TÉCNICO LISBOA

An artificial learning biophysical model for the evolution of land use and greenhouse gases emissions

Bruno Pereira Costa

Thesis to obtain the Master of Science Degree in

Engineering Physics

Supervisors: Prof. Ricardo Filipe De Melo Teixeira
Prof. João Carlos Carvalho de Sá Seixas

Examination Committee

Chairperson: Prof. Ilídio Pereira Lopes
Supervisor: Prof. Ricardo Filipe De Melo Teixeira
Member of the Committee: Prof. Susana Margarida da Silva Vieira
Prof. Luís Humberto Viseu Melo

January 2021

Dedicated to my mom

Acknowledgments

The realisation of this master thesis was a huge challenge, and its development was an enriching learning experience. Its accomplishment would not be possible without further human interaction. Firstly, I would like to manifest my personal thanks to Professor Ricardo Teixeira for the precise guidance, sharing of knowledge, availability, and confidence transmitted throughout this work. Secondly, I would like to thank all MEFT (and non-MEFT) teachers that contributed towards my academic progression. Thirdly, I cannot stress enough how vital my course colleagues and friends were for their companionship, help, and good times. Forthly, I am deeply thankful to all my family for their support at this demanding period. Lastly, to all the people with whom I have crossed paths with, a sincere thank you.

The work was inspired by the project “Intergenerational Impact of Biophysical Resource use in Portugal Within Planetary Environmental Boundaries” from the Gulbenkian Foundation, which kindly provided some of the data sources used in this work. Tiago Morais, Laura Felício and João Santos were part of the team that shared their precious resources.

“Our main business is not to see what lies dimly at a distance, but to do what lies clearly at hand.”

– Thomas Carlyle

Resumo

Modelos quantitativos de emissões de gases com efeito de estufa (GEE) ajudam a avaliar políticas sustentáveis para limitar o aquecimento global, respeitando o Acordo de Paris. Os sectores agrícola e energético são particularmente importantes, representando mais de dois terços das emissões globais. Esta tese utiliza redes neuronais recorrentes, convolucionais e híbridas para desenvolver um modelo biofísico para o uso do solo e emissões de GEE associadas em Portugal. O ajuste dos hiperparâmetros utiliza a optimização Bayesiana e é proposto um novo algoritmo de estimativa de erros baseado numa abordagem de janela de crescimento fora da amostra. O uso do solo foi modelado em função da exergia final no sector agrícola e do Produto Interno Bruto (PIB), e as emissões em função do uso do solo, da exergia final total e do PIB. Os modelos foram treinados para o período 1961-2016 e aplicados em 2017-2030 sob dois cenários económicos plausíveis com e sem influência da COVID-19. Mostra-se que o uso do solo está correlacionado com o PIB, e as emissões de GEE estão correlacionadas com a exergia final total. Crescimento económico leva à redução da área cultivada, ao aumento do consumo de energia, e a variações nas emissões sectoriais de GEE. A COVID-19 poderá abrandar a redução de área agrícola e mitigar o aumento de emissões de CO_2e até 2030. Apesar da complexidade do modelo, os erros estimados excederam a gama de variação prevista das variáveis. A incerteza é crítica para a avaliação de cenários, lançando dúvidas sobre modelos mais simples.

Palavras-chave: Agricultura, Otimização Bayesiana, Exergia, Portugal, Redes Neuronais Recorrentes, Redes Neuronais Convolucionais

Abstract

Quantitative models of greenhouse gases (GHG) emissions can help policy-makers gauge sustainable pathways towards limiting global warming to stay within 1.5°C of the Paris Agreement. The agricultural and energy sectors are particularly important, as they jointly represent more than two thirds of global GHG emissions. This thesis uses an ensemble of recurrent and convolutional neural networks and hybrid architectures to develop a biophysical model for the evolution of land use and associated GEE in Portugal. The models' hyperparameters tuning uses a state-of-the-art framework of Bayesian optimisation and a new error estimation algorithm based on an out-of-sample growing window approach is proposed. Land use shares were modelled as a function of final exergy in the agricultural sector and Gross Domestic Product (GDP), and emissions as a function of distribution of land use, total final exergy, and GDP. Models were trained for the period of 1961-2016 and applied from 2017 to 2030 under two plausible economic scenarios with and without COVID-19 influence. Results show that land use is correlated with GDP, and GHG emissions from agriculture and energy are correlated with total final exergy. Economic growth leads to a reduction in cropland area, increased intensity of energy consumption, and variations in sectoral GHG emissions. The novel coronavirus pandemic might decrease the cropland area reduction and mitigate the increase in emissions of CO_2e up to 2030. Despite the complexity of the model, estimation errors exceeded the variation range of the forecasted variables. Uncertainty is critical for scenario assessment, casting doubt over simpler models.

Keywords: Agriculture, Bayesian Optimisation, Exergy, Portugal, Recurrent Neural Networks, Convolutional Neural Networks

Contents

List of Tables

List of Figures

Nomenclature

Deep Learning notation

n	Number of features
m	Number of instances in the dataset considered
$\mathbf{x}$	Instance's feature vector, ranging from x_0 to x_n
x_i	i^{th} feature value
$\mathbf{x}^i$	Vector of all the features values of the i^{th} instance in the dataset
$\mathbf{X}$	Matrix consisting of all the feature values of all instances in the dataset
y^i	Label of the i^{th} instance in the dataset
$\boldsymbol{\theta}$	Model's parameter vector
h	System's prediction (hypothesis) function

Deep Learning for Time Series notation

r	Sampling rate
l	Lagging parameter
$\mathbf{x}_t$	Neuron input vector
y_t	Neuron output scalar
$\mathbf{y}_t$	Neuron output vector
$\mathbf{X}_t$	Matrix with inputs for all instances
$\mathbf{Y}_t$	Matrix with layer's outputs for each instance
$\mathbf{w}$	Weight vector for inputs
$\mathbf{W}$	Weight matrix with weight vectors $\mathbf{w}$
$\phi(\cdot)$	Activation function
$\mathbf{b}$	Bias vector

Glossary

Activation function Responsible for scaling the inputs and producing an output which is passed forward, according to a specified function. 11, 16–18, 20, 21, 27, 31, 34, 46–49, 73

AI Artificial Intelligence (abbreviation). 4, 5

ALES Automated Land Evaluation System (abbreviation). 3

Anergy Energy's not useful counterpart which cannot be transformed into work. 7

ANN Artificial Neural Network (abbreviation). 3, 4, 10, 12

Anthropocene The time from the 18th century until now. 3

APA Agência Portuguesa Ambiente (abbreviation). 58

API Application Programming Interface (abbreviation). 42

ARIMA Autoregressive Integrated Moving Average (abbreviation). 45

Backpropagation Computation of the gradient of the loss function with respect to the weights of a network. 11, 13, 18, 26, 28, 29, 49

Bias neuron Additional input into the layer that will always have a constant value. 11, 13

BPTT Backpropagation Through Time (abbreviation). 28, 29

CAMCM Cellular Automata-Markov Chain Model (abbreviation). 73

CAP Common Agricultural Policy (abbreviation). 5

CCI Climate Change Initiative (abbreviation). 38

CLC CORINE Land Cover (abbreviation). 39

CNN Convolutional Neural Network (abbreviation). 35

COS Carta De Uso E Ocupação Do Solo (abbreviation). 39

Cost function Metric for measuring the uncertainty of a prediction based on how much the prediction varies from the true value. 11, 13–15, 22, 29, 46, 47

Covariance Measure of the strength of the correlation between two or more sets of random variates. 3, 24

Covariate Possible predictive or explanatory variable of the dependent variable. 3, 4, 65, 72

CPU Central Processing Unit (abbreviation). 42

CSV Comma-Separated Values (abbreviation). 38, 43

Dense layer The neurons in this layer are fully connected to the neurons in the previous layer. 11

DL Deep Learning (abbreviation). 10, 12, 18

DNN Deep Neural Network (abbreviation). 12, 17, 18

DSSAT Decision Support System For Agrotechnology Transfer (abbreviation). 3

ELU Exponential Linear Unit (abbreviation). 20, 21

Epoch A epoch elapses when a full dataset is passed forwards and backwards through a neural network. 11, 22, 44, 46, 50, 51, 56

ESA European Space Agency (abbreviation). 38

EU European Union (abbreviation). 5, 58

Evapotranspiration Loss of water from the soil both by evaporation from the soil surface and by transpiration from the leaves of the plants growing on it. 4

Exergy Maximum amount of work theoretically available by bringing a resource into equilibrium with its surrounding through a reversible process. 3, 4, 6–8, 37–40, 57, 58, 68–70, 81, 96

Extrapolation Predict by projecting known data. 5

FAOSTAT Food And Agriculture Organization Corporate Statistical Database (abbreviation). 38, 39

Filter Convolutional kernel. 35, 48, 56, 61, 73

GDP Gross Domestic Product (abbreviation). 4, 37, 38, 40, 57, 58, 66–70, 72, 73, 96

GHG Greenhouse Gases (abbreviation). 2, 4, 5, 37, 38, 40, 56, 65, 68–73, 81

GIF Graphics Interchange Format (abbreviation). 24

GP Gaussian Process (abbreviation). 23

GPU Graphics Processing Unit (abbreviation). 42, 73

Gradient descent Iterative optimisation algorithm used in machine learning to minimise a loss function. 11, 13–15, 17, 19, 20, 22, 44

GRU Gate Recurrent Unit (abbreviation). 30, 32–34, 36, 48, 49, 59, 65

GVA Gross Value Added (abbreviation). 2

HANPP Human Appropriation Of Net Primary Productivity (abbreviation). 72

HAP Historical Aerial Photographs (abbreviation). 71

Hidden layer Intermediate layer of a neural network, excluding input and output layers. 12, 13, 21, 22, 46

Hidden neuron Neuron from hidden layers. 13

Holocene The period beginning around 11.000 years ago and continuing to the present. 3

Hyperparameter Values manually set (not estimated from data) that are used to help estimate model's parameters during a learning process. 19, 20, 22, 23, 25, 42, 49–52, 56, 57, 63, 65, 73

IEA International Energy Agency (abbreviation). 39

IFN Inventário Florestal Nacional (abbreviation). 39

INE Instituto Nacional De Estatística (abbreviation). 38, 39

Input layer Uppermost layer of a neural network that brings the data into the model for processing by subsequent layers. 12, 22, 48

Input neuron Neuron from input layer. 11, 13, 46

IPCC Intergovernmental Panel On Climate Change (abbreviation). 40

JSON JavaScript Object Notation (abbreviation). 51

Kernel Convolution window. 34–36, 48, 56, 61, 73

Layer Composition of artificial neurons. 11, 13, 17, 18, 21, 26–29, 31–36, 44, 46, 48, 49, 56, 73

Learning rate Parameter that controls the magnitude of change for a model in response to the measured error. 11, 12, 15, 16, 46, 49–51

LSTM Long Short-Term Memory (abbreviation). 30–34, 36, 48, 49, 59, 65

LULC Land Use And Land Cover (abbreviation). 72, 81

MAE Mean Absolut Error (abbreviation). 13, 49, 73

MCMC Markov Chain Monte Carlo (abbreviation). 73

ML Machine Learning (abbreviation). 4, 5, 9, 10, 23, 42, 43, 71

MLP Multi Layer Perceptron (abbreviation). 12, 25

MSE Mean Squared Error (abbreviation). 45, 46, 49, 51, 55, 59, 60

NAV Nesterov Accelerated Gradient (abbreviation). 49

OOS Out-Of-Sample (abbreviation). 53, 56

Optimiser Algorithms used to change the parameters of a neural network such as learning rate and weights in order to reduce the losses. 13, 15, 46, 47, 49, 63

Orthorectification Process of applying corrections for optical distortions from an image sensor system, and apparent changes in the position of ground objects caused by the perspective of the sensor view angle and ground terrain. 71

Output layer Lowermost layer of a neural network that delivers the output. 12, 46, 48, 73

Output neuron Neuron from output layer. 13, 46

PCA Principal Component Analysis (abbreviation). 72

Perceptron Single layer of threshold logic units. 10, 11, 26

Phytomass Plant biomass. 6

RELU Rectified Linear Unit (abbreviation). 18–21

RMSE Root Mean Squared Error (abbreviation). 13, 14, 50–52, 55, 59, 73

RNC Roteiro Para A Neutralidade Carbónica (abbreviation). 73

RNN Recurrent Neural Network (abbreviation). 4, 10, 25, 26, 28–32, 34, 43, 44, 46, 71, 72

SELU Scaled Linear Unit (abbreviation). 21

SGD Stochastic Gradient Descent (abbreviation). 15, 49, 63

Stride Displacement between receptive fields. 35, 48, 49

Supervised deep learning The machine uses labelled training data. 10

TIFF Tag Image File Format (abbreviation). 38

TLU Threshold Logic Unit (abbreviation). 10–12

Unsupervised deep learning The machine uses unlabelled training data. 10

Weight Tunable parameter of a machine learning model that controls the signal between two neurons. 11–13, 17–19, 21, 22, 27–29, 32, 35, 46

1

Introduction

The success and ultimately the demise of civilisations have not always been due to external factors such as cosmic events or impact events, but also due to endogenous factors within the civilisations themselves. The Mayans, the Indus, the Aztecs and the people of Eastern Island are examples of extinct ancient civilisations that vanished before they reached our level of sophistication. We are waking up to our technological capabilities - transformative and destructive; human civilisation faces yet again existential uncertainties induced by the violation of intrinsic, physical planetary boundaries. We have embarked into possibly our greatest evolutionary challenge and we have just started shedding light on it.

1.1 The foundations of a Civilisation

The pillars that support the development and maintenance of a civilisation are access to a stable food supply, social structure, record keeping, technology and arts [1]. Projected impacts of climate instabilities and resources misusage particularly affect one of those pillars: **agriculture**.

Over the past century, there has been spatial segregation between food producers and consumers, leading to changes in terms of energy consumption patterns and land use [2]. Globally, there is a progression towards a set up of densely populated cities whose sustenance comes from intensively cultivated lands/intensively raised livestock decoupled from the cities' site. Without agriculture, it is unmanageable to have modern institutions – it is unquestionably the foundation of our complex civilisation, which is pronouncedly city-based [3, 4]. Altogether, the fate of the civilisation follows the fate of agriculture - our survival is inherently dependent on agriculture's thriving [1, 5, 6].

1.2 Agriculture Exposure to Environmental Pressures

As of 2016, total agriculture land is approximately 40% of the entire planet land surface [7] which unequivocally exhibits the exposure to nature's destructive events and sensitivity to extensive climatic variations [8]. The intensity of storms is expected to rise [9], major cities might face submersion [10],

heatwaves have been more common and lasting longer than ever before [11, 12], heavy rainfalls have been inducing severe floods in several locations all around the world [13], forest fires and droughts' frequencies have increased [13], freshwater reservoirs have diminished [14], small mountain glaciers are disappearing [15], a rising number of species face extinction and many have become extinct [16], extensive damage to Coral Reefs has already occurred [17], and the risk of dangerous feedbacks and abrupt large-scale shifts in the climate system increased substantially [18]. Relative to the pre-industrial epoch, increments in global average temperature ranging from $0°$C to $5°$C encompass repercussions such as rising crop yields in high latitude regions, falling crop yields in developing regions and in the extreme scenario falling yields in developed regions [19].

Resource shortage of phosphorus for fertilisation is projected to increasingly lead to food insecurity while at the same time causing environmental degradation in places where phosphate fertilisers have been overapplied [20]. Likewise, for nitrogen fertiliser produced with the Haber–Bosch process[1], human impact on the global nitrogen cycle has already exceeded the limit of the planet's capacity by a factor of four [21], not to mention the fact that the Haber-Bosch process consumes large amounts of fossil fuels, and is associated with increased Greenhouse Gases (GHG) emissions and contamination of water bodies [22–25]. The effects of adding massive amounts of nitrogen to the nitrogen cycle are comparable to biodiversity loss and exceeds the scale of climate change and ocean acidification, which are also affected by nitrogen [21].

The oversupply of phosphorus and nitrogen also leads to another problem which is eutrophication. This has become the primary water quality issue for most of the freshwater and coastal marine ecosystems in the world [26]. Eutrophication is defined as being an excessive aquatic plant and algae growth resulting from nutrient enrichment by human activity. This process may result in taste, odour, and drinking water treatment problems. Ultimately, the dense presence of aquatic plant and algae limits light penetration, reducing growth and causing die-offs of plants in littoral zones. When these dense algal blooms die, microbial decomposition severely depletes dissolved oxygen, creating hypoxic zones lacking sufficient oxygen to support most organisms [27].

Additionally, agriculture intensification is anticipated in order to increase food production by 60-110%, fulfilling the expected needs for the world in 2050 [28]. Moreover, this complex dynamic is constrained by availability of nutrients, land area, access to clean water, as well as renewable and cheap energy. Providing adequate nutrients for increased yields would likely accelerate environmental degradation due to the oversupply of phosphorus and nitrogen as described previously. Agriculture extensification is an alternative strategy that focuses on reducing fertilisers inputs and livestock densities, whose benefits include higher local biodiversity and less environmental pollution, tackling some previously mentioned issues. Nonetheless, it too poses risks such as reduction of yields and therewith a decrease of both Gross Value Added (GVA) and farm income, smaller contribution to global food production and, potentially, an increase of global demand for land, causing deforestation [29].

[1] By transforming unreactive atmospheric dinitrogen (N_2) into a reactive form.

1.3 Biophysical Modelling

> *"A biophysical model is a simulation of a biological system using mathematical formalisations of the physical properties of that system. Such models can be used to predict the influence of biological and physical factors on complex systems."*
>
> – Nature Magazine

In order to mitigate dangerous ambiguity of scenarios for the future, one should focus on the deceleration or even reversion of the current transition from the stable and naturally driven conditions of the Holocene to the human driven unstable conditions of the Anthropocene [30]. For this purpose, integrated biophysical and economic models capable of explicitly depicting the results of possible sustainable pathways are becoming a common approach. For instance, Automated Land Evaluation System (ALES) is a computerised framework that allows estimating crop and consequent economic production; Decision Support System for Agrotechnology Transfer (DSSAT) is an explicit suite of crop plant production models [31]. More recently, the data generated in agriculture operations, gathered largely through remote sensing, has been demonstrated to be suitable for new developed computational techniques. Those methods brought bold and powerful ways of examining particular agriculture problems, being applied essentially to image processing and data analysis [32].

Such models contemplate equilibria between every agent constituting society–nature interactions and processes with well established frontiers, corresponding to intransgressible planetary boundaries [33]. Furthermore, underlying these models is usually a time series approach that can be used to address questions of causality, trends, and the likelihood of future outcomes. Hence, policy makers will have unveiled solid ground for the development and implementation of sustainable management strategies and pathways for a greener future.

1.4 Artificial Intelligence

It has already been verified the use of modern techniques, such as Artificial Neural Networks (ANNs), as a method for modelling and forecasting in environmental studies for Portugal. In a study, the useful exergy concept was used to predict energy consumption based strictly on the economic evolution of the country [34].

Classical biophysical models are theory-driven, or, differently, implemented with a bottom-up approach. This way of description poses a difficulty of interpreting non-linearities between all interplaying processes. On the other hand, there are alternative techniques that present themselves as data-driven or, differently, as top-down approaches. The idea is to scour data in order to find novel and useful patterns/relationships that might otherwise remain unknown with more traditional methods [35].

There is a strong need to deal with the already predictable high number of covariates in a way that enables the extraction of explicit answers on how materials, land and energy have been used to produce value for an economy over time. As mechanistic relationships between variables are unknown and conventional statistics is limited by high covariance and indeterminate causality, the application of

an advanced toolkit drawn from Computer Science will attempt to produce clarity regarding the interdependencies between drivers of change and outcomes. Therefore, Artificial Intelligence (AI)'s versatility and power are expected to commensurately match the challenge. More precisely, Machine Learning (ML) methods, a subcategory of AI, shall be put through. As argued, such methods are advantageous in situations where one does not posit an underlying process or any rules of such. The focus is rather on identifying patterns that describe the processes' behaviour in ways useful to predict the outcome of interest [36].

It is possible to condense all applications (of where ML has been predominantly applied) to four first level categories. Those are the management of crop, livestock, water, and soil. In crop management some sub-categories are yield prediction, disease detection, weed detection, crop quality, species recognition, fruits counting, and crop type classification (that could be generalised to land cover classification). Regarding livestock management, there are the animal welfare and livestock production sub-categories. In terms of water management, the main object of study is usually the evapotranspiration. Finally, soil management consists of studying agricultural soil properties, such as the estimation of soil drying, condition, temperature and moisture content [37]. Those problems deal with either classification or prediction, with more prevalence of classification.

For all of the previous subjects, the following ML methods have been deployed: support vector machines, Bayesian models, deep learning, decisions trees, ensemble learning, instance based models, ANNs and Recurrent Neural Networks (RNNs). Each one of these algorithms were designed for different purposes and comprehensive reviews with correspondence between problems under study and suitable models already exist [37, 38].

1.5 Objectives

Being aware of the importance of this outstanding topic, this thesis is explicitly going to dive into the Portuguese agricultural sector's sustainability and target an initial analysis of the period comprised between 1961-2016, as substantial changes took place as a consequence of the transition from organic fertilisers to chemical-based fertilisers, increased industrialisation, and the entry into the European single market.

The goal of the thesis is to contribute towards the development of a biophysical model framework. The model will be particularly applied to the evolution of Portuguese land use, with special focus on the agriculture, in order to explain how land and energy have been used to produce value for the Portuguese economy. Moreover, the model will be employed to forecast plausible pathways for the future of the sector, assessing land use emissions, up to 2030. More concretely, we are first going to try to explain the relationship between the land use distribution in terms of final exergy usage in the agricultural sector [39] and economic growth explained in terms of Gross Domestic Product (GDP). As a second step, we are going to try to study how the former three covariates, land use distribution, economic growth, and final exergy, relate to GHG emissions. By comprehensively studying individual sectors that constitute the global economic machine we expect to gain a more precise perspective on the reciprocal dynamics.

The work will be carried out at the macro level, taking the entire Portuguese agricultural sector as a whole. This goal will require the compilation of available time series of land use, energy usage, economic development, and GHG emissions. In case of missing data, the work will involve the application of data filling and extrapolation methods for completing incomplete time series. Using the time series data, the work will then involve the application of ML/AI tools for analysing hidden patterns in the data and establishing connections between variables. Ultimately, it is intended to assess how changes in energy usage and economic development affect farmer's choices regarding land use, and how those choices are translated in terms of sustainability.

Further, this work will be strongly multidisciplinary, as a set of tools from different areas will be applied to new problems regarding sustainability and agricultural production. Results will be interpreted depending on past and expected future policies. The effects of past policies, such as the wheat campaign or support offered under the framework of the Common Agricultural Policy (CAP) after joining the European Union (EU) will be visible in the data series. For the future, the model should also be able to provide some insight regarding the effects of expected new policies in economic and energy sectors, and the effects of the novel coronavirus pandemic in projected trajectories of evolution for economic and energy sectors. As some important policies for the future are novel and the models developed here were not trained to assess their effects, the scenario-building capacity of the models will be tested qualitatively.

2

Background

2.1 Agriculture, Energy and Economy

The introduction of the first part of this chapter is influenced by the book by Smil [1]. The thermodynamics concept of exergy is then associated with both economic growth and agriculture [40–42].

2.1.1 Agriculture and Energy

Energy can be recognised as the only universal currency: one of its many forms need to be transformed to get anything done. Universal demonstrations of these transformations range from the rotations of galaxies to thermonuclear reactions in stars. On Earth, they range from the forces of plate tectonics that wreck ocean floors and raise new mountains to the cumulative erosive impacts of raindrops. Life on Earth would be impossible without the photosynthetic conversion of solar energy into phytomass, it underpins all higher life. Humans depend on this transformation for their survival, and many more energy flows to support their existence within increasingly complex societies [1].

Evolution of Human societies has resulted in larger populations, a growing complexity of social and productive arrangements, and a higher quality of life for a growing number of people. From a biophysical perspective, both prehistoric human evolution and the course of history can be seen as a pursuance of controlling greater stores and flows of more concentrated and more versatile forms of energy and converting them, in affordable ways at lower costs and with higher efficiencies, into heat, light, and motion [1]. Alfred Lotka, an American mathematician, chemist, and statistician, generalised this tendency in his law of maximum energy as follows:

> *"In every instance considered, natural selection will so operate as to increase the total mass of the organic system, to increase the rate of circulation of matter through the system, and to increase the total energy flux through the system so long as there is present an unutilised residue of matter and available energy."*
>
> – Alfred Lotka, 1922

Similarly to what happens with any non photosynthesising organism, the most fundamental human energy need is food. This inescapable necessity, along with a combination of factors such as population growth and environmental stress, could persuasively explain the agricultural origins, according to [43], recognising that the transition to permanent cropping was driven by both natural and social factors. Elementally, crop cultivation is an effort to ensure an adequate food supply, and hence agriculture's origins and evolution could be explained as yet another instance of an energy imperative [1].

2.1.2 Exergy: why and what is it?

A better way of tracing the influence of energy is to express it in terms of its potential usefulness, *i.e.*, the actually delivered heat, light, and motion. This subdivision of a given amount of energy is called exergy, and it is not a conserved property because it can be transformed into anergy by irreversibilities[1], as a consequence of the 2^{nd} law of thermodynamics[2] [44].

The concept of exergy is defined as the maximum work obtainable when a system is brought to a reference state of thermodynamic equilibrium by means of completely reversible processes[3] or, differently, through ideal energy conversion processes. Therefore, a reference state must be defined. In standard uses, it is defined to be a state of thermodynamic equilibrium characterised by the same temperature, pressure and chemical composition as the environment. Exergy is also a thermodynamic measure of energy quality[4], measuring the availability to perform work of a certain amount of energy, given reference environmental conditions [44].

2.1.2.1 Exergy stages

The diversity of potential usefulness of energy in moments where energy usage occur leads to the necessity of accounting for the different stages of the energy flow. Three stages are often defined: primary, final and useful (see figure 2.1). An exergy approach to the energy flow is readily used because: 1) it enables a measure of the quality of energy, related to its potential usefulness, capable of being systematically applied across all types of energy carriers and end-uses, and 2) if one were to use an energy approach, a problem of consistency would arise due to the diversity of thermodynamic potentials[5] used to characterise final energy measurements of each energy carrier delivered by the energy sector. By using exergy, one accounts just for potential work, a unified single-potential approach [40].

[1] Examples of irreversibility sources are friction, heat transfer through a finite temperature difference, unrestrained expansion, mixing, etc. Irreversibilities occur only due to the interaction of the system with the exterior

[2] The Kelvin–Planck statement: "It is impossible to construct a device that will operate in a cycle and produce no effect other than the raising of a weight and the exchange of heat with a single reservoir."; The Clausius statement: "It is impossible to construct a device that operates in a cycle and produces no effect other than the transfer of heat from a cooler body to a warmer body." [44]

[3] A reversible process for a system is by definition a process that, once having taken place, can be reversed and in so doing leave no change in either system or surroundings.

[4] Energy quality is defined as a measure of the energy potential or ability to be utilised for useful work or another form of energy.

[5] Common used thermodynamic potentials are enthalpy and internal energy.

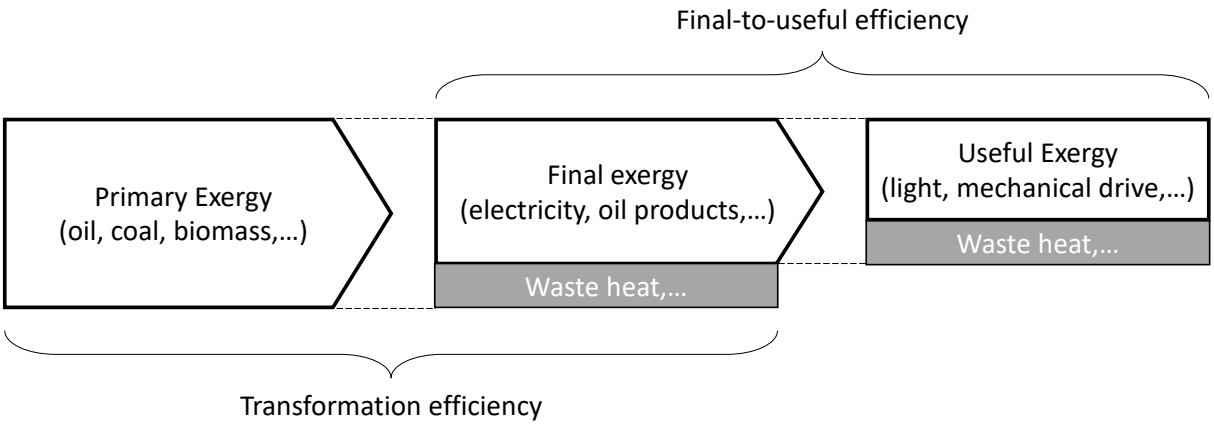


Figure 2.1: Primary-to-useful exergy flow.

The primary stage, often mentioned as primary exergy, is the first stage of the exergy flow and refers to the exergetic potential of the natural energy resources. The second stage of the exergy flow refers to the final exergy. This stage situates the exergy analysis after all energy transformation processes and immediately before their end-uses. Final exergy refers to exergy content the energy sector delivers to final consumers. It is the primary exergy discounted by all exergy losses from the transformation processes occurred in the energy sector, such as electricity generation, oil refining, co-generation, transport and distribution losses. These processes provide the next energy carriers to final consumers: coal products, oil products, natural gas, combustible renewables, electricity, heat, food, and feed [40].

Thirdly, we have the useful exergy, which measures the result of an energy use rather than the amount transferred to a final use, *i.e.*, the actual amount of exergy delivered to a final function, such as the mechanical work used by a car from its fuel, the exergy of the heat provided to a blast furnace, or the light emitted by an electric lamp. Essentially, it measures the proportion of energy resources used for productive purposes within a system.

2.1.3 Useful Exergy and Economic Growth

In an exergy framework, the useful stage of the energy flow accounts for satisfied energy needs. This means that a useful energy analysis with an exergy approach leads to a measure closer to the productive energy uses within an economy, providing better insights on the relation between economic growth and energy uses. It has been acknowledged as an appropriate stage for energy accounting, independent from efficiency improvements and technological progress at the different stages of the exergy flow [40].

Exergy is extensively presented in the literature as a good variable for economic and sustainability assessments of energy, as it accounts for the quality in use and conversion of energy vectors and materials [41]. Particularly, for Portugal, Serrenho et al. [41] concluded that there is an approximately linear relationship between useful exergy and the GDP throughout 1960 to 2010. Practically, this means that 60 years ago we needed $1MJ$ of useful exergy per unit of GDP ($1€$ in 2010 prices), and the same relation still holds today.

2.2 The Fundamentals of Machine Learning

The second part of this chapter is a brief introduction to key concepts required for the understanding of the problems and the goals of this project. The theoretical overview on ML is based on three of the most important books in the field – the books by Goodfellow, Bengio, Courville, and Bengio [45], Géron [46] and Nielsen [36], the last two providing state-of-the-art content on deep learning and time series.

2.2.1 What is Machine Learning?

> *"A computer program is said to learn from experience E with respect to some task T and some performance P, if its performance on T, as measured by P, improves with experience E."*
>
> – Tom Mitchell, 1997

ML is the field of study that gives computers the ability to learn from data. For instance, let us take the classic example of the spam filter. Our spam filter is a ML program that, given samples of spam and nonspam emails, can discover how to flag spam (classification). The samples that the program uses to learn are called the training set. Each training sample is called a feature or a training instance. In this case, the task T is to flag spam for new emails, the experience E is the training data, and the performance measure P could be defined by, for example, the ratio of accurately classified emails [46].

2.2.2 Why use Machine Learning?

Consider that we want to write a spam filter using standard programming techniques. Firstly, we would examine what does spam looks like. Typically, words or phrases such as "free" and "credit card" tend to occur frequently, also patterns in the email's body and sender's name could be noticed. Secondly, we would encode a detection algorithm with a set of rules for each recognised pattern, flagging emails if a number of these patterns were identified. Finally, we would run the program and iterate both of the previous steps until reaching satisfactory results. Given the richness of possible patterns, the program would probably become an extensive list of rules [46].

Opposing, a ML-based spam filter automatically learns to recognise which patterns are good spam predictors by detecting suspiciously recurrent patterns of words in the spam examples, compared to the nonspam. In a scenario where spammers detect that emails containing a certain word are getting blocked and decide to take action and replace it by an identical word, a spam filter using standard programming techniques would need to be updated to flag it, in contrast to the ML approach that starts flagging new words without users' intervention [46].

This type of algorithm falls into the category of supervised learning, which is the same category of the algorithms used in this thesis. Given the right type of architecture, the mentioned solution could be generalisable for a panoply of problems, such as time series analysis, which is of interest here. It is possible to implement for weather, economics, medicine and astronomy forecasting problems that also learn and adapt dynamically on the fly. Irrespective of the fact that one is dealing with classification, meaning we have discrete output variables, or regression problems, where we have continuous output

variables, in both positions the task is approximating a mapping function from input variables to output variables.

2.2.3 Types of Machine Learning Systems

Within ML itself there is a major branch called Deep Learning (DL). DL is an attempt to mimic the activity in layers of neurons in the neocortex, *i.e.*, build an ANN representation. If we draw a graph showing how these concepts are built on top of each other, the graph is deep, with many layers, which helps to have multiple states of nonlinear feature transformation. For this reason, this approach is named *deep* learning. DL algorithms can be generally categorised as unsupervised or supervised, depending on what kind of experience they have during the learning process [45].

Unsupervised deep learning algorithms experience a dataset containing features, then learn valuable properties of the structure of this dataset. In the context of DL, usually the goal is to learn the entire probability distribution that generated a dataset, whether explicitly as in density estimation or implicitly for tasks like synthesis or denoising. There are unsupervised learning algorithms that perform different roles, such as clustering, which consists of dividing the dataset into groups of similar examples [45].

Supervised deep learning algorithms experience a dataset containing features, but each example is associated with a target or label. For instance, an email dataset is annotated with the classification of whether each email is spam or not. A supervised learning algorithm can analyse the email dataset and learn to classify emails into two categories [45].

In this thesis, handling time series is strictly necessary and it happens to be a suited application of a type of DL algorithm called RNNs due to their architecture being designed for processing sequential data [45]. It provides the possibility of modelling highly complex and nonlinear temporal behaviour without having to guess at functional forms [36].

2.2.4 Deep Learning

DL has roots in biology and mathematics. Combining both areas, researchers aspired to build intelligent machines, that mimicked the human brain [47, 48]. While manifesting mathematical inspiration by presenting universal approximation theorems for various activations functions (equivalent to the neuron response) [36], associated with a growing computing capability and availability coupled with the expansion of ML, it became clear that with enough data and parameters, complex systems could be modelled and predicted. With DL all these ideas were used to propel the creation of networks described by millions of parameters, trained on large data sets, and with a theoretical grounding that a neural network should be able to represent an arbitrary and nonlinear function to a substantial degree of accuracy [36].

2.2.4.1 Fundamental Algorithm: The Perceptron

To better understand the algorithmic mechanism behind the further presented models, it is important to start with the definition of the most fundamental type of neural network: a perceptron (or a single layer of Threshold Logic Units (TLUs), figure 2.2). A perceptron is the elementary type of neural network.

The reciprocal and most direct comparison is normally the neuron. The human brain consists of a large number of neurons and these neurons allow us to “learn”, “understand” and “memorise”. A crucial observation is that the human brain can learn by establishing suitable communication lines between neurons as well. Hence, it is not only the neurons that cause “intelligence” but the connections between the neurons (which are about 5.000.000.000.000 connections in an adult brain - too many to simulate on a computer) [46]. Nonetheless, one can simulate a smaller set of neurons and their connections in an attempt to achieve the ability for a machine to “learn” and to “memorise” basic tasks.

Firstly, in the forward phase, a perceptron:

1. Takes an array or list of numbers as inputs;
2. Computes a weighted sum of all inputs;
3. Uses an activation function to compute the response;

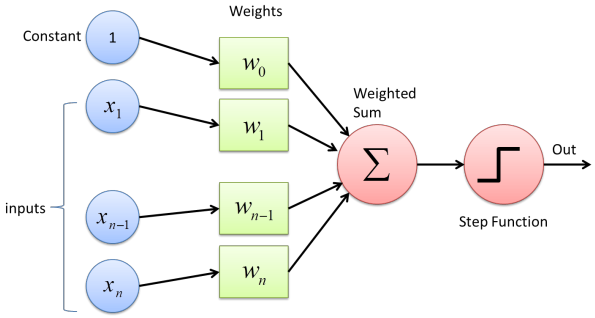


Figure 2.2: *Threshold logic unit* (TLU) [49].

A TLU is an artificial neuron which computes a weighted sum of its inputs and then applies a function to that sum and outputs the result. Multiple TLUs compose a perceptron.

