Multilayer Perceptron Model to Predict Yield of Crop Products

934G5: Machine Learning (2024 - 2025)

Candidate No. 291159

May 2025

Abstract

Agriculture plays an important role in the global economy. With the continuing expansion of the human population, understanding worldwide crop yield is central to addressing food security challenges and reducing the impacts of climate change. In this report, I am presenting a working multilayer perceptron model to forecast the yield of crop products for a geographical region a year in the future.

1 Performance

1.1 Metrics

We have used 3 different metrics to measure the model performance on test data (unseen dataset). Each metric measure different trait and capability of the model. The model achieved a Mean Absolute Error (MAE) corresponding to 7.63 % which indicate that predicted yield values deviated from the ground truth by over 7 %. The Root Mean Squared Error (RMSE) recorded was 16.03 % of the average yield, which is relatively low spread of prediction errors, given the heavy penalty for larger deviations. Finally the R^2 score was 0.9892 which means the model successfully explained 98.92 % of the variance in the yield values across the test set.

Metrics equations and parameters:

1. Mean Absolute Error (MAE): The average of the absolute differences between predicted and actual values. MAE help us in measuring the average performance of our model as it is robust to outliers compared to MSE, and all errors contribute equally regardless of the size (Terven et al., 2025).

   $\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \widehat{y_i}|$

2. Root Squared Mean Error (RMSE): The average of the squared differences between predicted and actual values. RMSE captures how large the errors are, giving more weight to larger deviations (Terven et al., 2025).

   $\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y_i})^2}$
   where:

   - y_i= actual value (true yield)
   - $\widehat{y_i}$ = predicted value
   - n = number of test samples

3. Coefficient of Determination R^2: Assess how well a model's predictions explain the variability of the actual data (Terven et al., 2025).

   $R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}i)^2}{\sum i=1^n (y_i - \bar{y})^2}$
   where:

   - $\bar{y}$ is the mean of actual values
   - Numerator = residual sum of squares
   - Denominator = total sum of squares

   when $R^2 = 1$, then the model predictions match actual values, and when $R^2 = 0$, the model does no better than predicting the mean.

Figure 1 shows the training and validation loss over 38 epochs. Training and evaluation losses decrease in the first 10 epochs and converge to near zero values. This indicates that the model is learning with generalization. The use of early stopping with a patience of 8, and a learning rate scheduler (plateau patience of 3), have led the model to a stable minimum while preventing excessive training.

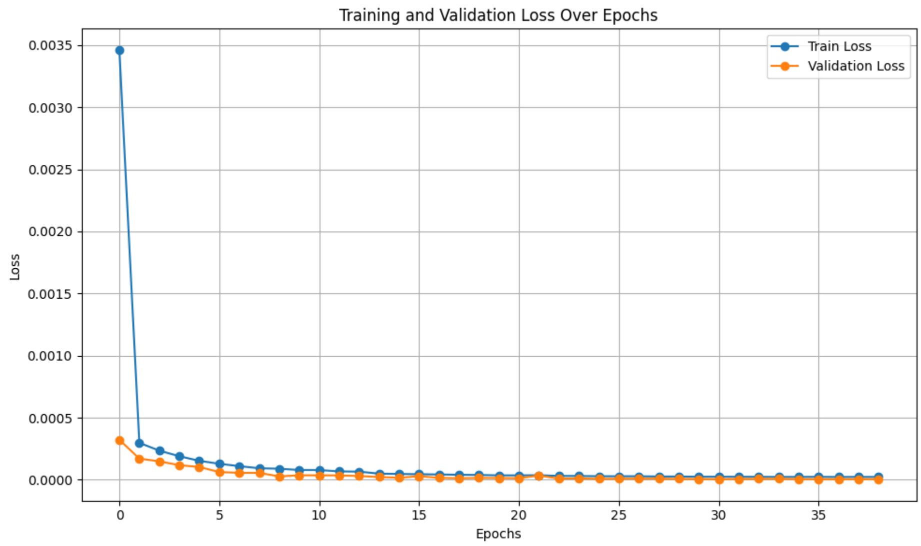

Figure 1: Training and validation loss over epochs.

