FORLand Technical Paper 03 (2024)

Application of Deep Learning to Emulate an Agent-Based Model

Ruth Njiru, Franziska Appel, Changxing Dong, Alfons Balmann

In light of the dynamic challenges facing agricultural land markets, the conventional analytical frameworks fall short in capturing the intricate interplay of strategic decisions and evolving complexities. This necessitates the development of a novel method, integrating deep learning into Agent-based Modelling, to provide a more realistic and nuanced understanding of land market dynamics, enabling informed policy assessments and contributing to a comprehensive discourse on agricultural structural change. In this paper, different deep learning models are tested and evaluated, as emulators of AgriPoliS (Agricultural Policy Simulator). AgriPoliS is an agent-based model used to model the evolution of structural change in agriculture resultant on the change in the policy environment. This study is part of preliminary works towards integrating deep learning methods and predictions with AgriPoliS to capture strategic decision making and actions of agents in land markets. The paper tests the models on their suitability, computational requirements and run-time complexities. The output from AgriPoliS serves as the input features for the deep learning models. Models are evaluated using a combination of coefficient of determination (R^2 score), mean absolute error, visual displays and runtime. The models were able to replicate the variable of interest with a high degree of accuracy with R^2 score of more than 90%. The CNN was the most suited for replicating the data. Through this work, we learned the required complexities, computational and training efforts needed to integrate deep learning and AgriPoliS to capture strategic decision-making.

Keywords: Agent-Based Model, Deep Neural Networks, predictions

JEL Code: Q15, C45

Published by
DFG Research Unit 2569 FORLand
https://www.forland.hu-berlin.de

Agricultural Land Markets – Efficiency and Regulation

1. Introduction

Amidst evolving dynamics in agricultural land markets, concerns about concentration and pricing have prompted regulatory proposals. Traditional methodological approaches struggle to capture strategic interactions and dynamic complexities. To address this, we propose fusing Deep Learning (DL), a subset of Machine Learning (ML) into Agent-based Modelling. DL excel in pattern recognition and strategic decision-making, offering a novel approach to enhance the realism of agent behaviours. This approach aims to provide a more nuanced understanding of land market dynamics and the potential impacts of regulatory interventions.

Agent-Based Modelling developed popularity in recent years for simulating complex and dynamic behavioural systems (Crooks & Heppenstall, 2012). It is a deductive-inductive process which begins with a set of assumptions and rules to produce simulated data which is then analysed to address emergent phenomena (An, 2012). An Agent-Based Model (ABM) consists of a group of decision-making entities called agents. The agents are autonomous, active and heterogenous in their decision making by following a set of behavioural rules or satisfying specific goals (e.g. profit maximization). They interact with other agents and with their environment while taking decisive action. They also can adapt/learn in their goal satisfying quest (Bonabeau, 2002; Epstein, 1999; Heppenstall et al., 2011; Macal & North, 2005; Railsback & Grimm, 2019)

The applications of ABM in the agricultural domain include their usage as farm level models that consider the interaction between farms, heterogeneity in farm behaviour, resource use changes, response to agricultural policy changes, etc. (Kremmydas et al., 2018). Some notable applications include AgriPoliS (Agricultural Policy Simulator) used to model the impact of agricultural policy change on the structure of agriculture within a specific region (Happe et al., 2006), MPMAS (Mathematical Programming-based Multi-Agent Systems) which combines bio-physical models and economic models to simulate household reactions to land use changes (Schreinemachers & Berger, 2011), RegMAS (Regional Multi-Agent Simulator) to simulate the effects of agricultural policies on income, land use and farm structures (Lobianco & Esposti, 2010), and the aporia framework for modelling the impact of decisions of farm managers on land use systems (Murray-Rust et al., 2014).

In recent years, several studies have explored the integration of ML techniques within ABM to enhance agent behaviour (Brearcliffe & Crooks, 2021). Augustijn et al. (2019) summarizes the integrations based on the stage in which ML is incorporated in the Agent-Based Modelling process; using the output of the ML model as the input to the ABM, using the ML model to predict agent behaviour within the ABM or using the output from the ABM in ML model for validation. Studies envision that use of ML techniques can lead to a better understanding of ABM output and through the integration of ML with ABM, generate models that can replicate the behaviour of ABMs with a high degree of precision and lower running time (Angione et al., 2020; Lamperti et al., 2018). Running ABMs in most cases is a time-consuming process with simulation times ranging from a few hours to even days on end. van der Hoog (2017) in a prospectus suggests that the ability of ML models to imitate the agents' behaviour has promising implications specifically for ABMs that produce time series data with long-term dependencies. However, this integration is still in its evolutionary phase and there is a lot of groundwork to cover.

In AgriPoliS, the focal ABM, we envision to integrate Deep Neural Networks (DNNs), the key component of DL, predictions to shape the agents' decision-making and subsequent action. DNNs are effective through their ability to learn complex patterns from inputted data and infer their own conclusions (Goodfellow et al., 2016). Angione et al. (2020), also demonstrated that DNNs outperformed other ML models in emulating ABM. DNNs techniques have also been applied successfully to analyse time series data due to the ability to capture temporal dependencies. This is important for AgriPoliS since it captures the evolution of farms over a 25-year period. The overall vision in our work is to generate artificially intelligent agents who make smarter decisions on farm production, investment, rental markets and exit/no exit decision. The generation of smarter agents would affect structural change through strategic interactions between farms and strategic responses to complexities.

A particular challenge for training the DNNs is the availability of data for the learning process. Because empirical data is neither available nor suitable, we generate data by following the "doppelgänger approach" described by van der Hoog (2019). The idea of the doppelgänger approach is to run a number of identical simulations with alternative decisions models, i.e., for each scenario there is one version with the standard myopic optimization (which also can provide benchmarks) and one version with a learning DNN. In this setting, scenarios have to be understood as variations in the initialization of the model. The specific development of AgriPoliS DNN will begin with a static farm-level approach, which serves to identify reasonable structures, interfaces for the DNN to the agent's decision, and initializations of the DNN. Practically, we train the DNN to make reasonable decisions on farm exit, investments, and land rentals for a farm agent in a certain state, period, and environment with specific information. Following the "doppelgänger approach" the benchmark for such a reasonable decision can be the myopic optimization of the farm within the standard AgriPoliS model. Therefore, we apply supervised training in this first step. This innovative methodology not only enhances the predictive power of the model but also offers insights into the strategic behaviour of agents in response to varying market conditions, thereby enriching our understanding of land market dynamics.

In light of this, in this study we seek to understand how DNNs work, suitability of method, computational requirements and time complexities to run them. We hence test different DNNs algorithms as computational models of AgriPoliS. The output from AgriPoliS serves as the input features for the DNNs. The DNNs algorithms are chosen based on their ability to emulate the whole panel data output structure of AgriPoliS and also the ability to capture temporal dependencies generated by AgriPoliS since it captures the evolution of farms over a 25-year period. The DNNs are then evaluated based on their performance. We also capture the training time required to run the models. For this study, we begin by focusing on a single variable of interest to emulate i.e. profit since the main objective of agents in AgriPoliS is to maximize profit. We also demonstrate how DNNs can be adapted to emulate multiple variables.

This paper is organised as follows: Section 2 provides an overview of AgriPoliS and an overview of DNNs. Section 3 describes the materials and methods used to emulate the ABM using DNNs. Section 4 presents the results from the training process. Section 5 deals with the discussion, conclusion and outlook.

2. Overview of ABM and DL

a. AgriPoliS

AgriPoliS is used to simulate the effect of diverse policies on farm structural change within a region over a certain period of time (Balmann, 1997; Happe et al., 2006). An overview of, including documentations, source code and manual is available on AgriPoliS. The focal agents in AgriPoliS are the farm agents. They are modelled to closely represent the typology of farms as would be observed in the region of interest. Each agent makes production, investments, land rental decisions and exit/not exit decisions on the farm for maximization of household income for family farms and maximization of profit in case of corporate farms. The farm agent can also adjust their decisions in response to changes in their environment and actions of the other agents (Happe, 2004).

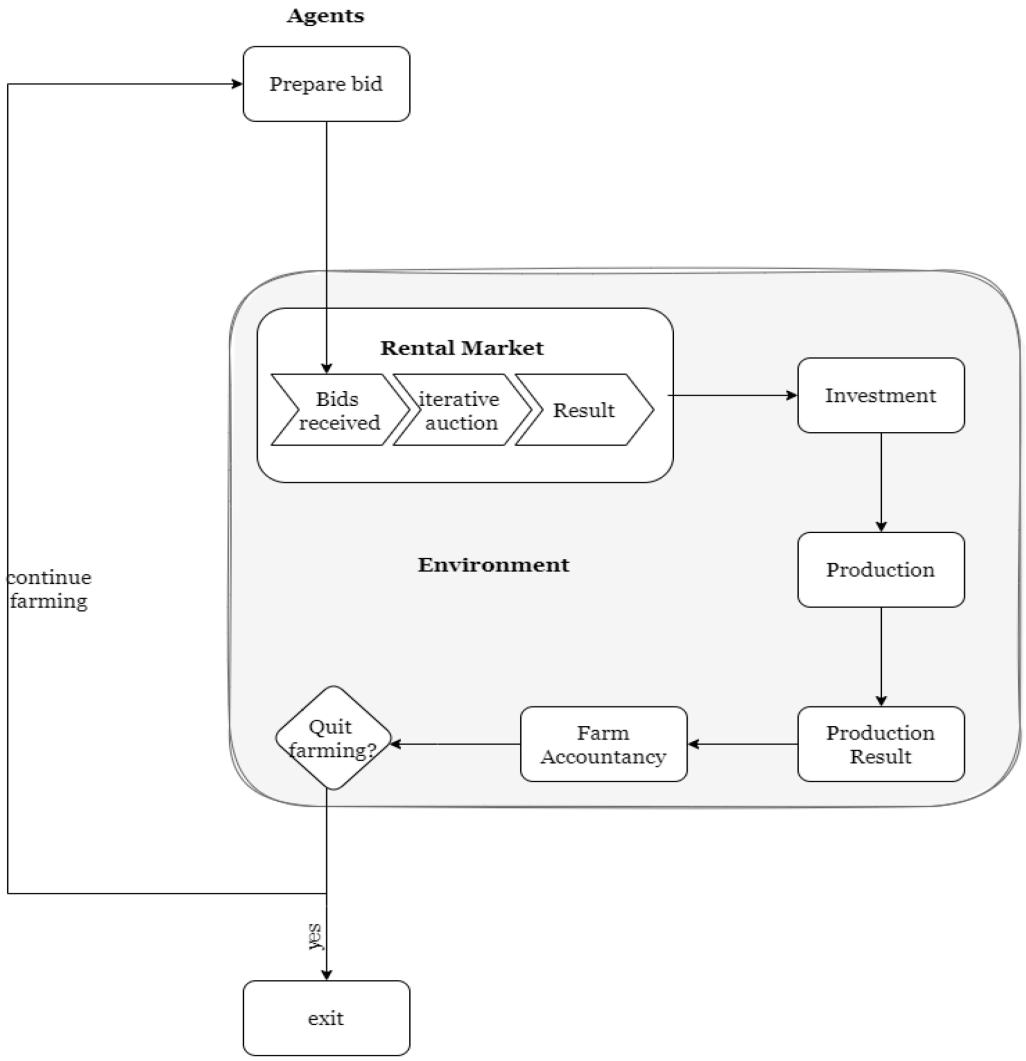
The AgriPoliS environment is created using data representing the aggregate regional capacities, organization and the economic indicators of regional farms. Data sources include European Union's Farm Accountancy Data Network (FADN), handbook data on farming practices (e.g., for Germany, Association for Technology and Construction in Agriculture(KTBL)), farm structural survey (FSS), farm data and/or expert knowledge (Sahrbacher & Happe, 2008). Key information defined prior to initialization include production activities, investment options, financing and labour.

The model is usually initialized with 15-20 typical representative farms. Using mixed-integer programming, the agents decide on production, investment and farm exit with the aim of maximizing their profit (corporate farms) or household income (family farms). The model typically runs over 25 iteration/years (Sahrbacher et al., 2012). This to account for generation change where the farm is handed over to the next generation of farmers.

In AgriPoliS, farms grow through renting additional land in the land rental market. The farms interact through the land market where they compete against each other by bidding for plots of land under auction. The farms present bids to the land rental market. The bids reflect the shadow price (additional benefit of renting the land), the spatial location of the plot (calculated by the transport cost between the farm plots and the plots available for rent) and a fixed land rental coefficient which represents the share of the bid that will go to the land owner. The agent with the highest bid receives the plot. The auction is held in an iterative manner until all the plots are allocated. The agents then decide on investment and what to produce.

At the end of the iteration/year, the financial statements are generated which inform the farms on whether to continue and/or exit farming. The farm then prepares his bid for the following year and subsequent decisions are made over 25 iterations. Figure 1 shows the framework for the farm agents. For every iteration/year, AgriPoliS produces farm investment data (e.g. stables, land, machinery), farm production data (e.g. level of crop yields, livestock units produced), farm standard indicators data (e.g. profit, equity capital, change in equity), sector data (number of farms, amount of used land) and sector prices data.

Figure 1: Simplified representational framework for AgriPoliS

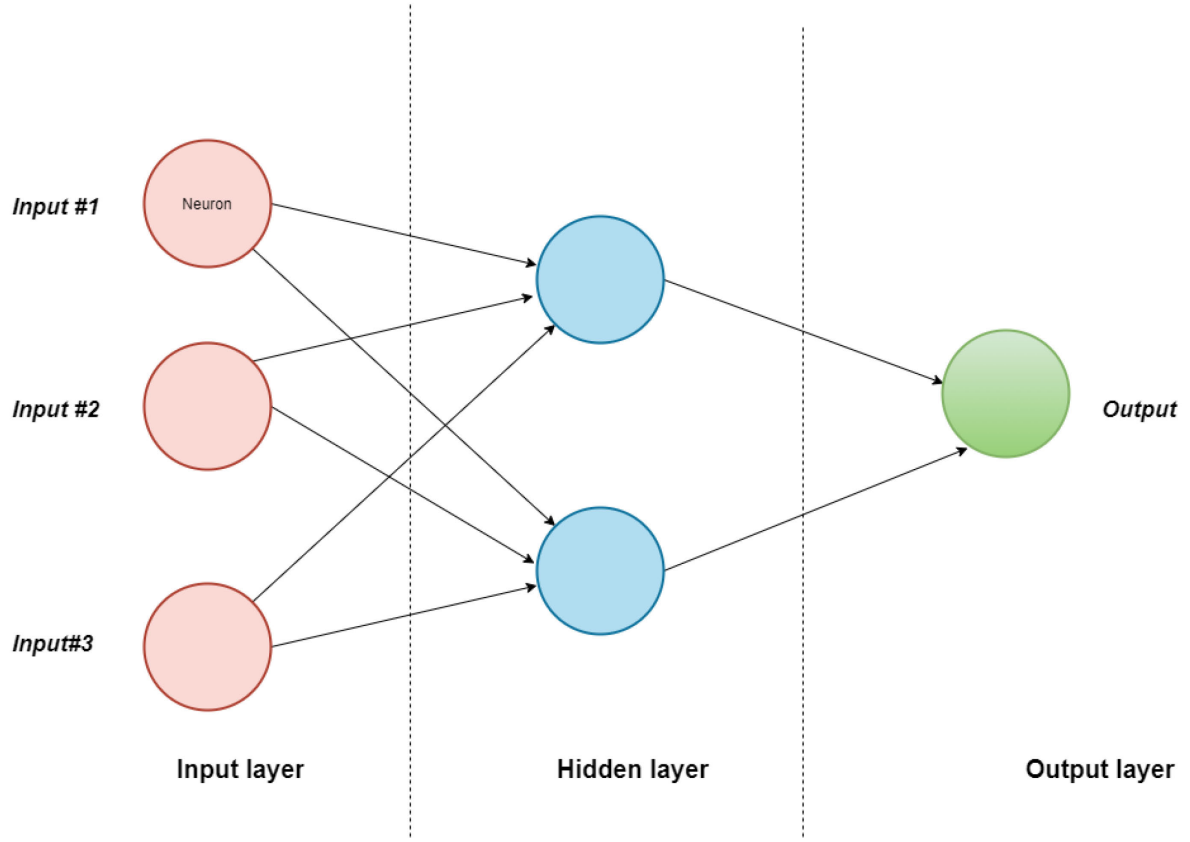


Source: Own figure

b. Overview of DL

DL is an area of ML that stems from the workings of the neurons in the brain. The fundamentals of DL are the Artificial Neural Networks (ANN). ANN are made up of the input layer, hidden layer(s) and output layer consisting of neurons which act as the data processing units (Goodfellow et al., 2016; LeCun et al., 2015). ANN can either be shallow i.e. having only one hidden layer between the input and output layer or deep where there are multiple hidden layers. For simplicity and consistency, we use the term Deep Neural Networks (DNNs) in the description of DL. The architecture of the DNNs is made up of the input layer which feeds the data into the DNN, the hidden layers act as the computational centre while the output layer does the predictions (Figure 2). The layers are fully connected with each neuron in one layer connected with a weight to every neuron in the next layer.