The next step consists of the backward phase in which the output of the network is compared with a given target value (supervised learning). As this is a parametric model, the error between both output and target values can be minimised with an error backpropagation algorithm [50] by adjusting the weights, which are randomly initialised at the beginning. The amount by which each weight needs to be changed in order to minimise the error is unknown a priori. However, the direction in which each weight needs to be changed to reduce the error can be discovered via gradient descent method. The direction of a weight change is computed by using the sign of the gradient of the cost function with respect to each of the weights and by using a tunable learning rate parameter that quantifies the magnitude of the adjustment [46].

The forward phase and reverse phase are repeated for a number of iterations (epochs) until the network’s error no longer decreases. A trained network is said to be a ‘model’ which ‘encodes’ the problem’s domain, and the model can then be applied to unseen data and the network will produce an output in accordance with the input pattern [46].

In order to add complexity, it is possible to form layers composed of perceptrons, and when all the neurons in a layer are connected to every neuron in an immediately adjacent layer (*i.e.*, its input neurons), the layer is called a fully connected layer or a dense layer. Moreover, an extra bias feature is generally added and it is typically represented using a special type of neuron called a bias neuron, which outputs a constant value of 1, allowing a translation of the activation function. Without the presence of a bias neuron, each neuron would take the input and multiply it by a weight, with nothing else added to the equation. Their weight is estimated as part of the overall model [46].

The archetypal example of a DL model is the feed-forward deep network or Multi Layer Perceptron (MLP). Comparatively to a perceptron, a MLP is just a more complex mathematical function mapping a set of input values to output values. The function is constructed by composing simpler functions. Each application of a different mathematical function can be thought as providing a new representation of the input [45]. It is composed of one (pass-through) input layer, one or more layers of TLUs called hidden layers, and one final layer of TLUs named output layer. If an ANN contains a deep stack of hidden layers then it is known as Deep Neural Network (DNN) [46].

However, MLP is a 'black box': we do not know (nor understand) how the model has encoded the information (there is no rule which we can extract), and this principle is also valid for all the models here presented. Some limitations transversal to this type of models are:

- Asymptotic algorithm: error reduction slows down with the approach to a minimum;
- Convergence could happen on a local minimum which is not necessarily the global minimum;
- It requires the setting of network parameters, such as number of hidden layers, number of neurons in each layer and initialisation of the weights [46].

These limitations can be addressed by using a momentum term or adaptive learning rate and by doing a careful network initialisation (trial and error is acceptable). MLPs are widely used for classification and regression problems, and are proven to solve any continuously differentiable function to any precision making it possible to greatly benefit from them [46, 51]. Given the importance of understanding the dynamics during training it is worth restating in detail how these algorithms are set up as well as the consecutive steps that they generally go through.

2.2.4.2 Model Set Up

For clarity, we will start by stating the notation in the following equations:

- n as the number of features;
- m as the number of instances in the dataset considered;
- $\mathbf{x}$ as the instance's feature vector, ranging from x_0 to x_n;
- x_i as the i^{th} feature value;
- $\mathbf{x}^i$ as a vector of all the features values of the i^{th} instance in the dataset;
- $\mathbf{X}$ as a matrix consisting of all the feature values of all instances in the dataset;
- y^i as the label of the i^{th} instance in the dataset (desired output for that instance in 1D prediction);
- $\boldsymbol{\theta}$ as the model's parameter vector, consisting of the bias term θ_0 and the feature weights θ_1 to θ_n;
- h as the system's prediction (hypothesis) function, using the model parameters $\boldsymbol{\theta}$ [46].

1. The network is initialised with random weights and a predefined configuration of number of input neurons, hidden neurons, output neurons and layers.

   It is crucial to initialise all the hidden layers' connection weights randomly, otherwise training will fail. If, for instance, all weights and bias neurons are initialised to any constant value the neurons in a given layer will be perfectly identical, and consequently, backpropagation will affect them in the same way, resulting in them remaining identical. This situation is equivalent to having only one neuron per layer. If instead they are randomly initialised the symmetry is broken, allowing the training of a diverse set of neurons [46].

2. The algorithm has to measure the network's error (or performance). This happens by the means of a cost function (loss function) that compares the desired output and the actual output of the network, returning a measure of the error.

   Typical loss functions used for regression problems, as in the present case, are:

   (a) Root Mean Squared Error (RMSE)

   $$RMSE(\mathbf{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(h(\mathbf{x}^i) - y^i)^2}\ , \tag{2.1}$$

   where m denotes the number of instances in the dataset in which the RMSE is being measured. This cost function penalises greatly large errors, thus it is more sensitive to outliers;

   (b) Mean Absolut Error (MAE)

   $$MAE(\mathbf{X}, h) = \frac{1}{m}\sum_{i=1}^{m}|h(\mathbf{x}^i) - y^i|\ . \tag{2.2}$$

   In a context where there are many outliers it may be preferable to use the MAE performance measure;

   (c) Huber Loss

   $$L_\delta(\mathbf{X}, h) = \frac{1}{m}\sum_{i=1}^{m}\begin{cases}\frac{1}{2}(h(\mathbf{x}^i) - y^i)^2 & ,\quad \text{for } |h(\mathbf{x}^i) - y^i| \leq \delta \\ \delta|h(\mathbf{x}^i) - y^i| - \frac{1}{2}\delta^2 & ,\quad \text{otherwise}\ .\end{cases} \tag{2.3}$$

   This is a combination of the previous two. The Huber loss is characterised by having a quadratic behaviour when the error is smaller than a threshold δ and linear otherwise [46].

3. An optimiser is chosen in order to precise how backpropagation occurs.

   Gradient descent is a generic optimisation algorithm capable of finding optimal solutions to a wide range of problems. In simple terms, the idea is to modify parameters iteratively with a view to minimising a cost function. The local gradient of the error function with regard to a parameter vector $\boldsymbol{\theta}$ is measured and the algorithm goes in the direction of descending gradient, aiming to reach a gradient of zero, *i.e.*, a local minimum (figure 2.3) [46].

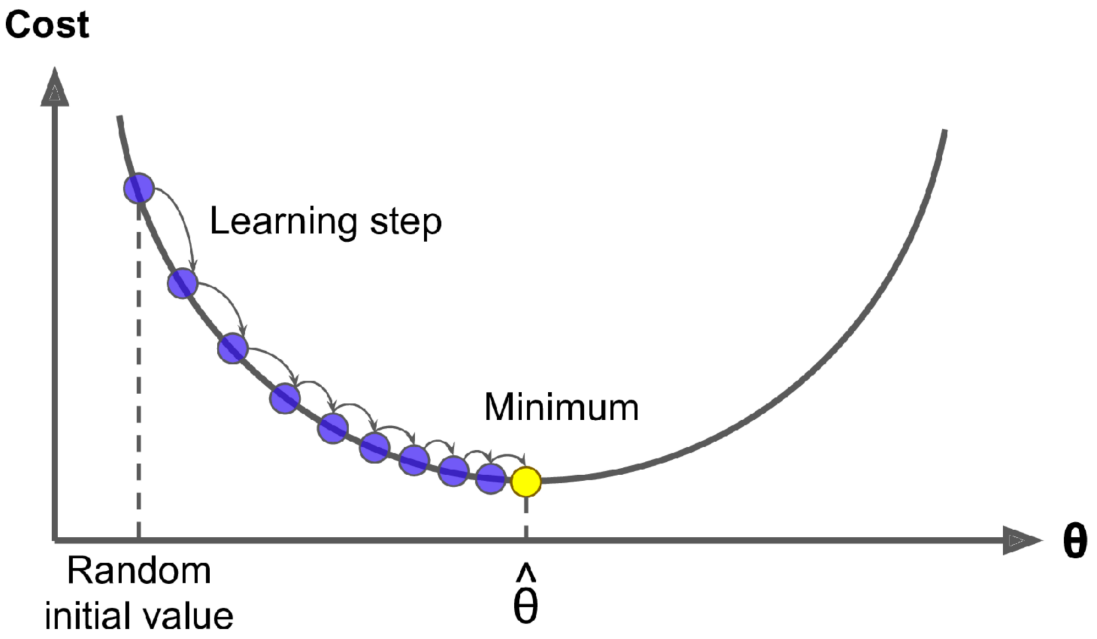


Figure 2.3: Cost function to minimise with a given optimisation algorithm [46].

The models are initialised randomly and get adjusted iteratively to minimise the cost function. The learning step size is proportional to the slope of the cost function, so the steps gradually get smaller as the parameters approach the minimum.[a]

[a] Permission to use this image was granted from the publisher.

Mathematically, this corresponds to first calculating the partial derivatives of a given cost function (RMSE, for instance) with respect to each tunable parameter, θ_j, of a randomly initialised parameter vector $\boldsymbol{\theta}$ and storing the results in a gradient

$$\frac{\partial}{\partial \theta_j} RMSE(\boldsymbol{\theta}) = \frac{2}{m} \sum_{i=1}^{m} (\boldsymbol{\theta}^\top \cdot \mathbf{x}^i - y^i) \cdot x_j^i \text{ [46].} \tag{2.4}$$

Instead of computing these partial derivatives individually, it is also possible to have them computed in one go for each model parameter

$$\nabla_{\boldsymbol{\theta}} RMSE(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} RMSE(\boldsymbol{\theta}) \\ \frac{\partial}{\partial \theta_1} RMSE(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial}{\partial \theta_n} RMSE(\boldsymbol{\theta}) \end{pmatrix} = \frac{2}{m} \mathbf{X}^\top (\mathbf{X}\boldsymbol{\theta} - \mathbf{y}) \,. \tag{2.5}$$

This formula involves calculations over the full training set $\mathbf{X}$, at each gradient descent step. As the algorithm uses the whole batch of training data at every time-step (making it slow for large training sets), this algorithm is called batch gradient descent [46].

Then, to solve for the gradient, we iterate through the data points using a new $\boldsymbol{\theta}$ at each iteration and compute the partial derivatives. In each time-step, there is an attempt to decrease the cost function, until the algorithm converges to a minimum

$$\boldsymbol{\theta}^{\text{next step}} = \boldsymbol{\theta} - \eta \cdot \nabla_{\boldsymbol{\theta}} RMSE(\boldsymbol{\theta}) \ , \tag{2.6}$$

where η is the learning rate. It is worth remarking that there are two other algorithms called Stochastic Gradient Descent (SGD) and mini-batch gradient descent. What they share in common is that instead of computing gradients based on the full training set, they rely on partial data at every step. The former computes the gradient based on one random instance while the latter computes it based on small random sets of instances called mini-batches. Working on less training data makes the algorithms run faster due to having fewer data to manipulate at every instance, thus enabling the training on larger training sets. On the other hand, due to its stochasticity, these algorithms are much less regular than batch gradient descent: instead of softly decreasing until reaching a minimum, the cost function value will go up and down, decreasing only on average. Once the algorithm stops, the final parameter values are expected to be good, though not optimal. However, if it happens that the cost function is irregular this can help the algorithm to escape from a local minima, therefore these stochastic algorithms have a better chance of finding the global minimum than the batch gradient descent does [46].

A parameter of major importance introduced in equation 2.6 is the size of the steps, specified by η, the learning rate. In case the learning rate is too small, the algorithm will need numerous iterations to converge. Conversely, if the learning rate is too high we might experience a phenomenon of divergence, with the algorithm failing to find a good solution.

There are faster variants of the regular gradient descent optimiser. The most popular algorithms for training deep neural networks are:

- Momentum Optimisation;
- Nesterov Accelerated Gradient;
- AdaGrad;
- RMSProp;
- Adam;
- Nadam;
- AdaMax.

Each of which are extensively explored in [45, 46] and will be adopted in this work. All these optimisation techniques rely on Jacobians (first-order partial derivatives). More robust algorithms based on Hessians (second-order partial derivatives) are available, but with them comes the price of complexity: there are p^2 Hessians per output (p being the number of trainable parameters), whereas for the Jacobians there are p per output. This caveat compromises the speed and memory [46].

4. Finding an adequate learning rate is now required. A frequent procedure consists of training the model for a few hundred iterations, exponentially increasing the learning rate from a residual value to a very large value, and then analyse the learning curve and select a slightly lower learning rate than the one that produces a curve that starts shooting back up, as in figure 2.4.

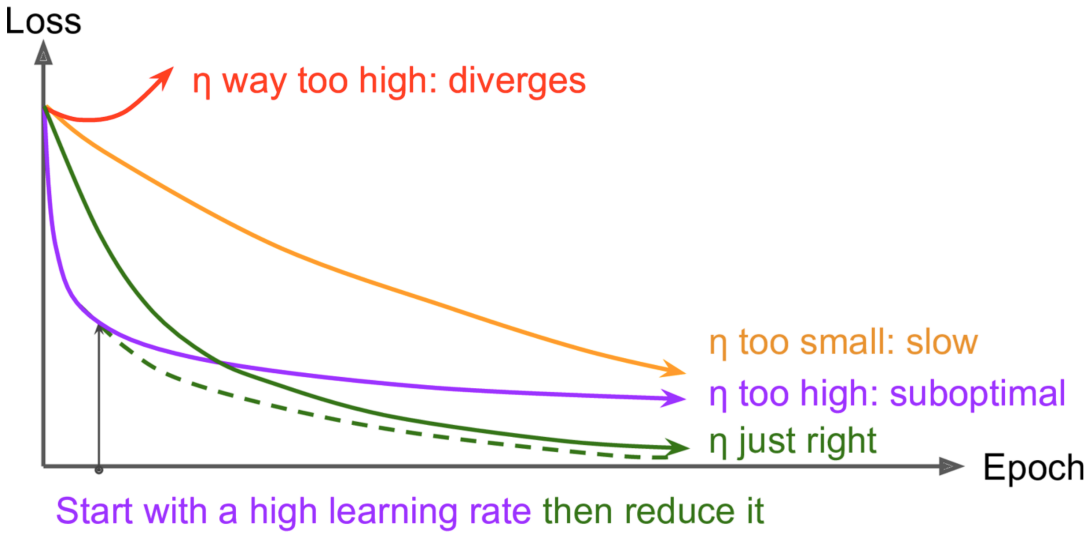


Figure 2.4: Learning curves for multiple learning rates η [46].

Training a model several times with distinct learning rates and then plotting the evolution of the loss is a way of finding an adequate learning rate.[a]

[a] Permission to use this image was granted from the publisher.

A more efficient approach is applying a dynamic learning rate. By starting with a large learning rate and then reducing it once the training stops making quick improvements, allows us to reach a good solution faster than with the optimal constant learning rate. There are several strategies applicable, such as:

- Power Scheduling;
- Exponential Scheduling;
- Piecewise Constant Scheduling;
- Performance Scheduling;
- 1cycle Scheduling.

Each of the schedules are variants of a simple concept, and their full description can be found in [46]. The chosen learning schedule for this work is the performance scheduling, which essentially consists of measuring the validation error every N steps and reducing the learning rate by a factor of λ when the error stops dropping.

5. The choice of the activation function for each of the neurons also plays one of the most important roles. It is also relevant to acknowledge why activation functions are used. The answer lies on the following: if several linear transformations are linked, in the end we get a linear transformation.

Therefore, if there is no nonlinearity between layers, a deep stack of layers would be equivalent to a single layer and we would not solve very complex problems with that. Conversely, as previously mentioned, a large enough DNN with nonlinear activations can theoretically approximate any continuous function [51].

Gradients often get too small as the algorithm progresses down to the lower layers. As a consequence, the gradient descent update leaves the lower layers' connection weights practically unaffected, and training does not converge to a good solution. This problem is known as the vanishing gradients problem. The opposite also happens, in which the gradients grow too big until layers get large weight updates and the algorithm diverges (exploding gradients). Broadly speaking, DNNs are affected by unstable gradients - different layers may learn at widely different paces [46].

Due to the aforementioned unfortunate behaviour, DNNs were largely abandoned in the early 2000s. However, in 2010 the problem was successfully tackled by a paper from Glorot and Bengio [52]. The authors found that the logistic sigmoid activation function, a popular choice at the time, and the weight initialisation technique that was also very popular at the time (a normal distribution with a mean of 0 and a standard deviation of 1) were the main culprits [46].

The logistic sigmoid function is characterised by saturating at 0 or 1 for large values, with an almost null derivative, and behaves linearly for values close to 0,

$$\sigma(z) = \frac{1}{1 + e^{-z}} \ , \tag{2.7}$$

as observable in figure 2.5.[8]

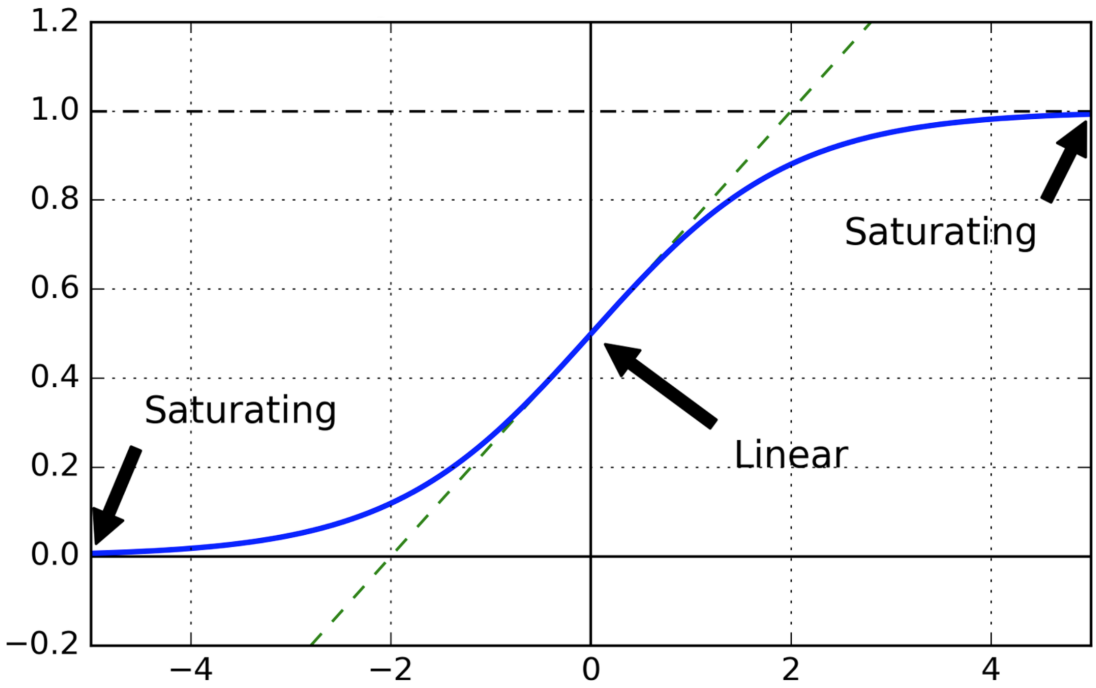


Figure 2.5: Logistic activation function [46].

[8] Permission to use this image was granted from the publisher.

They demonstrated that, with this activation function and the initialisation scheme used at the time, the variance of the outputs of each layer was much larger than the variance of its inputs. Proceeding in the network, the variance kept increasing after each layer until the activation function saturated at the top layers. Thus, when backpropagation occurred, it had virtually no gradient to propagate back through the network, and the remaining gradient was disappearing as backpropagation progressed down through the top layers, so there was not much left for the lower layers. This saturation was made even worse by the fact that the logistic function has a mean of 0.5, not 0. By comparison, the hyperbolic tangent has a better behaviour in deep networks because it has a mean of 0 [46].

In their paper, Glorot and Bengio [52] proposed ways, now known as Glorot Initialisation, to significantly alleviate the unstable gradients problem. They pointed out that the signal needs to flow properly in both directions: forward direction when making predictions, and in the reverse direction when backpropagating gradients. The main goal is to avoid the signal to get diluted and vanish or to explode and saturate [46].

For the signal to flow properly, the authors argued that the variance of the outputs of each layer should be equal to the variance of its inputs and also that the gradients should have equal variance before and after flowing through a layer in the reverse direction. Concretely, they proposed that the connection weight of each layer must be initialised randomly as described in equations 2.8 and 2.9, where $fan_{avg} = (fan_{in} + fan_{out})/2$, and fan_{in} and fan_{out} are defined as the number of inputs and neurons from a layer. Respectively,

$$\text{normal distribution with mean 0 and variance } \sigma^2 = \frac{1}{fan_{avg}} \tag{2.8}$$

or

$$\text{uniform distribution between } -r \text{ and } +r, \text{ with } r = \sqrt{\frac{3}{fan_{avg}}} \, . \tag{2.9}$$

Using Glorot initialisation can speed up training significantly, and it is one of the discoveries that led to the success of DL. A diversity of initialisations and associated activation functions have been proposed ever since, with each providing advantages for certain tasks. These strategies differ only by the scale of the variance and whether fan_{avg} or fan_{in} is used in the variance equations [46].

Alternatively, nonsaturating activation functions have also proven to behave very well in DNN. In particular, the Rectified Linear Unit (RELU) (figure 2.6[9]), mostly because saturation for positive values does not occur, and because it is fast to compute. The initialisation strategy for the RELU activation function (and its variants, described shortly) is sometimes called *He* initialisation, after the paper's first author [46],

$$ReLU(z) = max(0, z) \, . \tag{2.10}$$

[9]Permission to use this image was granted from the publisher.

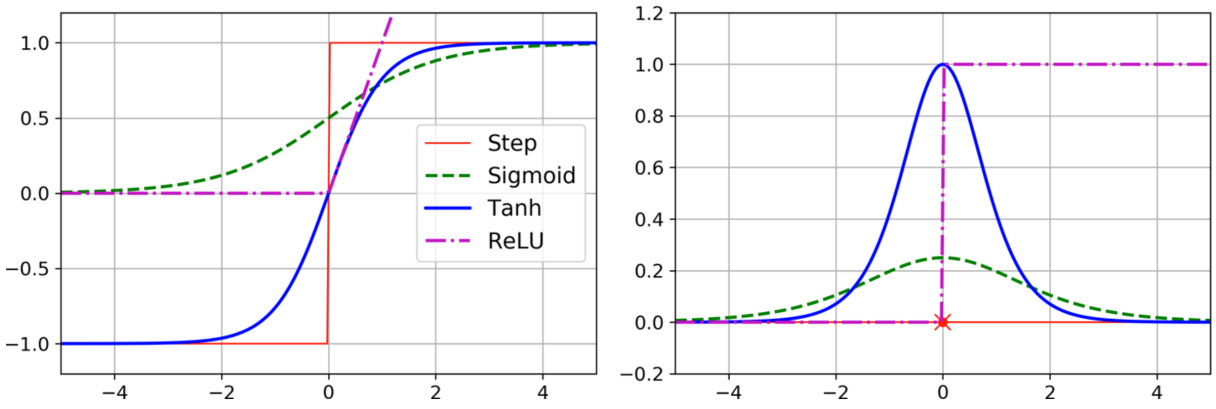


Figure 2.6: Activation functions (left) and their derivatives (right) [46].

Unfortunately, the RELU activation function suffers from a problem known as the dying ReLUs. During training, some neurons stop outputting anything other than 0, effectively 'dying'. A neuron dies when its weights get adjusted in such a way that the weighted sum of its inputs are negative for all features in the training set. Under these conditions, the output consists only on 0s, and gradient descent does not affect it anymore because the gradient of the RELU function is zero when its input is negative [46].

To tackle this problem, there are variants of the RELU function, such as the leaky RELU (see figure 2.7),

$$LeakyReLU_{\alpha}(z) = max(\alpha z, z) \ , \tag{2.11}$$

where the hyperparameter α defines how much the function 'leaks', *i.e.*, it is the slope of the function for $z < 0$. This small slope ensures that leaky RELUs will not 'die' [46].[10]

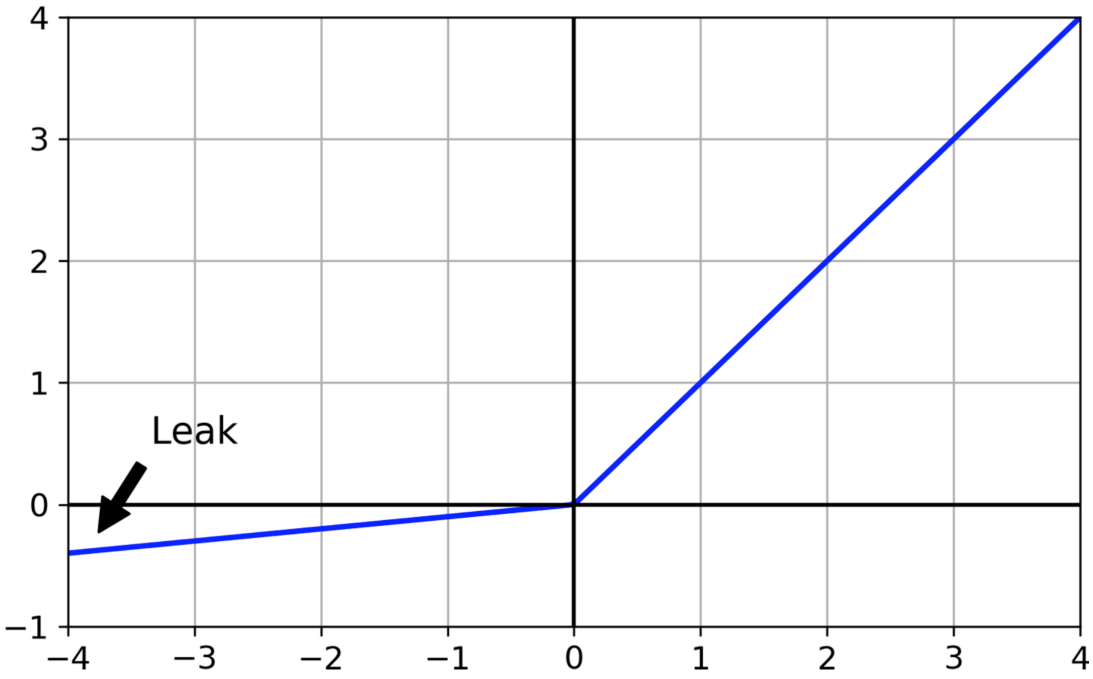


Figure 2.7: Leaky ReLU: identical to the ReLU [46].

[10] Permission to use the following image was granted from the publisher.

Later, in 2015 Clevert et al. [53] proposed a new activation function called the Exponential Linear Unit (ELU) that outperformed all the ReLU variants in the authors' experiments. Ultimately, training time was reduced, and the neural network performed better on the test set. Figure 2.8[11] graphs the function [46],

$$ELU_\alpha(z) = \begin{cases} \alpha(e^z - 1) & , \text{ for } z \leq 0 \\ z & , \text{ otherwise .} \end{cases} \tag{2.12}$$

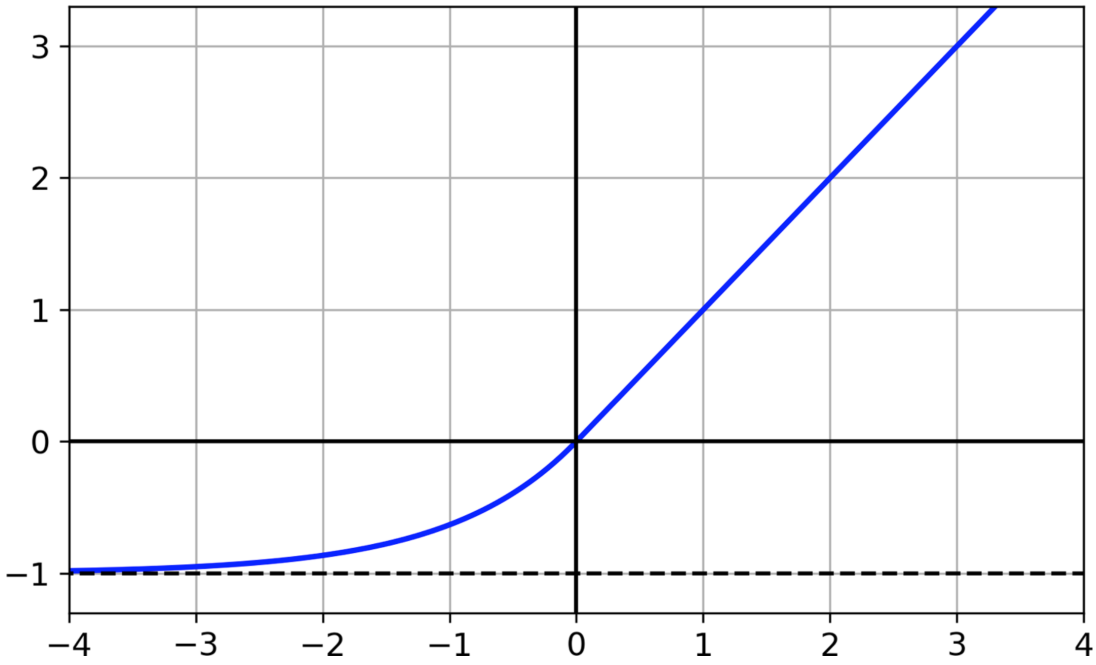


Figure 2.8: ELU activation function [46].

In comparison, both ELU and ReLU share similarities, with the ELU having the following distinctions:

- It outputs negative values when $z < 0$, allowing the unit to have an average output closer to 0, and simultaneously helping alleviate the vanishing gradients problem. The hyperparameter α sets the value that the ELU function asymptomatically approaches when z is a large negative number. Usually it is defined to be 1, however, it is tunable like any other hyperparameter;
- It has a nonzero gradient for $z < 0$, thus avoiding the 'dead' neurons problem;
- If α is equal to 1, then the function is smooth on its entire domain, including around $z = 0$. In turn, it helps speeding up gradient descent since it does not vary as much around $z = 0$.

The main drawback of the ELU activation function comes from the fact that it is slower to compute than the ReLU function and its variants, because it uses the, computationally expensive, exponential function. Its faster convergence rate during training compensates for the slow computation,

[11] Permission to use this image was granted from the publisher.

nonetheless, in terms of test time an ELU network is expected to be slower than a RELU network [46].

Then, a 2017 paper by Klambauer et al. [54] introduced the Scaled Linear Unit (SELU) activation function, which is a scaled variant of the ELU activation function. Its initialisation strategy is called *LeCun*, and the authors demonstrated that if we were to build a neural network composed of a stack of dense layers, and if all hidden layers use the SELU activation function, then the network would self-normalise. Essentially, this means that the output of each layer would tend to conserve a mean of 0 and standard deviation of 1 during training, which also solved the vanishing/exploding gradients problem. As a result, the SELU activation function often considerably outperforms other activation functions for, especially deep, neural networks. There are, however, a few conditions that should be fulfilled to ensure self-normalisation:

- The input features need to be standardised, *i.e.*, they must have to have a mean of 0 and an unitary standard deviation;
- All hidden layers' weights must be initialised with LeCun normal initialisation;
- The network's architecture needs to be sequential. Self-normalisation will not necessarily happen in nonsequential architectures, so is not guaranteed that SELU will outperform other activation functions [46].

Next is a summary of the initialisations and activation functions' definitions considered.

- Initialisation: Glorot;
  - Activation function:
    - $\sigma(z) = \dfrac{1}{1 + e^{-z}}$ (logistic) ;
    - $tanh(z) = 2\sigma(2z) - 1$. (2.13)
- Initialisation: He;
  - Activation function:
    - $ReLU(z) = max(0, z)$;
    - $LeakyReLU_{\alpha}(z) = max(\alpha z, z)$;
    - $ELU_{\alpha}(z) = \begin{cases} \alpha(e^{z} - 1) & , \text{ for } z \leq 0 \\ z & , \text{ otherwise .} \end{cases}$
- Initialisation: LeCun.
  - Activation function:
    - $SELU_{\alpha,\lambda}(z) = \lambda \begin{cases} \alpha(e^{z} - 1) & , \text{ for } z \leq 0 \\ z & , \text{ otherwise .} \end{cases}$ (2.14)

2.2.4.3 Training Dynamics

1. A set of features is selected to go into the network's input layer, called training dataset.

   (a) *Forward pass*: Each subset of sequential training data is passed to the network's input layer, which sends it to the first hidden layer. The algorithm then calculates the output of all the neurons in that layer (for every instance in the subset). The result is passed on to the next layer, its outputs computed and passed to the next layer, and so on until the output reaches the last layer.

   (b) The full training set goes through the network once, accounting for an epoch;

2. Next, the algorithm measures the network's performance through the selected cost function.

   (a) *Reverse pass*: The algorithm goes through each layer in reverse to measure the error contribution from each connection. This is done analytically by applying the chain rule[12]. The algorithm works backward until it reaches the upper most layer.

   (b) Ultimately, the algorithm performs a gradient descent (or equivalent) step to adjust all the connections weights in the network, using the error gradients it just computed, fitting the model to the training data.

3. The process is repeated until reaching a stopping criteria [46].

2.2.4.4 Validating and Testing

In order to infer the generalisability within a group of candidate models it is required to test them in on new and unseen data. The best way of attesting the model's generalisability power as well as finding out the best set of hyperparameters for a given model/dataset is by splitting the data into three sets: the training set, the validation set, and the test set.

Primarily, several candidate models are trained with various sets of hyperparameters by using the training set and validated throughout the process, by using the validation set. The models and respective hyperparameters that performed best on the validation set are then selected to proceed. Next, the training and validation sets are combined, while maintaining the test set. The best models are retrained on the new training set, resulting in a set of final trained models for that division of data. The error rate on new cases is called the generalisation error (or out-of-sample error), and by evaluating the set of final models on the test set, we get estimates of the error. These values tell us how well the model will perform on unseen instances. If the training error (in-sample error) is low, *i.e.*, the model makes few mistakes on the training set, but the out-of-sample error is high, it is an indication that the model is overfitting the training data.

[12] Suppose that two functions $f(x)$ and $g(x)$ are differentiable. If $y = f(u)$ and $u = g(x)$, then the derivative of y with respect to x is $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} \cdot \frac{\partial u}{\partial x}$.

Lastly, the splitting is remade with different training, validation, and test datasets, and the process repeated. The training is restarted, and in the end we get another estimate of the generalisation error after having concluding more evaluations on a different test set. Note that the problem with evaluating multiple models on a stationary dataset would be that we measured the generalisation error multiple times on the same data and adapt the model and hyperparameters to produce the best model for that particular set. Ultimately, the model is unlikely to perform as well on new data. Also, by averaging out all the evaluations of a model, we get a much more accurate measure of its performance. There is a drawback, however: the training time is multiplied by the number of validation sets or divisions.

It is worth mentioning that if the validation set is too small, model evaluations will be imprecise. We may end up mistakenly selecting a suboptimal model. On the other hand, if the validation set is too big, then the remaining training set will be much smaller than the full training set. Since the final model will be trained on the full training set, it is not recommendable to compare candidate models trained on a much smaller training set [46].

2.2.4.5 Automating Hyperparameter Tuning

The success of the model is closely dependent on the choice of good hyperparameters. The better they are, the smaller the network's error. One option would be to play around with the hyperparameters manually until finding a combination that provides good results. Besides being tedious, there would not be enough time to explore the search space thoroughly. This problem presents itself as an optimisation problem: the ultimate goal is finding the minimum of a 'black-box' function $f(\mathbf{x})$ on some bounded set χ, corresponding to diminishing the global network's error. As we do not have an analytical expression representative of the network, the analysis/evaluation is restricted to sampling. If the evaluation is cheap to perform we could sample at many combinations of hyperparameters randomly and extensively. However, if it turns out to be a computationally expensive task it is of major importance trying to minimise the number of samples.

Bayesian Optimisation

In this work, the problem is considered through the framework of Bayesian optimisation, in which a learning algorithm's generalisation performance is modelled as a sample from a Gaussian Process (GP). This choice has been shown to outperform other state-of-the-art global optimisation algorithms on a number of challenging optimisation benchmark functions [55]. What makes Bayesian optimisation different from other procedures is that it constructs a probabilistic model for $f(\mathbf{x})$ and then exploits this model to make decisions about where in χ to next evaluate the function, while integrating out uncertainty. The essential philosophy is to use all of the information available from previous evaluations of $f(\mathbf{x})$. This results in a procedure that can find the minimum of difficult non-convex functions with relatively few evaluations, at the cost of performing more computation to determine the next point to try. When evaluations of $f(\mathbf{x})$ are expensive to perform — as is the case when it requires training a ML algorithm — then it is easy to justify some extra computation to make better decisions [56].

There are two major choices that must be made when performing Bayesian optimisation. First, we must select a prior over functions that will express assumptions about the function being optimised. Second, we must choose an acquisition function, which is used to construct a utility function from the model posterior, allowing us to determine the next point to evaluate. Presenting the formalism of this matter would go beyond the scope of this work, but details can be found in Brochu et al. [57]. However, to visually illustrate the process, a Graphics Interchange Format (GIF)[13] was made[14], in it is possible to contemplate how well in a few iterations the optimisation algorithm started to converge to the minimum of the unknown function of equation 2.15,

$$f(x) = -\left(x + \frac{0.05^2}{0.05^2 + (x - 0.1)^2}\right) . \tag{2.15}$$

The added difficulty comes from the fact that the minimum from the peak at $x = 0.1$ is narrow and quite close to the local minimum at $x = 1$.