Figure 2 presents the predicted vs. actual yield in the test dataset. Most points are close to the perfect prediction diagonal line, showing that the model produced accurate predictions throughout most of the yield range. However, we have had several outliers that contributed to the RMSE increase. We identified two outliers as shown in figure 2. Those two outliers are: watermelons in the Dominican Republic and papayas in Guyana (see the end of the notebook for data). Upon checking raw dataset we noticed that watermelon item record in Dominican republic showed extremely high yield in 2022, but the model had access to only two prior years of data (2018–2019) during training due to limited availability (past years were missing), which was not enough for the model to generalize effectively. The second outlier, was papayas item in the country of Guyana, we noticed the yield values showed extreme variance, increasing from 14,871 in 2010 to over 308,000 by 2019, with irregular jumps and drops across years. Additionally, the corresponding feature data contained multiple missing values. In future implementation, we plan to filter out those entries during preprocessing to improve accuracy and stability of the model.

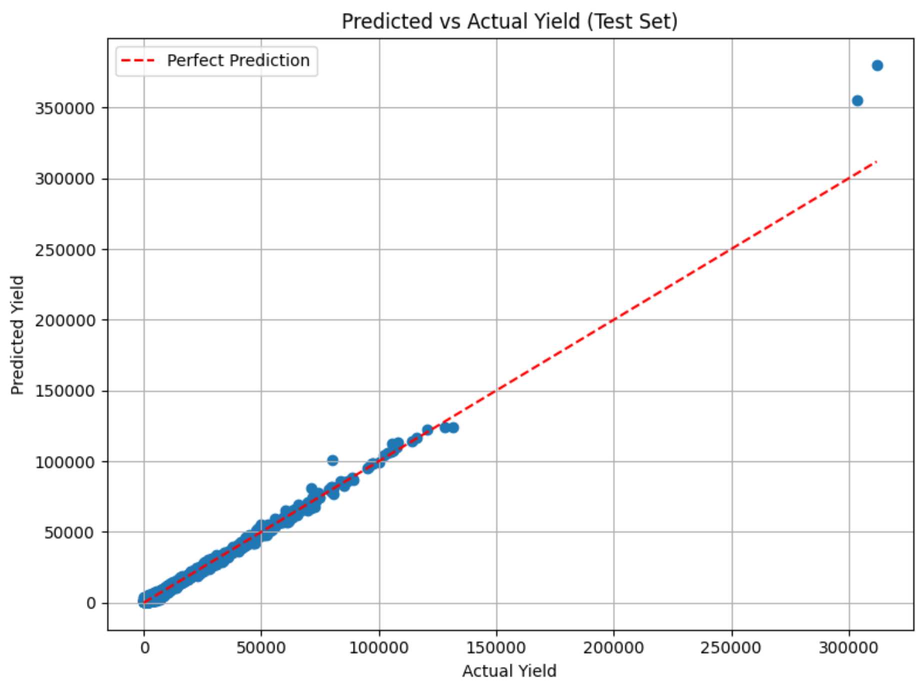

Figure 2: Predicted vs. actual yield on test dataset

1.2 Data instances, split and evaluation

Our training data shape is (52,163 x 249), validation data shape is (5,166 x 249) and test data shape is (5,169 x 249). We have 14 features (see section 3 for details), each feature has 12 months of data. After merging the geographical points and features, we sorted the years in ascending order for all countries, then shifted the target (yield) by -1, so that the current set of features will be trained on next year's yield. It was important to perform shifting step prior to splitting dataset. The rationale behind this step is to prevent data leakage from recent year to training and validation datasets. See table 1 below, where target column is the yield shifted by -1:

Year	Country	item	Features	Yield	Target
2019	Afghanistan	Apples	...	9083.2	10562.6
2020	Afghanistan	Apples	...	10562.6	10559
2021	Afghanistan	Apples	...	10559.4	10600
2022	Afghanistan	Apples	...	10600	-

Table 1: Shifting target of yield by (-1): current features results in next year yield.

We have performed splitting on the data based on two thresholds: test_year is set to 2021 (predicting targets for 2022) and the val_year value is 2020 (validating the model on data from 2020) and training dataset will contain all years below 2020 (2010 - 2019). The function ensures for each (country, item) pair, the data is split into train, val, and test, respecting the chronological order of years. This ensures that the model only sees past data during training.

Splitting data was based on two thresholds: test_year and val_year. the former is set to 2021 (predicting yield for 2022) and the latter is set to 2020 (predicting yield for 2021). Training dataset is set all years below val_year. This ensures that the model only sees past data during training.