Figure 2: A simple representation of a DNN



Source: Own figure based on (Goodfellow et al., 2016)

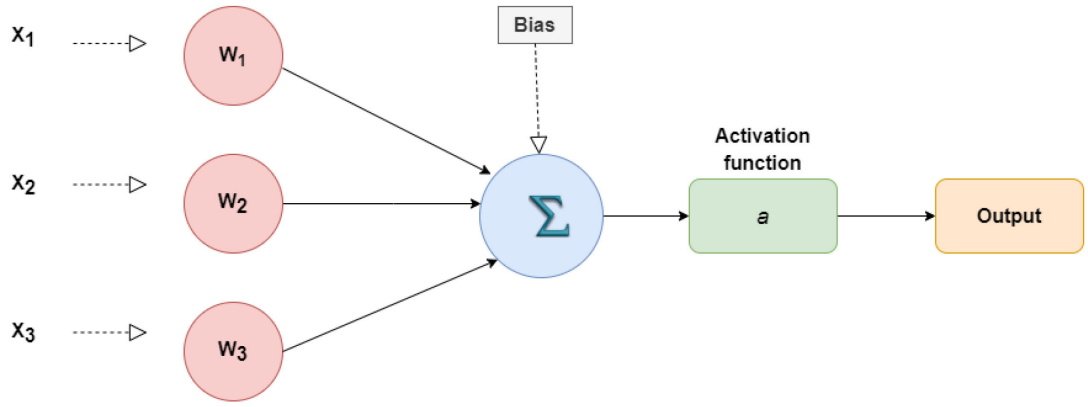
The artificial neuron is a mathematical function that takes the input (*x*) and multiplies it by a value known as weight (*w*) (DNNs algorithms work through a process known as optimization, i.e. learning from the data by minimizing the error between the predicted and the actual values using a loss function i.e. finding *w and b* that minimize the loss function. The optimization algorithm (optimizer) modifies the weight and learning rate to reduce the overall loss and improve the accuracy while the loss function evaluates how well the model is predicting. During training there is a forward propagation step which computes the output. This output is the propagated backwards to compute the gradients/derivatives of the loss function (Rumelhart et al., 1986). The gradients help in minimizing the deviations between the actual output and the predicted output. The choice of the best optimizers is dependent upon the task at hand and amount of data available. The most common optimizers are Stochastic Gradient Descent (SGD), Mini-Batch Gradient Descent, Adagrad, RMSProp, AdaDelta and Adam (Goodfellow et al., 2016). The choice of the loss function is dependent on whether the goal is predicting a categorical value e.g. employment status or a numeric value e.g. amount of income. There are different loss functions but the most common when dealing with numerical values are the Mean Absolute Error (MAE) and Mean Squared Error (MSE). The choice of the cost function and optimizers is made before starting the process of training the DNN

Figure 3). The weights are an indication of the importance associated with the inputs. At this stage a bias term (*b*) is then added which helps the model to best fit the given data by adjusting the output with the weighted sum of the inputs in the neuron. They are then transformed into output through activation function (*a*). Most activation functions are non-linear to capture

patterns in the data better. The choice of the activation function is dependent on the DNNs' goal (Chollet, 2021). The activation function then passes the value to all the neurons in the next layer or it returns it as the final value ($\hat{y}$) when it is passed from the last hidden layer.

DNNs algorithms work through a process known as optimization, i.e. learning from the data by minimizing the error between the predicted and the actual values using a loss function i.e. finding *w and b* that minimize the loss function. The optimization algorithm (optimizer) modifies the weight and learning rate to reduce the overall loss and improve the accuracy while the loss function evaluates how well the model is predicting. During training there is a forward propagation step which computes the output. This output is the propagated backwards to compute the gradients/derivatives of the loss function (Rumelhart et al., 1986). The gradients help in minimizing the deviations between the actual output and the predicted output. The choice of the best optimizers is dependent upon the task at hand and amount of data available. The most common optimizers are Stochastic Gradient Descent (SGD), Mini-Batch Gradient Descent, Adagrad, RMSProp, AdaDelta and Adam (Goodfellow et al., 2016). The choice of the loss function is dependent on whether the goal is predicting a categorical value e.g. employment status or a numeric value e.g. amount of income. There are different loss functions but the most common when dealing with numerical values are the Mean Absolute Error (MAE) and Mean Squared Error (MSE). The choice of the cost function and optimizers is made before starting the process of training the DNN

Figure 3: Representation of a Neuron in a DNN



Source: Own figure

Another key consideration is that there are different types of DL models and the choice of the model is dependent of the data type, data structure, data amount and the goal to be achieved. DL techniques are divided into 3 major classes: supervised learning, unsupervised learning and reinforcement learning. Supervised learning works well with labelled data and has a pre-defined output for prediction tasks. It is mainly used for classification and regression. Unsupervised learning is used for unstructured data for structuring or pattern identification. It can be classified into two categories; clustering and association. In reinforcement learning the agent acts which leads to a change in their environment. The agent consequently uses the newly acquired state to determine the next action with the goal of maximizing reward (Vasilev et al., 2019).

For this study, we employed supervised learning. There are different models under supervised learning but only a few can capture the temporal dependencies in the AgriPoliS model. To that effect, we choose to focus on DNNs models that would be able to better emulate the cross-

sectional time series nature of AgriPoliS. The DNNs models chosen were; Recurrent Neural Network (RNN), Convolution Neural Network (CNN), Long-Short Term Memory (LSTM), Bidirectional Long-Short-Term Memory (Bi-LSTM), Gated Recurrent Unit (GRU) (LeCun et al., 2015). A brief introduction of the models is supplied below:

RNN is used for modelling sequential data. The output from the previous layers act as the input to the current layers. They contain a hidden state that can remember information about the sequence. This feature makes it useful for time series prediction. Unfortunately, they suffer from gradient vanishing problems where the gradients tend towards zero and exploding gradients where the gradients become too large making it unsuitable for very long sequences. RNNs been applied in Natural Language Processing (NLP), sequence to sequence learning and speech recognition (LeCun et al., 2015).

LSTM, an advancement over RNN, has the ability to learn long term dependencies and was developed to deal with the problem of vanishing gradients by a data filtration system which discards irrelevant data and only retain relevant data for use in subsequent steps (Hochreiter & Schmidhuber, 1997). The key idea behind the LSTM is the cell state where the choice for information retention or discard is done through structures known as gates. Information flows through the three main gates. The first gate is the forget gate which makes a choice on whether to keep or discard information. Once, the choice is made, the retained information passes through the input gate to the cell state. In the cell state, the data is processed and new information added to it. It then passes to the output gate. The information from the output gate is sent as input to the next cell state. LSTMs have been applied in NLP, machine translation and time series prediction (Hua et al., 2019). Another modification is the Bi-LSTM, which can reverse the direction flow of information i.e. information can flow both forward and backwards. By combining information in both directions its able to produce more meaningful output.

Another variation of the RNN is the GRU which has 2 gates i.e. the reset gate that filters the information which needs to be forgotten and is directly connected to the previous state. It also contains the update gate which filter the information that will be used in the future (Cho et al., 2014; Yamak et al., 2019).

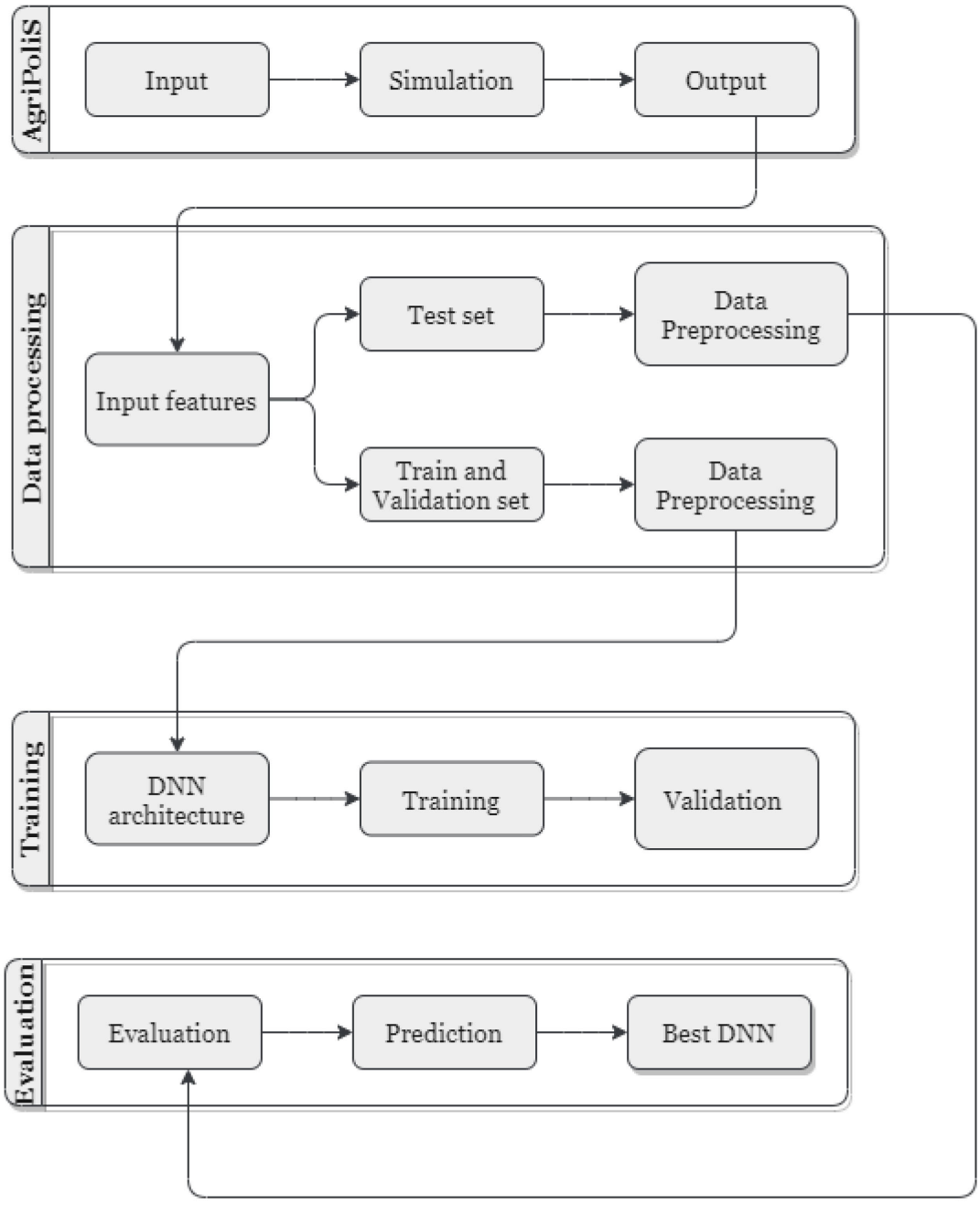
CNN is also another DNNs model that captures both spatial and temporal dependencies by extracting and learning valuable features of the time series task. Although the CNN was originally designed for image processing, they can be adapted for data processing by changing the output layer. CNNs are able process 1D (time series data, sequences), 2D (images or audio) and 3D (videos and images) data. They consist of a convolution layer, pooling layer and fully connected layer. A variation of the CNN is the CNN-LSTM model which combines CNN for feature extraction and then LSTM for time series prediction (Jin et al., 2020).

3. Methodology

In the study, we follow the “doppelgänger approach” by van der Hoog (2017). We first generated training data in AgriPoliS by running identical simulations with variations in the initialization of the model i.e. different scenarios. We then tested the ability of different DNNs to emulate the behaviour of standard AgriPoliS agents. In this study, we applied a supervised learning approach to test different DNNs to emulate one output variable i.e. profit in AgriPoliS. We also adapted the code to show the potential for multi-output variables replication by replicating three variables. The models were chosen based their ability to deal with the cross-sectional time series nature of the AgriPoliS output. RNN, CNN, LSTM, Bi-LSTM, CNN-LSTM

and GRU were tested and evaluated. Figure 4 shows the framework for the implementation of the work as discussed in the sections:

Figure 4: Methodology for replication of AgriPoliS profit using DNNs.



Source: Own figure

a. AgriPoliS data generation

The first step in the study was to generate the simulated data to be used to train the DNNs. AgriPoliS was randomly initialized with 205 farms/agents over a 25- iteration period to generate data. One iteration corresponds to one production year. To ensure robustness, there were 100 simulation runs using different initializations. This resulted in 20,500 farms measured over 25-iterations for use in model training. The simulated data was then split into the training

set and validation set during the data processing phase. For carrying out model evaluation and to deliver unbiased results, we generated test data of 30 simulation runs separately resulting in 6,150 farms. The key variables used are shown in Table 1.

Table 1:Selected AgriPoliS output farm level key variables used as the input features in the training of the DNNs

Variable	unit	Description
Output variables		
Profit	€	The difference between the revenue received from the livestock and crop production and the cost of production (inputs, machinery, labour)
Input variables		
Labour	labour unit	The amount of manpower expressed in hours that goes in the production process. Labour can either be family labour, fixed labour or variable labour.
Land	ha	Used for livestock and crop production. The land is either owned or rented and is further distinguished between arable land and grassland.
Investments	count	The number of fixed assets distinguished by type which are used for production. Examples include stables, milk parlours, machinery.
Rent	€	Amount of money paid for the use of rented land obtained from the land rental market. Distinguished between arable land and grassland.
Liquidity	€	The available cash to cater for the running costs of the farm
Borrowed capital	€	The amount of short term or long-term credit taken by the farm to meet its obligation. The cost of the capital is interest
Products	count	Total units of livestock (fattening pigs, broilers) and crops (wheat, barley, sugar beets, potatoes) produced.
Depreciation	€	The wear and tear from use of the fixed assets.

b. Data processing

The data processing was performed in Python 3.9.7. The output from the AgriPoliS simulations was used as the input features for the DNNs. The data processing and training was done purely on a CPU to allow for model comparison. Prior to training the DNNs, the data had to be processed into a form suitable for the DNNs. The agents were recoded to generate a unique identifier which composed of the farm_ID and the simulation run[1]. Due to the nature of the data being panel data, it was indexed using the unique identifier and the iteration period.

The simulated data set consisting of 20,500 farm agents was split into a train set and validation set in 70%:30% ratio ending up with 14,350 farm agents in the train set while 6,150 farm agents were in the validation set. The test set consisted of 6150 farms generated separately to avoid instances where the farm agents end up in both the training and test set. Since the main objective of agents in AgriPoliS is to maximize profit, the variable that the DNNs will

[1] For Example, 187R60 denotes the farm whose farm_ID is 187 that is generated in simulation run 60.

emulate is profit. The code was also modified and adapted to replicate three variables with just minimal changes to the code.

The input features were then normalized using the MinMaxScaler (Aggarwal, 2018). The goal of normalization is to change the values to a common scale without altering the differences in the range of values to ensure faster convergence, more stable and easier learning process. For example, the revenue ranges from -240,000 to 500,000 while the age of the farm ranges from 32 years to 95 years. The revenue would influence the result more due to its larger value. The normalization transferred the features range to values between 0 and 1. Prior to usage, the panel data was transformed into a supervised learning series using the algorithm as initiated by Brownlee (2017) and adapted by Shi (2020). This creates columns of lag values and columns of forecast values. The input data was then reshaped into 3-dimensional data (samples, time steps and features) which is the recommended format for use by the DNN architectures.

c. Data training

After the data was processed, the training set was used to train the different models. Multiple simulations were done to determine the optimal hyperparameters to use. Hyperparameters control the learning process to correctly map the input features to the target variable i.e. the number of hidden layers, optimizer, activation function, loss function, number of epochs, batch size, kernel size. For the models we used the Adam (adaptive moment estimation) optimizer that works well with time series data (Kingma & Ba, 2014). We used a small learning rate of 1e-5 since larger rates were causing the model to converge too quickly to a sub-optimal solution. There was no improvement in performance with smaller than 1e-5 running rate. ReLu (Rectified Linear Unit) activation function was applied with a batch size of 32. Batch size defines the number of training examples that will be propagated through the network before updating the model weights. It affects the stability of the training process. Since the models were ran on CPU, application of a smaller batch size worked better for all the models. We used MAE to compute the loss function and generate the loss curve. We also applied early stopping which stopped training process once the loss function stopped improving to avoid overfitting. To further counter overfitting we repeated the simulations to the LSTM, Bi-LSTM and the GRU while applying dropout. Dropout works by randomly dropping out some units during training to counter overfitting (Srivastava et al., 2014).