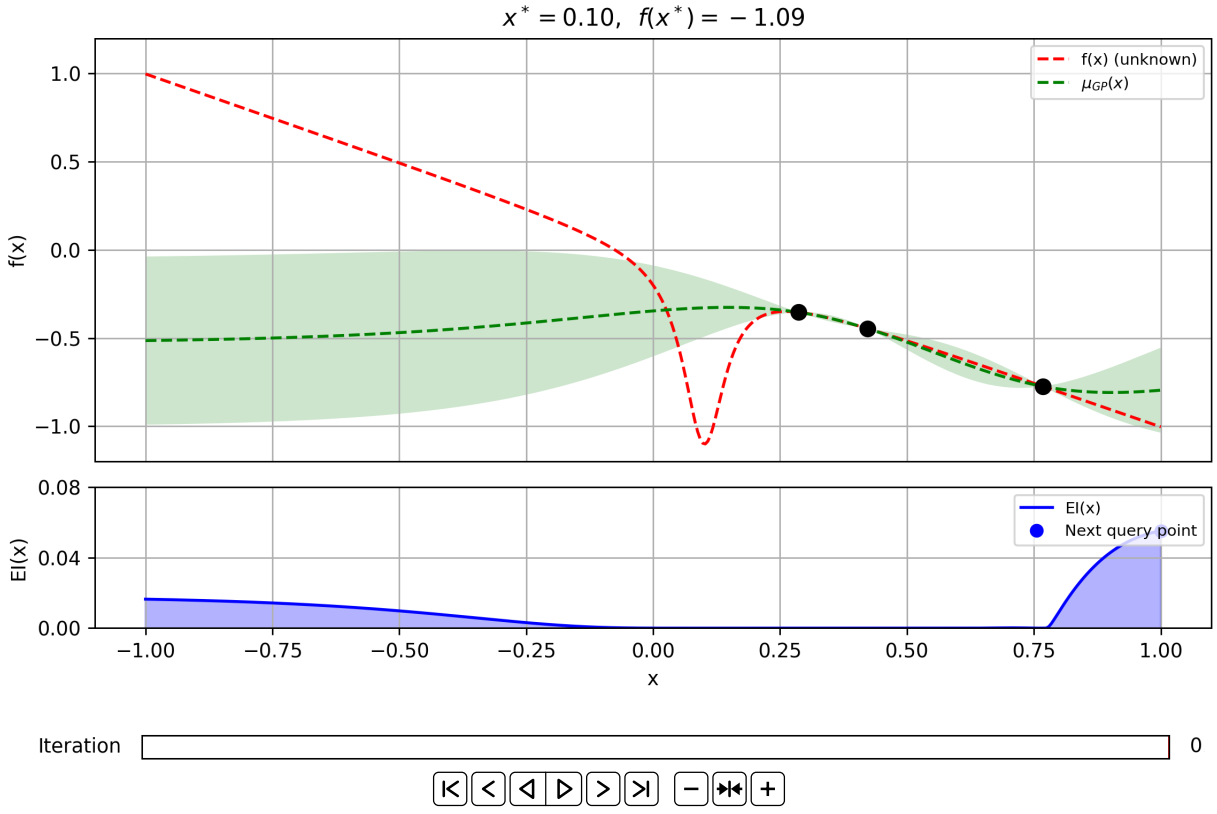


Figure 2.9: Bayesian optimisation of equation 2.15 using a Gaussian Process (GP).

In the top graphic there is the approximation by the GP model $\mu_{GP}(x)$ to an unknown function $f(x)$ and how sure the GP model is about the function, measured in terms of the covariance and represented in green. The lower graphic contains the acquisition values that determine the next point to be queried.

[13] We kindly request to open this PDF with *Adobe Acrobat Reader* (version $\geq$ 6) to enable the visualisation of the GIF. The commands present are, in order: first frame; previous frame; play (forwards or backwards); pause; next frame; last frame; reduce speed; default speed; increase speed.

[14] Along with my great friend Tiago França, in our spare time during the COVID-19 pandemic.

The total number of evaluations was set to 25 with 3 random initial evaluations. Given the initial evaluations, a probabilistic model is starting to be built for the objective function $f(x)$. The acquisition function chosen to be optimised was the Expected Improvement (EI) (see Brochu et al. [57]) and its evolution is also observable in the lower graphic. The exploration-exploitation trade-off is done with the default parameters.

2.2.5 Deep Learning for Time Series

2.2.5.1 Time Series Analysis

Time series are collections of data points arranged in chronological order, representing temporal or spatial evolution of the dynamics of a given variable. Their analysis is intended to extract meaningful summary and statistical information, hence enabling the diagnosis of past behaviour as well as the forecasting of future behaviour [36].

Time series data and their analysis are increasingly important due to the massive production of such data through the digitalisation of manuscripts and constant monitoring and storage of measurable variables. As continuous data collection become prevailing, the need for competent time series analysis will increase [36].

Something we must be vigilant and thoughtful about is the lookahead. The term lookahead is used in time series analysis to indicate any knowledge of the future. A lookahead is a way, through data, of learning something about the future earlier than we intended to, thus we should not have such knowledge when designing, training, or evaluating a model. To put it differently, lookahead is any way that information about the future might leak back in time in our modelling, hence affecting how the model behaves earlier in time. For example, when choosing hyperparameters for a model, we might test the model at different points in time in the data set, then choose the best model and start at the beginning of our data to test this model. This is troublesome because we chose the model for one time having information about what would happen at a subsequent time [36].

2.2.5.2 Processing Sequences Using RNNs

Despite its predicting features, the former MLP model manifests incompleteness drawn from the introduction of the dimension of time because it is no longer possible to have fixed-size inputs and produce fixed-size outputs, the causality of time is not encoded into the architecture of the model[15]. RNNs are useful because they allow variable-length sequences as both inputs and outputs, encoding the causality of time. To circumnavigate the question arisen from the need for input of variable size data there is a simple and effective approach known as the sliding window. Basically, a data interval of a fixed size m is taken and fed into the RNN. From these m elements, one outputs the y_{m+1} element, which will

[15]We could think about it in the following way: if the inputs' positions were to be scrambled it would not make any difference in the accuracy of the algorithm or, alternatively, the scrambling of the past would have no effect on the prediction

be identified as the target[16]. By continuing sliding the window across, the entire dataset will eventually be covered and input into the network. Backpropagation similarly occurs as in the perceptron, but in this case through time.

2.2.5.3 Recurrent Neurons and Layers

Essentially, a RNN cell does the exact same calculation as a perceptron, *i.e.*, a weighted sum with an activation function and it outputs a single number. In other words, it takes in a set of data over time and outputs a single number or a list of numbers over time. The major difference is that a RNN keeps a state that is passed on sequentially. In the following RNN (see figure 2.10), the described scheme is illustrated over all iterations.

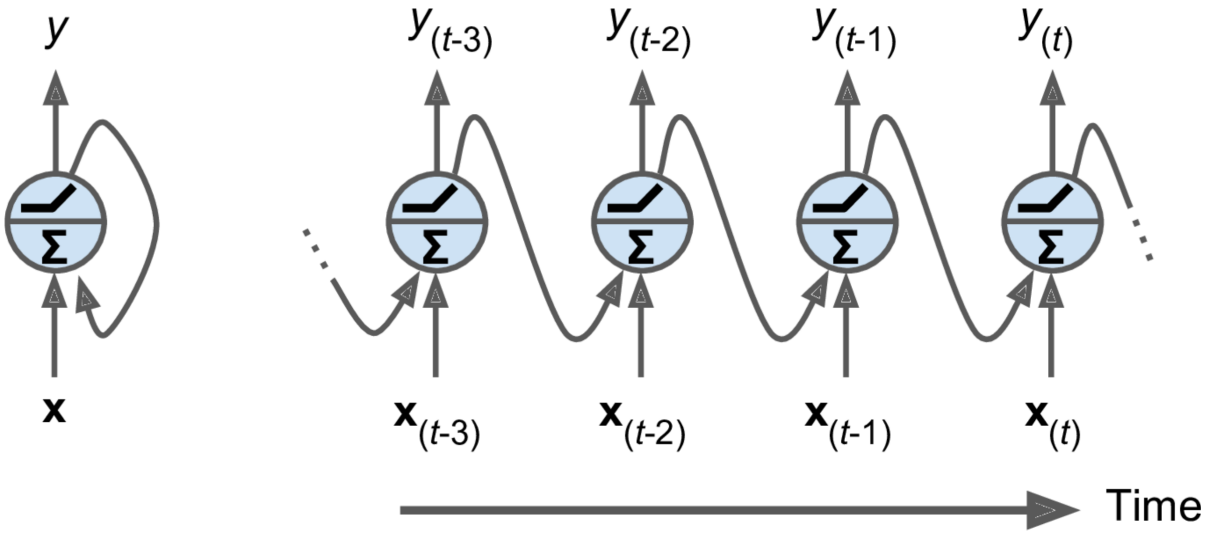


Figure 2.10: Recurrent neuron (left) unrolled through time (right) [46].

The notation is as follows: input vector - $\mathbf{x}_t$, output scalar - y_t, "hidden" state - y_{t-1}.[a]

[a] Permission to use this image was granted from the publisher.

At each time-step t, the recurrent neuron receives the inputs $\mathbf{x}_t$ as well as its own output from the previous time-step, y_{t-1}. Due to the non-existence of previous output at the first time-step, it is normally set to 0. This is called unrolling the network through time, given that it is the same recurrent neuron represented once per time-step [46].

It is now possible to imagine the concept of a layer of recurrent neurons. At each time-step t, all neurons, individually, receive both the input vector $\mathbf{x}_t$ and the output vector from the previous time-step $\mathbf{y}_{t-1}$, as displayed in figure 2.11. Note that both inputs and outputs are in the vectorial form (bold), whereas previously the output was a scalar [46].

[16] *Target* is defined as the output variable of the input variables whereas the *labelled target* is true outcome of the target variable.

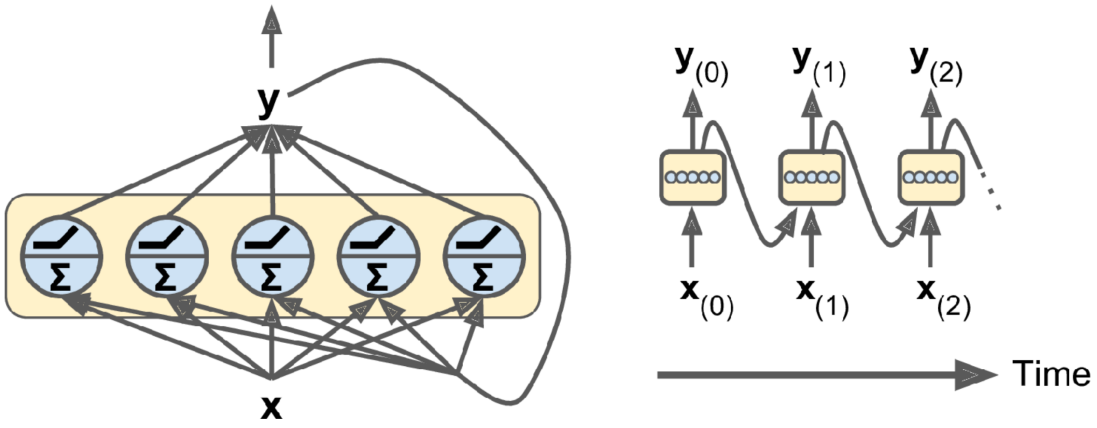


Figure 2.11: Layer of recurrent neurons (left) unrolled through time (right) [46].

The notation is as follows: input vector - $\mathbf{x}_t$, output vector - $\mathbf{y}_t$, "hidden" state - $\mathbf{y}_{t-1}$.[a]

[a] Permission to use this image was granted from the publisher.

Each recurrent neuron has two sets of weights: one for inputs $\mathbf{x}_t$ and the other for the outputs of the previous time-step (or "hidden" state), $\mathbf{y}_{t-1}$. Now we define these weight vectors $\mathbf{w}_x$ and $\mathbf{w}_y$. Considering the whole recurrent layer instead of just one recurrent neuron, we can place all the weight vectors in two weight matrices, $\mathbf{W}_x$ and $\mathbf{W}_y$. If we consider $\phi(\cdot)$ and $\mathbf{b}$ as the activation function and the bias vector, respectively, the output vector of the whole recurrent layer for a single instance is

$$\mathbf{y}_t = \phi(\mathbf{W}_x^\top \cdot \mathbf{x}_t + \mathbf{W}_y^\top \cdot \mathbf{y}_{t-1} + \mathbf{b}) \text{ [46].} \tag{2.16}$$

It is worth noticing that it is feasible to compute a recurrent layer's output in one iteration for a whole mini-batch[19] by placing all the inputs at time-step t in an input matrix $\mathbf{X}_t$, hence,

$$\mathbf{Y}_t = \phi(\mathbf{X}_t \cdot \mathbf{W}_x + \mathbf{Y}_{t-1} \cdot \mathbf{W}_y + \mathbf{b}) = \phi([\mathbf{X}_t \quad \mathbf{Y}_{t-1}] \cdot \mathbf{W} + \mathbf{b}), \text{ with } \mathbf{W} = \begin{bmatrix} \mathbf{W}_x \\ \mathbf{W}_y \end{bmatrix} \text{ [46].} \tag{2.17}$$

As the data during the implementation phase will take a tensorial form to embed a mini-batch approach, it is worth clarify that in this equation we have:

- $\mathbf{Y}_t$ as a matrix with dimensions $m \times n_{neurons}$ consisting of the layer's outputs at time-step t for each instance in the mini-batch (being m the number of instances in the mini-batch and $n_{neurons}$ the number of neurons);
- $\mathbf{X}_t$ as a matrix with dimensions $m \times n_{inputs}$ consisting of the inputs for all instances (n_{inputs} as the number of input features);
- $\mathbf{W}_x$ as a matrix with dimensions $n_{inputs} \times n_{neurons}$ consisting of the connection weights for the inputs of the current time-step;
- $\mathbf{W}_y$ as a matrix with dimensions $n_{neurons} \times n_{neurons}$ consisting of the connection weights for the outputs of the previous time-step;

[19] Mini-batch here is defined as a subset of sequential data (multiple instances) for some interval of time.

- $\mathbf{b}$ as a vector of size $n_{neurons}$ consisting of each neuron's bias term;
- The weight matrices $\mathbf{W}_x$ and $\mathbf{W}_y$ that have been concatenated vertically into a single weight matrix $\mathbf{W}$ with dimensions $(n_{inputs} + n_{neurons}) \times n_{neurons}$;
- The notation $[\mathbf{X}_t \quad \mathbf{Y}_{t-1}]$ representing the horizontal concatenation of the matrices $\mathbf{X}_t$ and $\mathbf{Y}_{t-1}$.

Bear in mind that $\mathbf{Y}_t$ is a function of $\mathbf{X}_t$ and $\mathbf{Y}_{t-1}$, $\mathbf{Y}_{t-1}$ is a function of $\mathbf{X}_{t-1}$ and $\mathbf{Y}_{t-2}$, and so on. Therefore, $\mathbf{Y}_t$ is a function of all the inputs since $t = 0$ (that is, $\mathbf{X}_0, \mathbf{X}_1, \ldots, \mathbf{X}_t$) [46].

Stacking multiple recurrent layers of cells results in a deep RNN (figure 2.12). Under that condition, a RNN can simultaneously take a sequence of inputs and produce a sequence of outputs. This type of sequence-to-sequence network is useful for predicting/forecasting time series: we feed it the data over the last n time-steps, and it outputs the data shifted by an arbitrary amount of days into the future. The advantage of this technique over sequence-to-vector or vector-to-sequence is that the loss will contain a term for the output of the RNN at every time-step, not just the output at the last time-step. Consequently, there will be many more gradients flowing through the model, and they will not have to flow only through time, they will also flow from the output of each time-step, resulting in a more stable and faster training [46].

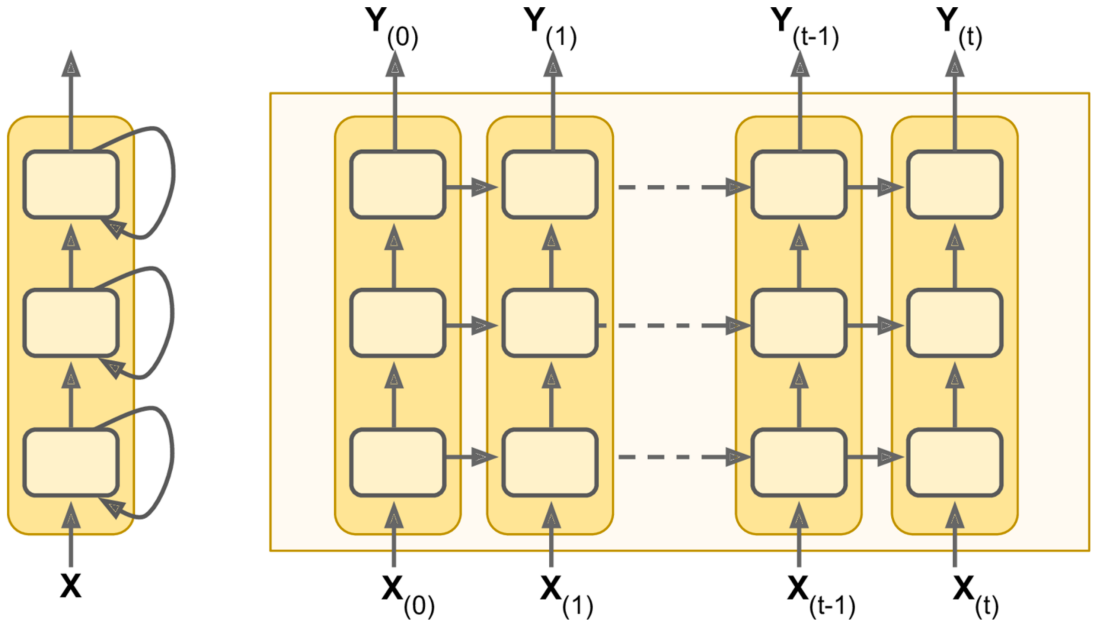


Figure 2.12: Deep RNN (left) unrolled through time (right) [46].

The notation is as follows: input matrix - $\mathbf{X}_t$, output matrix - $\mathbf{Y}_t$, "hidden" state - $\mathbf{Y}_{t-1}$.[a]

[a] Permission to use this image was granted from the publisher.

2.2.5.4 Training RNNs

In order to train a RNN, we have to unroll it through time and then use regular backpropagation. This strategy is named Backpropagation Through Time (BPTT) (see figure 2.13) [46].

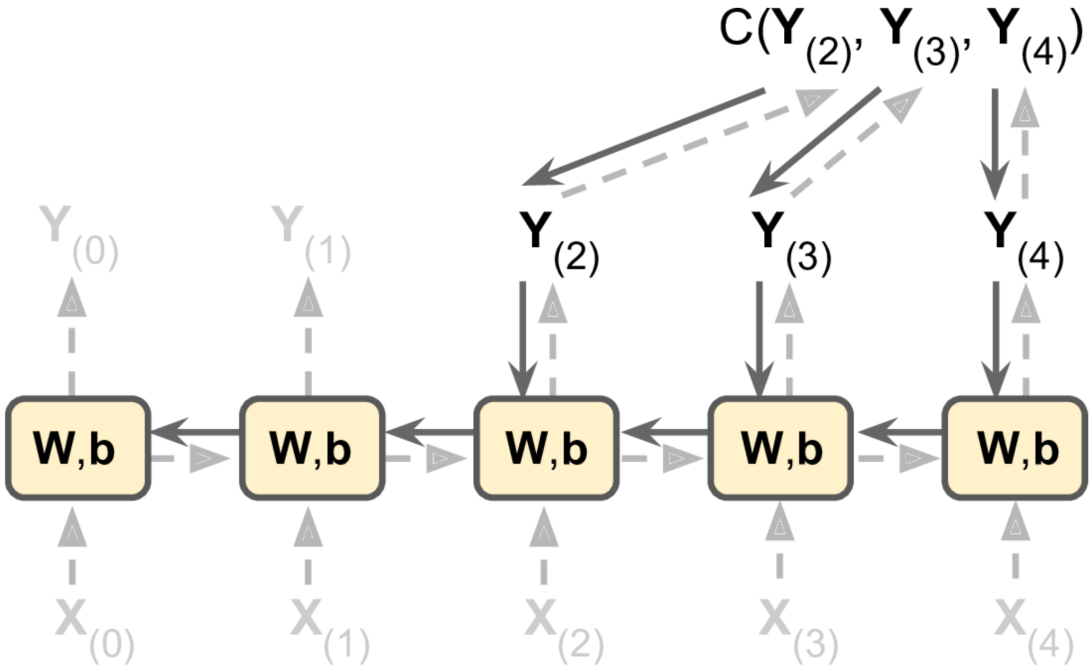


Figure 2.13: Backpropagation through time [46].

The notation is as follows: input matrix - $\mathbf{X}_t$, output matrix - $\mathbf{Y}_t$, weights matrix - $\mathbf{W}$, bias vector - $\mathbf{b}$, cost function - $C(\cdot)$.[a]

[a] Permission to use this image was granted from the publisher.

As with regular backpropagation, there is a first forward pass through the unrolled network, represented by the dashed arrows in figure figure 2.13. In the next step, the output sequence is evaluated using a cost function $C(\mathbf{Y}_0, \mathbf{Y}_1, ..., \mathbf{Y}_T)$, where T is the maximum time-step (such as the ones presented in section 2.2.4.1, item 2). Note that this cost function may ignore some outputs, as shown in figure 2.13 (for example, in a sequence-to-vector RNN, except for the last one, all outputs are ignored), contrary to what happens on a sequence-to-sequence RNN. The gradients of that cost function are then propagated backward through the unrolled network, here represented by the solid arrows. Lastly, the model parameters are updated using the gradients computed during BPTT. Since the same parameters $\mathbf{W}$ and $\mathbf{b}$ are used at each time-step, backpropagation will sum over all time-steps. Note that in our case, sequence-to-sequence, the gradients will flow backward through all the outputs used by the cost function, not just partially as displayed in the example figure 2.13, where the cost function is computed using the last three outputs of the network, promoting gradients flow through these three outputs, but not through $\mathbf{Y}_0$ and $\mathbf{Y}_1$ [46].

2.2.5.5 Tackling the Short-Term Memory Problem

As the output of a recurrent neuron at time-step t is a function of all the inputs from previous time-steps, it is said to possess a form of memory. By definition, a part of a neural network that preserves a given state across time-steps is called a memory cell. A single recurrent neuron, or a layer of recurrent neurons, is a basic cell, or layer, capable of learning only short patterns. Because of the transformation that the data goes through when passing over a vanilla RNN (see figure 2.10), some information is lost

at each time-step. After some steps, the RNN's state contains practically no trace of the first inputs [46].

Generally, a cell's state at time-step t, denoted $\mathbf{h}_t$, is a function of some inputs at that time-step and its state at the previous time-step: $\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t)$. Its output at time-step t, $\mathbf{y}_t$, is also a function of the previous state and the current inputs. In the case of the basic cells we have discussed until now, the output has explicitly been assumed to be equal to the state (hence the "" around *hidden* in the description of previous figures), but in more complex cells this is not always the case, as shown in figure 2.14. There are different types of cells specially designed to learn longer patterns and carry a different source of information memory besides the previous output [46].

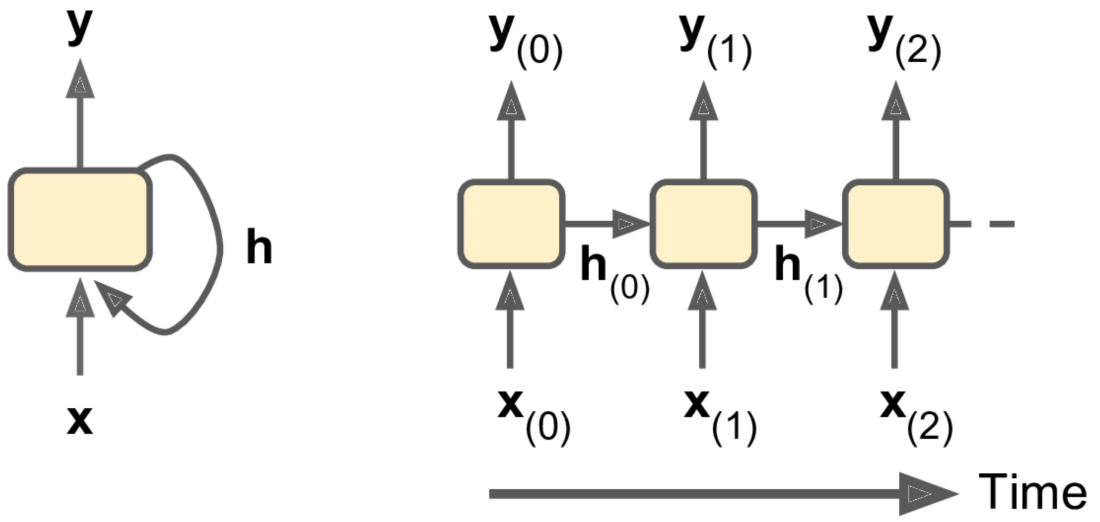


Figure 2.14: A cell's hidden state and its output may be different [46].

The notation is as follows: input vector - $\mathbf{x}_t$, output vector - $\mathbf{y}_t$, hidden state - $\mathbf{h}_t$.[a]

[a] Permission to use this image was granted from the publisher.

What happens inside each type of those RNNs is different depending on the type used, but the underpinning concept is that they are mathematically designed to accumulate information (such as evidence for a particular feature or category) over a long duration, and once that information has been used, it might be useful for the neural network to dynamically forget the old state at each time-step, another programmed possibility [45]. This is what gated RNNs do. These include the Long Short-Term Memory (LSTM) and Gate Recurrent Unit (GRU) [58], which tend to be the most common [59]. Further, the vanilla RNN already presented can too be considered a cell that sustains some memory, as already discussed.

2.2.5.6 LSTM Cells

In a LSTM cell, the state is split into two vectors: $\mathbf{h}_t$ and $\mathbf{c}_t$, where 'c' stands for 'cell'. The former vector represents the short-term state and the latter represents the long-term state (see figure 2.15).

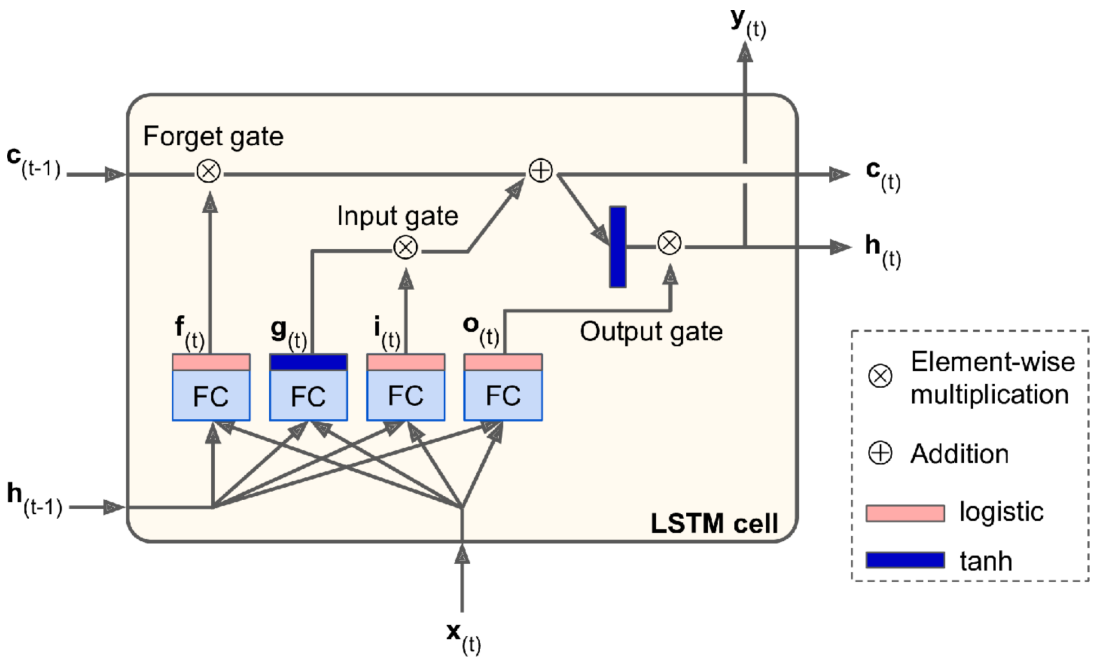


Figure 2.15: LSTM cell [46].

The notation is as follows: FC - fully connected layer, input vector - $\mathbf{x}_t$, output vector - $\mathbf{y}_t$, long-term state vector - $\mathbf{c}_t$, short-term state vector - $\mathbf{h}_t$, forget vector controller - $\mathbf{f}_t$, gate vector controller - $\mathbf{g}_t$, input vector controller - $\mathbf{i}_t$, output vector controller - $\mathbf{o}_t$.[a]

[a] Permission to use this image was granted from the publisher.

The long-term state $\mathbf{c}_{t-1}$ traverses the network from left to right, going through a "forget" gate, where it excludes some information ("memories"), and it adds some new information selected by an input gate via the addition operation. The result, $\mathbf{c}_t$, does not suffer any further modification until the next time-step. Therefore, at each time-step some "memories" are eliminated and some are added. Also, after each addition operation, the long-term state is duplicated and passed through the $tanh$ function, being the result further filtered by the output gate. This path produces the short-term state $\mathbf{h}_t$ which is equal to the cell's output for that time-step, $\mathbf{y}_t$. Focusing on the memories formation and gate functioning, we notice that first, the input vector $\mathbf{x}_t$, and the previous short-term state $\mathbf{h}_{t-1}$ are fed to four different fully connected layers [46]. They all serve a different purpose as:

- The main layer is responsible for outputting $\mathbf{g}_t$ and analysing the inputs $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$. In the basic vanilla RNN cell seen previously, its outputs would just go straight out to $\mathbf{y}_t$ and $\mathbf{h}_t$. Contrastingly, in a LSTM cell this layer's output have its most important parts stored in the long-term state, and the rest is dropped;

- The three other gates are gate controllers, and because they use the logistic activation function (see equation 2.7), their outputs range from 0 to 1. Those outputs are fed to element-wise multiplication operations[24], so if they output 0s they close the gate, whereas if they output 1s they open it. More explicitly:

[24] The notation used here to represent that operation is $\odot$.

- The forget gate, which is controlled by $\mathbf{f}_t$, controls which parts of the long-term state vector $\mathbf{c}_t$ should be eliminated;
- The input gate, which is controlled by $\mathbf{i}_t$, controls which parts of $\mathbf{g}_t$ should be added to the long-term state vector $\mathbf{c}_t$.
- The output gate, which is controlled by $\mathbf{o}_t$, controls which parts of the long-term state vector should be read and output at that time-step, both to $\mathbf{h}_t$ and to $\mathbf{y}_t$ [46].

In short, a LSTM cell can learn to recognise an important input, by means of the input gate mechanism, store it in the long-term state, preserve it for as long as it is needed, a task regulated by the forget gate, and extract it whenever it is needed. The following set of equations summarises the formulas behind the calculations of the cell's long-term state, its short-term state, and its output at each time-step for a single instance:

$$\begin{cases} \mathbf{i}_t = \sigma(\mathbf{W}_{xi}^\top \cdot \mathbf{x}_t + \mathbf{W}_{hi}^\top \cdot \mathbf{h}_{t-1} + \mathbf{b}_i) \\ \mathbf{f}_t = \sigma(\mathbf{W}_{xf}^\top \cdot \mathbf{x}_t + \mathbf{W}_{hf}^\top \cdot \mathbf{h}_{t-1} + \mathbf{b}_f) \\ \mathbf{o}_t = \sigma(\mathbf{W}_{xo}^\top \cdot \mathbf{x}_t + \mathbf{W}_{ho}^\top \cdot \mathbf{h}_{t-1} + \mathbf{b}_o) \\ \mathbf{g}_t = tanh(\mathbf{W}_{xg}^\top \cdot \mathbf{x}_t + \mathbf{W}_{hg}^\top \cdot \mathbf{h}_{t-1} + \mathbf{b}_g) \\ \mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t \\ \mathbf{y}_t = \mathbf{h}_t = \mathbf{o}_t \odot tanh(\mathbf{c}_t) \end{cases} . \tag{2.18}$$

For this set of equations:

- $\mathbf{W}_{xi}$, $\mathbf{W}_{xf}$, $\mathbf{W}_{xo}$, $\mathbf{W}_{xg}$ represent the weight matrices of each of the four layers for their connection to the input vector $\mathbf{x}_t$;
- $\mathbf{W}_{hi}$, $\mathbf{W}_{hf}$, $\mathbf{W}_{ho}$, $\mathbf{W}_{hg}$ represent the weight matrices of each of the four layers for their connection to the previous short-term state $\mathbf{h}_{t-1}$;
- $\mathbf{b}_i$, $\mathbf{b}_f$, $\mathbf{b}_o$, $\mathbf{b}_g$ represent the bias terms for each of the four layers [46].

2.2.5.7 GRU Cells

Presently, a GRU is one of the most widely used RNN cells. It is a simplified version of the LSTM cell, which operates in a similar way [36]. Both GRU and LSTM cells differ in the following:

- A GRU has two gates, whereas a LSTM has three. These gates are used to determine how much new information is passed through, and how much old information is preserved. As an LSTM has more gates, it has more parameters;
- A LSTM tends to have better performance, nonetheless there is a trade-off in terms of training speed due to the different number of parameters, resulting in the GRU being faster to train. Nevertheless, there are published results where a GRU outperforms a LSTM [60–62]. Particularly, a GRU likely to outperform a LSTM for non-language tasks [36].

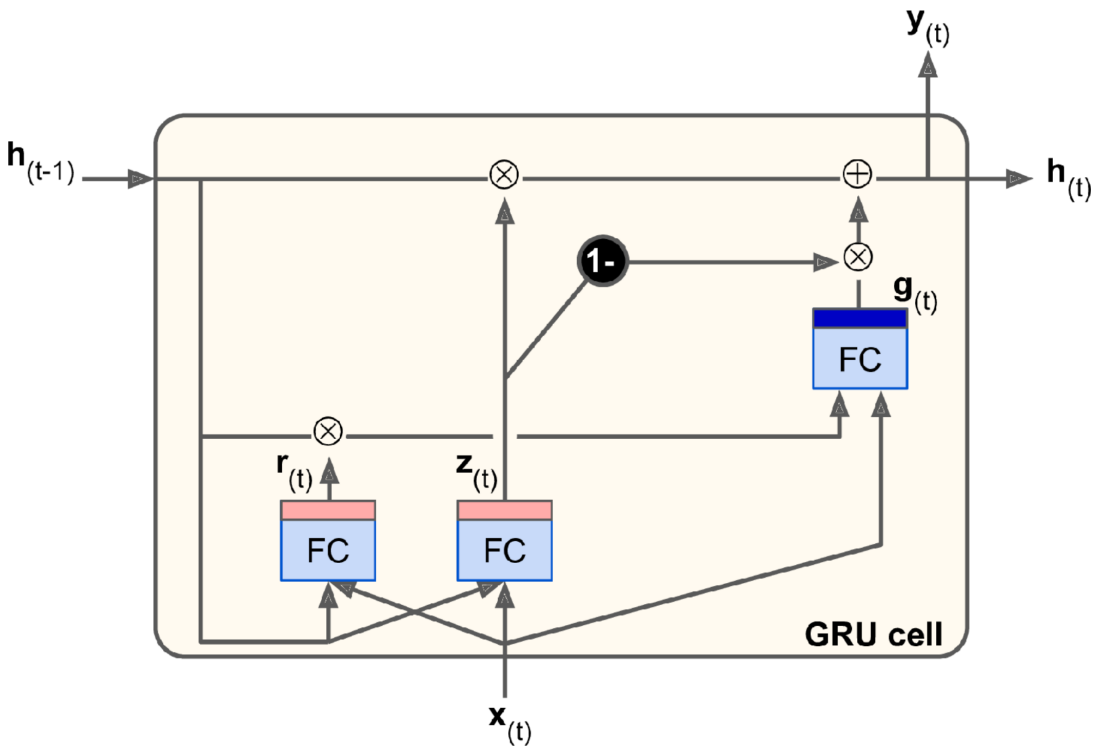


Figure 2.16: GRU cell [46].

The notation is as follows: FC - fully connected layer, input vector - $\mathbf{x}_t$, output vector - $\mathbf{y}_t$, state vector - $\mathbf{h}_t$, forget and input vector controller - $\mathbf{z}_t$, state vector controller - $\mathbf{r}_t$.[a]

[a] Permission to use this image was granted from the publisher.

The main simplifications can be summed up to:

- Both state vectors are combined into a single state vector $\mathbf{h}_t$;
- A single gate controller, $\mathbf{z}_t$, controls both the forget and input gates. If the controller outputs a 1, the forget gate is open and the input gate is closed, conversely if it outputs a 0 the opposite happens (note the addition of a negative unit constant in the input gate path). This simply means that when a memory must be stored, the location where it will be stored is erased first;
- There is no output gate, meaning that the full state vector is the output at every time-step. Nonetheless, there is an additional gate controller, $\mathbf{r}_t$, that controls which portion of the previous state will be allowed into the main layer, $\mathbf{g}_t$ [46].