2 Model

In our first iteration we used a Convolutional Neural Network (CNN) architecture by reshaping the data in a 3 dimension shape (n_samples, 12, n_features), the aim was to leverage the temporal structure of the data. However, the model struggled to generalize and training was unstable, likely due to overfitting and insufficient temporal depth. As a result, we abandoned CNN model. We have used Multilayer Perceptron (MLP) with embedding layers to increase the model capability in capturing categorical context. We used the following embeddings:

- Country embedding: to encode each country as a vector of 8 dimensions (value is chosen based on parameter sweep).
- Year embedding: to encode the year as a 4-dimension vector (value is chosen based on parameter sweep).

The two embeddings (country and year) were concatenated with one-hot encoding of items and scaled numerical features and passed through a feedforward neural network. In the below table, we describe the layers of the model:

Layer type	Description
Input layer	features and embeddings
Linear 1	512
BatchNorm, ReLU, Dropout	0.3 dropout
Linear 2	256
BatchNorm, ReLU, Dropout	0.3 dropout
Linear 3	128
BatchNorm, ReLU, Dropout	0.2 dropout
Output	1 for yield and a Sigmoid is used to limit the predictions between 0 and 1

Table 2: Layers of MLP

The optimisation algorithm used in our MLP model is Adam (Adaptive Moment Estimation) which is an extension to stochastic gradient descent and it's used to update neural network weights during training (see code below). It adapts the step size individually for each parameter, based on how noisy the gradient is.

```
# Adam optimiser
optimizer = torch.optim.Adam(model.parameters(), lr=0.0005)
```

Adam update: $\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}$, where:

- η is the learning rate. We have chosen learning rate of 0.0005 to balance fast learning and stability.
- ϵ is a small value for numerical stability (default 10^{-8}).
- β_1 and β_2 are decay rates for the moment estimates (defaults $\beta_1 = 0.9$, $\beta_2 = 0.999$).

In terms of Loss function, we've used smooth L1 (also known as Huber loss). The reason we picked Smooth L1 is it behaves like MSE when the error is small, and like MAE when the error is large helping in reducing the effect of outliers (Terven et al., 2025).

$$loss(x,y) = \begin{cases} 0.5(x-y)^2 & if|x-y| < 1 \\ |x-y| - 0.5 & otherwise \end{cases}$$

Below table we describe the hyperparameters of our model:

Parameter	Value
Learning Rate	0.0005
Batch Size	32
Epochs	Up to 100
Country Embedding Dim	8
Year Embedding Dim	4
Scheduler	ReduceLROnPlateau (factor = 0.5, patience=3)
Early Stopping	threshold = 8 epochs
Dropout	0.3, 0.3, 0.2

Table 3: Hyperparameters used in MLP

To prevent the model from overfitting we used various methods, from early data preparation to model development. Below is list of those methods:

1. Data splitting based on years: We did split based on chronological order per country and item such that:
    - Training: all years before the validation year.
    - Validation: Single fixed year (2020).
    - Test: future year (2021).

    This method will ensure no data leak from the recent years to early years.

2. Dropout layers: We added dropout layers after each layer to randomly deactivate neurons during training. This will ultimately prevent the model from depending on specific paths in the network.

3. Batch Normalisation: Applied after each linear layer to stabilize and normalise activations.

4. Early stopping: The validity loss was monitored for each epoch. If there was no improvement for 8 epochs, the training would stop automatically to prevent overfitting of the training set.

5. Learning Rate Scheduler: we used ReduceLROnPlateau to reduce the learning rate by a factor of 0.5 when validation loss plateaued. This allowed the model to refine its learning with smaller and more stable updates as training progressed.

6. Embedding Layers and Hyperparameter Sweep for Embedding Dimensions: We embedded country and year columns. This helped preventing overfitting and scale better with data complexity. To identify the optimal embedding sizes for country and year, we developed a parameter sweep to evaluate how different embedding sizes impact model performance on the validation set Instead of choosing arbitrary dimensions. The parameters we tested were:
    - Country embedding: [4,8,16,32]
    - Year embedding: [2,4,8]

    For each combination, we trained the model with the same architecture and settings and recorded the validation loss (MAE and RMSE). A larger embedding dimension allows the model to learn more complex relationships but may lead to overfitting, especially with small datasets. A smaller embedding is more compact but may under represent country level or year level patterns. We selected the combination that gave the lowest validation loss (MAE/RMSE) consistently and did not show signs of overfitting.