An additional note is that DNNs are stochastic i.e. they initialize using random weights thereby producing different results. While random initialization is the best fit for getting good approximations, it concomitantly does little for getting reproducible results. Due to that, it was necessary to run the models several times while doing hyperparameter tuning to get the best results. Afterwards, we used a random seed to achieve reproducible results (Brownlee, 2016). We also recorded the time it took to run the models and generated the loss curves for all the models to identify how well the data learnt during training.

d. Data predictions

Predicted profit values from the different models were generated from the model using the test data as shown below and then attached to the test data file.

$$\hat{y}_i = \text{model.predict}(y_i) \quad (1)$$

$\hat{y}_i$ are the predicted profit values.

y_i are the AgriPoliS (actual) profit values.

The mean values for both the predicted profit and the actual profit values were plotted against each other to generate the comparison graph. The predicted profit values obtained from the models were also plotted against the AgriPoliS profit values using the scatter plots. In a scatter plot, the distance of a predicted value from a 45% angle line indicates how well or poorly the prediction performed.

e. Metrics and evaluations

To evaluate the different DNNs approaches, and based on the structure of the profit variable, we used Mean Absolute Error (MAE), Root Mean Squared error (RMSE) and the R^2 score as the metrics of choice. The MAE calculates the mean absolute average distance between the predicted profit and the actual values as shown below:

$$\mathrm{MAE} = \frac{1}{\mathrm{N}}\sum_{\mathrm{i}=1}^{\mathrm{N}}|\mathrm{y_i} - \hat{\mathrm{y}}_\mathrm{i}| \quad (2)$$

$\hat{\mathrm{y}}_\mathrm{i}$ *are the predicted profit values*.

$\mathrm{y_i}$ *are the AgriPoliS (actual) profit values*.

We also computed the Root Mean Squared Error (RMSE) which calculates the square root of the mean average distance between the predicted profit values and the actual profit values below:

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{\mathrm{i}=1}^{\mathrm{N}}(\mathrm{y_i}-\hat{\mathrm{y}}_\mathrm{i})^2}{\mathrm{n}}} \quad (3)$$

$\hat{y}_i$ are the predicted profit values.

y_i are the AgriPoliS (actual) profit values.

We also normalized the MAE and RMSE results after computing the MAE. Since the profit variable is a continuous variable and the range varies from (-239,000-491,879), it is necessary to normalize the results to enable comparison over the models. We chose to normalize by taking the difference between the maximum ($\mathrm{y_{max}}$) and the minimum values ($\mathrm{y_{min}}$) (Otto, 2019). The values lie between 0 and 1. Values closer to 0 indicate a better fit.

$$\mathrm{Norm_{MAE}} = \frac{\mathrm{MAE}}{(\mathrm{y_{max}}-\mathrm{y_{min}})} \quad (4)$$

$$\mathrm{Norm_{RMSE}} = \frac{\mathrm{RMSE}}{(\mathrm{y_{max}}-\mathrm{y_{min}})} \quad (5)$$

The R^2 score which is also known as the coefficient of determination was also utilized. It measures the variation explained by the relationship between the input features and the dependent variable. A score of 1 would indicate that the model was able to fully replicate the dependent variable, while a score of 0 would indicate a complete failure to replicate the dependent variable (Unterthiner et al., 2020).

$$\mathrm{R^2} = 1 - \frac{\sum_{\mathrm{i}=1}^{\mathrm{N}}(\hat{\mathrm{y}}_\mathrm{i}-\mathrm{y_i})^2}{\sum_{\mathrm{i}=1}^{\mathrm{N}}(\mathrm{y_i}-\bar{\mathrm{y}}_\mathrm{i})^2} \quad (6)$$

$\hat{y}_i$ are the predicted profit values.

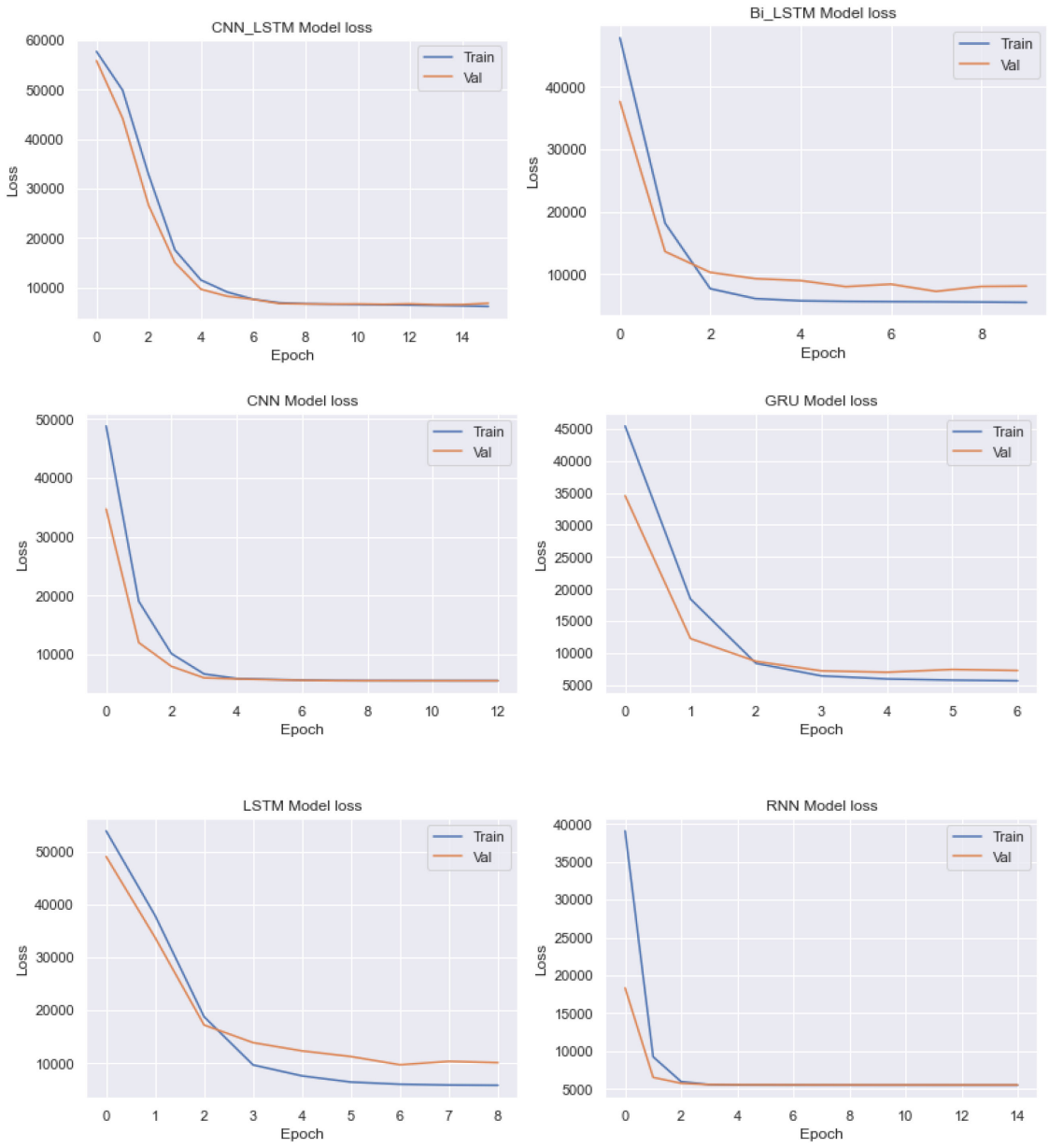
y_i are the actual profit values

$\bar{y}_i$ are the mean profit values.

4. Results

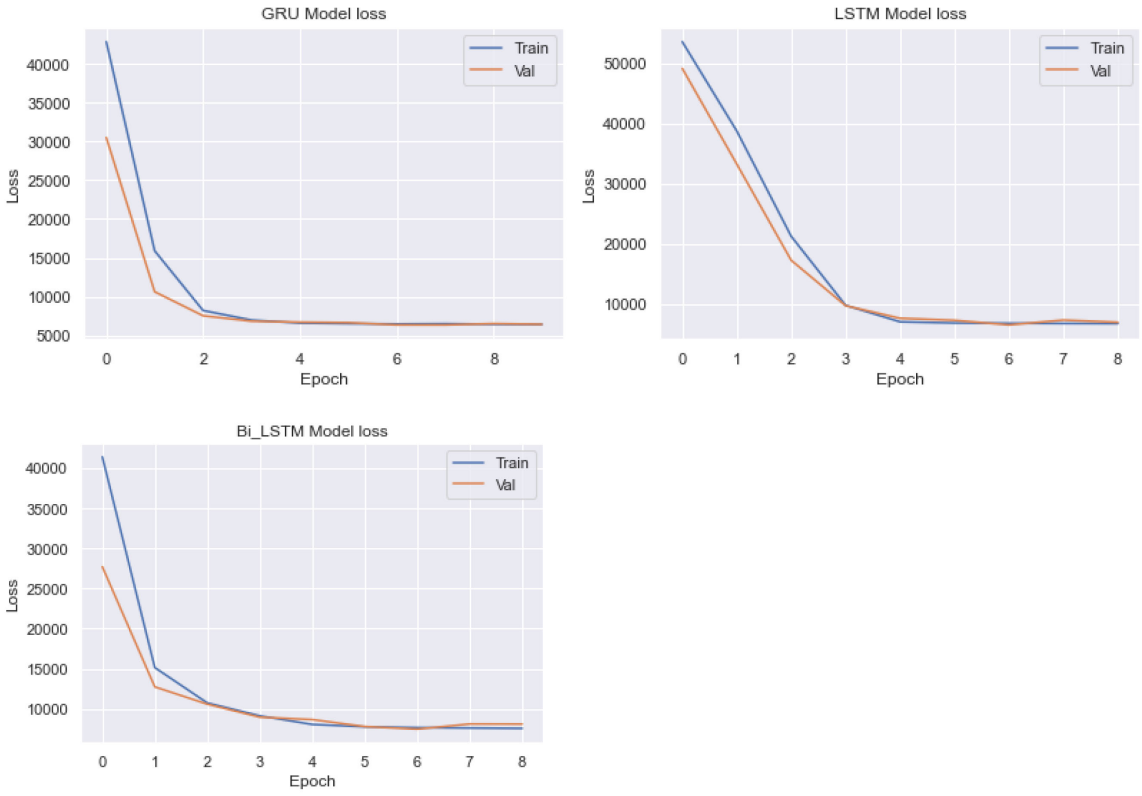
In this section, we present the results of the training the DNNs. Data generation with AgriPoliS roughly took around 6 hours. In total, 6 DNNs were successfully tested on their suitability and ability to emulate the output (profit) from AgriPoliS. The DNNs were assessed on their R^2 score, MAE, RMSE and runtimes. In Figure 5, we present the MAE loss curves for all the models. The loss curve is a good first indication of how well the model learnt by comparing the curves for both training and validation set over time.

Figure 5: Loss curves for all the DNN models



The CNN, CNN-LSTM and RNN seem to fit well on the validation data. On the contrary, the GRU slightly overfit the data while the LSTM and Bi-LSTM overfit on a higher magnitude. We repeated the simulations with the LSTM, Bi-LSTM and GRU and manually tested different dropout rates until the overfitting was reduced to a minimal level (Figure 6).

Figure 6: Loss curve using dropout to reduce overfitting



The next step was to compare the metrics for the different models. In Table 2, the DNNs were able to efficiently replicate the profit variable in AgriPoliS with a high degree of precisions as shown by the R^2 values of more than 90%. A comparison of the metrics shows that the CNN and the CNN-LSTM were the best performing models. CNN-LSTM was the best performance in terms of R^2 with a score of 91.4% and the lowest RMSE value (23,409.03). The CNN performed well based on the MAE value (5,482.34) but had the 2nd lowest R^2 score. In terms of computational time, the GRU was the fastest model followed closely by the CNN. Although dropout reduced overfitting, it came at the cost of longer running time as observed by comparing the models before after applying drop out. It was also quite expensive in computation due to manually testing different dropout values. We ran the models on CPU, however running the models on GPU should also result in stupendous reduction in run time.

Table 2: Comparisons of metrics from different DNNs architectures

Model	**Bi-LSTM**	**Bi-LSTM**	**CNN**	**CNN-LSTM**	**GRU**	**GRU**	**LSTM**	**LSTM**	**Simple RNN**
Dropout	**-**	**0.04**	**-**	**-**	**-**	**0.015**	**-**	**0.02**	**-**
R^2	0.913	0.912	0.903	**0.914**	0.911	0.908	0.90	0.91	0.904
MAE	7803.91	7991.45	**5482.34**	6980.44	6398.20	6520.83	8351.70	7306.105	5525.33
Norm_MAE	0.011	0.011	**0.007**	0.010	0.010	0.009	0.011	0.010	0.008
RMSE	23579.12	23798.22	24876.41	**23409.39**	23878.63	24258.31	23877.25	24116.59	24799.10
Norm_RMSE	0.032	0.032	0.034	**0.032**	0.033	0.022	0.033	0.033	0.034
Runtime	580s	836S	293s	324s	**248s**	533s	406s	446s	355s

We also supplemented the results with visual displays comparing profit predictions to the AgriPoliS values to check how the data fit through the iterations. Due to the large number of farms, Figure 7, Figure 8 and Figure 9 only present the comparison of the average profit predictions to the average AgriPoliS values for the CNN, LSTM and LSTM with dropout.

