Abstract

The aim of this thesis is to establish whether a *neural network* (NN) can be used for *emulation* of simulated global crop production - retrieved from the computationally demanding *dynamic global vegetation model* (DGVM) *Lund-Potsdam-Jena General Ecosystem Simulator* (LPJ-GUESS). It has been devoted to elaboration with various types of neural network architectures: *Branched NNs* capable of processing inputs of mixed data types; *Convolutional Neural Network* (CNN) architectures able to perform automated temporal feature extraction of the given weather time series; simpler *fully connected* (FC) structures as well as *Multitask NNs*. The NN crop emulation approach was tested for approximation of global annual spring wheat responses to changes in CO^2, temperature, precipitation, and nitrogen fertilizer levels.

The model domain is a multidimensional hyperspace of non-IID samples that can be blocked into climate-, temporal- and global position factor levels. NNs tend to focus on the normally behaving samples and take less notice of rare behaviors and relations. In this vast data set, the varying characteristics and relations can thus be hard to detect - even for a large and complex neural network. Due to these complexities in the LPJ-GUESS sample distributions and because neural network learning is heavily reliant on the data, a fair share of the thesis has been dedicated to network training and sample selection - with the purpose of improving learning without causing the network to overfit.

Further, the *Köppen climate classification system*, based on historical climate and vegetation, was used for aggregation of the emulator domain - in order to form smaller homogeneous groups. It is easier to dissect the data when these disjoint groups are analysed in isolation, which in turn can facilitate input variable selection. Moreover, by aggregating the model domain and allowing for separate deep learning of each domain-fraction, several sub-models can be constructed and trained for a specific Köppen climate region. These can then be combined into an *integrated composite emulator*. In contrast to an emulator trained to model crop production for the whole domain, a *model composition* emulator does not have to account for the differences between the sub-domains and hence only has to focus on learning the within-group relations and patterns in the disjoint climate classes.

Keywords: *Multitask Learning, Convolutional Neural Network (CNN), Branched Neural Network, Dynamic Global Vegetation Models (DGVM), Automated Feature Extraction, Feature Importance, Supervised Machine Learning, Emulator, Surrogate Model, Response Surface Model, Approximation Model, Metamodeling, Model Composition, Regularization, Robustness, Hyperparameter Optimization*

Acknowledgement

The assistance and guidance given by my supervisor Johan Lindström has been greatly appreciated and I am thankful for him letting me take so much of his time. I also wish to acknowledge Stefan Olin for providing me with data and insight in the LPJ-GUESS as well as the GGCMI study. By offering useful remarks and ideas, while also giving me free rein to conduct the study as I saw fit, they made working with this thesis particularly enjoyable.

Lund, May 15 2020

Table of Contents

List of Figures

List of Tables

1
Introduction

A growing world population alongside climate change and greater uncertainties in weather, increases the concern about food security and raises questions about vulnerabilities and potential adaptation strategies in the agricultural sector. This has eventuated in the need for *dynamic global vegetation models* (DGVMs) that both can predict vegetation in changing climate settings, many out of which have not yet been seen in historical record and also at new potential cultivation locations with no previous record of food production. (Franke et al. 2019, pp. 1-2).

A problem with such vegetation models is their computational burden, especially when many different climate scenarios are of interest. Cheap estimates of the simulator outputs can be retrieved from a mimicking *emulator*. Such approximation function can be seen as a statistical representation of the simulator, trained to model the mapping of input data to output targets. (Schmit and Pritchard 2018, p. 1).

This master's thesis seeks to find a surrogate model strategy that can be used for emulation of the DGVM *Lund-Potsdam-Jena General Ecosystem Simulator* (LPJ-GUESS), which is one of the models in the *Global Gridded Crop Model Intercomparison* (GGCMI) study. Active analysis of LPJ-GUESS inputs and outputs was carried through out the model search and iterative optimisation runs. The analysis of inputs mainly involved evaluation of their respective feature importance. But also additional simulator outputs, like above ground biomass and length of different development phases, have been considered with the idea that a multitask learning of these related tasks could improve yield predictions.

Because of the mixture of input data types, the model search has been restricted to *Branched Neural Networks*: in which one branch is constructed as a *Convolutional Neural Network* (CNN) to perform automated temporal feature extraction in the given weather time series; while numeric LPJ-GUESS input data like CO_2 levels, fertilization and location inputs, were passed through *fully connected layers*. This *Branched Multitask CNN* crop emulation approach have been tested for one of the high priority output in the GGCMI, namely *global annual spring wheat*.

Neural networks are heavily reliant on the data and tend to focus on the the normally behaving samples that make up the majority of the samples seen during training. Hence NNs takes less notice of unusually behaving samples and especially in intra diversity within such minority groups. The codomain, with one-to-one correspondence to the input domain, is a *multidimensional hyperspace* that can be blocked by climate-, temporal- and global position factor levels - including 31 annual crop responses to 672 different climate scenarios, at every global $0.5°$ by $0.5°$ grid cell location covering ice-free land. In this vast hypercube, with unbalanced and non-i.i.d distributions, many block-groups preserve attributes different to the majority and can also have large within-group variance.

Due to these complexities, much of the surrogate modeling involved parallel hyperparameter optimisation, adaptive parameter optimisation, implementation of regularization methods and sample selection for the purpose of improving learning of as many samples as possible without causing the network to overfit.

The presented emulator is a *model composition* of multi-task convolutional neural networks, each trained to predict annual spring wheat as well as above ground biomass in four disjoint regions, classified as tropical, dry, temperate or continental according to the *Köppen climate classification system*. Aggregation of the model domain can be of use if it manages to distribute the samples into subsets with smaller within-group variance, but

deficiencies in the Köppen classification resulted in great intra diversity within the dry climate class in particular. When it became evident that the simulations at dry locations are much harder to model than the others, the focus shifted from individual optimisation of the respective class models, to a search for an architecture that worked well for all climate class models and especially favoured the crop estimates in the dry climate region. This despite the fact that several network architectures had shown to be better suited for the other three climate class-models, in order to facilitate model comparison and conduct a useful analysis.

Notwithstanding the room for improvement, the neural network approach can evidently be used for emulation, especially if complemented with some of the methods suggested in this thesis. Particularly put forward for consideration is implementation of *sequential sample design* and suggested adjustments of the *Köppen climate* aggregation scheme, which most likely will increase the probability of success for combining separate branched multitask CNNs into an integrated composite emulator.

1.1 Background

In 2012 the *The Agricultural Model Intercomparison and Improvement Project* (AgMIP) - a large collaboration of climate, crop and economic modeling communities which connects models from different fields - initiated the Global Gridded Crop Model Intercomparison (GGCMI) (*GGCMI*) to investigate climate impact on agricultural production.

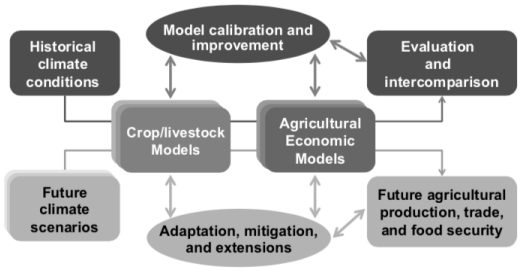


Figure 1.1: AgMIP's module lingkages. McDermid 2015.

GGCMI is a protocol-based research that started with a historic evaluation phase - presented in *The Global Gridded Crop Model Intercomparison: data and modeling protocols for Phase 1* (Elliott et al. 2015) - and continued in a second phase (Franke et al. 2019) in which the participating modeling groups were to simulate global crop responses to different combinations of carbon dioxide, temperature, precipitation and nitrogen levels (collectively refereed to as *CTWN* in GGCMI), based on data between 1980 and 2010. The phase 2 protocol (Franke et al. 2019), presents the data and which climate factor levels of carbon dioxide, nitrogen, temperature and precipitation that are of interest. The levels considered are 10, 60, and 200 kgN/ha nitrogen (N), $360, 510, 660$ and 810 ppm atmospheric carbon dioxide (C), temperature (T) kept unchanged and shifted with $-1, 1, 2, 3, 4$ and 6 Kelvin, and 9 levels of relative change in precipitation (W) at $\pm 50, \pm 30, \pm 20, \pm 10, 0$ percent and at full irrigation at the lowest thresholds for triggering irrigation.[1] That is, a total of 756 ($|C||T||W||N| = 3 \cdot 4 \cdot 7 \cdot 9$) combinations, or 672 ($3 \cdot 4 \cdot 7 \cdot 8$) without the last full irrigation scenarios.(see Franke et al. 2019). Finally, the Global Gridded Crop Models (GGCMs) were evaluated in the model intercomparison in order to understand the climate driven crop processes

[1]Note that the climate scenarios considered here are not identical with those in the older version of the protocol from 2015.

and to improve the models by for example constructing a statistical representation of the GGCMs simulations in form of an *emulator*, a.k.a *surrogate model*.

The emulators can be seen as statistical representations that can take a climate scenario point in a CTWN hypercube spanned by the climate variables C, T, W and N, and map it to a response, such as the expected crop production. (Müller 2019). One of the higher priority tasks to be emulated is how the simulated annual spring wheat responds to shifts in climate, at all potential cultivation locations on the globe, over a 31 year period. The world is represented by a grid-based map with $0.5°$ by $0.5°$ resolution and all ice-free land is assumed to be cultivation locations from 1980 to 2010, which are the 31 simulation years.. Each $0.5°$ by $0.5°$ grid cell thus corresponds to 31 annual spring yield simulations for every climate scenario.

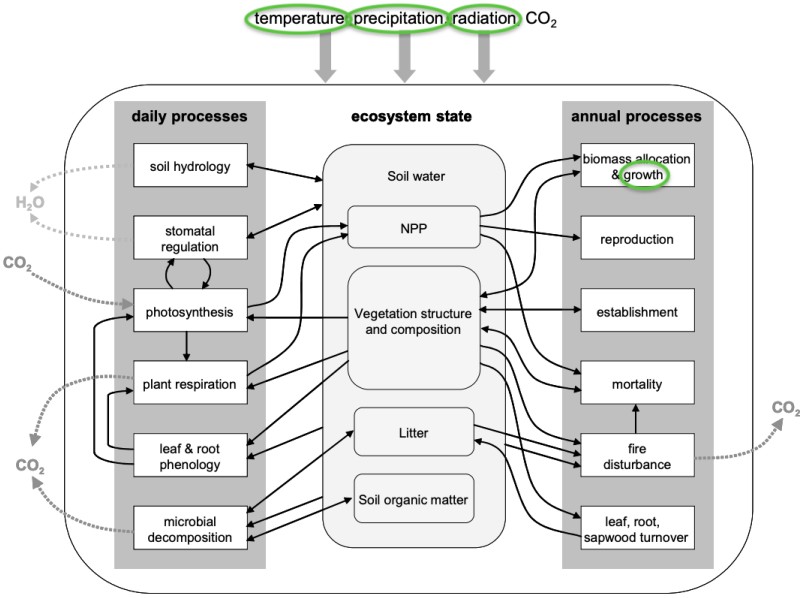


Figure 1.2: Conceptual representation of LPJ-GUESS, with the variables used in this project circled in green.

The DGVM *Lund-Potsdam-Jena General Ecosystem Simulator* (LPJ-GUESS), for which the neural network emulator approach will be tested, is one of many simulators taking part in the GGCMI study. LPJ-GUESS, developed by Ben Smith of Lund University in collaboration with Potsdam Institute for Climate Impact Research and the Max-Planck Institute for Bio-geochemistry (2001), is made up of several connected modules, each modeling an ecosystem processes. Based on input climate data, potential ecosystem processes, implemented at daily or annual timescales, are simulated across one or many grid cells. (Smith 2001). LPJ-GUESS also consist of ecosystem state variables connected to the processes, as can be seen in figure 1.2. (Smith 2001).

The LPJ-GUESS inputs and outputs to be used in the surrogate model, have one-to-one correspondence. That is, for every point $(c, t, w, n, y, l) \in CTWN \cup \mathcal{Y} \cup \mathcal{L}$ (where n is a specified nitrogen (N) level, t a specified shift in temperature (T), etc., $\mathcal{Y}$ the set of years $[y_1, y_2, ..., y_{31}]$ and $\mathcal{L}$ the set of $0.5°$ by $0.5°$ grid cells), the set of inputs $\mathbf{X}(c, t, w, n, y, l)$ is mapped to a crop response $\mathbf{Y}(c, t, w, n, y, l)$

1.1.1 Strategies and Earlier Attempts

The suitability of the emulator design will depend on the problem at hand, how it is formulated and the available data. There is no universally good model, or as George Box put it: "*all models are wrong, but some models are*

useful." (Box and Draper 1987, p. 424). The models usually take different forms for different purposes and fields, but even when restricted to one problem it is hard, if not impossible, to verify that a model is the best among all possible models. Thankfully, there exist several good enough models. By restricting the model search to a smaller set of models, it becomes much easier to find a good or even a best model among the ones given. This subset can be chosen in infinitely many ways, depending on the considerations taken into account, but modelers often restrict the model search to a subset of models with common characteristics and structure that are likely are to perform well. This reduces the problem to finding an optimal or at least good model of a certain type, like linear regression, Gaussian process regression or, as here, Neural Network.

The emulator can be constructed in several different ways, using different inputs and problem formulations. Two previous attempts, using quadratic regression and Gaussian processes, are briefly described below as well as the preparatory bachelor's thesis that laid the foundation for this thesis.

Location Based Quadratic Regression Model for Time-Averaged Yield

One approach suggested in the GGCMI study, was to model time-averaged yield responses to shift in climate parameters, using location-based quadratic regression models with interaction terms.

$$\begin{aligned} Y_{emulator} = \beta_0 &+ \beta_1 c + \beta_2 t + \beta_3 w + \beta_4 n \\ &+ \beta_5 c^2 + \beta_6 t^2 + \beta_7 w^2 + \beta_8 n^2 \\ &+ \beta_9 ct + \beta_{10} cw + \beta_{11} cn + \beta_{12} tw + \beta_{13} tn + \beta_{14} wn \end{aligned} \tag{1.1}$$

where the β parameters have different values for each grid cell. Depending on the variables importance at the considered location, this model consist of up to 15 parameters for all of the locations. Since the parameters are uniquely determined for every 0.5x0.5 grid cell covering ice-free land, this would amount to hundreds of thousand model parameters. [2]

Gaussian Process Emulator (tested for one climate scenario)

The Master's thesis *Emulators For Dynamic Vegetation Models - Supervised Learning In Large Data Sets* (Olsson 2017) investigated whether *Gaussian processes* (GP) could be used for emulation of spring wheat. Rather than constructing a complete emulator, the GP-approach was tested on 196 spread out grid cells, by mapping daily weather strings to predicted annual spring wheat for the climate scenario. The climate scenario was similar to the current, where carbon dioxide (C) is increased to 480 ppm, the levels of rain (W) and temperature (T) are kept unchanged and where fertilization is taken to be 60kg of nitrogen (N) per hectare, which is considered to be a medium level of nitrogen supply.

A set of GP regression models, mainly with constant mean and anisotropic Matérn covariance function were tested using the different summary statistics of precipitation, temperature, radiation and grid cell coordinates latitude and longitude.[3]

[2] Another suggested approach in the GGCMI study was to emulate individual years in a similar fashion as the above mentioned, but instead of using the average annual yield (averaged over the time dimension), by modeling yield of the one year most similar to the season to be emulated. In order to select the season that is most similar in temperature and precipitation, they suggested to make use of summary statistics taken from the four months before harvest (which they call the key growing period). For this case more non-linearities are expected to be found in the CTWN responses produced by this model, compared to the averaged climatological yields, and can thus require higher order polynomial fits.

[3] Tested combinations of summary statistics: Annual averages $[\bar{pr}_{year}, \bar{tas}_{year}, \bar{rsds}_{year}]$; four month growing season averages and grid

The Gaussian Process study suggested a more complex model, but it mat have performed better for other, more informative, summary statistics than those considered as inputs (min, max and mean of the weather time series). (Olsson 2017, p. 29).

Convolutional Neural Network (tested for one climate scenario)

The preparatory Bachelor's thesis (Nilsson 2019), that laid the foundation for this Master's thesis, investigated whether a CNN could be used for emulation of annual spring wheat. Similar to the thesis testing a Gaussian Process, the goal was not to construct a complete emulator, but rather to test it for one climate scenario. Namely the same climate scenario considered in the GP master's thesis, using the same set of locations.

The weather data used in the GP-study was also considered in this bachelor's thesis, but kept as daily time series of observed precipitation, temperature and radiation, instead of using summary statistics. The time series stretched over the 150 days anterior to the harvesting to ensure that all series covered the grain-filling phase (delimited by onset of anthesis and maturity).

The hope was that the CNN could be used for data mining and give insight into aggregation of the input time series, by e.g. looking at the kernel windows that scans the input series and whose weights are optimised during training of the network. The results underlined the need for making more interpretable kernel windows, or some incorporated penalty on the weights, in order to provide a clearer insight in the importance of the features in the inputs. Though well performing, the CNNs could not give enough insight in which, if any, summary statistics to use as inputs. It did show that convolution layers could be used as a pre-processing step. The automatically extracted features coming out of the convolution layer could hence be seen as inputs - despite not being as simple and easily interpreted as e.g. the summary statistics used in the GP-study.

cell coordinates [$\bar{pr}_{120}, \bar{tas}_{120}, \bar{rsds}_{120}$, lat, lon], [$\bar{pr}_{120}, \bar{tas}_{120}, \bar{rsds}_{year}$], [$\bar{pr}_{120}, \bar{tas}_{120}, \bar{rsds}_{year}$, lat] and [$log(\bar{pr}_{120}), \bar{rsds}_{year}$]. The summary statistics $\bar{pr}_{120}$, $\bar{tas}_{120}$ and $\bar{rsds}_{120}$ were taken from the four months (120 days) of growing season also used in the emulator approach in the previous section. It was motivated as a suitable period for the reason that it is close to the average number of growing days for spring wheat. Which 120 days depends on the harvest date, which is a LPJ-GUESS simulator output

2

Data

The neural network emulator to be constructed will be designed to map LPJ-GUESS inputs to the simulated spring wheat yield. The domain considered contains the 672 CTWN combinations[1], the 192 globally spread locations the simulation years 1982-2009[2]. Even with the number of sites and years reduced to these 192 locations and 28 years, respectively denoted $\mathcal{L}$ and $|\mathcal{Y}$, the total number of data samples becomes as great as 3 612 672 (where $|C||T||W||N||\mathcal{Y}||\mathcal{L}| = 4 \cdot 7 \cdot 8 \cdot 3 \cdot 28 \cdot 192$).

The vegetation models in the GGCMI study are to simulate several types of annual crop, like spring wheat, winter wheat and maize, as well as produce some lower prioritized simulator outputs related to that crop - like *total above ground biomass, plant date, days until anthesis* and *days until maturity*. These related crop outputs mentioned can also be of use as emulator outputs - which will be shown later in this thesis.[3]

Inputs used in LPJ-GUESS are the climate variables carbon dioxide (C), precipitation change (W), shift in temperature (T) and nitrogen (N) as well as weather sequences of daily averages of precipitation, temperature and radiation and soil-mixture, where year and the $0.5°$ by $0.5°$grid cell coordinates in latitude and longitude are specified for all samples. As mentioned in section 1.1.1 it is not mandatory to include all simulator inputs in the emulation, but no other inputs than those used in the LPJ-GUESS simulation will be considered. However, the inputs can be transformed and pre-processed in various ways. The inputs and transformations considered in this study are presented in section 2.3.

2.1 Outputs

The wheat simulations have been divided into groups of spring wheat and winter wheat due to their difference in length of growing periods. Winter wheats are often planted during the fall, while spring wheats are planted in the spring, both are harvested between late summer and early autumn. It generally grows faster than winter wheats that require a long period of low temperatures during the vegetative phase (Underwood 2018, p. 89).

