UNIVERSITY OF TURKU

Emulation of Dynamic Process-Based Agroecosystem Models Using Long Short-Term Memory Networks

Viivi Aakula

Master's thesis
April 2024

Supervisors:
Julius Vira, Ph.D.
Prof. Ion Petre

DEPARTMENT OF MATHEMATICS AND STATISTICS

UNIVERSITY OF TURKU
Department of Mathematics and Statistics

VIIVI AAKULA: Emulation of Dynamic Process-Based Agroecosystem Models Using Long Short-Term Memory Networks
Master's thesis, 56 pages
Mathematics
April 2024

Modeling carbon balance in agroecosystems help monitoring changes in carbon emissions and influence in ecosystem functioning and productivity. Process-based models are widely used in modeling diverse agroecosystems, and also enable quantification of carbon balance in agroecosystems. However, process-based models tend to be very computationally demanding, due to their complex computations based on hypotheses and assumption of the dynamics of the system. The computational demands complicate performing large scale simulations, needed when simulating several different parameter scenarios, such as model calibration and sensitivity analysis.

In order to mitigate the computational burden of large scale simulations, a surrogate model utilizing neural networks is developed to emulate the behavior of a process-based land model BASGRA_N, obtaining a fast execution time. The emulator recognizes sequentially dependent data by networks specifically designed for sequential learning. Additionally, it is applicable to other similar agroecosystem models. The model is evaluated by 5-fold cross validation, achieving RMSEs of 0.0290 ($g\,C\,m^{-2}h^{-1}$) and 0.322 (m^2m^{-2}) for weekly mean values of hourly NPP and LAI, respectively. Each of the 5 folds give R^2 of > 0.91 for NPP and > 0.93 for LAI.

The thesis begins with basic concepts on neural networks, concerning to regression tasks, covering a fundamental neural network model, its architecture, features, and general training methods. Subsequently, the study continues to sequential modeling and introduces neural networks designed for processing sequentially structured data. Subsequently, an overall review on existing research on machine learning applications, especially in emulation of process-based models, is provided. Lastly a novel emulator model applying neural networks is introduced for emulation of an agroecosystem model.

This project was done in collaboration with Carbon Cycle group of Finnish Meteorological Institute, for their requirement for an emulator for a process-based agroecosystem model BASGRA_N [27] to enable large scale simulations for simulator calibration purposes.

Keywords: Carbon balance, Grassland, Agroecosystem, Machine learning, Emulation, Deep learning, LSTM

Contents

1 Introduction

Quantifying the carbon exchange between land and atmosphere in arable lands enables predicting the influence of vegetation dynamics on climate change and its effects on crop productivity [39]. Carbon exchange in agroecosystems is influenced by various factors such as genetics of the crop, environment and land management [13], which makes its quantification challenging. Carbon cycle is often assessed by process-based models. Process-based models, here often referred as simulators, are computational representations simulating the interactions in a system, which can be of various domains, from ecological to financial systems. Often all the relevant fluxes and state variables of the system cannot be monitored with the required level of detail. By process-based models it is possible to simulate the dynamic processes in a system and extrapolate available data. Developing an accurate process-based model requires a deep overall understanding of the system and its factors.

However, accurate process-based models can be computationally heavy, especially when modeling complex systems. Despite the wide spread access to large scale computing infrastructure, brute-force methods still encounter limits when running complex ecological models over large areas and/or long periods [51]. Additionally, uncertainties may occur in process-based models if the underlying mechanisms are oversimplified or the calibration of the many free parameters is not done properly.

Numerous studies have tried to encounter this problem with diverse modeling approaches. One approach is leaving the parametrization entirely to a data-driven machine learning model. However, machine learning methods are often seen as "black box"-methods and not derived from physical principles, leading to unreliability. Even though, interpretation techniques of machine learning algorithms is an emerging area of research [74, 47]. An alternative approach is creating a surrogate model learning the behavior of the process-based model. Emulation, of the full model or a part of it, aims to reduce the computational cost and complexity of a model by mimicking the behavior of the process-based models with lower computational requirements and faster execution time [55].

The objective of this study is to develop a surrogate model to emulate dynamic agroecosystem models, and to apply it to emulate a grassland model BASGRA_N. Particularly the goal is to emulate the carbon exchange features of the grassland model, including leaf area index (LAI) and carbon net primary production (NPP), which are influenced by many factors including meteorological, vegetation and soil properties. The grassland model predicts the daily values of LAI and NPP, with inputs being daily meteorological values and values on vegetation, soil properties and harvest. Thus, the emulator should be designed for sequential predictions, though for weekly periods instead of daily, since daily values would expand the size of the input sequence significantly. This sequential prediction enables studying the temporal dynamics of meteorological events and their impact on the ecosystem. To achieve this objective, it is necessary to find and apply a suitable machine learning algorithm to effectively learn the dynamical behavior of the model with time dependent input and output values. The training requires sufficient data, which represents the model accurately. The training data is obtained from the simulator runs, consisting of input-output pairs. Subsequently it is crucial to find a method for accurate model validation. The emulator's purpose is to be a faster version of the process-based model without sacrificing too much accuracy. This enables the emulator to serve as a secondary model for large scale simulations.

Machine learning (ML) methods are in increasing use in emulation of complex simulators to overcome their computational limitations. A machine learning emulator replicates the behavior of the simulator by using a ML algorithm to learn the relationships between the simulator input-output pairs. Many diverse ML algorithms have been used for emulation tasks and they all have their strengths and weaknesses thus choosing a suitable ML approach is fundamental. Multiple approaches are represented in Section 4.1.

A deep learning approach is chosen, since there is a need for a ML algorithm able to learn to emulate a complex simulator and to process the sequential values with temporal dependencies. Deep learning is a machine learning approach based on artificial neural networks, here referred only as neural networks (NN). Deep learning is a type of representation learning, meaning that it can learn patterns from raw data [37], while many other machine learning methods require some kind of feature extraction of data [60]. NN emulators are widely used for emulation tasks, as they learn complex patterns effectively [60], though to perform well they require large data sets for learning [37]. Generally, the training time of a NN can be long, as there are easily many parameters to learn in the network [60]. However, they can achieve simulations of orders of magnitudes faster than the process-based simulator, while maintaining a high level of accuracy [55]. Furthermore, deep learning encompasses methods specifically for learning sequential dependencies.

Emulation with machine learning methods is an evolving area of research. Many heavy computational models have been emulated with different ML approaches. However in agroecosystem research, especially in simulating carbon dynamics, emulator models are still in their early stage. Previous research has mainly focused on emulation of static simulations [72, 49, 12, 45], referring to simulations where parameters remain constant over time, thus dynamic changes are not considered. Some studies have included climatic impact by taking the average or sum of climatic values over the whole simulation time period or taking some extreme climatic values as inputs, like maximum air temperature over the simulation period. Agroecosystems are heavily affected by dynamical changes, such as weather, hence, they are often modeled by dynamical process-based models. Dynamical emulation has not gained so much attention in agroecosystem carbon cycle research, though some efforts have been done [1, 42]. However, using Long Short-Term Memory networks, a type of deep learning to comprehend temporal dependencies, is not greatly explored in simulation based agroecosystem carbon dynamic emulation, even though it has shown great capabilities in empirical predictive modeling in the field [44]. Though a study based on a similar method to LSTM [42] introduces hybrid-model combining machine learning with a process-based model. The developed emulator is a relatively straightforward and practical approach, enabling possible applicability to other agroecosystem models, with consideration of temporal dependencies.

The emulator facilitates large scale simulations of carbon exchange in grasslands, which is beneficial for tasks investigating the relationships within the simulator, including calibration and sensitivity analysis. The emulator can be applied to other computational agroecosystem simulators with dynamical and static inputs. Obviously diverse agroecosystem models have their differences, thus some modification should be made.

The following two sections go through the required theory for the emulator, starting from introducing artificial neural networks, their features and general training methods, and consequently presenting neural network methods for sequential learning. Lastly the process of developing an emulator is described starting on related work on possible approaches, continuing to description of the simulator, then data generation, network training and lastly to the performance of the emulator.

2 Artificial neural networks

The goal of a *regression task* is to model the continuous relationship between input variables and corresponding continuous *target* values [23]. An example of a regression task [28] is forecasting the CO_2 emissions in a global level, given input parameters such as the gross domestic product, urban population ratio, and trade openness.

Regression is a supervised learning technique, in other words it learns with labeled data. During the training process, training data is used with examples of input-target pairs, which all include an input vector x and a target vector t with continuous target variables [23]. The goal is to predict the target value t for a new input value x. The challenge is not just to learn to understand the relationship between the inputs and target values of the training data but to also perform well with new input data. Thus, the performance of the model on new data has to be measured. For this the training data is split to a training set and a separate validation set. This way the network can be trained with a separate set and to test its generalization capability with the separate validation set. To reach a small generalization error the model needs to be able to generalize the learned patterns beyond the training set. There are several methods for reducing this error introduced later in Section 2.6.

There are many possible approaches for a regression task. One of which is using neural networks, which stand out with their capability of learning complex patterns within high dimensional data [60]. Neural network tools are effective in problems where prediction without interpretation is the goal [23] and are capable of processing large amounts of data for building an effective data-driven model [60]. The main idea of neural networks is to take linear combinations of the input vector and then to model the target as a nonlinear function of these combinations [23].

Returning to the previous example of a regression task [28], an approach utilizing neural networks is taken, more precisely a feed-forward neural network which is introduced in this chapter. This chapter introduces the basic concepts of deep neural networks with the help of the simplest members of the deep neural networks family. *Feed-forward neural networks (FNNs)* are the fundamental deep learning models. The objective of a FNN is to approximate some function g. In a regression task the network is given an input x and the goal is to find a mapping $y = f(x; \theta)$ and learn the model parameters θ that result in the best possible function approximation [22]. Finding the best approximation is about finding the minimum approximation error, which is the closest function to the desired function g.

Many traditional deep learning model (e.g. FNNs) are not designed for capturing dependencies in time sequences, because they do only take the current input of the model into account [22]. FNNs do not have feedback connections which send the outputs of the model back to itself. Using *Recurrent Neural Networks (RNNs)* is a better choice for capturing time dependencies. RNNs are presented in Section 3.

2.1 Definition and structure of feed-forward neural networks

An FNN constructs of *layers* which are functions connected in a chain $f(x) = f^L(\cdots(f^2(f^1(x)))) : \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}$, L being the number of layers, n_0 the dimension of the input and n_L the dimension of the output. During the training, the function

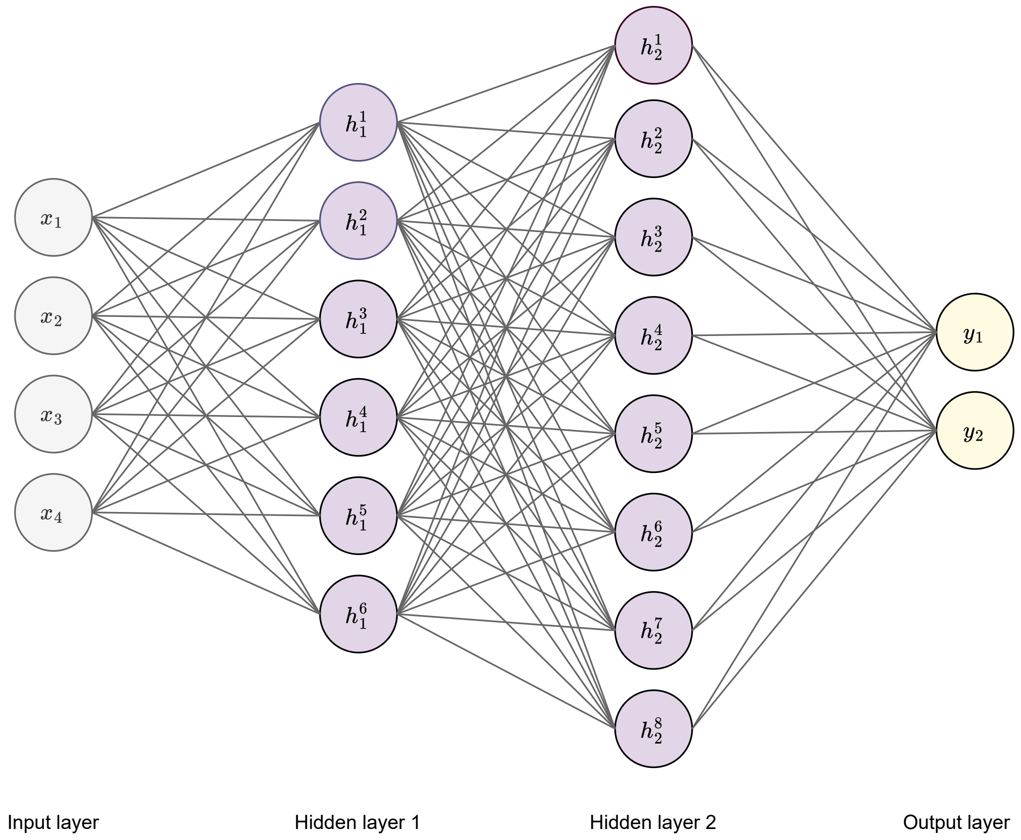


Figure 1: A Feed-forward neural network with two hidden layers. The four grey circles on the left denote the input layer, which consists of the input vector. Each circle, called a unit, corresponds to a value of the vector. Similarly the purple circles denote the units of a hidden layer and the yellow circles denote the units of the output layer. The lines between units are called connections and each of them has a corresponding weight deciding on how much information to pass to the next unit. The connections between two layers can be seen as a function from the previously presented chain of functions.

$f(x)$ is trained to match $g(x)$ as well as possible. The *output layer* f^L's job is to give values as close to the value $y \approx g(x)$ as possible. All the other layers are called *hidden layers*, as their job is defined with the learnable parameters.

Figure 1 has an example of a three layer ($L = 3$) feed-forward neural network $f(x) = f^3(f^2(f^1(x))) : \mathbb{R}^4 \to \mathbb{R}^2$, where f^1 maps the input layer of size 4 to the first hidden layer of size 6, $f^1(x) = (h_1^1, ..., h_1^6) = h_1$, f^2 maps the first hidden layer to the second hidden layer of size 8, where $f^2(h_1) = (h_2^1, ..., h_2^8) = h_2$, and lastly f^3 maps the second hidden layer to the output layer of size 2, where $f^3(h_2) = (y_1, y_2) = y$.

The feed-forward network consists of an input layer, an output layer and any number of hidden layers in between. The layers are interconnected with weights, but information flows only forward. FNNs are acyclic, so there are no backward connections in the network.

First the input vector is given to the network by setting the states of the input units. Then the network transforms the states in order, each layer at a time until the states of the units of the output layer are defined. At each layer of the network the states of the layer's units are determined by applying Equations 1 and 2 to the connections from preceding layers [58].

Activation functions applied to the output introduce non-linearity to the output of the unit. Section 2.2 provides commonly used activation functions.

Let a be the activation function, $W \in \mathbb{R}^{n_l \times n_{l-1}}$ the weight matrix, and $b \in \mathbb{R}^{n_l}$ the bias vector, where n_{l-1}, n_l are the sizes of the previous and current hidden layer. Now the *hidden state* or output h_l of a fully connected hidden layer l is calculated by the following equations

$$z_l = W_l h_{l-1} + b_l, \tag{1}$$

$$h_l = a(z_l). \tag{2}$$

Adding the bias vector is optional. Also any input-output function with a bounded derivative can be used instead of 1 and 2, but using a linear function for combining the inputs to a unit before adding non linearity simplifies the learning process [58]. In this work, the previously defined fully connected layer is referred as the *dense layer.*

Consider the example of Figure 1 and let the Equations 1 and 2 be used in the computations of the hidden layer and the output layer. Now the output would be $y = a(W_2(a(W_1 x + b_1) + b_2)$, where $W_1 \in \mathbb{R}^{6\times 4}$, $b_1 \in \mathbb{R}^6$, $W_2 \in \mathbb{R}^{2\times 6}$ and $b_2 \in \mathbb{R}^2$.

2.2 Activation functions

Activation functions are applied to the output of a node to add non-linearity to a neural network [22].

The activation function for the output layer is chosen by the characteristics of the data and the expected distribution of target variables [3]. For standard regression problems the usual output layers activation function used is the identity and the *sigmoid activation function* is a common one in classification problems [3]. There are various types of activation functions from which three are introduced.

Definition 1. *Sigmoid Activation Function σ is defined by the equation:*

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

The sigmoid function compresses the given value to range of $[0, 1]$. With very large or small values the output tends to be either extremely close to 0 or 1, which raises difficulties in gradient-based training, know as the *vanishing gradient problem* [14], which is covered in Section 3.2.

The *tanh activation function* is similar to the sigmoid function. This function compresses the given value to range of $[-1, 1]$.

Definition 2. *Tanh Activation Function* tanh *is defined by the function:*

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

The tanh function also suffers from the vanishing gradient problem [14].

Lastly the *ReLU activation function* is introduced.

Definition 3. *ReLU (rectifier linear unit) function R is defined by the equation:*

$$R(x) = max(0, x).$$

The ReLU activation function is a commonly used activation function due to its simplicity and it does not suffer from the vanishing gradient problem [14].

2.3 Loss functions

Loss functions, also referred as *error functions*, denoted by E, are functions for measuring the performance of the model. The error occurring during training the model with a training set is called *train error* and the error of the model performance with a test set is called *test error* or *generalization error* [23]. One common loss functions for regression tasks is the *mean squared error.*

Definition 4 ([22])**.** *Mean squared error (MSE) is a loss function calculated by*

$$MSE(y, \tilde{y}) = 1/n \sum_{i=1}^{n} (y - \tilde{y})_i^2,$$

where y is a vector of the models predictions for some data set and $\tilde{y}$ is the target vector.

Denote train MSE by MSE_{train} and test MSE by MSE_{test}. The MSE function is a convex function with respect to its inputs, as it is quadratic. Thus, minimizing MSE on its inputs is a convex optimization problem, which can be solved with numerical algorithms. However, regarding to neural networks, the MSE function is not convex, when given the non-linearities of the activation functions. *Root mean squared error (RMSE)* is also a commonly used measure for the model performance, which is obtained by the square root of MSE.

2.4 Backpropagation

The goal of training a NN model, is to improve the weights of the model in a way that reduces the loss function on the test set. One way of doing this is to minimize the loss function for the train set, which can be done by minimizing the error gradients with respect to the weights of the network. The gradients are calculated by moving backwards through the network, starting from the last layer and moving towards the first layer. This procedure for calculating the error gradients, also referred to as *error signals*, is called *backpropagation.* It is a part of the learning algorithm where the weights of the connections are repeatedly adjusted with the purpose of minimizing the loss function. Thus, the goal of the learning procedure is to find a set of weights that ensure each input vector having an output vector close to the desired output.