Figure 7: Comparison between the average predicted values and the average AgriPoliS values for CNN model

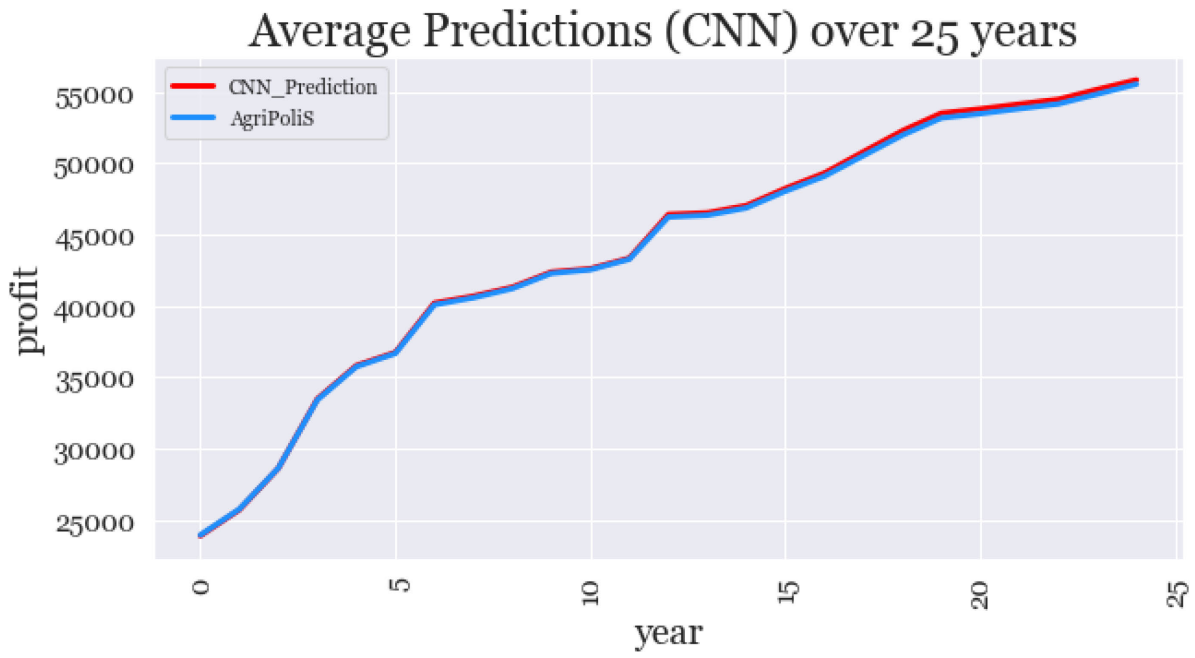


Figure 8: Comparison between the average predicted values and the average AgriPoliS values for LSTM model

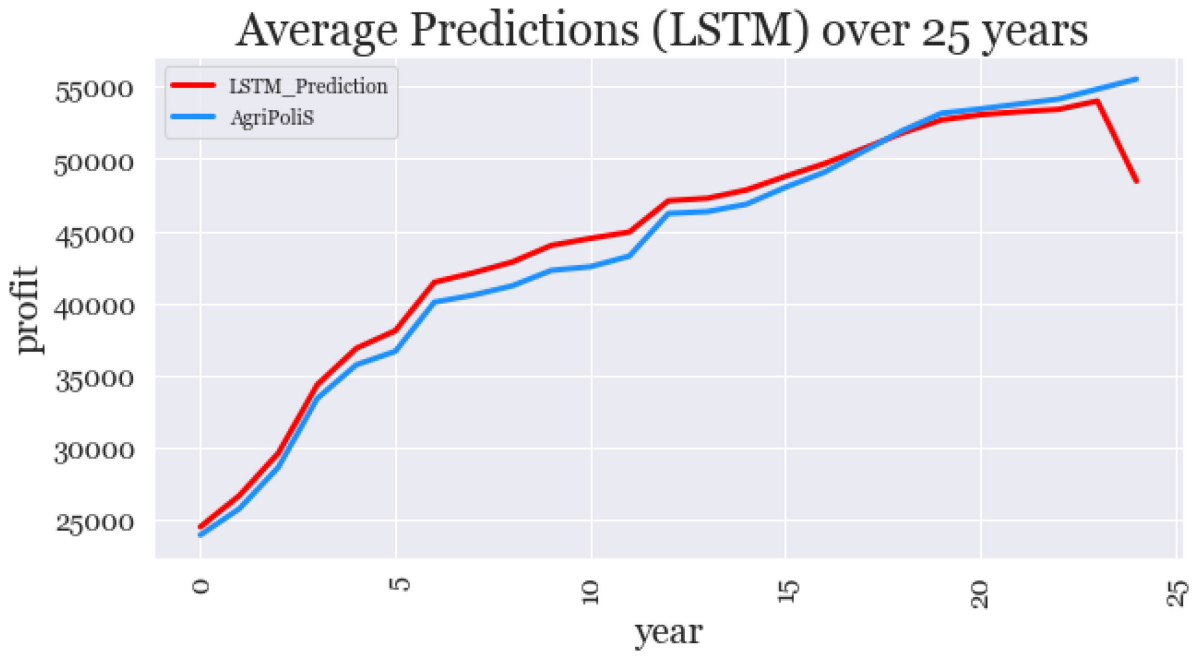
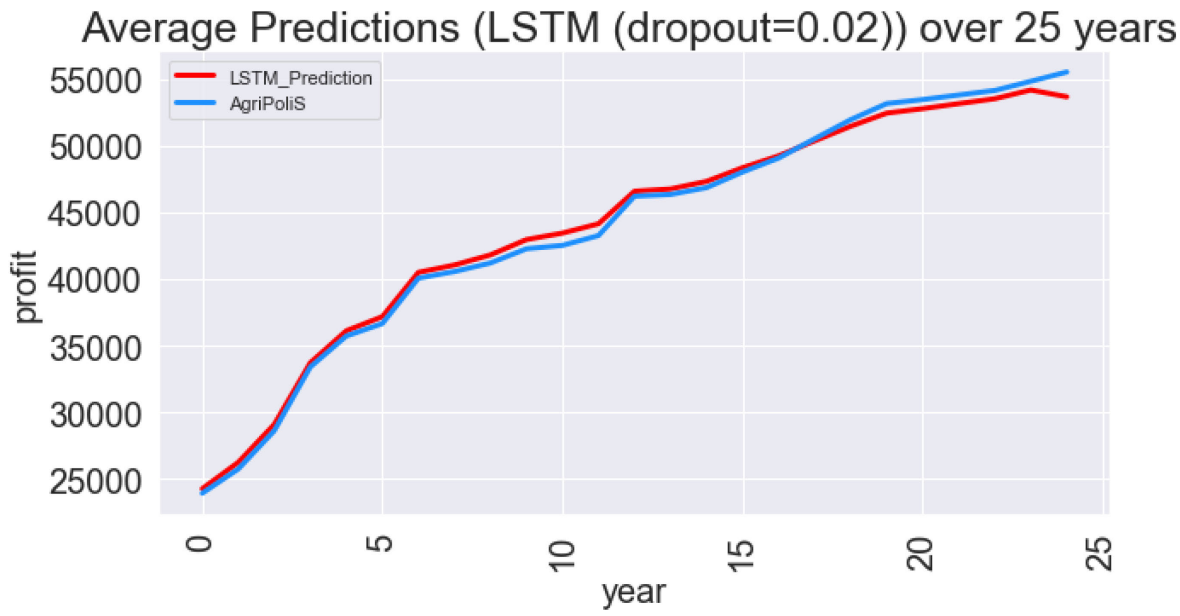


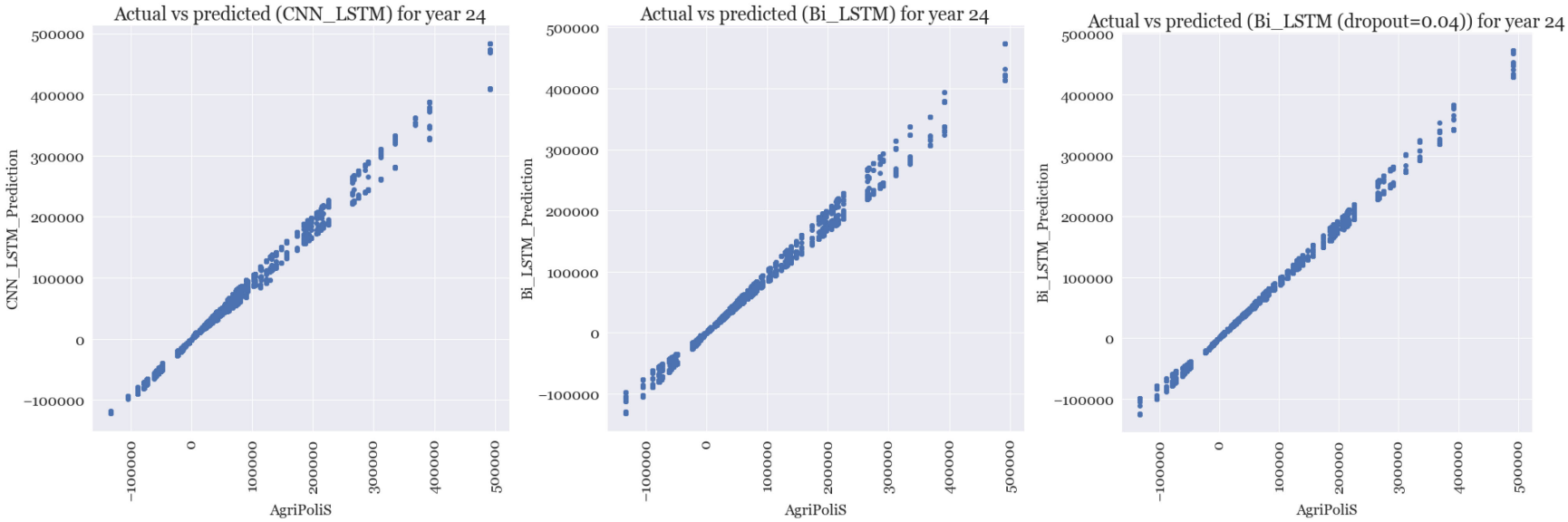
Figure 9: Comparison between the average predicted values and the average AgriPoliS values for LSTM model with dropout

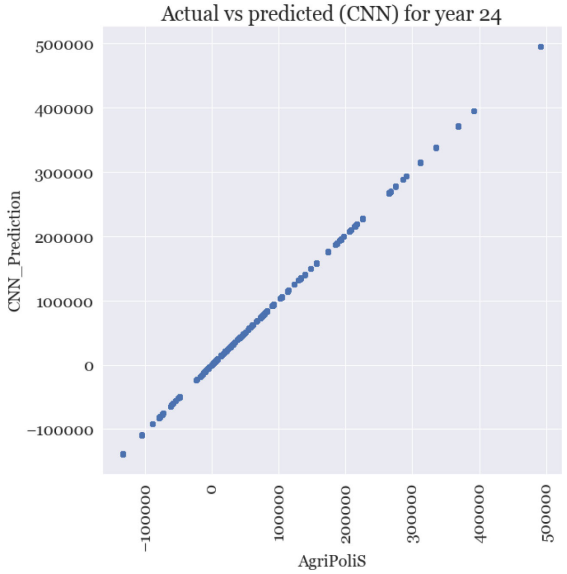


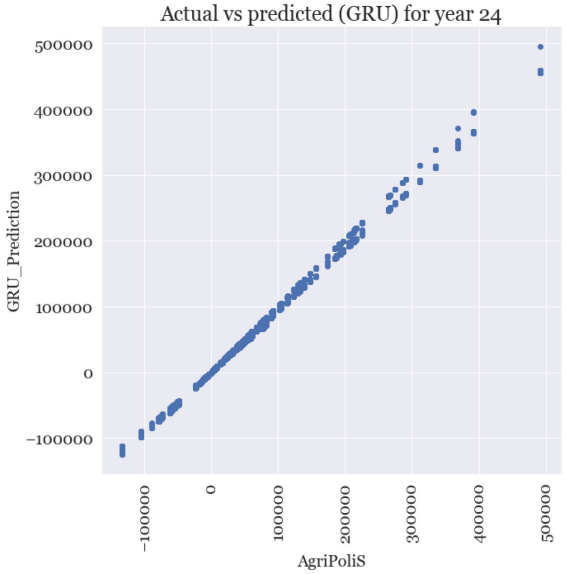
The average prediction in the CNN models were a close approximation to the AgriPoliS values. There is a clear disparity between the predictions and AgriPoliS values for the LSTM over time. The average predictions in the LSTM model fluctuated over time. From year 1 -15 the predicted values were higher than the actual values and were lower for the remainder of the year. This demonstrated that the LSTM was overfitting. This is further reinforced by looking at the loss curve for the LSTM in Figure 4. This shows that the LSTM was not a good fit for the data. For the GRU, CNN-LSTM and Bi-LSTM, the predicted values were lower in the last iteration (iteration 24) and were navigating towards zero. Based on these charts, it appears that the CNN was the best fit because it gave quite consistent results from iteration to iteration.

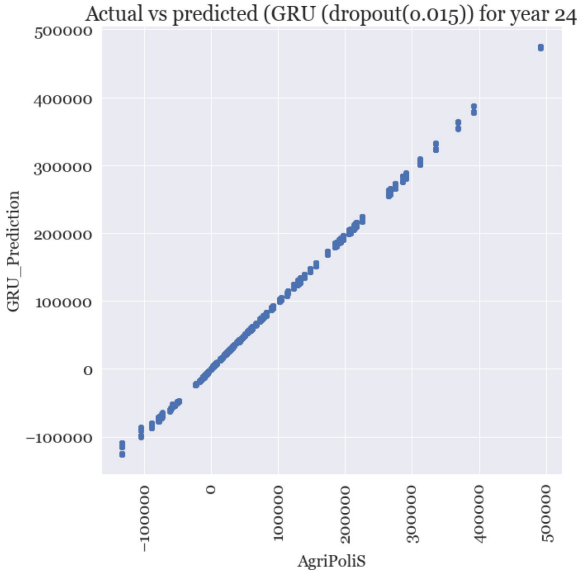
We also compared scatter plots between the predicted values and the AgriPoliS values at each iteration. In this section, we present only the scatter plots for iteration 24 (Figure 10). Additional scatter plots for iteration 0 and iteration 10 are presented in the Appendix. A look at the scatter plot shows that the CNN and the RNN models were able to predict the data well while the rest of the models did not predict the data well.

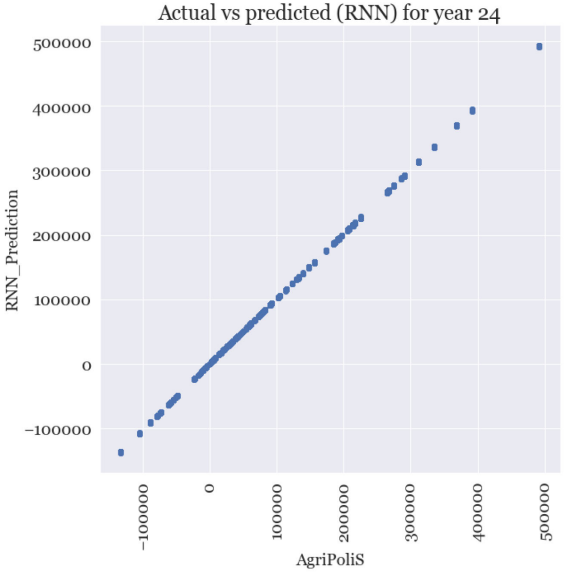
Figure 10: Scatter plots showing the predicted values (y-axis) versus AgriPoliS values (x-axis) at iteration (year 24)

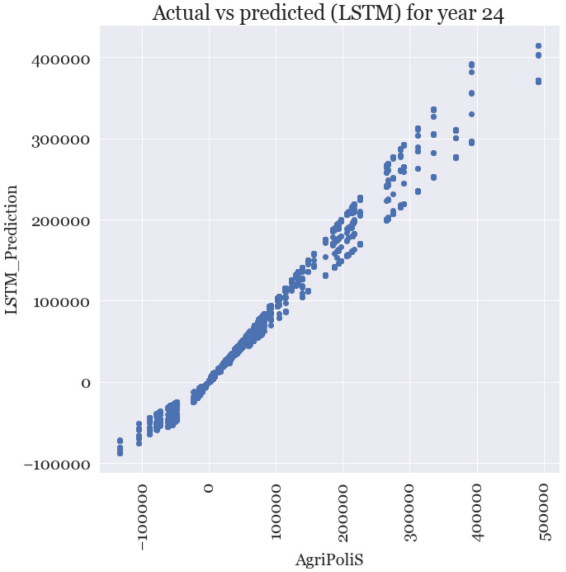


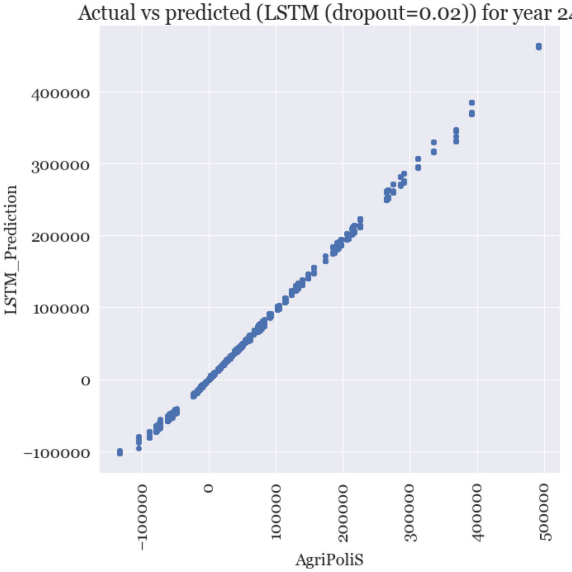

Actual vs predicted (CNN) for year 24
CNN_Prediction
AgriPoliS


Actual vs predicted (GRU) for year 24
GRU_Prediction
AgriPoliS


Actual vs predicted (GRU (dropout(0.015)) for year 24
GRU_Prediction
AgriPoliS


Actual vs predicted (RNN) for year 24
RNN_Prediction
AgriPoliS


Actual vs predicted (LSTM) for year 24
LSTM_Prediction
AgriPoliS


Actual vs predicted (LSTM (dropout=0.02)) for year 24
LSTM_Prediction
AgriPoliS

a) Adaptation of CNN for Multi-Output predictions

The earlier parts focused on only a single variable to replicate. It is possible to replicate more than one variable with minimal changes to the code. Hyper parameter tuning would also need to be carried out to fit the best model fit for the data. In this section we adapted the code to replicate 3 variables: profit, change in equity and revenue. Figure 11, Figure 12 and Figure 13 show the predicted and actual values for the 3 variables (profit, income and revenue).