Annual *Spring Wheat* is one of the simulated crop responses that are of highest priority for emulation, but other lower priority outputs related to spring wheat were also considered as outputs in this study, namely *total above ground biomass, length of anthesis period* and *maturity season*. These four simulator outputs are further

[1] 672 CTWN combinations are the 4 atmospheric carbon dioxide concentrations (C) levels $360, 510, 660$ and 810 ppm, the 7 sifts in temperature (T) of $-1, 1, 2, 3, 4$ and 6 Kelvin, the 8 relative changes in precipitation (W) at $-50, \pm 30, \pm 20, \pm 10, 0$ percent (excluding the full irrigation case) and the 3 fertilizer levels at 10, 60, and 200 kilos of nitrogen (N) per hectare kgN/ha.

[2] Here, year 1980, 1981 and 2010 was removed from the data set. Some samples planted 2010 and spanned over the following year were not accountable since the whole growing season would not have be covered for those, and for simplicity all observations that year was removed. The first two years were contained many outliers so all samples from those years were removed, also for simplicity, because sample analysis could take time and is not of focus here. Further, locations for which some yield responses were missing in the data set were completely excluded from in the study (hence the reduction from 200 to 192 locations.

[3] They could also be used as inputs, if modeled independently of the yield output, which easily can be done, but in surrogate modelling it is customary to take only simulator inputs as explanatory variables. However, the weather sequence, used as inputs in the LPJ-GUESS, were here cut based on the simulator outputs plant date and maturity season and therefore made the model dependent on simulator date-outputs. But this type of dependence was also introduced in the suggested quadratic regression approach for emulation of individual years as well as in the GP-emulator study where the summary statistics used as inputs where taken from weather time series that ranged from the day of harvest to the longest observed anthesis period, which indeed are a simulator output.

described under the following subsections and their distribution is visualized in figure C.2a. The histograms of the four outputs reviles the similarities in distribution between yield and biomass as well as between the length of the period before anthesis and growing season. In the end, only total above ground biomass was included as an additional output to yield in the final multi-output model. However, the growing season output was used for cutting the daily weather time series.

Annual Spring Wheat Responses to Change in Climate

Annual Spring Wheat, measured in dry matter $ton/(ha \cdot year)$, is simulated for all $3 \cdot 7 \cdot 8 \cdot 4 = 672$ climate scenarios (in the CTWN hyper-cube) considered here, at 192 locations for the 29 years between 1981 and 2009.

Total Above Ground Biomass Responses to Change in Climate

Annual total above ground biomass is, like the corresponding spring wheat output, measured in $ton/(ha \cdot year)$ and simulated for all $3 \cdot 7 \cdot 8 \cdot 4 = 672$ climate scenarios (in the CTWN hyper-cube) considered here, at 192 locations for the 29 years between 1981 and 2009.

Length of Developing Phases

The *pre-anthesis phase*, i.e. the days before anthesis covers the, more established termed, *vegetative phase* (between sowing and floral initiation) and the consecutive *reproductive phase* (Satorre 1999, p. 14). The number of days between sowing and maturity will here be referred as length of the *growing season*, containing both the pre-anthesis phase and the post-anthesis period, conventionally described as the *vegetative phase, reproductive phase* and *grain-filling phase* (Satorre 1999, p. 14).

Growing season was however used for cutting the weather time series so that they all covered the longest observed growing season. These development phase lengths are set to be the same for all climate scenarios and only varies between locations and over time. Length of the pre-anthesis phases and growing season were both considered as outputs in some models, but in the final model not used.

2.2 Inputs

The daily weather data used in the Bachelor's thesis was also used here:

- *Precipitation Flux* (*pr*) denoted $\mathbf{pr}_{i,y}$, measured in kg/m^2.
- *Surface Downwelling Shortwave Radiation Flux* (*rsds*) denoted $\mathbf{rsds}_{i,y}$, measured in W/m^2.
- *Surface Air Temperature* (*tas*) at 2 meter denoted $\mathbf{tas}_{i,y}$, measured in Kelvin.
- *Latitude Coordinate*, $\mathbf{lat}_i$,
- *Soil Mixture* of clay, silt and sand:

$$\mathbf{soil}_i = [clay_i, silt_i, sand_i] \qquad \text{where} \quad clay_i + silt_i + sand_i = 1 \tag{2.1}$$

where i and y denotes the location index and year, respectively. Note that these inputs are the actual observed, hence kept fixed for all points in the the CTWN climate hypercube. The soil vector contains the observed fractions

of clay, silt and sand that sum up to 1. Figure C.1 in *Appendix* C.1 displays how the three processes *rsds*, *pr* and *tas* can behave at a single location, during a year as well as over the longer period between 1980 and 2010.

2.2.1 Climate Adjusted Series

Instead of feeding the same times series for every climate scenario, as if the above mentioned where to be used, it has in the GGCMI study been suggested to transform the precipitation and temperature series according to the shifts along W and T axes in climate hypercube. That is, climate adjusted precipitation $\tilde{\mathbf{p}}$ and temperature $\tilde{\mathbf{t}}$ could be derived from the observed time series of precipitation $\mathbf{p}$ and temperature $\mathbf{t}$ using the climate variables $\mathbf{P} := \frac{\mathbf{W}}{100} = \frac{[-50,-30,-20,-10,0,10,20,30]}{100} = [0.5, 0.7, 0.8, 0.9, 1, 1.1, 1.2, 1.2]$ [4] and $\mathbf{T} = [-1, 1, 2, 3, 4, 6]$ accordingly

$$\tilde{pr}_{w,i,y} = pr_{i,y} \cdot P_w \tag{2.2}$$

$$\tilde{tas}_{t,i,y} = tas_{i,y} + T_t \tag{2.3}$$

for all locations i, years y, climate levels w and t along the TW dimensions in the climate hypercube. Note that each entity (corresponding to e.g a day or month) in these new time series $\tilde{pr}_{w,i,y}$ and $\tilde{tas}_{t,i,y}$ are adjusted by an equal amount. Thus, the climate changes of precipitation and temperature are uniformly distributed over time.

These anomaly adjusted series can generated in the `GetClim` tool, accessed within LPJ-GUESS.

2.3 Pre-Processing of Data

Preprocessing of the data, aiming to reduce skewness and heavy tails, can facilitate construction of the network. The transformations considered here are *power transformations*, that can make left skewed data more symmetric as well as root transformations which are often used for right skewed data.

For inputs that vary in range and/or are measured in different units it is often convenient to apply input normalization. Especially in network modeling, where difference in input ranges can slow down the learning process or even make it unstable. The most common *range-scale method* for real valued inputs is the *standardization*,

$$\frac{x - \mu_{training}}{\sigma_{training}} \tag{2.4}$$

or *z-score normalization* derived from the training samples. Here the mean $\mu_{training}$ and standard deviation $\sigma_{training}$ statistics were also used for standardization of the samples used for validation and final evaluation. It is also possible to use range normalization (Heaton 2015, p. 84), s.a. the *min-max normalization*,

$$\frac{x - min(x_{training})}{max(x_{training}) - min(x_{training})} \tag{2.5}$$

This transformation maps the values of x onto the range between 0 and 1. So in contrast to the z-score normalization, min-max normalization assures a common range for the training data. It should be noted that both methods are sensitive to outliers. (Heaton 2015, pp. 84-86).

[4] The precipitation climate variable W is usually defined with percentage units, i.e. $\mathbf{W} = [-50, -30, -20, -10, 0, 10, 20, 30]$ but here, when used for scaling in the climate equations 6.4 the defined P with fraction units is used.

3
Emulation

Simulations produced by as large and complex systems as LPJ-GUESS are computationally demanding, so when many simulations are to be retrieved, a simpler *emulator* that can approximate the simulators behavior is often of use. Mimicking approximation models also goes under other names like *surrogate models* and *response surface models*. (Balduin 2018, p. 403).

A *simulator* is a *deterministic* model that describes the cause-and-effect between explanatory variables $X = [x_1, x_2, \ldots, x_n]^T$ and corresponding outputs

$$Y_{simulator} = f(X). \tag{3.1}$$

Due to their complexities these are often treated as *black-boxes* (Balduin 2018, p. 403), i.e. that only the inputs and outputs are assumed to be known. An *emulator* trying to model the simulator output will thus consist of a *deterministic* component $f(X)$ and a *stochastic* error term ϵ, explaining the unaccounted dynamics, caused by e.g. observation and modeling errors (Tangirala 2015, p. 3, 17), i.e.

$$Y_{emulator} = f(X) + \epsilon. \tag{3.2}$$

A model trained to learn the relations between the simulator inputs $X = [x_1, x_2, \ldots, x_n]^T$ and outputs $Y_{simulator} = f(X)$ can serve as a computational cheaper version of a simulator, if it is also general enough to produce acceptable estimates of new simulator outputs $Y_{simulator} = f(X_*)$ based on new inputs $X_* = [x_1^*, x_2^*, \ldots, x_n^*]^T$. (Bishop 2006, p. 2; Schmit and Pritchard 2018, p. 1)

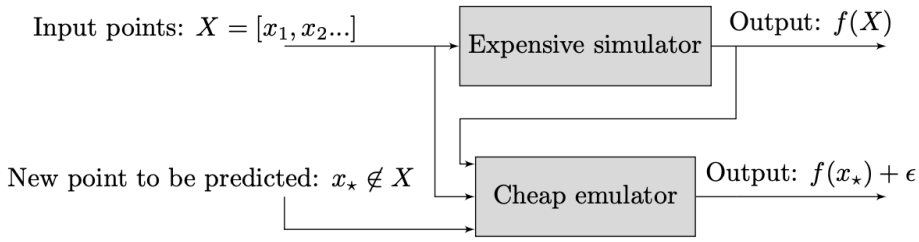


Figure 3.1: The emulator process (Olsson 2017).

An emulator designed to estimate the LPJ-GUESS responses to climate change for all locations (i.e. the $0.5°$ by $0.5°$ grid cells) and the years 1982-2009, the CTWN hypercube of climate scenarios would be extended to the $CTWN \cup \mathcal{Y} \cup \mathcal{L}$ hyperspace, where $\mathcal{Y}$ is the time dimension (with 28 levels) of the years 1982-2009 and $\mathcal{L}$ is the set of $0.5°$ by $0.5°$ grid cells. This is illustrated in 3.1.

A surrogate model can also be a linkage of models corresponding to separate parts of the whole simulator domain, i.e. a *model composition*. (Petty and Weisel 2019). For example, a surrogate model consisting of several location-specific emulators designed to model crop responses to shifts in climate between 1982 and 2009, would require as many emulators as there are locations of interest. Each model domain would be the $CTWN \cup \mathcal{Y}$, i.e the CTWN climate hypercube extended only with time dimension.

Surrogate modeling can be comprised of more than just model design, like the experimental design for *sample selection*. The number of emulator samples (the simulations) are often limited if the simulator is computationally demanding and take long time to produce new outputs. Hence a wise sample selection can yield better emulation results in shorter time. A sample can

be chosen initially or optimised iteratively along with emulation model reruns. The latter is often used in surrogate modeling of *black-box functions* that are expensive to run and can be implemented trough e.g. *Bayesian Optimisation* (see B.3).

4
Neural Network Architectures

Neural Network (NN) modeling is an algorithmic machine learning approach to taking actions by parsing data and can massively parallel process data using many types of well established machine learning and statistical techniques. (Bishop 1995; Waszczyszyn 1999, p. 7). A network, modeling the relationship between inputs and outputs, often consist of an initial *layer* accepting the input and forwards it to a "black box" of *hidden layers*, which in turn is connected to final layer that returns the output (Heaton 2015, p. Preface, xxxv). The black box can consist of one or several hidden layers containing few or many connected *neurons*.

A layer can either take all of the received inputs simultaneously, one or a fraction at a time and the transforms can take different forms but normally involves a set of trainable weights and a consecutive wrapper, or *activation function*. The activation function can be of many forms, some of which are mentioned in section 4.5, and often have the important task of introducing non-linearities to a neural network.

The network architecture, i.e. the number of layers and nodes, type of layers and activation function, etc., depends on the data and problem at hand. A brief description of the layers considered in this study is given under Fully Connected Layers 4.1, Convolution Layers 4.2 and Pooling Layers 4.2.1.

The neural network architectures considered for this problem has to preserve a special structure in order for it to map a set of mixed data inputs to a set of multiple outputs. The section *Branched Neural Networks for Mixed Data Inputs* describes how a network can deal with multiple inputs and different data types. The following section *Multitask Neural Networks* describes how a network can be extended to produce estimates of multiple outputs.

4.1 Fully-Connected Feed-Forward Neural Networks

The *Fully-Connected Feed-Forward Neural Network* (FFNN), is the the most basic form of *Artificial Neural Networks* (ANN) and consist only of *fully-connected layers* (Schmidhuber 2014). These fully-connected layers, a.k.a. *dense layers*, consist of operating neurons, each corresponding to a set of trainable weights which they use to transform the received inputs, by computing their dot product (element-wise). A bias weight is then added to this weighted sum, before it gets wrapped in an activation function and passed forward to the next layer. Figure 4.1 illustrates the operation inside a neuron. (Waszczyszyn 1999, p. 4, 6–7).

4.1.1 FFNNs for Regression Problems

A neural network with trainable weights w that accepts inputs x to model an output y, can be defined as an approximator of the mean of the conditional distribution $p(y \mid x, w)$, and hence be formulated as a regression problem.

Linear regression models on the form

$$f(x, w) = w_0 + \sum_{i=i}^{N} w_i x_i = \mathbf{x}^T \mathbf{w} \tag{4.1}$$

are often used for modeling responses based on some explanatory variables. Note that $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$ where x_0 is a dummy variable always equal to one, introduced in order to absorb w_0 into the set of weight parameters.

This regression model fails when the relation between the true function and the dependent variables are not linear (Hastie 2009, p. 139). One can move beyond such simple linear models, by letting the function be a linear combination of nonlinear

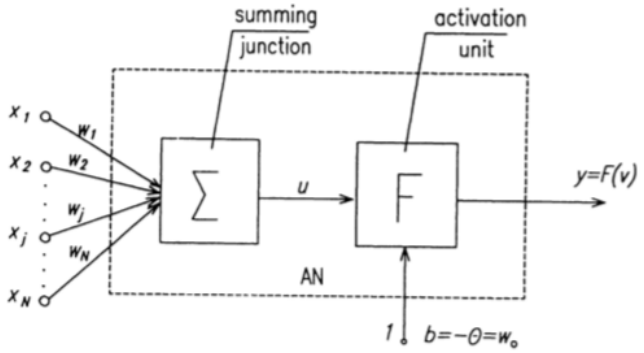


Figure 4.1: Operations inside a neuron. $x_1, x_1, ..., x_N$ are either the input data or outputs coming from neurons in a previous layer. The summoning junction produces the the dot product of the received inputs and node weights u. The activation unit wraps u in an activation function F. (Waszczyszyn 1999).

basis functions

$$f(x, w) = w_0 + \sum_{i=i}^{N} w_i \phi(x_i) = \phi(\mathbf{x})^T \mathbf{w} \tag{4.2}$$

This function is nonlinear in the input variables $\mathbf{x}$, but linear in the basis functions $\phi(\mathbf{x})$ and can hence be seen as a linear model in the space spanned by these nonlinear functions. (Bishop 2006, p. 137-139)

Assume that the first hidden network layer has M_1 neurons, each producing a linear combination of the set of input variables $\{x_i, i = 1, ..., N\}$ which form the first *activations*,

$$a_j^{(1)} = w_{j0}^{(1)} + \sum_{i=1}^{N} (w_{ji}^{(1)} x_i) = \sum_{i=0}^{N} (w_{ji}^{(1)} x_i) = \mathbf{x}^T \mathbf{w_j^{(1)}} \tag{4.3}$$

for $j = 1, ..., M_1$, where $w_{ji}^{(1)}$ is the weight corresponding to the input variable x_i connected to the the j:th neuron in the first layer, and $w_{j0}^{(1)}$ is the bias added in the j:th neuron. The initial activations $a_j^{(1)}$ are then transformed by using a nonlinear activation function $h^{(1)}(.)$ to *hidden units*

$$z_j^{(1)} = h^{(1)}(a_j^{(1)}) \tag{4.4}$$

These hidden units can be seen as basis functions $\phi(.)$ that, in contrast to the fixed basis functions in equation 4.2, depend on adaptive model parameters. In fact, for any layer H, every hidden unit $h^{(L)}(a_j^{(L)})$ can be expressed as bias function $\phi(\mathbf{x}, \mathbf{w^{(1)}}, \ldots, \mathbf{w^{(L)}})$ that depend on the initial inputs $\mathbf{x}$ and the network weights in the current and foregoing layers (i.e. $\mathbf{w^{(L)}}$ resp. $\{\mathbf{w^{(1)}}, \ldots, \mathbf{w^{(L-1)}}\}$).

If the inputs $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$ are to be mapped to the outputs $\{y_k, k = 1, 2, ..., K\}$ by a single-hidden-layer network, which has the total of two layers, the first hidden units $z_j^{(1)} = h^{(1)}(a_j^{(1)}), j = 1, \ldots, M_1$ are assigned the final weights to produce K *output unit activations*

$$a_k^{(2)} = \sum_{j=0}^{M_1} (w_{kj}^{(2)} z_j) \tag{4.5}$$

These are then transformed to form the final network outputs, by applying a suitable output activation function $\sigma(.)$,

$$\hat{y_k} = \sigma(a_k^{(2)}) \tag{4.6}$$

Note that the final activation function is like any other (hidden) activation, but here the hidden activation functions were denoted with $h(.)^{(layer)}$ to emphasize their respective distinction.

For a *deep neural network*, i.e. one with multiple hidden layers, each hidden unit $z_j^{(H)}$ in the H^{th} layer is assigned a weight $w_{kj}^{(H)}$ for every neuron k it is connected to in layer $H+1$. If the number of hidden neurons in layer $H+1$ is equal to $M_{(H+1)}$, the summoning junction results in $M_{(H+1)}$ weighted linear combinations $a_k^{(H+1)} = \mathbf{z^{(H)}}^T \mathbf{w_k^{(H+1)}}$. These activations are then wrapped in non-linear activation functions $h^{(H+1)}(.)$ forming the hidden units $z_k^{(H+1)}$ passed forward to the next layer. This process is repeated until the final output unit activations are formed.

The final output units, like the hidden ones, accepts the connecting input units from layer $L-1$ and their corresponding weights, passes them and the bias w_0 through the summoning junction

$$a_k^{(L)} = \sum_{j=0}^{M_{L-1}} (w_{kj}^{(L)} z_j^{(L-1)}) \tag{4.7}$$

for $k = 1, 2, ..., K$ where j denotes the neuron in the L-1th layer. The final output unit activations $a_k^{(L)}$ are then transformed into the network outputs units by applying a suitable output activation function $\sigma(.)$. That is,

$$y_k = \sigma(a_k^{(L)}) \tag{4.8}$$

In the case of regression problems, this function is usually set to be the identity in order to introduce the linear relationship described by 4.1.1. (Bishop 2006, p. 227-229; Nielsen 2015, ch. 2).

4.2 Convolutional Neural Networks

A neural network with *convolution layers*, convolving received inputs with discrete kernels of trainable weights, are referred to as *Convolutional Neural Network* (CNN) (Bishop 2006, ch. 5.5.6; (Chollet 2018, ch. 5)). CNNs have become a popular tool for feature recognition and forecasting in weather and climate related problems. For example, a convolutional neural network performing 2D convolution operations, can take a weather map to extract spatial features. If temporal features are also of interest, the map can be extended to a third time dimension and 3D convolutions can be used to detect spatiotemporal relations. (see Larraondo and Lozano 2017, Klein and Afek 2015 and Chattopadhyay and Pasha 2018). In this project however, rather than finding spatial or temporal features on a world map, the interest lies in finding patterns in daily weather time series that can help predict annual wheat production. A *1D convolution layer* that scans the received time series can achieve just that.