To minimize the loss function, denoted by E, by methods covered in the following section, it is first necessary to calculate the partial derivative of E with respect to each of the weights of the network. Let w^{jk} be a single weight from unit k to unit j. The gradient for the single weight w^{jk} is calculated as follows [58]. By applying the chain rule

$$\frac{\partial E}{\partial w^{jk}} = \frac{\partial E}{\partial z^j} \cdot \frac{\partial z^j}{\partial w^{jk}}.$$

By the definition of z^j

$$\frac{\partial E}{\partial w^{jk}} = \frac{\partial E}{\partial z^j} \cdot \frac{\partial}{\partial w^{jk}}(\sum_{k=1}^{n} w^{jk}h^k + b^j).$$

By differentiation

$$\frac{\partial z^j}{\partial w^{jk}} = h^k.$$

Thus, the partial derivative of E with respect to the weight w^{jk} is

$$\frac{\partial E}{\partial w^{jk}} = \frac{\partial E}{\partial z^j} \cdot h^k.$$

The partial $\frac{\partial E}{\partial z^j}$ is the error signal denoted by ϑ^j. Now $\frac{\partial E}{\partial z^j}$ can be computed as follows

$$\vartheta^j = \frac{\partial E}{\partial z^j} = \frac{\partial E}{\partial h^k}\frac{\partial h^k}{\partial z^j} = \frac{\partial E}{\partial h^k}a'(z^j).$$

And lastly the partial $\frac{\partial E}{\partial h^k}$ can be computed as follows:

$$\frac{\partial E}{\partial h^k} = \sum_j \frac{\partial E}{\partial z^j}\frac{\partial z^j}{\partial h^k} = \sum_j \frac{\partial E}{\partial z^j} w^{jk}.$$

Now the *backpropagation* formula is

$$\vartheta^k = a'(z^k)\sum_j w^{jk}\vartheta^j, \tag{3}$$

which shows that the error signal of a particular hidden unit can be obtained by propagating the error signals backwards from subsequent units [2].

This procedure can be repeated successively through all the layers from last to first. A similar calculation can be applied for finding the gradient of a single bias weight b^j.

2.5 Optimization algorithms

The next problem is to find the parameters minimizing the error function. The previously calculated gradients of the error function with respect to network parameters play a central role in optimization of the parameters of the model.

One of the simplest and most popular optimization algorithm is the *gradient descent*, which iteratively minimizes the loss function. Gradient descent and its variants are first-order optimization algorithms, leading to computationally efficient optimization of the large numbers of parameters in the networks. Gradient descent requires an initial guess for the weights, which is often chosen at random. Then the weight vector is updated iteratively so that at each step the weights are adjusted by taking a step α, called the *learning rate*, in the direction of the negative gradient as follows [2]:

$$\Delta w = -\alpha \frac{\partial E}{\partial w},$$

where the error is calculated for the whole training set. At each step, the gradient is re-evaluated. One sweep through the entire data set is called a *training epoch* [23]. In the *stochastic* version of the gradient descent, an error is calculated for one data point at a time and the weights are updated according to the single error function gradient [2]. This version allows the network to handle large data sets and to update the weights as new data points come in.

The regular vanishing gradient method is more expensive computationally, as it evaluates the error function for the whole data set. The stochastic version, on the other hand, can help in escaping from a local minima, since a stationary point for the error function of the whole set is not necessarily a stationary point for the error function of a single data point [3]. But obviously single data points introduce more variance in the weights. A variation of these two algorithms processes subsets of data points, called *batches*, and updates the weights according to them [3].

The challenge in gradient descent is that it requires a fixed learning rate for all parameters, which can significantly affect the model performance. Usually the learning rate is chosen to be a constant and it can be challenging to learn an optimal value for it. If the value is chosen to be too large, the error may increase and the algorithm may not be able to converge. In the other hand if the learning rate is chosen to be too small, the search may proceed very slowly, leading to long computational time [3].

Also often the error function is highly sensitive to some directions in the parameter space and insensitive to others. Assuming that the directions of sensitivity align somewhat along axes, then one solution is using separate learning rates for each parameter and automatically adapting them throughout the learning process [22].

Adam [32] is an optimization algorithm using adaptive learning rates. It is a stochastic gradient descent method that only requires first order gradients and has little memory requirement. The name derives from “adaptive moment estimation”. The method calculates individual adaptive learning rates for the loss functions parameters from estimates of first and second moments of the gradients.

Definition 5 (The Adam algorithm [32]). *Let $f(\theta)$ be a stochastic scalar function that is differentiable w.r.t the parameters θ. By $f_1(\theta), ..., f_T(\theta)$ denote the realisations of $f(\theta)$ at consecutive time steps $1, ..., T$. And by $g_t = \nabla_\theta f_t(\theta)$ denote the gradient vector consisting of the partial derivatives of f_t w.r.t θ at time step t. Now for time* *step t, the algorithm calculates the updated parameters θ as follows:*

$$\begin{aligned}
&m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t,\\
&v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2,\\
&\hat{m}_t = m_t/(1-\beta_1^t),\\
&\hat{v}_t = v_t/(1-\beta_2^t),\\
&\theta = \theta_{t-1} - \alpha \hat{m}_t/(\sqrt{\hat{v}_t} + \epsilon),
\end{aligned}$$

where g_t is the given gradient for time step t, m_t denotes the biased first moment vector, v_t the biased second moment vector, $\hat{m}_t$ denotes the bias-corrected first moment vector, $\hat{v}_t$ the bias-corrected second moment vector, θ denote the parameters and the hyperparameter α denotes the learning rate and β_1, β_2 denote the exponential decay rates for the moment estimates. The algorithm iteratively updates the parameters θ, until the converge criterion is met.

The Adam algorithm (5) requires a step size α, exponential decay rates for the moment estimates β_1 and β_2, an initial parameter vector θ_0 and a noisy stochastic objective function or loss function $f(\theta)$ that is differentiable with respect to its parameters θ [32]. One step includes first updating the exponential moving averages of the gradient m_t and the squared gradient v_t, which are estimates of the 1st moment and the 2nd moment. The hyperparameters $\beta_1, \beta_2 \in [0, 1)$ control the exponential decay rates of the moving averages. The 1st moment captures the exponentially decaying average gradient of past gradients and the algorithm moves in their direction. The 2nd moment captures the variance of the gradients, which is used in adjusting the weights per each parameter. These moving averages are initialized as zeros, which leads to moment estimates biased towards zero. To prevent this, the bias-corrected estimates $\hat{m}_t, \hat{v}_t$ are presented.

All of the introduced algorithms, and many others, require initialization for the weights of the network. The algorithms can be sensitive to the initial conditions, thus choosing suitable initial weights is crucial. A common practice is to choose the weights to have random values near zero [23]. Random values are used for avoiding problems due to symmetries in the networks [2]. Symmetry here could be two units with same activation functions connected to the same inputs. If these units have the same initial parameters, then a deterministic algorithm applied to a deterministic model and error function would update these units the same way. Usually the random values are drawn from a uniform distribution or Gaussian distribution, which have not shown great differences in model performance [22]. The scale of the initial distribution has a large effect on the optimization process and generalization of the model [22]. Large initial weights lead to a stronger effect on breaking the symmetry in networks. Also they can help in preventing the vanishing gradient problems. However when they are set to large values it may result in exploding gradient problem.

The optimal scale depends also on the activation function. Considering a FNN with sigmoid activation function, the weights can not be too small or the activation functions are roughly linear, which can lead in slow training. With large initial weight values the sigmoidal functions are driven to the regions where $\sigma'(z)$ is very small, leading to small ΔE [2].

2.6 Overfitting and regularization techniques

As mentioned before, one of the goals of a machine learning model is to minimize the train error. Another important goal is to minimize the gap between the train error and the test error [22]. *Underfitting* is the term for the the machine learning model's inability to obtain a low train error. And *overfitting* is the term used when the gap between train error and test error is too large. Overfitting is a common problem in neural networks when they have too many weights to learn. There are various techniques to control the complexity of a neural network in order to prevent the model from overfitting.

Finding optimal hidden sizes and amount for layers have a big effect on the models generalization performance [23]. The manual tuning of the parameters requires an overall understanding of the relationship between the hyperparameters and a good generalization performance of the model. Another hyperparameter tuning technique is using optimization algorithms, which though are more computationally expensive [22]. For cases with a small amount of hyperparameters, a commonly used algorithm is the *grid search.* The algorithm requires a finite set of possible hyperparameter values for each parameter. Then the algorithm trains the model for every combination of the hyperparameter values in the Cartesian product of the set of values for each hyperparameter. Finally the hyperparameter combination resulting in the best performance is chosen as the model hyperparameters. Furthermore, in general the selection of small batch size can lead to better regularization and often the best choice for batch size is 1 [71]. The selection of the learning rate is also crucial and affects the regularization performance. The choice of batch size and learning rate can also be done by grid search in order to find the optimal values for the specific task and learning algorithm.

Modifications of the model towards decreasing the generalization error without decreasing the train error are called regularization techniques [22]. One technique to avoid overfitting is the procedure of *early stopping*, where the model is trained for only a while and stopped well before the global minimum [3]. A validation set is used for determining when to stop, before the model overfits to the training data. Another regularization technique is using the *weight decay*, where a regularization term or a penalty [23] is added to the error function $E(w) + \frac{\lambda}{2}x^T w$, where $\lambda \geq 0$ is a hyperparameter [3]. When λ is chosen to be larger, the weights are forced to become smaller. This leads to a trade off between fitting the training data and small weights. Also many other regularization terms can be used [22].

Creating a network to map raw data directly to the desired output may not be optimal in practice. Generally it is advantegous to apply *pre-processing* transformation to the raw input data [2]. Also *post-processing* for the outputs of the network is often done to get the required output values. Pre-processing can have a significant effect on the generalization performance of the network. Usual pre-processing form is reduction of the dimensionality of the input, which can involve taking a subset of the original data or forming combinations of the original variables. Less inputs lead to less weights, thus the network is less likely to overfit to the training data [2]. Also this can affect to faster convergence during training. Of course in some cases this kind of reduction can result in loss of information.

Another pre-processing form consists of a linear rescaling of the inputs. Scaling

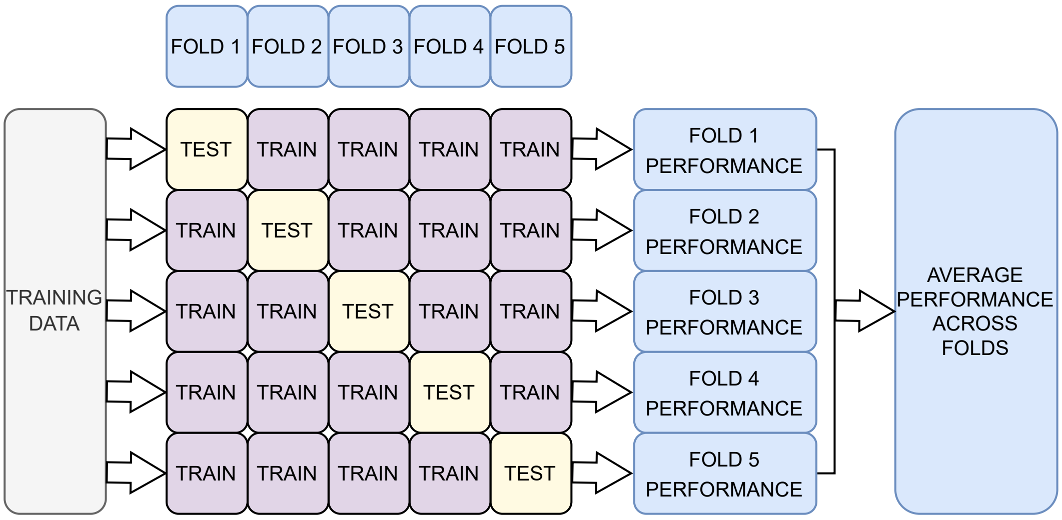


Figure 2: K-fold cross-validation with 5 folds.

of the inputs can have a large impact on the model performance as it also determines the scaling of the weights of the input layer [23]. Rescaling can be useful in cases including variables with significantly differing values, which may not reflect their relative importance in producing the desired output values [2]. For example with values given in different units the difference may be crucial. The rescaling is performed by first calculating the mean $\overline{x}_i$ and the standard deviation σ_i^2 of each variable with respect to the training set. Then the rescaled variables are given by:

$$x_i^* = \frac{x_i - \overline{x}_i}{\sigma_i^2}.$$

This way the inputs are standardized to have mean of zero and standard deviation of one. Now all inputs are treated equally in the network and the weight can be given a random initialization without having to deal with some weights having remarkably different values from others. Similar rescaling is usually done also to the target values. In that case the prediction has to be post-processed by the inverse of the target scaling in order to get the final prediction in the right scale.

2.7 K-fold cross-validation

Division of the data set to fixed training set and test set with a limited data can be problematic. If the test set is too small, the test error may not give an accurate measure of the performance of the model.

K-fold cross-validation is a data resampling method for estimating the error of the model performance [23]. The data set is split to K separate and roughly same sized subsets called *folds*. The model is trained K times, using one fold at a time as the test sample and the rest of the folds as the training sample. So at the k'th training iteration, the model is fitted to the other $K-1$ subsets of the data. Each of the K iterations give a test error, which can be averaged for estimating the model performance.

In Figure 2 the training data is divided in to 5 folds, each fold performing once as the test set and the other four times as a part of the train set. Finally the performance is evaluated by taking the average performance across folds.

Stratified sampling is a common practice in cross-validation. In this method the training data is split to folds of good representatives of the whole data set [54]. In a classification task this means that the proportions of different classes in the whole data set are maintained in the folds. In a regression task the sampling can be done by defining classes based on the distribution of the continuous target values. The stratified sampling in cross-validation is recommended by [33], which compares several accuracy estimation approaches. They concluded that stratified 10-fold cross-validation tends to provide less biased estimation of the accuracy.

While cross-validation is a valuable technique for model estimating the model performance, it is worth noting that with a high value of K, cross-validation can be problematic in cases where the training itself is computationally expensive.

3 Recurrent neural networks

Feed forward neural networks assume independence among the data, in other words after each data point processed by the network the state of the network is lost. This procedure is not optimal in the case of data points with temporal dependencies or sequential patterns. There are numerous tasks involving modeling sequential data. Some tasks, such as image captioning [29], require sequential outputs. Other tasks, such as grammatical classification of natural language sentences [34], require sequential inputs and others, like translation of natural language [63], require both sequential input and sequential output.

Recurrent neural networks (RNNs) [57] are models designed for processing sequential data. RNNs have the same central idea with the previously presented FNNs but with the addition of feed-back connections. They process sequential data one element at a time while selectively passing information across the time steps. RNNs are able to handle interdependent sequential data as input and/or output. In this thesis the consideration is on the development of models for tasks including sequential input and sequential output. This chapter gives and overview on classical RNNs, goes through their training challenges and finally presents a RNN model designed for overcoming these challenges.

Recurrent neural networks consist of recurrent units which are dependent on the previous elements in the sequence, unlike in traditional neural networks which assume the outputs to be independent of each other [22]. Let x be an input sequence, where $x = x_1, x_2, ..., x_T$, where all $x_t \in \mathbb{R}^d$. Now if a traditional FNN model was in training, it would be given the whole sequence at once. This approach would ignore the possible temporal dependencies in the sequence x. RNNs take in consideration the temporal dependencies of the data by going through the input sequence one step at a time and by sharing the output of each time step to the next one. As in feed forward neural networks, the weights of a recurrent neural network are trained with backpropagation and optimization algorithms based on gradient descent.

3.1 Recurrent units

A standard recurrent unit can process the sequential data one time step at a time. At time step t the unit reads the input x_t and the previous hidden state h_{t-1}. The output is the new hidden state h_t, which is then again given to the recurrent unit with the next input vector.

At time step t, the hidden state h_t is calculated by:

$$h_t = f(h_{t-1}, x_t, \theta),$$

where f represents the transformation of the recurrent unit [15].

A recurrent fully connected layer can be seen as its own network, each time step having its own layer. This is visualized in Figure 3, where on the right the RNN layer is unfolded and can be seen as a network with T layers.

Let a be the activation function, $W \in \mathbb{R}^{n\times n}$ the recurrent weight matrix, $U \in \mathbb{R}^{n\times d}$, and $b \in \mathbb{R}^n$ the bias vector, where n denotes the hidden size and d the size of the input vector. Now for time step t the weighted input z_t and the hidden state h_t

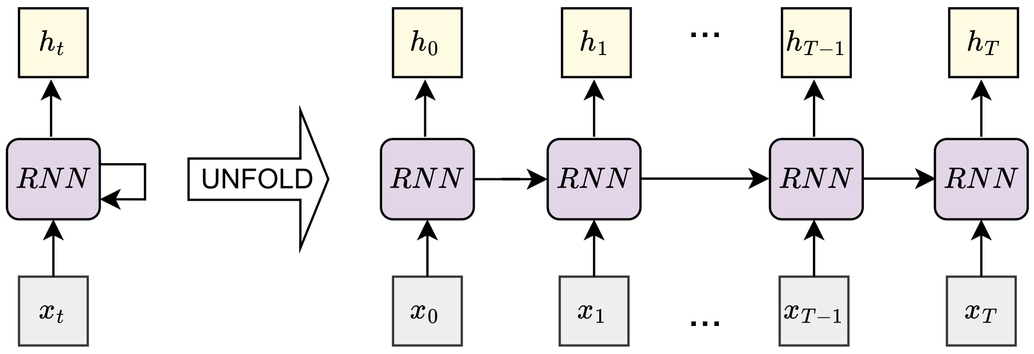


Figure 3: A RNN unit loops through time steps by taking an input x_t of some time step t and outputting h_t and a loop to give information to the next time step $t + 1$. On the right the loop is unfolded to visualize the flow of information through the step-by-step computations.

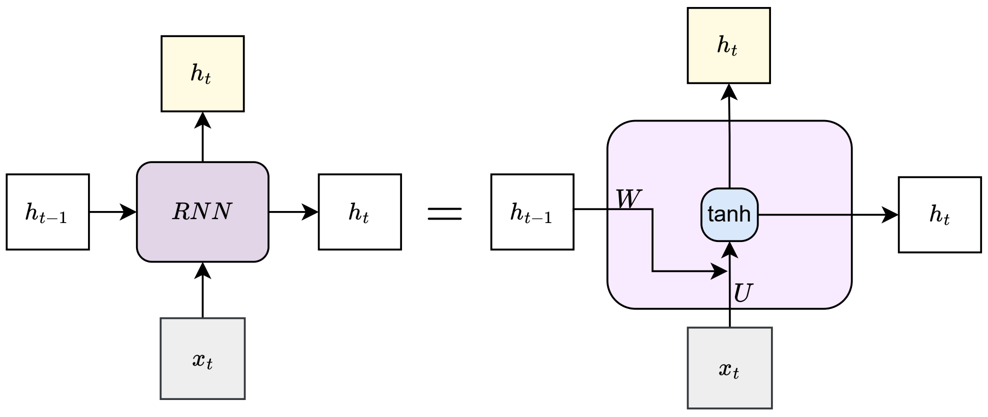


Figure 4: RNN unit with tanh-activation.

of a standard recurrent hidden layer are calculated by the following equations [50]:

$$z_t = Wh_{t-1} + Ux_t + b, \tag{4}$$

$$h_t = a(z_t). \tag{5}$$

Figure 4 visualizes a RNN unit with tanh-activation. The previous hidden state is multiplied by the weight W and the current input by the weight U, then tanh-activation is applied and the unit outputs the new hidden state and gives it to the next unit of the following time step.