Figure 11: Comparison between the average predicted profit values and the average AgriPoliS profit values (Multioutput)

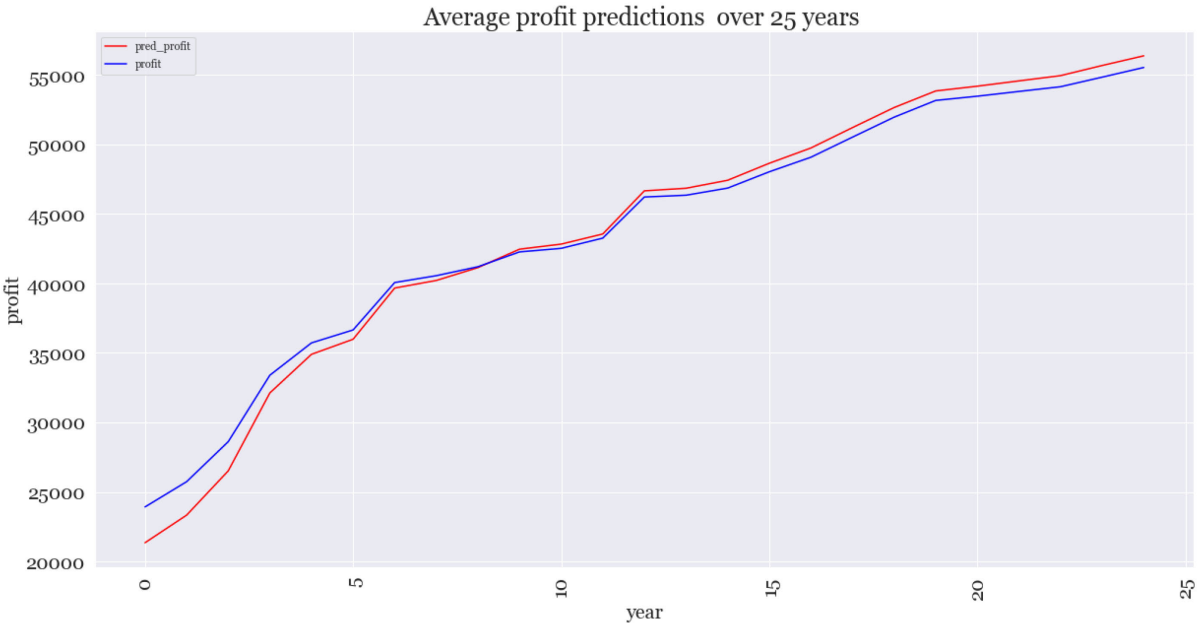


Figure 12: Comparison between the average predicted income values and the average AgriPoliS income values (Multioutput).

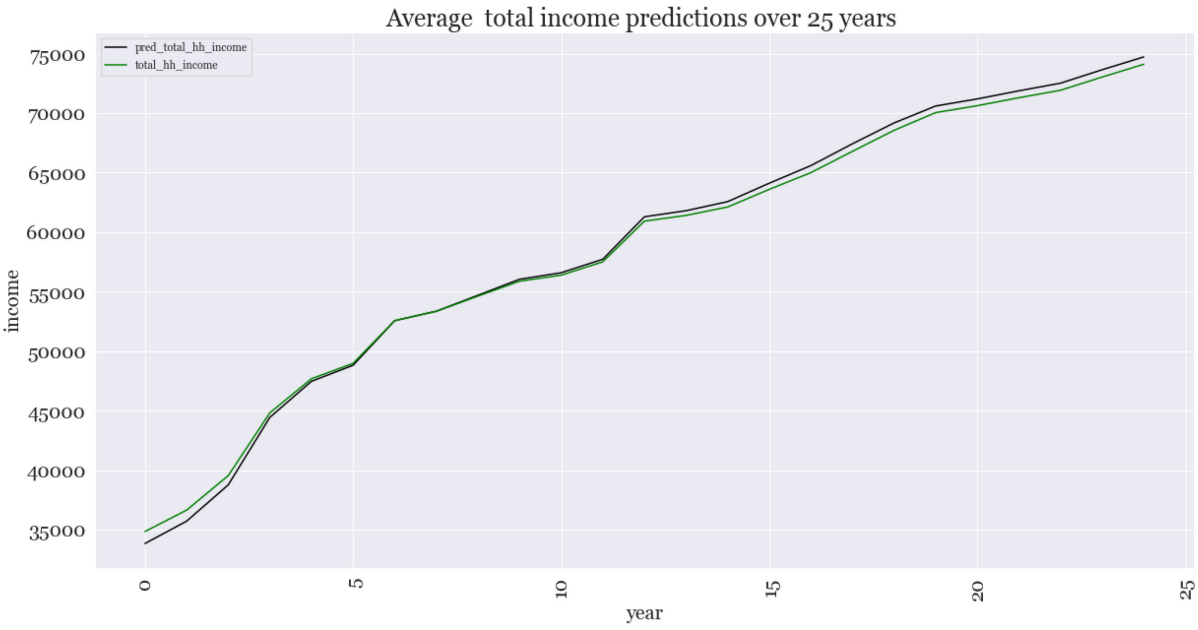
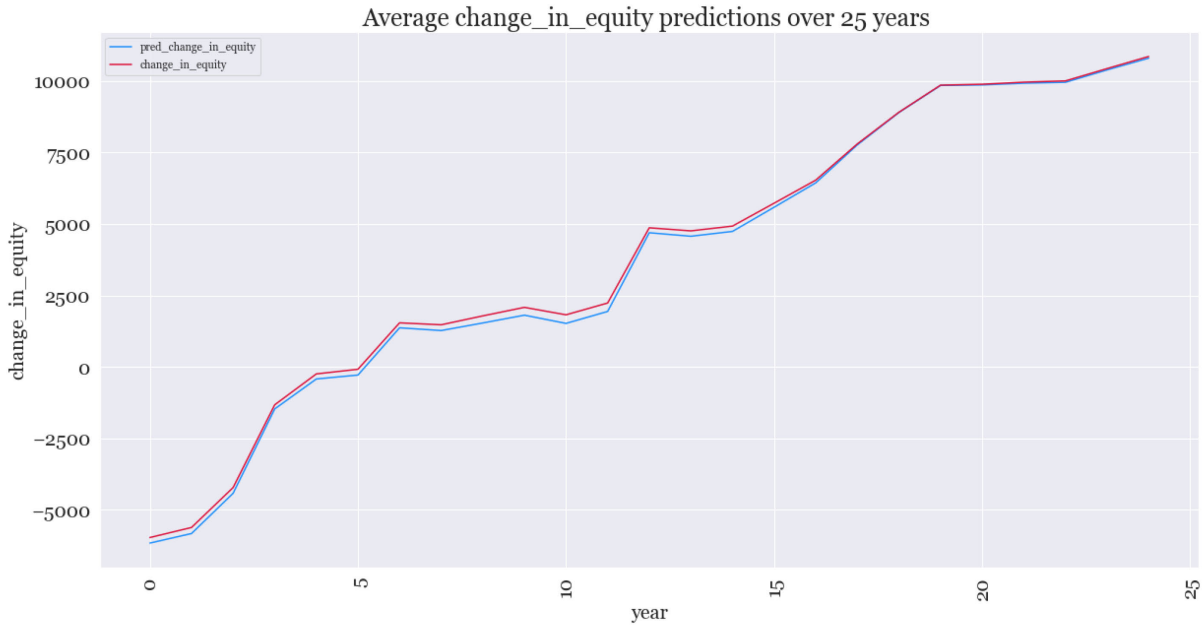


Figure 13: Comparison between the average predicted change in equity values and the average AgriPoliS change in equity values (Multioutput).



5. Summary and Discussion

In this paper, we explored and tested different DNNs architectures on their ability to replicate the ABM. The work presented in the paper is limited to simulated data generated from AgriPoliS and only focused on DNNs which have had the most promise in the use of cross-sectional time series data. We focused majorly it to one single variable of interest i.e. profit. We also demonstrated it is possible for multiple variables by extending it to three variables of interest. This could form the basis for future work e.g. in generating surrogate models.

From the results, we can conclude that DNNs models were able to replicate the AgriPoliS data with a high degree of accuracy as seen in the coefficient of determination score. A comprehensive comparison of the MAE, R2 score and the visual displays indicates that the CNN was the most suited for replicating the data. The mean profit predictions were also quite close to the mean profit values at every iteration as compared to the other models. For the GRU, CNN-LSTM and Bi-LSTM, the predicted values were lower in the last iteration (iteration 24) and were navigating towards zero. Based on Abduljabbar et al. (2021) and Plaster and Kumar (2019), the LSTM and its variants work better with longer predictive time horizon than shorter predictive horizon. This might be a plausible explanation why the predictions were lower in the LSTM, CNN-LSTM and Bi-LSTM.

Although the results focused more on the output, it is noteworthy that hyperparameter optimization took a considerable amount of work. Hyperparameters such as the learning rate, number of layers, batch size and dropout rate had to be manually tested to find the best fit as there is not a single ‘one fits all’ scenario. The computation time required to run the model was significantly lower than the computational time to run the ABM while using the CPU. However, the computation might be biased since the ABM and the DNN use different programming languages. In future work, we envision to use a GPU which would considerably improve running time.

6. Conclusion and Outlook

The work presented in this paper followed the “doppelgänger” approach by van der Hoog (2017). We generated training data from AgriPoliS and tested the ability of different DNNs to emulate the behaviour of standard AgriPoliS agents. Although the DNNs could mimic the behaviour of the agents, they were not equipped in capturing behaviour of strategic relevance such as focusing on the long-run performance of the farm instead of short-term profit-maximizing behaviour, strategic interactions with neighbouring farms and capability to learn and adapt to counter changes in the environment that would lead to increased efficiency, resilience against shocks and increased competitiveness. Nevertheless, this approach provided profound insights to guide our subsequent steps to model strategic behaviour of agents in AgriPoliS.

To that effect, we will implement Deep Reinforcement Learning (DRL) which uses DNNs in their architecture to enable formulation of strategic bids in the land markets to maximize the farm agents’ long-term profitability. DRL agents would interact with the AgriPoliS environment directly through a framework of states. actions, transitions and rewards. The training process for the DRL refines the agents’ policies through the repeated interactions with the environment. The actions of the DRL agents would also impact other agents in the region through their competition in land markets. This necessitates an interface where the DRL agents’ environment interacts with the AgriPoliS environment, thereby enabling concurrent competition in the land markets. This has an implication in that we are not able to use custom made DRL environments such as gym and thus have to design our environment which interlinked the DRL and AgriPoliS. The DRL would feed strategic bids to AgriPoliS and the results of the bid after investments, production and farm accounting has taken place would be transmitted back to the DRL agents to modify behaviour and then transmitted back to AgriPoliS thus capturing strategic behaviour and highlighting the role of strategic interaction between farms

References

Abduljabbar, R. L., Dia, H., & Tsai, P.-W. (2021). Development and evaluation of bidirectional LSTM freeway traffic forecasting models using simulation data. Scientific Reports, 11(1). https://doi.org/10.1038/s41598-021-03282-z.

Aggarwal, C. C. (2018). Neural networks and deep learning. Springer, 10, 978-973.

An, L. (2012). Modeling human decisions in coupled human and natural systems: Review of agent-based models. Ecological modelling, 229, 25-36. https://doi.org/10.1016/j.ecolmodel.2011.07.010.

Angione, C., Silverman, E., & Yaneske, E. (2020). Using Machine Learning to Emulate Agent-Based Simulations. arXiv:2005.02077. https://doi.org/10.48550/arXiv.2005.02077.

Augustijn, E.-W., Kounadi, O., Kuznecova, T., & Zurita-Milla, R. (2019). Teaching Agent-Based Modelling and Machine Learning in an integrated way. GeoComputation 2019. https://auckland.figshare.com/articles/Teaching_Agent-Based_Modelling_and_Machine_Learning_in_an_integrated_way/9848804

Balmann, A. (1997). Farm-based modelling of regional structural change: A cellular automata approach. European Review of Agricultural Economics, 24(1), 85-108. https://doi.org/10.1093/erae/24.1.85.

Bonabeau, E. (2002). Agent-based modeling: Methods and techniques for simulating human systems. Proceedings of the national academy of sciences, 99(suppl 3), 7280-7287. https://doi.org/10.1073/pnas.082080899.

Brearcliffe, D. K., & Crooks, A. (2021). Creating Intelligent Agents: Combining Agent-Based Modeling with Machine Learning. Proceedings of the 2020 Conference of The Computational Social Science Society of the Americas, Cham. https://doi.org/10.1007/978-3-030-83418-0_3.

Brownlee, J. (2016). Deep learning with Python: develop deep learning models on Theano and TensorFlow using Keras. Machine Learning Mastery.

Brownlee, J. (2017). How to convert a time series to a supervised learning problem in Python. Retrieved 23/05/2022 from https://machinelearningmastery.com/convert-time-series-supervised-learning-problem-python/.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078. https://doi.org/10.48550/arXiv.1406.1078.

Chollet, F. (2021). Deep learning with Python. Simon and Schuster.

Crooks, A. T., & Heppenstall, A. J. (2012). Introduction to Agent-Based Modelling. In A. J. Heppenstall, A. T. Crooks, L. M. See, & M. Batty (Eds.), Agent-Based Models of Geographical Systems (pp. 85-105). Springer Netherlands. https://doi.org/10.1007/978-90-481-8927-4_5.

Epstein, J. M. (1999). Agent-based computational models and generative social science. Complexity, 4(5), 41-60. https://doi.org/10.1002/(SICI)1099-0526(199905/06)4:5%3C41::AID-CPLX9%3E3.0.CO;2-F.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.

Happe, K. (2004). Agricultural policies and farm structures - Agent-based modelling and application to EU-policy reform Leibniz Institute of Agricultural Development in Transition Economies (IAMO), Halle (Saale)]. http://ageconsearch.umn.edu/record/14945/files/st040030.pdf.

Happe, K., Kellermann, K., & Balmann, A. (2006). Agent-based analysis of agricultural policies: an illustration of the agricultural policy simulator AgriPoliS, its adaptation and behavior. Ecology and society, 11(1). https://www.jstor.org/stable/26267800.

Heppenstall, A. J., Crooks, A. T., See, L. M., & Batty, M. (2011). Agent-based models of geographical systems. Springer Science & Business Media. http://dx.doi.org/10.22004/ag.econ.120248.

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation, 9(8), 1735-1780. https://doi.org/10.1162/neco.1997.9.8.1735.

Hua, Y., Zhao, Z., Li, R., Chen, X., Liu, Z., & Zhang, H. (2019). Deep learning with long short-term memory for time series prediction. IEEE Communications Magazine, 57(6), 114-119. https://doi.org/10.1109/MCOM.2019.1800155.

Jin, X., Yu, X., Wang, X., Bai, Y., Su, T., & Kong, J. (2020). Prediction for Time Series with CNN and LSTM. In R. Wang, Z. Chen, W. Zhang, & Q. Zhu, Proceedings of the 11th International Conference on Modelling, Identification and Control (ICMIC2019) Singapore. https://doi.org/10.1007/978-981-15-0474-7_59.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980. https://doi.org/10.48550/arXiv.1412.6980.

Kremmydas, D., Athanasiadis, I. N., & Rozakis, S. (2018). A review of agent based modeling for agricultural policy evaluation. Agricultural Systems, 164, 95-106. https://doi.org/10.1016/j.agsy.2018.03.010.

Lamperti, F., Roventini, A., & Sani, A. (2018). Agent-based model calibration using machine learning surrogates. Journal of Economic Dynamics and Control, 90, 366-389. https://doi.org/10.1016/j.jedc.2018.03.011.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444. https://doi.org/10.1038/nature14539.

Lobianco, A., & Esposti, R. (2010). The Regional Multi-Agent Simulator (RegMAS): An open-source spatially explicit model to assess the impact of agricultural policies. Computers and electronics in agriculture, 72(1), 14-26. https://doi.org/10.1016/j.compag.2010.02.006.

Macal, C. M., & North, M. J. (2005). Tutorial on agent-based modeling and simulation. Proceedings of the Winter Simulation Conference, 2005., https://doi.org/10.1109/WSC.2005.1574234.

Murray-Rust, D., Robinson, D. T., Guillem, E., Karali, E., & Rounsevell, M. (2014). An open framework for agent based modelling of agricultural land use change. Environmental Modelling & Software, 61, 19-38. https://doi.org/10.1016/j.envsoft.2014.06.027.