The 1D convolution layer in this project will accept a 2D tensor with C *channels* or *input features* containing different weather series, derived from the daily observations of precipitation (*pr*), temperature (*tas*) and radiation (*rsds*) given under 2.2.

$$\mathbf{x} = [\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_C}]^T = \begin{bmatrix} x_{11} & x_{12} & x_{13} & \ldots & x_{1T} \\ x_{21} & x_{22} & x_{23} & \ldots & x_{2T} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{C1} & x_{C2} & x_{C3} & \ldots & x_{CT} \end{bmatrix} \tag{4.9}$$

A convolution layer consist of a pre-defined number of filters, or kernel windows with fixed height and and depth dimension equal to the number accepted input channels. A convolution layer receiving C input channels will thence have the following kernel windows:

$$\begin{matrix} [w_{1,1,1}, w_{1,1,2}, \ldots, w_{1,1,K}] & [w_{1,2,1}, w_{1,2,2}, \ldots, w_{1,2,K}] & \ldots & [w_{1,c,1}, w_{1,c,2}, \ldots, w_{1,c,K}] \\ [w_{2,1,1}, w_{2,1,2}, \ldots, w_{2,1,K}] & [w_{2,2,1}, w_{2,2,2}, \ldots, w_{2,2,K}] & \ldots & [w_{2,c,1}, w_{2,c,2}, \ldots, w_{2,c,K}] \\ \vdots & \vdots & \ddots & \vdots \\ [w_{F,1,1}, w_{F,1,2}, \ldots, w_{F,1,K}] & [w_{F,2,1}, w_{F,2,2}, \ldots, w_{F,2,K}] & \ldots & [w_{F,c,1}, w_{F,c,2}, \ldots, w_{F,c,K}] \end{matrix}$$

where F is the number of kernel strings of length K. All C channels corresponding to F different kernel windows contain a total of $F \cdot C \cdot K$ number of weights.

Every channel in the kernel windows will scan the corresponding feature sequence over the time dimension as illustrated in figure 4.2.

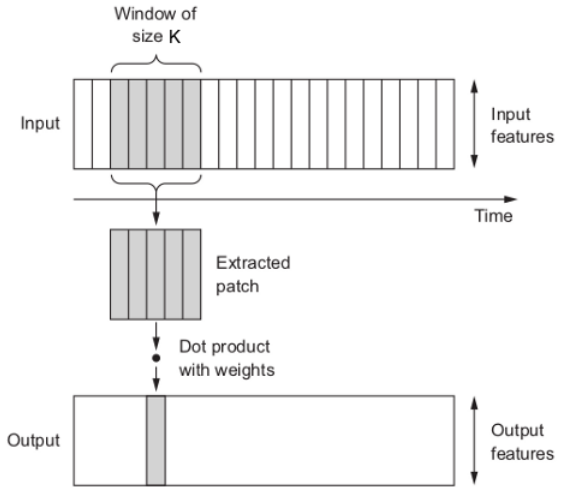


Figure 4.2: The convolution window of size K scans the input of height T and depth C and step by step produces weighted dot products that together constitutes the 2D tensor output of height T_C and depth F. (Biney 2018)

For every filter $f = 1, 2, \ldots, F$, the layer takes the (discrete) convolution of the window and a input series sub-interval, of K subsequent entities at a time, for the C channels in parallel. At every time step t, it produces one convolution output

$$\sum_{k=1}^{K} x_{c,(t+k-1)} w_{f,c,k}^{(1)} \tag{4.10}$$

for every channel c, which are summed together in similar fashion to a dense layer,

$$\begin{aligned} a_{f,t}^{(1)} &= b_f^{(1)} + \sum_{c=1}^{C} \sum_{k=1}^{K} x_{c,(t+k-1)} w_{f,c,k}^{(1)} \\ &= b_f^{(1)} + \sum_{c=1}^{C} (x_{c,t} w_{f,c,1}^{(1)} + x_{c,(t+1)} w_{f,c,2}^{(1)} + \ldots + x_{c,(t+K-1)} w_{f,c,K}^{(1)}) \\ &= b_f^{(1)} + \sum_{c=1}^{C} \left[w_{f,c,1}^{(1)}, w_{f,c,2}^{(1)}, \ldots, w_{f,c,K}^{(1)} \right] \left[x_{c,t} + x_{c,(t+1)}, \ldots, x_{c,(t+K-1)} \right]^T \end{aligned} \tag{4.11}$$

The superscript $^{(1)}$ just refers to that the weights and entities are that of the first layer and $b_f^{(1)}$ is a trainable bias assigned to all sums produced by kernel weights in filter f. This is calculated for every time point t=l, 2, ..., T_C, meaning that for the window jumps forward one step and repeats the process until the whole time series have been scanned by the kernel window. The size of the jumps is called (window) stride S (see below).

When the time series have been scanned by all filters, F *output features*, or *feature maps*, have been produced,

$$\mathbf{a}_f^{(1)} = \left[a_{f,1}^{(1)}, a_{f,2}^{(1)}, \ldots, a_{f,T_C}^{(1)} \right], \qquad f = 1, 2, \ldots, F \tag{4.12}$$

and together they form a 2D tensor of height T_C and depth F. T_C is the (new) length of the time dimension.[1]

[1]This yields also if zero padding is used, where the added zeros also can be denoted as above, but with $a_{c,t*}$ for t*= P_-, $P_- + 1$, ... ,

The length of the feature maps T_C, and thereby the number of outputs, is determined by the *stride* S between each convolution computation as well as *zero padding*, by

$$T_C = \frac{T - K + P}{S} + 1.$$

where the number of zeros added before and after the input time string is P. The final model in this thesis had stride $S = 1$ and used no *zero padding*, $P = 0$. By equation 4.2, the length of the feature maps is therefore

$$T_C = T - K + 1.$$

The dot products 4.2 (stacked in a 2D tensor of height T and depth F) can be passed through an activation unit, where they are activated by a non-linear wrapper $h(.)$, i.e.

$$z_{f,t}^{(1)} = h(a_{f,t}^{(1)}) \tag{4.13}$$

for all filters $f = 1, 2, ..., F$ and time units $t = 1, 2, ..., T_C$. (Goodfellow 2016, p. 338). These convolution layer outputs form the following 2D tensor

$$\mathbf{Z}^{(1)} = \left[\mathbf{z_1^{(1)}}, \mathbf{z_2^{(1)}}, \ldots, \mathbf{z_F^{(1)}}\right]^T = \begin{bmatrix} z_{11}^{(1)} & z_{12}^{(1)} & z_{13}^{(1)} & \cdots & z_{1T_C}^{(1)} \\ z_{21}^{(1)} & z_{22}^{(1)} & z_{23}^{(1)} & \cdots & z_{2T_C}^{(1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ z_{F1}^{(1)} & z_{F2}^{(1)} & z_{F3}^{(1)} & \cdots & z_{FT_C}^{(1)} \end{bmatrix} \tag{4.14}$$

Interpretation of What CNNs Learn

The kernel weights in a trained convolutional neural network which exposes the weather patterns the CNN is looking for. The convolution, or dot product, will be large if the sub-interval being scanned follows a similar pattern as the one in the scanning window. Since kernel weighted layer inputs are added, the sum is largest when all weather inputs simultaneously behaves like their corresponding weights in the feature map.

In this thesis, mainly the the *Rectified Linear Unit* (ReLU) activation function $\sigma_{ReLU}(.) = max(0, x)$ will be considered for the activation units and is properly presented in *ReLU and Other Activation Functions* 4.5. By using this ReLU wrapper we can find the patterns that the network looks for by investigating the filters and biases. The biases determines how large the sum of the feature channel dot products must be, to not be set to zero by the ReLU activation function in the convolution layer. I.e

$$z_{f,t} = \sigma_{ReLU}(a_{f,t} = \begin{cases} a_{f,t} & \text{if } a_{f,t} > 0 \quad \iff \sum_{c=1}^{C}\sum_{k=1}^{K} z_{c,(t+k-1)} w_{f,c,k} > -b_f \\ 0 & \text{otherwise} \quad \iff \sum_{c=1}^{C}\sum_{k=1}^{K} z_{c,(t+k-1)} w_{f,c,k} \leq -b_f \end{cases} \tag{4.15}$$

Hence, the biases could be seen as a feature thresholds for how well the input entities must follow the pattern in the convolution window, to be accounted for.

If a bias b is positive, i.e. $b = c$ for some constant $c > 0$, then the output will be included (not be set to zero by the ReLU function) for all values larger than -c. If the bias is negative, i.e. $b = -c$, then the outputs must be larger than c to be included.

The following section presenting *Pooling Layers* and how these transform convolution layer outputs, explains how this interpretation can be translated to pooling layer outputs and in *Dense Layers In CNNs* 4.2.3 this is even further extended.

---

l, 2, ..., T, ..., T +P_+, where P_- is the number of initial zeros and P_- number of zeros added in the end. I.e. $a_{c,t*}$ is equal to $a_{c,t}$ when $t* = t.$, and set to zero for $t* >\mid t \mid$.

4.2.1 Pooling Layers

Convolution layers are often followed by a *pooling layer* that takes a fraction, if not all, of the convolution layer outputs and returns a *summary statistic*.

A *pooling layer* neither contains a summoning junction nor an activation function, it only slides a window across the received tensor and replaces the entities in each window with some summary statistic. In that way it reduces the output size, its complexity and deals with problematic observations. Common pooling layers are the *max pooling layer* that takes the maximum of each window being scanned and the *average pooling layer* that computes their averages. (Goodfellow 2016, p. 330).

The output size can be derived from the same formula used for calculating the convolution layer output length in equation 4.2. E.g., if a pooling layer with pool window size $K = W$, stride S that accepts a 2D tensor with feature channels of length T and adds P zeros at the beginning and the end, then the pooling layer output for each feature channel, will be of length

$$\frac{T - W + P}{S} + 1.$$

This implies that the outputs from a pooling layer with no padding and no overlapping scanning windows (that is, stride $S = W$) will be of length

$$\frac{T - W + 0}{W} + 1 = \frac{T - W}{W} + 1 = \frac{T}{W}$$

Average Pooling Layer

If an *average pooling layer*, with no padding and no overlapping of scanning windows, accepts the 2D tensor 4.14, then it will produce a C channeled tensor of length $\frac{T}{W_{ave}}$ (according to equation 4.2.1), where W_{ave} is the length of the average pooling windows.

$$\mathbf{Z}^{(2)} = \left[\mathbf{z_1^{(2)}}, \mathbf{z_2^{(2)}}, \ldots, \mathbf{z_F^{(1)}}\right]^T = \begin{bmatrix} z_{11}^{(2)} & z_{12}^{(2)} & z_{13}^{(2)} & \cdots & z_{1,\frac{T_C}{W_{ave}}}^{(2)} \\ z_{21}^{(2)} & z_{22}^{(2)} & z_{23}^{(2)} & \cdots & z_{2,\frac{T_C}{W_{ave}}}^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ z_{F1}^{(2)} & z_{F2}^{(2)} & z_{F3}^{(2)} & \cdots & z_{F,\frac{T_C}{W_{ave}}}^{(2)} \end{bmatrix} \tag{4.16}$$

The superscript $^{(2)}$ is a denotation for the layer, which here is the second following the first convolution layer in the example above. Each entity $z_{cj}^{(2)}$ in this 2D average pooling output tensor is derived as such

$$z_{cj}^{(2)} = \frac{1}{W_{ave}} \sum_{i=(j-1)W_{ave}+1}^{jW_{ave}} x_{ci} \tag{4.17}$$

for $c = 1, 2, \ldots, C$ and $j = 1, 2, \ldots, \frac{T_C}{W_{ave}}$.

Recall from the end of section 4.2, that when a filter scans the time series, if a matching to the window weights is detected, the convolution will increase. Each entity coming from an average pooling layer, following a convolution layer, can thus revile to which extent the inputs followed the patterns in the scanning filter *on average*, within a time interval of $K + W_{ave}$ (if the stride in the convolution layer is equal to one and the stride in the pooling layer is equal to the size of the pooling window, i.e. W_{ave}).

4.2.2 Local Equivariance and Invariance

By letting the f^{th} kernel window channel c $[w_{f,c,1}, w_{f,c,2}, \ldots, w_{f,c,K}]$ scan the the whole input sequence c $z_{c,1}, z_{c,2}, \ldots, z_{c,T}$ it finds a specific pattern whenever it appears between the first and the last time step and will produce the exact same convolution product. In other words, convolution layers have the property *equivariant under translation* T, where *equivariance* is

defined as (Goodfellow 2016, p. 329-330)

$$f(T(X)) = T(f(X)). \tag{4.18}$$

Further, a function f is said to be *invariant to translations* T if

$$f(T(X)) = f(X) \tag{4.19}$$

A CNN, whose convolution outputs are replaced with summary statistics computed by a pooling layer, can be trained to become *approximately invariant to small local translations* in the input data. In this context meaning that the pooling layer output will not change by much if a feature is delayed a bit. E.g. a returned summary statistic of outputs corresponding to different time intervals (overlapping or not), will not be highly affected by which of the intervals a feature was detected in.(Goodfellow 2016, p. 331-331).

4.2.3 Dense Layers in CNNs

The convolution layer outputs will be passed through a dense layer in order to produce estimates of the LPJ-GUESS outputs, whether the convolution outputs are passed through a pooling layer (or several subsequent convolution and pooling layers) first or not. Dense layers can only take 1D tensors, so in order for it to transform a 2D tensor (coming from a previous 1D convolution or pooling layer) it must first be column stacked.

Assume we want to process the $\tilde{T}\times$ 2D tensor coming from layer L-1 (either a 1D convolution layer, with $\tilde{T} = T_C$ or a pooling layer, with $\tilde{T} = \frac{T_C}{W_{ave}}$,),

$$\mathbf{Z}^{(L-1)} = \begin{bmatrix} z_{11}^{(L-1)} & z_{12}^{(L-1)} & z_{13}^{(L-1)} & \cdots & z_{1\tilde{T}}^{(L-1)} \\ z_{21}^{(L-1)} & z_{22}^{(L-1)} & z_{23}^{(L-1)} & \cdots & z_{2\tilde{T}}^{(L-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ z_{F1}^{(L-1)} & z_{F2}^{(L-1)} & z_{F3}^{(L-1)} & \cdots & z_{F\tilde{T}}^{(L-1)}. \end{bmatrix} \tag{4.20}$$

Then $\mathbf{Z}^{(L-1)}$ would have to be passed through a flattening layer (the L^{th} layer), producing the column stacked version,

$$\begin{aligned} \mathbf{Z}^{(L)} &= \left[z_{11}^{(L-1)}, z_{12}^{(L-1)}, \ldots, z_{1\tilde{T}}^{(L-1)}, \quad z_{21}^{(L-1)}, z_{22}^{(L-1)}, \ldots, z_{2\tilde{T}}^{(L-1)}, \quad \ldots, \quad z_{F1}^{(L-1)}, z_{F2}^{(L-1)}, \ldots, z_{F\tilde{T}}^{(L-1)}\right]^T \\ &:= \left[z_1^{(L)}, z_2^{(L)}, \ldots, z_{\tilde{T}}^{(L)}, z_{\tilde{T}+1}^{(L)}, z_{\tilde{T}+2}^{(L)}, \ldots, z_{2\times\tilde{T}}^{(L)}, \ldots, z_{(F-1)\times\tilde{T}+1}^{(L)}, \ldots, z_{F\times\tilde{T}}^{(L)}\right]^T. \end{aligned} \tag{4.21}$$

This column stacked output $\mathbf{Z}^{(L)}$ could then be passed through a dense layer and all its nodes. I.e as shown in figure 4.1, get passed through the summoning junction of and assigned a bias accordingly,

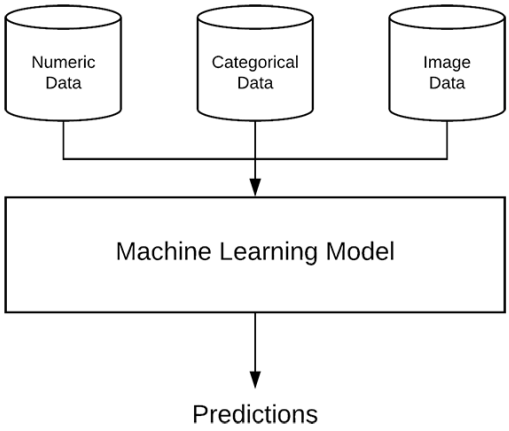
$$\begin{aligned} a_i^{(L+1)} &= w_i^{(L+1)}0 + \sum_{j=0}^{F\tilde{T}} w_{ij}^{(L+1)} z_j^{(L)} \\ &= \sum_{j=0}^{\tilde{T}} w_{ij}^{(L+1)} z_j^{(L)} + \sum_{j=\tilde{T}}^{2\tilde{T}} w_{ij}^{(L+1)} z_j^{(L)} + \cdots + \sum_{j=(F-1)\tilde{T}}^{F\tilde{T}} w_{ij}^{(L+1)} z_j^{(L)} \end{aligned} \tag{4.22}$$

for every node $i = 1, 2, \ldots, I$. The separation in the second line explicate how the entities are connected to the channels in the 2D tensor: the first sum is connected to the first channel, the second sum to the second channel, and so on.

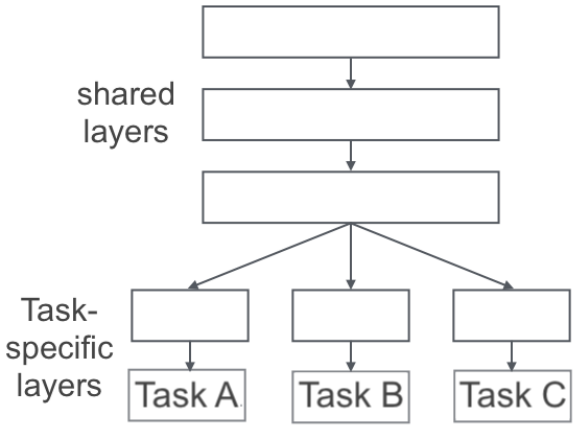
Using the same reasoning as in 4.2, that continued in 4.2.1: since the flattened outputs received by a dense layer can be seen as its explanatory variables, the dense layer weights reviles the effect of finding a specific pattern in a specific time interval.

4.3 Branched Neural Networks for Mixed Data Inputs

If a network is to take mixed data, i.e. multiple inputs of different data types it is preferable, if not crucial, to pass them through separate initializing branches designed for the respective inputs. These branches can take several forms, e.g. be composed by one or many layers, and each can be uniquely designed for the data type to be accepted.



(a) A branched neural network can accept inputs of different data types. Figure taken from PyImageSearch blogpost (Rosebrock 2019)



(b) A multitask network initiated with three shared layers, in which the weights benefit from the shared representations between tasks, and then divided into three separate task specific branches. Figure from (Ruder 2017, p. 2).

Figure 4.3: Figure (a) illustrates a branched neural network and figure (b) how shared layers in a network can be split into separate task-specific branches.

4.4 Multitask Neural Networks

Multitask neural networks can perform several tasks based on the same input data samples. I.e. instead of producing one output for every point in the model domain (like a sequential neural network), it can produce several outputs. The different tasks can, besides sharing inputs, also share generic parameter weights. Such multiple output neural networks often consist of both shared generic parameters and task-specific parameters, as in figure 4.4b illustrating *hard parameter sharing*. (Goodfellow 2016, p. 237).