3.2 Backpropagation through time and the vanishing gradient problem

As in Section 2.4, a similar backpropagation procedure is done to RNN layers. The gradients are calculated with *backpropagation through time (BPTT)* [70], where the recurrent network is presented as a FNN, each time step having its own layer, and backpropagation is applied to the unfolded network (example in Figure 3). The difference here is that the network has an output for each time step, so the error

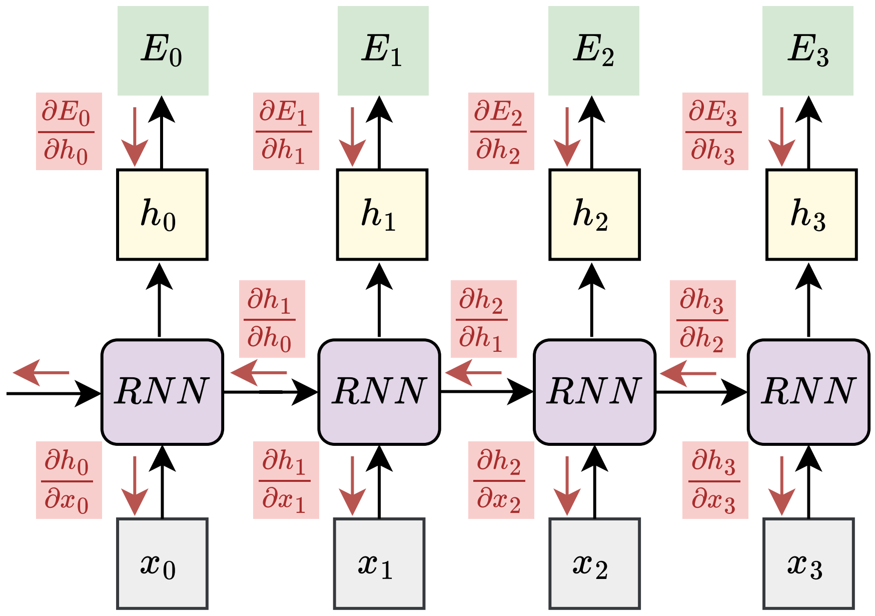

E_0
E_1
E_2
E_3
$\frac{\partial E_0}{\partial h_0}$
$\frac{\partial E_1}{\partial h_1}$
$\frac{\partial E_2}{\partial h_2}$
$\frac{\partial E_3}{\partial h_3}$
h_0
h_1
h_2
h_3
$\frac{\partial h_1}{\partial h_0}$
$\frac{\partial h_2}{\partial h_1}$
$\frac{\partial h_3}{\partial h_2}$
RNN
RNN
RNN
RNN
$\frac{\partial h_0}{\partial x_0}$
$\frac{\partial h_1}{\partial x_1}$
$\frac{\partial h_2}{\partial x_2}$
$\frac{\partial h_3}{\partial x_3}$
x_0
x_1
x_2
x_3

Figure 5: The backpropagation of a recurrent network with 4 time steps. The sequential calculation is represented by the red arrows in the unfolded network. For example, the second unit gets an error signal from its output and the following time steps unit.

signal of each output has to be noted in the backpropagation calculations. This is illustrated in Figure 5 with an example of a RNN with four time steps.

The training of a RNN model is problematic, because the backpropagated gradients tend to either increase or decrease at each time step, which can result in "explosion" or "vanishing" of the gradients [25].

The *vanishing gradient problem* is the scenario where the gradient of the objective function with respect to a parameter becomes very close to zero, which leads to an extremely small update to the parameters during the training of a network with gradient descent [14]. The vanishing gradients lead to uncertainty about the optimal direction for weight updates, which causes the network to be difficult and slow to learn. Exploding gradients, on the other hand, lead to large weight updates which causes unstable learning [22]. It can occur in networks with extremely many layers, or recurrent layers. The backpropagation process may consist of multiplying the same weight matrix over and over again which will eventually lead to vanishing or explosion of the gradients [22] or this can be caused by an activation function squashing the value to a small interval [14].

This section explains the BPTT for a RNN and how the vanishing gradient problem occurs. The following backpropagation calculations and error flow analysis are derived from the Article [26].

Assume a fully connected RNN with one recurrent layer with n units. Let a_j be the activation function for a non-input unit j and $z_j^t = \sum_k w^{jk} h_{t-1}^k$ the weighted input and $h_t^j = a_j(z_t^j)$ the hidden state, where w^{jk} is the single weight for a connection from unit k to j. Now define the external error e and the error E for an output unit i at current time t using MSE:

$$E_t^i = (h_t^i - \tilde{y}_t^i)^2,\ e_t^i = \frac{\partial E_t^i}{\partial h_t^i} = 2(h_t^i - \tilde{y}_t^i),$$

where h_t^i denotes the value of the prediction given by the output unit i and $\tilde{y}_t^i$ denotes the target value. The external error is the error signal of the output unit. Now the backpropagated error signal of a non-input unit k at an arbitrary time $\tau \leq t$ is the sum of the external error and the backpropagated error signal from the following time step $\tau + 1$, given by the previously derived Equation 3:

$$\vartheta_\tau^k = a_k'(z_\tau^k)(e_t^k + \sum_j w^{jk}\vartheta_{\tau+1}^j).$$

Now the weight update with gradient descent at time τ is

$$w^{jk} = w^{jk} + \alpha\vartheta_\tau^j h_{\tau-1}^k,$$

where α is the learning rate and k is an arbitrary unit connected to unit j. Now an error occurring at an arbitrary unit u at time step t is backpropagated to a unit v over q time steps. This scales the error by

$$\frac{\partial\vartheta_{t-q}^v}{\partial\vartheta_t^u} = \begin{cases} a_v'(z_{t-1}^v)w^{uv}, & q = 1, \\ a_v'(z_{t-q}^v)\sum_{l=1}^n \frac{\partial\vartheta_{t-q+1}^l}{\partial\vartheta_t^u}w^{lv}, & q > 1. \end{cases}$$

Now denote the units from u to v by $l_0, ..., l_q$. The total error backflow is

$$\frac{\partial\vartheta_{t-q}^v}{\partial\vartheta_t^u} = \sum_{l_1=1}^n \cdots \sum_{l_{q-1}=1}^n \prod_{m=1}^q a_{l_m}'(z_{t-m}^{l_m})w^{l_m l_{m-1}}.$$

Now if for all m:

$$|a_{l_m}'(z_{t-m}^{l_m})w^{l_m l_{m-1}}| > 1, \tag{6}$$

then the maximum product increases exponentially with q. In this situation the error "explodes" and can cause instability during learning. And, if

$$|a_{l_m}'(z_{t-m}^{l_m})w^{l_m l_{m-1}}| < 1, \tag{7}$$

for all m, then the maximum product decreases exponentially with q. In this situation the error vanishes, and nothing can be learned within a reasonable time.

3.2.1 Constant Error Carousel (CEC)

Assume a single unit k with a single connection to itself. According to previously shown calculations, k's local error backflow at time t is $\vartheta_t^k = a_k'(z_\tau^k)(E_t^k + \sum_j w^{kk}\vartheta_{t+1}^k)$. Equations 6, 7 indicate that for ensuring constant error flow through k it is required that

$$a_k'(z_t^k)w^{kk} = 1.$$

By integration,

$$a_k(z_t^k) = \frac{z_t^k}{w^{kk}},$$

for arbitrary z_t^k. Thus, a_k has to be linear and the activation of k has to remain constant:

$$h_{t+1}^k = a_k(z_{t+1}^k) = a_k(z_t^k) = h_t^k.$$

This is ensured by using the identity function as the activation function a_k and by setting $w_{kk} = 1$ [26]. This is called the constant error carousel (CEC) and is the central feature of *Long Short-Term Memory (LSTM)* network introduced in the following chapter. The CEC enforces a constant error flow within units, which allows learning long term dependencies. Obviously k is not only connected to itself, but has also connections to other units. Input and output weights are taken care of with additional features in the LSTM network, introduced in the following chapter.

3.3 Long Short-Term Memory network

Long Short-Term Memory (LSTM) neural network is presented in [26] with the goal of solving the vanishing gradients problem. The key idea consists of the Constant Error Carousel (CEC) and the addition of nonlinear and data dependent gates to the RNN unit [26].

3.3.1 LSTM cell

The LSTM unit, here referred to as the *cell*, has more internal structure than the previously covered units. In addition to the recurrency of the network, the LSTM cell has an internal recurrence, an internal loop. The i'th LSTM cell has two recurrent features; the hidden state h_i and the cell state c_i. The hidden state works as the short-term memory and the cell state works as the long-term memory of the cell.

Without new inputs to the cell, CEC's error flow remains constant. For the weighted inputs and outputs, [26] introduces gates to control the information flow through the cells. The input gate controls the information from the network to the cell state from irrelevant inputs and the output gate protects other units from currently irrelevant information stored in the cell state. An additional forget gate introduced by [21] learns to reset old, thus irrelevant, information from the cell state.

The cell, denoted as C_{LSTM} is given three inputs: the hidden state h_{t-1} and cell state c_{t-1} of the previous time step and the current input vector x_t. And it produces two outputs: the new hidden state h_t and cell state c_t [15]:

$$(h_t, c_t) = C_{LSTM}(h_{t-1}, c_{t-1}, x_t). \tag{8}$$

Definition 6 ([22]). *An LSTM cell C_{LSTM} consists of the following components:*

$$f_t = \sigma(z_t^f) = \sigma(W_f h_{t-1} + U_f x_t + b_f), \tag{9}$$
$$i_t = \sigma(z_t^i) = \sigma(W_f h_{t-1} + U_i x_t + b_i), \tag{10}$$
$$o_t = \sigma(z_t^o) = \sigma(W_o h_{t-1} + U_o x_t + b_o), \tag{11}$$
$$g_t = \tanh(z_t^g) = \tanh(W_g h_{t-1} + U_g x_t + b_g), \tag{12}$$
$$c_t = f_t c_{t-1} + i_t g_t, \tag{13}$$
$$h_t = o_t \tanh(c_t). \tag{14}$$

Let n denote the hidden size and d the size of the input vector. Now $f_t \in (0,1)^n$ denotes the activation vector of the forget gate, $i_t \in (0,1)^n$ the activation vector of the input gate, $o_t \in (0,1)^n$ the activation vector of the output gate and $g_t \in (-1,1)^n$ the activation vector of the cell input. The notation $x_t \in \mathbb{R}^d$ denotes the input vector, $h_t \in (-1,1)^n$ the hidden state vector and $c_t \in \mathbb{R}^n$ the cell state vector. $W \in \mathbb{R}^{n \times n}$, $U \in \mathbb{R}^{n \times d}$ and $b \in \mathbb{R}^n$ respectively denote input weights, recurrent weights and biases.

Figure 6 gives a visualization of the LSTM cell. At time step t the cell gets h_{t-1}, c_{t-1} and the current input vector x_t as inputs. The forget gate f_t takes the weighted combination of the current input x_t and the weighted hidden state h_{t-1} from the previous time step and squashes the values between 0 and 1 to decide how much to be forgotten of the last cell state c_{t-1}. The input gate i_t works similarly to the forget gate and decides on which values to update. g_t is a vector of new candidate values and is combined with i_t to create the new cell state c_t. Finally the output is filtered from the new cell state: the gate o_t decides on how much information to keep from c_t and the cell state goes through tanh and is multiplied with the output of the output gate to keep only the information decided to preserve.

3.3.2 BPTT in LSTM

The backpropagation for a LSTM network differs from the BPTT for RNNs, as the LSTM has two memory units and three gates, each depending on the previous time steps. The signals are calculated for each of the variables.

Assume a network with a standard input layer and one hidden layer consisting of a LSTM cell, and the output of the network is given by the hidden states of each iteration of the LSTM cell, thus $y = (h_1, ..., h_T)$. Again define the external error e_t^l and the error E_t^l for an output unit l at current time t using MSE:

$$E_t^l = (h_t^l - \tilde{y}_t^l)^2, \; e_t^l = \frac{\partial E_t^l}{\partial h_t^l} = 2(h_t^l - \tilde{y}_t^l),$$

where h_l^t represents the value of the prediction given by the output unit l and $\tilde{y}_t^l$ denotes the target value.

Now calculate, using chain rule, the signals to the gates and the cell input

$$\vartheta_t^f = \frac{\partial E_t}{\partial f_t} = \frac{\partial E_t}{\partial c_t}\frac{\partial c_t}{\partial f_t} = \vartheta_t^c c_{t-1},$$
$$\vartheta_t^i = \frac{\partial E_t}{\partial i_t} = \frac{\partial E_t}{\partial c_t}\frac{\partial c_t}{\partial i_t} = \vartheta_t^c g_t,$$

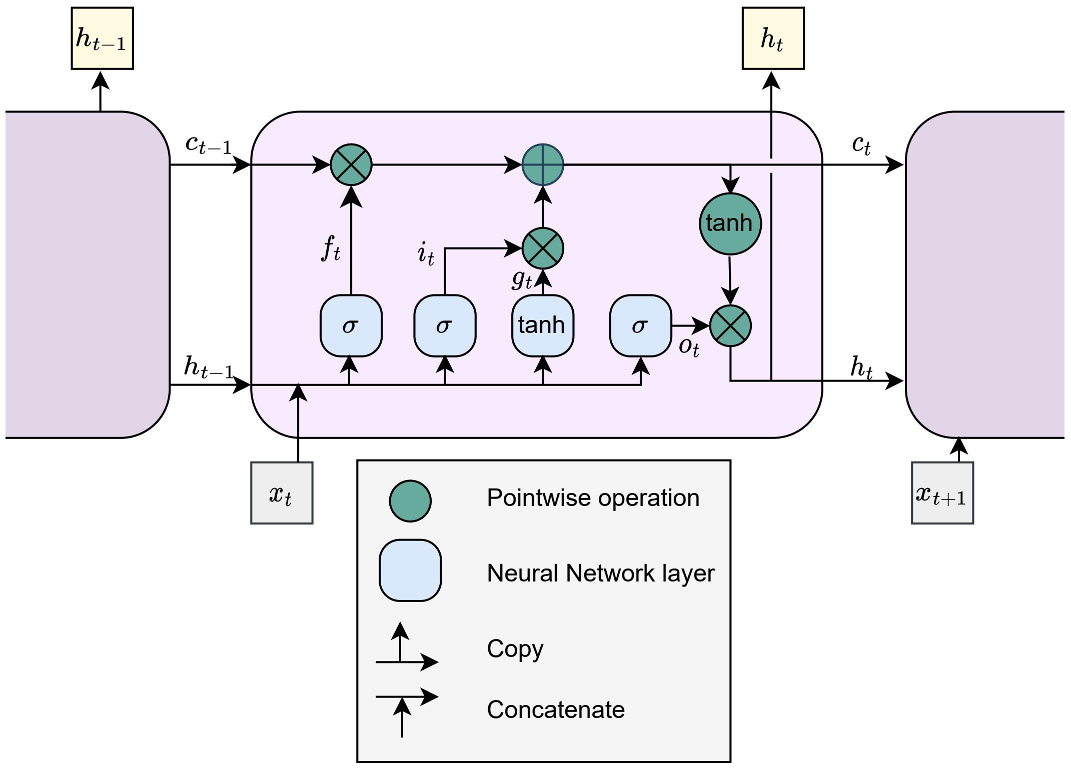


Figure 6: The LSTM cell at time step t is given the hidden state h_{t-1} and cell state c_{t-2} of the previous time step and the current input vector x_t. The forget gate f_t processes h_{t-1} and x_t by taking their weighted sum and applying it through the sigmoid activation function, as shown in Equation 9. The output of the forget gate is then multiplied with the previous cell state. The input gate i_t has the same functioning as the forget gate but with its own set of weights (Equation 10). And the cell input activation vector g_t also takes the weighted sum of h_{t-1} and x_t and applies it through the tanh activation function (Equation 12). Now the new cell state c_t is the sum of the product of i_t and g_t and the product of previous cell state c_{t-1} and the output of the forget gate f_t (Equation 13). Similarly the output gate o_t applies the sigmoid activation to the weighted sum of h_{t-1} and x_t (Equation 11). Now the new hidden state h_t is the product of the new cell state applied through a tanh activation and the output of the output gate o_t (Equation 14).

$$\vartheta_t^o = \frac{\partial E_t}{\partial o_t} = \frac{\partial E_t}{\partial h_t}\frac{\partial h_t}{\partial o_t} = \vartheta_t^h \tanh{(c_t)},$$
$$\vartheta_t^g = \frac{\partial E_t}{\partial g_t} = \frac{\partial E_t}{\partial c_t}\frac{\partial c_t}{\partial g_t} = \vartheta_t^c i_t.$$

The error for the cell state is propagated from the current error and the cell state from the following time step. Thus, the signal to the cell state is

$$\vartheta_t^c = \frac{\partial E_t}{\partial c_t} = \frac{\partial E_t}{\partial c_t} + \frac{\partial E_{t+1}}{\partial c_t} = \frac{\partial E_t}{\partial h_t}\frac{\partial h_t}{\partial c_t} + \frac{\partial E_t}{\partial c_{t+1}}\frac{\partial c_{t+1}}{\partial c_t} = \vartheta_t^h o_t tanh'(c_t) + \vartheta_{t+1}^c f_{t+1}.$$

Finally the error for the hidden state is propagated from the gates and the cell input:

$$\begin{aligned}\vartheta_t^h &= e_t + \frac{\partial E_{t+1}}{\partial f_{t+1}}\frac{\partial f_{t+1}}{\partial h_t} + \frac{\partial E_{t+1}}{\partial i_{t+1}}\frac{\partial i_{t+1}}{\partial h_t} + \frac{\partial E_{t+1}}{\partial g_{t+1}}\frac{\partial g_{t+1}}{\partial h_t} + \frac{\partial E_{t+1}}{\partial o_{t+1}}\frac{\partial o_{t+1}}{\partial h_t} \\ &= e_t + \vartheta_{t+1}^f \sigma'(z_t^f)W_f + \vartheta_{t+1}^i \sigma'(z_t^i)W_i + \vartheta_{t+1}^g \tanh'(z_t^g)W_g + \vartheta_{t+1}^o \sigma'(z_t^o)W_o.\end{aligned}$$

4 Emulator for a process-based agroecosystem model

Quantifying the carbon balance of grasslands is important in monitoring changes in carbon emissions and effects on the productivity of grasslands. Perennial grasslands occupy from 20 to 40 % of Earths surface, depending on the used definition of a grassland [56], constituting of diverse dynamical processes and carbon fluxes. Climate change and rising CO2 concentration will affect the biomass productivity and stability of grasslands [66]. Thus, it is important to study the grasslands responses to climate change, how it will adapt to the changes and how diverse mitigation options could help.