Otto, S. A. (2019). How to normalize the RMSE [Blog post]. https://www.marinedatascience.co/blog/2019/01/07/normalizing-the-rmse/#.

Plaster, B., & Kumar, G. (2019). Data-driven predictive modeling of neuronal dynamics using long short-term memory. Algorithms, 12(10), 203. https://doi.org/10.3390/a12100203.

Railsback, S. F., & Grimm, V. (2019). Agent-based and individual-based modeling: a practical introduction. Princeton university press.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323(6088), 533-536. https://doi.org/10.1038/323533a0.

Sahrbacher, C., & Happe, K. (2008). A methodology to adapt AgriPoliS to a region. http://www.agripolis.de/documentation/adaptation_v1.pdf.

Sahrbacher, C., Sahrbacher, A., Kellermann, K., Happe, K., Balmann, A., Brady, M., Schnicke, H., Ostermeyer, A., Schönau, F., & Dong, C. (2012). AgriPoliS: An ODD-Protocol. http://www.iamo.de/agripolis/documentation/ODD_AgriPoliS.pdf.

Schreinemachers, P., & Berger, T. (2011). An agent-based simulation model of human–environment interactions in agricultural systems. Environmental Modelling & Software, 26(7), 845-859. https://doi.org/10.1016/j.envsoft.2011.02.004.

Shi, H. (2020). COVID-19 Global Data -Time Series Panel Data with LSTM. https://melaniesoek0120.medium.com/covid-19-global-data-time-series-prediction-with-lstm-recurrent-neural-networks-f7825c4a1f6f.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), 1929-1958. https://www.scopus.com/inward/record.uri?eid=2-s2.0-84904163933&partnerID=40&md5=b865fd654b3befc5d829dbe5d42b80c3.

Unterthiner, T., Keysers, D., Gelly, S., Bousquet, O., & Tolstikhin, I. (2020). Predicting neural network accuracy from weights. arXiv preprint arXiv:2002.11448. https://doi.org/10.48550/arXiv.2002.11448.

van der Hoog, S. (2017). Deep learning in (and of) agent-based models: A prospectus. arXiv preprint arXiv:1706.06302. https://doi.org/10.48550/arXiv.1706.06302.

Vasilev, I., Slater, D., Spacagna, G., Roelants, P., & Zocca, V. (2019). Python Deep Learning: Exploring deep learning techniques and neural network architectures with Pytorch, Keras, and TensorFlow. Packt Publishing Ltd.

Yamak, P. T., Yujian, L., & Gadosey, P. K. (2019). A comparison between arima, lstm, and gru for time series forecasting. Proceedings of the 2019 2nd International Conference on Algorithms, Computing and Artificial Intelligence, https://doi.org/10.1145/3377713.3377722.

Appendix

Figure 14: Predicted (y-axis) versus actual (x-axis) for LSTM

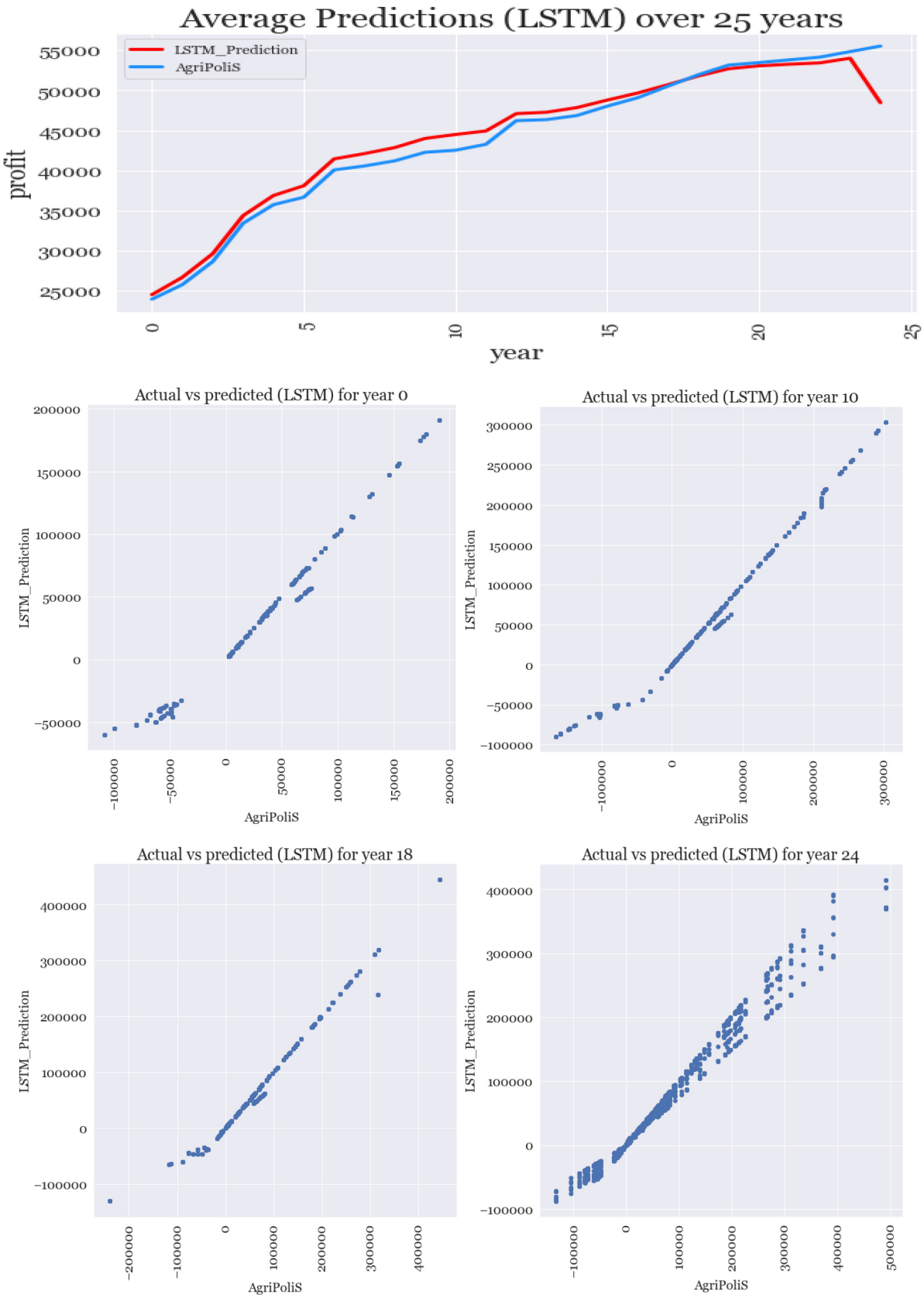


Figure 15: Predicted (y-axis) versus actual (x-axis) for LSTM (dropout=0.02)

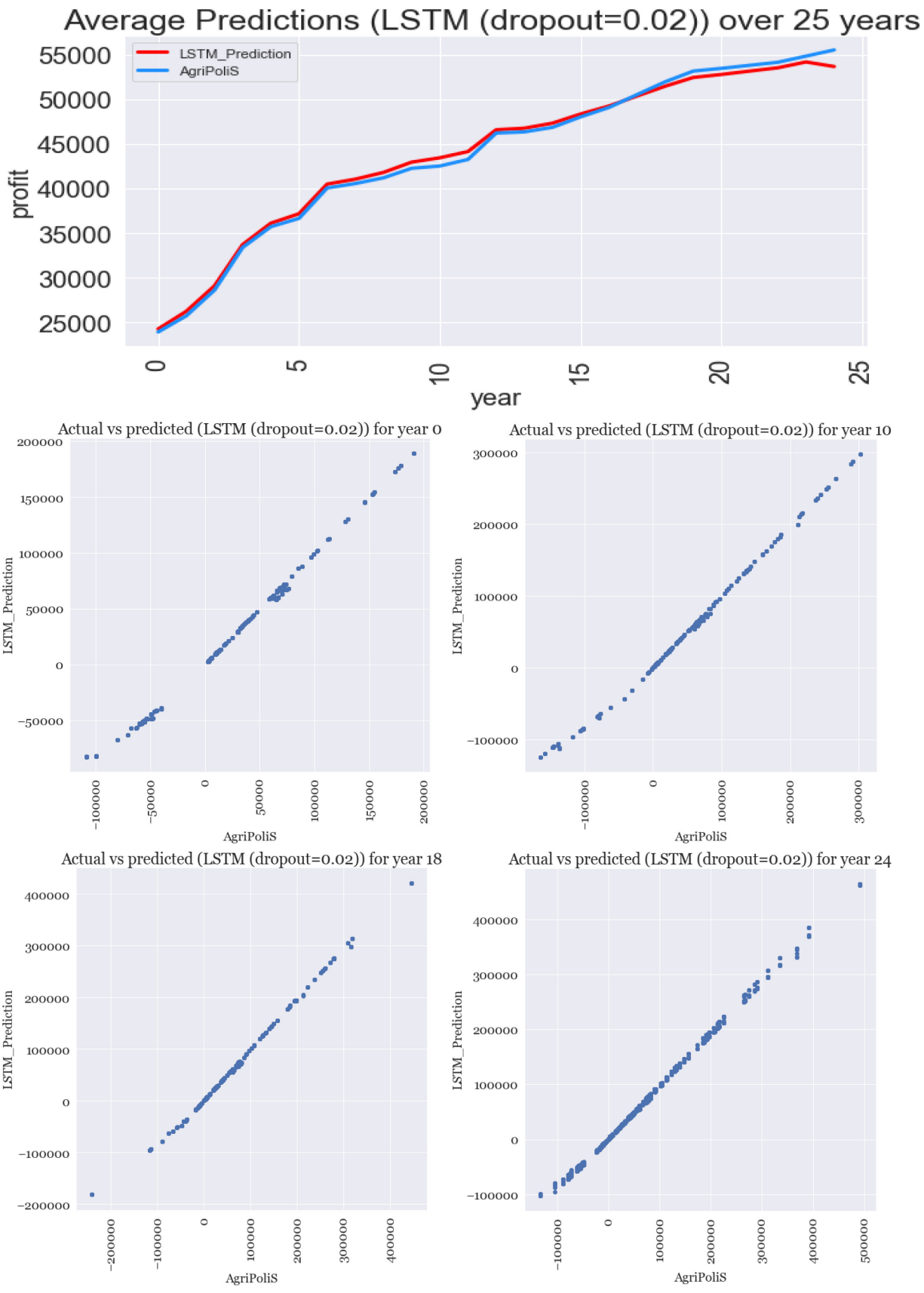


Figure 16: Predicted (y-axis) versus actual (x-axis) for Bi-LSTM

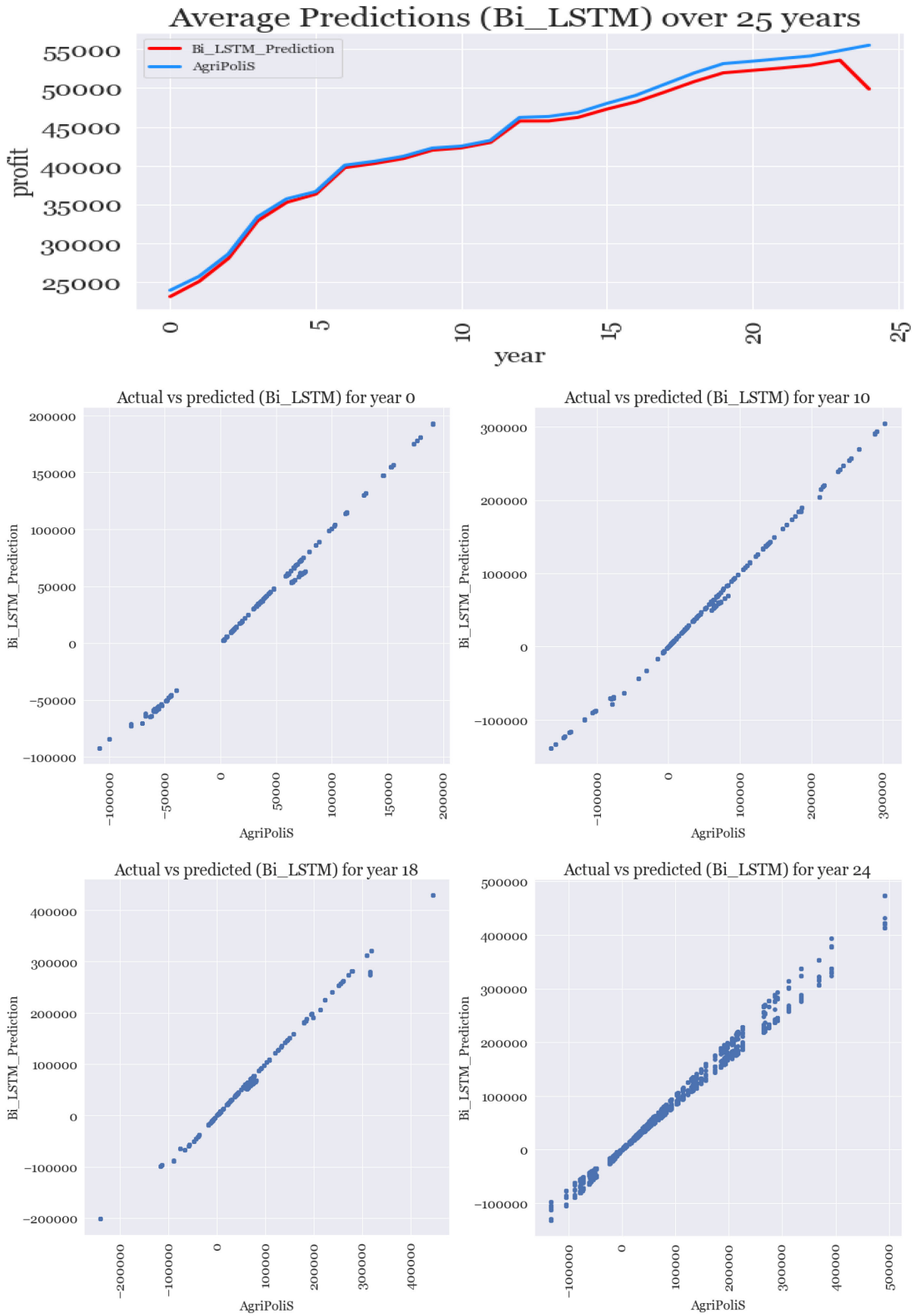


Figure 17: Predicted (y-axis) versus actual (x-axis) for Bi-LSTM((dropout=0.04)