Generalization of a task, like prediction of the high priority output annual spring wheat, can be greatly improved through simultaneous multitask learning of related tasks, like above ground biomass and length of different wheat developing phases. The number of training samples needed for pressuring the weights to values that makes the network generalized better, could be heavily reduced if a single-output network was forced to learn other tasks as well, because of the gain in statistical power of the shared parameters in the extended multi-task network. (Baxter 2011, p. 150; Ruder 2017, p. 1)

Multitask learning of length of the related outputs pre-anthesis and growing season may help the network learn some features better than a single-output network only predicting yield could; if the importance of those features were to be more evident in relation to those tasks than for the main task to estimate yield (Ruder 2017). The biomass output behaves similarly to the yield outputs and might therefore not have this effect on learning, or at least not to the same extent as the phase-length outputs. However, the biomass simulations presumably have different noise patterns than simulated yield, so simultaneous learning may average out their noise and help the network to become more general (Ruder 2017).

For more information, I suggest reading *"An Overview of Multi-Task Learning in Deep Neural Networks"* (Ruder 2017), that summaries the effects of learning related tasks and provides additional references.

4.5 ReLU and Other Activation Functions

As mentioned, nonlinear activation can be used to wrap the layer operations and are especially useful when the relations between the inputs and outputs are not obvious. The *Rectified Linear Unit* (ReLU) function is a simple and widely used activation function in both convolution layers and dense layers.

$$\sigma_{ReLU}(x) = max(0, x) \tag{4.23}$$

It introduces nonlinearities by setting all negative activation inputs to zero. Other commonly used activation functions are the *Sigmoid* activation function $\sigma_{Sigmoid}(x) \mapsto [0, 1]$

$$\sigma_{Sigmoid}(x) = \frac{1}{1 + e^{-x}}, \tag{4.24}$$

the *hyperbolic tangent*, usually termed *tanh* activation, $\sigma_{tanh}(x) \mapsto [-1, 1]$

$$\sigma_{tanh}(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{4.25}$$

The ReLU activation has the perk of being sparsely activated, since negative values are set to zero and is otherwise cheap to compute, unlike e.g. the tahn or the sigmoidal activation function that require an exponential computation. It has the derivative

$$\sigma'_{ReLU}(x) = \begin{cases} 1 & \text{if x} < 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.26}$$

which in does not raise the problem of *vanishing gradients* caused by multiplication of many very small derivatives during backpropagation, in contrast to the other two activation functions that run a greater risk of causing this a problem. (Heaton 2015, p. 124)).

In fact the ReLU activation function is less likely to raise any problems during training due to the one-directional saturation. Further, hard saturation at zero have shown to help supervised deep neural networks to learn (Glorot and Bengio 2011, pp. 318–319), given that some units are activated. However, though one of the perks is the sparsely activation, when a ReLU node gets stuck and never gets activated for any input, it becomes a so called *dead state*, though which no gradients can flow back and if too many ReLU nodes are set to zero a type of vanishing gradient problem can in fact occur. (Maas 2013, p. 2; Glorot and Bengio 2011, pp. 318-319). This *dying ReLU* problem can however be evaded through some adjustments of the ReLU function as well as of the learning rate. These adjustments are presented under the respective sections *Leaky ReLU and Thresholded ReLU* 5.2.1 and *Learning Rate* 5.2.7.

ReLU As Final Wrapper

As mentioned in 4.1.1, the output activation function of a regression network is the identity, meaning that the output activation unit will also become the output. However, since the network ought to produce non-negative estimates of the non-negative simulator outputs yield, biomass and number of days until anthesis and until maturity, ReLU might serve better as a final wrapper of the network output. The ReLU and identity function are very similar and become even more so as the precision of the summoning junction (i.e. the weighted sums that the activation function receives (see figure description 4.1) in the final layer improves - since most of them then would be positive and the other negative estimates should not stretch that far below zero.

4.6 Implementation in Keras

Neural networks can be constructed using the modular library *Keras* (*(2015) Keras*). Here a Keras implementation was run top of the open source software library *TensorFlow* (Abadi et al. 2015) was used. In order to implement a branched multitask network in Python, *Keras functional API* was used[2] and the NN branches were constructed using the *Keras core layers* `Dense`, `Conv1D` and `AveragePooling1D`.

Keras is very user friendly and allows for writing customized layers and functions that can be implemented together with already defined. It also includes built-in methods for training and evaluation of neural networks. Those used in this thesis are presented in the next chapter 5 *Training of a Neural Network* .

[2]Branched Multitask NNs can not be constructed in the commonly used sequential model type.

5
Training of a Neural Network

During training a neural network can learn how to act by passing forward training input samples through the network, evaluate how well it estimates the corresponding training response and finally adjust the weights according to the level of satisfaction of the returned output, as measured by the *loss function*. In other words: the loss produced by this *objective function* to be minimized is used as a *feedback signal* directing the weight adjustments. (Chollet 2018, p. 11). Figure 5.1 illustrates this iterative training loop.

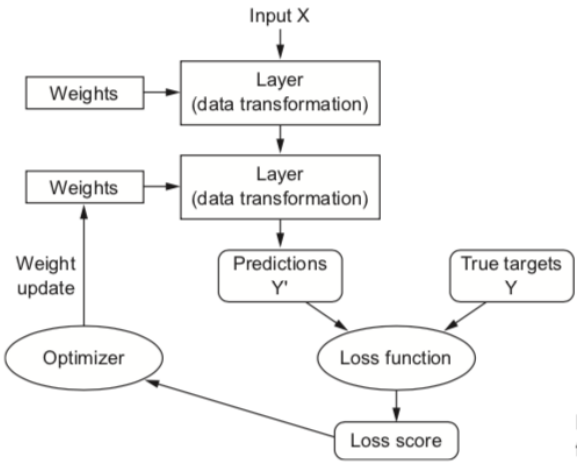


Figure 5.1: Illustration of how deep neural networks learn. Figure from *Deep Learning with Python* by François Chollet (2018).

Usually the training data is divided into smaller *batches* for which the loss function is calculated and fed back to the network, and minimized with respect to the weights and biases. This loss minimization is done iteratively for all batches. (Heaton 2015, pp. 125-126).

The losses from the batches creates an *error surface* of which the (local) minimum is to be found during training. This can be done by moving along the error surface in the opposite direction of its *gradient*, which gives the direction of the steepest ascent. Since the losses are functions of the network outputs, which are produced by the outputs of previous layers, that in turn are functions of the network weights and biases, the error surface can be differentiated with respect to the weight parameters. Thus the gradient of the error surface can be composed by a vector whose components are its *partial derivatives with respect to the weight parameters*, each giving the direction of the steepest increase of the error surface.

The method of calculating the gradient of the error surface is often referred to as *backpropagation* and can be seen as a way of "going back" in the network to find each neurons contribution to the error. (Heaton 2015, pp. 114-118; Nielsen 2015, ch. 2). Backpropagation can thus be used in the optimisation of the error surface that aims to find the weights and biases, i.e. model parameters, that minimizes the loss.

There exists several *optimisation algorithms*, many of which are gradient based (Goodfellow 2016, ch. 4.3). One commonly used *gradient based optimisation algorithm* is the *adaptive moment estimation* backpropagation algorithm *Adam* (Kingma and Ba 2014), that was specifically designed for training of deep neural networks and works efficiently in a high dimensional parameter space. Unlike the *stochastic gradient descent* algorithm that keeps the *learning rate*, determining the step size along the error surface (hence also the weight adjustments), fixed for all weight updates, *Adam* maintains and adapts the learning rate

for each weight and does so using estimates of both the first and second order moments of gradients (i.e. mean and unshifted variance) derived during training. It was the only optimisation algorithm considered in this project and can be implemented using `Adam` in Keras (*(2015) Keras*). The pseudocode for the Adam algorithm is appended in A.1. More on the Adam optimiser and the benefits of having adaptive learning rates is found in section 5.2.7, under *Some Practical Considerations For Training* 5.2.

5.1 Loss Function

Adequacy of a loss functions can be expressed in terms of its ability to match the actual harm the given observation would cause. It can also regard the loss functions mathematical properties and its effect on the gradient optimisation.

The widely used *Mean Squared Error Loss* (MSE) is a common choice for a neural network loss function. If our target $y_n, n = 1, 2, \ldots, N$ follows a normal distribution $N(y \mid f(x, w), \frac{1}{\beta})$ and the corresponding input variables $x_n, n = 1, 2, \ldots, N$ are i.i.d., then the weight parameters that minimises the MSE are equal to the maximum likelihood estimates, i.e. the parameter values for which the response value is the most likely to occur, given the set of observations. This since maximizing the likelihood function of the target given the data and the weights and hyperparameters

$$P(y \mid x, w, \theta),$$

is equivalent to minimizing its negative logarithm (Bishop 2006, p. 9)

$$-ln(P(y \mid x, w)) = -\sum_{n=1}^{N} \ln(p(y_n \mid x_n, w)) \tag{5.1}$$

If the normality assumption holds the negative likelihood function is equal to

$$\frac{\beta}{2}\sum_{n=1}^{N}(f(x_n, w) - y_n)^2 - \frac{N}{2}\ln(\beta) + \frac{N}{2}\ln(2\pi) \tag{5.2}$$

which depends on the weight parameters only through the sum of squares. The maximum likelihood parameters is thus,

$$w_{ML} = argmin \sum_{n=1}^{N}(f(x_n, w) - y_n)^2 = w_{MSE} \tag{5.3}$$

This also gives the ML estimate of the precision hyperparameter Beta (Bishop 2006, p. 29, 233), as

$$\frac{1}{\beta_{ML}} = \frac{1}{N}\sum_{n=1}^{N}(f(x_n, w) - y_n)^2. \tag{5.4}$$

5.1.1 Robust Loss Function

When dealing with non-normal distributions, the mean squared error (MSE) can lead to bad optimisation since it is derived from Gaussian assumptions. This quadratic loss function can make the modeling heavily influenced by the outliers as well as cause overfitting problems and thus raise the need for more robust loss functions. (Bishop 2006, p. 277).

Mean Absolute Error (MAE), like the MSE, measures the magnitude of the errors and does not take direction into account. The main differences between the two is that the squared loss is more sensitive to outliers and that their gradient behaves differently in the training of a neural network. The MAE loss has constant gradient, making it harder to find the minimum during training, compared to the gradient of the quadratic loss which gets smaller the closer we get to the minimum.

If robustness against outliers is desired, it is possible to use the *log-cosh loss* function. It treats errors much like the MAE loss, but are smoothed around zero and hence eases the problem of finding the minimum. Another function that share

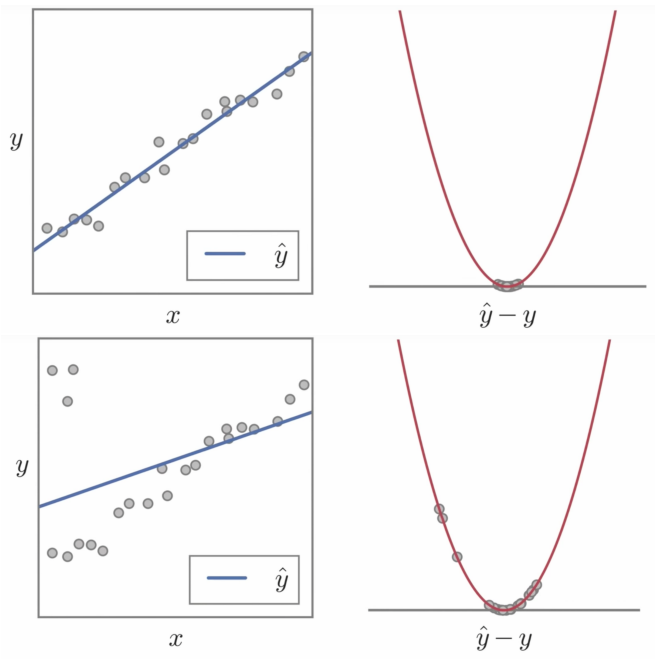


Figure 5.2
MSE is a common choice for a neural network loss function, but can result in bad modeling when we have outliers in the data. The figure shows to data sets modeled with simple linear regression: one with well behaving data and one case with outliers. The predictions $\hat{y}$ is much worse in the second case, even for some well behaving samples, as consequences of that MSE penalizes outliers quite hard. Screenshots from video posted by Barron (2019).

properties of the squared and absolute loss is the Huber Loss, which is a composition of the two.

$$L_{Huber}(x, \delta) = \begin{cases} \frac{1}{2}x^2 & \text{if } |x| \leq \delta, \\ \delta(|x| - \frac{1}{2}\delta), & \text{otherwise.} \end{cases} \tag{5.5}$$

δ is an adjustable hyperparameter that determines its sensitivity to outliers, or rather when errors should be treated as such. Errors smaller than δ are minimized with MSE and penalized harder than those larger than δ, as the Huber Loss becomes more linear (see how the the Huber loss changes for different δ in figure 5.3). (*TensorFlow: Huber Loss*). The Huber loss function is

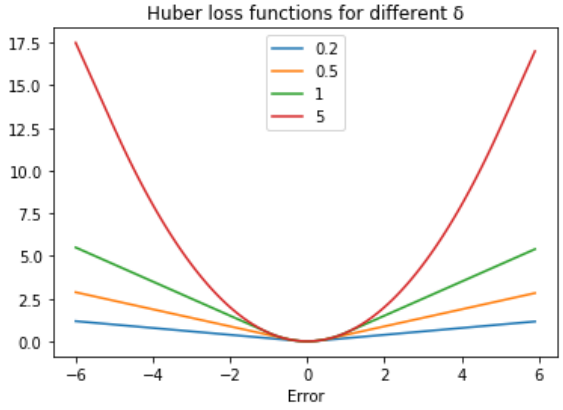


Figure 5.3: Huber loss function for different δ

not smooth, but can be replaced by a smooth approximation, the *Pseudo-Huber* loss function, defined as

$$L_{PseudoHuber}(x, \delta) = \delta^2 \left(\sqrt{1 + (x/\delta)^2} - 1 \right). \tag{5.6}$$

Unlike the above mentioned loss functions that can be implemented using the built-in loss functions `mean_squared_error`,

`mean_absolute_error`, `log_cosh` and `huber`, this smoothed Huber loss is not pre-defined in Keras TensorFlow, but can be easily be implemented as any other custom function (see e.g. *Keras, Custom Losses*).

All these functions can also be implemented using the (custom) loss function, presented in the paper "*A General and Adaptive Robust Loss Function*" by Jonathan T. Barron (2017).

$$\rho(x,\alpha,c) = \begin{cases} \frac{1}{2}(\frac{x}{c})^2 & \text{if } \alpha = 2, \\ \log\left(\frac{1}{2}(\frac{x}{c})^2 + 1\right) & \text{if } \alpha = 0, \\ 1 - \exp\left(-\frac{1}{2}(\frac{x}{c})^2\right) & \text{if } \alpha = \infty, \\ \frac{|2-\alpha|}{\alpha}\left(\frac{(\frac{x}{c})^2}{|2-\alpha|}^{\frac{\alpha}{2}} - 1\right) & \text{otherwise.} \end{cases} \tag{5.7}$$

This generalized loss function $\rho(x,\alpha,c)$ is a shifted version of the negative log-likelihood corresponding to the probability,

$$p(x \mid \mu,\alpha,c) = \frac{1}{cZ(\alpha)} e^{-\rho(x-\mu,\alpha,c)} \tag{5.8}$$

$$Z(\alpha) = \int_{-\infty}^{\infty} e^{-\rho(x,\alpha,1)} \tag{5.9}$$

and is extended all real values of α, unlike the probability function which is only defined for non-negative values of $alpha$.

The author Jonathan T. Barron describes the generalized loss function as a "superset of many common robust loss functions" (Barron 2017, p. 1). It is in fact a composition of single-parameter loss functions - amongst which the pseudo-Huber Loss is one - and is equipped with two hyperparameters - a shape parameter α and a scale parameter c (Barron 2017, p. 8). Figure 5.1.1 shows how the loss function gets increasingly harder on the errors the bigger they get. Errors smaller than $\frac{c}{2}$ are treated the same independently of α, but when errors grow larger the rate of change will depend on α. Put simply, the shape parameter α determines the sensitivity to outliers by how they should be penalized, and the scale parameter c determines how large an error must be for the function to assume that the associated target value is an outlier.

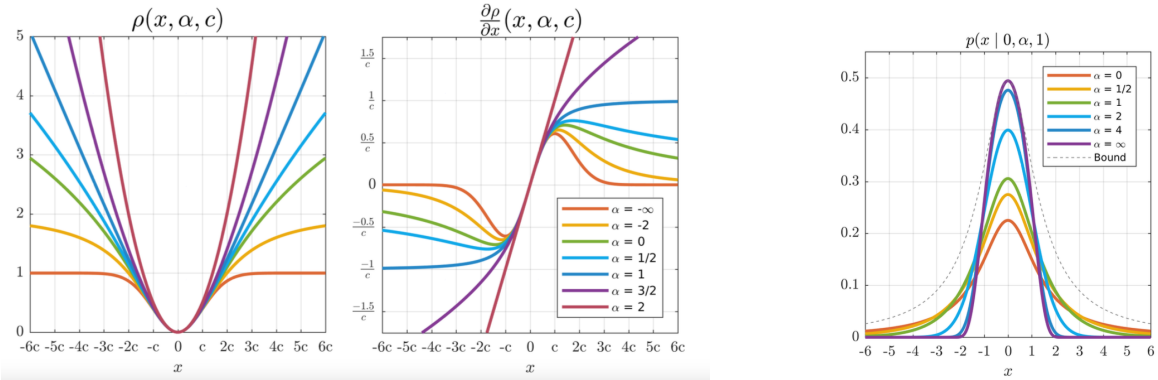


Figure 5.4: The loss function and their derivatives for different shape parameters on the left and the corresponding probability function to the right Figures from the paper *A General And Adaptive Robust Loss Function* by Jonathan T. Barron (2017)

The shape and scale parameter could either be pre-defined, or optimised as any other hyperparameter. One could also let the loss parameters be adaptive and allow it to take different values for different samples. More on the generalized and adaptive robust loss function and how to treat shape and scale parameters α resp. c is found in *Appendix* A.2. Further suggestions of how one can make use of generalized loss function is found under *VAE for Compression of Weather Series* B.5 as well as *Varying Robustness in Loss Function* B.6.

5.1.2 Multiple Losses and Loss Weights

In multitask learning a loss will be calculated for each and every target, using their pre-specified loss functions. These loss functions can, as for a single-target neural network, be on various forms and does not have to be the same for all target outputs. The total network loss can be set to be the sum of the separate output losses, but also be a weighted sum of the losses, weighted with target-specific loss weights.

$$Loss_{total} = \sum_{target=1}^{n} w_{target} \cdot Loss_{target} \tag{5.10}$$

See the `code` [1] used for calculating the loss in Keras found in the GitHub repository (*Keras*).

Though inclusion of the seasonal variables can help the network become more generalized, it can also cause problems. Yield and biomass predictions may demand more information than the (easily modeled) seasonal outputs. So if their errors are equally accounted for during training (meaning that model improvement of one output yields the same loss reduction as any of the others), it's fair to assume that it will be easier for the network to focus on representations favourable to the season variables. This could result in the network ignoring information useful for the yield output, due to it being complex to detect. Not only can search for weather patterns important for yield be evaded, also the climate inputs C, T, W and N can be ignored because they're of no use for the easily modeled phase length outputs that are the same for all climate scenarios.

Among the outputs presented in the data section, annual spring wheat is our main interest - and also the most complex (along with biomass), in terms of distribution and relation to the inputs. Without any loss weights - determining the importance of the outputs in terms of how much of their loss should contribute to total model loss - the network will produce very good estimates of the season variables, enhancing overall model performance. This can (and also did when tested) result in a model with very good seasonal output estimates, but worse yield and biomass estimates. By putting a constraint on the season outputs in terms of loss weights, the goodness of the season length variables will not be taken into account as much as the other outputs, forcing the network towards modeling the yield and biomass.