Process-based simulators offer a systematic approach for modeling these kind of systems based on theoretical understanding of them. A process-based simulator is a computational model representing a system based on assumptions and hypotheses of the dynamics of the system. The system can often be very complex, and to develop a model for it requires an overall understanding of the system and the computations of the various processes in it.

This study focuses on a process-based grassland model BASGRA_N, described in Section 4.3, which simulates yield and biogeochemistry in grasslands, particularly carbon fluxes in plants and soil. The simulator models important factors regarding to carbon balance, two key ones being net primary production and leaf area index. Net primary production describes the balance between the carbon gained from photosynthesis and the carbon released by the plant respiration, and leaf area index describes the green leaf area of a horizontal ground surface area which affects the carbon uptake capability.

However, the model, as many other process-based ecosystem simulators, includes computations of diverse relevant fluxes. Even though it is a relatively fast model compared to many other computational models, yet the computations at a large scale remains computationally demanding. Performing simulations with several different parameter cases is often needed in cases like model calibration, which tries to find the best inputs to match observed physical data, and sensitivity analysis, where the effect of varying input parameters to output variables is studied. This is slow, as the model can run only one parameter set at a time through the heavy computations defining the simulator.

A surrogate model is constructed for addressing the computational burden of large scale simulations of carbon exchange. This can be accomplished by approximating the behavior of the simulator by a possibly simpler and computationally more efficient emulator model. Emulators are often built by using machine learning methods enabling faster simulations once trained. Net primary production and leaf area index are chosen as the target of emulation, as they are key factors in carbon exchange. For training a machine learning emulator, a suitable machine learning model and comprehensive learning data of simulator inputs and outputs are required.

Carbon exchange in grasslands is affected by many factors such as weather conditions, land management, soil and vegetation. Predicting sequential output values, daily, weekly or monthly, enables understanding seasonal patterns and affects of changing meteorological factors and management practices. Weekly averaged data is chosen, in order to get seasonal patterns, but still with the input sequence being of a reasonable size. Hence, the prediction requires input variables including soil and vegetation properties, management and weekly meteorological data, chosen by sensitivity analysis. Thus, the emulator should be able to process both sequential and static inputs. The training data can be generated by sampling soil and vegetation parameter values for diverse locations on a chosen geographical area. The meteorological data, on the other hand, can be attained from daily historical meteorological data.

In summary, the goal is to develop an emulator for the simulator outputs concerning carbon balance quantification, with the purpose of providing an alternative model for cases requiring numerous simulations. The emulator is needed for use in Finland, thus the training data should include simulations relevant for that regional area and climate. The aim is to emulate key factors concerning carbon balance including weekly carbon net primary production (NPP) and leaf are index (LAI) based on time dependent and stationary inputs. This involves constructing a model to predict sequential values with dynamical and stationary inputs. Thus, a model combining capability in learning temporal dependencies and complex patterns is needed. The following section explores several possible approaches for addressing the challenges with process-based models.

4.1 Related work

Machine learning emulation is an evolving area of research, which has gained interest in the recent years in addressing computational difficulties of large process-based models. Emulators are surrogate models, mimicking the behaviour of the process-based model, enabling computationally more efficient simulations. Machine learning methods have shown great performance in predictive modeling in various domains. Diverse techniques such as Gaussian process [64], random forest [65, 41], XGBoost [41] and neural networks [59, 16, 40] have been used in carbon cycle modeling in diverse ecosystems, using observed learning data. Similarly, machine learning techniques can effectively learn the functioning of diverse simulators, given the simulator's input-output pairs as learning data to the emulator. This literature review goes through various emulation approaches beginning with an overview of machine learning emulator methods used in ecological research, then moving to specific techniques more suitable for sequential tasks. And finally, a comparative analysis is given, highlighting the most promising and suitable methodologies for the task of this thesis.

Diverse machine learning emulators have already been widely used for different purposes in ecological research. A study using multiple machine learning approaches [72] tries addressing the challenges with the computational cost of the agricultural production system simulator *APSIM*, which is applied to simulate soil organic carbon and crop yield over the period 1980–2080 under various agricultural management practices. The study focuses on emulating the overall change in soil organic carbon and average yield over the simulation period. The study tries several machine learning methods for the emulation; multivariate adaptive regression splines (MARS), random forest (RF) and boosted regression trees, which all explained $> 92\%$ of the variance of the simulation of soil organic carbon, wheat yield and maize yield. RF [4] is seen to outperform the others in the emulation task with the most accurate performance. The ML model inputs are initial soil organic carbon content, mean annual temperature and precipitation (for future events predicted with global circulation models), and/or variables indicating management practices. The predicted meteorological values are seen as the least important factor in the simulated system, which the study states to possibly mean that the future change of temperature and precipitation has limited effect on the soil organic carbon. Also the study does not consider climate extremes in the projection of future climate.

Also Gaussian processes have been a popular approach in diverse emulation tasks [1, 11, 19, 20]. Gaussian process (GP) is a probabilistic model, capable of interpolating observed model runs and provides a probabilistic prediction [11]. Studies [1] and [11] have a Gaussian process approach for emulation of a land surface model and a climate model, respectively. The study attempting to emulate a dynamical land surface model [1] develops separate emulators for diverse plant functional types to emulate GPP predictions on parameters including plant physiology, radiative properties, seasonal responses, climatic values and other attributes. The temporal structure of the internal state variables of the simulator is accounted for in the emulators by using 8-day averages of the data. For controlling long-term dependencies a variable that records the day of the year is added to the input. To reduce the dimensionality of the output, it is broken down into a large set of one dimensional outputs. The removal of the temporal structure is often a simplification of the model. As the study [1] noted, Gaussian processes are not well suited for handling large data sets, as they require the inversion of an $n \times n$ covariance matrix. However, the study uses a method for mitigating the computational issues of GPs by training the emulator with much smaller data set where the input points can be placed at key locations in the input space. Gaussian process emulation has also been used in dynamical terrestrial ecosystem model calibration, focusing on soil respiration and net ecosystem exchange and latent heat flux [19]. GPs have also been used to emulate a land surface model for sensitivity analysis of soil moisture to static uncertain parameters [20].

A study based on Gaussian processes [10] introduces different approaches for emulation of dynamic models and multi-output with Bayesian emulation, which are then applied for a global vegetation model. The model simulates carbon dynamics of forests and other kinds of vegetation, producing time series of outputs. Three GP-emulation approaches are compared; One GP method uses multi-output (MO) emulator, another uses multiple single-output emulators (MS) and the third is based on treating time as an auxiliary input (TI). The MS method, applied in this study, fails to address correlations through time. Overall, compared to the TI approach, the MO method is seen to give better predictive performance, produce narrower prediction bounds and have a more reliable covariance estimation. However, the study points up that neither of MO or TI are capable to decently capture the correlation structure featured by the simulator through time. The study also emphasizes the difficulty to identify a suitable GP emulator for dynamic simulator outputs.

Moreover, deep learning has shown great potential in ecosystem model emulation [49, 12, 45]. Deep learning is a kind of representation learning, meaning it can be fed with raw data and automatically learn the needed presentations [37], while many other machine learning methods require some feature extraction from raw data, based on domain knowledge [60]. With minimal assistance, deep learning can capture complex patterns in data [37]. Feed-forward neural networks have been widely used in cases with non-sequential inputs [49, 12]. Once trained, deep learning emulators can achieve simulations orders of magnitudes faster than the process-based simulator, while maintaining a high level of accuracy [55]. A study using deep learning methods [49] emulates a simulator of biogeochemical flows of carbon, nitrogen and water in diverse ecosystems. A FNN emulator is constructed for predicting 30-year average values of corn grain, stover yields and the soil organic carbon at the end of the simulation period (30 years). The primary inputs of the simulator include daily maximum and minimum air temperature and precipitation, soil properties, land use data and management practice. For the emulator the meteorological values are aggregated into growing season values and averaged over the simulation period. Thus, no dynamical values are given to the emulator. The one layer FNN performs well in this task with 99% capture of the variations of the simulator. FNNs perform well also in an emulation task for predicting annual mean spatial variability in carbon and water fluxes, given biophysical parameters as inputs [12]. The study constructs a two layer FNN for emulating gross primary production and latent heat flux outputs of a simulator land model, giving a $> 86\%$ capture of the variations of the simulator.

A study combining different machine learning techniques [45] aims to emulate carbon cycle processes in an Earth system model in order to perform large regional ensembles. The study develops a surrogate model combining singular value decomposition (SVD), Bayesian optimization, and a FNN to simulate carbon cycle processes relevant for Earth system models. The model outputs include annual GPPs for many different locations over 30 years. The study states that reducing model output dimensions is crucial when working with a limited sample size. Also a simpler network is faster to evaluate as the weight matrices are smaller. The Earth system model output dimensions are reduced by SVD to improve the computational efficiency of training and evaluating a FNN surrogate. The study states that neural networks involve hyperparameters that largely affect the model performance, thus they are optimized by the Bayesian algorithm. Also it mentions that NNs suffer from high computational cost when including many quantities of interest. The emulator inputs compose of static parameters describing plant growth and development. The emulator performs well, capturing $> 96\%$ of the variations of the simulator.

Diverse emulation methods are explored by a comparative study [31] attempting to emulate a crop model *APSIM*. The study focuses on FNN, RF and MARS and compares their performances on simulating chickpea crop with static inputs including climatic, soil, phenological and management data. The study shows the RF as the most consistently accurate method compared to the other approaches, though with computationally demanding requirements. The study indicates that RF tends to take the longest to train, and MARS the fastest.

Another comparative study [68] shows a strong performance of RF compared to many other methods, now in a different emulation task. The study tries several emulation approaches for simulation of N2O fluxes and N leaching from corn crops with static input related to N input, soil and climate. Diverse linear and non-linear statistical methods are compared, including FNNs, RF, support vector machines

(SVM) [30] and splines and kriging. The study concludes that spline methods perform best with very limited data and with larger data the RF and the SVM outperform the others.

In addition to emulation, hybrid-models incorporating machine learning methods into a process-based model, often called knowledge-guided machine learning (KGML) models, are promising in addressing challenges occurring in complex process-based models applied to large regions at high spatio-temporal resolution. A study based on KGML [18] implements a process-based crop model (*APSIM*) and the random forest algorithm for yield forecasting, using growth stage-specific information. The process-based model is used for simulating the dynamical phenology and biomass values, with input data including daily meteorological data, daily drought index, soil hydraulic properties and wheat trial data. The predictions are then given to the random forest model for the yield estimates. Also the performance of multiple linear regression is tested in place of the RF, but the RF outperforms it. The study highlights the effect of drought events throughout the growing season as they are identified as the main factor causing yield losses. Another study using hybrid methods [61] introduces a model for predicting the LAI and NEE of slash pine forests. A feed forward neural network model generates estimates of LAI, giving RMSEs less than 0.50 and R^2 of 0.77, utilizing ground plot and satellite data. Following that, the LAI predictions are used as inputs to a slash pine simulation model to make NEE predictions.

FNNs process each input independently, thus they can't capture temporal dependencies in sequential data. Hence, FNNs would not be suitable for the task including temporally dependent inputs and outputs. However, neural networks with a recurrent design could possibly be suitable for that case. Dynamical emulation tasks involving sequential, time dependent inputs and/or outputs have often been approached by diverse recurrent neural networks, such as LSTM or its simplified variant GRU [8], both designed for learning temporal dependencies. LSTM and GRU type recurrent neural networks have been used in diverse emulators and hybrid-models, such as hydrological models [46, 53], crop production models [69], and agroecosystem models simulating greenhouse gas dynamics [42, 43].

Furthermore, more recent studies incorporate recurrent neural networks with process-based models, to improve the performance of agroecosystem models [42, 73, 69]. A study based on KGML [42] develops a model *KGML-ag-Carbon*, which combines a process-based agroecosystem model *ecosys* and a gated recurrent unit GRU. The KGML model addresses key carbon budget factors including ecosystem autotrophic respiration (Ra), ecosystem heterotrophic respiration (Rh) and net ecosystem exchange (NEE) on a daily scale and yield on an annual scale. First, synthetic data are generated for a crop field to train the GRU model. The inputs of the simulator include soil information, planting and harvest dates, crop parameters, crop rotation information and daily climate data. This data is then used to pretrain the GRU model, after which the model is fine-tuned by using observed low-resolution crop yield data and carbon fluxes from sparsely distributed eddy-covariance sites. The GRU model employs five modules including four GRU layers and a FNN with an attention mechanism, one GRU layer serving as the foundational layer, while each of the remaining GRU layers contribute in predicting one of the daily outputs. The annual yield prediction is produced by a FNN with an attention mechanism which learns the importance of each time step by additional weights, enabling learning only values which affect the yield. The system works hierarchically, where the outputs of some modules become inputs for others. The study shows the model capturing 86 % more spatial detail of soil organic carbon changed than conventional coarse-resolution approaches. The study focuses on corn and soybean production in U.S. Midwest. Another study based on a similar approach [73] of emulates the *ecosys* model with GRU for KGML data assimilation in carbon budget quantification. Furthermore, another similar approach [43] was used in estimation of nitrous oxide (N2O).

A comparative study [69] integrates data assimilation and machine learning methods for regional daily summer maize LAI prediction. Several ML methods are compared, including random forest, LSTM and support vector regression. The study chooses the three methods for investigation because LSTM is known for handling data with long-term dependencies, SVM demonstrates strong robustness in small sample regression tasks and RF exhibits good resistance to noise and outliers in data. The study shows best performance by LSTM with data from last 15 days to predict the subsequent 15 days, while RF and SVM struggled in predicting time series of LAI.

A case worth mentioning of a observed data-driven ML model is a study based on LSTM [44], where the goal is to create a machine learning model for crop prediction with dynamical meteorological and static inputs. The study compares several methods; XGBoost, RF and three variations of LSTM, of which one of the LSTM variations gives the best performance. The study uses a LSTM layer for processing the sequential data and regular FNNs for the static inputs and then the outputs are combined with an attention mechanism, which differentiates the contribution of different periods to the yield. Lastly the combined output is given to a FNN for creating the final prediction. The other LSTM variations use different combination techniques for the static and dynamic data, by another concatenates the processed outputs and other uses the FNN processed data as initial hidden state for the LSTM layer. Nevertheless, the model predicts the yield for a whole year, thus the model does not give sequential predictions, which is needed in the task of this thesis. The LSTM model is shown to explain 73% of the spatio-temporal variance of an observed maize yield, while a widely used regionally calibrated process-based model explains only 16%.

There are a few possible methods for approaching the problem of this thesis. Some key studies on process-based model emulation, requiring only static inputs, are discussed above. SVM and especially RF appear to perform well for many tasks with static inputs [68, 31, 18, 72], even better than FNNs. However, these approaches are mostly applied to static emulation tasks, thus they may not be so relevant, as the problem addressed in this thesis requires dynamic meteorological inputs. However, the comparative study [69] studies RF and SVM performance in predicting LAI time series, and shows the superior performance of LSTM, compared to performances of RF and SVM. Some approaches used averages or sums of meteorological inputs over long times instead of sequential meteorological data of shorter periods, leading to overly simplified weather conditions, which complicate detecting some weather events and their impact on the ecosystem.

Alternatively, as mentioned previously, Gaussian processes have successfully been applied for both static and dynamic tasks. A comparative study [48] investigates the GP and NN methods in computer emulation and calibration of physics based simulator of domains like cosmology, nuclear theory, and materials science, concluding that GP performs overall well. The study compares the performance of the emulator methods for models of unrelated systems for this thesis, but the analysis can be partly generalized as the focus is on emulating process-based simulation of systems, including many computations. However, for each of the emulation tasks, the training data includes of no more than 1000 simulations, which gives poor circumstances for NN to learn effectively as they excel in learning behaviour in large data sets, allowing them to capture a wide variety of potential behavior [48]. In addition, the study does not compare the GP and recurrent networks for sequential prediction, which would be beneficial for the problem of this thesis. Also the simulation outputs change relatively smoothly as a function of the inputs, which is beneficial to GP as it fits well for Gaussian process assumption. With more discontinuous output behavior, GPs would likely struggle.

Furthermore, GPs have also been used in emulation tasks including sequential data [19, 10]. An advantage of the GPs is that they provide uncertainties of their predictions. Also GPs are capable to accurately fit complex data patterns, while providing smooth predictions. Gaussian processes may perform better in some regression tasks with limited data, but with larger data sets neural networks can learn the complex patterns well. However, GPs presented difficulties in learning temporal dependencies [10]. The problem addressed in this thesis likely requires capabilities in processing large data sets, as the input parameters include sequential meteorological parameters in addition to static ones.

Deep learning methods have shown great capabilities in learning complex patterns, particularly in larger data sets. Generally LSTMs have proved to perform well in many previously presented sequential tasks with temporal dependencies. The GRU is also used by some studies, but comparative analyses with other methods have not been presented. LSTM is more frequently used, as GRU is a relatively new method which has been shown to have a comparable performance to LSTM [9]. Compared to the problem addressed in this thesis, a few studies with similar tasks with both dynamic and static inputs [42, 44] have taken an approach of recurrent networks. Combining learning from static and dynamical data, through neural networks, can be done by developing a network with both FNNs and LSTMs/GRUs [44].

As discussed before, emulators are only approximations of the process-based simulator model, leading to a simplified model excluding some possibly important details of the system behavior. The process-based model is based on the human understanding of the dynamics of the system of interest. Thus, as a model of a process-based model, the emulator is not a perfect representation of the real life system. However, they can be very beneficial in cases where many simulations of the computational model are needed, such as calibration [19] and sensitivity analysis [20]. When emulating with deep learning methods it is difficult to understand how the trained deep learning model is obtained, which is why a NN model is often called a "black box". Although, the model is trained based on the process-based model's computations, which are based on theoretical understanding of the simulated system.

4.2 The simulator model

The simulator model, the target of emulation, is called the BASGRA_N model [27]. It is a process-based model for managed grasslands, designed to perform daily simulations of the impact of N-supply on the plants and their environment, the fluxes of greenhouse gases in plants and soil, and the harvest yield and the digestibility of leaves and stems. The model includes variables for carbon in the above and below-ground plant part, plant phenology and soil water and carbon and snow depth [27].

The model simulates plant biomass on a daily time step with meteorological variables, including radiation, temperature, precipitation, wind speed, humidity, atmospheric CO2 concentration and management events [27].

4.3 Simulator data generation

Simulations of the BASGRA_N model are used for training the emulator. In this thesis the consideration is on emulating carbon net primary production (NPP) and leaf area index (LAI) with time dependent and static inputs.