Equations set 2.19 encapsulates how to compute the cell's state at each time-step for a single instance,

$$\begin{cases} \mathbf{z}_t = \sigma(\mathbf{W}_{xz}^\top \cdot \mathbf{x}_t + \mathbf{W}_{hz}^\top \cdot \mathbf{h}_{t-1} + \mathbf{b}_z) \\ \mathbf{r}_t = \sigma(\mathbf{W}_{xr}^\top \cdot \mathbf{x}_t + \mathbf{W}_{hr}^\top \cdot \mathbf{h}_{t-1} + \mathbf{b}_r) \\ \mathbf{g}_t = tanh(\mathbf{W}_{xg}^\top \cdot \mathbf{x}_t + \mathbf{W}_{hg}^\top \cdot (\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_g) \\ \mathbf{y}_t = \mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \mathbf{g}_t \end{cases} \quad [46]. \tag{2.19}$$

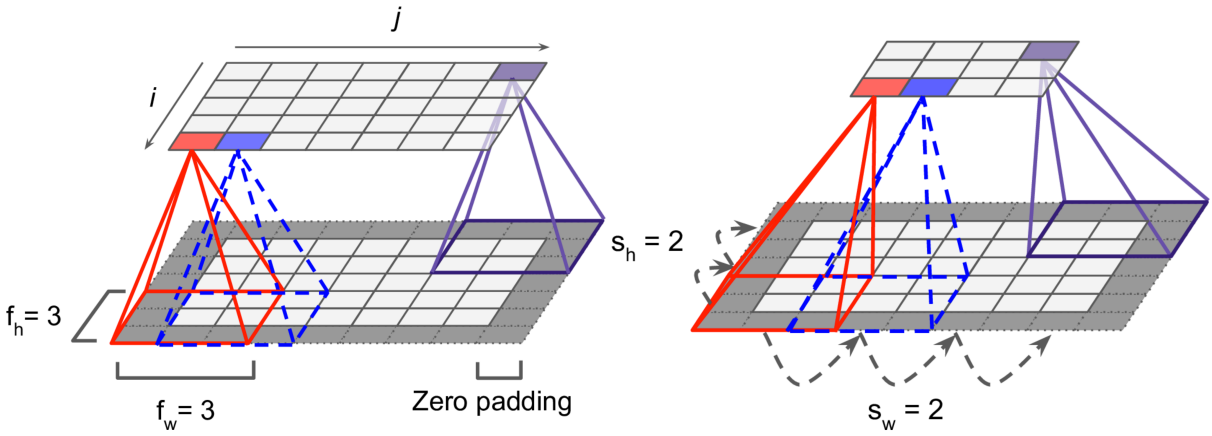
Note that both GRUs and LSTMs helped solving the exploding and vanishing gradients problem,

explained in item 5 from section 2.2.4.2, that was first encountered when RNNs were used. This problem was addressed with GRU and LSTM as a consequence of their tendency to keep inputs and outputs from the cell in tractable value ranges. This is due both to the form of the activation function they use and to the way that the update gate can learn to allow information in or not, leading to higher probability of having reasonable gradient values than in a vanilla RNN cell, which has no encoded notion of a gate [36].

2.2.5.8 Convolutional Neural Networks

Most modern time series analysis problems are undertaken with recurrent network structures, or, less commonly, with convolutional network structures. Convolutions are a way to capture information about the ordering of the entries on a matrix and it is described by applying a kernel (matrix) to a larger input matrix by sliding it across, forming a new, convoluted, matrix [36].

Each neuron located in row i, column j of the convolutional layer (upper) in figure 2.17a is the sum of element-wise multiplication (Hadamard product) of the kernel and a subsection (given by the size of the receptive field) of the larger matrix [36], *i.e.*, that neuron is related to the neurons in the previous matrix located in rows i to $i + f_h - 1$ and columns j to $j + f_w - 1$, where f_h and f_w are the height and width of the receptive field. This kernel is applied repeatedly as it slides across a matrix and it incorporates information about the entries' neighbours values into its own value. This is accomplished by pre-specifying a number of sets of kernels, so that different features can emerge. In order for a layer to have the same dimensions as the previous layer, it is common to add zeros around the inputs, this method is called zero padding [46].



(a) Connections between layers and zero padding [46]. (b) Reducing dimensionality using a stride of 2 [46].

Figure 2.17: CNN layers with rectangular local receptive fields.

The notation is as follows: height of the receptive field - f_h, width of the receptive field - f_w, height of the stride - s_h, width of the stride - s_w.[a]

[a] Permission to use this image was granted from the publisher.

It is also feasible to connect a large input layer to a much smaller layer by spacing out the receptive fields, as in figure 2.17b. This contributes to the reduction of the model's computational complexity. This

shift from one receptive field to the next is defined as the stride [46].

Neuron's set of weights are called filters (or convolutional kernels), and a layer full of neurons using the same filter outputs a feature map, which highlights areas in a matrix that activates the filter the most. A convolutional layer can have an arbitrary number of filters, hence outputting a feature map per filter. This means that a convolutional layer simultaneously applies multiple trainable filters to its inputs, making it capable of detecting multiple features anywhere in its inputs. During training, the convolutional layer will automatically learn the most useful filters for its task, and the layers above will learn to combine them into more complex patterns. One advantage of this type of network is that they have few parameters since the same convolutional kernels are repeated over and over, meaning that there are not too many weights to train [46].

Mathematically speaking, the output of a given neuron in a convolutional layer is given by

$$z_{i,j,k} = b_k + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k'=0}^{f'_n-1} x_{i',j',k'} \cdot w_{u,v,k',k}, \text{ with } \begin{cases} i' = i \times s_h + u \\ j' = j \times s_w + v \end{cases} . \tag{2.20}$$

For this equation:

- $z_{i,j,k}$ represents the output of the neuron located in row i, column j in feature map k of the convolutional layer l;
- s_h and s_w represent both vertical and horizontal strides
- f_h and f_w represent both height and width of the receptive field; $f_{n'}$ is the number of feature maps in the previous layer $l-1$;
- $x_{i',j',k'}$ is the output of the neuron located in layer $l-1$, row i', column j', and feature map k';
- b_k is the bias term for feature map k in layer l;
- $w_{u,v,k',k}$ is the connection weight between any neuron in feature map k of the layer l and its inputs located in row u, column v, and feature map k' [46].

Causal Convolutions

Traditional convolution is a poor match to time series because one of the main features consists in treating all spaces equally. This makes sense for images, the main area of application of Convolutional Neural Network (CNN), but it does not fit the philosophy of time series, where some points in time are necessarily closer than others. Convolutional networks are also structured to be scale invariant, however, in time series we likely want to preserve scale and scaled features [36].

Nonetheless, there is an architectural transformation that includes modifications to be time aware. In a dilated causal convolution, as in the example shown in figure 2.18, 1D convolutional layers are stacked and the dilation rate[27] is doubled at every layer, *i.e.*, how spread apart each neuron's inputs are.

[27] This is equivalent to using a regular convolutional layer with a filter dilated by inserting rows and columns of zeros. For example, a 1×3 filter equal to $[[1, 2, 3]]$ may be dilated with a dilation rate of 2, resulting in a dilated filter of $[[1, 0, 2, 0, 3]]$.

With this configuration, the first layer gets access to two time-steps at a time, while the next one sees four time-steps, the next one sees eight time-steps, and so on [46]. Therefore, the lower layers learn short-term patterns, while the higher layers learn long-term patterns, being also capable of processing arbitrarily long sequences [63]. This also promotes model sparsity and reduces redundant or overlapping convolutions, allowing the model to look further back in time, while keeping overall computations contained [36]. To conclude, this example of dilated causal convolution introduces the notion of temporal causality by permitting only data from prior time points.[28]

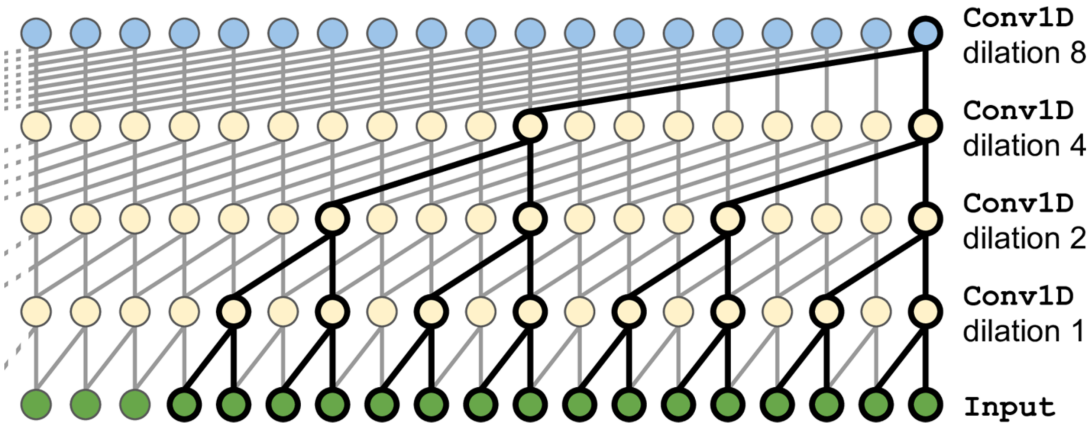


Figure 2.18: Modified convolutional neural network appropriate to time series [46].

2.2.5.9 Hybrid Networks

Continuing with the rationale of what was explained in section 2.2.5.8, we can have a 1D convolutional layer sliding several convolutional kernels across a sequence, producing a 1D feature map per convolutional kernel. Each convolutional kernel would learn to detect a single sequential pattern with a size no longer than the convolutional kernel's size. If we use k convolutional kernels, then the layer's output will be composed of k 1D sequences, all with the same length, or, equivalently, a single k-dimensional sequence. This means that we can build a neural network composed of a mix of recurrent layers, such as LSTM or GRU, and 1D convolutional layers. By using the zero padding method, the output sequence will have the same dimensions as the input sequence. Alternatively, it is possible to use a stride greater than 1, downsampling the input sequences. By shortening the sequences, the convolutional layer may help the recurrent layers detect longer patterns [46].

[28] Permission to use the following image was granted from the publisher.

3

Implementation

3.1 Framing the Problem

As seen in section 1.5, the first objective is to explain the relationship between the distribution of land use in terms of final exergy in the agricultural sector and economic growth explained in terms of GDP. Concretely, we fed a neural network with 6 features, *i.e.*, all the land use classes but one (5 were available in total) plus final exergy data for the agricultural sector, and GDP. Then, the neural network was set to output the 4 classes of land use and the 5^{th} was then determined, as they summed up to 1. By excluding one feature from the training, we were promoting smaller optimisation times. To understand how the energy and economic factors affect land use distribution, data regarding energy and economy was lagged backwards one time period (1 year) in relation to the input land use data. We assumed that the direction of influence is *past* (exergy + GDP) $\rightarrow$ *future* land use distribution with support on the argument that patterns of energy consumption and economic development in the past dictate how the farmer is going to utilise the land in the future. Money and energy are intuitively beforehand required to consummate significant modifications in land and infrastructures. Useful exergy usage was demonstrated to be positively correlated with GDP growth Serrenho et al. [41], therefore, as the GDP already explained the dynamics of useful exergy, final exergy was chosen to further help to model the dynamics in the sector.

Secondly, with aid of the framework already established in the land use modelling, neural networks were fed 8 features and retrained. Those features were split into land use (4), total final exergy usage and GDP for Portugal (2), and data regarding emissions from the agricultural and energy sectors (2). As the goal is to explain those emissions, the neural networks were set to output 2 values. Here, the lagging of the data was differently done. The data regarding land use and GHG emissions were lagged backwards one time period (1 year), in relation to the energy and economic variables. The intuition behind lagging the land use distribution data is that decisions regarding land use management are expected to affect land use emissions in the following years. If, for instance, agriculture area is massively converted into forest area, we will observe reduced emissions from the forest for years after the conversion, but not necessarily in the year of the conversion. The effect is anticipated to start happening starting only in the

following year. The inclusion of land use data in this model was due to the fact we are trying to forecast GHG from agriculture, and land use data contains information concerning that matter. In terms of data about total final exergy, its use is justified because we are trying to forecast GHG from energy sectors. All models involved can be seen as autoregressive, because their outputs depends on observations from previous time-steps.

3.2 Data

The main data and respective known sources available for this work were organised into land use and land cover statistics, energy, economy, and policies. The first subject referred to 1) permanent pasture area, permanent culture and arable land area, forest area and urban/artificial area, and 2) land use emissions. The energy data consisted of 3) final exergy [64]. Economy was represented by 4) GDP growth rate [65]. Lastly, policies data [66, 67] concerned the following: 5) wheat campaigns and policy reform, 6) private forestation policy, 7) agrarian reform and 8) agricultural transitory measures and policy reform. As the goal of the thesis was to include a historical perspective from the 1960's onwards, but the data available was insufficient to depict long trends, data reconstruction techniques were applied when necessary, as explained in each sub-section below, and supported with historical policies.

3.2.1 Land Use and Land Cover

Data from the Climate Change Initiative (CCI) by the European Space Agency (ESA) [68] was firstly considered, given its potential demonstrated in [69]. In the first place, the data, which consisted of annual and consistent time series of land change maps from 1992 to 2018 for the whole planet, was downloaded in Tag Image File Format (TIFF) format. The maps were overlapped with the contour of the Portuguese territory [1] and the global maps for every year using R and QGIS tools, extracting a clipped TIFF file containing only information about Portugal. For every TIFF file there was a latitude/longitude coordinate reference system, given by the position of the pixels, also mapping the land use with a third dimension given in a grey scale, where each value has a unique correspondence to a land use category. Next, the information from all the TIFFs was converted into a XYZ format, digestible for R to read as a Comma-Separated Values (CSV). Given the richness of all possible classes for land use, similar classes were aggregated into 5 comprehensive classes: cropland, pasture, forest, artificial, and others (water bodies, humid zones, permanent snow and ice, etc.). However, despite its validation in [69], the total values of the grouped categories fell behind the expectations by not agreeing with the trends and the absolute values registered in Instituto Nacional de Estatística (INE) and Food and Agriculture Organization Corporate Statistical Database (FAOSTAT). The adversity found possibly came from the fact that during the procedure of class aggregation there was uncertainty due to several overlapping categories, thus leading to results divergent from the data sources used for comparison. Therefore, another procedure was developed to harmonise data sources and obtain the final data set used here.

[1] The contour data was downloaded from this repository: https://gadm.org/download_country_v3.html.

The final data set was constructed from several sources and a concise explanation for the origin of the data is as follows:

1. Permanent Pasture - from 1961 to 1988 there was data from FAOSTAT, where it was observable no variation of the evolution of pasture's area; from INE[2] there was data for the years of 1989, 1993, 1995, 1997, 1999, 2003, 2005, 2006, 2007, 2008, 2013 and 2016; the rest of the data was imputed through linear interpolation using neighbouring data points. Given the proximity between known points, more complex methods of imputation seemed unjustifiable, hence linear interpolation was the chosen method;

2. Permanent Culture & Arable Land - from 1989 to 2016 the data was also populated from INE and the imputation was done in a similar fashion as in the previous category; from 1986 to 1988 there was data from INE for culture land without arable land being taken into account, so to simplify we assumed that the variation of culture land for those 3 years would be similar to the variation of arable land and estimated those 3 years using that variation; from 1985 to 1961 we used data from FAOSTAT. As a validation, if we sum the data from 1) and 2) year-wise we got the total agricultural area per year, which was consistent with the combined data from INE for 7 sparse years;

3. Forest - for the years of 1961 and 1980 we got the approximate values from Nunes [70]; initially we estimated the intermediate values via interpolation, but for the years of 1965 and 1985 we found historical values from ICNF [71] that corroborate the interpolation and were then used; for the intervals $[1962; 1964]$, $[1966; 1979]$ and $[1981; 1984]$ regular linear interpolation was used; from the Inventário Florestal Nacional (IFN) of 1995, 2005, 2009 and 2015 more nodes were added and another set o linear interpolations were made, filling the data until 2016. As validation, exists FAOSTAT data in agreement with the data from IFN;

4. Urban/Artificial - for the years of 1995, 2005, 2010 and 2015 data points from the IFN were used; for the year of 1980 data points from Meneses [72] were used; the data used was coherent with data from cartography projects Carta de Uso e Ocupação do Solo (COS) and CORINE Land Cover (CLC); linear interpolation was used to fill the missing data until 2016;

5. Others - remaining area, such as water bodies, humid zones, permanent snow and ice, etc.

All this data that was used for modelling is accessible on appendix A, table A.1.

3.2.2 Final Exergy

Final exergy time series were purchased from International Energy Agency (IEA) [73] up to the year of 2016, which set the boundary up to which the models were trainable. The first forecast necessarily occurred past that limit. For the land use modelling, final exergy for the agricultural sector was used. Concerning the emissions modelling, we used total final exergy, representing the usage from all sectors.

[2]Available in: https://www.pordata.pt/Portugal

3.2.3 Economy

All the GDP data from 1961 to 2016 was fetched from [74]. Figure 3.1 incorporates all the data formerly discussed and input to the models during the first training stage.

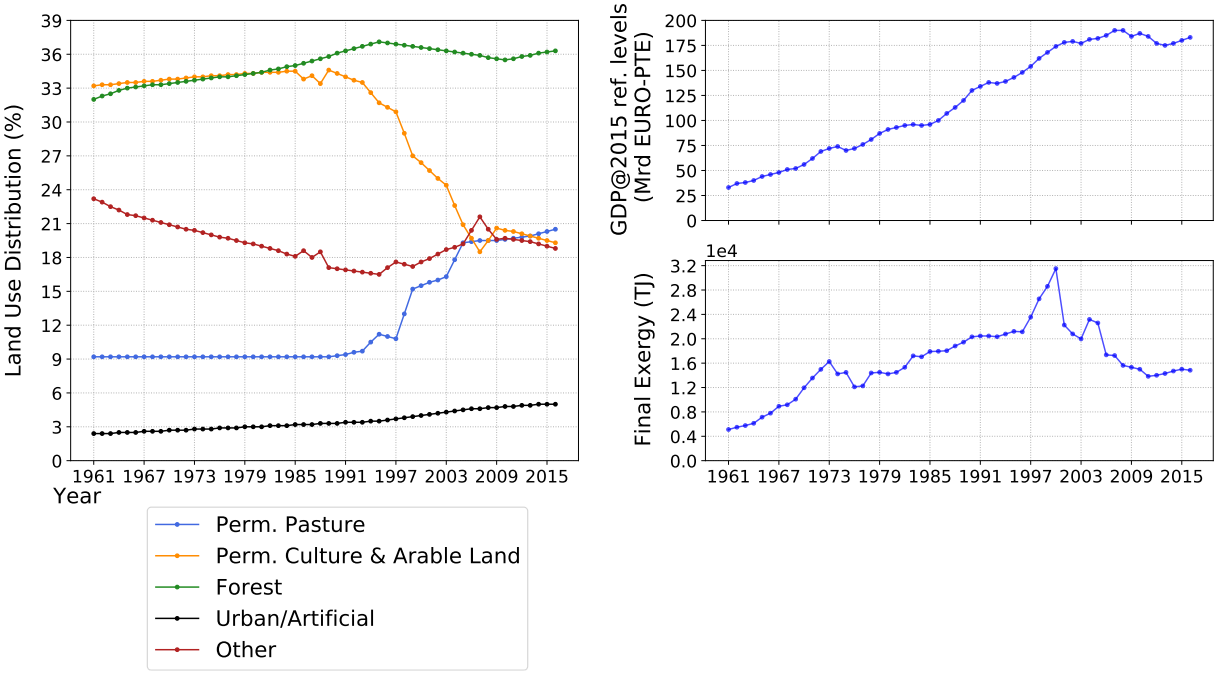


Figure 3.1: Data input to the model in the land use modelling.

The left panel contains the land use distribution data throughout time; the top right panel depicts the GDP evolution at 2015 reference levels; the bottom right panel trends the agriculture/forestry final exergy. During data preparation, the GDP and exergy data referring to 1962 is left out of the input data to allow for a lag within the features.

3.2.4 Greenhouse Gases

At last, we compiled GHG time series that went up to 2016. These series were divided into two, which consisted of emissions from the agriculture sector [75] and emissions from energy sectors [76], whose references are the emission factors from the Intergovernmental Panel on Climate Change (IPCC) 2006 Guidelines [77]. The data used as input for the emissions modelling is plotted in figure A.1.

3.2.5 Policies

Some land-use changes were supported by the policy measures implemented at the time, whose combined effects had an impact in recent land-use observations. Table 3.1 presents a summary of the main changes in the socio-economic drivers for the period of 1928 to 2013, that helped to explain the temporal behaviour of the data sources. Most notably, afforestation incentives from private forestation policies (1965-1983), wheat policy reforms (1970), and common agricultural policy reforms (1992-1996) are reflected in the forest land evolution. Further, agrarian reforms (1975-1982) and agricultural transitory measures (1980-1986) explain the positive evolution of permanent culture area for that period.

Policy	Relevant measures	Scope of change	Temporal scope
Wheat campaign	Minimum prices for wheat; Land clearing subsidies; Protectionist policy.	Until 1963 the support scheme includes minimum prices, land clearing and fertiliser's subsidies. From then on land clearing subsidy is suspended.	1928-1963
Private forestation policy		The support includes loans covering 50% of installation costs.	1965-1983
Wheat policy reform	Support for the afforestation of marginal lands.	The reform includes more incentive to marginal land rehabilitation through afforestation and structural measures.	1970
Agrarian reform	Output price supports; Input price subsidies; Land market regulations; Agricultural credit programmes.	After 1974 output price supports, input price subsidies, land market regulations, and agricultural credit programmes are implemented. Cereals and milk/meat sectors are among the most supported.	1975-1982
Agricultural transitory measures	Access to various EC financial support programs to develop the agriculture and forestry sector, strength production capacity and prepare the country for international markets.	Modernization of agricultural infrastructures and investments in agriculture intensification. In some areas, support to regionally adapted crops and breeds. This corresponded to the CAP transition period.	1980-1986
PEDAP — specific programme for the Portuguese agriculture	Investment support; Production support.	Intensification of production. Livestock increased due to direct payments.	1986-1992
Common agricultural policy reform (MacSharry's)	Direct payments and coupled supports; Compensation measures as forestation support; Agri-environment measures; Early retirement; Set-aside.	The 1992 CAP reform introduced direct payments per animal, reducing intervention prices. This was an incentive to livestock intensification. Parallel, measures were created to plant new forest areas and to protect extensive pastures (agri-environmental schemes) — but the budget of these agri-environmental and afforestation measures are much smaller than production payments and the impact is mostly on marginal and follow-up measures aimed at reducing the negative impacts of agriculture in Natura 2000 and improve agro-ecosystems services.	1992-1996
Common agricultural policy	Compensation payments.	This period resulted in a continuation of the previous reform with direct payments and intensification incentives.	1996-2000
Common agricultural policy	Compensation payments; Environmental and cultural landscape payments (2nd pillar).	In this period the rural development share of the CAP was separated into the second pillar although with a small budget. Direct area payments were implemented. The 2003 mid term review introduced the decoupled support, although in Portugal livestock payments were kept coupled.	2000-2013

Table 3.1: Main changes in socio-economic drivers of land use from 1928 until 2013 [66, 67].

3.3 Coding Framework

In terms of execution, some caveats are already predictable. The data has to be divided into three sets: training, validation, and testing. The training set is used to train the model, the validation set is used for an unbiased evaluation of the model's fit on the training data by helping to tune the model's hyperparameters[3], and the testing set serves the purpose of providing a final unbiased evaluation of a final model's fit on the training dataset. This already poses the problem of having to define a weighted proportion for each set. Then, there is a need of deliberating another collection of parameters: the total proportion of data to be used, *i.e.*, how far back the model "sees" the data, which regulates the information allowed for the model to study; the window/batch size and lag, which are a couple of parameters that permit regulate each block's size and time-space between them, strongly shaping the way data is to be fed into the model sequentially; the sampling rate of information, which gives additional context from the observations to be input, affecting the performance of a predictive model in situations of slow departure from stationarity, for example, and so forth. All of the above has to be conjugated with two major problems: overfitting and low representativeness of data (roughly 60 points for each covariate are expected due to the annual frequency of the data). Hence, we are standing before some obstacles that had to be overcame with systematic adjustments to the parameters.

3.3.1 Creating the Workspace

Keras is a high-level[4] deep learning Application Programming Interface (API) written in Python that accommodates libraries used to build, train, evaluate, and execute neural networks. Keras has been gaining popularity due to its flexibility and ease of use, by providing essential abstractions and building blocks. To perform the required computations it has to run on top of the ML platform tensorflow[5], which is known for efficiently executing low-level tensor operations on Central Processing Unit (CPU) and Graphics Processing Unit (GPU), and for computing the gradient of arbitrary differentiable expressions, a much needed ability for solving machine learning problems, as explained in section 2.2.4.2.

For reproducibility, the software's versions of the programming language and main libraries used in the project were:

- Python 3.8.4;
- Keras 2.3.1;
- Tensorflow 2.3.0;
- Sklearn 0.22.2.post1;
- Scipy 1.4.1;

[3]Note that the model "learns" from the training set only.

[4]High-level interfaces are designed to enable the programmer to write code in a shorter amount of time and to be less involved with the details of the software module.

[5]As keras does not handle itself low-level operations such as tensor products, convolutions and so on, it relies on a specialised tensor manipulation library to do so, serving as the "backend engine". Tensorflow is an open-source symbolic tensor manipulation framework developed by Google.

- Kerastuner 1.0;
- Numpy 1.18.5;
- Pandas 1.0.4;
- Matplotlib 3.2.1.

As neural network algorithms are stochastic, there is randomness associated, such as when initialising the network's random weights. Hence, for complete reproducibility, the seed of all the random generators involved in the dynamics of the multiple algorithms was set constant. By doing this, we guarantee stable and repeatable results. Neural networks use randomness by design to ensure they effectively learn the function being approximated for the problem, as this class of ML algorithm performs better with it.

3.3.2 Methods

To keep track of all the dynamics, the code was built modularly, having multiple scripts performing different tasks. This enables the reproduction of precise transformations on any dataset and it results in the construction of libraries of transformation functions that could be reused in future projects. The built modules are:

- A data transformation pipeline that shapes the data into the appropriately expected format, including the following transformation functions:
    1. Loading method: Firstly, the data was loaded from a CSV file into a Pandas data frame, then the column containing information about the year was dropped, favouring automatic indexing;
    2. Lagging method: To the loaded data frame was applied a negative index shifting (backwards) on the data referring to the features to be lagged, for t periods. t instances of data were lost in the process;
    3. Splitting method: The data was divided into training, validation, and testing sets. A minimum proportion of roughly 70% of the data was set aside for training, the remaining was equally split between the validation and testing sets. With proper validation in mind, several folds of data were created with different proportions of data, and a fold selector parameter was designated to make use of them individually when training (please see section 3.3.3);
    4. Feature scaling method: Although in neural networks there is no theoretical requirement for normalising the data, it has been shown that it can lead to faster convergence of the training criterion in RNNs [78]. This transformation was done for each feature (column) by subtracting the minimum value registered in the test set and dividing by the result of subtracting that minimum to the maximum value registered in the same interval, rescaling the values so that they end up ranging from 0 to 1. As mentioned previously, lookahead must be prevented and that was ensured because the normalisation scaling factor was obtained only from the

training set and was then applied to the three sets in order to avoid propagating back in time information from the future. If we were to use a scaling factor obtained from the entire data set we would possibly retrieve information about the maximum and minimum values from the future, jeopardising the learning process;

5. Batching method: Provided that neural networks were trained via variations of (stochastic) gradient descent, training took place on small batches of data at a time[6], with one epoch meaning that all data had been used for training (although not all at the same time). Therefore, it was required to transform the full array of sequential data into batches of data with a specific window size m, *i.e.*, the number of samples in the sequence, and also with a sampling rate of information r. For a model that saw m time-steps looking backwards (window/batch size), the data was chopped every m rows and the resulting sub-list was appended to a list that contained all the data transformed. The batch size m determined how many batches made up an epoch. The transformation was necessarily different for the labelled targets, where a lag l had to be established based on how far in the future the prediction was expected to happen. The following tables 3.2 and 3.3 of pseudo multivariate time series depict an example of batching with window size $m = 3$, sampling rate $r = 1$ and lagging of $l = 1$ time-step across all features (predicting one step ahead).

Timestep	A	B	C
$t-2$	1	2	3
$t-1$	4	5	6
t	7	8	9

Table 3.2: Features' kth batch

Timestep	A	B	C
$t-1$	4	5	6
t	7	8	9
$t+1$	0	1	2

Table 3.3: Labelled targets' kth batch

As stated before, the number of data points used for training here was lower than what is typically found in similar exercises. With the previous batching approach, each data point had only been included in one sample time series. However, with a different approach, each data point could be in multiple different time series, occupying a different position in each one. Taking a simple example, the time series from tables 3.4 and 3.5 show the $kth + 1$ batch for both features and targets.

Timestep	A	B	C
$t-2$	4	5	6
$t-1$	7	8	9
t	0	1	2

Table 3.4: Features' $kth + 1$ batch

Timestep	A	B	C
$t-1$	7	8	9
t	0	1	2
$t+1$	3	4	5

Table 3.5: Labelled targets' $kth + 1$ batch

When comparing with tables 3.2 and 3.3, it is clear that what happened was that the window was slid over the data set by 1 unit forward (the time-steps appear static because the reference was set within the batch itself). This approach generated $(m-1) \times -1$ more

[6] If we were to train the recurrent neural network directly on the full sequences, the RNN would be equivalent to a deep net with hundreds of layers and we would have a single, very long, instance to train it.

individual overlapping time series samples than with the approach of chopping the data into non-overlapping time series samples (assuming that for the latter the data set size was a multiple of the batch size). At the end of this pipeline, 6 3-dimensional tensors ($\#bathces \times \#features \times \#window\ size$) sets (training, validation, and testing for both features and labelled targets) and a transformation scaler were returned, with the latter serving the purpose of scaling back the outputs.

An important detail about the splitting is that it was done in such a way that there was some overlapping between the training, validation, and test sets, with the intention of enhancing the cardinality of batches. This is usually forbidden because it is a way of lookahead, however, in this case, it was allowed due to the following: the type of the network was sequence-to-sequence, meaning that data over the last m time-steps was fed to the network and the network output the data shifted by l days into the future. All the outputs were needed during training, but only the output at the last time-step was useful for predictions and evaluation. Hence, instead of a hard splitting between sets, the validation set contained the last $m-1$ time-steps from the training set and an unseen time-step, and the test set contained the last $m-1$ time-steps from the validation set and another unseen time-step. When validating, we first fed the network m time-steps, from which $m-1$ were already used during training, and a final new time-step, which was the only instance of interest in terms of evaluation. Likewise for the test set;

6. Extract method: As only the output at the last time-step was useful for predictions and evaluation, it is handy to have a function that would extract the last instance from all batches in a set of batches. In this way, data was readily available for analysis and visualisation;
7. Missing feature appender: Since, for land use modelling, one feature was left out to decrease the computational loading strain during the training phase, it was required to add it posteriorly.

Splitting across time assumes that the patterns the neural network can learn in the past will still exist in the future. In other words, it was assumed that the time series were stationary. In a stationary time series, its mean, variance, and autocorrelations (*i.e.*, correlations between values in the time series separated by a given interval) do not change over time. Statistical methods for time series analysis, such as Autoregressive Integrated Moving Average (ARIMA), require a stationarised time series for forecasting [36], and this is a little restrictive in the sense it excludes time series with trends or cyclical patterns. Nonetheless, recurrent neural networks do not have this requirement, they can, in principle, learn the non-linear and non-stationary nature of a time series [79], with the model relying further on the quality and quantity of data.

- Ensemble of neural networks for Bayesian optimisation;

1. Custom metric: Although we rely on the Mean Squared Error (MSE) over all the outputs for training, we used a custom metric for evaluation, to only compute the MSE over the output at

the last time-step. This metric was called upon the compilation of all models;

2. Baseline model: It is useful to have a baseline model, in order to attest whether the models work or they are doing worse than the basic model. A simple approach was to use a fully connected network, the simplest trainable model. Since it expects a flat list of features for each input and the data was in the form of batches, we needed to add a flattening layer. The model was just a simple linear regression so that each prediction would be a linear combination of the values in the time series. As input, the first layer expected a shape of $[\#steps \backslash batch, \#features]$ and the last layer was outputting all desired targets. The function that generated the model had as arguments the features and labelled targets' training and validation data, number of epochs and the number of output neurons. The model was compiled using MSE as cost function and the default Adam optimiser;

3. Simple (vanilla) RNN model $\#1$: The first model of a set of 6 consists of a stack of simple recurrent layers. Initially, the type of initialisation and respective activation function, as well as the optimiser and cost function were set. As a regularisation, a constraint to the maximum norm of the weights was also set to helping alleviate possible unstable gradients. Then, an initial learning rate was chosen.

   As the first model's layer was already defined by the inputs, architecture-wise, the first element to be added was the first hidden layer, where within it a number of neurons had to be determined. Besides the antecedently fixed variables, the `return_sequences` and `stateful` variables needed to be addressed. By default, recurrent layers in Keras only return the final output. To make them return one output per time-step, `return_sequences` must be set to True in order to be coherent with the sequence-to-sequence philosophy. A stateful RNN preserves the hidden state between training iterations and continues reading where it left off, allowing it to learn longer patterns. This is obviously the ideal configuration, however, that would imply a non-overlapping batching of the data, and as our data did not suffice the `stateful` argument had to be set to False - the hidden state feature was only used within the batch, discarding the final hidden state when starting the next batch.

   Afterwards, an arbitrary number of layers similar to the previous was added. Those layers differed from the first only in terms of the number of neuron units, where a different number of neurons was assigned to each new layer.

   Finally, the output layer was specified. This layer contained a number of neurons equal to the number of desired outputs. In order to fully turn the model into a sequence-to-sequence model we had to apply a dense layer[7] at every time-step, and this was conceivable with a special layer called `TimeDistributed`. This layer wraps any layer and applies it at every time-step of its input sequence.

   The output layer's type was not set as `SimpleRNN` due to the hidden states being most useful in previous layers to carry over all information, and also because a layer of type `Dense`

[7] Remember that we have a dense or fully connected layer when all the neurons in a layer are connected to every neuron in the previous layer (*i.e.*, its input neurons).

runs faster. As no activation function was specified, by default, no activation was applied, and the output was the result of the operation $weights \times inputs + bias$.

After the model was created, we had to call `compile()` method to specify the cost function, the optimiser to be used and, optionally, any list of extra metrics to compute during training and evaluation. In this case it was also necessary to set the argument `sample_weight_mode` to Temporal because we wanted to do time-step-wise sample weighting (2D weights). Here it is shown a partially pseudo code of what the model implementation looks like.

```
def Simple(hyperparameters,nr_features,output_neurons,stateful=True):
  #an initialiser is chosen from a list of all initialisers
  kernel_initialiser = hyperparameters.Choice('initialisers')

  #an activation function available to the previously
  #chosen kernel initialiser is chosen from a list
  activation = hyperparameters.Choice('activation functions')

  #an optimiser is chosen from a list of configured optimisers
  optimiser = hyperparameters.Choice('optimisers')

  #a loss function is chosen from a list
  loss = hyperparameters.Choice('losses')

  #initial learning rate is set
  learning_rate = 0.2

  #it rescales the weights incident to each hidden unit
  #to have a norm less than or equal to a desired value
  kernel_constraint = MaxNorm(1.)

  #set up of sequential model composed of stacks of
  #layers connected sequentially
  model = Sequential([
    layers.SimpleRNN(units=hyperparameters.Int(min,max,step),
    kernel_initialiser,activation,kernel_constraint,
    return_sequences=True,stateful,input_shape),
    for i in range(hyperparameters.Int(min,max,step)):
      layers.SimpleRNN(units=hyperparameters.Int(min,max,step),
      kernel_initialiser,activation,kernel_constraint,
      return_sequences=True,stateful,input_shape),
    layers.TimeDistributed(layers.Dense(output_neurons))])
  model.compile(optimiser,loss,sample_weight_mode='temporal',
  metrics=['Custom metric'])
```

```
35          return model
```

4. LSTM model $\#2$: This model is structurally equivalent to model $\#1$, except that every `SimpleRNN` cell was replaced by a LSTM cell;
5. GRU model $\#3$: Here the explanation is similar to the previous one, but, instead of LSTM cells, GRU cells were used. No code is exposed given the fact that it would be redundant for both models $\#2$ and $\#3$;
6. Convolutional model $\#4$: On the causal convolutional model, besides the setting of the same variables as in model $\#1$, the number of filters and the kernel size needed to be specified for the convolutional layers. This sequential model started with a layer called `InputLayer` which was responsible for explicitly setting the input dimensions. Besides the explicit input layer, this model was constituted only by convolutional 1D layers, each of which had an arbitrary number of filters and kernel size attributed[8].

