Crop Yield Predictions using LSTM and TCN

Abdullah Asad (2020-EE-61)

Areeba Kushef (2020-EE-58)

M. Zafeer Zafar (2020-EE-174)

Problem

Greenhouse cultivation has become increasingly popular among crop growers due to its ability to extend the growing season, protect crops from adverse weather conditions, and provide a controlled environment for optimal growth. One of the critical requirements for optimal greenhouse management is the accurate prediction of crop yields based on environmental parameters. Accurate crop yield forecasting is pivotal for effective farming planning and management, enabling cultivators and farmers to make informed decisions regarding cultivation practices and financial planning.

Predicting crop yields in greenhouses is a complex task due to the multitude of factors that influence crop growth, including radiation levels, carbon dioxide concentrations, temperature, seed quality, soil quality and fertilization, and disease occurrences. Constructing an explicit model to capture the intricate relationships between these factors and crop yield is not straightforward. This complexity necessitates the development of advanced predictive models that can effectively process temporal sequences of environmental parameters and historical yield information to provide accurate yield predictions

Source of Data: National Institute of Biotechnology and Genetic Engineering (NIBGE) Faisalabad, Pakistan

Methodology

In this work, we propose a novel approach for greenhouse crop yield prediction by leveraging the capabilities of two state-of-the-art networks for temporal sequence processing: Recurrent Neural Network (RNN) and Temporal Convolutional Network (TCN). The proposed deep learning-based model is designed to predict future crop yields in a greenhouse by analyzing a sequence of historical greenhouse input parameters, such as temperature, humidity, carbon dioxide concentration, and radiation, along with the corresponding yield information.

The Temporal Convolutional Network (TCN) is utilized to capture the temporal dependencies in the input sequences, while the Recurrent Neural Network (RNN) is employed to process the sequential data and learn the long-term dependencies. The combined RNN+TCN-based deep learning approach is trained and evaluated on multiple datasets collected from various greenhouses across different time periods.

Comprehensive evaluations of the proposed algorithm are conducted based on a statistical analysis of the Root Mean Square Errors (RMSEs) between the predicted and actual crop yields. The experimental results demonstrate that the RNN+TCN-based deep learning approach outperforms traditional machine learning methods and other classical deep neural networks in terms of prediction accuracy, achieving smaller RMSEs.

Data preprocessing

The original dataset consist of 26 files of tabular data (which was unorganized but we appended certain values to make it consistent and structured so there are no dimension mismatch errors). 13 of the files 'tomato_sensor_data#.xlsx' (# means 1 to 13) represented sensor data of the tomato crop over a span of 120 days with samples collected every 5 minutes. The fields include: timestamp, temperature(degree C), humidity(%), CO2 concentration (ppm), Soil Moisture(%), Nitrogen, Phosphorus, Potassium (all in mg/kg), pH and light intensity(lux). Now the agriculturists measured the yields of tomato plants every after every 2 days by the following means: root depth, stem length and thickness, leaf area index and amount of fruits all depending on the stage of the crop in the cycle. We were just given approximate values of yield in Kg in 13 other files named 'tomato_yield_data#.xlsx' which had 2 fields: age of crop in days, and yield in kg. The yield is negligible in the initial values but then increases drastically to reach a final static value. Since the data is of 120 days and yield is found out after every 2 days, the yield file contains of 60 rows/records only although the sensor data file has thousands of rows. In order to make a consistent tabular dataset, we had to calculate and make the files 'stats#.xlsx'. The value of all 9 input parameters (other than timestamp) were fetched on 2 day intervals (makes 576 rows). The following statistics were calculated for each parameter for those 576 values: min, max, average and standard deviation. Eventually, we were left with 60 rows and 36 columns per file (9 features * 4stats = 36 final input features). To these 36 columns, the crop age and yield columns were also added. The crop age represented time series and yield was the only output variable.

Model Training and details

The model architecture comprises a combination of Long Short-Term Memory (LSTM) and Temporal Convolutional Networks (TCN), designed to predict yield trends. The TCN block is implemented using a custom ResidualBlock class, which consists of two 1D convolutional layers (conv1 and conv2) with a ReLU activation function. The ResidualBlock also includes an optional downsampling layer (downsample) to adjust the dimensions of the residual connection when the output channels differ from 1. The TCN network itself is constructed as a sequence of these ResidualBlock instances, with varying dilation rates to capture different temporal dependencies.

The overall TCN model, represented by the TCN class, employs a stack of these ResidualBlock layers, with the number of blocks determined by the num_blocks parameter. Following the TCN layers, a 1D convolutional layer (conv_output) is used to produce the final output of the TCN model.

To integrate LSTM with TCN, the LSTMTCN class combines an LSTM layer and the aforementioned TCN model. The LSTM layer, with a hidden size of 64, processes the input sequences and produces a sequence of hidden states. The output from the LSTM is reshaped and transposed to match the input format required by the TCN model. Subsequently, the TCN model is applied to the LSTM output to capture temporal features across the sequences effectively. The final layer is a fully connected dense layer with a ReLU activation, designed to produce the predicted yield values.

Testing and Prediction Results

The testing methodology involves evaluating the model's performance using the Root Mean Square Error (RMSE) metric. The last 3 files (11-13) were used for testing. The test_model function takes the model, test features, test targets, scaler, and the number of samples to be tested as inputs. It first normalizes the test features using the provided scaler and then predicts the yield values. The predicted values are then inverse transformed to obtain the actual yield values. The RMSE is calculated between the actual and predicted values, and the result is divided by 1000 for normalization. The testing is conducted for varying numbers of samples ranging from 1 to 59, and the RMSE is printed for each set of samples. This is done because we wanted to analyse the performance of model prediction based on the yield values provided and the ones predicted. 1 here means what in a sequence of 60 yield values, the first one is provided with input features and the rest 59 are predicted and compared with actual ones. Input features of all are provided. Then we increase the number of initial sequence values provided and this decreases the amount of values predicted and we evaluate the performance of the model depending on how much information it has regarding the current crop cycle.

Results:

- We used 50 training Epochs. The RMSE Loss started from 20.86 by epoch 1 and was down to 0.1349 by epoch 50
- RMSE during testing was ranging from 0.35 to 0.25 depending on the number of yield sequence values provided and rest predicted
- In case of only initial yields provided, deviation is great and the model doesn't perform well enough
- Exact results have been displayed in the jupyter notebook

Future improvements

A great amount of data is required in order to train the model to achieve accurate yield values. Also, the 13 files we have used contain very different combination of input parameters in sensors since they have been collected in different regions, greenhouse settings and different scenarios. The key here is to use several files (several crop cycles) of the same crop grown in the same setting so that the model first learns firmly according to that specific setting.