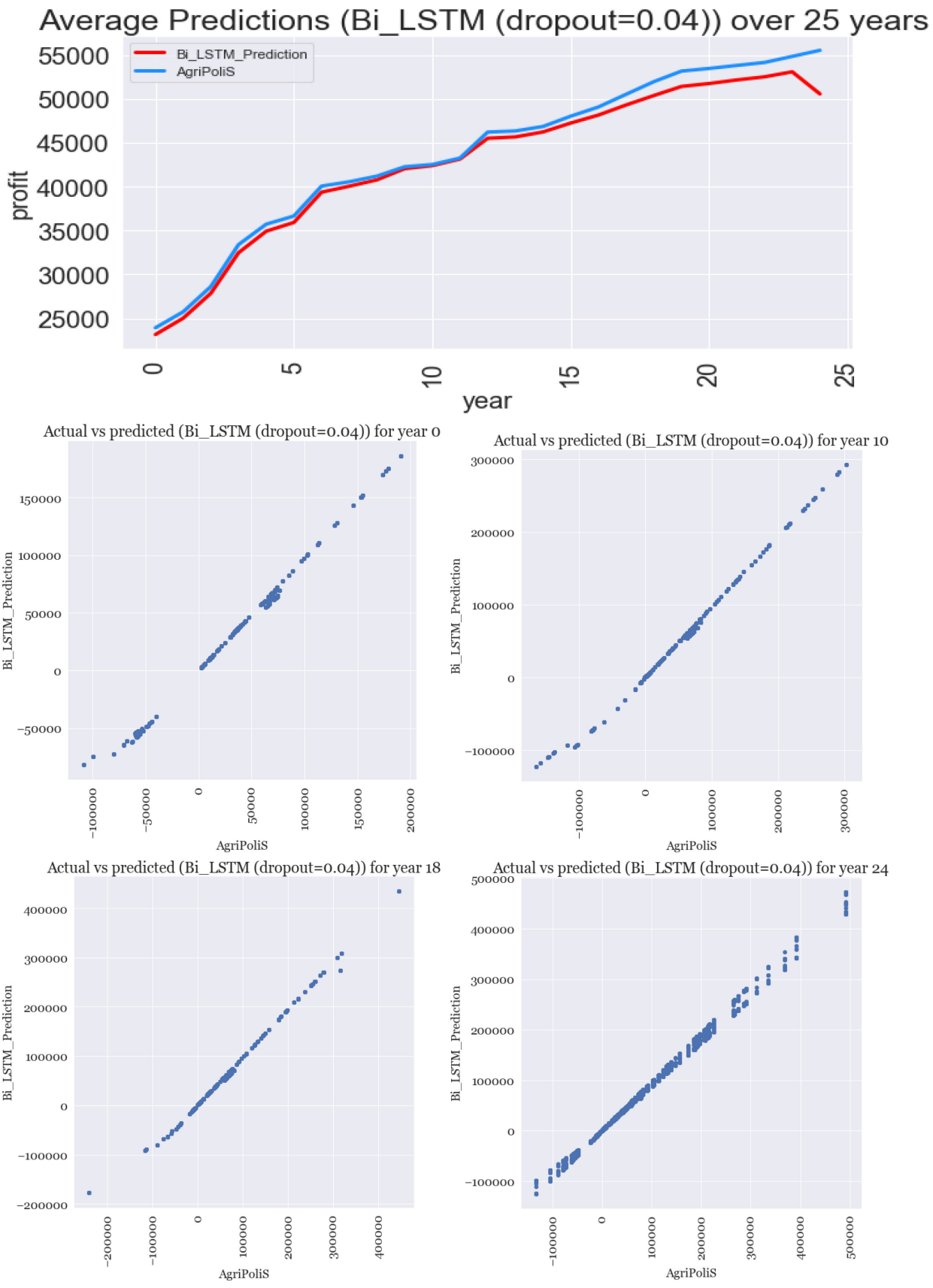


Figure 18 Predicted (y-axis) versus actual (x-axis) for CNN-LSTM

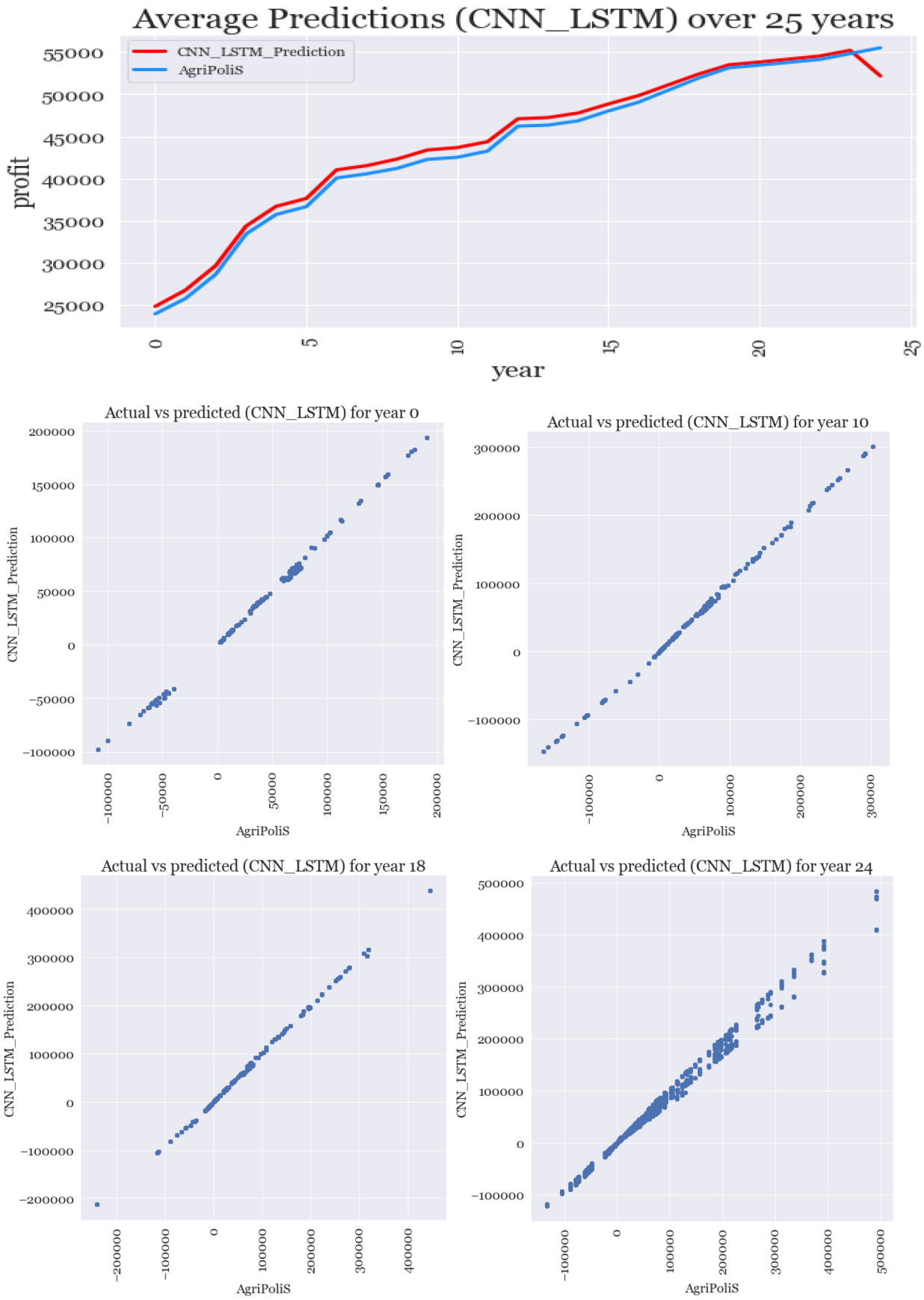


Figure 19: Predicted (y-axis) versus actual (x-axis) forCNN

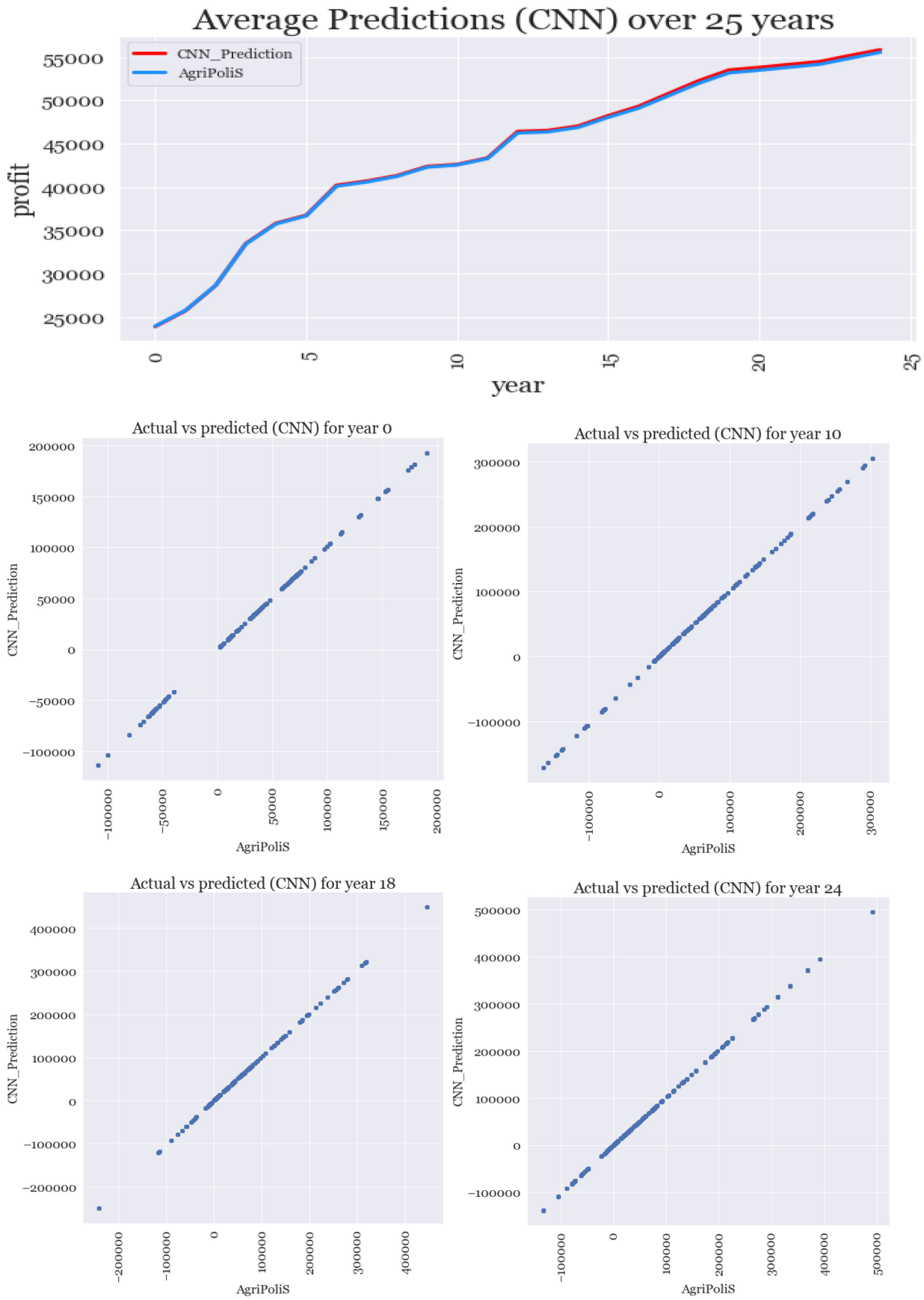


Figure 20: Predicted (y-axis) versus actual (x-axis) for GRU

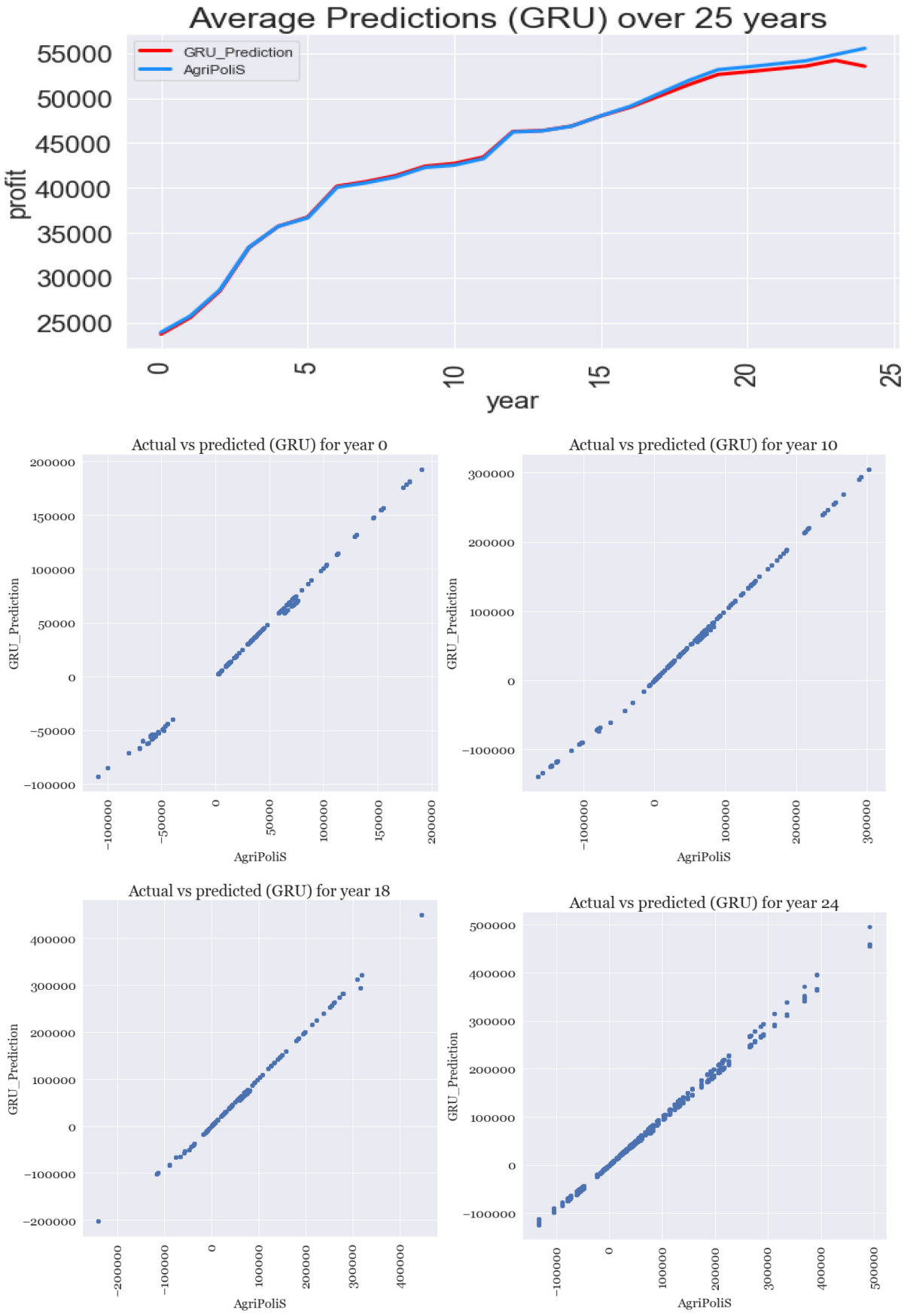


Figure 21: Predicted (y-axis) versus actual (x-axis) for GRU(dropout (0.015)