   Every convolutional 1D layer used 'causal' padding enabling dilated convolutions and ensuring that no lookahead happens, by padding the inputs with the right amount of zeros on the left so that they won't contribute to the final value and making the convolution to run only against the actual boundaries of the matrix without including imaginary empty cells, such as in zero padding. Also, the stride number was set to 1 by default. Multiple sets of convolutional layers were added using growing dilation rates (the cardinality of dilation rates fixed the cardinality of layers within each set), a concept presented in section 2.2.5.8. At last, the output layer was defined with a number of filters equal to the expected number of outputs, a unitary kernel size and with no activation function. A portion of partially pseudo code is shown here.

```
1       def Convolutional(hyperparameters,nr_features,output_neurons):
2         #ipsis verbis from line 2 to line 16 in model #1
3         (...)
4
5         #a number of intermediate layers is chosen within a range
6         n_layers = hyperparameters.Int(min,max,step)
7
8         #a number of filters is chosen within a range
9         filter = hyperparameters.Int(min,max,step)
10
11        #the kernel size is chosen within a range
12        kernel_size = hyperparameters.Int(min,max,step)
13
14        #it rescales the weights incident to each filter tensor matrix
15        #to have a norm less than or equal to a desired value
16        kernel_constraint = MaxNorm(axis=[0,1])
17
```

[8] As well as in preceding models, the total number of intermediate layers was going to be determined upon call from the Bayesian algorithm, this and previous arbitrarinesses would be commanded by it.

```
        #set up of sequential model composed of stacks of
        #layers connected sequentially
        model = Sequential([
          layers.InputLayer(input_shape),
          for rate in (1,2,4,8)*n_layers:
            layers.Conv1(filter,kernel_size,padding='causal',kernel_initialiser,
            kernel_constraint,activation,dilation_rate=rate),
          layers.Conv1D(filters=output_neurons,kernel_size=1)])
        model.compile(optimiser,loss,metrics=['Custom metric'])
        return model
```

7. Convolutional 1D-LSTM model $\#5$: The first hybrid model from the set is an interplay between convolutional and LSTM models. As explained in section 2.2.5.9, on this model the first layer is convolutional and it differs from the layers in model $\#4$ just in the padding. Here that argument was set to 'same' (equivalent to zero padding), along with an unitary stride, to force the output sequence to have the same length as the input sequence. The subsequent layers were of the type LSTM, and a final `TimeDistributed` layer, and no modifications had been further introduced when comparing to model $\#2$;
8. Convolutional 1D-GRU model $\#6$: The second hybrid model is a reimplementation of the first hybrid model but with GRU layers, instead of LSTM.

The lists of kernel initialisers and respective activation functions are both in agreement with what has been presented in item 5, section 2.2.4.2, and all were inserted in the list of possibles choices. In terms of optimisers, `RMSProp`, `SGD`, `Adam`, `AdaMax` and `Nadam` were available configured options. Across all optimisers, they share both learning rate and `clipnorm` parameters. Clipping gradients during backpropagation so that they never exceed some threshold is another technique used to mitigate possible exploding gradient problems, and the `clipnorm` parameter served that purpose. Every component of the gradient vector was clipped if its l_2 norm (Euclidean norm) was greater than the chosen threshold. Additionally, the SGD optimiser was set to adopt a variant called Nesterov Accelerated Gradient (NAV) when the parameter `nesterov` was set True. NAV is generally faster than regular momentum optimisation [46]. For the loss parameter, both MSE and MAE were viable choices.

All model builder functions, except for the Baseline model, returned a compiled model that used hyperparameters defined inline during the hypertuning process. For every hyperparameter, a search space, from which they can be sampled, was bounded.

- Hyperparameter tuner based on Bayesian optimisation;

1. Seed setter: this function set the `PYTHONHASHSEED` environment variable to 0, this is necessary in Python 3.2.3 onwards to have reproducible behaviour for certain hash-based operations; it set the NumPy seed because Keras gets its source of randomness from the NumPy

random number generator; TensorFlow has its own random number generator that is also seeded constantly with this function; the core Python random generator was seeded as well. The top of the code had this function called before anything else, providing reproducibility;

2. Optimiser method: This is a multi-purpose function that harboured several interlinked methods with the final goal of extracting useful statistics for the analysis of the best models. As arguments, this function received one (hyper)model from the ensemble, the full dataset, the fold selector and a path to indicate where to write all the statistics. Making use of the data transformation module, the desired datasets (training, validation, testing) were then created;
3. Column-wise RMSE method: It is desirable to have disaggregated calculations of the error, one for each category. Hence, a method was written to provide this functionality when two matrices with the same dimensions were fed into this function. Each column of both matrices represented one category, evolving through time in the rows dimension.

The hyperparameter optimisation framework was specified by first instantiating a tuner of the type `BayesianOptimization`. For this, we indicated the hypermodel, the objective (metric) to optimise, the total number of trials (model configurations) to test at most (if the search space was exhausted the search would be interrupted before), the number of randomly generated samples as initial training data for Bayesian optimisation, a working directory to save the results from all trials, and a project name that allowed to store related searches within the same working directory (it is useful because there would be different proportions of data tested in the same model).

Before running the hyperparameter search, it was relevant to define some handy callbacks[9]. The `search()` method accepts a callbacks argument that lets us specify a list of objects that Keras would call at the start and end of training or each epoch.

Earlier, all learning rates were set constant and this is not optimal, as discussed in item 4, section 2.2.4.2. Therefore, the learning schedule `ReduceLROnPlateau` (equivalent to the performance scheduling) was created in the form of a callback, later passed into the search function. This callback monitored our custom metric and if no improvement was seen for 6 (empirical choice) epochs, the learning rate was reduced by half, halting when the learning rate got lower than 0.001 (also empirical).

Another important feature to have is the ability to halt the training process whenever no progress on the customised metric, concerning the validation set, is measured[10], sparing computational resources. A way of implementing this was by using the `EarlyStopping` callback. This callback monitored the same quantity as the previous callback, however, it intervened at a later number of epochs, 14. The justification for this number being larger than the 6 epochs defined anteriorly is that the dynamic reduction of the learning rate was expected to prevail at the beginning of the training session, and after it had reached its minimum of

[9] A callback is a function that gets passed into another function as an argument, and it is invoked during the execution of the parent function.

[10] Progress on the training set is expected to occur continuously due to the natural overfitting of the training data.

0.001 the training was kept being monitored for 14 epochs, if no improvements were observed in the MSE of the output at the last time-step then the training was stopped.

TensorFlow provides a visualisation tool called TensorBoard that allowed us to view the learning curves during training, and Keras provides a `TensorBoard()` callback. The way it works is by outputting the data we wanted to visualise to special binary files, then the TensorBoard server monitored the logging directory, automatically picking up the changes and updating the visualisations. This tool was pertinent to track the evolution of the custom metric on the validation data and also to observe how the learning rate was being adjusted.

To run the hyperparameter search, the `search()` function was called. As arguments, this method took training data, validation data (to evaluate the loss and any model metrics at the end of each epoch), the number of epochs to train the model, and a list of callbacks to apply during training. A list of the three mentioned callbacks was homogeneously applied on all tested models, baseline model included.

After the search on the hyperspace, for a given number of trials, had been completed, the three best models were saved, hereafter designated sub-models. By doing so, we were able to generate enough statistics on how the specified architecture was behaving with the fed data. In terms of calculated statistics, the MSE was being computed, by using the `evaluate()` method, for the three best models when features and labelled target validation data was fed. The purpose was to further compare it with the MSE from the benchmark Baseline model for the same data. In the end, the three models were set to predict, with the `predict()` method, on the test data, and with those results the RMSE per target was calculated to better understand the deviation from the true values for unexplored data. It is worth mentioning that the MSE concerning the validation data was calculated on scaled data during the training period, whereas the RMSE with respect to the testing data was calculated on non-scaled data.

To finalise, the model's name, the working fold of data, the MSE from all evaluations on validation data, the MSE from the Baseline model's evaluation on validation data, the RMSE from the prediction on test data, the nominal predicted/forecasted values, and the nominal true values were automatically saved in a .json file[11]. The storing of the predicted and true values was carried on with the help of a custom JSON encoder class that is used to serialise NumPy arrays, not readily JSON serialisable.

- Score iterator for a rigorous exploration of all details within all trials;

1. Get scores method: score here is defined as the lowest MSE on the validation data (from the custom metric) achieved during the training process at a given epoch. This number is associated with a set of hyperparameters for a given trial, hence it is important to search

[11] "JavaScript Object Notation (JSON) is a lightweight data-interchange format. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.", in https://www.json.org/.

through possibly thousands of trials to study the conditions that originated the best trials. For that, firstly, a path of the location of all trials for a given model was provided to the function. Then, all sub-folders were recursively accessed and registered (1 level of depth is expected). Within each sub-folder, the program looked for .json type files, where all the labelled target information was located (product of the Bayesian search) and saved the respective file paths. Afterwards, for every file path, the score was retrieved and saved along the file path in a dictionary (Python data structure). Subsequently, the generated dictionaries were sorted by score, with the lowest on top of the list. Now that we had the path associated with the lowest score, it was a matter of accessing the file and search for the model configuration responsible for the good results. This method returned multiple dictionaries with all hyperparameters for the best trials, separately.

- Utilitarian tools for data analysis and visualisation.

  1. Deserialisation method: Before analysing the results, it is fundamental to have a tool that eases the burden of retrieving the results from several .json files across models and folds. Provided we input a path, fold number and a boolean, stating whether or not inclusion criteria should be applied, this method returned the RMSE from the prediction/forecasting on test data, the nominal predicted/forecasted values, and the nominal true values previously stored, constrained to some criteria about the inclusion of data. By doing so, we had data easily processable;
  2. Customised plotting methods: Data visualisation is critical to gain insights on the data we are manipulating. Also, after going through the training procedure and analysing the inclusion of the results from each of the models, as well as any other forecasting exercise, it was desirable to have the capability of visualising the results along with, if applicable, the respective errors. In order to accomplish this, multiple visualisation methods were designed. As some features' values differed greatly, the technique of broken axis was manually implemented as well as plotting two scales on the same axes, when suitable. As arguments, those methods took the full data set, an integer to select the fold of data, a string to indicate which scenario was being studied for the final analysis and respective data to load, a boolean that dictated if error bars should be included when applicable, and a boolean that stated whether the plot was automatically saved. Hence, the code remained unchanged even if new data were to be generated, the linking was directly made to local paths[12].

3.3.3 Performance Estimation

In a forecasting task it is extremely important to be capable of estimating an error that a predictive model will incur on unseen data. The time-agnostic solution for validation proposed in section 2.2.4.4 needed to be adapted for time series, where dependency among observations was expected, instead of independent and identically distributed data. There is no consensus on what is the most appropriate

[12] All code & data used in this thesis are available on GitHub at https://github.com/brunopereiracosta/masterthesis.git.

way to estimate model's performance on time series. Nonetheless, in a study [80] the authors compared different variants of cross-validation (detailed in section 2.2.4.4) and different variants of Out-Of-Sample (OOS) approaches using two cases study: one with real-world time series and another with synthetic time series. They concluded that cross-validation fits better synthetic data and that in real-world scenarios the most accurate estimates are produced by OOS methods, which preserve the temporal order of observations. Due to the real-world nature of our data, proceeding with an OOS method was the natural choice.

Out-of-sample method

Initially, a time series was split into two parts: the first part served as an initial fitting period in which a model is trained. This part was further split into training and validation sets. The last part of the time series was used for testing on unseen observations and then estimating the true loss of the model. This follows the gold standard of testing with training, validation and testing periods that roll forward, avoiding lookahead [36].

Within this method, it is possible to adopt different strategies regarding training/testing split point and growing or sliding window settings, for example. The best way of getting robust estimates of predictive performance is to employ these strategies in multiple test periods, generating more unbiased measures of the generalisation error. The authors of the study [80] applied 6 different strategies, including Monte Carlo simulation, prequential evolution in blocks, growing window, among others. For the neural network model, the strategies that offered the best results were the growing and sliding windows, by order. With this in mind, the OOS growing window method was chosen[13]. Figure 3.2 depicts the algorithm behind the expected error estimation.

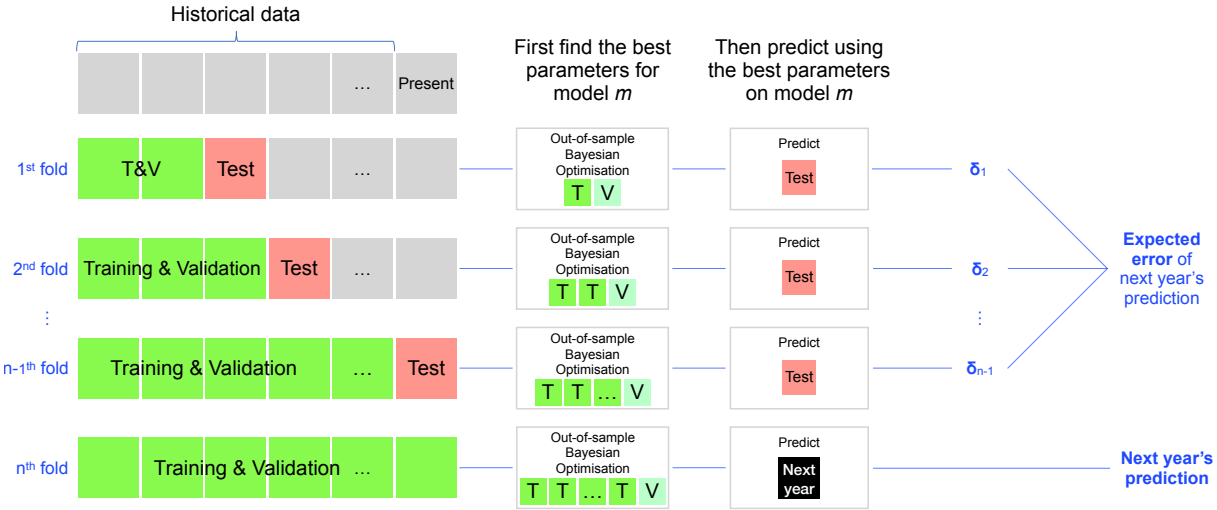


Figure 3.2: Error estimation algorithm based on an out-of-sample growing window approach.

First, by fold, and excluding the n^{th} fold, a model m was built on the training set and the loss estimate that a predictive model m would incur on new observations was computed on the validation set

[13]The OOS with Monte Carlo simulation had comparable results, however, this method brings about the risk of data not being chosen, and given the shortage of data this is undesirable.

(green extension on Figure 3.2). Reiterating the process, a set of models was generated and sorted by their loss estimate. Second, in order to unbiasedly evaluate the estimations produced by the best model's configurations, the top models were re-tested using the testing set (red extension in Figure 3.2). Effectively, we obtained a measure of the error δ, the ground true loss that a model m incurred on new data.

Across each model's type, the performance estimation method was conceived with evaluations in two dimensions: 1) preliminary loss estimate, by converging to the ideal architecture's set up – this measured the magnitude of the difference between the estimated and the actual error on all models; and 2) final loss estimate, by repeating the previous measurement on new data and on the best models. One could think that the generalisation error (out-of-sample error) from the evaluation of the model on the validation data would suffice to infer an error for the prediction, however, it is extremely important to note that by selecting the best configuration based on the results from the validation data we were incurring in an implicit overfitting of the model to that data. Hence, if we want to avoid being biased it was mandatory to compute a second error by evaluating the model on the testing data set.

Ultimately, the process was repeated on the n^{th} fold, the difference is that no final loss was estimated because we were trying to forecast unregistered data, so the past final loss estimates would be the expected error incurred on this data. It is arguable that we were not using all the available data for training and that the predictions were made with a chronological gap, non-consecutively. Nonetheless, as the main interest was to forecast future scenarios, the model should be able to predict several time-steps ahead. Thus, this algorithmic proposal made it more resilient to fit the aim.

For each model m and fold n_{fold}, three of the following prediction matrices for the testing data were generated (see equation 3.1), one per sub-model s,

$$\hat{Y}_{m,s,n_{fold}} = \begin{bmatrix} \hat{y}_{1,1} & \hat{y}_{1,2} & \cdots & \hat{y}_{1,l-1} & \hat{y}_{1,l} \\ \hat{y}_{2,1} & \hat{y}_{2,2} & \cdots & \hat{y}_{2,l-1} & \hat{y}_{2,l} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \hat{y}_{k-1,1} & \hat{y}_{k-1,2} & \cdots & \hat{y}_{k-1,l-1} & \hat{y}_{k-1,l} \\ \hat{y}_{k,1} & \hat{y}_{k,2} & \cdots & \hat{y}_{k,l-1} & \hat{y}_{k,l} \end{bmatrix}, \tag{3.1}$$

where k and l are the indexes for the time-step and target, respectively. Each row contained the prediction $\hat{y}$ for each feature for a given year. Associated with the previous matrices, there was a counterpart matrix with the true data

$$Y_{n_{fold}} = \begin{bmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,l-1} & y_{1,l} \\ y_{2,1} & y_{2,2} & \cdots & y_{2,l-1} & y_{2,l} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ y_{k-1,1} & y_{k-1,2} & \cdots & y_{k-1,l-1} & y_{k-1,l} \\ y_{k,1} & y_{k,2} & \cdots & y_{k,l-1} & y_{k,l} \end{bmatrix}. \tag{3.2}$$

Focusing on the land category that would be left out of the model during the land use analysis, we could estimate the value corresponding to the proportion of that land use by calculating for each time-step

$i = \{1, \ldots, k\}$ and for both matrices 3.1 and 3.2 the following equations 3.3a and 3.3b:

$$\begin{cases} \hat{y}_{i,l+1} = f(\hat{y}_{i,1}, \hat{y}_{i,2}, \ldots, \hat{y}_{i,l}) = 1 - \sum_{j=1}^{l} \hat{y}_{i,j} & \text{(3.3a)} \\ y_{i,l+1} = f(y_{i,1}, y_{i,2}, \ldots, y_{i,l}) = 1 - \sum_{j=1}^{l} y_{i,j} \ . & \text{(3.3b)} \end{cases}$$

Next, the RMSE (see equation 2.1) was calculated column-wise using the previous matrices 3.1 and 3.2 (both updated with the appended feature $l+1$ for the first analysis), for all prediction matrices. These calculations resulted in equation 3.4, an error vector for all sub-models

$$\boldsymbol{\delta}_{m,s,n_{fold}} = [\delta_{\hat{y}_1}, \ldots, \delta_{\hat{y}_{l+1}}] = \left[\sqrt{\frac{1}{k} \cdot \sum_{i=1}^{k} (y_{i,1} - \hat{y}_{i,1})^2}, \ldots, \sqrt{\frac{1}{k} \cdot \sum_{i=1}^{k} (y_{i,l+1} - \hat{y}_{i,l+1})^2} \right] . \quad \text{(3.4)}$$

These exploratory results served the purpose of diagnosing which models or sub-models should proceed in the analysis. There were two acceptability criteria: 1) the MSE from each sub-model on validation data shall not surpass the MSE obtained from the baseline model; 2) the total RMSE for each sub-model on testing data shall be inferior to 15%, which represented an average tolerance of 3% of error per land use category (there are 5 categories), and 7.5% per emission category. For each of the remaining candidate models, an average of the prediction matrices 3.1 from the respective sub-models was computed, as

$$\overline{\hat{Y}}_{m,n_{fold}} = \frac{1}{max(s)} \sum_{j=1}^{max(s)} \hat{Y}_{m,j,n_{fold}} = \begin{bmatrix} \overline{\hat{y}}_{1,1} & \overline{\hat{y}}_{1,2} & \cdots & \overline{\hat{y}}_{1,l-1} & \overline{\hat{y}}_{1,l} \\ \overline{\hat{y}}_{2,1} & \overline{\hat{y}}_{2,2} & \cdots & \overline{\hat{y}}_{2,l-1} & \overline{\hat{y}}_{2,l} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \overline{\hat{y}}_{k-1,1} & \overline{\hat{y}}_{k-1,2} & \cdots & \overline{\hat{y}}_{k-1,l-1} & \overline{\hat{y}}_{k-1,l} \\ \overline{\hat{y}}_{k,1} & \overline{\hat{y}}_{k,2} & \cdots & \overline{\hat{y}}_{k,l-1} & \overline{\hat{y}}_{k,l} \end{bmatrix} . \quad \text{(3.5)}$$

The aggregate results represent the expected response for each model to this type of data. Reducing a model to a sample sub-model would be insufficiently representative of how well it predicts, hence the ensemble of sub-models for intermediate predictions with respective errors and posteriorly the ensemble of models utilised for the final prediction. A richer response is expected to arise with the contributions of multiple sources of models.

Now, a vector composed of the expected errors for each category was estimated. These RMSE calculations were carried out with the results from 3.5 and true values from 3.2,

$$\overline{\boldsymbol{\delta}}_{m,n_{fold}} = [\delta_{\bar{\hat{y}}_1}, \ldots, \delta_{\bar{\hat{y}}_{l+1}}] = \left[\sqrt{\frac{1}{k} \cdot \sum_{i=1}^{k} (y_{i,1} - \overline{\hat{y}}_{i,1})^2}, \ldots, \sqrt{\frac{1}{k} \cdot \sum_{i=1}^{k} (y_{i,l+1} - \overline{\hat{y}}_{i,l+1})^2} \right] . \quad \text{(3.6)}$$

Lastly, all errors from equation 3.6 were combined across all folds and models, resulting in global error vector. These values would be the uncertainty associated with all models' averaged out prediction for each feature:

$$\boldsymbol{\delta} = \frac{1}{max(m) \cdot max(n_{fold})} \sum_{i=1}^{max(m)} \sum_{j=1}^{max(n_{fold})} \overline{\boldsymbol{\delta}}_{i,j} \ . \tag{3.7}$$

In the end, this committee of models was expected to produce a robust estimation of $Y^{predicted}$ by computing

$$Y^{predicted} = \frac{1}{max(m)} \sum_{i=1}^{max(m)} \left(\frac{1}{max(s)} \sum_{j=1}^{max(s)} Y_{i,j}^{predicted} \right) \ , \tag{3.8}$$

where $Y_{i,j}^{predicted}$ represents the prediction matrix of each model m and each sub-model s optimised in the last existing fold. To this forecast, the expected error $\boldsymbol{\delta}$ would be associated.

This rationale is identically applicable to the problem regarding GHG emissions provided we do not add more dimensions as in equations 3.3a and 3.3b. In that circumstance, the dimensions of the matrices were adjusted according to the dimensions of the network's output.

3.3.4 Training Procedure

To wrap it up, here is the flow of the training process, explicitly stating the parameters used:

- The data was sub-divided into 5 folds[14] with the growing window method, and the last fold was constituted only by training and validation data. Oppositely to what happens in the training set, both validation and testing set preserved their absolute size for all folds (see item 3, section 3.3.2, for more details on how the splitting is performed);
- For each fold, the flatten data was transformed into batches and normalised taking into account just the training set for the scaling factor;
- All 6 models hyperparameters' boundaries were specified, the models were then ready for instantiation. Some worth mentioning are the maximum number of intermediate layers and the maximum number of neurons for each of those layers, which were set to 3 and 100 (with steps of 5), respectively, for all models where applicable. The convolutional layers had a maximum of 15 filters (with steps of 2) and a kernel with maximum size of 6. The learning rate was universally set to start at 0.2, as indicated in the code presented in item 3, section 3.3.2;
- For every model and for every fold, OOS Bayesian optimisation was deployed, enabling the search of the hyperparameters that best generalise for both training and validation data. A maximum of 1000 epochs was set as well as 1500[15] trials (maximum) and 300 random generated samples for the initial training. Those numbers were expected to allow for an extensive search and meticulous tuning;

[14] The data partition was as follows (training, validation, testing):
$1^{st} fold : [0, 36], [31, 42], [37, 48]$
$2^{nd} fold : [0, 38], [33, 44], [39, 50]$
$3^{rd} fold : [0, 40], [35, 46], [41, 52]$
$4^{th} fold : [0, 42], [37, 48], [43, 54]$
$5^{th} fold : [0, 48], [43, 54], -$

[15] For the emissions modelling this value was set to 2000.

- Within every model and fold's search, the best 3 configurations achieved were saved, as there could be innumerable sets of hyperparameters (local minima) that could conduct to good results. Because of this choice of picking the best models, there was an implicit overfitting, so it was favourable to have more models to generate more statistics for the expected error of a forecast. This generated 3 measures of the error for every combination of model m and fold n_{fold} (excluding $n_{fold} = 5$), 72 in total. How these measures of error were treated is explained in section 3.3.3;

- Lastly, the same optimisation was redone on the last fold, $n_{fold} = 5$, which included all data for training and validation. 3 forecasts per model were made and averaged out if previously all models presented an acceptable generalisation power. Otherwise, only the best architectures for our type of data were included.

3.3.5 Forecasting Procedure

3.3.5.1 Land Use Distribution

It is worth to elaborate on how consecutive forecasts were handled when using unrecorded data. As seen on tables 3.2 and 3.3, during the batching process, the features ended up with one less time-step, when compared to the labelled targets. From table A.1 we can see that the last instance of data went up to 2016. This means that the last instance of the last feature's batch was from 2015, limit preserved even after applying the lag. In this case, the lost instance of data was from the year of 1961.

Taking the last feature's batch of data, we dropped the first instance and added to the end a new instance with land use distribution data from 2016, and exergy and GDP data from 2015. This constituted the first effective batch for forecasting on unseen data. As we were constrained to the availability of data on land use distribution, the forecast starting point was 2017. Effectively, we had

$$LULC_{t+1} = f(LULC_t, Exergy_{t-1}, GDP_{t-1}) \xrightarrow{1^{st}forecast} \overbrace{LULC_{2017}\underbrace{+Exergy_{2016} + GDP_{2016}}_{\text{juxtaposed data}}}^{\text{new instance}} . \quad (3.9)$$

3.3.5.2 Greenhouse Gases

The second batch of models was input with the land use data produced from the first forecasting exercise. Due to the way lagging was performed, one instance of data regarding the year of 2016 was lost, which results in an earlier starting forecasting point, 2016 (see analogous justification in 3.3.5.1). The procedure for forecasting batching creation was reiterated equally as in the former case. Forward, the forecasts were iteratively done and the batch was dynamically adjusted to accommodate the new instances, while dropping the oldest:

$$GHG_t = f(LULC_{t-1}, Exergy_t, GDP_t, GHG_{t-1}) \xrightarrow{1^{st}forecast} \xrightarrow{1^{st}forecast} \overbrace{GHG_{2016}\underbrace{+Exergy_{2017} + GDP_{2017} + LULC_{2016}}_{\text{juxtaposed data}}}^{\text{new instance}} . \quad (3.10)$$

As at the time of writing we were approaching the end of the year of 2020, solid estimations of the GDP for 2020 already existed [74], hence they were justifiably included in the data considered for forecasting (see tables A.3, A.4, A.6, and A.7). From this point onwards, new data was adopted from two scenarios for Portugal in 2030 developed in the MEET project report [42], in order to extend the final exergy data, available up to 2016, and to extend the GDP data that stopped at 2020. These scenarios are a product of a collaborative partnership between industry leaders from around 30 Portuguese companies, from 13 different sectors, representing approximately 20% of the nation's GDP, and government agents, such as the Portuguese Ministry of Environment, the Agência Portuguesa Ambiente (APA), the Portuguese Energy Agency and the Secretary of State for Industry. Plausible trajectories for the Portuguese economy in the context of the fourth industrial revolution (stronger linking between technologies and the physical world) were disputed.

A more pessimistic future marked the first scenario, from now one denoted by scenario P, which was characterised by evolutionary stagnation. As a consequence, both exergy and GDP steadily (but not drastically) decreased until 2030. The scenario described a situation in which Portugal lagged behind, in a world transformed by new developments in technology, the environment and energy. In this scenario of stagnation, Portugal managed to comply with the existing, less demanding requirements of the National Low-Carbon Roadmap, but with a fall in GDP. More unstable international geopolitical and demographic contexts, in terms of political issues, ageing population, and migrations were considered, as well as an increasing global distrust [42]. The simulated data for the land use distribution forecast was placed in appendix A, table A.5.

Antagonistically, a more optimistic scenario, here denoted by scenario O, described a brighter future where Portugal excelled in growth and evolution, resulting in a continuous increase of both final exergy consumption and GDP. This was a scenario of high stability and competitiveness, with significant EU economic growth, financial stabilisation of the Portuguese economy and higher levels of investment [42]. For this scenario, the simulated data was placed in appendix A, table A.8. Note that the GDP time series included a severe economic contraction in 2020, consequence of the 2019/2020 beginning of a worldwide pandemic, caused by a new coronavirus. Supplementary, two sets without the new coronavirus influence were created, with the properties stated for scenarios P and O. Those sets will be part of a sensitivity analysis and are identified as studies A and B, respectively.

In order to deepen the analysis on the dependency of the model on the input parameters, 4 additional sets of data were built. The first set consisted of constant GDP values starting at 2017, and yearly decreases of 5% in the exergy values (study C). For the second set, the modifications were performed in the opposite direction, with an annual increase of 5% for the exergy (study D). The third (study E) and forth (study F) sets are identical to the previous ones, the difference relies on the fact that the changes are mode on swapped time series, *i.e.*, the exergy time series was kept constant and the GDP was varied accordingly.

4

Results and Discussion

4.1 Model Performance

The first phase consisted of the determination of the error associated with the fitting to each category. This is the foundation of every good model and it is necessary to have reasoned conclusions.

4.1.1 Land Use Distribution

Below are the results by fold of data. Starting by analysing the results from table 4.1, four instances (highlighted in red) were classified as non-valid within the criteria set in section 3.3.3. The selected sub-models presented a total RMSE bigger than the threshold of 15%, and for this data the architecture (model) #4 performed the worst, with just one model accepted.

Table 4.2 summarises the calculations from equation 3.6 for each model separately, after averaging the predictions within each model and comparing the result with the true outcome. The more performant networks were both hybrid convolutional 1D-LSTM and convolutional 1D-GRU models. Also, the arithmetic mean for each category was computed, resulting in the error vector presented in the last row of the table. With these errors, it is noticeable that the models struggled to correctly predict the second category. The causal convolutional network had the worst overall score.

In the second fold of data, table 4.3, only one instance was excluded. In this case, the total RMSE surpassed 15%, even though it had the best MSE during the training phase for all folds. This indicates that this instance of model #5 could not achieve a good generalisation power for the test set, strongly overfitting to both training and validation sets.

Table 4.4 denotes that the more performant models in the previous fold are not the same in this fold, they actually became the less performant, along with the causal convolutional model. These variations are expectable due to the first fold being the one with less available training data, some models may pick up patterns faster and be more suitable for less data than others. Globally, the total average error decreased roughly by half with this fold, hinting that having few data leads to worst results, given the learning nature of this type of models.

	Model	Sub-model	Categories					MSE*	
			Permanent Pasture	Permanent Culture & Arable Land	Forest	Urban/ Artificial	Other	Training	Baseline**
RMSE (%)	M1	SM1	2.9	3.4	1.1	0.5	1.3	0.019	1.895
		SM2	3.9	5.9	0.6	0.5	2.2	0.022	
		SM3	3.9	6.2	3.9	1.4	4.3	0.033	
	M2	SM1	3.7	4.5	0.5	0.7	0.5	0.022	
		SM2	4.7	4.9	1.6	0.3	1.6	0.024	
		SM3	1.9	5.9	1.9	0.9	6.4	0.030	
	M3	SM1	4.5	4.0	0.4	0.2	1.0	0.019	
		SM2	4.7	6.2	0.8	0.5	1.9	0.025	
		SM3	2.3	1.5	1.3	0.1	2.0	0.027	
	M4	SM1	2.7	5.4	1.1	0.7	3.2	0.032	
		SM2	4.7	8.1	0.3	0.5	2.8	0.099	
		SM3	8.4	6.1	1.5	0.8	1.8	0.133	
	M5	SM1	2.9	1.1	0.3	0.6	2.5	0.015	
		SM2	2.9	1.8	1.2	0.2	1.9	0.018	
		SM3	3.9	5.4	1.4	0.5	2.5	0.026	
	M6	SM1	2.3	4.4	0.3	0.4	2.0	0.012	
		SM2	4.7	1.4	1.8	0.8	2.8	0.014	
		SM3	4.5	4.5	1.1	0.4	1.0	0.038	

* MSE is measured with scaled data, non directly comparable with RMSE calculated on non-scaled data

** Baseline MSE is estimated only once because there is a unique baseline model

Table 4.1: Losses that models incurred on testing (left) and validation (right) data from fold #0.

Models or sub-models highlighted in red were excluded from the analysis due to not meeting the established criteria for a total Root Mean Square Error (RMSE) below 15%.

	Model	Categories					Total RMSE (%)
		Permanent Pasture	Permanent Culture & Arable Land	Forest	Urban/ Artificial	Other	
RMSE (%)	M1	3.4	4.6	0.9	0.5	1.7	11.1
	M2	4.2	4.7	0.8	0.5	0.9	11.1
	M3	2.3	3.6	0.2	0.3	1.4	7.8
	M4	2.7	5.4	1.1	0.7	3.2	13.1
	M5	1.4	2.2	0.9	0.5	1.4	6.3
	M6	0.8	3.2	0.2	0.5	1.9	6.7
$\delta_{n_{fold}=0}$		2.5	4.0	0.7	0.5	1.7	9.4

Table 4.2: Aggregated (per model) total average losses measured in terms of RMSE for fold #0.

For the two folds on tables B.1 and B.2, presented in the appendix B, no sub-model violated the criteria, which is a strong indicator of the stated fact that indeed more data conducts to more solid results. On tables 4.5 and 4.6 the total average error kept decreasing in comparison to the anterior folds, stabilising below 4%. In all the folds, the causal convolutional model had the worst training MSE when compared to its peers, and in all folds, except for one, it was outperformed by the remaining models. This

suggests that either this model does not fit well to our data or stronger regularisation techniques such dropout, to mention one, or a more varied layer initialisation, with different number of filters and/or kernel size within intermediate layers, needs to be implemented to enhance the training/learning process.

			Categories					**MSE***	
	Model	**Sub-model**	Permanent Pasture	Permanent Culture & Arable Land	Forest	Urban/ Artificial	Other	Training	Baseline**
RMSE (%)	M1	SM1	1.2	2.6	1.3	0.6	1.4	0.022	0.799
		SM2	1.2	2.4	0.2	0.6	1.0	0.025	
		SM3	1.0	2.6	0.6	0.4	2.0	0.027	
	M2	SM1	2.1	0.9	0.8	0.4	3.1	0.013	
		SM2	0.1	0.5	1.1	0.4	0.4	0.015	
		SM3	0.7	1.9	0.6	0.3	0.9	0.016	
	M3	SM1	1.0	2.0	0.8	0.2	0.7	0.006	
		SM2	2.3	2.0	1.1	0.3	1.1	0.008	
		SM3	0.4	0.7	0.4	0.4	1.0	0.012	
	M4	SM1	1.7	1.9	1.5	0.1	1.4	0.042	
		SM2	2.6	1.1	0.3	0.1	3.1	0.044	
		SM3	5.1	3.6	0.8	0.3	1.1	0.045	
	M5	SM1	5.6	7.6	0.5	0.1	1.6	0.003	
		SM2	4.0	1.0	0.5	0.3	3.7	0.011	
		SM3	2.3	0.5	1.0	0.5	0.7	0.014	
	M6	SM1	2.1	0.9	0.5	0.1	1.5	0.014	
		SM2	4.0	1.3	1.1	0.6	1.1	0.019	
		SM3	0.9	1.1	1.5	0.8	0.6	0.022	

* MSE is measured with scaled data, non directly comparable with RMSE calculated on non-scaled data

** Baseline MSE is estimated only once because there is a unique baseline model

Table 4.3: Losses that models incurred on testing (left) and validation (right) data from fold #1.

Models or sub-models highlighted in red were excluded from the analysis due to not meeting the established criteria for a total Root Mean Square Error (RMSE) below 15%.

		Categories					
	Model	Permanent Pasture	Permanent Culture & Arable Land	Forest	Urban/ Artificial	Other	**Total RMSE (%)**
RMSE (%)	M1	0.7	1.1	0.7	0.5	0.7	3.7
	M2	0.8	0.3	0.3	0.1	0.7	2.3
	M3	1.2	1.5	0.3	0.3	0.9	4.2
	M4	1.5	2.1	0.7	0.2	1.2	5.7
	M5	1.0	0.6	0.6	0.4	1.6	4.3
	M6	2.3	0.6	0.9	0.5	0.8	5.2
$\delta_{n_{fold}=1}$		1.3	1.0	0.6	0.3	1.0	4.2

Table 4.4: Aggregated (per model) total average losses measured in terms of RMSE for fold #1.

	Model	Categories					Total RMSE (%)
		Permanent Pasture	Permanent Culture & Arable Land	Forest	Urban/ Artificial	Other	
RMSE (%)	M1	1.8	1.4	0.3	0.7	0.2	4.4
	M2	0.3	0.8	0.4	0.2	0.9	2.6
	M3	0.7	1.7	0.1	0.5	1.5	4.5
	M4	0.2	0.2	0.4	0.6	0.8	2.2
	M5	1.9	0.7	0.2	0.1	2.6	5.6
	M6	0.4	1.4	0.3	0.2	0.9	3.2
$\boldsymbol{\delta}_{n_{fold}=2}$		0.9	1.0	0.3	0.4	1.2	3.8

Table 4.5: Aggregated (per model) total average losses measured in terms of Root Mean Square Error (RMSE) for fold #2.