The choice of emulating LAI and NPP is due to their key role in agroecosystem carbon dynamics. Net primary production is a key variable considering carbon balance, describing the net amount of carbon gain over a time period, in other words the balance between the carbon gained by photosynthesis and the carbon released by the plant respiration [5]. The leaf area index is defined as one half the total green leaf area per unit of horizontal ground surface area [17] and is one of the most important variables observable through remote sensing, enabling validation at a broad scale. The LAI has also been used in inversion modeling of NPP [52].

The simulator is given daily values of the dynamical inputs and the stationary input values, listed in Table 1, and as outputs it provides daily values of NPP ($kg\,C\,m^{-2}s^{-1}$) and LAI (m^2m^{-2}). The time dependent inputs include meteorological values and the harvest carbon flux, giving the dates of harvest which is chosen to be the same for each sample. The static inputs describe the vegetation and soil properties.

The meteorological data is obtained from ERA5 [24] data, produced by the European Centre for Medium-Range Weather Forecasts (ECMWF). ERA5 is a reanalysis for global climate and weather from 1940 to present and it provides hourly estimates for a large number of atmospheric, ocean-wave and land-surface quantities [24].

As the model is meant for use in Finland, it requires training data representative of the regional characteristics. Thus, the training set should include similar data points characteristic of the region, but also incorporate some variance. The data points are selected to be within Europe and filtered based on weather (air temperature and total precipitation) conditions, to exclude the hot summer-climates like the regions of hot-summer Mediterranean climates.

The data points are evenly spaced across the chosen geographical area. The spacing for latitudes is $0.25°$ and the spacing for longitudes is $0.25° \cdot 1/cos(\phi)$, where ϕ denotes latitude in radians. A scaling factor $1/cos(\phi)$ [62] is introduced to ensure that the data points are distributed evenly across the area by taking the curvature

Table 1: Model inputs: first six are the time dependent values and the rest are stationary values.

Component	**Description**	**Unit**
air_temperature	Air temperature	K
relative_humidity	Relative humidity	$\%$
surface_downwelling_-shortwave_flux_in_air	Flux of solar radiation reaching the surface	$W\,m^{-2}$
precipitation_flux	Precipitation flux	$kg\,m^{-2}s^{-1}$
wind_speed	Wind speed	ms^{-1}
harvest_carbon_flux	Carbon flux during harvesting	$kg\,C\,m^{-2}s^{-1}$
K	Light extinction coefficient	m^2m^{-2}
LAICR	LAI above which shading induces leaf senescence	m^2m^{-2}
LAITIL	Maximum ratio of tiller and leaf appearance at low leaf area index	
ROOTDM	Initial and maximum value rooting depth	m
LAT	Latitude	$^\circ N$
RDRTMIN	Minimum relative death rate of foliage	d^{-1}

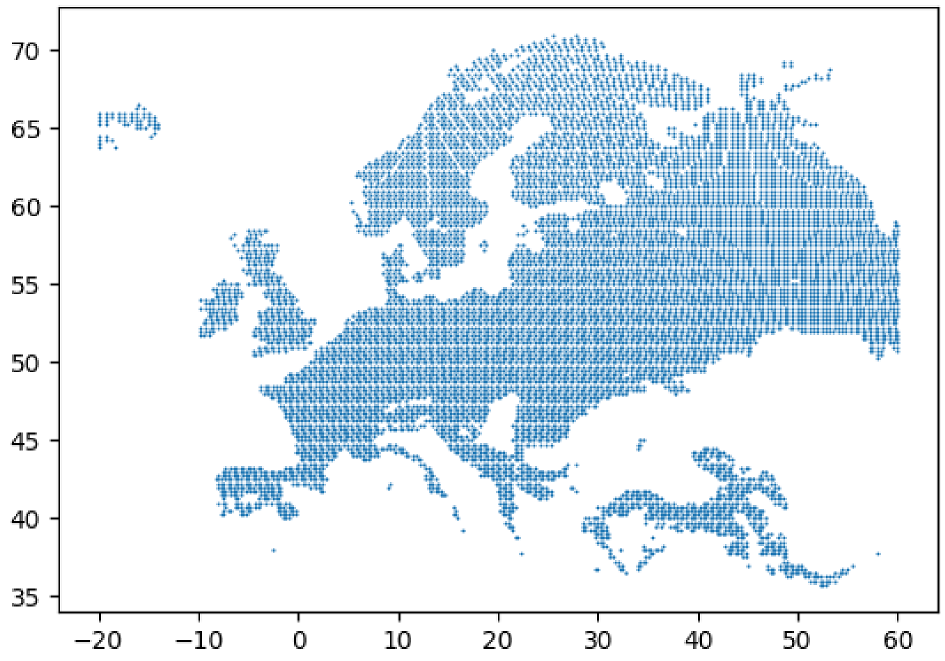


Figure 7: Selected data points across Europe.

of the earth into account. By the preceding filtering, 10085 data points are selected for training the emulator. The selected data points are shown in Figure 7.

The simulator encompasses numerous parameters on vegetation and soil properties, which are reduced to 6 most contributing parameters in NPP and LAI prediction. The vegetation property parameters are selected through a variance based sensitivity analysis, as described in [36]. Each data point gets randomly sampled parameter values from specified distributions. A uniform distribution is used for most of the parameters and for light extinction coefficient a beta distribution is used. The distributions and shape parameters for each parameter are listed in Table 2.

Each data point corresponds to a specific year-long time period from May-to-May. For example a data point d could correspond to the period from 1/5/2022 to

Table 2: Light extinction coefficient (K) values are sampled from beta-distribution and the other parameters are sampled from uniform distribution. The parameters of the distributions are denoted as a and b.

Component	**Distribution**	a	b
K	Beta distribution $Beta(a, b)$	7.000	3.000
LAICR	Uniform distribution $U(a, b)$	2.000	7.000
LAITIL	Uniform distribution $U(a, b)$	0.300	1.100
ROOTDM	Uniform distribution $U(a, b)$	0.500	1.500
RDRTMIN	Uniform distribution $U(a, b)$	0.008	0.012

30/4/2023. This way the period starts from the beginning of the growing season. The data point d corresponds also to a location and parameter values: meteorological values for each day of the time period and the sampled parameter values describing the vegetation properties.

The data points are divided to five separate subsets with separate time periods and locations for the purpose of performing cross-validation. Each subset includes 2017 locations and a set of time periods which is a randomly chosen one-fifth of the year range 1940-2023. Each of the 2017 data points of a subset is given a random time period from the corresponding set. A total of 10085 simulations are performed, by using the preceding sampling.

4.4 Emulator data generation and pre-processing

The training data for the emulator is generated from the simulator's daily input and output values. For the usage of the emulator weekly, values are suitable, which allow increased efficiency and enables better generalization performance of the model as there are less inputs leading to less weights [2], as discussed in Section 2.6. The weekly means of the hourly meteorological input values are calculated and used as inputs to the emulator model. And the output values, or targets, for training the emulator are derived from the weekly means of the simulation outputs. The vegetation property parameters remain the same and the data, including the corresponding targets, remains to be divided to the previously defined folds.

The input parameters and target values are standardized separately to have mean of zero and standard deviation of one, for all the inputs initially to be treated equally to prevent overfitting, which is described in Section 2.6.

4.5 Model description

A model consisting of regular feed-forward neural networks (FNN) and long-short term memory (LSTM) network is constructed. The characteristics of the stationary parameters are learned by a FNN and the sequential input is learned by an LSTM network. The choice of neural networks is due their ability in capturing complex patterns in data with limited assistance [37]. Especially incorporating LSTM networks was chosen due to their ability to learn temporal dependencies.

Figure 8 demonstrates the emulator models architecture. The emulator consists of one LSTM layer and multiple dense layers. The model takes the meteorological variables and vegetation property parameters as inputs. The input describing meteorological and management events consists of the weekly meteorological values and the harvest dates. Thus the input consists of 52 vectors of size 6, notated by m in Figure 8. The vegetation property input vector, noted by p in Figure 8, includes the 6 corresponding parameters.

The vegetation property input vector is passed through two dense layers with ReLU activation. The output is two dimensional and replicated by the number of time steps along a new dimension in order to be able to concatenate it with the 3 dimensional LSTM output. The meteorological input vector is fed to the LSTM layer, which loops through the cell for the 52 time steps, each time step getting the

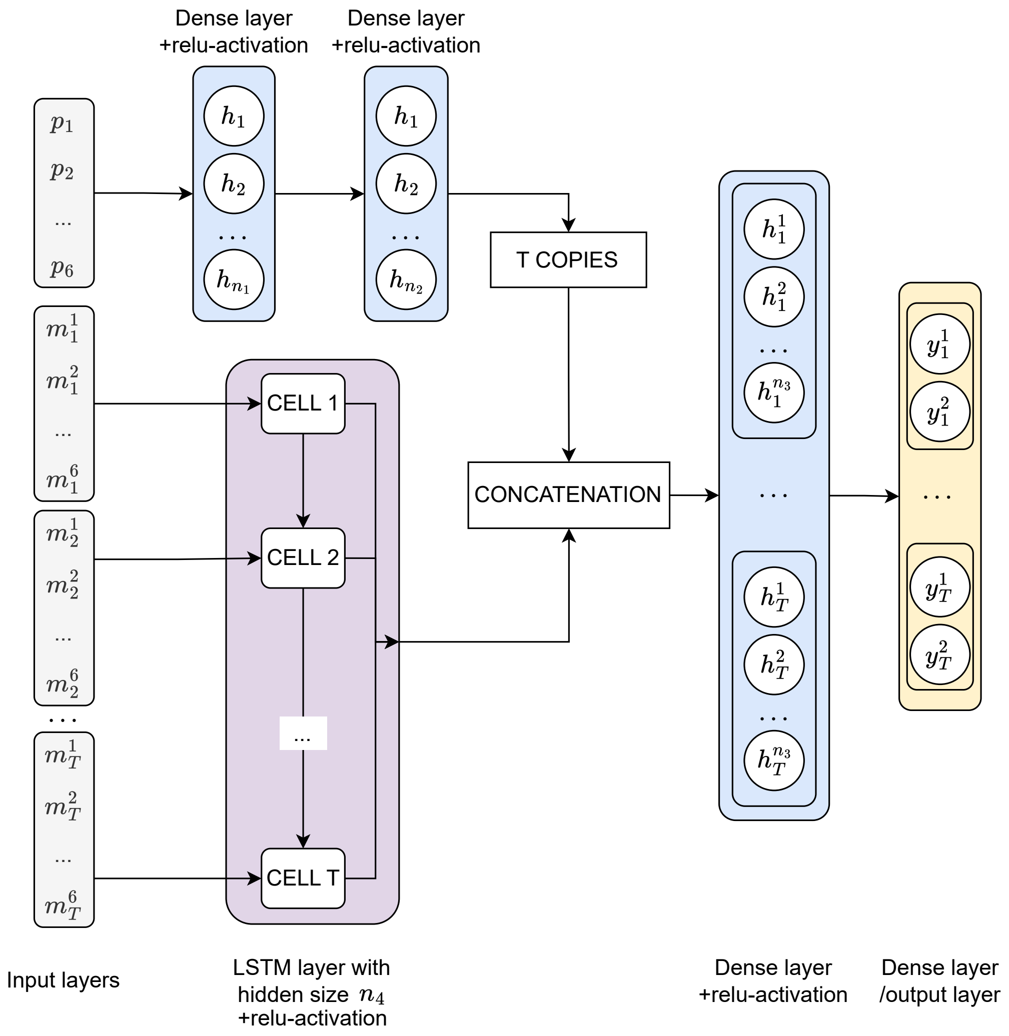


Figure 8: LSTM-FNN model architecture: the gray boxes present the inputs, blue represent the dense layers, violet the LSTM layer and yellow the predictions or output of the model. T denotes the number of time steps, h the hidden states, y the output vector and m indicates the time dependent inputs and p the static inputs.

corresponding meteorological input vector and the hidden state and the cell state from the previous time step as inputs. The hidden states of each loop are collected as the output of the LSTM layer, thus the output has again 52 vectors of hidden states.

Next, after the static inputs and dynamical inputs are processed separately, they are concatenated along the last axis to combine the meteorological and vegetation property information. This concatenated array is then passed through another dense layer which produces the final predictions of LAI and NPP per each time step.

4.6 Training and validation

The model is trained by applying backpropagation as demonstrated in Section 2.4 for the dense layers and in Section 3.3.2 for the LSTM layer. The weights are updated by utilizing the Adam-optimization algorithm, introduced in Section 5, with addition of weight decay in order to prevent overfitting, as discussed in Section 2.6. The Algorithm 1 is used for the training of the model, looping through the training set for a specified number of epochs and batches of a specified batch size. For each batch of an epoch, the algorithm goes through the network and then performs backpropagation and weight update. The forward pass of a batch through the network includes the following steps, as described in the previous section.

The algorithm starts on initializing the hidden states of each of the layers and the cell state of the LSTM layer. All the states are initialized to small random values in order to prevent symmetries in the network, as discussed in Section 2.5. The LSTM cell loops through the sequential input one time step at a time, each step updating the cell state and the hidden state. The hidden states are captured at each time step to produce the output of the LSTM layer. The parameter input goes through the dense layers and ReLU activation functions, and is replicated by the number of time steps, and finally is concatenated to the LSTM output.

After the batch has been processed, the output is compared to the target values, by calculating the mean squared error of the output. The train error is fed to the backpropagation algorithm, which sends the error signal through all the layers from last to first as described in Sections 2.4 and 3.3.2. The error signal is then given to the optimization algorithm Adam which then calculates the updated model parameters consisting of weights and biases. Subsequently, the next batch is processed similarly with the updated model.

Lastly the trained model's performance on new data is measured. Thus, the model makes predictions based on inputs provided by the test set. The predictions are then again compared with the targets of the test set with mean squared error. This procedure is done for each of the fold divisions and the generalization error is then estimated as the mean of the test errors across folds.

Algorithm 1: Training algorithm for a LSTM-FNN model with BPTT and Adam optimization.

```
input  : Model with LSTM layer $C_{LSTM}$, activation functions of the LSTM
         cell (sigmoid and tanh), fully connected layers $Dense^{6 \to n_1}$,
         $Dense^{n_1 \to n_2}$, $Dense^{n_2+n_4 \to n_3}$, $Dense^{n_3 \to 2}$, ReLU activation
         function, training data in batches of size $b$ and number of epochs.
output : Model with trained weights.
for epoch in number of epochs do
    for batch in number of batches do
        initialize hidden states of all dense-layers:
        $h_{d,0} \leftarrow random \approx 0 \in \mathbb{R}^{b \times n_d}$ for all dense-layers $d$
        initialize LSTM cell state and hidden state:
        $c_0 \leftarrow random \approx 0 \in \mathbb{R}^{b \times n_4}$
        $h_{c0} \leftarrow random \approx 0 \in \mathbb{R}^{b \times n_4}$
        for time step $t$ in number of time steps $T$ do
            $(h_{ct}, c_t) \leftarrow C_{LSTM}(h_{c,t-1}, c_{t-1}, m_t)$
            $H_m \leftarrow H_m + h_{ct}$
        end
        $H_m \leftarrow ReLU(H_m)$
        $h_1 \leftarrow Dense^{6 \to n_1}(p)$
        $h_1 \leftarrow ReLU(h_1)$
        $h_2 \leftarrow Dense^{n_1 \to n_2}(h_1)$
        $h_2 \leftarrow ReLU(h_2)$
        $H_p \leftarrow h_2 \times T$
        $H \leftarrow H_m + H_p$
        $h_3 \leftarrow Dense^{n_2+n_4 \to n_3}(H)$
        $h_3 \leftarrow ReLU(h_3)$
        $h_4 \leftarrow Dense^{n_3 \to 2}(h_3)$
        $y \leftarrow h_4$
        $MSE_{train}(y, \tilde{y})$
        BPTT($MSE_{train}$)
        Adam weight update
    end
end
return trained model
```

The network and the training algorithm require several hyperparameters, including hidden sizes, batch size and learning rate. Hyperparameter tuning is a crucial part of the training process, which includes experimenting multiple variations of the model. Several variations of the the depth of the network are explored with the intention of developing a relatively simple model for computational efficiency and good generalization performance.

The hyperparameters, in addition to the depth of the model, consist of learning rate, batch size and the sizes of the hidden states of the layers. In Section 2.6 it was stated that choice of small batch size is good in general, but it is beneficial to explore it by grid search. The batch size also affects the memory, and especially GPUs offer better runtime if power of 2 batch sizes are used [22]. This can be generalized also

Table 3: Hidden sizes for each layer: n_1, n_2, n_3 denote the hidden sizes of the first second and third dense layer, respectively, and n_4 denotes the hidden size of the LSTM cell, n_5 denotes the hidden size of the last dense layer, the outputs layer which is 2 as two values are predicted.

1. dense layer (n_1)	256
2. dense layer (n_2)	32
3. dense layer (n_3)	256
LSTM layer (n_4)	128
4. dense layer / output layer (n_5)	2

to hidden sizes.

Grid search, covered in Section 2.6, is applied in order to find the optimal hidden sizes for the layers. The chosen grid is $[16, 32, 64, 128, 256]$, consisting of values of only of powers of 2. The hidden sizes are chosen to be relatively small for faster execution time and better regularization. The model is trained with each of the hidden size combinations, and the model performances are compared with cross-validation.

Multiple values for batch size and learning rate are also tested: $[1, 4, 8, 16, 32, 64]$ for batch sizes and $[0.1, 0.01, 0.001, 0.0001]$ for learning rate. Batch size of 1 significantly increased execution time. The optimal values values for batch size and learning rate for the Adam algorithm are 32 and 0.001, respectively. And the selected hidden sizes are listed in the following table.

The number of epochs, defining the iteration times, is selected by finding the minima of the models generalization error with respect to epochs. Figure 9 shows the generalization error of each of the model per epoch. The training should be stopped before the generalization error starts to increase, which is at 78 epochs.

Figure 10 shows the progress of train and test RMSEs throughout the training process. At the beginning, with increased epochs, the train and test RMSEs decrease significantly. At some point the test error converges to a stable point, and with more training epochs the test error increases slowly. The train error keeps on decreasing. The training should be stopped at the point where the test error is at minima, otherwise the network overfits to the training data and can not generalize to other samples.

4.7 Model performance

Now that the optimal hyperparameters have been found, the model is trained and evaluated separately with data from each of the folds. The mean RMSEs across folds are 0.322 ($m^2 m^{-2}$) for LAI and 0.0290 ($g\, C\, m^{-2} h^{-1}$) for NPP.