7. Parameter sweep for batch size: We basically wanted to test different batch sizes when training the model and evaluate their performance based on validating MAE and RMSE. This allowed us to select batch size that balance stable gradient updates with efficient training.

3 Features and Labels

3.1 Input (features):

1. Climate features (12 months):
   - Data points:
     - Rainfall (rain_1 to rain_12)
     - Snow (snow_1 to snow_12)
     - Soil moisture, 0–10cm layers (soilmoisture_0_10_1 to soilmoisture_0_10_12)
     - Soil moisture, 10–40cm layers (soilmoisture_10_40_1 to soilmoisture_10_40_12)
     - Soil moisture, 40–100cm layers (soilmoisture_40_100_1 to soilmoisture_40_100_12)
     - Soil moisture, 100–200cm layers (soilmoisture_100_200_1 to soilmoisture_100_200_12)
     - Soil temperature, 0–10cm layers (soiltemp_0_10_1 to soiltemp_0_10_12)
     - Soil temperature, 10–40cm layers (soiltemp_10_40_1 to soiltemp_10_40_12)
     - Soil temperature, 40–100cm layers (soiltemp_40_100_1 to soiltemp_40_100_12)
     - Soil temperature, 100–200cm layers (soiltemp_10_200_1 to soiltemp_100_200_12)
     - Transpiration (tveg_1 to tveg_12)
     - Terrestrial water storage (tws_1 to tws_12)
     - Plant canopy surface water (canopint_1 to canopint_12)
   - Extraction process:
     - Climate datasets were provided with geographical points (latitude/longtiude). We joined those points with countries using geospatial nearest neighbour join (see 4 preprocessing).
     - For each climate variable, 12 monthly values were included, and summary stats were computed (Mean, Standard Deviation, Min, Max) to capture seasonality and variability, which are important to predicting yield.
   - Rationale:
     - Yield is influenced by weather conditions such as rainfall, snow, temperature, and soil temperature and moisture. The monthly data provides seasonal patterns, to give the model ability to learn data variability and seasonal changes in-depth we've added summary statistics for each feature as follows:
       (a) Mean
       (b) Standard deviation
       (c) Minimum
       (d) Maximum
2. Land cover features (17 categories):
   - Data points:
     - mean_cov_1 to mean_cov_17
   - Extraction process:
     - Land cover is categorized into physical surface types (e.g., cropland, urban, forest) in each country and represented as percentage coverage in 17 categories.
     - Aggregated from Land_Cover_Percent_data.csv by spatially joining each point to the nearest country (mean_cov_1 to mean_cov_17).
     - Multiple land cover points were matched to each country, so we aggregated them by taking the mean value per land cover class, producing one average percentage per class for each country.
   - Rationale:
     - Including the land coverage gives the model a richer understanding of the agricultural land dynamics of each country.
3. Country, Year and Items:
   - Data points:
     - Country (e.g., Afghanistan, Albania, etc)
     - Year (e.g., 2010, 2011, etc)
     - Items (e.g., Apple, Watermelon, etc)
   - Encoding:
     - Country and year were embedded with 8 and 4 dimensions respectively.
     - Items were one-hot encoded to capture crop type without implying ordinal relationship.
   - Rationale:
     - Embeddings helps the model to learn latent patterns (e.g., country specific). While one-hot encoding of items prevent the model from learning numerical representations and adding bias.

3.2 Output (label):

The target variable is yield. in our model it will predict yield for the year 2022, and we will compare it against ground truth, predictions are exported inside the file yield_predictions.csv

- Data points:
  - Yield
- Extraction process:
  - From the FAOSTAT Yield_and_Production_data.csv.
  - Filtered to only include element = "yield", and exclude "production".
  - The column "yield" was shifted by -1 per (country, item) group: $target(t) = yield(t + 1)$. This makes the model predict future yield based on this year's features.
  - The label column was scaled using MinMaxScaler during the preprocessing of data. The rationale behind applying MinMaxScaler to the label (common in regression tasks) is we want our features and labels to be the same scale so that the model can learn effectively, the other reason is we're using sigmoid() on the last layer, and the output is bounded by [0,1].
- Rationale:
  - Predicting future yield based on past data.