Apart from the first fold, the results are good, with a maximum error per category of 1.3%. One critical component of the error vector would be the category Urban/Artificial, because of its low absolute value ($\sim 5\%$) in the data it was important that the error would be the lowest of them all. This fact is substantiated across all folds with an error below 0.4% except, again, on the first fold. Nonetheless, its value on first fold is quite close to being the lowest and to the ones obtained in the other folds: 0.5%.

	Model	Categories					Total RMSE (%)
		Permanent Pasture	Permanent Culture & Arable Land	Forest	Urban/ Artificial	Other	
RMSE (%)	M1	0.8	0.5	0.7	0.5	0.2	2.7
	M2	1.5	1.7	0.5	0.4	0.8	4.9
	M3	1.2	0.5	0.1	0.2	1.0	3.0
	M4	1.8	1.3	1.0	0.4	1.9	6.3
	M5	0.6	0.4	0.6	0.4	1.2	3.2
	M6	0.3	0.6	0.7	0.3	1.3	3.2
$\boldsymbol{\delta}_{n_{fold}=3}$		1.0	0.8	0.6	0.4	1.0	3.8

Table 4.6: Aggregated (per model) total average losses measured in terms of Root Mean Square Error (RMSE) for fold #3.

Ultimately, the attained error vector of equation 3.7 was set at

$$\boldsymbol{\delta}_{LULC} = [1.4, 1.7, 0.5, 0.4, 1.2]\ (\%)\ . \tag{4.1}$$

which is expected to prevail up to a 7-year long prediction if the evolution of statistical dynamics of the input variables do not drastically change in the future. This corresponds to the temporal length established during the batching process. In other words, the model was explicitly trained with a 7 time-step horizon

to generate sequences with that length in which the last instance (seventh time-step) represents a forecast. Also, let us remember that the forecasting exercises and respective errors estimation is performed on data non-consecutive to the data used for training.Therefore, this value of 7 years would in principle represent the minimum admissible validity for the error of equation 4.1 to be considered. Adding the number of batches from the testing set, 5, the maximum admissible validity is arguably 12 years. The achieved magnitude of the errors is at par with the measurements' errors reported in [81].

To extend the error validity, the whole process of performance estimation would need to be carried out every year with the arrival of new data. The results from the first fold and from the causal convolutional model were included, so the error is expected to be majorated. The prediction for earlier steps will usually be more accurate than the predictions for later time-steps, further than that errors might accumulate, thus the error bars on the plots may be included up to the 7^{th} year of forecasting.

In figure 4.1 it is depicted, along with the true values, the predicted values by each model for the penultimate fold of data. It is noticeable that the results from the causal convolutional model across all folds (see remaining figures B.1, B.2, and B.3) are relatively constant throughout the time, this potentially means that the model did not uncover hidden patterns, hence outputting almost static values. This endorses the suspicion of either the model being inadequate for our already scarce data set or a better implementation is required. Let us keep in mind that for the time period of 5 years (which represents the number of batches within the test set), there is an understandable slow variation of the distribution of the land use.

In terms of the specificities of each model, all the hyperparameters that aggregated contributed to the above results were compiled in tables B.3, B.4, B.5, B.6, B.7, and B.8, which can be found in the appendix B. Without going too much into detail due to lack of space, some pertinent remarks can be made. First, in regards to the first fold, some models, namely the last two, presented low diversification in terms of the chosen set of hyperparameters. This effect was expected to prevail across most models in the first set due to the fact that with few data the cardinality of performant models is expected to drop, hence the search is more likely to converge to a local minimum, possibly in a narrow region. When that happens, and under an environment of low exploration, the Bayesian algorithm may get stuck on that minimum and provide similar sets of hyperparameters. In the subsequent folds, more data was available, therefore there was more diversity of sets of hyperparameters in the search space that could generate equally good models. This can be thought of as a plateau where in that region of minima the Bayesian algorithm will do some exploitation and generated dissimilar sets of hyperparameters that would score similarly. Next, the networks were given the possibility of having an either more deep or shallow and broad structure. The latter possibility was favoured, with most of the models not fully using the maximum admissible depth. Also, the SGD optimiser was unequivocally preferred over the remaining optimisers across all architectures. Some interesting patterns can too be addressed when inspecting which architecture used which kernel initialiser and activation function, however, such analysis might be out of the scope of this thesis.

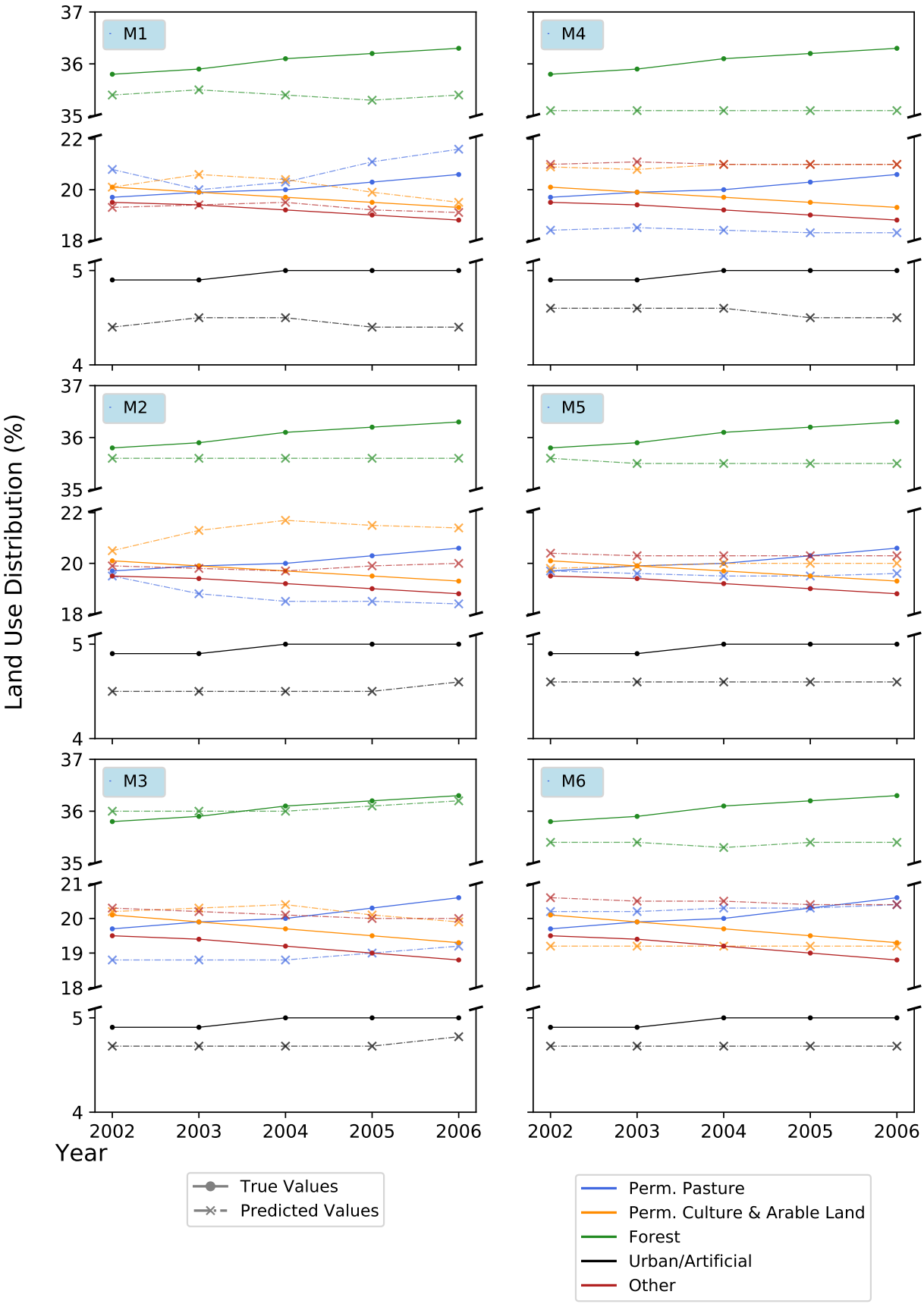


Figure 4.1: Comparison between true and predicted values by each model *M* for fold #3.

4.1.2 Greenhouse Gases

For the application of the previous framework to the data regarding emissions, the error/predictive performance analysis was repeated. Interestingly, the acceptance rate of sub-models, within the criteria set in section 3.3.3, decreased appreciably.

Earlier, from tables 4.1, 4.3, B.1, and B.2 the acceptance rate was set at 93%, with 67 measures of the error that culminated in the error vector presented in equation 4.1. Here, overall the Bayesian search denoted much more difficulties in finding appropriate sets of hyperparameters that would result in well fitted models to the data, with just 50% of the sub-models being considered as valid. The most notorious case is the one respecting the causal convolutional architecture. For that type of network, only two sub-models (out of a universe of possible 12) fell under the 15% threshold for all data folds. Low performance issues relating to the convolutional architecture were found as well in section 4.1.1. Consequently, that architecture's results were rejected and no forecast was attempted with architecture #4. The best performant models with this data across all data folds were the LSTM, GRU, and hybrid convolutional 1D-LSTM networks.

These results might be an indicator of either insufficient covariates to fully explain GHG emissions, inadequate lagging, or that in more recent periods the statistical properties of data regarding GHG emissions have greatly changed, with those new dynamics suggesting a transition to different emitting patterns than those of the past, and consequently complexing the learning process. The last argument could be also supported on the fact that the training scores achieved on the validation data do not differ appreciably from the scores previously displayed in section 4.1.1. Nonetheless, 34 measures of the error contributed to the final error estimation from equation 4.2, divided into emissions from agriculture and energy sectors, respectively, legitimating the joined forecasting capability of all models, excluding the left-out convolutional architecture. The error vector,

$$\boldsymbol{\delta}_{GHG} = [0.2, 4.2] \text{ (Mt } CO_2\text{e)} , \tag{4.2}$$

relatively to the values registered in the year of 2016 (see table A.2), represents a deviation of 3.2% and 8.3%, respectively.

4.2 Model Forecasting

After the new set of models were trained and validated on the last fold, each model's last output was continuously fed back into itself at each step, along with covariates from scenarios P and O, and forecasts were made conditioned on previous, unrecorded, data, as further explained in section 3.3.5.

4.2.1 Land Use Distribution

Figure 4.2 translates the forecast performed under the conditions from scenario P. Note that although we are using shares of land use, the sum of the forecasted classes is a conserved quantity, *i.e.*, it is

always equal to 1, which corresponds to the total area of the country.

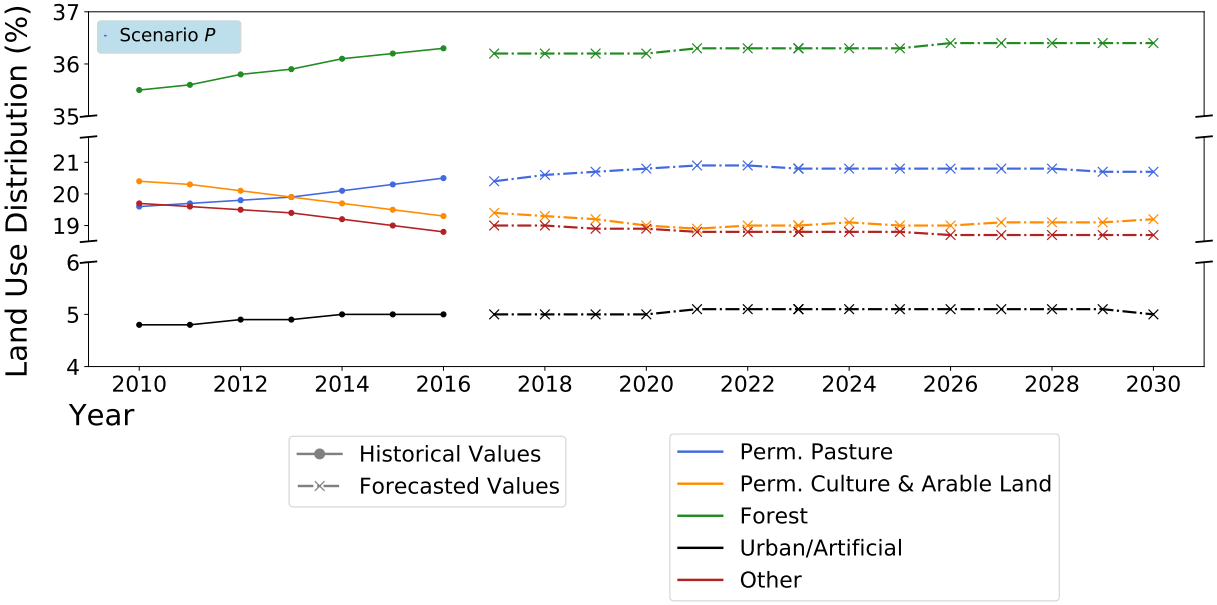


Figure 4.2: Land use distribution forecast for the pessimistic scenario.

A generalised stagnation of the evolution of land use distribution is evident, as we can see in table 4.7. Despite the good fit, the error from equation 4.1 exceeds largely the variation scale of the variables in this scenario. Hence, error bars were not added to the slowly varying trends, focusing the analysis on the qualitative side.

It is remarkable that even though the model is very well fitted to the data, its estimated (already low) uncertainty is greater than the expected variations in the variables for an extreme plausible scenario. This fact confirms the inherent complexity and uncertainty in this forecasting exercises, despite the rigorous treatment. This also hints that simpler methods should be subject to stronger scrutiny regarding the uncertainty of the forecasts, which is frequently omitted from reports and scientific publications.

	Categories (%)				
Year	Permanent Pasture	Permanent Culture & Arable Land	Forest	Urban/ Artificial	Other
2017	20.4 ± 0.7	19.4 ± 0.9	36.2 ± 0.3	5.0 ± 0.2	19.0 ± 0.6
2030	20.7 ± 0.7	19.2 ± 0.9	36.4 ± 0.3	5.0 ± 0.2	18.7 ± 0.6

Table 4.7: LULC's first and last numerical results from the forecasts for scenario P

In our simulated data, the GDP decreases down to lower levels than those of the MEET project report [42] for the pessimistic scenario. The effect is due to the fact that in 2020 the pandemic introduced some severe volatility in the Portuguese economy, rendering an abrupt divergence from historical values in terms of GDP. As the utilised GDP value for the year of 2020 got closer to the worst economic case foreseen for 2030, it may be understandable that no major dynamics were reflected in the land use, partially validating the argument used in the problem framing (see section 3.1) that economic resources

were required to consummate significant modifications in land and infrastructures.

In turn, the GDP values from the optimistic scenario did not achieve levels as high as those from the MEET project report [42], due to the lower starting point. In figure 4.3, it is noticeable that towards the year of 2030 some trends are already notorious. Nonetheless, this economic lower starting point may explain the slow departure from stability, and prevent significant land use changes.

The differences between the optimistic scenario and the present are therefore slim. Firstly, there is a slight increase in the forest area. Secondly, there are non negligible changes in both permanent pasture area, and permanent culture and arable land. While the former yielded a higher area, the later steadily started decreasing. Thus, there is evidence pointing towards the reduction of cropland (permanent culture and arable land) area with economic acceleration.

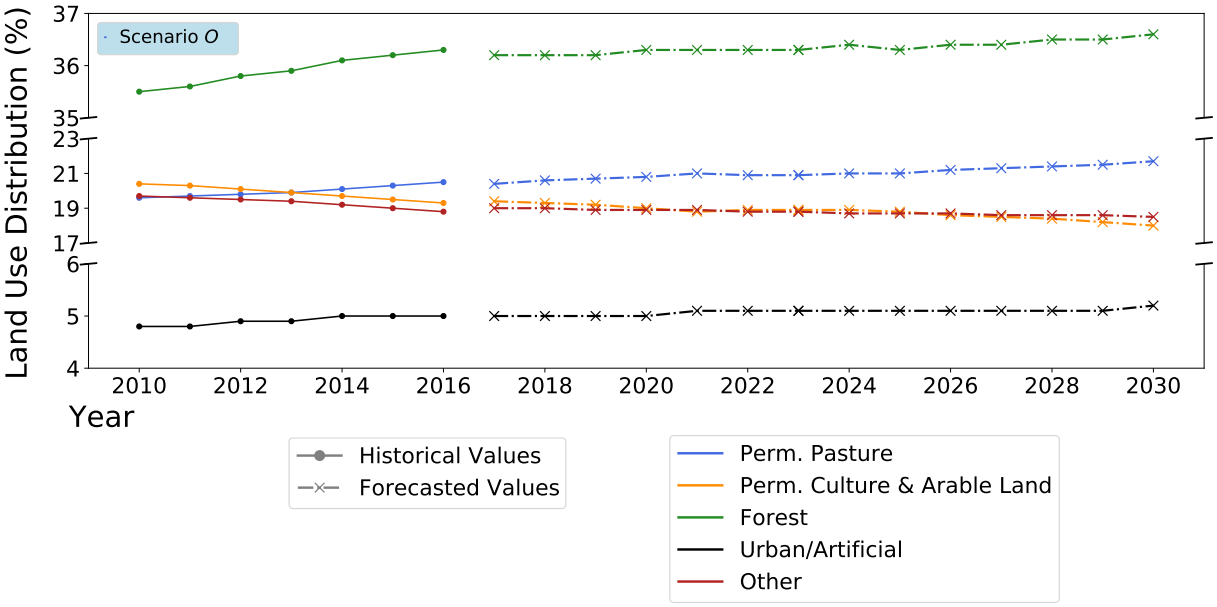


Figure 4.3: Land use distribution forecast for the optimistic scenario.

	Categories (%)				
Year	Permanent Pasture	Permanent Culture & Arable Land	Forest	Urban/ Artificial	Other
2017	20.4 ± 0.7	19.4 ± 0.9	36.2 ± 0.3	5.0 ± 0.2	19.0 ± 0.6
2030	21.7 ± 0.7	18.0 ± 0.9	36.6 ± 0.3	5.2 ± 0.2	18.5 ± 0.6

Table 4.8: LULC's first and last numerical results from the forecasts for scenario O

4.2.1.1 Sensitivity Analysis

In regards to the effects the pandemic introduced in the final results, the most striking difference was that in the economic optimistic scenario the reduction in cropland land area was much more pronounced ($\approx 1\%$ less area than in the original scenario), as stated in table C.1. That difference was mainly transferred into gains for grassland (permanent pasture) area.

The data sets where GDP varied the most were the major contributors to new dynamics. It is clear that for our model, the coupling between the GDP and the output variables is stronger than with the final exergy. In the case where the GDP growth was accelerated, cropland area sharply decreased, approximately 4%, and forest area, grassland area, and urban/artificial area steadily increased by 3.1%, 1.1%, and 0.4%, respectively, when compared to 2017. The opposite trends were observed when the GDP growth was inversely accelerated (economic contraction).

When it came to the altered final exergy time series, the most significant change occurred in both cropland and grassland areas when the final exergy was set to increase. In that case, compared to 2017, their areas diverged, with a smaller final cropland area (0.8% less) and bigger final grassland area (0.9% more), though those variations are not as abrupt as with the GDP. All the results here expressed were a comparison with the results from the original scenarios, and were plotted in figures C.1, C.2, C.3, C.4, C.5, and C.6.

4.2.2 Greenhouse Gases

In the GHG emissions results, represented in figure 4.4 and summarised in table 4.9, two interesting patterns arose in both scenarios. For the economically pessimistic scenario, the emissions from energy tended to stall, whereas the emissions from agriculture rose, though negligibly. In the optimistic case, emissions from agriculture decreased slightly, while the emissions from energy sharply increased - roughly 10% more GHG emissions than that of scenario P in the forecast of 2030. One might argue that there is a trade-off where an increasingly richer country starts to abandon agricultural land, and it uses activities that consume more energy. This may be due to: a) a structural change in the balance between economic sectors and labour shifts away from the primary sector due to better opportunities elsewhere, replacing domestic production with trade, or b) increasing yields due to more machinery and energy use for farming, with decreasing land demand. Oppositely, in an increasingly poorer country that operates less high-energy activities, the demand for farmland may be higher. These observations are well supported by historical data for Portugal, presented in A.1.

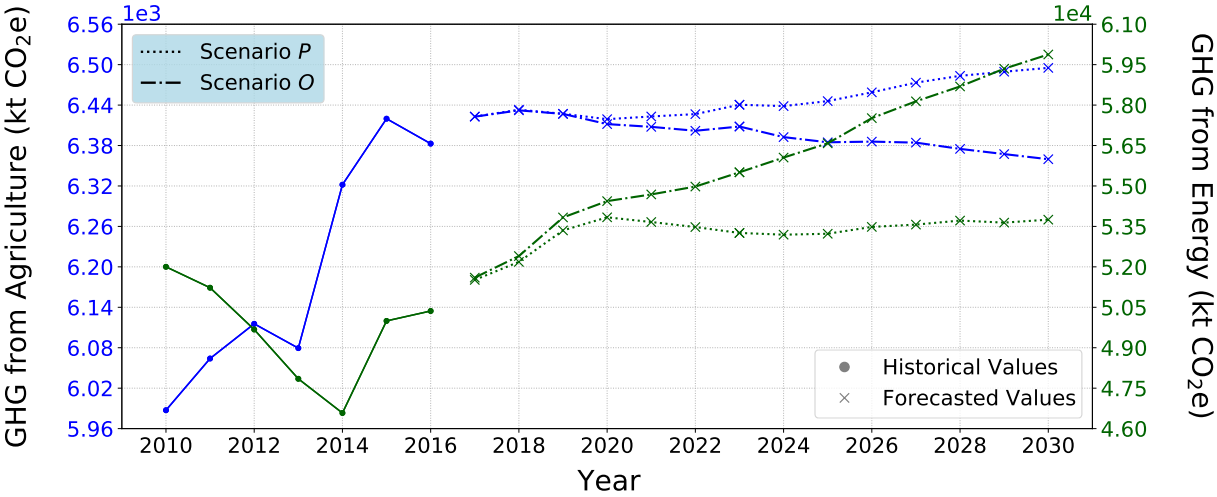


Figure 4.4: GHGs from agriculture and from energy forecasts under both pessimistic and optimistic scenarios.

	Scenario P		Scenario O	
Year	$GHG_{agriculture}$*	GHG_{energy}*	$GHG_{agriculture}$*	GHG_{energy}*
2017	6.4 ± 0.1	51.5 ± 2.1	6.4 ± 0.1	51.6 ± 2.1
2030	6.5 ± 0.1	53.8 ± 2.1	6.4 ± 0.1	59.9 ± 2.1

* All values measured in terms of metric megaton of CO_2 equivalent (Mt CO_2e)

Table 4.9: GHG's first and last numerical results from the forecasts for both scenarios P and O

4.2.2.1 Sensitivity Analysis

Here, it was concluded that in the coronavirus-free built scenarios, emissions from the energy sector increased, with final greater values in 2030 than those of the original scenarios. The biggest fluctuation was observed for the emissions from energy in the optimistic scenario. Overall, it was evidenced that COVID-19 might mitigate the increase in emissions of CO_2e to the atmosphere up to 2030, under scenarios P and O.

In terms of coupling, it was observed that total final exergy data was responsible for introducing major variations, having 2017 as reference. On the case where final exergy grew year-over-year 5%, emissions from agriculture strongly decreased, close to 4%, and emissions from energy sharply increased, 30%. For the decreasing final exergy time series, the new exergy dynamics induced greater emissions from agriculture ($\approx 4\%$) and a steep decrease in emissions from energy ($\approx 12\%$). The GDP time series did not contributed to significant modifications, when compared to the original forecast from figure 4.4. The results suggest that, operating at the business as usual mode, where no additional policies are introduced, generating more wealth and consuming more energy comes at the expense of greater GHG emissions [40, 42]. All results were tabled in C.2, C.3, and C.4, along with figures C.7, C.8, and C.9.

5

Conclusions

5.1 Findings and Achievements

In this work, we set out to: 1) explain the relationship between the distribution of land use in terms of final exergy in the agricultural sector and economic growth in terms of GDP, and 2) explain the emissions of GHG for both agricultural and energy sectors in terms of total final exergy used in Portugal, economic growth in terms of GDP, and distribution of land use. This process involved not only the study and subsequent programming of the whole procedure, but also a thorough examination of its capabilities and shortcomings.

The major findings were that: 1) there is a strong coupling between GDP and the dynamics of land use distribution, as well as between total final exergy and GHG emissions from both agriculture and energy sectors, 2) when a country gets richer (higher GDP) it tends to decrease its cropland area while increasing the intensity of activities that consume more energy, emitting GHG at higher levels, 3) in ever increasing impoverished countries cropland area grows and energy intense activities are reduced, leading to overall lower levels of GHG emissions, 4) the coronavirus pandemic might decrease cropland area reduction in a forecasted positive economic scenario, and 5) the coronavirus pandemic might have induced a slight deceleration for emissions related to agriculture and energy sectors up to 2030.

It was also evident that the pre-COVID-19 scenarios, from collaboration between companies' representatives and government agents, were not sufficiently aggressive (within a plausible pathway) when combined with data that took into account the economic downturn. These new economic conditions hindered the evolution of land use in both studied scenarios, being required a sensitivity analysis to further explore the interplay between all factors.

In the conditions of the contemplated scenarios, the forecasted evolution for 2030 of land use distribution and GHG emissions was tenuous when compared with the estimated uncertainty. Despite the very well fitted models and rigorous uncertainty estimation, the compound evolution of the plausible scenarios stayed always within the uncertainty, complexing the analysis. This conclusion casts doubts over simpler methods. Future scenario-making exercises should always include a formal uncertainty analysis in order to transparently report the level of confidence of forecasts.

Methodologically, a new performance estimation method for time-series was developed within an ensemble of models environment. This method involved measures in two dimensions: 1) preliminary loss estimate on validation data, by converging to the ideal architecture's set up; and 2) final loss estimate, by repeating the previous measurement on new data and on the best models. Hence, we got control over two potential overfitting situations, being them the overfitting to the training data, and the implicit overfitting of the model to the validation data when selecting the best model's configurations. As we are forcing the model to predict several time-steps ahead and estimating the error on that prediction, 7 years would in principle represent the minimum admissible validity for the uncertainty to be considered. Adding the number of batches from the testing set, 5, the maximum admissible validity is arguably 12 years.

The insufficiency of data motivated the creation of a new batching method. It is usually advisable to use hard frontiers between the training, validation, and testing data sets. However, in time series analysis, when using a sequence-to-sequence configured network for predicting m time-steps ahead, the error is only accounted for that last prediction. Therefore, overlapping data sets were constructed in a way that no lookahead was introduced, maximising the batches cardinality.

The originality of our work came from the fact that in the literature this is the first study where there is a compounded approach to the problems of forecasting land use change and associated emissions, by coupling both modelling exercises into an integrated framework, hence strengthening the predictive power. Commonly, land use changes and GHG emissions have been individually forecasted with ML, by directly inputting either land use distribution data and related data or GHG emissions data and related data to the models, respectively [82, 83].

5.2 Limitations and Hypotheses

The foremost limiting aspect came from the scarcity of historical data regarding land use distribution. Even for existing data, there was considerable uncertainty associated with it. Historical Aerial Photographs (HAP) are the oldest form of remotely sensed data, yielding a spectrum of valuable historical information on vegetation and land cover. Aerial photographs started to be produced from as early as the 1930s, however, the orthorectification of those photographs is a complex and difficult process since historical ground control information, calibration certificates, and precise or even basic knowledge of camera parameters are often not available [84], compromising the standardisation process, and resulting in the found uncertainty. There was the possibility of searching through documents from the History field attempting to gather more data, however, that would bring the need of making the sources compatible, by harmonising assumptions from each source, a task out of the scope of this thesis.

The second most influential limitation resulted from the fact that the time scale of this study contributed to shorter time series. Given the nature of the models employed, longer series are preferable in order to increase the likelihood of patterns being revealed.

On the technical side, due to training data constraints, we have only used stateless RNNs, where at each training iteration the model starts with a hidden state full of zeros, then it updates the state

at each time-step, and after the last time-step it throws it away. It is possible to preserve this final state after processing one training batch and use it as the initial state for the next training batch. By doing so, the model could learn longer long-term patterns even though it only backpropagates through short sequences. This defines what it is called as stateful RNNs and they make sense in a context of abundant data, because the condition for this model to be used is that each input sequence in a batch starts exactly where the corresponding sequence in the previous batch left off, therefore it is required to use sequential and non-overlapping input sequences. Also, shuffling batches for stateless RNNs is a possible approach[1].

Another major constraint was that the training times were quite high, with an estimation of 30h per thread per fold of data per model. With 24 threads available, the running trials had to be capped, with roughly 1-5% of the search space studied during the Bayesian optimisation. As a consequence, we had to compromise in terms of which covariates would be included. This constraint left out of the study variables such as carbon sequestration, water consumption, Human Appropriation of Net Primary Productivity (HANPP), rice production emissions, use of nitrogen and phosphorus fertilisers (production in the country and importation statistics), fossil fuel and renewables, and cropland, forest and animal products output. The accommodation of a broader set of covariates was expected to enhance the complexity of the model while enabling the construction of a more fundamental model or set of models.

In terms of hypotheses, with support on studies in the area, the two hypotheses conjectured were that: 1) Land Use and Land Cover (LULC) are, besides to its own lagged values, tightly related to both energy usage and economic development, and 2) GHG emissions can be defined in terms of its own lagged values, land use and land cover, energy usage, and economic development. Also, in autoregressive models there is the assumption that the observations at previous time-steps are useful to predict the value at the next time-step are made, *i.e.*, there is a correlation between variables. It was also hypothetised that GDP as an economic indicator would reliably help reflecting land use changes as well as GHG emissions for the agriculture and energy sectors. This indicator has been criticised for ignoring the depreciation of assets, non-market economy, damages to the environment, and for being a poor proxy for societal well-being [85, 86].

5.3 Future Work

Due to the intricate nature of the dynamics of models in the field of nature-related processes, this work was carried out based on empirical evidence in what regards the relationship between the input features and the targets, meaning that it was assumed that the input variables to the model contained useful information that would allow us to predict the target variables based on those features. In the future, a thorough study on statistical relationships, such as correlation, autocorrelation, spurious correlations, among other statistics, shall be done for a broader set of covariates. Specific data-analysis techniques on multivariate time series, such as Principal Component Analysis (PCA), similarity searches, feature-

[1] Jason Brownlee, a reference in the field of machine learning, does a thorough study about the different combinations (stateless vs stateful vs shuffling) and respective results in here: https://machinelearningmastery.com/stateful-stateless-lstm-time-series-forecasting-python/.

subset-selection, and clustering are proposed [87]. Even though correlation does not imply a causal relationship, and non-linear causal relationships are difficult to demonstrate, such tests might lead to new directions on which features are the most relevant and how they are temporally coupled.

Data reconstruction/imputation and data preprocessing could be subject to different approaches. For the first, we have either Markov Chain Monte Carlo (MCMC) method or Cellular Automata-Markov Chain Model (CAMCM) for multiple imputation, as well as higher-order interpolation methods. For the second part, the adopted strategy involved a range scaling comprehended between 0 and 1, *i.e.*, a normalisation technique. The standardisation technique and numerical inputs ranging within a distinct range should be explored.

Besides, there is a huge performance dependency on how the splitting and batching processes are done, as well as on how the time series are lagged. By plotting the model's errors on the validation set across time it would be assessable if the model performs better on the first part of the validation set than on the last part, if so, then it would be advisable to shorten the window size, training the model on a shorter time span. However, to automate the process the ideal would be to apply a second Bayesian optimisation for the the window size, splitting point, and lagging period hyperparameters. Additionally, a non-default set of exploitation/exploration parameters for both Bayesian optimisations is suggested.

In this work, we evaluated models' loss according to two distinct metrics: RMSE and MAE. In the future, the inclusion of the metric explained by equation 2.3 could be an option to consider. Likewise, more sophisticated overfitting-preventing/regularisation techniques present a great potential. In the list, one could mention the L1 and L2 regularisation, dropout, recurrent dropout, and Monte Carlo dropout. Architecture-wise, alterations that could render better results are the introduction of customised output layers, different activation functions in different layers, the use of different number of filters and kernel size within convolutional layers, and trying to use of a vanilla 1D-convolutional model. Additionally, online computer capacity could be purchased and GPUs could be used to increase the number of trialled models in the case we have larger batches. Finally, having a method to estimate forecast errors for longer periods would come handy.

In terms of applications, the methods here presented have also been extended to the field of physics, by helping detecting outliers in data from high-energy physics experiments, identifying new phases of matter, and, for instance, by founding astronomical objects [88]. In the context of the Roteiro para a Neutralidade Carbónica (RNC) for 2050, it would be interesting to apply the model here developed to ascertain the results from the scenarios employed. There, the GDP, energy efficiency, and GHG emissions from agriculture were forecasted to increase up to 2030. However, the modelling was performed sector-wise, not accounting for mutual influence between sectors [89]. Increases of energy efficiency lead to an increase in the GDP, as seen in section 2.1.3, which in turn imply an increase of total GHG emissions. Yet, a GDP increase is correlated with reduced emissions from agriculture due to land use change, diverging from the results of the RNC. To conclude, with our model there is an opportunity of interconnecting the forecasted sectors and provide more cohesion to the analysis. The range of applications is arguably limitless.

Bibliography

[1] V. Smil. *Energy and Civilization: A History*. The MIT Press, rev - revised, 2 edition, 2017. ISBN 9780262035774. URL http://www.jstor.org/stable/j.ctt1pwt6jj.

[2] K. C. Seto and N. Ramankutty. Hidden linkages between urbanization and food systems. *Science*, 352(6288):943–945, 2016. ISSN 0036-8075. doi: 10.1126/science.aaf7439. URL https://science.sciencemag.org/content/352/6288/943.

[3] A. K. Styring et al. Isotope evidence for agricultural extensification reveals how the world's first cities were fed. *Nature Plants*, 3(6):17076, 2017.

[4] G. Algaze et al. Initial social complexity in southwestern asia: the mesopotamian advantage. *Current Anthropology*, 42(2):199–233, 2001.

[5] D. R. Montgomery. Is agriculture eroding civilization's foundation? *GSA TODAY*, 17(10):4, 2007.

[6] D. Q. Fuller and C. J. Stevens. Between domestication and civilization: the role of agriculture and arboriculture in the emergence of the first urban societies. *Vegetation history and archaeobotany*, 28(3):263–282, 2019.

[7] The World Bank. Agricultural land (% of land area). https://data.worldbank.org/indicator/AG.LND.AGRI.ZS, last accessed on 19/12/19.

[8] B. Campbell et al. Agriculture production as a major driver of the earth system exceeding planetary boundaries. *Ecology and Society*, 22(4), 2017. doi: 10.5751/ES-09595-220408.

[9] G. M. Karagiannis, M. Cardarilli, Z. I. Turksezer, J. Spinoni, L. Mentaschi, L. Feyen, and E. Krausmann. Climate change and critical infrastructure—storms. *European Union: Luxembourg*, 2019.

[10] F. Antonioli et al. Relative sea-level rise and potential submersion risk for 2100 on 16 coastal plains of the mediterranean sea. *Water*, 12(8):2173, Aug 2020. ISSN 2073-4441. doi: 10.3390/w12082173. URL http://dx.doi.org/10.3390/w12082173.

[11] N. Lin. Tropical cyclones and heatwaves. *Nature Climate Change*, 9(8):579–580, 2019.

[12] S. Perkins-Kirkpatrick and S. Lewis. Increasing trends in regional heatwaves. *Nature communications*, 11(1):1–8, 2020.

[13] G. Forzieri, L. Feyen, S. Russo, M. Vousdoukas, L. Alfieri, S. Outten, M. Migliavacca, A. Bianchi, R. Rojas, and A. Cid. Multi-hazard assessment in europe under climate change. *Climatic Change*, 137, 07 2016. doi: 10.1007/s10584-016-1661-x.

[14] A. W. Griffith and C. J. Gobler. Harmful algal blooms: A climate change co-stressor in marine and freshwater ecosystems. *Harmful Algae*, 91:101590, 2020. ISSN 1568-9883. doi: https://doi.org/10.1016/j.hal.2019.03.008. URL http://www.sciencedirect.com/science/article/pii/S1568988319300344. Climate change and harmful algal blooms.

[15] National Oceanic and Atmospheric Administration (NOAA). Climate Change: Glacier Mass Balance. https://www.climate.gov/news-features/understanding-climate/climate-change-glacier-mass-balance, last accessed on 11/12/20.

[16] C. Román-Palacios and J. J. Wiens. Recent responses to climate change reveal the drivers of species extinction and survival. *Proceedings of the National Academy of Sciences*, 117(8):4211–4217, 2020.