Other way of predicting the model performance is the coefficient of determination R^2. It is defined as the quotient of the explained variation to the total variation [6]:

$$R^2 = 1 - \frac{SSE}{SST} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2},$$

where SSE is the sum of squared errors between the predictions $\hat{y}$ and the targets y and SST is the total sum of squares representing the sum of the squared differences

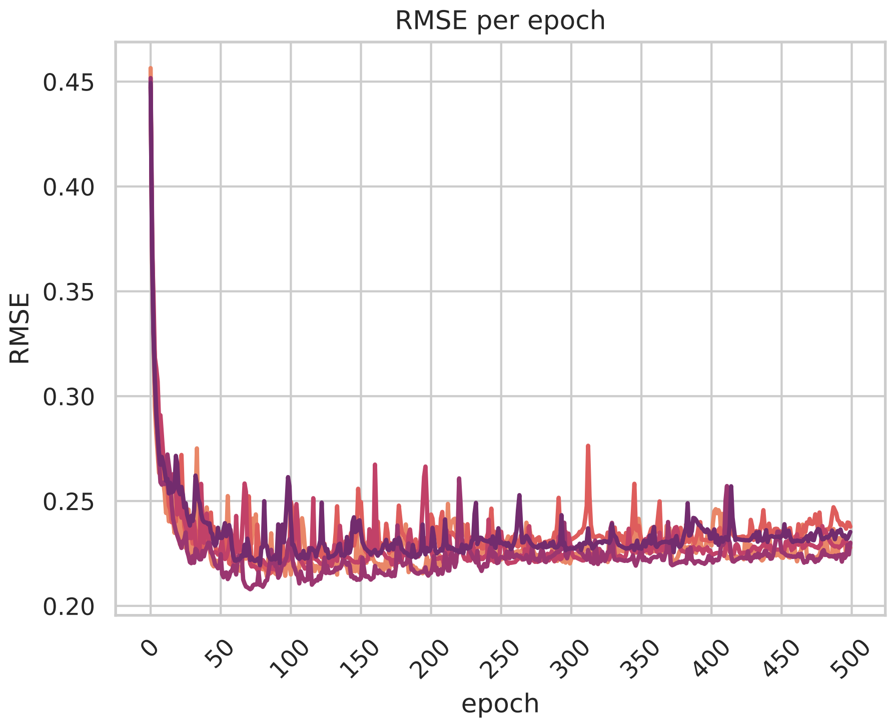


Figure 9: The normalized validation RMSEs for 500 epochs representing the accuracy of the models across both NPP and LAI values for each epoch. The minimum RMSE is achieved after 78 epochs.

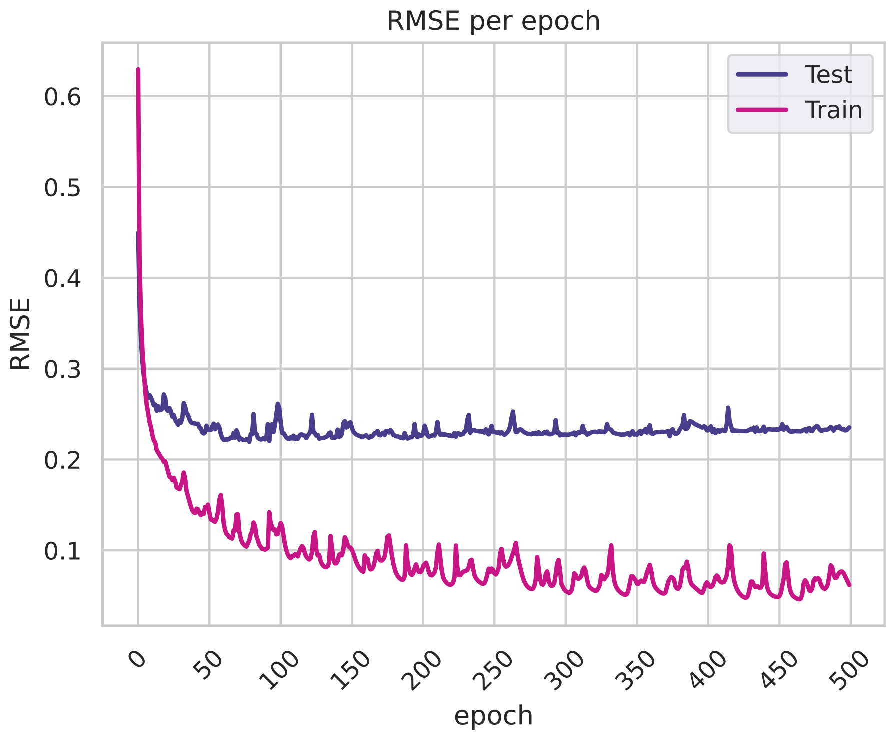


Figure 10: Train vs. test RMSEs per epoch for one fold, presenting the normalized RMSE across NPP and LAI predictions.

Table 4: R^2 and RMSEs per each fold.

Fold	**NPP R^2**	**NPP RMSE**	**LAI R^2**	**LAI RMSE**
1	0.918	0.0303	0.937	0.331
2	0.918	0.0290	0.937	0.328
3	0.922	0.0287	0.938	0.327
4	0.927	0.0282	0.947	0.298
5	0.923	0.0288	0.939	0.328

Table 5: Annual performance for NPP per each fold.

Fold	**Annual NPP RMSE**	**Annual NPP R^2**	**Mean Annual NPP prediction**	**Mean Annual NPP target**
1	76.5	0.979	-836.6	-854.9
2	79.5	0.976	-862.6	-839.4
3	74.2	0.978	-827.9	-821.3
4	71.9	0.981	-849.9	-869.2
5	85.9	0.974	-801.5	-849.0

between the targets and the mean of the targets $\bar{y}$. A R^2 value of 0 indicates that the regression model does not explain any of the variability of the target variable and 1 indicates perfect explanation of it's variability.

All folds give R^2 of > 0.93 for LAI and for NPP > 0.91. The separate R^2 and RMSE values for each fold are listed in Table 4. Mean annual RMSEs for NPP have values of less than 86. The RMSEs and R^2s for each fold are listed in Table 5 with the corresponding mean annual NPP predictions and targets.

Figure 11 combines six scatter plots, of NPP and LAI for first, second and third 4-week periods, starting from May. This 12 week period includes the beginning of the growing season and one harvest and weeks of growth after it. The emulator predictions are contrasted with the simulator predictions for these three periods. The plots indicate a high level of accuracy of prediction for the given 12 week period, as the values align closely to the $x = y$ line.

Figure 12 combines two line plots of NPP and LAI of three random examples from the validation set. The emulator predictions and simulator predictions are compared over the course of the simulation period of one year from the beginning of May to the end of April of the following year. The sudden decreases in the values, or increases for NPP, result from the harvests. The predictions seem accurate in the beginning of the simulation time period and after the growing season the predictions get less precise when going towards the end of the simulation period. Obviously the three examples can't be generalized to the whole set of predictions and the predicted features have much smaller values regarding to winter, thus they are not as relevant as the values of the growing season.

A more comprehensive visualization of the evolution of RMSE through time across each fold is shown in Figure 13. A similar performance can be seen throughout the folds. Both NPP and LAI RMSEs increase at the start of the period, then suddenly

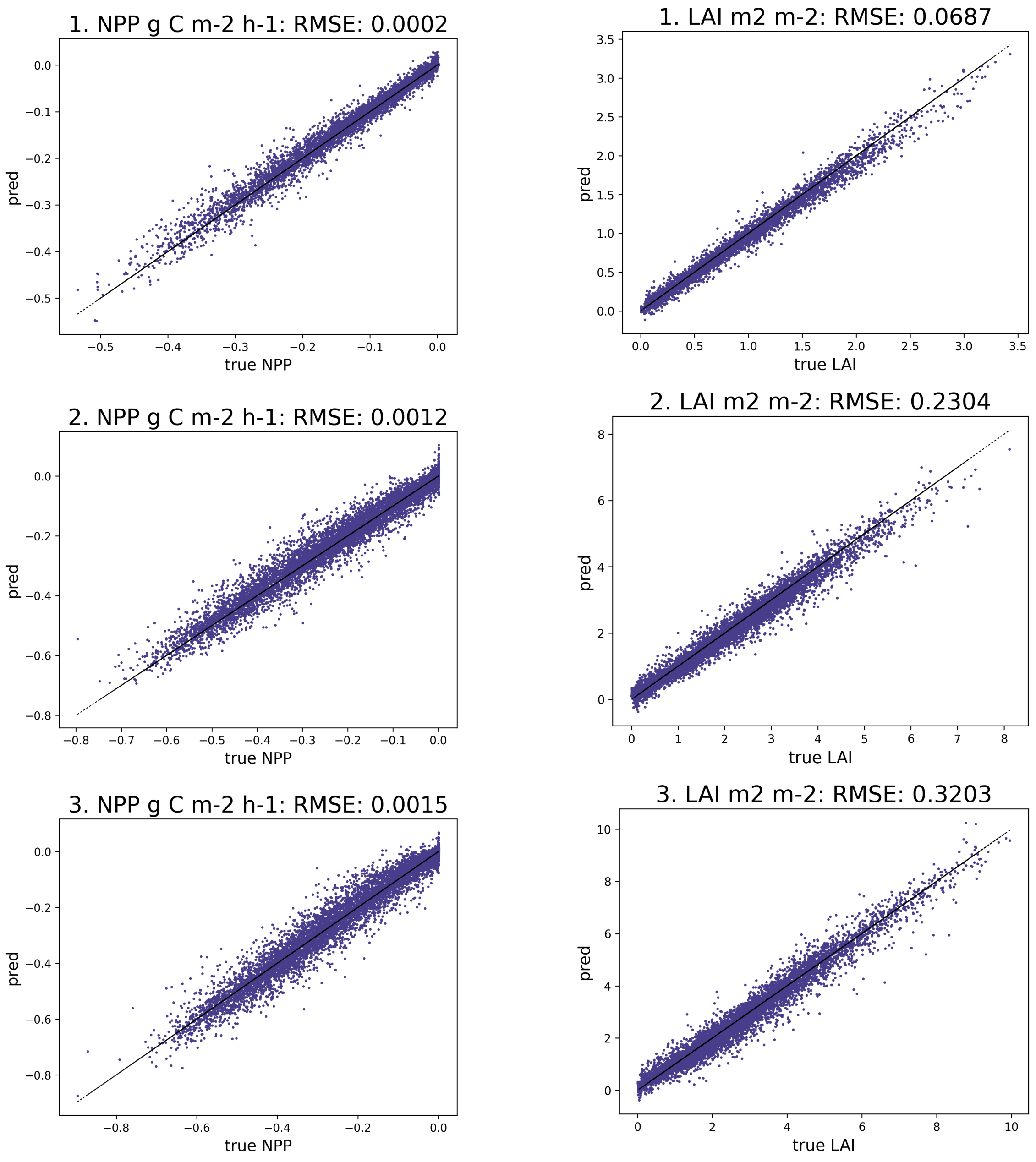
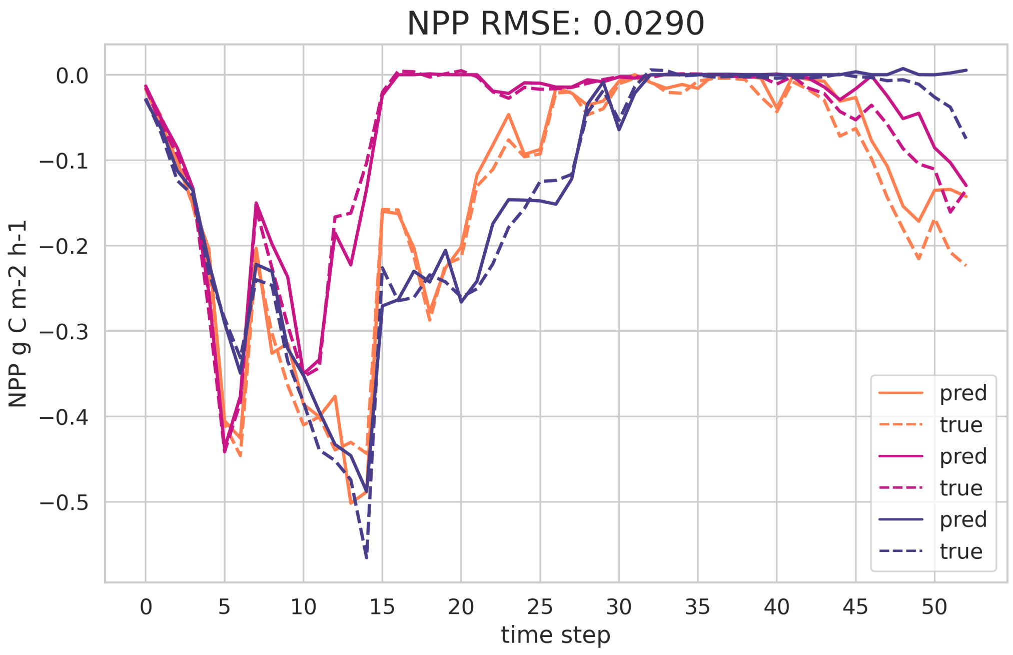
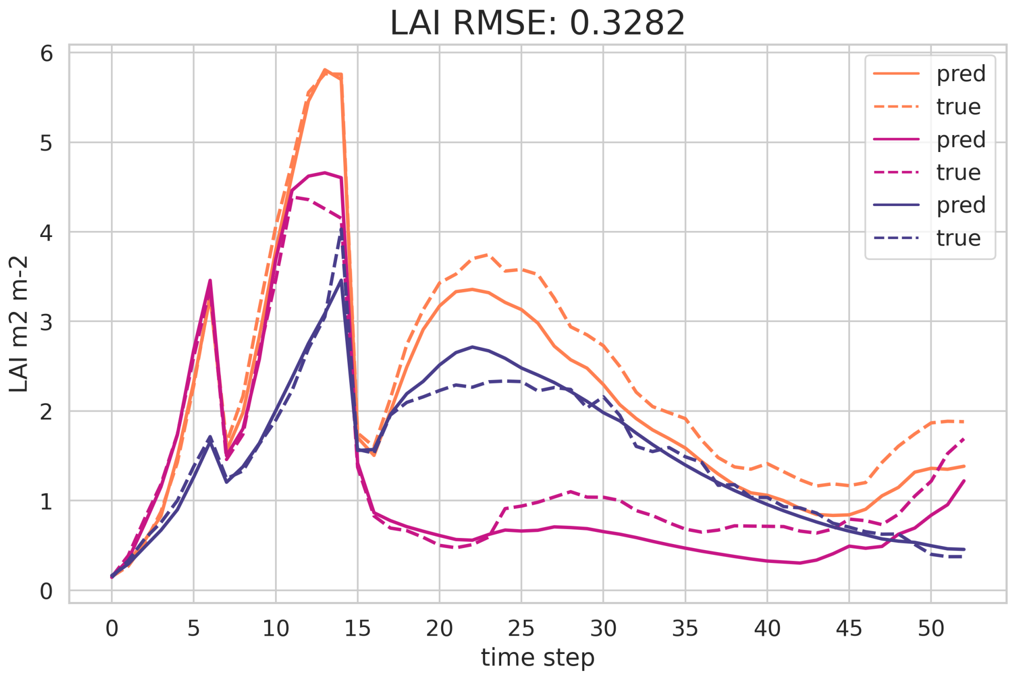


Figure 11: Scatter plots describing the relationship between targets (simulator outputs) and emulator predictions for NPP ($g\,C\,m^{-2}h^{-1}$) and LAI (m^2m^{-2}) for first, second and third 4-week periods, starting from May.



(a) NPP ($g\,C\,m^{-2}h^{-1}$) emulator predictions (solid line) vs. simulator predictions (dashed line).



(b) LAI (m^2m^{-2}) emulator predictions (solid line) vs. simulator predictions (dashed line).

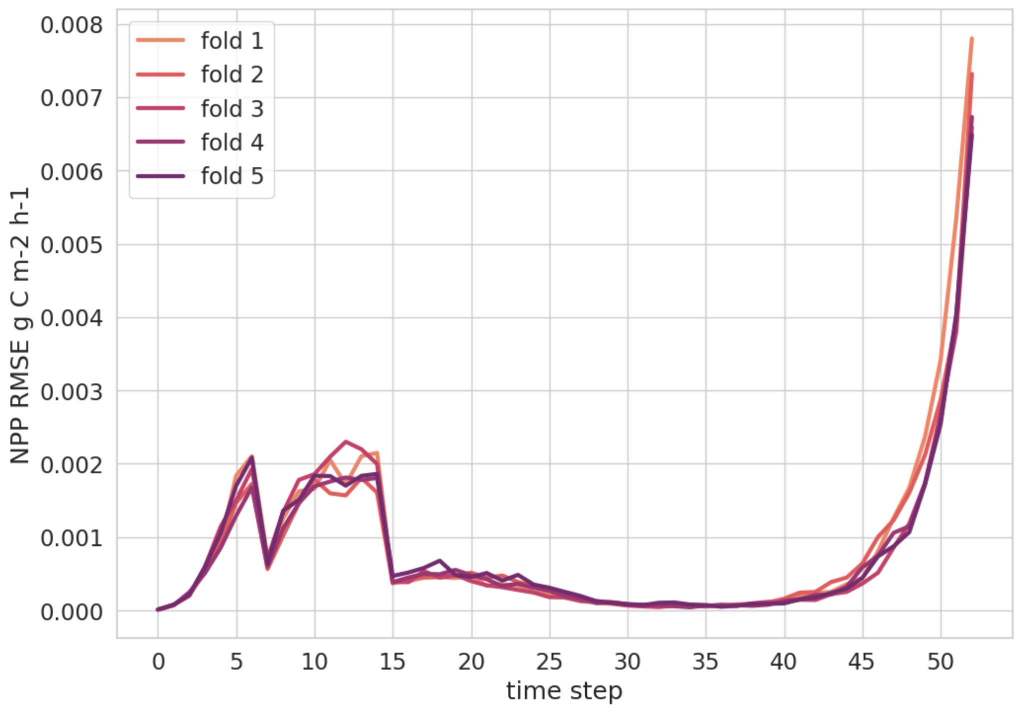
Figure 12: Three random examples of predictions and their corresponding targets across the time steps.

the errors drop and this behavior repeats again. The sudden drops in RMSE are at the harvest time, and the RMSEs increase again when the feature values increase following from the regrowth of the grass. Estimation of smaller feature values gives smaller errors. Again after the second harvest, a smaller increase in RMSE can be observed for NPP and a more notable one for LAI. This is probably because of the regrowth restarts but at a smaller extent.