The importance of having outputs on the same scale will be brought up in the next chapter 6, but in the context of multi-task learning, it is particularly the resulting loss sizes that facilitates interpretation and adjustment of the loss weights. By range scaling the outputs to the same interval between zero and one, their losses will be, approximately, on the same scale. Each loss weights can thus be set to the fraction of the corresponding output loss that should contribute to the total model loss[2]. Note however, that though the loss weight can be described as a factor of how much the corresponding output should be accounted for (when the outputs have the same range), a model could produce better model scores for some low loss weight assigned output, if these are easier to model. The loss weights may therefore not exactly cohere with how well the model weights suit the corresponding output.

The loss weights were manually tuned and optimised through grid search, but there exist several automated strategies for determining the loss weights, see for example *Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics* 2018, *GradNorm: Gradient Normalization for Adaptive Loss Balancing in Deep Multitask Networks* 2017 and *Dynamic Task Prioritization for Multitask Learning* 2018.

5.2 Some Practical Considerations For Training

Providing higher degrees of freedom by adding more layers and nodes will enhance the networks expressibility. However, a network that can predict the exact simulator output for any given input from the training sample, might not be able to do so when receiving new observations. Algorithms with too many degrees of freedom are prone to learn the noise, hence taking the

[1] If the pdf reader can't read the hyperlink, copy:
`https://github.com/keras-team/keras/blob/9118ea65f40874e915dd1299efd1cc3a7ca2c333/keras/engine/training.py#L816-L848`

[2] In the case of different ranging outputs and therefore also unequally sized losses, the loss weights would have to be the product of a parameter value bringing the losses to the same scale and the fraction of how much the loss should be taken into account.

behaviour of the error into account when trying to model the response variable. This is termed *overfitting* and can be prevented e.g. through a robust loss function, described above in 5.1.1; by using the samples in a serviceable manner; or through handling of the weights. The latter two are presented below, in addition to some *regularization* techniques, that serves to generalize the model by preventing it from overfitting, together with other methods for helping the network learn better.

Hyperparameters are the parameters that defines the network structure and control the learning process. These parameters are not learned or optimised during training of the network like the weights and biases in the layers. They have to be pre-defined before training and hence must be optimised separately - or at least initialized if optimised through embedded hyperparameter optimisation.

Activation functions, the number of hidden layers, nodes, convolution filters and their kernel sizes are all examples of hyperparameters. Type of optimisation algorithm, loss function, the number of batches and the number of *epochs*, i.e. the number of times all batches are passed forward and backward trough the network, are all hyperparameters that effect the training. For example, smaller batches can make learning more precise while larger batches can converge faster to a solution faster and prevent overfitting (Heaton 2015, pp. 125-126).

Choice of Network Structure and Hyperparameters 5.2.8 provides some examples of iterative hyperparameter optimisation algorithms and how to implement them. Those optimisation algorithms are however not suitable for all hyperparameters - like e.g the learning rate for reasons explained under 5.2.7. Nonetheless, by getting an intuitive understanding of how they affect the network training for different problems, one can probably reduce the number of combinations to be compared, or at least know where to start.

5.2.1 Leaky ReLU and Thresholded ReLU

There exist several versions of the ReLU Activation Function and two of them - the *Leaky ReLU* and the *Thresholded ReLU* will be presented here. These share the many advantages with the regular ReLU Activation Function mentioned in section 4.5.

Leaky ReLU

$$\sigma_{LeakyReLU}(x) = max(\alpha \cdot x, x) \qquad for \quad \alpha \leq 1. \tag{5.11}$$

The *Leaky ReLU* does not have the regular ReLU zero-slope, i.e. no hard saturation at zero, and can therefore be used to alleviate the *dying ReLU* problem (Maas 2013, p. 2; Glorot and Bengio 2011, pp. 318-319). In Keras the activation layer `LeakyReLU(`α`)`, α can take values larger than one, but was here, as in many other applications, set to some small fraction between zero and one. Another option is to let α be an adaptable vector of the same shape as the input tensor x that is learned during training, but then the activation function usually is referred to as the *Parametric ReLU* (PReLU) (He and Sun 2015), implemented using `PReLU(`$\alpha_{initial}$`)` in Keras. (*Keras, Activation Layers*). The Leaky ReLU was however the only option considered here and the coefficient α was optimised trough grid search given a range of some pre-defined values between zero and one.

Adaptive Upper Threshold in Final Wrapper

The adaptive version of generalized robust loss function (presented in 5.1.1) penalizes negative and positive residuals equally (see extended explanation in B.6). However, the distribution of annual spring wheat is heavily right tailed at most locations, where some potentially influential, but likely miss-representative, outliers are far higher than other values. It could therefore be of interest to introduce an outlier-limit preventing the final network estimates to grow larger than a given threshold. If applied, the best estimates for samples beyond the threshold would be the actual threshold and hence restrict the related losses from never decreasing further than the difference between the threshold and the actual target. When prohibited from improving estimates of the outliers (beyond the given threshold) in this manner, the network can only reduce the loss through betterment of the other estimates (below the given threshold).

This activation function was tested (see how in outline A.7), but was not properly investigated nor used in the final model. More information about this activation function and how it can be applied, particularly for this emulation problem, is found in

section A.3.

5.2.2 Batch Normalization

Shifting and rescaling of inputs can facilitate the modeling - whether it's a min-max range scaling of nonnegative inputs or normalization of real valued inputs. Neural network training can benefit from normalized inputs, just as normalization can speed up training of all regression models. As the inputs are passed down a network, the distribution is likely to change and become less favourable. ((*2015) Keras*; Goodfellow 2016, pp. 309-312). A solution is to incorporate a normalization step before some, if not all, layers, using the Keras layer `BatchNormalization` ((*2015) Keras*).

5.2.3 Weight Initialization

In order to find the optimal weights and biases, they first have to be initialized (see for example the Adam algorithm A.1). In Keras, the bias weights are initially set to zero and the default initializer for the weights is the *Glorot uniform initializer* (Glorot and Bengio 2010), a.k.a. *Xavier uniform initializer.* ((*2015) Keras*). However, these initializers are not optimal for all problems. The suitability of certain weight initialization schemes vary for different activation functions.

Ill-fitting initializers can cause *vanishing gradients* and prevent the networks from learning. The *Glorot weight initializer* conditions the layer inputs and outputs to have the same variance as well as the gradients to have the same variance before and after having passed through the layer nodes. This constraint was integrated in order to prevent the signal from dying, exploding or saturating - both in the forward passes made to produce estimates and in the backpropagation of gradients. (Glorot and Bengio 2010).

If the weights are initialized from normal distribution and condition for the variances holds, then they have the variance $Var_{Glorot}(W) = 2/(n_{in} + n_{out})$, where n_{in} is the number of input units and n_{out} is the number of output units in the weight tensor. That is, if the Glorot normal initializer is used, then the weights are initialized with the truncated normal distribution $\mathcal{N}\big(0, \sqrt{2/(n_{in} + n_{out})}\big)$. (Glorot and Bengio 2010 *(2015) Keras*).

If the weights W instead are assumed to be uniformly distributed[3], then if the condition of equal variances holds, i.e. that the weights W have the Glorot Variance $Var_{Glorot}(W) = 2/(n_{in} + n_{out})$ we get,

$$Var(W) = L^2/3 := Var_{Glorot}(W) = 2/(n_{in} + n_{out}) \tag{5.12}$$
$$\Leftrightarrow \tag{5.13}$$
$$|L| := \pm\sqrt{6/(n_{in} + n_{out})} \tag{5.14}$$

Meaning that, if the Glorot uniform initializer is used, then the weights will be drawn uniformly from the interval $\big[-\sqrt{6/(n_{in} + n_{out})}, \sqrt{6/(n_{in} + n_{out})}\big]$. (Glorot and Bengio 2010; *(2015) Keras*).

He-initialization is a similar weight initialization, but designed for weights in layers producing ReLU-wrapped output. It makes use of the preserved variance derived from the Glorot Initialization by

$$Var(y) = Var(x)Var_{Glorot}(W) \tag{5.15}$$

and defines the variance of the ReLU wrapped layer outputs in terms of the He-initialization weight variance, by halving the weight variance accordingly,

$$Var(y) = ReLU(Var(x)Var_{He}(W)) = \frac{Var(x)Var_{He}(W)}{2}. \tag{5.16}$$

(He and Sun 2015). So the He-initialization assumes the weight variance to be twice the size of the Glorot weight variance, i.e.

$$\frac{Var(x)Var_{He}(W)}{2} = Var(x)Var_{Glorot}(W) \tag{5.17}$$
$$\Leftrightarrow \tag{5.18}$$
$$Var_{He}(W) = 2Var_{Glorot}(W). \tag{5.19}$$

[3] $W \sim \mathcal{U}(-L, L)$ then $Var(W) = L^2/3$ by definition of uniformly distributed r.v.

5.2.4 Weight Regularization

Weight regularization can prevent a neural network from overfitting by adding a regularizer $R(f)$ for the network function $f : x \mapsto y$, to the loss function $L(f, y)$,

$$L(f, y)_{penalized} = L(f, y) + R(f). \tag{5.20}$$

The regularizer can for example be expressed using the summed magnitude (ℓ_1-norm) of the weights w,

$$\lambda \sum_{i=1}^{n} |w_i| \tag{5.21}$$

perhaps scaled with some value λ. This is either termed the ℓ_1 or *lasso* regularizer. The *ridge* regularizer is similarly defined, but instead uses the ℓ_2-norm,

$$\lambda \sum_{i=1}^{n} w_i^2 \tag{5.22}$$

Both regularizers ensure that the weights do not grow too large, but differs in that the ℓ_1 regularizer prioritize sparsity by forcing seemingly irrelevant weights to zero, whereas the ℓ_2 regularizer is better suited for high dimensional spaces and can handle correlations better than ℓ_1. *Elastic net regularization* (Zou 2005) makes use of both the ℓ_1 and ℓ_2 penalty, by combining the two,

$$\sum_{i=1}^{n} \lambda_1 |w_i| + \lambda_2 w_i^2 \tag{5.23}$$

In the *tensorflow* module `tf.keras.regularizers`, lasso, ridge and elastic net regularizers are available. These can be applied on layer parameters in Keras models through `kernel_regularizer` and `bias_regularizer` that penalizes the layer's kernel and bias, as well as `activity_regularizer` that adds a penalty on the layer output.

5.2.5 Dropout

Dropout is the term for the regularization strategy to exclude, or drop out, a set of randomly chosen network nodes during training (Heaton 2015, pp. 175-176, 228–229). Exclusion (dropout) of different layer inputs during training, prevents the network from relying on single layer inputs. Similarly to the weight penalization, node-dropout can be used when the layer inputs correlate, to prevent canceling-out-effects. Further, it forces the network to make use of layer input that might have been assigned less importance when all layer inputs are provided.

Dropout can be implemented in Keras models by adding the layer `Dropout(rate)` prequel to the layer to be affected by dropout. Here, `rate` is a hyperparameter taking values between 0 and 1 and determines the fraction of inputs be excluded during training. (*(2015) Keras*).

`SpatialDropout1D(rate)` is another dropout-layer in Keras, but excludes whole feature channels instead of single elements and is often used after convolution layers. In the Keras layer description (*Keras: SpatialDropout1D layer*), the usage of this sequential dropout is motivated with following: *"If adjacent frames within feature maps are strongly correlated (as is normally the case in early convolution layers) then regular dropout will not regularize the activations and will otherwise just result in an effective learning rate decrease. In this case, SpatialDropout1D will help promote independence between feature maps and should be used instead."*.

5.2.6 Early Stopping

Early stopping can be seen as a optimisation in time and controls the complexity during a training session. By specifying a monitor metric in `EarlyStopping`, a Keras model can be set to only run for as long as it improves more than some pre-defined rate parameter. The network training can for example be set to stop after the validation loss improves by less than the rate parameter given, or let the network run as long as it improves at all. When training is allowed to continue after the validation metric has reached its minimum, improved estimates of training samples it can cause the validation metric to deteriorate. Hence a validation metric can be seen as a measuring of generalization the and a way to detect overfitting.

5.2.7 Learning Rate

The *learning rate* determines how much the weights are to be adjusted during training and can have huge impact on learning. A model can converge faster to a solution with a large learning rate, but larger learning rates runs a higher risk of convergence to a suboptimal solution. Too high of a learning rate can further lead to the *dying ReLU problem* mentioned in 4.5, especially for deep learning networks. On the other hand, too small learning rates can cause the learning to make insufficient progress in each step. (Reed and Marks 1999, p. 87; *(2015) Keras*).

A uniquely determined learning rate which is optimal during the whole training session might not exist. It can therefore be serviceable to allow for a varying learning rate. This could be implemented by setting the learning rate to decay according to a pre-defined *learning late scheduler* , or through *adaptive learning rate methods*, like *Adam* (Kingma and Ba 2014), where the weights are individually adjusted according to the specified learning rate as well as their iteratively updated respective moment estimates (see pseudo code A.1). (*(2015) Keras*).

Adam and other adaptive learning rate methods generally improves deep learning and usually converges faster than a simple optimiser with a fixed and improper learning rate (Kingma and Ba 2014; Reed and Marks 1999, p. 72).

The learning rate scheduler `ReduceLROnPlateau` is set to reduce the learning rate once some given metric stops improving. It could for example monitor the validation loss and keep track of when it stagnates as an indication of when generalization stops improving. In order to do so, the scheduler needs an *initial learning rate*, a specified *reduction factor* determining how much the learning rate is to be reduced by and must know when to reduce the learning rate. The learning rate scheduler can be set to reduce the rate after a pre-determined number of epochs, expressed by a *patience* parameter, of no further loss decrease; or if oscillation is to be accounted for: when the loss has stagnated on a plateau (neither increased nor decreased) for that long. (*(2015) Keras*)

The initial learning rate, reduction factor and patience parameter can be arbitrary chosen or optimised like other hyperparameters as described in the next chapter under *Choice of Network Structure and Hyperparameters* 5.2.8.

5.2.8 Choice of Network Structure and Hyperparameters

There is no straightforward or standard model selection method that can be used for all neural networks. It is not possible to compare all possible neural network architectures, but by training networks of different architectures and comparing them on validation-data, through non-nested model comparison measures like R^2 and root mean squared error (*RMSE*) (Rawlings and Dickey 1998, p. 220), optimal *hyperparameters* and network structures can be found among the ones given. Whether or not it is possible to base model design only on theory and intuition, some iterative trial and error procedure can streamline the search for a good model design.

Potential network candidates and hyperparameter combinations can be compared through *grid search* - parameter sweeps of the space with the pre-defined parameter subsets, in which every hyperparameter combinations is evaluated with the associated loss and performance metrics.

Talos (Talos 2019) is an automated hyperparameter optimisation tool for Keras models. By defining a Keras model frame and specifying a range of values for each hyperparameter to be evaluated, Talos can find the optimal hyperparameter combination among the ones given. It provides several options, s.a. *Grid search* and *Random search*, where a variety of random methods can

be used for drawing a set of hyperparameter values among the given ones: from uniform selection methods to lower discrepancy methods like *Latin hypercube sampling*. These can in turn be combined with *probabilistic reducers* that reduces the remaining search space by removing poorly performing values of hyperparameters. For more information see the Talos GitHub repository.

Another strategy for finding suitable hyperparameter values is through incorporation of penalties in the training of the network. LASSO, or ℓ_1-penalties (5.2.4) on weights and activations increases the loss function when they're included and forces unnecessary weights to zero. A display of the LASSO-network weights can reveal which layer inputs that have no impact on the result and thus indicate nodes or kernels that can be removed. This can also be used for input selection, where the importance of the received inputs can be measured by their corresponding weights in the first layer.

6

Preprocessing and Analysis of Data

In the Bachelor's thesis (Nilsson 2019), the square root transform was used for annual spring wheat because it follows symmetric distribution. The squared rooted yield is in this case also presumably much easier to model than the untransformed yield, but even those leaves much to be desired. However, while other data transformations possibly are be easier to model, complex transformations can impede interpretability. The distribution of above ground biomass is very similar to that of yield, so the square root transformation will be applied both outputs.

Also the number of days before anthesis and before maturity outputs behave very similarly. This becomes even clearer when we also apply min-max standardization,

$$\tilde{Y} = \frac{\sqrt{Y} - min(\sqrt{Y})}{max(\sqrt{Y}) - min(\sqrt{Y})} \tag{6.1}$$

See their distributions in figure C.2b. It will make the outputs similar outputs even more similar and make the losses of the same scale. The latter facilitates interpretation and adjustment of loss weights used during training of a multitask network (see 5.1.2).

The histograms C.2a of all four the outputs related to spring wheat show that many samples are stacked at zero and that the distribution is long-tailed and multimodal. This becomes even clearer when the samples are blocked by location and by climate scenario and analyzed separately. This could be due to measure errors or some simulator deficiency, but the more likely explanation is that the samples belong to differently distributed populations. See for example the varying ranges and multiple nodes in figure C.4a showing the distributions of global annual yield in different climate settings. The many modes could is a result of varying means in different types of sample groups, out of which most presumably only has a single mode in their distribution, whether the samples are grouped by location or similarly behaving locations. See for example figure C.4b displaying the distribution of annual crop responses to the base scenario $C_{360}T_0W_0N_{200}$ at some random locations, in four different climates: at which some have the about the same crop production every years, while at others the variance of annual yield is large; and the location-wise annual simulations are both right- and left skewed. The differences in skewness and length of tails between the blocks could be due to e.g. that extreme high yield is be more probable to occur at some locations and/or climate scenarios than others; just as e.g. the difference in number of zero-cases at some locations could be caused by varying probability of premature plant death.

As was previously mentioned, the sample space can be described as a climate hypercube spanned by the climate variables, or an extended version with the additional time-dimension and axis for the geographical position. Hence the samples could be blocked into smaller groups of different variable combinations, after their respective factor levels, which could be compared and analysed through e.g. ANOVA (ANalysis Of VAriance).

Section 6.1 below mentions some considerations that were taken into account before surrogate modeling. The following section introduces the *Köppen climate classification system* used in this surrogate modeling approach; it is properly explained in appendix A.4.

As was mentioned in chapter 3, a wise sample selection and split for training and testing can help control the surrogate models ability to predict in unseen scenarios. *Sample Selection and Split* 6.3 describes how this was handled. The final two section describes how the inputs where preprocessed before being passed into the emulator (6.4) and how the input variables selection were selected (6.5).

6.1 Considerations and Cautions

Outliers

Outliers can influence the training of a neural network. Besides from using a robust loss, an alternative is to remove outliers and miss-representative samples. In a data set as skewed and complex as the one considered here, it can be hard to determine which samples are miss-representative. The thresholds suggested by Tukey (1977) could possibly be useful for detecting outliers and are defined as follows:

$$T_{lwr} = Q1 - c \cdot IQR \tag{6.2}$$

$$T_{upr} = Q3 + c \cdot IQR \tag{6.3}$$

where Q1 and Q3 are the first resp. third quartile, or 25^{th} resp. 75^{th} percentile, and IQR is the the *inter quartile range* $Q3 - Q1$. The scale parameter c is usually is set to be 1.5, but sometimes take other values. Note that increased c gives a wider outlier-interval and that decreased c implies narrower interval. An alternative is to use the similarly defined thresholds below,

$$T_{lwr} = Q1_{lwr} - c \cdot IQR_{lwr} \tag{6.4}$$

$$T_{upr} = Q3_{upr} + c \cdot IQR_{upr} \tag{6.5}$$

where $Q1_{lwr}$, $Q3_{lwr}$, $Q1_{upr}$ and $Q3_{upr}$ are the 12.5^{th}, 37.5^{th}, 62.5^{th} and 87.5^{th} percentiles respectively, and the inter quartile range for the lower threshold IQR_{lwr} is set to be the range between the first and third quartile among the low-valued samples below the median, i.e. $IQR_{lwr} = Q3_{lwr} - Q1_{lwr}$, and the inter quartile range for the high-valued samples above the median lower threshold $IQR_{upr} = Q3_{upr} - Q1_{upr}$.