[17] N. A. Graham, J. P. Robinson, S. E. Smith, R. Govinden, G. Gendron, and S. K. Wilson. Changing role of coral reef marine reserves in a warming climate. *Nature Communications*, 11(1):1–8, 2020.

[18] W. J. Ripple et al. World Scientists' Warning of a Climate Emergency. *BioScience*, 11 2019. ISSN 0006-3568. doi: 10.1093/biosci/biz088. URL https://doi.org/10.1093/biosci/biz088.

[19] N. Stern et al. *Stern Review: The Economics of Climate Change*. London: HM Treasury, 2006.

[20] J. G. Shepherd et al. The future of phosphorus in our hands. *Nutrient Cycling in Agroecosystems*, 104(3):281–287, 4 2016. ISSN 1573-0867. doi: 10.1007/s10705-015-9742-1. URL https://doi.org/10.1007/s10705-015-9742-1.

[21] M. Gross. We need to talk about nitrogen. *Current Biology*, 22(1):R1 – R4, 2012. ISSN 0960-9822. doi: https://doi.org/10.1016/j.cub.2011.12.033. URL http://www.sciencedirect.com/science/article/pii/S0960982211014461.

[22] J. Galloway et al. Transformation of the nitrogen cycle: Recent trends, questions, and potential solutions. *Science (New York, N.Y.)*, 320:889–92, 06 2008. doi: 10.1126/science.1136674.

[23] B. L. Bodirsky et al. Reactive nitrogen requirements to feed the world in 2050 and potential to mitigate nitrogen pollution. *Nature Communications*, 5(1):3858, 2014. doi: 10.1038/ncomms4858. URL https://app.dimensions.ai/details/publication/pub.1017892767andhttps://www.nature.com/articles/ncomms4858.pdf.

[24] K. Carlson et al. Greenhouse gas emissions intensity of global croplands. *Nature Climate Change*, 7, 11 2016. doi: 10.1038/NCLIMATE3158.

[25] L. F. Razon. Life cycle analysis of an alternative to the haber-bosch process: Non-renewable energy usage and global warming potential of liquid ammonia from cyanobacteria. *Environmental*

Progress & Sustainable Energy, 33(2):618–624, 2014. doi: 10.1002/ep.11817. URL https://aiche.onlinelibrary.wiley.com/doi/abs/10.1002/ep.11817.

[26] V. H. Smith et al. Eutrophication science: where do we go from here? *Trends in Ecology & Evolution*, 24(4):201 – 207, 2009. ISSN 0169-5347. doi: https://doi.org/10.1016/j.tree.2008.11.009. URL http://www.sciencedirect.com/science/article/pii/S016953470900041X.

[27] M. F. Chislock, E. Doster, R. A. Zitomer, and A. E. Wilson. Eutrophication: causes, consequences, and controls in aquatic ecosystems. *Nature Education Knowledge*, 4(4):10, 2013.

[28] J. Rockström et al. Sustainable intensification of agriculture for human prosperity and global sustainability. *Ambio*, 46(1):4–17, 2 2017. ISSN 1654-7209. doi: 10.1007/s13280-016-0793-6. URL https://doi.org/10.1007/s13280-016-0793-6.

[29] H. J. van Grinsven, J. W. Erisman, W. de Vries, H. Westhoek, and L. Lassaletta. Potential of extensification of european and dutch agriculture for a more sustainable food system focusing on nitrogen and livestock. In *Just Enough Nitrogen*, pages 83–98. Springer, 2020.

[30] O. Bertolami and F. Francisco. A physical framework for the earth system, anthropocene equation and the great acceleration. *Global and Planetary Change*, 169:66 – 69, 2018. ISSN 0921-8181. doi: https://doi.org/10.1016/j.gloplacha.2018.07.006. URL http://www.sciencedirect.com/science/article/pii/S0921818118302182.

[31] G. Hoogenboom et al. Chapter 13 - biophysical agricultural assessment and management models for developing countries. In C. A. Hall, C. L. Perez, and G. Leclerc, editors, *Quantifying Sustainable Development*, pages 403 – 422. Academic Press, San Diego, 2000. ISBN 978-0-12-318860-1. doi: https://doi.org/10.1016/B978-012318860-1/50020-4. URL http://www.sciencedirect.com/science/article/pii/B9780123188601500204.

[32] N. Zhu et al. Deep learning for smart agriculture: Concepts, tools, applications, and opportunities. *International Journal of Agricultural and Biological Engineering*, 11, 07 2018. doi: 10.25165/j.ijabe.20181104.4475.

[33] M. E. Qureshi et al. A biophysical and economic model of agriculture and water in the murray-darling basin, australia. *Environmental Modelling & Software*, 41:98 – 106, 2013. ISSN 1364-8152. doi: https://doi.org/10.1016/j.envsoft.2012.11.007. URL http://www.sciencedirect.com/science/article/pii/S1364815212002782.

[34] R. Alves. Aplicação de modelos de redes neuronais para previsão de consumos de energia [bachelor's thesis]. *Lisbon, Portugal: Instituto Superior Técnico, University of Lisbon*, 2016.

[35] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to data mining*. Pearson Education India, 2016.

[36] A. Nielsen. *Practical Time Series Analysis*. O'Reilly Media, Inc., 1 edition.

[37] K. G. Liakos et al. Machine learning in agriculture: A review. *Sensors*, 18(8), 2018. ISSN 1424-8220. doi: 10.3390/s18082674. URL https://www.mdpi.com/1424-8220/18/8/2674.

[38] A. Kamilaris and F. X. Prenafeta-Boldú. Deep learning in agriculture: A survey. *Computers and Electronics in Agriculture*, 147:70 – 90, 2018. ISSN 0168-1699. doi: https://doi.org/10.1016/j.compag.2018.02.016. URL http://www.sciencedirect.com/science/article/pii/S0168169917308803.

[39] R. Manso et al. The way forward in quantifying extended exergy efficiency. *Energies*, 11(10), 2018. ISSN 1996-1073. doi: 10.3390/en11102522. URL https://www.mdpi.com/1996-1073/11/10/2522.

[40] A. Serrenho. Useful work as an energy end-use accounting method: historical and economic transitions and european patterns [phd dissertation]. *Lisbon, Portugal: Instituto Superior Técnico, University of Lisbon*, 2013.

[41] A. C. Serrenho, B. Warr, T. Sousa, R. U. Ayres, and T. Domingos. Structure and dynamics of useful work along the agriculture-industry-services transition: Portugal from 1856 to 2009. *Structural Change and Economic Dynamics*, 36:1 – 21, 2016. ISSN 0954-349X. doi: https://doi.org/10.1016/j.strueco.2015.10.004. URL http://www.sciencedirect.com/science/article/pii/S0954349X15000508.

[42] A. Alvarenga. Towards a carbon neutral economy how is portugal going to create employment and grow? – technical report, 2017.

[43] C. Meiklejohn. The food crisis in prehistory: Over-population and the origins of agriculture. by mark nathan cohen. yale university press, new haven, connecticut. 1977. × + 341 pp., bibliography, index. $15.00 (cloth). *American Journal of Physical Anthropology*, 49(2):286–287, 1978. doi: https://doi.org/10.1002/ajpa.1330490221. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/ajpa.1330490221.

[44] R. E. Sonntag, C. Borgnakke, and G. J. Van Wylen. *Fundamentals of thermodynamics*, volume 6. 1998.

[45] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[46] A. Géron. *Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow*. O'Reilly Media, Inc., 2 edition.

[47] H. Markram. The blue brain project. *Nature Reviews Neuroscience*, 7(2):153–160, 2006.

[48] K. Amunts, C. Ebell, J. Muller, M. Telefont, A. Knoll, and T. Lippert. The human brain project: creating a european research infrastructure to decode the human brain. *Neuron*, 92(3):574–581, 2016.

[49] S. Sharma. What the hell is perceptron? https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53, last accessed on 31/12/19.

[50] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[51] R. Lopez and E. Oñate. A variational formulation for the multilayer perceptron. In *International Conference on Artificial Neural Networks*, pages 159–168. Springer, 2006.

[52] X. Glorot and Bengio. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*, volume 9, pages 249–256, 2010.

[53] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus), 2015.

[54] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter. Self-normalizing neural networks, 2017.

[55] D. Jones. A taxonomy of global optimization methods based on response surfaces. *J. of Global Optimization*, 21:345–383, 12 2001. doi: 10.1023/A:1012771025575.

[56] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. 2012.

[57] E. Brochu, V. M. Cora, and N. de Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. 2010.

[58] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014. https://arxiv.org/abs/1412.3555.

[59] R. Karim. Animated rnn, lstm and gru. https://towardsdatascience.com/animated-rnn-lstm-and-gru-ef124d06cf45, last accessed on 31/12/19.

[60] S. Khandelwal, B. Lecouteux, and L. Besacier. COMPARING GRU AND LSTM FOR AUTOMATIC SPEECH RECOGNITION. Research report, LIG, Jan. 2016. URL https://hal.archives-ouvertes.fr/hal-01633254.

[61] S. Gao, Y. Huang, S. Zhang, J. Han, G. Wang, M. Zhang, and Q. Lin. Short-term runoff prediction with gru and lstm networks without requiring time step optimization during sample generation. *Journal of Hydrology*, 589:125188, 2020. ISSN 0022-1694. doi: https://doi.org/10.1016/j.jhydrol.2020.125188. URL http://www.sciencedirect.com/science/article/pii/S002216942030648X.

[62] N. Gruber and A. Jockisch. Are gru cells more specific and lstm cells more sensitive in motive classification of text? *Frontiers in Artificial Intelligence*, 3:40, 2020. ISSN 2624-8212. doi: 10.3389/frai.2020.00040. URL https://www.frontiersin.org/article/10.3389/frai.2020.00040.

[63] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio, 2016.

[64] A. C. Serrenho et al. Decomposition of useful work intensity: The eu (european union) - 15 countries from 1960 to 2009. *Energy*, 76:704 – 715, 2014. ISSN 0360-5442. doi: https://doi.org/10.1016/j.energy.2014.08.068. URL http://www.sciencedirect.com/science/article/pii/S0360544214010214.

[65] Organisation for Economic Co-operation and Development (OECD). OECD-FAO Agricultural Outlook. https://www.oecd-ilibrary.org/agriculture-and-food/oecd-fao-agricultural-outlook-2019-2028_agr_outlook-2019-en, last accessed on 11/01/20.

[66] C. A. Guerra et al. Policy impacts on regulating ecosystem services: looking at the implications of 60 years of landscape change on soil erosion prevention in a mediterranean silvo-pastoral system. *Landscape ecology*, 31(2):271–290, 2016. doi: 10.1007/s10980-015-0241-1.

[67] N. Jones, J. de Graaff, I. Rodrigo, and F. Duarte. Historical review of land use changes in portugal (before and after eu integration in 1986) and their implications for land degradation and conservation, with a focus on centro and alentejo regions. *Applied Geography*, 31(3):1036 – 1048, 2011. ISSN 0143-6228. doi: https://doi.org/10.1016/j.apgeog.2011.01.024. URL http://www.sciencedirect.com/science/article/pii/S0143622811000385.

[68] European Space Agency - Climate Change Initiative. Land cover classification gridded maps from 1992 to present derived from satellite observations. https://cds.climate.copernicus.eu/cdsapp#!/dataset/satellite-land-cover?tab=overview, last accessed on 27/11/19.

[69] C. Fonte, L. See, M. Lesiv, and S. Fritz. A preliminary quality analysis of the climate change initiative land cover products for continental portugal. 07 2019. doi: 10.5194/isprs-archives-XLII-2-W13-1213-2019.

[70] A. Nunes. Uso do solo em portugal continental: aspectos gerais da sua evolução. 2002. doi: http://dx.doi.org/10.14195/0871-1623_23_8.

[71] ICNF. Relatório final da proposta técnica de plano nacional de defesa da floresta contra incêndios - perspetiva histórica sobre a floresta portuguesa e a sua defesa contra incêndios. pages 4,10.

[72] B. M. Meneses. *A caraterização do uso e ocupação do solo de Portugal Continental*, pages 1–5. 01 2014.

[73] International Energy Agency (IEA). World energy balances and statistics, . https://www.iea.org/statistics/balances/, last accessed on 2/12/20.

[74] European Comission. Macro-economic database AMECO. https://ec.europa.eu/info/business-economy-euro/indicators-statistics/economic-databases/macro-economic-database-ameco/download-annual-data-set-macro-economic-database-ameco_en, last accessed on 8/12/20.

[75] Food and Agriculture Organization of the United Nations (FAO). Emissions Shares. http://www.fao.org/faostat/en/#data/EM, last accessed on 1/12/20.

[76] International Energy Agency (IEA). CO_2 emissions statistics, . https://www.iea.org/subscribe-to-data-services/co2-emissions-statistics, last accessed on 2/12/20.

[77] International Panel on Climate Change (IPCC). 2006 IPCC Guidelines for National Greenhouse Gas Inventories - Energy. https://www.ipcc-nggip.iges.or.jp/public/2006gl/vol2.html, last accessed on 2/12/20.

[78] C. Laurent, G. Pereyra, P. Brakel, Y. Zhang, and Y. Bengio. Batch normalized recurrent neural networks, 2015.

[79] R. Bala, R. P. Singh, et al. Financial and non-stationary time series forecasting using lstm recurrent neural network for short and long horizon. In *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–7. IEEE, 2019.

[80] V. Cerqueira, L. Torgo, J. Smailovic, and I. Mozetic. A comparative study of performance estimation methods for time series forecasting. pages 529–538, 10 2017. doi: 10.1109/DSAA.2017.7.

[81] ICNF. 6º inventário florestal nacional. page 1, 2019.

[82] M. H. Saputra and H. S. Lee. Prediction of land use and land cover changes for north sumatra, indonesia, using an artificial-neural-network-based cellular automaton. *Sustainability*, 11(11):3024, 2019.

[83] A. Hamrani, A. Akbarzadeh, and C. A. Madramootoo. Machine learning for predicting greenhouse gas emissions from agricultural soils. *Science of The Total Environment*, 741:140338, 2020.

[84] A. T. Pinto, J. Gonçalves, P. Beja, and J. Pradinho. From archived historical aerial imagery to informative orthophotos: A framework for retrieving the past in long-term socio-ecological research. *Remote Sensing*, 06 2019. doi: 10.3390/rs11111388.

[85] S. Stjepanović, D. Tomić, and M. Škare. A new approach to measuring green gdp: a cross-country analysis. *Entrepreneurship and sustainability issues*, 4(4):574–590, 2017.

[86] J. D. Ward, P. C. Sutton, A. D. Werner, R. Costanza, S. H. Mohr, and C. T. Simmons. Is decoupling gdp growth from environmental impact possible? *PloS one*, 11(10):e0164733, 2016.

[87] K. Yang and C. Shahabi. On the stationarity of multivariate time series for correlation-based data analysis. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 4–pp. IEEE, 2005.

[88] G. Carleo, I. Cirac, K. Cranmer, L. Daudet, M. Schuld, N. Tishby, L. Vogt-Maranto, and L. Zdeborová. Machine learning and the physical sciences. *Reviews of Modern Physics*, 91(4):045002, 2019.

[89] Roteiro para a neutralidade carbónica 2050. Technical report, Resolução do Conselho de Ministros nº 107/2019, Diário da República, 123, July 2019.

Appendix A

Input Data

This appendix contains data gathered through several sources and data estimated with interpolation. The gathering procedure is described in chapter 3. Some data resultant from the modelling phase as well as from the prediction are also included, the origin of the data is explained in chapter 4.

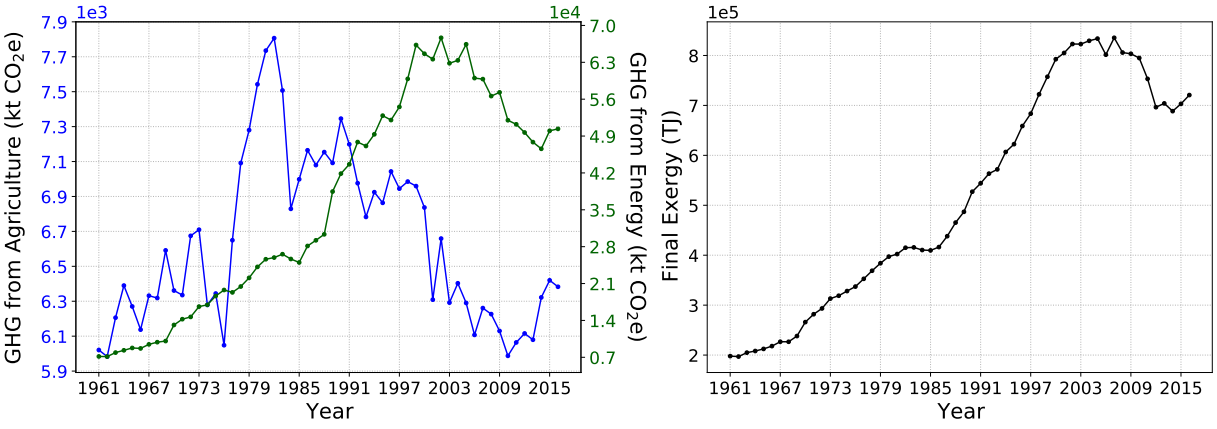


Figure A.1: Data input to the model in the emissions modelling.

The left panel contains both time series for GHG emissions throughout time; the right panel depicts the total final exergy. During data preparation, the LULC and GHG data referring to 2015 is left out of the input data to allow for a lag within the features.

Year	Categories (%)					Final Exergy (TJ)	GDP (2015 Mrd EURO-PTE)
	Permanent Pasture	Permanent Culture & Arable Land	Forest	Urban/ Artificial	Other		
1961	9.2	33.2	32.0	2.4	23.2	33	5111.5
1962	9.2	33.3	32.3	2.4	22.9	37	5487.1
1963	9.2	33.3	32.5	2.4	22.5	38	5761.7
1964	9.2	33.4	32.8	2.5	22.2	40	6130.1
1965	9.2	33.5	33.0	2.5	21.8	44	7134.3
1966	9.2	33.5	33.1	2.5	21.7	46	7791.8
1967	9.2	33.6	33.2	2.6	21.5	48	8924.9
1968	9.2	33.6	33.3	2.6	21.3	51	9169.8
1969	9.2	33.7	33.3	2.6	21.1	52	10085.6
1970	9.2	33.8	33.4	2.7	20.9	56	11958.6
1971	9.2	33.8	33.5	2.7	20.7	62	13556.6
1972	9.2	33.9	33.6	2.7	20.5	69	14983.3
1973	9.2	34.0	33.7	2.8	20.4	72	16265.6
1974	9.2	34.0	33.8	2.8	20.2	74	14231.4
1975	9.2	34.1	33.9	2.8	20.0	70	14483.2
1976	9.2	34.1	34.0	2.9	19.8	72	12106.8
1977	9.2	34.2	34.0	2.9	19.7	76	12263.1
1978	9.2	34.2	34.1	2.9	19.5	81	14389.0
1979	9.2	34.3	34.2	3.0	19.3	87	14498.2
1980	9.2	34.3	34.3	3.0	19.2	91	14228.9
1981	9.2	34.4	34.4	3.0	19.0	93	14487.1
1982	9.2	34.4	34.6	3.1	18.8	95	15319.7
1983	9.2	34.4	34.7	3.1	18.6	96	17194.9
1984	9.2	34.5	34.9	3.1	18.3	95	17044.9
1985	9.2	34.5	35.0	3.2	18.1	96	17905.4
1986	9.2	33.8	35.2	3.2	18.6	100	17948.0
1987	9.2	34.1	35.4	3.2	18.0	107	18040.2
1988	9.2	33.4	35.6	3.3	18.5	113	18812.5
1989	9.2	34.6	35.8	3.3	17.1	120	19444.4
1990	9.3	34.3	36.1	3.3	17.0	130	20319.1
1991	9.4	34.0	36.3	3.4	16.9	134	20460.0
1992	9.6	33.7	36.5	3.4	16.8	138	20460.0
1993	9.7	33.5	36.7	3.4	16.7	137	20337.0
1994	10.5	32.6	36.9	3.5	16.6	139	20778.1
1995	11.2	31.7	37.1	3.5	16.5	143	21213.2
1996	11.0	31.3	37.0	3.6	17.1	148	21139.3
1997	10.8	30.9	36.9	3.7	17.6	154	23551.9
1998	13.0	29.0	36.8	3.8	17.4	162	26547.2
1999	15.2	27.0	36.7	3.9	17.2	168	28602.3
2000	15.5	26.4	36.6	4.0	17.6	174	31515.8
2001	15.8	25.7	36.5	4.1	17.9	178	22252.5
2002	16.0	25.0	36.4	4.2	18.3	179	20796.6
2003	16.3	24.4	36.3	4.3	18.7	177	19976.7
2004	17.8	22.6	36.2	4.4	18.9	181	23176.9
2005	19.3	20.9	36.1	4.5	19.2	182	22593.4
2006	19.4	19.7	36.0	4.6	20.4	185	17356.9
2007	19.5	18.5	35.9	4.6	21.6	190	17248.9
2008	19.5	19.5	35.7	4.7	20.5	190	15632.7
2009	19.5	20.6	35.6	4.7	19.6	184	15319.0
2010	19.6	20.4	35.5	4.8	19.7	187	14997.7
2011	19.7	20.3	35.6	4.8	19.6	184	13837.3
2012	19.8	20.1	35.8	4.9	19.5	177	14009.2
2013	19.9	19.9	35.9	4.9	19.4	175	14302.3
2014	20.1	19.7	36.1	5.0	19.2	177	14707.1
2015	20.3	19.5	36.2	5.0	19.0	180	15009.3
2016	20.5	19.3	36.3	5.0	18.8	183	14835.1

Table A.1: Data used for land use modelling.

Data points highlighted in green were retrieved via official sources, blue data points were estimated via linear interpolation. Final exergy data is from agriculture sector, and GDP data is at 2015 reference levels.

Year	**GHG** (kt CO_2e)		**Final Exergy** (TJ)
	Agriculture	Energy	
1961	6020.7	7160.5	197882.4
1962	5983.0	7136.4	196988.0
1963	6205.8	7892.6	204871.5
1964	6389.8	8326.2	208122.0
1965	6270.1	8772.5	212337.6
1966	6137.3	8683.3	217938.6
1967	6331.8	9456.2	226701.3
1968	6318.4	9869.4	226637.4
1969	6591.2	10084.0	238092.0
1970	6361.3	13136.0	266006.4
1971	6335.2	14234.6	281783.3
1972	6674.6	14664.7	293384.2
1973	6709.6	16622.7	313107.4
1974	6278.8	16965.3	318842.8
1975	6344.2	18634.5	328180.1
1976	6048.1	19774.5	337152.3
1977	6649.3	19321.8	352716.0
1978	7091.5	20436.7	368872.1
1979	7280.0	22077.9	383900.9
1980	7542.2	24164.2	397150.1
1981	7735.2	25615.2	402272.0
1982	7806.7	25933.5	415008.2
1983	7507.1	26572.5	415629.2
1984	6829.1	25660.2	410342.3
1985	6998.4	25002.0	409616.7
1986	7164.5	28132.0	416247.9
1987	7079.6	29205.5	438083.4
1988	7154.1	30342.8	465289.2
1989	7092.1	38463.6	486915.7
1990	7345.7	41866.5	527164.0
1991	7198.8	43637.1	543981.8
1992	6976.1	47863.1	563363.6
1993	6782.8	47112.5	571907.4
1994	6924.6	49330.0	606699.0
1995	6863.5	52873.5	622399.2
1996	7043.2	52037.7	658711.1
1997	6944.8	54531.8	683443.2
1998	6985.3	59852.9	722227.4
1999	6959.3	66280.8	757365.6
2000	6836.8	64630.7	792521.6
2001	6308.9	63588.9	805046.7
2002	6658.9	67655.3	822973.2
2003	6292.0	62815.3	822776.5
2004	6402.9	63370.8	829360.8
2005	6290.1	66423.9	833758.2
2006	6107.2	60003.5	801516.2
2007	6260.8	59812.2	835426.7
2008	6226.3	56587.4	805845.7
2009	6129.8	57291.4	803400.5
2010	5987.3	52001.8	794908.6
2011	6064.0	51225.5	753131.4
2012	6115.5	49676.1	696407.5
2013	6079.4	47848.1	704014.8
2014	6321.9	46579.4	688496.1
2015	6419.8	49995.1	703093.1
2016	6382.8	50362.3	720712.4

Table A.2: Additional data used for emissions modelling.

Data points highlighted in green were retrieved via official sources. Final exergy data is from every sector in Portugal.

Year	GDP (2015 Mrd EURO-PTE)	Final Exergy (TJ)
2015	180	15009.3
2016	183	14835.1
2017	189	14683.3
2018	195	14531.4
2019	200	14379.6
2020	181	14227.8
2021	180	14075.9
2022	178	13924.1
2023	177	13772.3
2024	175	13620.4
2025	174	13468.6
2026	172	13316.8
2027	171	13164.9
2028	169	13013.1
2029	167	12861.3

Table A.3: Data from scenario P for land use distribution forecast.

Year	GDP (2015 Mrd EURO-PTE)	Final Exergy (TJ)
2015	180	15009.3
2016	183	14835.1
2017	189	14951.3
2018	195	15067.6
2019	200	15183.8
2020	181	15300.1
2021	186	15416.3
2022	190	15532.6
2023	195	15648.8
2024	200	15765.1
2025	205	15881.3
2026	210	15997.6
2027	214	16113.8
2028	219	16230.1
2029	224	16346.4

Table A.4: Data from scenario O for land use distribution forecast.

Table A.5: Joined data from both scenarios considered for land use distribution forecast.

Year	GDP (2015 Mrd EURO-PTE)	Final Exergy (TJ)
2016	183	720712.4
2017	189	719347.2
2018	195	717982.1
2019	200	716616.9
2020	181	715251.7
2021	180	713886.5
2022	178	712521.4
2023	177	711156.2
2024	175	709791.0
2025	174	708425.9
2026	172	707060.7
2027	171	705695.5
2028	169	704330.3
2029	167	702965.2
2030	169	701600.0

Table A.6: Data from scenario P for GHG emissions forecast.

Year	GDP (2015 Mrd EURO-PTE)	Final Exergy (TJ)
2016	183	720712.4
2017	189	728254.4
2018	195	735796.3
2019	200	743338.3
2020	181	750880.3
2021	186	758422.3
2022	190	765964.2
2023	195	773506.2
2024	200	781048.2
2025	205	788590.1
2026	210	796132.1
2027	214	803674.1
2028	219	811216.1
2029	224	818758.0
2030	231	826300.0

Table A.7: Data from scenario O for GHG emissions forecast.

Table A.8: Joined data from both scenarios considered for GHG emissions forecast.

Appendix B

Training Results

	Model	Sub-model	Categories					MSE*	
			Permanent Pasture	Permanent Culture & Arable Land	Forest	Urban/ Artificial	Other	Training	Baseline**
RMSE (%)	M1	SM1	4.3	1.6	0.3	0.6	2.6	0.015	0.133
		SM2	0.9	1.9	0.9	0.9	1.4	0.023	
		SM3	0.4	1.1	0.3	0.6	1.5	0.026	
	M2	SM1	0.2	0.6	0.2	0.3	0.8	0.008	
		SM2	1.7	0.6	0.9	0.2	2.5	0.009	
		SM3	2.3	1.6	0.2	0.4	0.5	0.012	
	M3	SM1	1.6	2.1	0.8	0.4	0.1	0.006	
		SM2	2.4	0.4	0.1	0.8	2.8	0.008	
		SM3	2.7	3.2	1.0	0.3	1.8	0.008	
	M4	SM1	2.0	1.7	1.3	0.3	1.2	0.028	
		SM2	0.7	1.7	0.2	0.6	0.5	0.032	
		SM3	1.3	0.3	0.6	0.8	1.6	0.035	
	M5	SM1	2.2	0.2	0.3	0.5	2.5	0.009	
		SM2	3.7	0.4	0.4	0.1	3.6	0.009	
		SM3	0.5	1.8	0.3	0.0	1.7	0.016	
	M6	SM1	0.2	1.8	0.8	0.3	1.4	0.013	
		SM2	1.5	0.4	0.4	0.2	1.1	0.013	
		SM3	2.7	2.5	0.3	0.1	0.3	0.014	

* MSE is measured with scaled data, non directly comparable with RMSE calculated on non-scaled data

** Baseline MSE is estimated only once because there is a unique baseline model

Table B.1: Losses that models incurred on testing (left) and validation (right) data from fold #2.

Models or sub-models highlighted in red were excluded from the analysis due to not meeting the established criteria for either the Mean Square Error (MSE) or Root Mean Square Error (RMSE).

	Model	Sub-model	Categories					MSE*	
			Permanent Pasture	Permanent Culture & Arable Land	Forest	Urban/ Artificial	Other	Training	Baseline**
RMSE (%)	M1	SM1	2.4	3.0	0.2	0.5	0.2	0.008	0.065
		SM2	2.4	0.8	1.5	0.8	0.2	0.010	
		SM3	2.5	1.8	0.4	0.3	0.4	0.011	
	M2	SM1	0.9	1.6	0.9	0.3	0.4	0.004	
		SM2	1.2	0.4	0.3	0.4	1.4	0.005	
		SM3	2.6	3.1	0.3	0.7	0.6	0.005	
	M3	SM1	1.3	0.7	0.3	0.3	1.3	0.006	
		SM2	0.8	0.6	0.2	0.1	0.1	0.007	
		SM3	1.5	0.4	0.1	0.2	1.4	0.007	
	M4	SM1	0.6	0.4	1.7	0.3	2.0	0.030	
		SM2	2.9	0.5	0.5	0.4	3.3	0.033	
		SM3	1.8	3.1	0.9	0.6	0.3	0.034	
	M5	SM1	0.8	0.0	0.3	0.3	1.3	0.007	
		SM2	1.2	1.5	0.7	0.6	1.1	0.009	
		SM3	0.9	1.0	0.7	0.1	1.0	0.010	
	M6	SM1	1.1	0.4	0.7	0.3	1.8	0.007	
		SM2	1.5	0.9	0.5	0.5	1.5	0.009	
		SM3	3.1	2.6	1.0	0.2	0.7	0.011	

* MSE is measured with scaled data, non directly comparable with RMSE calculated on non-scaled data

** Baseline MSE is estimated only once because there is a unique baseline model

Table B.2: Losses that models incurred on testing (left) and validation (right) data from fold #3.

Models or sub-models highlighted in red were excluded from the analysis due to not meeting the established criteria for either the Mean Square Error (MSE) or Root Mean Square Error (RMSE).

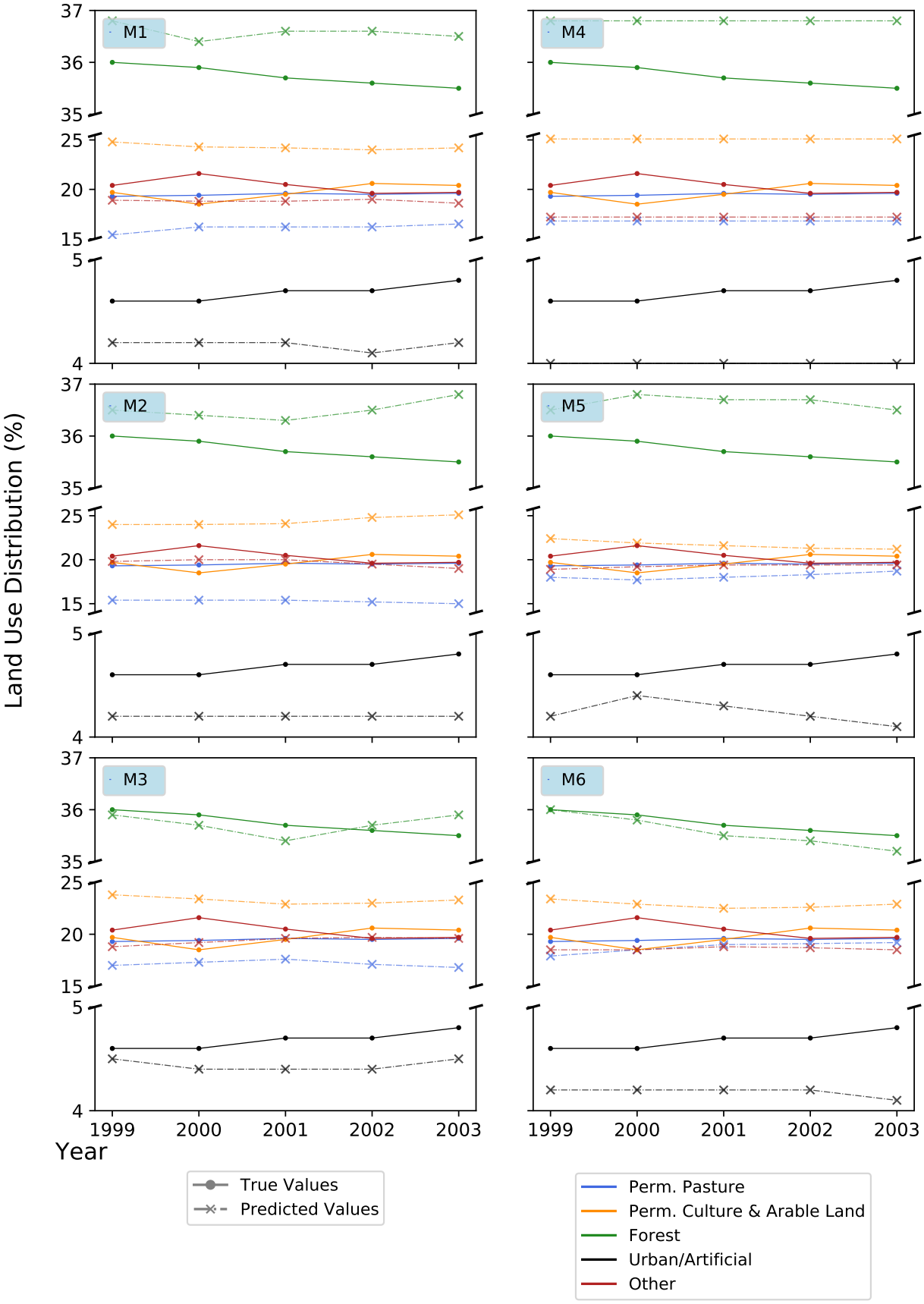


Figure B.1: Comparison between true and predicted values by each model *M* for fold #0.

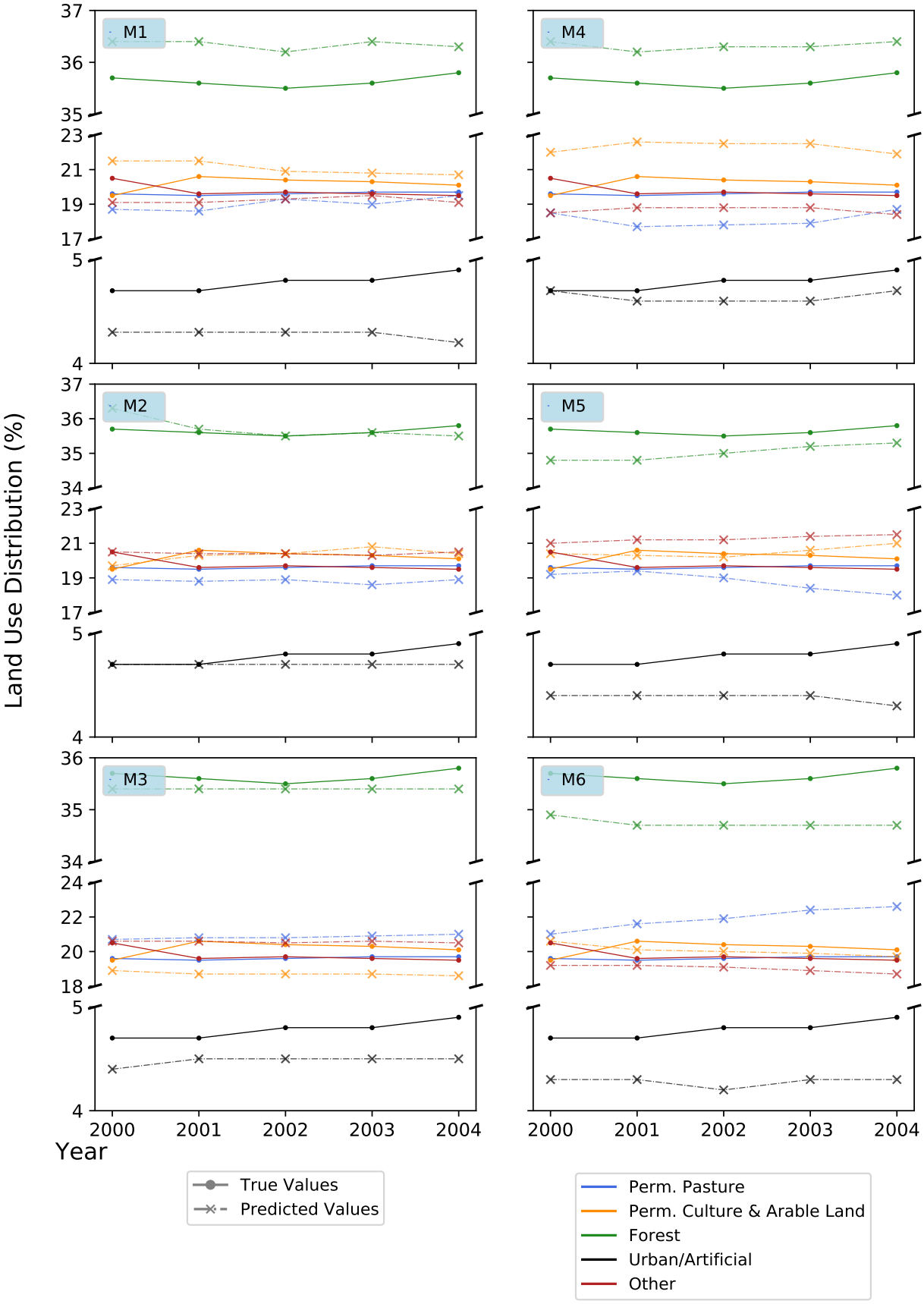


Figure B.2: Comparison between true and predicted values by each model *M* for fold #1.