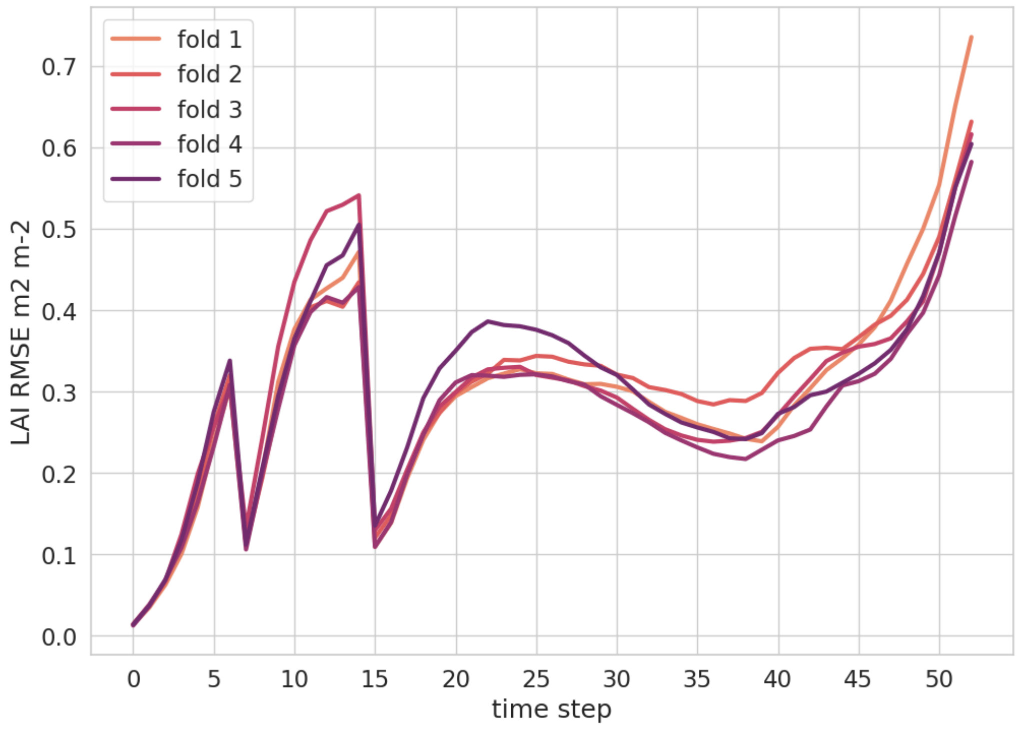
The NPP RMSEs are very low in winter time as NPP outside growing season decreases close to zero because of the reduced activity in photosynthesis, which can also be seen in the examples of Figure 12. LAI, on the other hand, has a more notable increase in RMSE after the last harvest but then it decreases for the winter months. Going towards the end of the simulation periods, the RMSE grows significantly. There are only two months for training the behavior on overwintering, which may not be sufficient to capture the full range of conditions for it. In addition, in northern latitudes these two months have still a very low activity in photosynthesis, due to the effect of colder temperatures. This property could be improved by creating more training data for that time period, either by changing to a bit longer simulation period or just adding more data points in general. However, the predictions of the emulator seem to have a high level of accuracy for the growing season, which is the crucial period regarding to carbon balance dynamics.

The model performance on the geographical area is shown in Figure 14. A consistent performance can be seen across the area, indicating that the model maintains stable predictions across the region.

Finally, Figure 15 offers an overview of the model performance over epochs. The mean RMSE across folds is given per each epoch with an error bar representing one standard deviation of the mean RMSEs of the folds. The error bars provide a range where the true error is expected to be. The sample size is relatively small, with only 5 folds. A more robust estimation could be obtained with more folds. However, the error bars are mostly small, with occasional fluctuations, indicating model stability. For the first epochs, there is more variance in model performances, but with more epochs there can be seen a decrease in variance, with occasional increase in variance. The final epoch and the ones close before it are again very small.

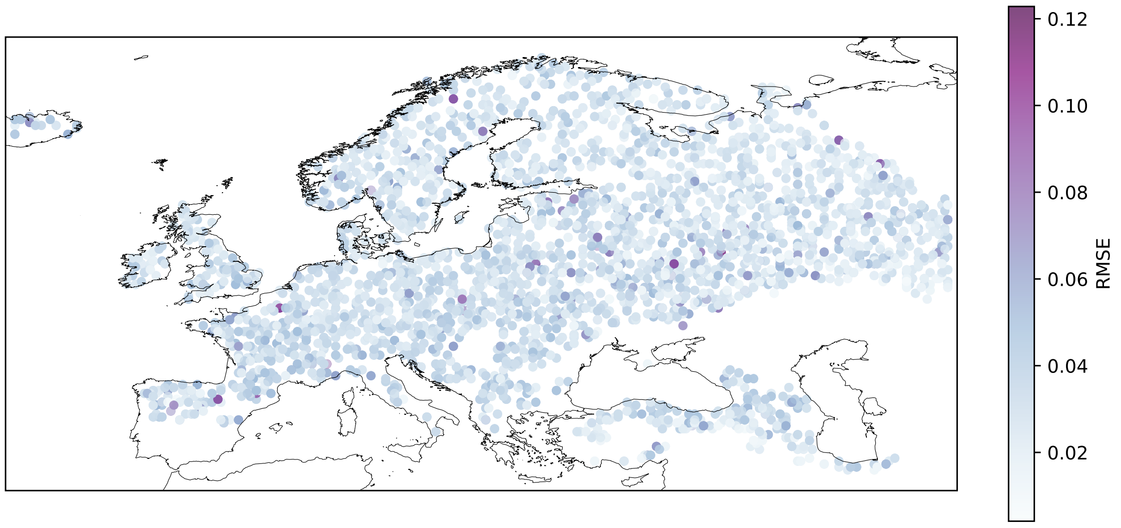


(a) NPP RMSEs ($g\ C\ m^{-2}h^{-1}$) across the time steps.

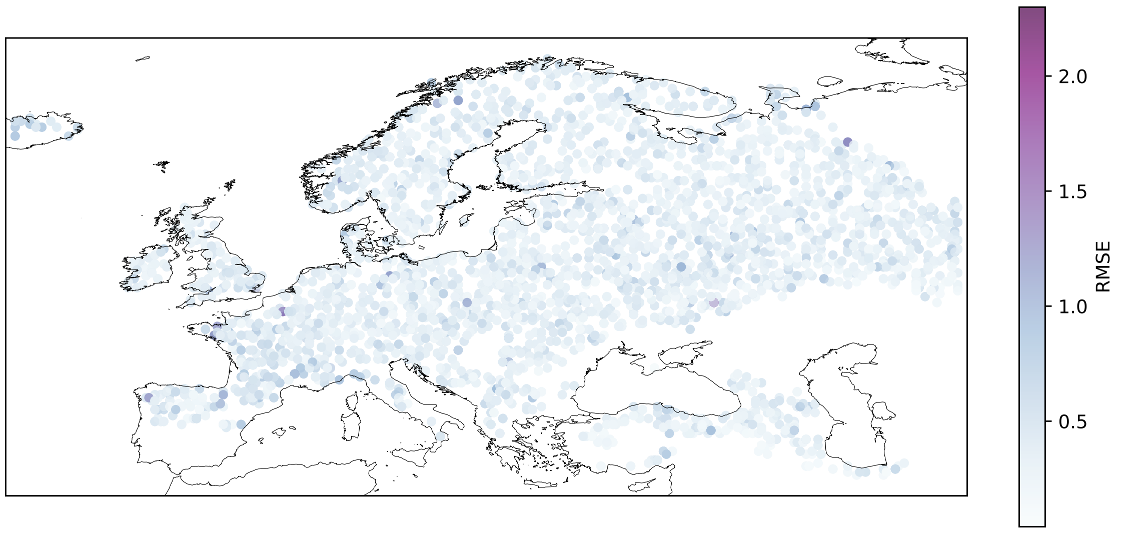


(b) LAI RMSEs (m^2m^{-2}) across the time steps.

Figure 13: Mean RMSEs of each fold with respect to the 53 weekly time steps starting from the beginning of May to the end of April of the following year.



(a) NPP RMSEs ($g\,C\,m^{-2}h^{-1}$).



(b) LAI RMSEs ($m^2 m^{-2}$).

Figure 14: Test RMSEs across folds for NPP and LAI on the geographical area.

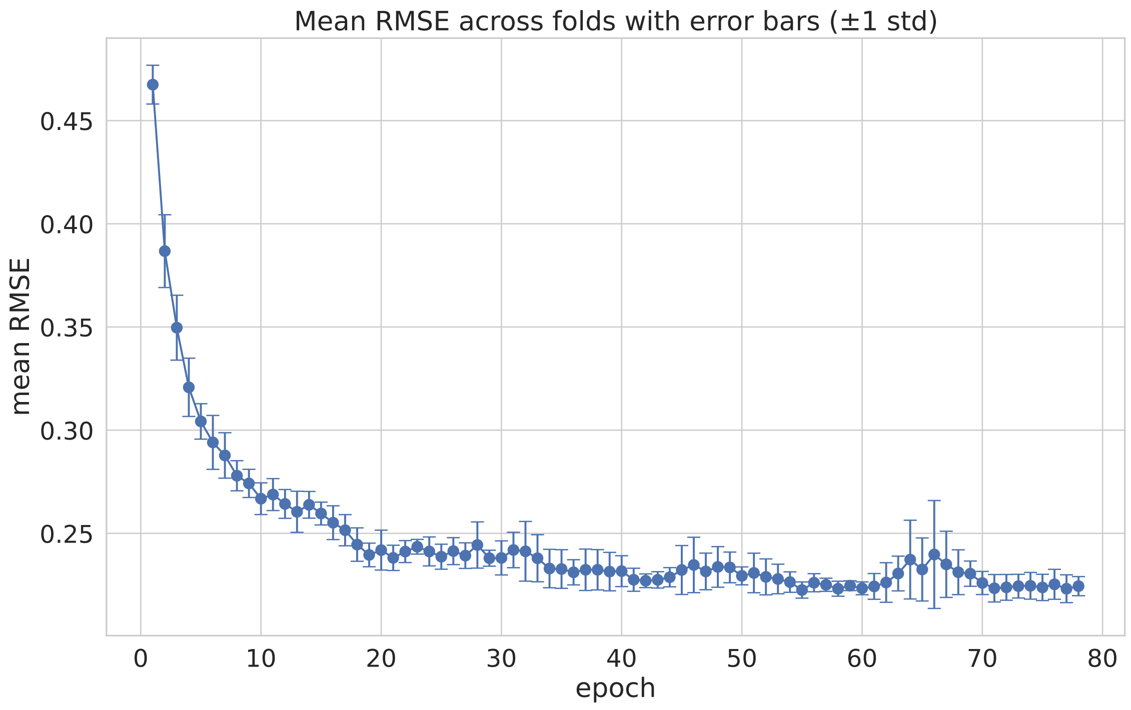


Figure 15: Normalized mean RMSE for both NPP and LAI across the five folds over the course of 78 epochs, with error bars representing one standard deviation (± 1).

5 Discussion

Accurate process-based modeling can be computationally heavy when modeling complex systems, which leads to slow execution time, sensitivity analysis and calibration. This study introduces an approach using LSTM network model to emulate LAI and NPP outputs of a process-based agroecosystem model. The approach of LSTM and FNN is taken due to their abilities to learn complex patterns within data and the LSTM's capability to process time dependencies. The emulator demonstrates to effectively emulate the behavior of the simulator, with a notably faster performance.

Even though the model is trained only for use in Finland, it is also applicable for use in other areas inside the geographical area of the training data. A large area of training data, compared to the area where the model is applied to, enables diverse climatic cases, which could help in handling extreme weather situations better. The model could possibly be applicable to same northern latitudes around the globe and other regions with similar climate. By enlarging the training area, the emulator could be more applicable for also other regions.

Other concern could be the applicability of the model for future cases, with the increasing impact of climate change. This could be tested with meteorological inputs generated by a climate model predicting for future values, as done in [72], where global circulation models are used for prediction of future climate. Also, the current emulator considers same harvest dates for all samples. Thus, for ensuring the emulator to generalize to diverse harvest dates the harvest parameter should vary.

Even though the model achieves a relatively strong performance on test data, there is still a notable gap between the train and test error seen in Figure 10, which could possibly be improved by considering diverse possibilities of further refinement of the model. Further improvement would include a larger training sample, as NNs generally perform better with increasing amount of training data. Furthermore, the effect of diverse training samples on model performance is an area worth studying of. Different sampling methods could be used for the cross-validation folds. The current model's folds have each separate randomly chosen geographical points and years, which may not be the optimal approach. Of course there are many diverse sampling methods and it is challenging to find the optimal one. One alternative approach could be dividing the data in uniform geographical areas and perhaps using all years available. This way the model performance would be tested always with a new geographical area separate from the other areas. As mentioned in the previous chapter, the test error increases when moving towards the end of the simulation period, which may be due to the challenges in modeling the overwintering. The challenges could be improved with more training data to understand the effect of winter conditions on the following months' features. Also it could be worth trying diverse input time step lengths in addition to the weekly values to find the optimal length for best result, but still with a reasonable input length.

Selection of input parameters is challenging because the predicted LAI and NPP are influenced by diverse factors. The selection has to include important factors for both of them, by maybe excluding some possibly important ones for either of the features. It is worth considering additional parameters, which possibly are important for some of the output features and are excluded from the current version. This is a possible direction for future exploration and refinement of the model. Also for a wider picture of the carbon dynamics in the system, prediction of other features regarding to carbon dynamics could be added.

As [45] noted, a reduced output dimension is helpful when working with limited amount of training data. This work uses only about 8000 data points for training per fold which is very few by today's standards. The small output dimension and the relative simplicity of the model is most possibly advantageous for the learning with limited amount of data. However, repeating what is mentioned before, a larger training data could arguably be beneficial. At least when more output parameters are required, a larger training data should be used.

Although the emulator architecture was designed for BASGRA_N, it could be applied to other computational agroecosystem simulators with dynamic and static inputs. The network is not limited for only grassland LAI and NPP prediction, but it is rather designed for learning the relationships between sequential predictions and sequential and static inputs. Thus, the emulator architecture could also be applied to some other large computational models to mitigate the computational challenges in large scale simulations. However modifications should be done depending on the model in target of emulation.

In addition to the method used in this thesis, other alternative approaches such as the Gaussian processes could also be effective for addressing the given task. As seen in Section 4.1, Gaussian processes do not generally learn temporal dependencies and can be computationally heavy with large data sets. However, a study emulating a complex land surface model [1] gives a promising approach for preventing both of these problems. For learning temporal dependencies, a superior performance is observed in recurrent neural networks compared to GP. A study using a GRU approach [42] develops a model with a hierarchical structure employing four GRU layers and a FNN with an attention mechanism, where each of the temporal predicted outputs are given by its own GRU layer. Also, the study does not process the static soil property inputs by a separate network, but processes them through the GRU layers, together with the daily inputs. Compared to the LSTM model of this work, the multiple GRU layers may increase the computational demands of the model, while the method of this thesis uses a single LSTM layer giving predictions for both of the sequential outputs. Still, the hierarchical structure may be beneficial as it incorporates the causal relations between the different variables and processes. If applied to this work, both NPP and LAI would have their own LSTM layers, which both could have initial hidden states given by a "basis" LSTM layer and perhaps the LAI predictions could work as an input for the LSTM layer predicting the NPP. The application of a hierarchical structure is a potential area for future research. A study based on LSTM networks [44] has a similar approach to this thesis for processing both dynamic and static inputs, though for prediction of overall yield and not sequential values. Similarly to the approach of this thesis, the study uses a LSTM layer for processing sequential time dependent inputs and a FNN for processing the static inputs. The study investigates different combination techniques for the static and dynamic data, and ultimately uses an attention technique which learns weights for the values of each time step. The comparison of methods is not directly applicable for sequential prediction. This thesis combines the dynamic time dependent data and the static data by concatenating the outputs of a LSTM layer and a FNN. Alternative techniques in combining dynamic and static inputs in sequential prediction could be explored further.

Also other more recent neural network approaches, such as transformers [67] and temporal convolutional networks [35], are a promising area for further research. Compared to LSTM, transformers process the entire data sequence, instead of one at a time, and uses a self-attention mechanism to learn the temporal dependencies in the data. Temporal convolutional networks, on the other hand, learn temporal dependencies by stacking convolutional layers [38]. They both have shown promising performance in other machine learning tasks with long sequential prediction [75], [7] and could also be applied in emulation of dynamical process-based models.

6 Conclusions

This study develops a framework for emulating dynamic agroecosystem models using recurrent neural networks. The emulator takes in to account the dynamical and static features of the agroecosystem and produces sequential values. The surrogate is specifically build for the grassland model BASGRA_N, to emulate the weekly key features regarding to carbon balance: net primary production and leaf area index. Training data is generated from input-output pairs of simulations of the grassland model. The model is evaluated for a geographical area in Europe with synthetically sampled input parameters on vegetation and soil properties and meteorological inputs from historical data. The emulator is trained to process one year worth of weekly meteorological data and the property parameters to create weekly predictions of NPP and LAI for the corresponding year. The model is evaluated by 5-fold cross-validation, which gives mean RMSEs of 0.322 ($m^2 m^{-2}$) for LAI and 0.0290 ($g\ C\, m^{-2} h^{-1}$) for NPP across all folds, and all folds give R^2 of > 0.93 for LAI and > 0.91 for NPP. The results indicate that the proposed framework can accurately emulate the dynamic variables of the agroecosystem model. The emulator has potential to work as a general foundation for emulating agroecosystem models, mitigating large scale simulations, thus enabling improvement of the simulator and advanced modeling.

References

[1] Evan Baker et al. "Emulation of high-resolution land surface models using sparse Gaussian processes with application to JULES". In: *Geoscientific Model Development* 15.5 (2022), pp. 1913–1929. DOI: 10.5194/gmd-15-1913-2022. URL: https://gmd.copernicus.org/articles/15/1913/2022/.

[2] Christopher Bishop. *Neural Networks for Pattern Recognition.* Advanced Texts in Econometrics. Clarendon Press, 1995. ISBN: 9780198538646. URL: https://books.google.fi/books?id=T0S0BgAAQBAJ.

[3] Christopher Bishop. *Pattern Recognition and Machine Learning.* Springer, Jan. 2006. URL: https://www.microsoft.com/en-us/research/publication/pattern-recognition-machine-learning/.

[4] Leo Breiman. "Random forests". In: *Machine Learning* 45 (2001), pp. 5–32. DOI: 10.1023/A:1010933404324. URL: https://doi.org/10.1023/A:1010933404324.

[5] F. Stuart Chapin and Valerie T. Eviner. "8.06 - Biogeochemistry of Terrestrial Net Primary Production". In: *Treatise on Geochemistry.* Ed. by Heinrich D. Holland and Karl K. Turekian. Oxford: Pergamon, 2007, pp. 1–35. ISBN: 978-0-08-043751-4. DOI: https://doi.org/10.1016/B0-08-043751-6/08130-5. URL: https://www.sciencedirect.com/science/article/pii/B0080437516081305.

[6] Samprit Chatterjee and Ali S. Hadi. *Regression Analysis by Example.* Wiley Series in Probability and Statistics. Wiley, 2006. ISBN: 9780470055458. URL: https://books.google.fi/books?id=uiu5XsAA9kYC.

[7] Yitian Chen et al. "Probabilistic forecasting with temporal convolutional neural network". In: *Neurocomputing* 399 (2020), pp. 491–501. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2020.03.011. URL: https://www.sciencedirect.com/science/article/pii/S0925231220303441.