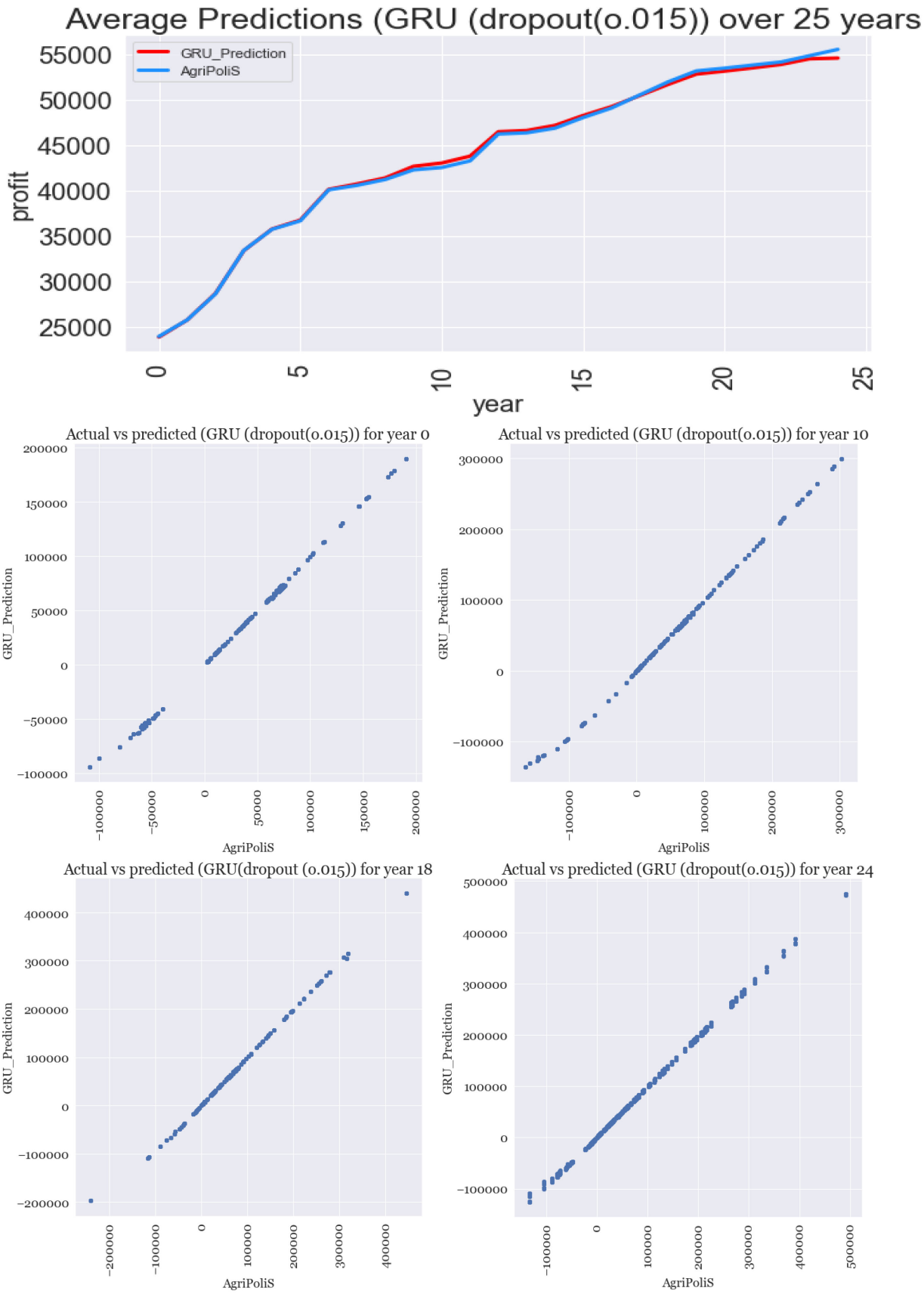


Figure 22 Predicted (y-axis) versus actual (x-axis) for RNN

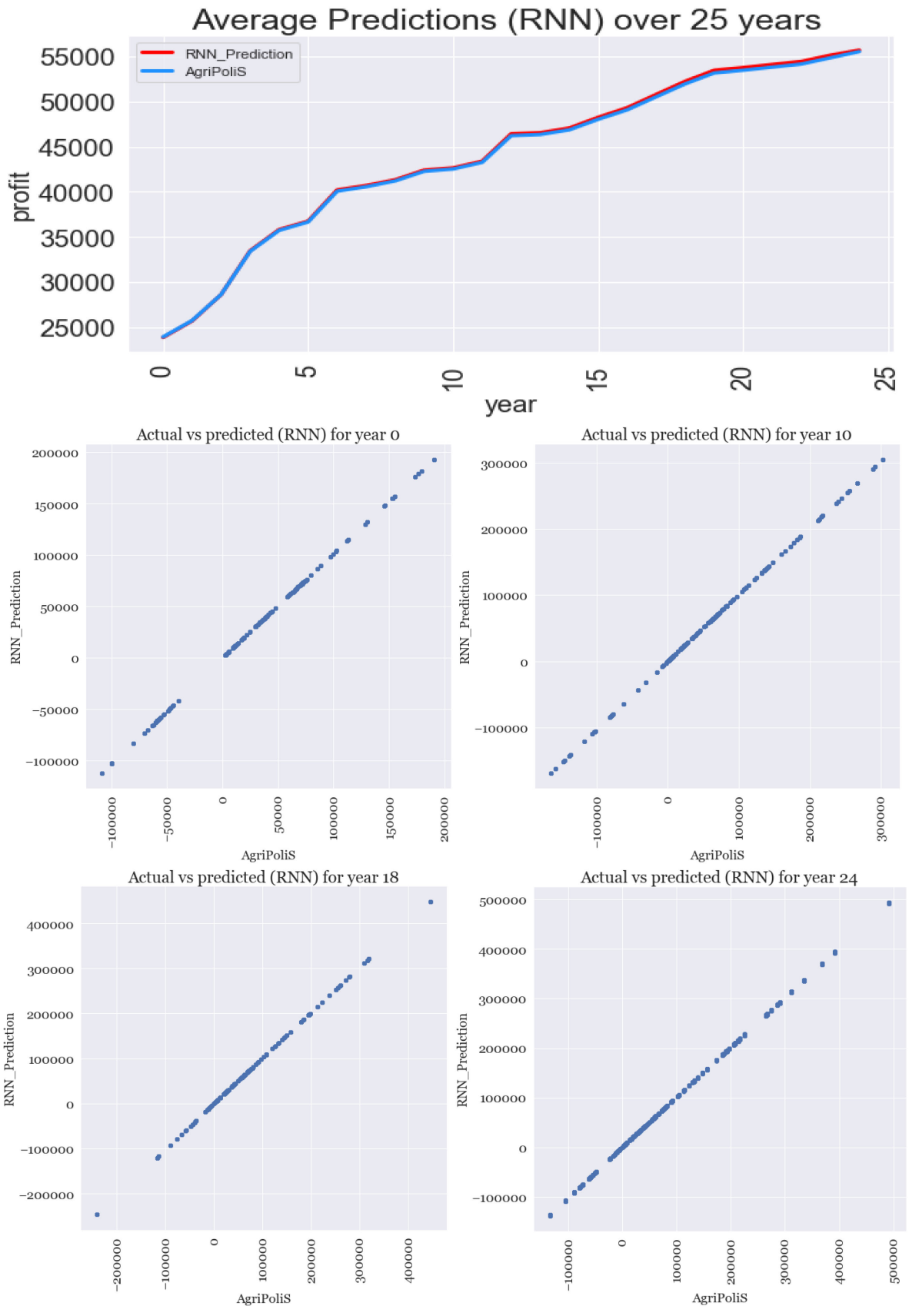
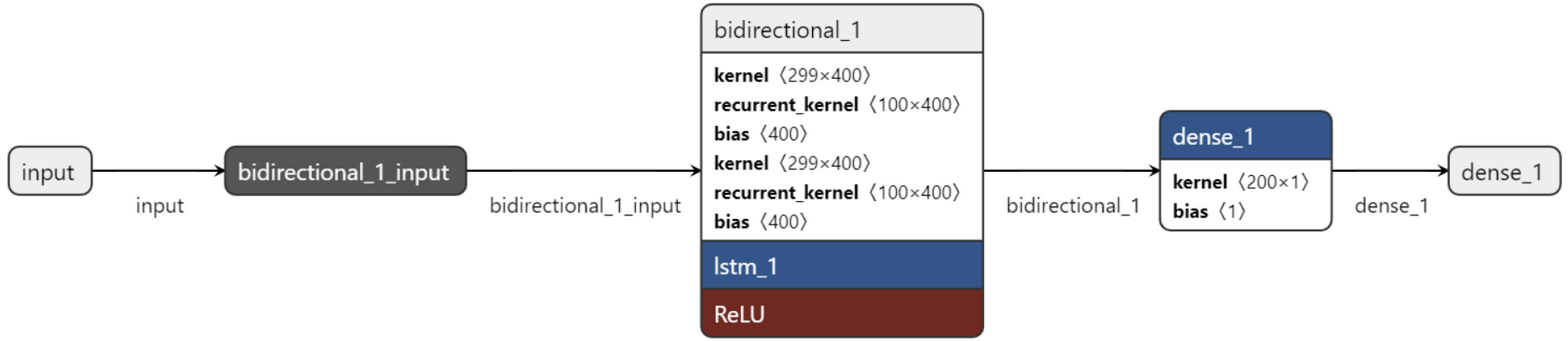
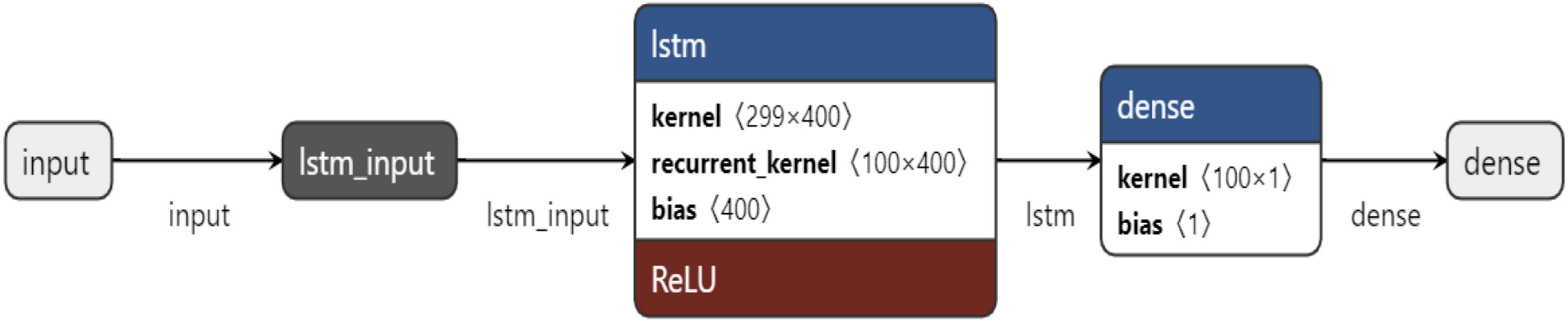


Figure 23: DNN models' architecture

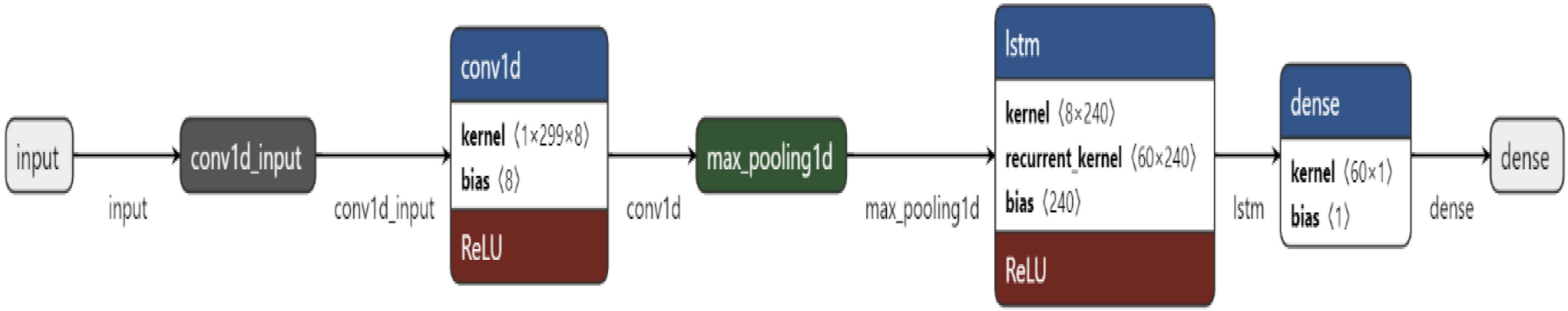
Bi-LSTM



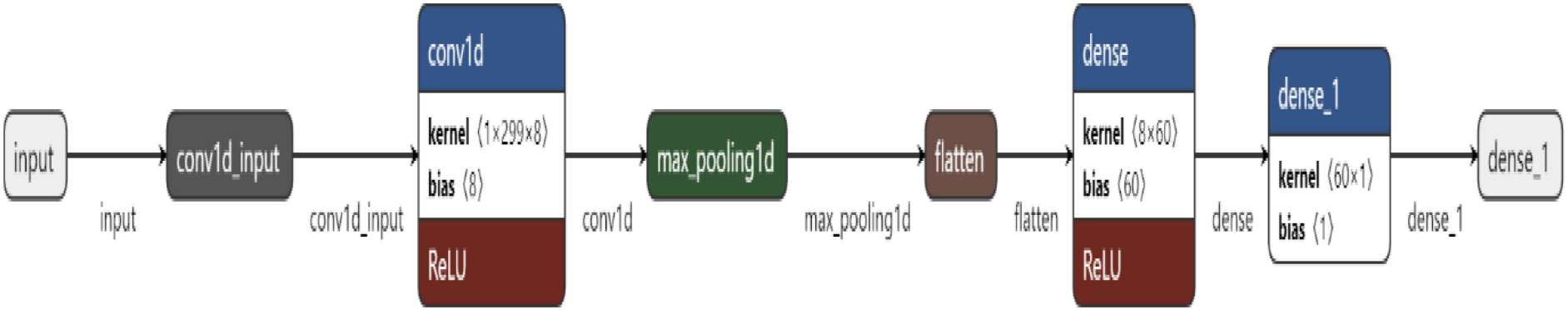
LSTM



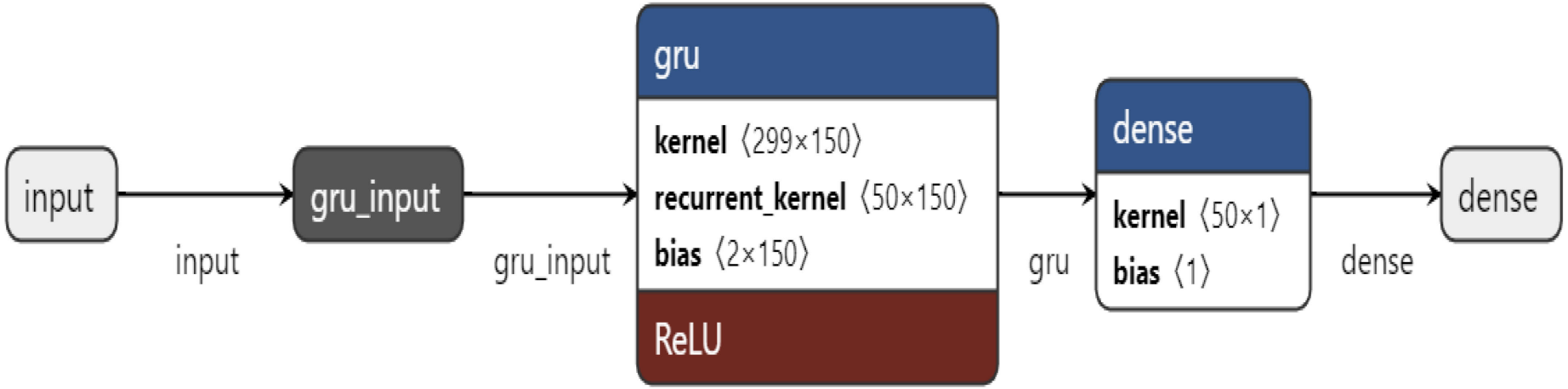
CNN-LSTM


input
conv1d_input
conv1d
kernel ⟨1×299×8⟩
bias ⟨8⟩
ReLU
max_pooling1d
lstm
kernel ⟨8×240⟩
recurrent_kernel ⟨60×240⟩
bias ⟨240⟩
ReLU
dense
kernel ⟨60×1⟩
bias ⟨1⟩
dense
input
conv1d_input
conv1d
max_pooling1d
lstm
dense

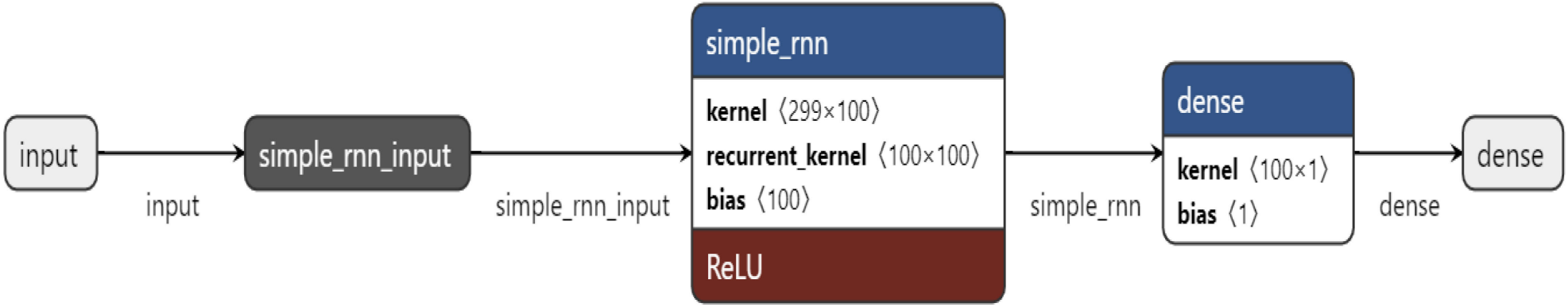
CNN


input
conv1d_input
conv1d
kernel ⟨1×299×8⟩
bias ⟨8⟩
ReLU
max_pooling1d
flatten
dense
kernel ⟨8×60⟩
bias ⟨60⟩
ReLU
dense_1
kernel ⟨60×1⟩
bias ⟨1⟩
dense_1
input
conv1d_input
conv1d
max_pooling1d
flatten
dense
dense_1

GRU



Simple RNN



Further Information

Contact

Ruth Njiru (njiru@iamo.de) - corresponding author
Franziska Appel (appel@iamo.de)
Changxing Dong (dong@iamo.de)
Alfons Balmann (balmann@iamo.de)
All authors are at Leibniz Institute of Agricultural Development in Transition Economies (IAMO), Department of Structural Development of Farms and Rural Areas, Theodor-Lieser-Str. 2, D-06120 Halle (Saale).

Acknowledgements

Financial support from the German Research Foundation (DFG) through Research Unit 2569 “Agricultural Land Markets – Efficiency and Regulation” is gratefully acknowledged. The authors would also like to thank IAMO's IT-team for their patient and constructive support.