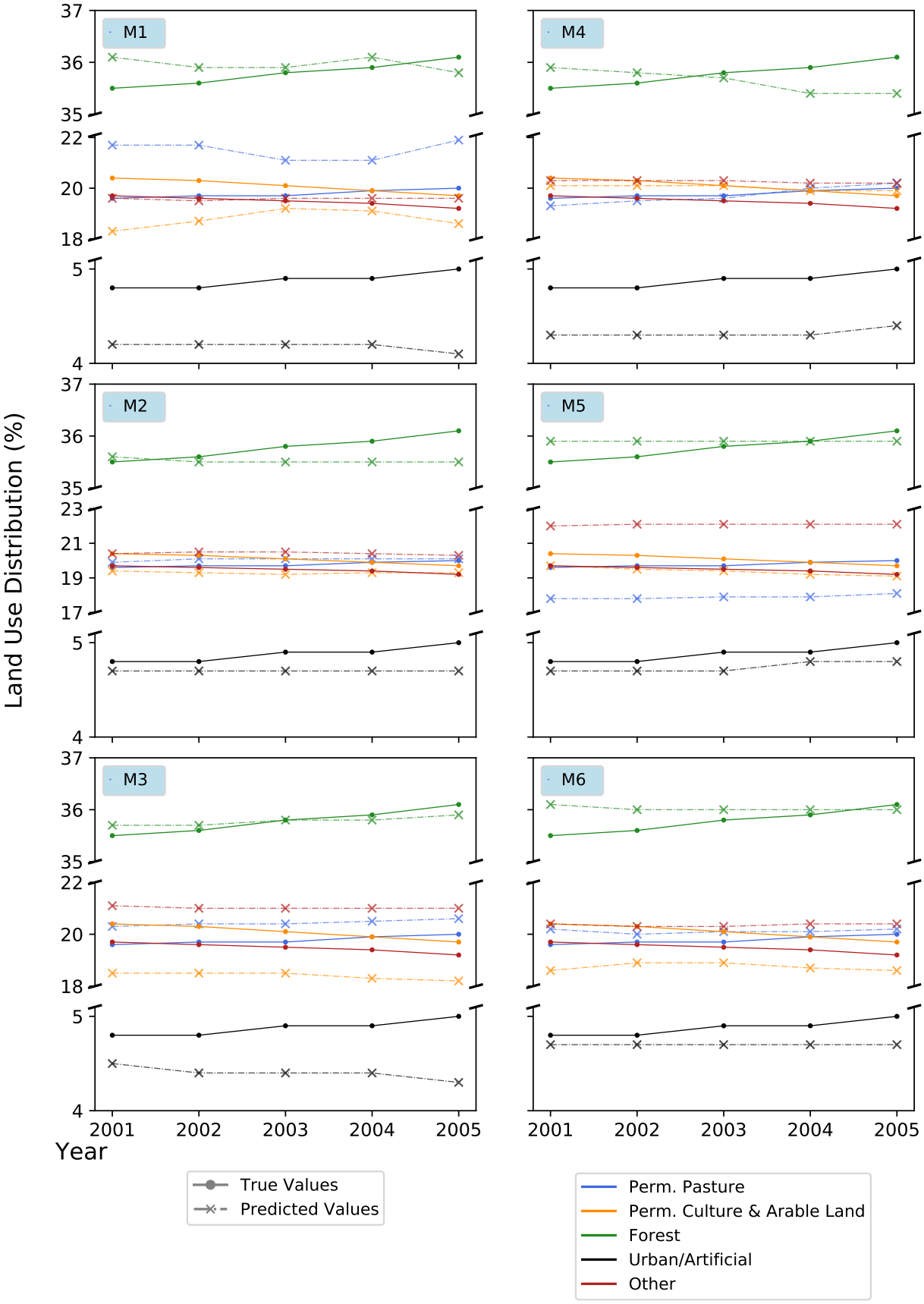


Figure B.3: Comparison between true and predicted values by each model *M* for fold #2.

	M1					
	Fold 0			Fold 1		
Parameters	SM1	SM2	SM3	SM1	SM2	SM3
kernel_initializer	he_uniform	glorot_normal	lecun_uniform	he_normal	glorot_normal	he_uniform
activation	elu	sigmoid	selu	elu	sigmoid	relu
loss	mae	mae	mae	mse	mse	mse
optimizer	opt2	opt3	opt2	opt2	opt1	opt3
clipnorm	1.0	2.0	1.0	1.0	1.0	1.0
nesterov	False	-	False	True	-	-
n_layers	1	2	2	1	1	1
n_neurons	71	6	96	46	81	56
n_neurons0	61	51	66	76	6	51
n_neurons1	-	-	66	16	-	-
n_neurons2	-	-	-	-	-	-

	M1								
	Fold 2			Fold 3			Fold 4		
Parameters	SM1	SM2	SM3	SM1	SM2	SM3	SM1	SM2	SM3
kernel_initializer	he_uniform	he_normal	lecun_uniform	glorot_uniform	glorot_normal	glorot_normal	glorot_uniform	glorot_normal	glorot_normal
activation	elu	elu	elu	tanh	tanh	tanh	tanh	tanh	tanh
loss	mae	mse	mse	mae	mse	mse	mse	mae	mae
optimizer	opt2	opt2	opt4	opt2	opt2	opt2	opt2	opt2	opt2
clipnorm	1.0	1.0	1.0	1.0	1.0	2.0	1.0	0.5	0.5
nesterov	False	True	-	False	False	False	True	False	True
n_layers	1	2	1	3	2	2	1	2	1
n_neurons	71	91	36	51	51	56	56	71	26
n_neurons0	61	21	71	21	31	26	41	26	26
n_neurons1	-	76	-	46	51	46	-	51	-
n_neurons2	-	-	-	56	-	-	-	-	-

Table B.3: Optimal sets of hyperparameters for each sub-model (SM) of model architecture #1.

	M2					
	Fold 0			Fold 1		
Parameters	SM1	SM2	SM3	SM1	SM2	SM3
kernel_initializer	he_uniform	lecun_normal	lecun_normal	he_uniform	he_uniform	he_normal
activation	elu	selu	selu	relu	elu	elu
loss	mse	mse	mse	mae	mse	mse
optimizer	opt5	opt2	opt2	opt2	opt5	opt3
clipnorm	0.5	1.0	1.0	2.0	0.5	2.0
nesterov	False	False	False	False	-	-
n_layers	0	2	1	1	0	2
n_neurons	21	96	51	96	21	21
n_neurons0	-	81	71	81	-	36
n_neurons1	-	66	-	-	-	31
n_neurons2	-	-	-	-	-	-

	M2								
	Fold 2			Fold 3			Fold 4		
Parameters	SM1	SM2	SM3	SM1	SM2	SM3	SM1	SM2	SM3
kernel_initializer	he_normal	he_uniform	lecun_normal	lecun_uniform	lecun_normal	he_uniform	lecun_normal	he_uniform	he_uniform
activation	elu	relu	selu	selu	selu	relu	selu	elu	elu
loss	mae	mae	mae	mae	mse	mae	mae	mae	mse
optimizer	opt4	opt2	opt2	opt2	opt1	opt2	opt2	opt4	opt3
clipnorm	2.0	2.0	0.5	2.0	0.5	1.0	1.0	1.0	1.0
nesterov	-	False	True	True	-	True	False	-	-
n_layers	0	1	1	2	1	2	3	1	1
n_neurons	71	96	61	91	21	86	71	46	41
n_neurons0	-	81	76	26	26	36	56	51	56
n_neurons1	-	-	-	96	-	86	51	-	-
n_neurons2	-	-	-	-	-	-	26	-	-

Table B.4: Optimal sets of hyperparameters for each sub-model (SM) of model architecture #2.

	M3					
	Fold 0			Fold 1		
Parameters	SM1	SM2	SM3	SM1	SM2	SM3
kernel_initializer	he_normal	glorot_normal	lecun_uniform	lecun_uniform	glorot_normal	he_normal
activation	elu	tanh	selu	selu	tanh	elu
loss	mse	mse	mse	mse	mse	mse
optimizer	opt2	opt3	opt2	opt2	opt2	opt2
clipnorm	1.0	0.5	2.0	0.5	0.5	1.0
nesterov	False	-	True	False	False	True
n_layers	2	0	0	2	0	1
n_neurons	26	41	76	36	26	96
n_neurons0	56	-	-	46	-	21
n_neurons1	26	-	-	86	-	-
n_neurons2	-	-	-	-	-	-

	M3								
	Fold 2			Fold 3			Fold 4		
Parameters	SM1	SM2	SM3	SM1	SM2	SM3	SM1	SM2	SM3
kernel_initializer	lecun_normal	lecun_uniform	he_uniform	he_normal	lecun_uniform	lecun_normal	glorot_normal	lecun_uniform	glorot_normal
activation	selu	selu	elu	elu	selu	selu	tanh	selu	tanh
loss	mse	mse	mae	mae	mae	mae	mae	mse	mae
optimizer	opt2	opt2	opt2	opt5	opt2	opt2	opt2	opt4	opt1
clipnorm	2.0	0.5	1.0	1.0	0.5	1.0	0.5	1.0	0.5
nesterov	False	False	False	-	True	False	True	-	-
n_layers	1	2	1	0	2	3	1	1	0
n_neurons	56	36	71	86	41	71	26	36	81
n_neurons0	46	46	61	-	86	56	26	71	-
n_neurons1	-	86	-	-	91	51	-	-	-
n_neurons2	-	-	-	-	-	26	-	-	-

Table B.5: Optimal sets of hyperparameters for each sub-model (SM) of model architecture #3.

	M4					
	Fold 0			Fold 1		
Parameters	SM1	SM2	SM3	SM1	SM2	SM3
kernel_initializer	lecun_uniform	glorot_uniform	he_uniform	he_uniform	he_normal	glorot_uniform
activation	selu	tanh	elu	relu	relu	tanh
loss	mse	mae	mae	mse	mse	mse
optimizer	opt4	opt4	opt4	opt3	opt3	opt5
clipnorm	2.0	2.0	1.0	2.0	1.0	2.0
nesterov	-	-	-	-	-	-
n_layers	2	1	2	3	2	3
filters	15	15	7	3	13	11
kernel_size	4	6	6	3	3	2

	M4								
	Fold 2			Fold 3			Fold 4		
Parameters	SM1	SM2	SM3	SM1	SM2	SM3	SM1	SM2	SM3
kernel_initializer	lecun_normal	lecun_normal	glorot_uniform	he_normal	lecun_normal	he_normal	he_normal	he_normal	lecun_normal
activation	selu	selu	tanh	elu	selu	relu	elu	relu	selu
loss	mae	mse	mae	mae	mae	mae	mae	mse	mse
optimizer	opt4	opt1	opt5	opt3	opt5	opt4	opt5	opt3	opt4
clipnorm	0.5	0.5	2.0	0.5	2.0	2.0	1.0	1.0	0.5
nesterov	-	-	-	-	-	-	-	-	-
n_layers	1	1	2	2	2	3	1	2	1
filters	13	5	15	11	9	11	5	15	5
kernel_size	5	4	6	6	2	4	4	3	6

Table B.6: Optimal sets of hyperparameters for each sub-model (SM) of model architecture #4.

	M5					
	Fold 0			Fold 1		
Parameters	SM1	SM2	SM3	SM1	SM2	SM3
kernel_initializer	lecun_uniform	lecun_uniform	lecun_normal	he_uniform	lecun_uniform	he_normal
activation	selu	selu	selu	elu	selu	elu
loss	mse	mse	mse	mae	mse	mse
optimizer	opt2	opt2	opt5	opt2	opt2	opt2
clipnorm	0.5	0.5	0.5	2.0	1.0	1.0
nesterov	True	True	-	True	True	True
n_layers	1	1	1	1	2	1
n_neurons	46	51	71	81	16	36
n_neurons0	71	66	31	71	66	26
n_neurons1	-	-	-	-	81	-
n_neurons2	-	-	-	-	-	-
filters	5	5	5	11	3	9
kernel_size	4	4	6	4	2	4

	M5								
	Fold 2			Fold 3			Fold 4		
Parameters	SM1	SM2	SM3	SM1	SM2	SM3	SM1	SM2	SM3
kernel_initializer	lecun_normal	lecun_uniform	lecun_uniform	he_normal	he_normal	he_normal	lecun_normal	glorot_normal	lecun_uniform
activation	selu	selu	selu	elu	elu	elu	selu	sigmoid	selu
loss	mse	mae	mae	mse	mae	mae	mae	mae	mae
optimizer	opt2	opt2	opt2	opt3	opt2	opt2	opt2	opt1	opt4
clipnorm	1.0	2.0	1.0	1.0	0.5	0.5	2.0	0.5	2.0
nesterov	False	True	False	-	True	True	False	-	-
n_layers	3	1	1	1	3	2	1	0	0
n_neurons	16	91	66	36	96	96	46	1	36
n_neurons0	71	96	96	41	86	26	51	-	-
n_neurons1	26	-	-	-	61	96	-	-	-
n_neurons2	71	-	-	-	56	-	-	-	-
filters	13	15	13	3	15	15	5	9	5
kernel_size	3	6	4	6	6	6	4	6	2

Table B.7: Optimal sets of hyperparameters for each sub-model (SM) of model architecture #5.

	M6					
	Fold 0			Fold 1		
Parameters	SM1	SM2	SM3	SM1	SM2	SM3
kernel_initializer	lecun_uniform	lecun_uniform	lecun_normal	he_uniform	lecun_uniform	he_normal
activation	selu	selu	selu	elu	selu	elu
loss	mse	mse	mse	mse	mse	mse
optimizer	opt2	opt2	opt4	opt2	opt2	opt2
clipnorm	0.5	0.5	0.5	2.0	2.0	1.0
nesterov	True	True	-	False	False	True
n_layers	1	1	0	1	2	2
n_neurons	46	41	1	56	96	16
n_neurons0	76	76	-	21	51	61
n_neurons1	-	-	-	-	56	56
n_neurons2	-	-	-	-	-	-
filters	3	3	3	7	13	9
kernel_size	4	4	4	2	2	4

	M6								
	Fold 2			Fold 3			Fold 4		
Parameters	SM1	SM2	SM3	SM1	SM2	SM3	SM1	SM2	SM3
kernel_initializer	he_uniform	lecun_normal	lecun_uniform	lecun_normal	glorot_uniform	he_normal	he_uniform	lecun_normal	lecun_uniform
activation	relu	selu	selu	selu	tanh	elu	elu	selu	selu
loss	mae	mse	mse	mae	mae	mse	mae	mae	mse
optimizer	opt4	opt2	opt2	opt2	opt3	opt2	opt4	opt2	opt2
clipnorm	1.0	0.5	0.5	0.5	2.0	0.5	1.0	2.0	0.5
nesterov	-	True	False	True	-	False	-	False	True
n_layers	1	2	1	3	2	3	0	1	1
n_neurons	51	36	46	21	66	11	51	41	36
n_neurons0	31	91	41	86	21	81	-	51	86
n_neurons1	-	31	-	11	86	31	-	-	-
n_neurons2	-	-	-	86	-	81	-	-	-
filters	7	3	11	3	11	11	13	13	3
kernel_size	4	4	4	6	4	6	2	4	4

Table B.8: Optimal sets of hyperparameters for each sub-model (SM) of model architecture #6.

Appendix C

Sensitivity Analysis Results

Year	Study	Categories (%)				
		Permanent Pasture	Permanent Culture & Arable Land	Forest	Urban/Artificial	Other
2017	-	20.4 ± 0.7	19.4 ± 0.9	36.2 ± 0.3	5.0 ± 0.2	19.0 ± 0.6
2030	A	21.2 ± 0.7	18.6 ± 0.9	36.5 ± 0.3	5.1 ± 0.2	18.6 ± 0.6
	B	22.4 ± 0.7	17.1 ± 0.9	36.8 ± 0.3	5.3 ± 0.2	18.4 ± 0.6
	C	20.8	19.2	36.5	5.1	18.5
	D	21.3	18.6	36.4	5.1	18.6
	E	18.8	21.5	35.9	4.7	19.0
	F	23.5	15.5	37.3	5.4	18.2

Table C.1: Sensitivity analysis' results for land use distribution

Study A: Scenario P without COVID-19 data points
Study B: Scenario O without COVID-19 data points
Study C: Final exergy year-over-year growth of -5%, constant GDP
Study D: Final exergy year-over-year growth of 5%, constant GDP
Study E: GDP year-over-year growth of -5%, constant final exergy
Study F: GDP year-over-year growth of 5%, constant final exergy

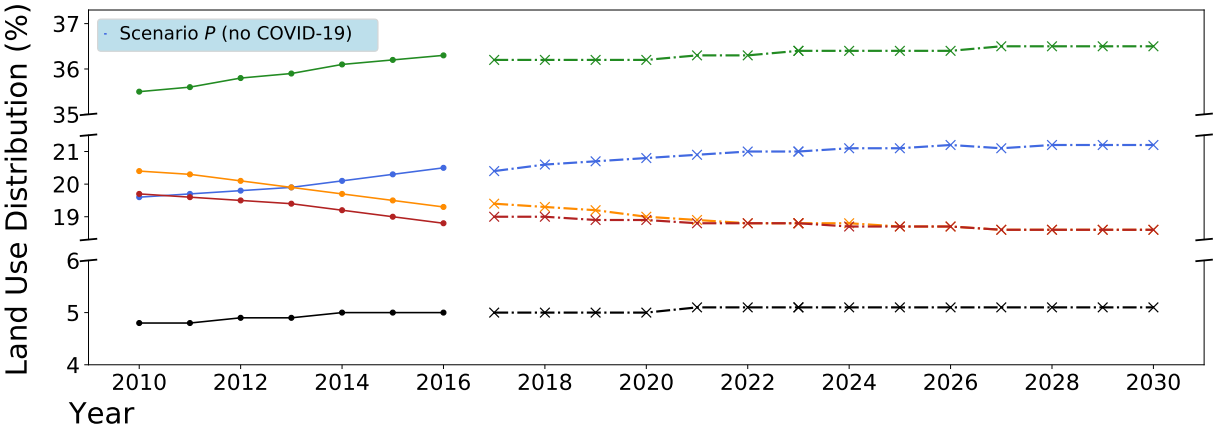


Figure C.1: Land use distribution forecast for a scenario P without COVID-19 data points.

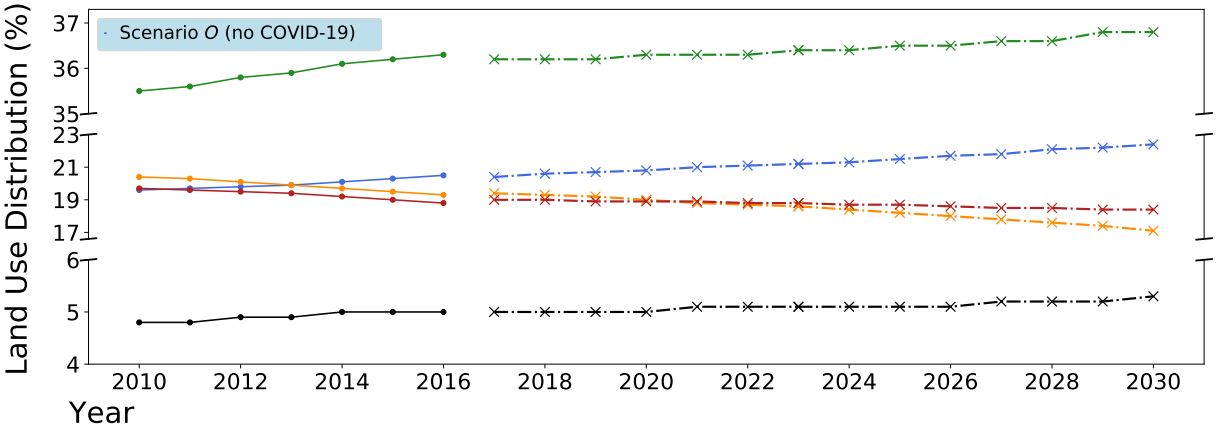


Figure C.2: Land use distribution forecast for a scenario O without COVID-19 data points.

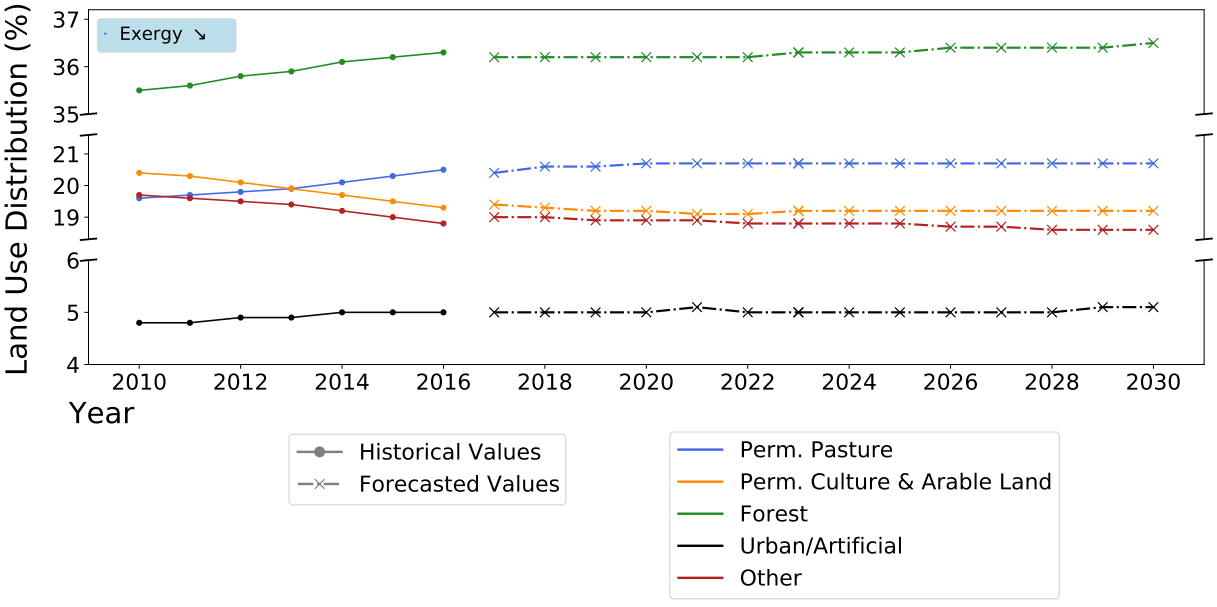


Figure C.3: Land use distribution forecast for a scenario with declining final exergy and constant GDP.

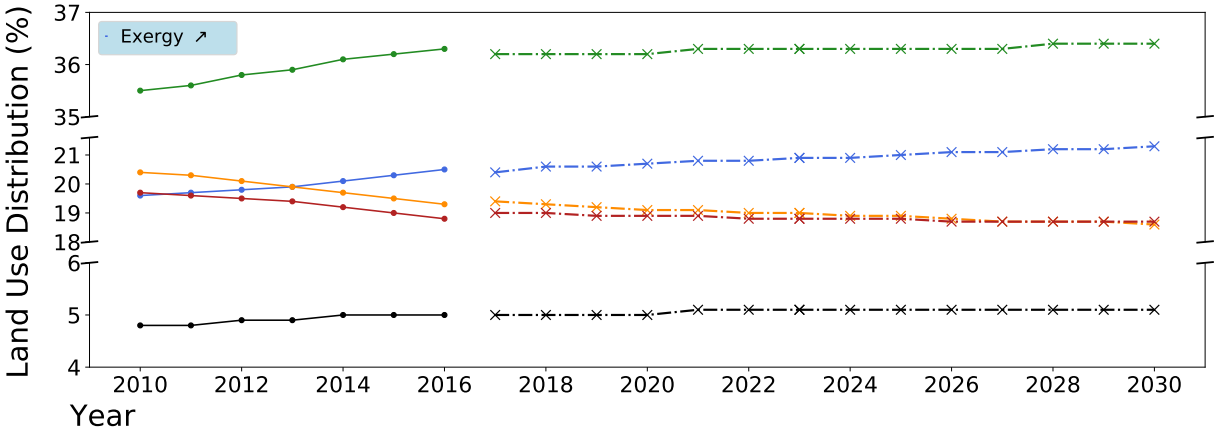


Figure C.4: Land use distribution forecast for scenario with growing final exergy and constant GDP.

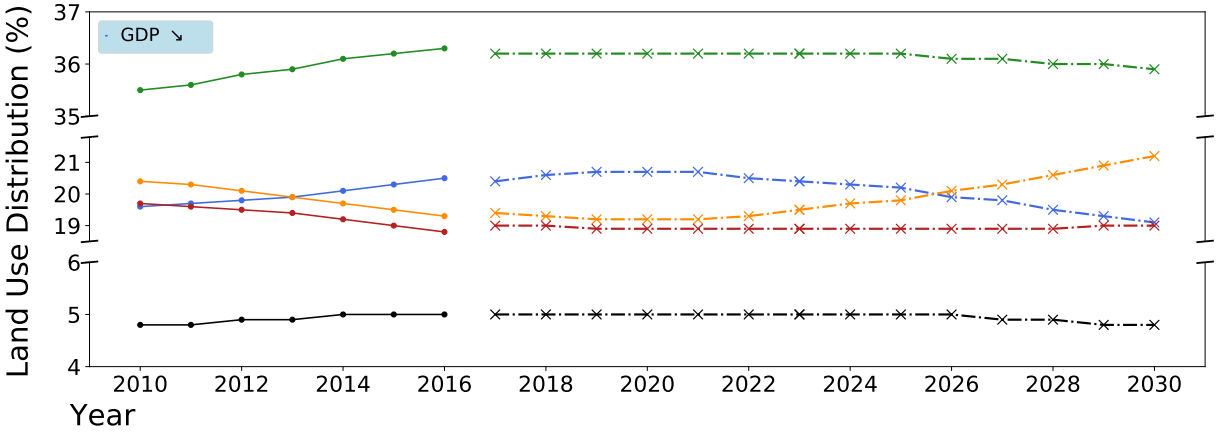


Figure C.5: Land use distribution forecast for scenario with declining GDP and constant final exergy.

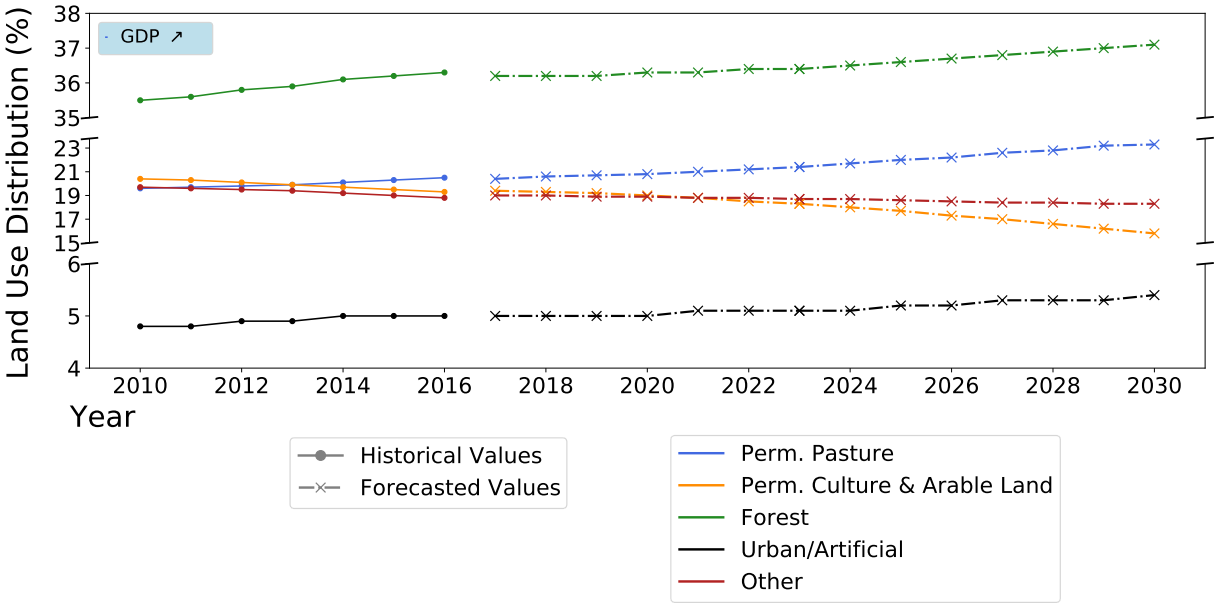


Figure C.6: Land use distribution forecast for scenario with growing GDP and constant final exergy.

	Scenario P (no COVID-19)		**Scenario O (no COVID-19)**	
Year	$GHG_{agriculture}$*	GHG_{energy}*	$GHG_{agriculture}$*	GHG_{energy}*
2017	6.4 ± 0.1	51.5 ± 2.1	6.4 ± 0.1	51.6 ± 2.1
2030	6.5 ± 0.1	54.8 ± 2.1	6.4 ± 0.1	61.4 ± 2.1

* All values measured in terms of metric megaton of CO_2 equivalent (Mt CO_2e)

Table C.2: GHG's first and last numerical results from the forecasts for both scenarios P and O without COVID-19 data points

	Exergy ↘		**Exergy ↗**	
Year	$GHG_{agriculture}$*	GHG_{energy}*	$GHG_{agriculture}$*	GHG_{energy}*
2017	6.4	51.5	6.4	51.5
2030	6.7	45.4	6.2	66.5

* All values measured in terms of metric megaton of CO_2 equivalent (Mt CO_2e)

Table C.3: GHG's first and last numerical results from the forecasts for the study where final exergy was changed and GDP kept constant

	GDP ↘		**GDP ↗**	
Year	$GHG_{agriculture}$*	GHG_{energy}*	$GHG_{agriculture}$*	GHG_{energy}*
2017	6.4	51.5	6.4	51.6
2030	6.5	49.6	6.3	61.8

* All values measured in terms of metric megaton of CO_2 equivalent (Mt CO_2e)

Table C.4: GHG's first and last numerical results from the forecasts for the study where GDP was changed and final exergy kept constant

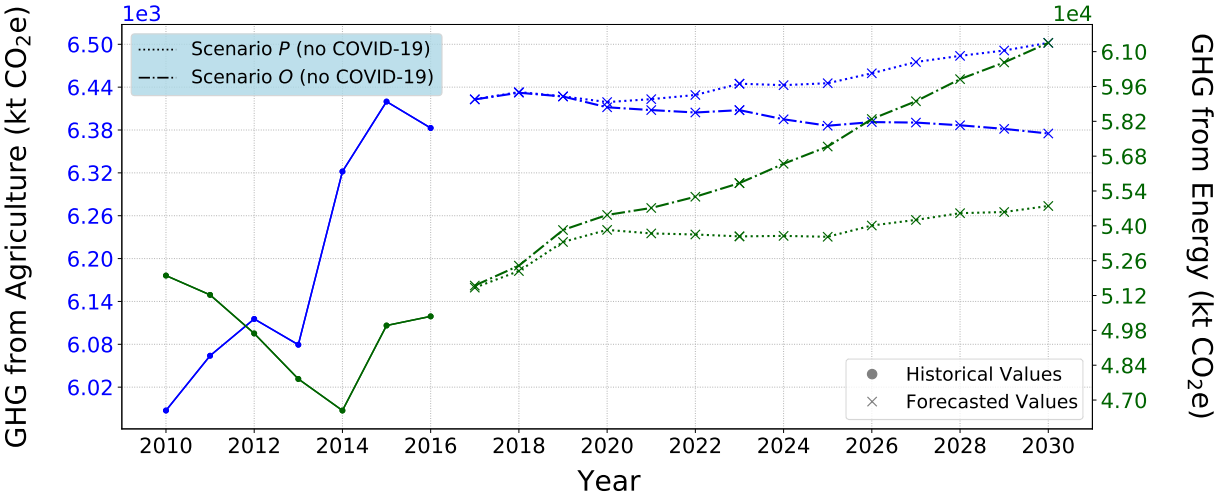


Figure C.7: GHGs from agriculture and from energy forecasts under both pessimistic and optimistic scenarios without COVID-19 data points.

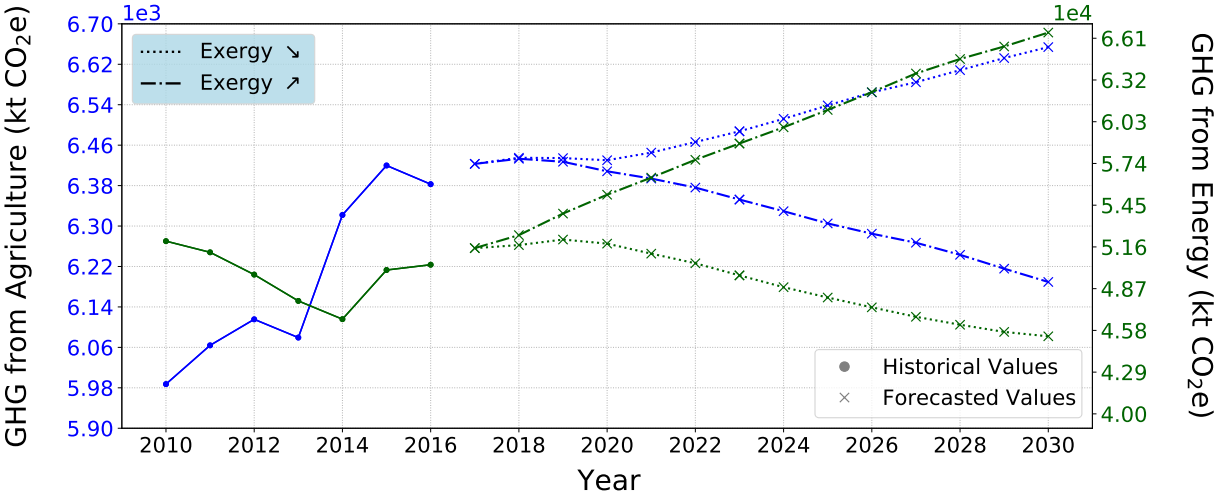


Figure C.8: GHGs from agriculture and from energy forecasts for the study where final exergy was changed and GDP kept constant.

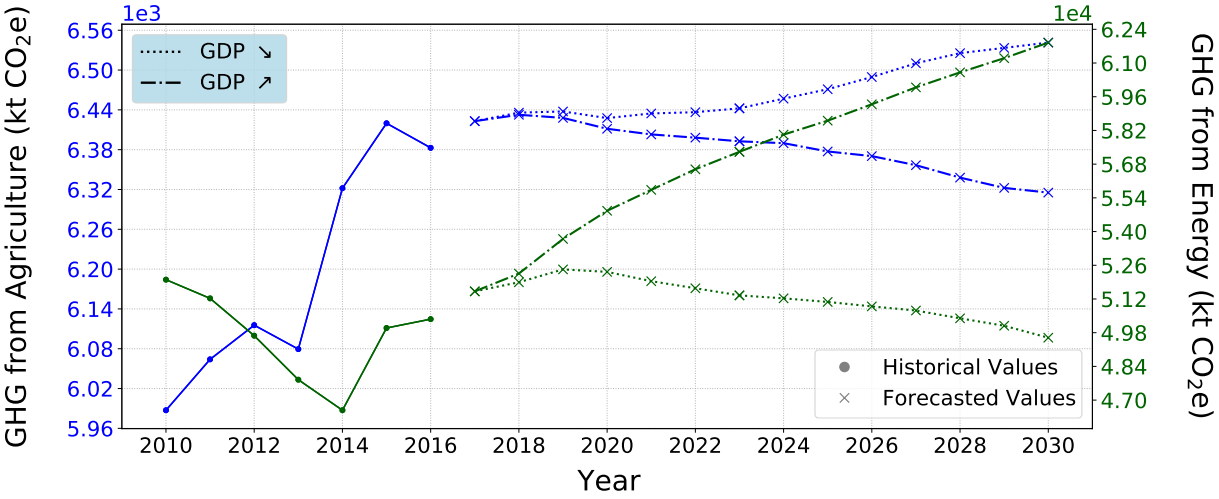


Figure C.9: GHGs from agriculture and from energy forecasts for the study where GDP was changed and final exergy kept constant.