4 Preprocessing

To prepare the data for MLP model, several preprocessing steps were applied. These ensured data quality, consistency, and model readiness, especially for handling temporal, categorical, and geospatial information. Below are the steps taken, and the rationale behind using them:

1. Data Loading and Initial Cleaning: We loaded the country lookup table (with latitude and longitude), the FAOSTAT yield and production data, land cover percentage data, and all climate datasets (See 3 features and labels). In this early preprocessing step our aim was to have one core sheet with all data points standardized and clean the input data to ensure that all downstream merges and transformations can be performed reliably. Additionally, we performed cleaning to column names and filtering to yield values, which eliminated irrelevant data early and avoided mistakes later.

2. Spatial joins of climate features: We "spatially" joined each climate dataset geopoints (latitude, longitude) to the nearest country using a nearest neighbor approach. This step took a lot of iterations. To assign each latitude and longitude to a country, we considered each country as a circle (given that radius is available in dataset) and we matched geo points within circles, but this method had substantial gaps (not accurate, some circles were overlapping). We then transitioned into polygons method, we downloaded the polygons of all countries (as a geojson file) and then checked whether geo points lie within the polygon of a country, this method was slow and it required downloading geojson dataset. Lastly, we tried nearest neighbour approach and it worked perfectly. After joining spatial data with countries, we calculated the mean value of climate data for each (country, year) pair across all matched points, for example if Afghanistan have 10 records of rain feature then we calculate the mean across those records and the result will be one record only per country per year. We aggregated by country-year to align the climate data with the yield data. Here is how the function "process_monthly_climate()" we've developed will do the spatial join:

   (a) first, we load and read the climate data (features).
   (b) We convert the latitude and longitude into shapely point objects.
   (c) Next, we convert the climate points and the country reference points to GeoDataFrames and reproject them into a metric coordinate system (EPSG:3857) to allow accurate distance-based matching.
   (d) We then match each climate point to its nearest country point using: gpd.sjoin_nearest()
   (e) Now that we have the data joined and countries are assigned, we rename the columns (e.g., rain_1, rain_2).
   (f) Lastly, aggregate by (country, year) using group by and mean() to align with the target yield.

3. Spatial join of land cover features: Land cover percentage points were spatially joined to the nearest country. We then computed the mean percentage for each land cover class at the country level. Since land cover does not vary annually in the available data, aggregating it once at the country level was sufficient.

4. Dataset Merging: Merged climate features, land cover features, and yield data into one dataset. It's worth noting that merged dataset have 165 distinct countries and 12 total years (2010, 2021).

5. Adding Summary Statistics for Monthly Features: For each set of 12-monthly features (e.g., rainfall, snow, soil temperature 0–10cm, ..etc), we calculated the "mean", "standard deviation", "minimum", and "maximum" values across the 12 months. We added summary statistics to capture variations across seasons, and improved the model's ability to catch agricultural patterns (such as drought seasons or flooding).

6. Shifting the yield to create the target (yield): The rationale behind this step is to predict next year's yield using this year's features. Shifting the target ensures that the model learns to predict next year's outcome rather than simply fitting to the same year's data.

7. Missing Value Handling: We filled missing values in features with the mean of training data, and we converted all features to numerical float type. This step was necessary to prevent the model from crashing due to NaNs. Additionally, using the mean from training makes the model avoid data leakage.

8. Dropping Rows with Missing Target: After the shift, rows where the target became missing (i.e., the last year available for a (country, item) pair) were dropped from the dataset. Dropping these rows ensured a clean training set without introducing NaNs into the labels.

9. Data split: Split dataset based on year into train, validation, and test respecting chronological order.

10. Feature scaling: All numerical features were scaled to a [0,1] range using MinMaxScaler, with the scaler fitted only on the training data and then applied to the validation and test sets.

11. Target Scaling: Scaled the target yield separately. We noticed from the raw data that yield varies enourmeously across items, countries and years. This may lead the model to converge and cause instability. Thus we decided to scale the target to [0,1] values before training. After prediciton we transformed back to original outputs using the inverse of the scaling function.

12. Tensor Conversion: We converted all inputs to pyTorch tensors: Numerical input, country/year embeddings, and scaled target. This step prepares data for training using pyTorch DataLoader function.

5 References

1. Terven, J., Cordova-Esparza, D. M., Ramirez-Pedraza, A., Chavez-Urbiola, E. A., Romero-Gonzalez, J. A. (2025). Loss Functions and Metrics in Deep Learning. Artificial Intelligence Review, 58(7), 195. https://doi.org/10.1007/s10462-025-11198-7