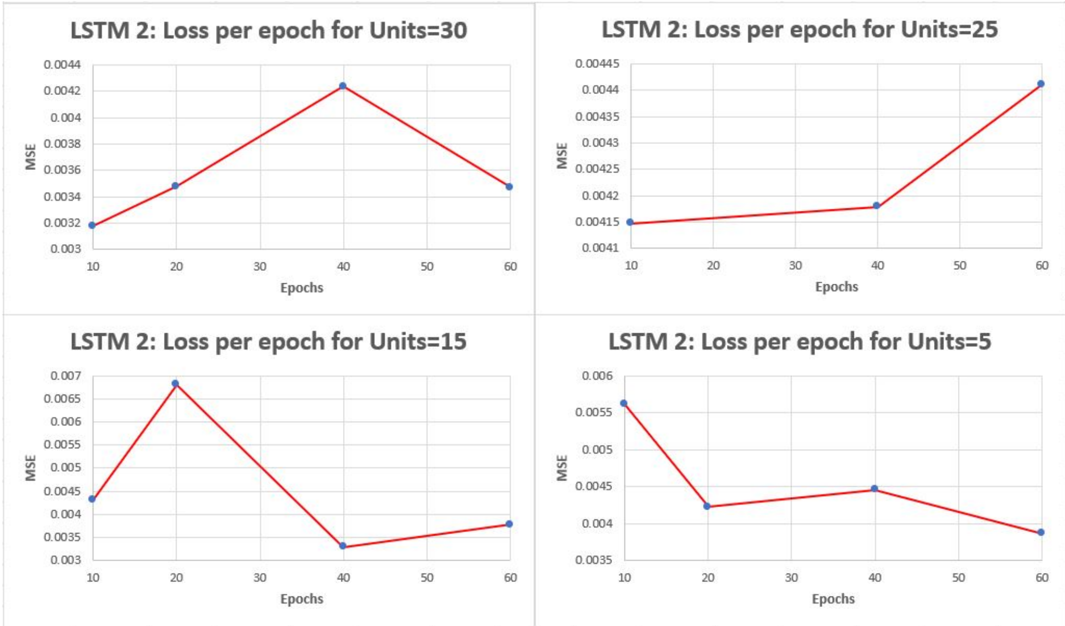

Figure A.3: MSE scores per amount of epochs for each amount of LSTM units separately, input size 2 weeks

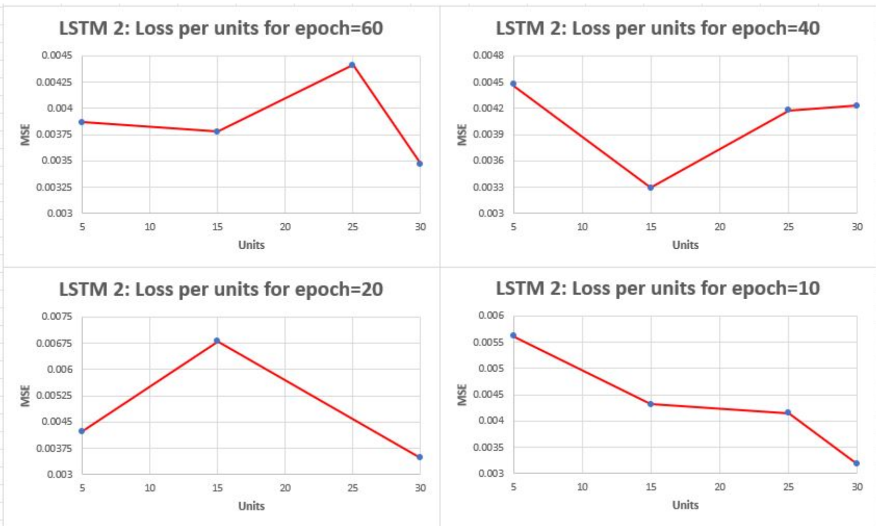

Figure A.4: MSE scores per amount of LSTM units for each amount of epochs separately, input size 2 weeks.

A.5 Code

In this BSc Artificial Intelligence thesis research Juptior Notebook (Anaconda 3) is used [26]. The following libraries facilitated this research:

1. Matplotlib [24]
2. Tensorflow [10]
3. Pandas [29]
4. Numpy [20]
5. Sklearn [12]
6. Keras [6]
7. Math [33]
8. JSON [14]
9. Random [33]
10. SciPy [2]

The notebooks used for this BSc thesis are available upon request.