These thresholds were used in the preparatory data analysis in comparisons of samples with and without outliers, but in regards to the modeling: besides from the years and locations mentioned in chapter 2, no other outliers were removed. Outliers were instead dealt with through other training-techniques, to prevent them from having to large of an impact in the overall performance - through e.g. tuning of the shape and scale parameters in the robust loss function; and elaboration of the ReLU activation function, by applying an upper threshold in the final wrapper - as explained explained in section 5.2.1.

Unbalanced Data

Neural networks tend do focus on the normally behaving samples that make up the majority of the training set and takes less notice of rare behaviours and relations. Exclusion of outliers may therefore not significantly enhance the model performance. In fact, outliers and samples behaving differently to the majority can provide useful information and the difficulty of modeling rare samples might even be of a greater concern than the problem of modeling the over-all behavior.

Not only would the statistical power be reduced with the decreased data set, the rare samples will already be less accounted for during training as it is, since their related errors easily be can evened out in the cumulative loss and merely be read as noise. For the same reason, intra diversity within minority sample groups may be even harder to detect. (Zhou, Hu, and Wang 2018; Wang and Hebert 2017).

This property can be very rewarding, especially if the minority groups are misrepresentative, but could hinder learning of useful information in the data considered here. This because the vast hyperspace of unbalanced and non-i.i.d data samples - that can be blocked by climate-, temporal- and global position factor levels (i.e. the 29 annual crop responses to 672 different climate scenarios, at every global $0.5°$ by $0.5°$ grid cell location covering ice-free land) - includes many groups that preserve attributes different to the majority and some have large within-group variance. For this reason, many samples run the risk of being treated as extremes, or "minorities", in relation to the main stream and thereby also the risk of being ignored during training. Hence the loss of information provided by these samples that could be of use - if not in isolation, at least as a collective.

Multicollinearity

The potential problem with correlated inputs and inclusion of redundant information was discussed in the Bachelor's thesis (Nilsson 2019). Strong multicollinearity is mostly an obstacle for linear regression and becomes less of problem when nonlinearity is introduced. Neural networks usually consist of a vast amount of weight parameters and overparameterization itself includes redundant information. However, previous results from the Bachelor's thesis underlined that correlated inputs should be included with caution - where canceling-out-effect could be seen in e.g. the kernel weights in the convolution layers in which negatively correlated inputs could be assigned equally heavy weights with opposite sign and vice versa with positively correlated inputs.

Instead of excluding related inputs, multicollinearity can be dealt with through methods like those presented in section 5.2. For example, the sequential dropout can be tested on the convolution layer outputs, as suggested in section 5.2.5, but also applied to the input time series before passing them into the initial convolution layer, in order to prevent overfitting as well as the canceling-out-effects in the kernel windows.

6.2 Köppen Climate Classification System

The varying characteristics and preserved relations can be hard to detect in the large data set, even for a neural network (as explained in previous section 6.1), but if homogeneous samples are analysed in isolation patterns may become more distinguishable and the smaller groups can in turn be intercompared.

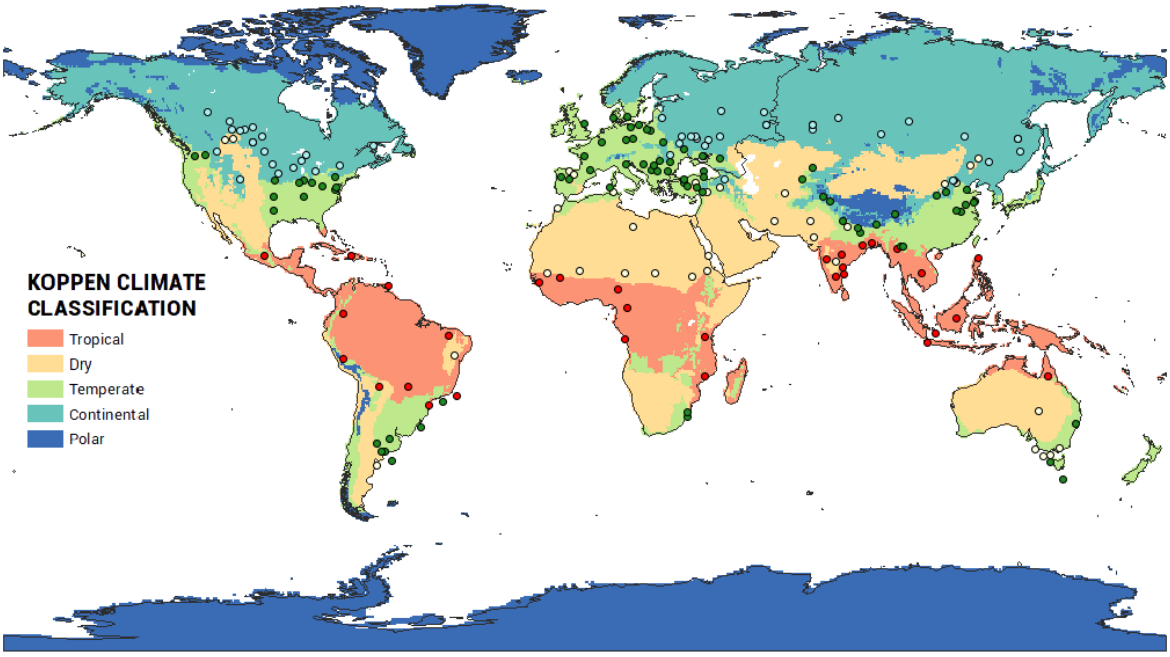


Figure 6.1: The five main Köppen climate classes tropical (A) covering 22.2% of the land, dry (B) covering 32.2%, temperate (C) covering 16.5%, continental (D) covering 24.0% and polar (E) covering 5.1%. Figure from *"What Are the 5 Koppen Climate Classification Types?"* 2020. The 192 locations are also mapped out and marked with the color of their class.

The widely used *Köppen climate classification system*, based on historical climate and vegetation, can classify the heterogeneous data samples to separate regions around the globe. It was used in the data analysis, in the evaluation of feature importance on which the input variable selection was based as well as for aggregation of the emulation domain (elaborated under 7.1).

Using temperature and precipitation summary statistics, the *Köppen climate classification system* separates the world into the five main climate classes: tropical (A), dry (B), temperate (C), continental (D) and polar (E), shown in figure 6.1. It can further divide these into several sub-classes, but these mostly larger groups where considered here. The definitions and conditions used for the five main classes can be found under section A.4.

In this study, the input weather time series of precipitation (pr) and temperature (tas) were either based on the actual observed or the climate adjusted versions defined in section 2.2.1 - where pr is changed with the climate variable W and tas shifted with the climate variable T. The actual observed time series were used for aggregation of the emulator domain and is further discussed under 7.1. If the climate adjusted series are used for classification, then for some locations, the crop responses to changes in T and W would be classified into different climate classes. I.e. a location could belong to some climate regions for some TW-combinations and another for other settings. In that case, samples derived from the same location would be estimated with different climate class models, which in turn would introduce more uncertainty in the sensitivity study across the climate dimensions T and W. However, the climate adjusted series it yielded better clustering in terms of similarly behaving groups and was therefore applied in the data analysis and input variable selection 6.5.

However, locations' different climate scenario crop responses (in the space spanned by T and W) would be classified into different climate classes. That is, samples derived from the same location would be estimated with different climate class models, which in turn would introduce more uncertainty in the sensitivity study across the climate dimensions T and W.

6.3 Sample Selection and Split

A wise sample selection and split for training and testing can help us control the surrogate models ability to predict in unseen scenarios. It can be useful to implement a *sample selection method* to find the optimal or most representative sample to train and test the surrogate model on. A *sequential model design* (mentioned in chapter 3) could be of use to construct a surrogate model for LPJ-GUESS. However, such sample selection was not embedded in the model search conducted here, because the initially given large set of already simulated data for globally spread locations (assumed to be representative of all the grid cells covering ice-free land). These locations were also the ones used in the preparatory Bachelor's thesis (Nilsson 2019) and in the thesis testing the Gaussian Process approach (Olsson 2017). Nevertheless, emulation can be improved by a wise sample selection, that accounts for both *exploitation* of samples proven to be awarding and *exploration* of uncertain samples. A Bayesian optimisation strategy to accomplish this is described in appendix B.3.

Cross Validation Split and Randomly Selected Subset

An emulator for a global vegetation model like LPJ-GUESS ought to be generalized enough to produce estimates of crop at any given location, based on learning of crop production from only a fractional part of the world. In order to test how well the trained emulator can estimate crop responses at unseen locations, some sites should be kept out of training. For that reason, the *cross validation* split was based on randomly drawn locations, where the *test set* was given 20% of the locations and remaining 80% were divided into a *validation set* with $20\% \cdot 80\% = 16\%$ of the locations and *training set* with $80\% \cdot 80\% = 64\%$ of the locations. That is, 64% of the locations were used for the learning process with iterative weight adjustments, described in the training chapter 5, 16% were used for validation of the weight adjustments made each epoch and the 20% contained in the test set was applied in the final evaluation of the trained network.

The risk of including all sequentially dependent samples is that the network learns location-specific information and hence runs the risk of overfitting. It can therefore be useful to exclude some of the the annual samples from each location and climate scenario. A random selection of years for every location and climate scenario, proved to help the networks tested to generalize better. For more details, see sample selection pseudocode A.5 in Appendix.

6.4 Preprocessing of Inputs

If the chosen set of weather series $S_{i,y}$ are not adjusted for the climate anomalies T and W, the emulator $f(\cdot)$ would have to take T and W separately,

$$f(S_{i,y}, C_c, T_t, W_w, N_n, \mathbf{X}_{c,t,w,n,i,y}). \tag{6.6}$$

Whereas an emulator $f(\cdot)$ given the climate adjusted series $S^{clim}_{t,w,i,y}$ would be able to account for the information provided by T and W without necessarily including the climate variable T and W directly,

$$f(S^{clim}_{t,w,i,y}, C_c, N_n, \mathbf{X}_{c,t,w,n,i,y}) \tag{6.7}$$

for all locations i, years y and climate factor levels c, t, w, n in the CTWN space, where $\mathbf{X}_{c,t,w,n,i,y}$ denotes some arbitrary input (here chaining over all climate dimensions, between locations and years).

The final considered $\mathbf{X}$ was a vector with the location varying inputs soil $\mathbf{soil}_i$ defined in 2.2 and latitude, lat_i, defined in 6.4,

$$\mathbf{X_i} = [lat_i, soil_i]. \tag{6.8}$$

The cube root is often used for transformation of rain data and improved training of the network trained in the Bachelor's thesis. Though the cube root was used during the model search, only using range scaling was sufficient for the final model. In the bachelor's thesis, the daily data was aggregated to 5-day averages. These as well as averages over even longer periods than five days, were considered in this continued study. Since longer averages could result in loss of important information, the section *Monthly Time Series* presents a suggestion for how time series of monthly averages could compensate for potential loss of useful information, through inclusion of additional summary statistics (taken from the daily weather time series).

Monthly Time Series

The initial attempt in the bachelor's thesis was to find summary statistics of the weather data based on the patterns automatically detected by the train CNN. Elaboration with longer time averages of the daily time series (see A.7) resulted in seven monthly times series derived from the simulator inputs **pr**, **rsds** and **tas**.

The 13 months long time series started 24 days before planting and stretched over the 365 days following planting. Hence first 30 days ($\approx$ one month) covered the 24 days before planting, the sowing day and the 5 days after. The second "month" covered the 6^{th} day after sowing and the 30 consecutive days, the third month covered the following 30 days and so on, so that the last, 13^{th}, "month" covered the final 30 days with the 365^{th} day after sowing as the final day. The seven considered sequences contained the following summary statistics and are visualized in figure C.1

- $\bar{T}_m$: $mean(tas)$ in month m,
- $T_m^{(Q_{10})}$: 1^{th} decile of the daily observed temperatures in tas during month m,
- $T_m^{(Q_{90})}$: 9^{th} decile of the daily observed temperatures in tas during month m,
- $\bar{P}_m$: $mean(pr)$ in month m,
- $P_m^{(\%0)}$: fraction of days in pr without rain during month m[1],
- $P_m^{(Q_{90})}$: 9^{th} decile of the daily rain observations in pr during month m,
- $\bar{R}_m$: $mean(rsds)$ in month m,

For more information on why and how these sequences were derived, see section A.7. [2].

[1] A 10% percentile would be zero for the majority of month, since there can only be $0.1 * 30 = 3$ observations (days) under the lower quantile, implying that no more than two days in one month can be rain free for the lower quantile to be larger than zero. So instead of choosing another quantile threshold, I figured I would use this fraction of no-rain days. Additionally previous tested CNNs often seemed to find long periods of no or little rain to be of importance (e.g. according to kernel windows) and it could help indicate probable drought.

[2] Note that the monthly sequence of measured fraction of days without precipitation (defined in 6.4) is not affected by precipitation anomaly modification in 2.2.1, since change of rain is adjusted in the daily time sequence *pr* through the scaling factor *P*, hence the fixed number of zeros (days without rain) in the sequence left unchanged.

Latitude as Indicator Function

The aim is to construct an emulator general enough to predict spring wheat production in all grid cells covering ice-free land, but only 192 locations are given (out of which only a subset will be used for learning). So if the network is fed the precise degrees of latitude, it would learn the exact latitudinal relations only for individual grid cells given during training and thereby run the risk of overfitting. This could be prevented by aggregating the latitude coordinates to less specific information, e.g. latitudinal band belonging.

An *indication function*, $\mathbf{1}_{\mathcal{L}}$ of a subset $\mathcal{L}$ of the latitudes $\{-55.75, -55.25, \ldots, -0.25, 0.25, \ldots, 81.25\}$ that are considered in the GGCMI study, is for a latitude i defined by

$$\mathbf{1}_{\mathcal{L}}(i) := \begin{cases} 1 & \text{if } i \in \mathcal{L}, \\ 0 & \text{if } i \notin \mathcal{L}. \end{cases} \tag{6.9}$$

The latitude input was defined as a vector of eight indicator variables (containing a one for belonging and seven zeros for the rest of the intervals to which the sample does not belong),

$$\mathbf{lat}_i = [\mathbf{1}_{\mathcal{L}_1}(i), \mathbf{1}_{\mathcal{L}_2}(i), \mathbf{1}_{\mathcal{L}_3}(i), \mathbf{1}_{\mathcal{L}_4}(i), \mathbf{1}_{\mathcal{L}_5}(i), \mathbf{1}_{\mathcal{L}_6}(i), \mathbf{1}_{\mathcal{L}_7}(i), \mathbf{1}_{\mathcal{L}_8}(i)] \tag{6.10}$$

for all considered locations $i = 1, 2, ..., 192$. The limits were determined so that every interval contained about the same fraction of samples locations considered in GGCMI, i.e. each band contains around 12.5% the locations.[3]

Figure C.7 shows how the 192 locations used in this surrogate modeling approach are distribution over latitude as well as the distribution of all locations considered in the GGCMI study, where also the limits of the eight latitudinal bands are specified. Figure C.8 show two plots containing the same information as the first mentioned, but distinguishes between the training, validation and test set in one and between the four different climate classes derived from the aggregation scheme used in 6.2.

Range Scaling of Inputs

The z-score normalization, will be used for the weather time series , but the soil input vector and the latitude input vector of indicator variables, both have range [0,1], hence do not have to be range-scaled. The final network structure only has one linking of branches, where the climate variables will be concatenated directly with the other branch outputs, which will be wrapped in the ReLU function and thus be non-negative. Therefore a min-max normalization of the climate variables (making them range between 0 and 1) will be used.

6.5 Input Variable Selection

Investigation of input importance was conducted separately for the four main classes A, B, C and D. Samples derived for the same locations were allowed to be classified into different climate classes. The reason for analysing the different classes separately was to get better insight and to easier detect relations between inputs and outputs.

A simple way to get an indication of which variables can be of importance for predicting a certain output, is to look at their linear relation, by e.g. evaluating the *Pearson Correlations*. Another motive for exposing the correlations is that it can be good to keep track of the correlations when using interpretation methods that do not account for feature dependence. Lasso (5.2.4) is one such method, which can be used for input variable selection by penalizing the weights in the initial layers accepting the inputs, since a ℓ_1-penalty forces weights corresponding to unnecessary inputs to zero, as described at the end of 5.2.8. Since we're also interested in intertwined and non-linear relations, a *Gradient Boosting Regressor* was trained in order to to see which features that mattered the most for predicting yield. It was implemented through the ensemble-based method

[3]Initially the latitude dummy variables were uniquely defined for the different climate class models, because the climate regions only stretch over some latitudes and also differ between the classes (see the map 6.1).

`GradientBoostingRegressor` in the `sklearn.ensemble` module. The perk of using a random forest instead of a chosen network is that the results are independent of the network structure.

The network used for the input importance analysis, shown in figure 4.1.1 was not the same as the final model. It contained

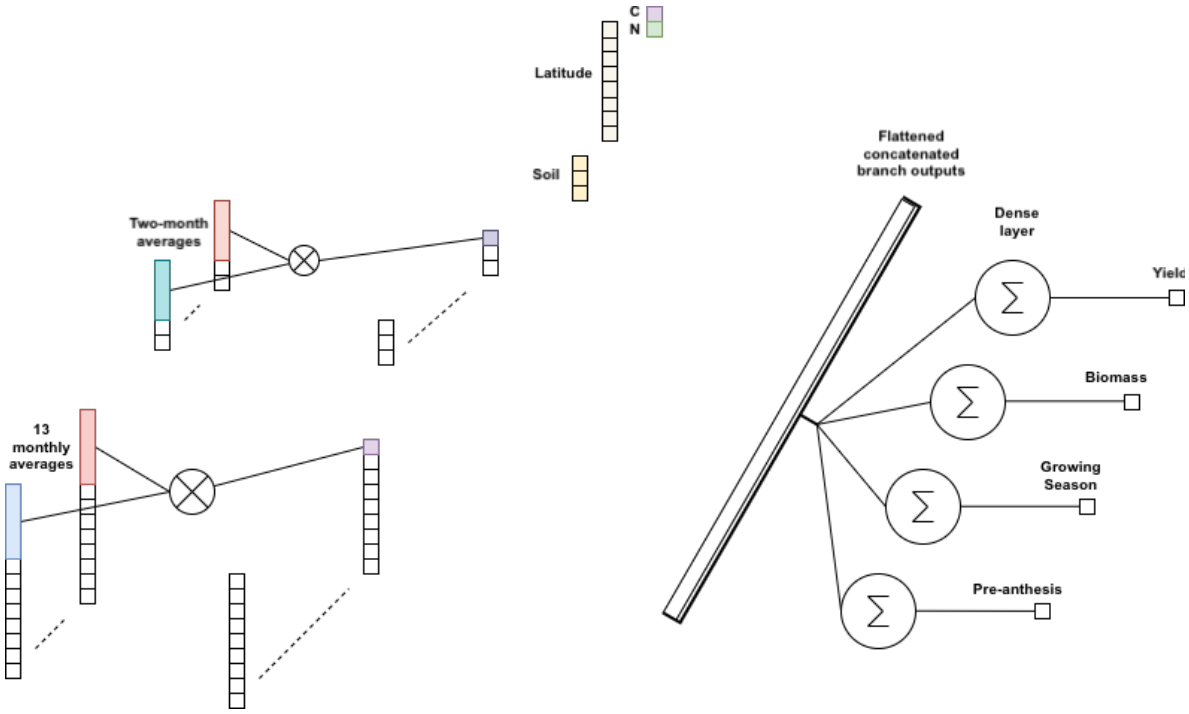


Figure 6.2: Neural network used for evaluation of permutation feature importance.

one convolution branch scanning the climate adjusted monthly weather sequences, a second convolution branch initialized with average pooling layer taking the same weather series excluding the first month and producing pairwise averages (because an average pool of size 2 was proven to be the best among the tested pool sizes: 2, 3, 4 and 6 respectively). The extracted features coming from the two different convolution branches were both flattened and concatenated with the other input variables: soil, latitude and the climate variables C and N. The following sets of inputs were permuted separately,

- every weather sequence taken by the first convolution branch, one at a time,
- every average-pooled weather sequence taken by the second convolution branch, one at a time,
- every weather sequence and their the corresponding average-pooled version, one (pair) at a time,
- every soil variable, one at a time,
- all latitude dummy variables/indicator functions
- climate variable C
- climate variable N.