[8] Kyunghyun Cho et al. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation.* 2014. arXiv: 1406.1078 [cs.CL].

[9] Junyoung Chung et al. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling.* 2014. arXiv: 1412.3555 [cs.NE].

[10] Stefano Conti and Anthony O'Hagan. "Bayesian emulation of complex multi-output and dynamic computer models". In: *Journal of Statistical Planning and Inference* 140.3 (2010), pp. 640–651. ISSN: 0378-3758. DOI: https://doi.org/10.1016/j.jspi.2009.08.006. URL: https://www.sciencedirect.com/science/article/pii/S0378375809002559.

[11] Fleur Couvreux et al. "Process-Based Climate Model Development Harnessing Machine Learning: I. A Calibration Tool for Parameterization Improvement". In: *Journal of Advances in Modeling Earth Systems* 13 (Feb. 2021). DOI: 10.1029/2020MS002217.

[12] Katherine Dagon et al. "A machine learning approach to emulation and biophysical parameter estimation with the Community Land Model, version 5". In: *Advances in Statistical Climatology, Meteorology and Oceanography* 6.2 (2020), pp. 223–244. DOI: 10.5194/ascmo-6-223-2020. URL: https://ascmo.copernicus.org/articles/6/223/2020/.

[13] Mathieu Delandmeter et al. "A comprehensive analysis of CO2 exchanges in agro-ecosystems based on a generic soil-crop model-derived methodology". In: *Agricultural and Forest Meteorology* 340 (2023), p. 109621. ISSN: 0168-1923. DOI: https://doi.org/10.1016/j.agrformet.2023.109621. URL: https://www.sciencedirect.com/science/article/pii/S016819232300312X.

[14] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. "A Comprehensive Survey and Performance Analysis of Activation Functions in Deep Learning". In: *CoRR* abs/2109.14545 (2021). arXiv: 2109.14545. URL: https://arxiv.org/abs/2109.14545.

[15] Steven Elsworth and Stefan Güttel. *Time Series Forecasting Using LSTM Networks: A Symbolic Approach.* 2020. DOI: https://arxiv.org/abs/2003.05672. arXiv: 2003.05672 [cs.LG].

[16] İlker Ercanlı, Alkan Günlü, and Muammer Şenyurt. et al. "Artificial neural network models predicting the leaf area index: a case study in pure even-aged Crimean pine forests from Turkey". In: *Forest Ecosystems* 5 (2018), p. 29. DOI: 10.1186/s40663-018-0149-8.

[17] Hongliang Fang and Shunlin Liang. "Leaf Area Index Models". In: *Encyclopedia of Ecology.* Ed. by Sven Erik Jørgensen and Brian D. Fath. Oxford: Academic Press, 2008, pp. 2139–2148. ISBN: 978-0-08-045405-4. DOI: https://doi.org/10.1016/B978-008045405-4.00190-7. URL: https://www.sciencedirect.com/science/article/pii/B9780080454054001907.

[18] Puyu Feng et al. "Dynamic wheat yield forecasts are improved by a hybrid approach using a biophysical model and machine learning technique". In: *Agricultural and Forest Meteorology* 285-286 (2020), p. 107922. ISSN: 0168-1923. DOI: https://doi.org/10.1016/j.agrformet.2020.107922. URL: https://www.sciencedirect.com/science/article/pii/S0168192320300241.

[19] Istem Fer et al. "Linking big models to big data: efficient ecosystem model calibration through Bayesian model emulation". In: *Biogeosciences* 15.19 (2018), pp. 5801–5830. DOI: 10.5194/bg-15-5801-2018. URL: https://bg.copernicus.org/articles/15/5801/2018/.

[20] Xiang Gao et al. "Emulation of Community Land Model Version 5 (CLM5) to Quantify Sensitivity of Soil Moisture to Uncertain Parameters". In: *Journal of Hydrometeorology* 22.2 (2021), pp. 259–278. DOI: 10.1175/JHM-D-20-0043.1. URL: https://journals.ametsoc.org/view/journals/hydr/22/2/jhm-d-20-0043.1.xml.

[21] Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. "Learning to forget: continual prediction with LSTM". In: *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470).* Vol. 2. 1999, 850–855 vol.2. DOI: 10.1049/cp:19991218.

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[23] Trevor Hastie et al. *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. Springer, 2009.

[24] Hans Hersbach et al. *ERA5 hourly data on single levels from 1940 to present*. Accessed on 1-12-2023. 2023. DOI: `10.24381/cds.adbb2d47`.

[25] Sepp Hochreiter. "The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6 (Apr. 1998), pp. 107–116. DOI: `10.1142/S0218488598000094`.

[26] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: `10.1162/neco.1997.9.8.1735`.

[27] Mats Höglind et al. "BASGRA_N: A model for grassland productivity, quality and greenhouse gas balance". In: *Ecological Modelling* 417 (2020), p. 108925. ISSN: 0304-3800. DOI: `https://doi.org/10.1016/j.ecolmodel.2019.108925`. URL: `https://www.sciencedirect.com/science/article/pii/S0304380019304338`.

[28] Pradyot Ranjan Jena, Shunsuke Managi, and Babita Majhi. "Forecasting the CO2 Emissions at the Global Level: A Multilayer Artificial Neural Network Modelling". In: *Energies* 14 (Oct. 2021), p. 6336. ISSN: 1996-1073. DOI: `10.3390/en14196336`. URL: `https://www.mdpi.com/1996-1073/14/19/6336`.

[29] Wenhao Jiang et al. *Recurrent Fusion Network for Image Captioning*. 2018. DOI: `https://doi.org/10.48550/arXiv.1807.09986`. arXiv: `1807.09986 [cs.CV]`.

[30] Thorsten Joachims. *Making large-scale SVM learning practical*. eng. Technical Report 1998,28. Dortmund, 1998. URL: `http://hdl.handle.net/10419/77178`.

[31] David B. Johnston et al. "Comparison of machine learning methods emulating process driven crop models". In: *Environmental Modelling and Software* 162 (2023), p. 105634. ISSN: 1364-8152. DOI: `https://doi.org/10.1016/j.envsoft.2023.105634`. URL: `https://www.sciencedirect.com/science/article/pii/S1364815223000208`.

[32] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. Presented as a conference paper at the 3rd International Conference for Learning Representations (ICLR), San Diego, 2015. 2014. arXiv: `1412.6980 [cs.LG]`.

[33] Ron Kohavi. "A study of cross-validation and bootstrap for accuracy estimation and model selection". In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*. IJCAI'95. Montreal, Quebec, Canada: Morgan Kaufmann Publishers Inc., 1995, pp. 1137–1143. ISBN: 1558603638.

[34] Steve Lawrence, Clyde Lee Giles, and S. Fong. "Natural language grammatical inference with recurrent neural networks". In: *IEEE Transactions on Knowledge and Data Engineering* 12.1 (2000), pp. 126–140. DOI: 10.1109/69.842255.

[35] Colin Lea et al. *Temporal Convolutional Networks for Action Segmentation and Detection.* 2016. arXiv: 1611.05267 [cs.CV].

[36] David Lebauer et al. "Facilitating feedbacks between field measurements and ecosystem models". In: *Ecological Monographs* 83 (May 2013), pp. 133–154. DOI: 10.1890/12-0137.1.

[37] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521 (2015). Received 25 February 2015, Accepted 01 May 2015, Published 27 May 2015, Issue Date 28 May 2015, pp. 436–444. DOI: 10.1038/nature14539.

[38] Yann LeCun et al. "Object Recognition with Gradient-Based Learning". In: *Shape, Contour and Grouping in Computer Vision.* Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 319–345. ISBN: 978-3-540-46805-9. DOI: 10.1007/3-540-46805-6_19. URL: https://doi.org/10.1007/3-540-46805-6_19.

[39] S. Lehuger et al. "Predicting the net carbon exchanges of crop rotations in Europe with an agro-ecosystem model". In: *Agriculture, Ecosystems & Environment* 139.3 (2010). The carbon balance of European croplands, pp. 384–395. ISSN: 0167-8809. DOI: https://doi.org/10.1016/j.agee.2010.06.011. URL: https://www.sciencedirect.com/science/article/pii/S0167880910001659.

[40] Peng Li et al. "Quantification of the response of global terrestrial net primary production to multifactor global change". In: *Ecological Indicators* 76 (2017), pp. 245–255. ISSN: 1470-160X. DOI: https://doi.org/10.1016/j.ecolind.2017.01.021. URL: https://www.sciencedirect.com/science/article/pii/S1470160X17300274.

[41] Jianzhao Liu et al. "Comparative Analysis of Two Machine Learning Algorithms in Predicting Site-Level Net Ecosystem Exchange in Major Biomes". In: *Remote Sensing* 13.12 (2021). ISSN: 2072-4292. DOI: 10.3390/rs13122242. URL: https://www.mdpi.com/2072-4292/13/12/2242.

[42] Licheng Liu, Wang Zhou, Kaiyu Guan, et al. "Knowledge-guided machine learning can improve carbon cycle quantification in agroecosystems". In: *Nature Communications* 15 (2024), p. 357. DOI: 10.1038/s41467-023-43860-5. URL: https://www.nature.com/articles/s41467-023-43860-5.

[43] Licheng Liu et al. "KGML-ag: a modeling framework of knowledge-guided machine learning to simulate agroecosystems: a case study of estimating N2O emission using data from mesocosm experiments". In: *Geoscientific Model Development* 15 (2022), pp. 2839–2858. DOI: 10.5194/gmd-15-2839-2022. URL: https://doi.org/10.5194/gmd-15-2839-2022.

[44] Qinqing Liu et al. “Machine Learning Crop Yield Models Based on Meteorological Features and Comparison with a Process-Based Model”. In: *Artificial Intelligence for the Earth Systems* 1.4 (2022), e220002. DOI: https://doi.org/10.1175/AIES-D-22-0002.1. URL: https://journals.ametsoc.org/view/journals/aies/1/4/AIES-D-22-0002.1.xml.

[45] D. Lu and D. Ricciuto. “Efficient surrogate modeling methods for large-scale Earth system models based on machine-learning techniques”. In: *Geoscientific Model Development* 12.5 (2019), pp. 1791–1807. DOI: 10.5194/gmd-12-1791-2019. URL: https://gmd.copernicus.org/articles/12/1791/2019/.

[46] Hadi Mohammed, Hoese Michel Tornyeviadzi, and Razak Seidu. “Emulating process-based water quality modelling in water source reservoirs using machine learning”. In: *Journal of Hydrology* 609 (2022), p. 127675. ISSN: 0022-1694. DOI: https://doi.org/10.1016/j.jhydrol.2022.127675. URL: https://www.sciencedirect.com/science/article/pii/S0022169422002505.

[47] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. “Methods for interpreting and understanding deep neural networks”. In: *Digital Signal Processing* 73 (2018), pp. 1–15. ISSN: 1051-2004. DOI: https://doi.org/10.1016/j.dsp.2017.10.011. URL: https://www.sciencedirect.com/science/article/pii/S1051200417302385.

[48] Samuel Thomas Wilkins Myren and Earl Christopher Lawrence. “A comparison of Gaussian processes and neural networks for computer model emulation and calibration”. In: *Statistical Analysis and Data Mining* 14.6 (Mar. 2021). ISSN: 1932-1864. DOI: 10.1002/sam.11507. URL: https://www.osti.gov/biblio/1825406.

[49] Trung H. Nguyen, Duy Nong, and Keith Paustian. “Surrogate-based multi-objective optimization of management options for agricultural landscapes using artificial neural networks”. In: *Ecological Modelling* 400 (2019), pp. 1–13. ISSN: 0304-3800. DOI: https://doi.org/10.1016/j.ecolmodel.2019.02.018. URL: https://www.sciencedirect.com/science/article/pii/S0304380019300870.

[50] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “On the difficulty of training recurrent neural networks”. In: *Proceedings of the 30th International Conference on Machine Learning.* Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, June 2013, pp. 1310–1318. URL: https://proceedings.mlr.press/v28/pascanu13.html.

[51] George L. W. Perry, Rupert Seidl, Ana Maria Bellvé, et al. “An Outlook for Deep Learning in Ecosystem Science”. In: *Ecosystems* 25 (2022), pp. 1700–1718. DOI: 10.1007/s10021-022-00789-y. URL: https://doi.org/10.1007/s10021-022-00789-y.

[52] Ana Prieto-Blanco et al. “Satellite-driven modelling of Net Primary Productivity (NPP): Theoretical analysis”. In: *Remote Sensing of Environment* 113.1 (2009), pp. 137–147. ISSN: 0034-4257. DOI: https://doi.org/10.1016/j.rse.2008.09.002. URL: https://www.sciencedirect.com/science/article/pii/S0034425708002708.

[53] Siyu Qi et al. “Multi-Location Emulation of a Process-Based Salinity Model Using Machine Learning”. In: *Water* 14.13 (2022). ISSN: 2073-4441. DOI: 10.3390/w14132030. URL: https://www.mdpi.com/2073-4441/14/13/2030.

[54] Payam Refaeilzadeh, Lei Tang, and Huan Liu. “Cross-Validation”. In: *Encyclopedia of Database Systems*. Ed. by LING LIU and M. TAMER ÖZSU. Boston, MA: Springer US, 2009, pp. 532–538. ISBN: 978-0-387-39940-9. DOI: 10.1007/978-0-387-39940-9_565. URL: https://doi.org/10.1007/978-0-387-39940-9_565.

[55] Markus Reichstein et al. “Deep learning and process understanding for data-driven Earth system science”. In: *Nature* 566.7743 (2019), pp. 195–204. DOI: 10.1038/s41586-019-0912-1. URL: https://doi.org/10.1038/s41586-019-0912-1.

[56] Stephen Reynolds and John Frame. *Grasslands: developments, opportunities, perspectives.* Science Publishers, 2005.

[57] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning internal representations by error propagation”. In: 1986. DOI: 10.21236/ada164453. URL: https://api.semanticscholar.org/CorpusID:62245742.

[58] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323 (1986), pp. 533–536. DOI: https://doi.org/10.1038/323533a0.

[59] Babak Safa et al. “Net Ecosystem Exchange (NEE) simulation in maize using artificial neural networks”. In: *IFAC Journal of Systems and Control* 7 (2019), p. 100036. ISSN: 2468-6018. DOI: https://doi.org/10.1016/j.ifacsc.2019.100036. URL: https://www.sciencedirect.com/science/article/pii/S2468601817302584.

[60] Iqbal Sarker. “Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions”. In: *SN Computer Science* 2 (Aug. 2021). DOI: 10.1007/s42979-021-00815-1.

[61] Douglas A. Shoemaker and Wendell P. Cropper. “Application of remote sensing, an artificial neural network leaf area model, and a process-based simulation model to estimate carbon storage in Florida slash pine plantations”. In: *Journal of Forestry Research* 21 (2010), pp. 171–176. DOI: 10.1007/s11676-010-0027-x. URL: https://doi.org/10.1007/s11676-010-0027-x.

[62] John P. Snyder. *Map projections: A working manual.* English. Tech. rep. Report. Washington, D.C., 1987. DOI: 10.3133/pp1395. URL: https://doi.org/10.3133/pp1395.

[63] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. *Sequence to Sequence Learning with Neural Networks*. 2014. DOI: https://doi.org/10.48550/arXiv.1409.3215. arXiv: 1409.3215 [cs.CL].

[64] G. Tramontana et al. "Predicting carbon dioxide and energy fluxes across global FLUXNET sites with regression algorithms". In: *Biogeosciences* 13.14 (2016), pp. 4291–4313. DOI: 10.5194/bg-13-4291-2016. URL: https://bg.copernicus.org/articles/13/4291/2016/.

[65] Gianluca Tramontana et al. "Uncertainty analysis of gross primary production upscaling using Random Forests, remote sensing and eddy covariance data". In: *Remote Sensing of Environment* 168 (2015), pp. 360–373. ISSN: 0034-4257. DOI: https://doi.org/10.1016/j.rse.2015.07.015. URL: https://www.sciencedirect.com/science/article/pii/S0034425715300699.

[66] Marcel Van Oijen, Gianni Bellocchi, and Mats Höglind. "Effects of Climate Change on Grassland Biodiversity and Productivity: The Need for a Diversity of Models". In: *Agronomy* 8.2 (2018). ISSN: 2073-4395. DOI: 10.3390/agronomy8020014. URL: https://www.mdpi.com/2073-4395/8/2/14.

[67] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL].

[68] Nathalie Villa-Vialaneix et al. "A comparison of eight meta-modeling techniques for the simulation of N2O fluxes and N leaching from corn crops". In: *Environmental Modelling and Software* 34 (2012), pp. 51–66. DOI: 10.1016/j.envsoft.2011.05.003. URL: https://hal.archives-ouvertes.fr/hal-00654753.

[69] Yongqiang Wang et al. "Combining Data Assimilation with Machine Learning to Predict the Regional Daily Leaf Area Index of Summer Maize (Zea mays L.)" In: *Agronomy* 13.11 (2023). ISSN: 2073-4395. DOI: 10.3390/agronomy13112688. URL: https://www.mdpi.com/2073-4395/13/11/2688.

[70] Ronald Williams and Jing Peng. "An Efficient Gradient-Based Algorithm for On-Line Training of Recurrent Network Trajectories". In: *Neural Computation* 2 (Sept. 1998). DOI: 10.1162/neco.1990.2.4.490.

[71] D.Randall Wilson and Tony R. Martinez. "The general inefficiency of batch training for gradient descent learning". In: *Neural Networks* 16.10 (2003), pp. 1429–1451. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(03)00138-2. URL: https://www.sciencedirect.com/science/article/pii/S0893608003001382.

[72] Liujun Xiao et al. "Coupling agricultural system models with machine learning to facilitate regional predictions of management practices and crop production". In: *Environmental Research Letters* 17.11 (Nov. 2022), p. 114027. DOI: 10.1088/1748-9326/ac9c71. URL: https://dx.doi.org/10.1088/1748-9326/ac9c71.

[73] Qi Yang et al. “A flexible and efficient knowledge-guided machine learning data assimilation (KGML-DA) framework for agroecosystem prediction in the US Midwest”. In: *Remote Sensing of Environment* 299 (2023), p. 113880. ISSN: 0034-4257. DOI: https://doi.org/10.1016/j.rse.2023.113880. URL: https://www.sciencedirect.com/science/article/pii/S0034425723004315.

[74] Carlos Zednik. “Solving the Black Box Problem: A Normative Framework for Explainable Artificial Intelligence”. In: *Philosophy & Technology* 34 (2021), pp. 265–288. DOI: 10.1007/s13347-019-00382-7. URL: https://doi.org/10.1007/s13347-019-00382-7.

[75] Haoyi Zhou et al. “Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.12 (May 2021), pp. 11106–11115. DOI: 10.1609/aaai.v35i12.17325. URL: https://ojs.aaai.org/index.php/AAAI/article/view/17325.