For every model taking one of the above listed feature permutations, the permutation feature importance error difference resp. ratio measures - comparing predictions of the permuted data with those without permutation - were calculated and analysed together with the models scores R^2 and RMSE.

Details of how the input importance was analysed can be found under section A.6. It includes an example of how the presented methods were applied.

7
Implementation

This chapter presents the implemented surrogate model approach, motivations behind the emulator design and the chosen inputs and outputs. An elaborate outline of the model search, including the decisions made, considered problem formulations and re-formulations as well as the reasoning behind, is given in Appendix A.7. Details of the network design and hyperparameters are summarized under *Results and Evaluation* 8.

The *Universal Approximator Theorem* proves the existence of a neural network that can approximate the function f(.) (Goodfellow 2016, p. 499) which leaves us with the problem of deciding what the neural network should contain. Potential model architecture candidates were tested and optimised through hyperparameter optimisation using either *grid search* or *random search* in *Talos*. Also L1 penalization was used for fine tuning (see 5.2.8). When no good enough model was found and when problems could not be solved by training-adjustments (see 5.2) nor through elaboration with inputs (A.7), I had to go back and reformulate the model nstructure.

As was mentioned under 6.1, neural networks are heavily reliant on the data used during training. Regularization methods can prevent overfitting, they do tend focus on the the normally behaving samples that make up the majority of the training set and takes less notice of rare behaviours and relations. Even very large and complex networks had difficulty with predicting extremes and distinguishing between modes. This was a problem already in the Bachelor's thesis (Nilsson 2019), where only one climate scenario was modeled and where the main variation (in terms of distribution and input-output relations) was that between locations. This eventuated in exploring of domain aggregation, detailed under section 7.1, with the hope of creating a model composition of disjoint models (see chapter 3) that could overcome the difficulty of modeling the unbalanced and seemingly non-i.i.d data considered here.

7.1 Model Composition and Aggregation of Domain

As was mentioned in the precious chapter, the diversity in the global crop responses to shifts in climate - caused by variation in impact of the different climate scenarios at different geographical locations, among which many diverge from the main stream - can aggravate learning of general behaviours. If too complex of a network is needed in order to perform well for all scenarios, then several models for separate domains may have to be constructed, e.g. a *model composition*. Such integrated composite model can be trained similarly to the location based regression in the GGCMI study (see 1.1.1). Considering that even simple models trained for individual grid cells can result in hundreds of thousands parameters, larger model domains are to prefer.

Worth taking into account is the fact that the vegetation models in the GGCMI study takes part in a larger linkage of models (e.g. economic models) and should be adaptable also in the integrated modeling (see section 1.1). The *integrated assessment modelling* (IAM) users can be interested in running simulations for a smaller set of grid cells, say a country, which could possibly overlap several model domains. They would then have to use several models: one for for every respective model domain. Therefore the predictions should be equally reliable in every grid cell Hence *aggregation of model domain* and a *model composition* emulator may be advocated if the separate models are equally reliable - or at least not significantly different.

One approach suggested in the GGCMI is to aggregate grid cells to *latitudinal levels* or bands. This leaves us with the choice of width and limits of the latitudinal band [1]. Though there is a clear relation between crop yield and latitude, data analysis of simulated spring wheat suggests that crop outputs not necessarily behave similarly at the same latitude, so this type of aggregation may be too rough.

[1]Latitudinal bands could be uniformly distributed or set in an automated fashion to find the optimal latitudinal bands

The *Köppen climate classification system*, used to facilitate data analysis and input variable selection in the previous chapter (see 6.2), was in the end also used for aggregation (see details 7.1.1). An even better classification in terms of similar behaviour could perhaps have been derived from automated unsupervised clustering methods, like *K-means, Gaussian Mixed Models* or *Markov Random Field models*. These *Bayesian classification methods* were not thought of until late in the process and was therefore left for further research, so only a brief explanation of the Bayesian classification methods is given in Appendix B.4. If the Köppen climate classification system proves to cluster the samples in an appropriate fashion in terms of modeling, it has the advantage over the other Bayesian classification methods, in that it is easily comprehensible to the user.

7.1.1 Köppen Climate Aggregated Model Domain

The definitions and conditions used for this classification system are found under section A.4, together with examples of how it can be applied on the data used in the LPJ-GUESS. As was stated earlier (6.2), if the Köppen climate classification were to be based on the climate adjusted time series it would yield better clustering of similarly behaving groups. Furthermore, when these four model domains were modeled separately (A.7), the evaluated model compositions outperformed networks trained for the whole domain.

In order to conduct a reliable sensitivity analysis of how spring wheat responds to shifts in climate, simulations corresponding to the same locations should preferably be modeled by the same model, hence be classified into the same Köppen climate class. For this reason, it is probably better to classify solely based on the observed weather series. The resulting climate classes may have larger within-group variation, but data analysis and comparison of the data belonging to different climate classes suggested that also this could be a good clustering method for grouping of the samples into domains of similarly behaving ones.

Splitting of different climate change responses corresponding to the same locations could also be prevented if all climate scenario responses at a location was included in a class as soon as a location was found in a class, or when the class contains more than half of the climate scenario responses at that location. The former would result in (all scenarios at) some locations being classified into several classes, but the latter would put all locations in the class to which the majority of the scenarios belongs.

7.2 Selected Variables and Neural Network Architecture

A network taking using only the observed time series and introducing the cumulative changes measured by T and W in separately (together with C and N), describes how the outputs would change with T and W at a location with the given weather and soil characteristics within a certain latitudinal band, at any year - whereas the estimates from a network taking the shifted time series could be interpreted as predictions of how the outputs would change with T and W if rain and temperature would change equally much every month (or day if daily time series).

Initially branched networks was trained to map annual spring wheat by taking the climate variables C and N trough one branch and scanning the time series of observed radiation (*rsds*), climate adjusted precipitation ($\tilde{pr}$) and temperature ($\tilde{tas}$) in a separate convolution branch. [2] This was done both with and without the latitude and soil inputs (see A.7). When it became evident that non of the tested network were able to properly model the crop response to changes along the T- and W-dimension, the other approach described by equation 6.4 was tested instead. That is, networks using only the observed weather data, instead of taking the climate adjusted time series, in which T and W are introduced separately, as the variables C and N. These networks turned out to be much better suited for modeling of crop responses across the shifts in climate if the shifts of T and W.

The *Köppen climate classification system* was used for aggregation of the model domain and resulted in a model composition of four neural network trained for the disjoint climate classes A, B, C and D, all using the same inputs and outputs (where spring wheat is of main interest), listed under 7.2 resp. 7.2.

[2] ($\tilde{pr}$) and ($\tilde{tas}$) adjusted according to the shifts in their respective climate variables (W) and (T) in accordance to description 2.2.1

Selected Inputs

The inputs were selected using the methods presented in 6.5, as described in the outline on page 39, and resulted in the following intpus:

- $\bar{P}_{i,y,m}$: $mean(pr_{i,y})$ in month $m \; \forall \; m = 1, 2, \ldots 13$,
- $P^{(\%0)}_{i,y,m}$: fraction of days in $pr_{i,y}$ without rain during month m for $\forall \; m = 1, 2, \ldots 13$,
- $\bar{T}_{i,y,m}$: $mean(tas_{i,y})$ in month $m \; \forall \; m = 1, 2, \ldots 13$,
- $\bar{R}_{i,y,m}$: $mean(rsds_{i,y})$ in month $m \; \forall \; m = 1, 2, \ldots 13$,
- $\mathbf{lat}_i = [\mathbf{1}_{\mathcal{L}_1}(i), \mathbf{1}_{\mathcal{L}_2}(i), \mathbf{1}_{\mathcal{L}_3}(i), \mathbf{1}_{\mathcal{L}_4}(i), \mathbf{1}_{\mathcal{L}_5}(i), \mathbf{1}_{\mathcal{L}_6}(i), \mathbf{1}_{\mathcal{L}_7}(i), \mathbf{1}_{\mathcal{L}_8}(i)]$,
- $\mathbf{soil}_i = [clay_i, silt_i, sand_i]$ as defined in 6.4,
- C_c, T_t, W_w, N_n

for all years $y = 1, 2, \ldots 29$, locations i in the respective climate class region (A, B, C and D) and climate levels, or points (c,t,w,n), in the CTWN hypercube.

Selected Outputs

When all four outputs presented under 2.1 were considered, the hope was partly that the number of days til anthesis and maturity could help the convolution branch focus on important periods: That they could help the network distinguish between the different developing phases and in turn shred some light on the differences between the phases. E.g. to help figure out whether some patterns mattered more during some period, like growing season or grain-filling phase, and less during others. If nothing else, they could be of use for interpretation of the weights.

However, in the end only the the priority yield output and biomass was used, or to be more precise: the min-max range scaled square root transformed yield and biomass,

- $\tilde{Y}_{i,y,c,t,w,n} = \frac{\sqrt{Y_{i,y,c,t,w,n}} - min(\sqrt{Y_{i,y,c,t,w,n}})}{max(\sqrt{Y_{i,y,c,t,w,n}}) - min(\sqrt{Y_{i,y,c,t,w,n}})}$: range scale square root of annual yield,
- $\tilde{B}_{i,y,c,t,w,n} = \frac{\sqrt{B_{i,y,c,t,w,n}} - min(\sqrt{B_{i,y,c,t,w,n}})}{max(\sqrt{B_{i,y,c,t,w,n}}) - min(\sqrt{B_{i,y,c,t,w,n}})}$: range scale square root of annual biomass,

for all years $y = 1, 2, \ldots 29$, at all locations i in the respective climate class region and in every climate scenario (c,t,w,n).

One reason for excluding the seasonal outputs was that they are set to the same for all climate scenarios and are therefore not representative. Secondly, they did not help the model to predict better with the shifts in climate. Inclusion of seasonal outputs seemed to improve the estimates of the yield-outputs, in at least B and D climate classes, but for the other two classes A and C, the network performed best with small loss-weights for the seasonal output (determining how much och the seasonal outputs should be accounted for during training and adjustment of weights). It felt however reasonable to keep the biomass variable - due to its similarity to the crop yield.

Final Input Branches

The figure below exemplifies a set of branches that could be used for the spring wheat emulator. These branches are in fact the ones used in the final chosen model, but other branch compositions and designs were also considered and tested during the model search.

The first branch in figure 7.1[3] does not perform any transformation of the received inputs, it is only an array holding four inputs (in the final model these are the climate variables C,T,W and N). The second branch (in the final model taking the soil variables) contains one dense layer with one node, thus producing only a single branch output. The third branch also processes

[3]Note that the network and the branches will receive tensors containing batches of multiple samples and these input tensors will thus be of the same shape as the inputs shown in the figure, but will have an extra dimension for the batches - i.e. have the shape *input shape*$\times$ *batch size*. So the inputs in the figure can be seen as input tensors with one sample, i.e. of shape *input shape*$\times 1$.

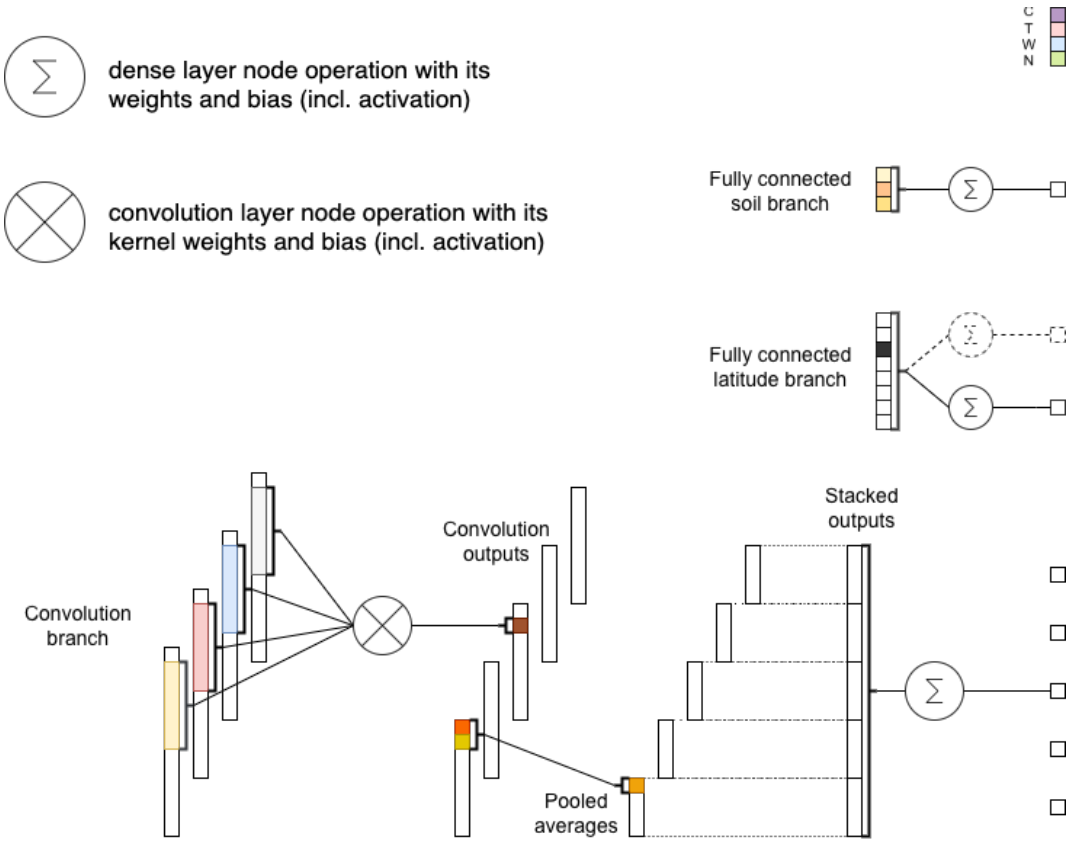


Figure 7.1: Illustration of the four branches used in the final model presented under results. The dense layer node operation is denoted $\bigoplus$ in the text, but the convolution node operation is denoted as in the figure with $\bigotimes$.

the inputs (the latitude indicator variables described in section latitude, i.e. zeros and ones, in model presented) through a dense layer, but here with two nodes together producing a total of two branch outputs.

The final branch is the convolution branch taking the time series. $\bigotimes$ symbolizes the convolution layer node operation and the activation function transformation[4]. The convolution outputs are then (average) pooled, two by two, thus producing pooling outputs of half the length. These are then column stacked and passed through a dense layer.[5]

As was illustrated in the figure, a branch can be anything from just a column stacking of inputs (as the branch taking the climate variables in the figure) to a deep neural network (like the convolution branch). These branch outputs can then be passed into the network simultaneously, by concatenating them as in figure 7.2 in the following section, or at different stages. The former was done in the finally chosen model presented under **??**.

Considered Output Branches

Figure 7.2 shows the continued part of the branched network shown in figure 7.1, in which the branch outputs are concatenated, passed through shared layers and then separated into k output branches. The k targets. This network could of course have been designed in several ways - it could for example accept the inputs at different steps along the network and the k output branches could contain more layers than just one, as in figure 7.2.

[4]The one visualized in the figure is the convolution node producing the third convolution layer output array. The other four convolution layer outputs each corresponds to a convolution node $\bigotimes$ with another set of kernel windows

[5]Also here, only one dense layer node operation $\bigoplus$ (the third) is visualized in the figure, but all five dense layer outputs were produced by five different layer nodes.

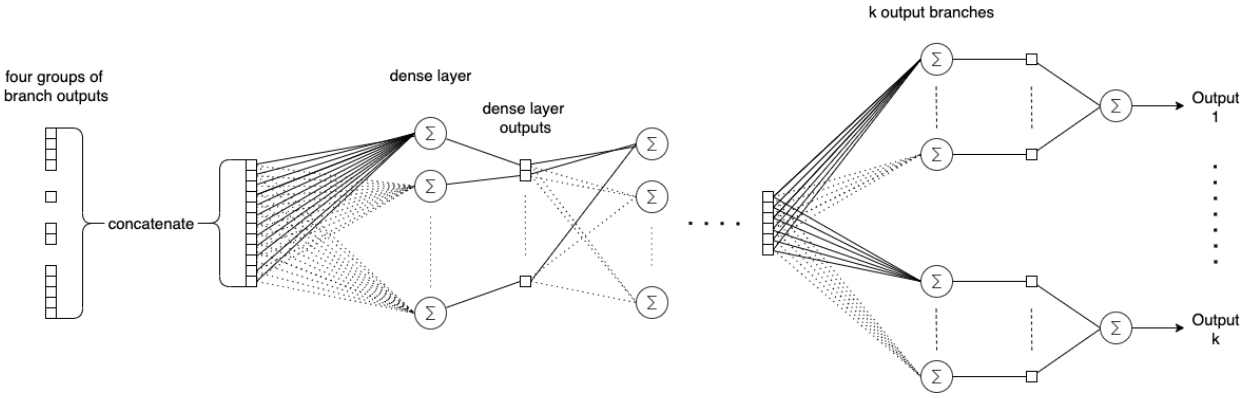


Figure 7.2: A network concatenating the branch outputs from figure 7.1[a]

[a]The network passes the inputs through two or more dense layers, before splitting into k output branches, each a containing a dense layer with a set of nodes producing separate outputs, which in turn are passed through the final dense layers single node that produces the estimate of its corresponding target value.

Initially when yield, biomass, anthesis and maturity outputs all were considered, the best estimates were retrieved by networks with an initial generic part (often sufficient with one generic layer), after input concatenation, that then was split into two branches: one containing layers shared by the yield and biomass output, followed by a subsequent splitting into the task specific branches; and one similarly designed branch corresponding to the other two seasonal outputs. I.e. the network contained generic parameters that were shared among all outputs, parameters that were shared only by the yield and biomass (with the unit of measurement), parameters shared by the two seasonal output (with the unit of measurement) and respective task specific parameters. The latter output branch, specified for prediction of length pre-anthesis and maturity season usually required less layers and nodes than the other two.

It also became evident that no more than one finalizing task specific layer was needed in order to retrieve good estimates of the respective outputs. Meaning that the two pairs of outputs could share parameters all the way until the final dense layer, forming a regression equation of the outputs from the layer shared with the output it was paied with.

8

Results and Evaluation

The *Köppen climate classification system* (defined in A.4) was used for aggregation of the model domain. The resulting model composition of four neural networks corresponding to the disjoint climate classes A, B, C and D: trained to map the same inputs to the range scaled square root transformed yield and biomass targets. Each climate-class model preprocessed the inputs listed in 7.2 trough the separate branches, visualized in figure 7.1 - i.e. the monthly weather sequences where passed through a CNN, both the input vectors containing indicator latitude variables resp. the observed soil ratios of clay, silt and sand, where passed through a dense layer with two and one node, respectively, and the climate variable inputs C, T, W and N where passed directly to the concatenation of the branch outputs. Also a sequential dropout (5.2.5) was used: it randomly excluded one of the four weather time series during training, which helped both in terms of reduced overfitting as well as reduced canceling-out-effects. Though the input branches were equally designed for all, they were retrained, i.e. optimised, individually for every climate class.

The part accepting the branch outputs were designed according to figure 7.1. This part was also weight-optimised separately (as the input branches) and though most model design parameters were set to be the same, they were allowed to have different numbers of nodes in the layer receiving the branch outputs.

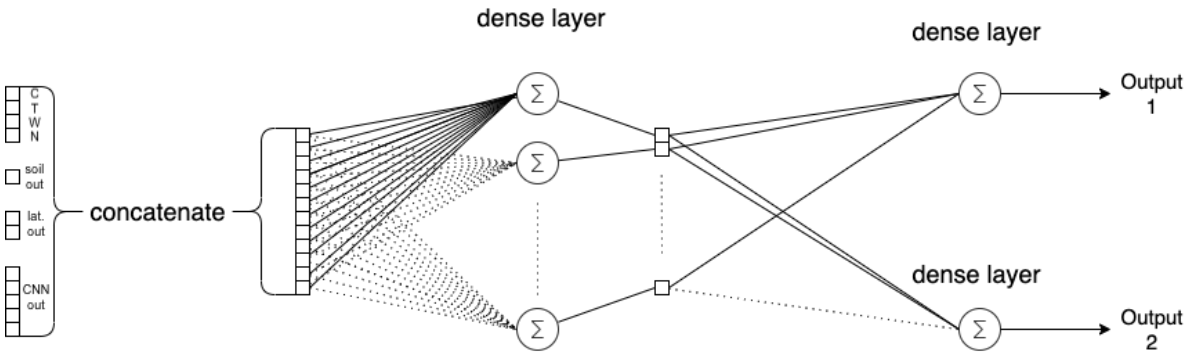


Figure 8.1: The architecture of the network[a]

[a]It takes the four climate variables, the outputs coming from the soil-branch, latitude-branch and convolution-branch, and producing the estimates of annual production of spring wheat and above ground biomass. All climate-class models are structured in this fashion, but differ in the number of nodes in the initial dense layer that takes the concatenated branch outputs

For simplicity the hyperparameter values shown in table 8.1 was chosen for all. When it became evident that the simulations at locations in climate class B turned out to be much harder to model than the other classes, the focus shifted from fine tuning of the individual classes, to a search for a simple model architecture, that worked well for all climate class models and especially favouring the more demanding modeling of climate region B. This to facilitate performance comparison and model analysis that can be of use for further research - despite of the fact that several types of hyperparameter combinations had shown to be better suited for the three other classes.

Some hyperparameters were however allowed to vary between the climate class models and are presented in table 8.2, where their respective optimal value is given for the separate climate class models. The total number of parameters in the separate models are also displayed in the table.

The samples were split by location to form the training, validation and test sets, but the network only got to see samples form seven randomly drawn years for each location and climate scenario as described in section 6.3. I.e. for every combination

Model Parameters:				Training Parameters:			
Nr Filter	$Nodes_{conv}$	$Nodes_{lat}$	$Nodes_{soil}$	*Batch size*	*Weight init.*	*Nr epochs before LR red.*	*Nr epochs before stop*
5	5	2	1	8000	*He Uniform*	4	15

Table 8.1: Table of model hyperparameters and training-parameters common for all climate class models [a]

[a]The parameters given here were proven to be suitable for all climate models, however not necessarily optimal for every separate model - at least not according to the conducted hyperparameter optimisation described in the *Model Search Outline* A.7. The parameter combination that favoured the more demanding modeling of climate region B were preferred over others, given that they also were good for the other climate class models.

Class	Total parameters	Model Parameters:	Training Parameters:						
		$Nodes_{L_1}$	*LR init.*	*LR red.*	$L2pen$	$L2pen_{lat}$	$Loss$	$w_{L(B)}$	$Dropout$
A	374	8 (M)	0.0075	0.3	1e-04	1e-07	$Huber_{0.3}$	0.7	0.25
B	404	10 (L)	0.0075	0.4	1e-03	1e-05	$Huber_{0.3}$	0.3	0.25
C	374	8 (M)	0.0075	0.1	1e-04	0	$Huber_{0.3}$	0.65	0.25
D	344	6 (S)	0.0075	0.3	0	0	$Huber_{0.3}$	0.9	0.25

Table 8.2: Table of model hyperparameters and training-parameters optimal for the different climate class models.[a]

[a]$nodes_{L_1}$ is the number of nodes in the dense layer accepting the concatenated branch-outputs, *LR init.* is the initial learning rate, *LR red.* the learning rate reduction factor (activated at stagnation during 4 epochs). $L2pen$ and $L2pen_{lat}$ are the L2-penalty in the dense layers following the branch connections (i.e. the dense layer taking the branch outputs and the dense layers in the separate output branhces) resp. the L2-penalty in the latitude branch. *Dropout* denotes the *Sequential Dropout* applied on the input series before passed into the convolution layer. All parameters in the table were suggested by the thoroughly conducted hyperparameter grid search optimisation described at the end of the *Model Search Outline* A.7.

of location and CTWN-combination only a quarter of the annual samples were included in the training and validation set.

The learning rate was set to be reduced at stagnation after 4 epochs (see the learning rate section 5.2.7), with a climate class specific reduction factor (optimised through grid search) and the network training was stopped when the of the validation loss of the high priority output yield had not decreased for 15 epochs (see *Early Stopping* 5.2.6). The choice of parameters determining how patient the callback function `EarlyStopping` and the learning rate scheduler `ReduceLROnPlateau` should be was based in previously observed loss trajectories (as described at the of the model search outline A.7).

After initial optimisation and fine tuning of the shape and scale parameter in the generalized loss function (5.1.1), determining when and how the residuals should be penalized, it soon became apparent that the model performance was not highly dependent on the choice of loss function. Yet, three loss functions of varying robustness was included in the final grid search (see A.7): *Pseudo-Huber loss* with $\delta = 0.3$, *log-cosh loss* and *MSE loss* (in declining order of robustness). The *Pseudo Huber loss* was the best for all climate models and also for the majority of the hyperparameter combinations evaluated in the end. Even so, the MSE and log-cosh loss produced almost as good model scores (in some cases better).

8.1 Model Evaluation

The performance measures R^2 and RMSE were calculated for the network estimates of the range-scaled root yield Y_{target} and range-scaled root biomass B_{target} as well as for the untransformed yield Y_{orig} and biomass B_{orig} on the original form. Table

8.4 and 8.4 displays these yield and biomass scores, calculated separately for the test set and for all locations included in the training and validation set , i.e. including those years not seen during training.

Class	Total parameters	$\lvert S_{locations}\rvert$ S_{test} (S_{train}, S_{val})	Y_{target}		Y_{orig}	
			R^2	RMSE	R^2	RMSE
A	374	7 (19 5)	0.85 (0.91 0.78)	0.081 (0.068 0.113)	0.84 (0.9 0.69)	0.706 (0.626 1.182)
B	404	7 (21 6)	0.65 (0.83 0.38)	0.095 (0.073 0.12)	0.66 (0.74 0.47)	0.759 (0.857 0.885)
C	374	16 (50 13)	0.78 (0.82 0.62)	0.06 (0.057 0.072)	0.73 (0.82 0.61)	0.963 (0.788 0.896)
D	344	10 (30 8)	0.87 (0.86 0.62)	0.048 (0.059 0.096)	0.86 (0.83 0.57)	0.456 (0.633 1.094)
All	1496	40 (120 32)	0.83 (0.87 0.69)	0.069 (0.062 0.095)	0.8 (0.83 0.63)	0.782 (0.741 0.995)

Table 8.3: Table of model scores for the range scaled root yield Y_{target} and the untransformed simulator outputs annual spring wheat Y_{orig}. $|\mathcal{L}_{locations}|$ is the number samples in the set of locations $\mathcal{L}_{locations}$. The third column contains the number of locations included in each model and sample set. The model score for the unseen test set $\mathcal{L}_{test}$ is the first value in the cells, the other two values within the brackets corresponds to $(\mathcal{L}_{train}, \mathcal{L}_{val})$, i.e. are the model scores derived from the training and validation set, respectively

Class	Total parameters	$\lvert\mathcal{L}_{locations}\rvert$ $\mathcal{L}_{test}$ $(\mathcal{L}_{train}, \mathcal{L}_{val})$	B_{target}		B_{orig}	
			R^2	RMSE	R^2	RMSE
A	374	7 (19 5)	0.85 (0.92 0.77)	0.08 (0.062 0.115)	0.83 (0.91 0.68)	0.877 (0.721 1.572)
B	404	7 (21 6)	0.71 (0.83 0.58)	0.09 (0.071 0.104)	0.66 (0.75 0.53)	1.193 (1.089 1.316)
C	374	16 (50 13)	0.8 (0.86 0.61)	0.064 (0.054 0.081)	0.79 (0.87 0.6)	1.145 (0.906 1.213)
D	344	10 (30 8)	0.86 (0.85 0.58)	0.046 (0.059 0.099)	0.87 (0.81 0.53)	0.516 (0.783 1.426)
All	1496	40 (120 32)	0.83 (0.88 0.68)	0.068 (0.06 0.096)	0.81 (0.86 0.61)	0.987 (0.886 1.348)

Table 8.4: Table of model scores for the range scaled root biomass B_{target} and the untransformed simulator outputs total above ground biomass B_{orig}. Same as table , but here for the lower prioritized simulation output total above ground biomass.

The overall model scores derived from the model composition are relatively high. According to the coefficient of determination R^2, the complete model can explain over 80 percent of the variation of both yield and biomass in unseen data (i.e. the test set that has not been used during training).

The performance measures derived separately for each climate model are also relatively high, but varies both between the climate classes and between the training, validation and test sets [1]. The high model scores are supported by the the histograms

[1] R^2 and RMSE are better for the training set compared to the validation set, which usually is the case, since the weights are based on the error surface created by the training loss. This can also be seen in the four climate model loss plots C.2 of the trajectories of the training and validation losses for yield, biomass and their weighted sum, for all separate climate models, evaluated for every epoch, where the training loss is lower than the validation loss. These figures also display the initial learning rates and the factors of how much they should be reduced with when the loss has converged on a plateau. For the same reasoning, the network could not fit the test set equally well as for the training set. However, the model scores are higher for the test set than for the validation set.

Difference in model scores between the test and validation set could be explained by difference in sample size or by differences in sample behaviour between the two sets. Compare for example the distribution of yield and biomass in the test, training and validation, respectively, in the histograms C.2 and C.2.

In addition, the model selection was solely based on how well the networks fitted samples from the test set, since it indicates how well the

C.2 and C.2 displaying the outputs on the original form (measured in $ton/(ha \cdot year)$) together with the (inverse-transformed) network estimates (measured in $ton/(ha \cdot year)$), for the the climate classes A, B, C and D. They show that the distribution of the (inverse-transformed) estimates are similar to that of the actual targets.

However, in the scatter plot C.13, it becomes clear that the network always underestimates the spring wheat production for the largest values, but for the other samples it is not as clear due to the vast number of samples - with 752 640 in the test set . The more detailed figures C.14 - C.17 displays different parts of figure C.13.

Further, the range of the residuals are quite wide which can be seen in the scatter plot C.18, where also the the residual distribution for the separate classes are shown. Nevertheless, the vast majority of the residuals do not stretch further than the standard deviation of the transformed wheat targets (± 0.17). The visualized percentile lines, based on the residuals in the plot, show that 95% lies within the interval [-0.162,0.133] and only one per mille grow beyond the [-0.311, 0.325].

The network's estimate of annual spring wheat are plotted against the actual targets (measured in $ton/(ha \cdot year)$) in C.24. The figure also displays a line marking the perfect fit together with a somewhat flatter fitted regression line. The outliers are also here evident and seems to preserve structures. It is hard to detect whether there are any structures among the estimates closer to the perfect fit and though the estimated regression line suggests that the overall trend of the estimates does not comply with the true targets, it is not fully reliable since it is highly sensitive to outliers (as was illustrated in figure 5.2 comparing simple linear regression modeling of samples with and without outliers).

Though the above mentioned could be said about the fitted line in C.20 where residuals are plotted against the targets, it shows that of two independent x- and y-variables. But there are obvious structures in the residuals. The scatter plot looks however less dissatisfying, in an overall sense, if analysed together with the residual distribution visualized alongside the plot. Note that the lower limit of the residuals is initially expanding linearly as the estimates grow which is the result of using the ReLU-function as a final wrapper (see 4.5).

8.2 Block-wise Analysis

It can be hard to interpret visualizations of large data sets and much can be canceled out or not be accounted for in the model scores presented above, but there seems to be some explainable variation in the residuals. It might be easier to analyse the estimates and residuals if they are blocked into sub-groups of different variable combinations, after their respective factor level, as was suggested in the chapter 6. Since the residuals showed that there's room for improvement of the models, only a simple analysis for the top priority yield estimates will be conducted here.

The models should preferably be equally reliable for every point in the domain. As mentioned, IAM users might want simulations from e.g. from a country covering grid cells from several climate class regions which hence would be provided by different climate-class models. If the models are to produce estimates of every location with (about) the same precision, then the model scores for each climate class model should be about the same. One could therefore start off by comparing the different climate class models.

The kernel density estimation (KDE) of the underlying distribution of spring wheat residuals for each climate class can be seen in figure 8.2. Climate class C and D seems to be the only classes with well behaving residuals, despite the tails (which in relation to the other classes are not that long). The residual distributions for the other two classes are not as smooth. This is probably due to that the final ReLU wrapper - setting all negative network estimates to zero - is activated more often in the climate regions A and B with many LPJ-GUESS zero-yield simulations. Though the residuals in climate class B does not range as far as the ones in class A, it looks like class A has more small residuals.

network performs for unseen data. This in contrast to the model scores for the training and validation sets, which does not say much about the general performance, but rather reviles how well the network performs for those specific samples - on which the network weights have been based on, or at least have been seen and accounted for during training.

That being said, model comparison solely based on the model scores for the test set can be somewhat misleading, since one model among a set of equally generalized models can fit the considered test set better than the others. There is a risk that the chosen models by chance were particularly suitable for the test set, hence the doubt the test score can fully represent the generalization of a neural network.

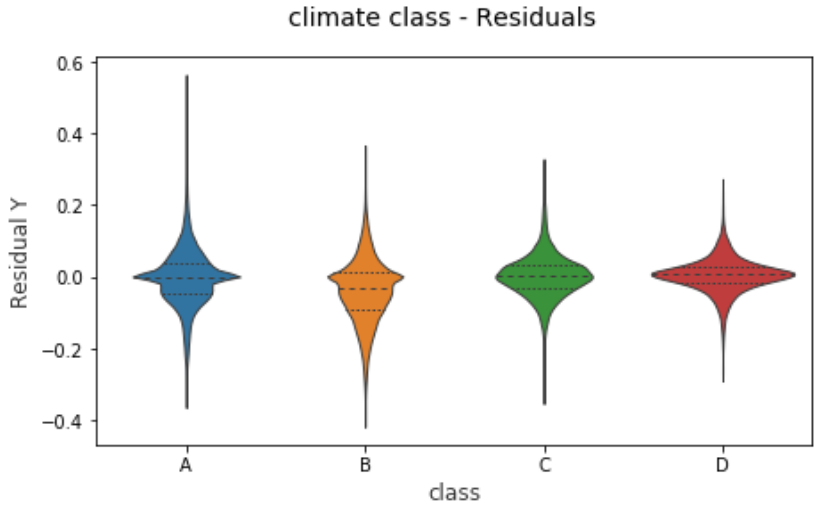


Figure 8.2: Violin plot of Spring Wheat Residuals

The separate climate class model scores (presented in table 8.4) also indicate that the uncertainty is unevenly distributed among the climate classes.

Comparison of the estimated residual distribution for every separate unseen location (i.e the locations in the test set) in the violin plot C.22a also show that the performance varies between locations.

Differences Across Climate Variable Dimensions

More violin plots for the separate climate models are appended in appendix, where the residual distributions are estimated for every separate climate variable level in the CTWN hypercube C.2.

The residual distribution for the climate model A and B varies along all climate dimensions C, T, W and N, but are seemingly unchanged for some adjacent factor levels climate variable levels e.g the residual distributions are gradually changing along the W-dimension and in model A but for C-levels, the residuals mainly behave differently in the first carbon dioxide level C_{360} (the latter is also true for climate model D). The other two climate class models C and D show far less variation in the residuals along the four climate dimensions. However, in the nitrogen violin plot, a more drastic gradual change of the residual distributions can be seen along the N-levels.

There could be several explanations for why the residuals change more dramatically along the N and C dimension. Firstly, N and C have far fewer factor levels. In fact, N and C, with three resp. four levels, combined have as many climate levels as T and less than W with eight precipitation levels. Thus more samples contains the same N and C levels than T and W levels, respectively. So during training of the network, each N and C level was fed into, or seen by, the network about twice as many times, compared to the respective T and W levels. It could thus be argued that N and C run a greater risk of overfitting. Secondly, precipitation and temperature evidently have large impact on spring wheat - which is why the complementary inputs *pr* and *tas* are used in the crop simulation, so the network might therefore be more considerate in the modeling of those inputs, since neural networks are prone to focusing on the most important features.

Though the aim is to model responses to shift in climate, somewhat poorer accuracy at some climate factor levels can perhaps be tolerated, especially if greater uncertainty can be explicitly displayed and thereby accounted for in the sensitivity analysis. This however only applies if uncertainty is distributed in a similar fashion at all considered locations (and therefore also climate classes), hence not applicable for this model composition. The above mentioned seems to be less of a problem along the time dimension. According to the violin plot C.22b for the respective climate models' residuals that distinguishes between years, the uncertainty does not vary that much between the years. Mainly the length of distribution tails appear to

vary.

This type of comparison is important if we want the emulator to be useful for the AMI users, but the above mentioned visualizations can only give an indication of how the model performance differ between locations and other dimensions in the sample space. As mentioned, it can be hard to detect dependencies and underlying explanations when working with such a vast amount of data, where also the separate blocks contain many samples. Therefore a more statistically sound comparison could be of use, once an emulator yielding good overall models scores has been found (whether the final emulator will consist of several models with disjoint model domains, like the climate-class models, or whether it will be one single model able to predict responses to shift in climate for all locations). ANOVA of blocked residuals could for example be used, in which the analysis can be designed to distinguish between the subgroups of the years, locations and climate levels and compare the group residuals (through the statistical tests used in typical ANOVA).

9
Conclusions and Further Research

The model composition of separate Köppen climate class networks showed that sample aggregation can improve the estimation of extremes and help the networks to better distinguish between modes, due to the reduced within-class variation. As a result, the need for complex models was greatly reduced. Also less fine tuning of training parameters was required because of the less complected variations within each of the disjoint model domains.

Above ground biomass is very similar to the spring wheat simulations in distribution (recall the min-max range scaled distributions shown in figureC.2b), but was easier to model. It can be seen as an noise reduced version of the top priority output spring wheat (at least by the network) which could be the reason for the seemingly improved estimates of the yield outputs. Although the biomass outputs behaves better form a modelers perspective, the same sample dependencies and complex output structures are preserved, hence biomass is also hard to model. Multitask learning of yield and biomass would certainly not demands as many model parameters as if they were to be modeled separately.

The seasonal outputs *number of days before anthesis* and *length of growing season* was initially considered in the multitask learning, but was later removed because they (in contrast to yield and biomass) were fixed for all CTWN climate scenarios. This in spite of the results insinuating that the modeling of spring wheat was improved through the simultaneous learning of the lower priority tasks. Even if they had been adjusted for the climate changes, it could be better to model them separately, as they are very easily modeled (alone) and in need of only a few parameters. Whether or not the climate adjusted seasonal outputs could help the network learn better, including them would most likely require many more model parameters than if the seasonal and wheat mass outputs were to be modeled separately.

Though robust loss functions sometimes could improve model scores, they could not handle all types of outliers. This is probably due to the difference in target distributions. The loss function can be linked to the target distribution and it is reasonable to assume that the optimality of a loss function depends on the sample distribution. *Varying Robustness in Loss Function* B.6 proposes how sample dependent robustness could be implemented in the loss function.

Many methods and adjustments implemented during training had positive effects on performance (for example the learning rate scheduler, early stopping, weight penalties and He-initialization), but overfitting seems yet to be a problem.

Input Variable Selection

First and foremost, the tested networks conformed that if the climate variables W ans T were introduced separately, where only the observed weather series were used as inputs, responses to changes in precipitation levels and temperature could be modeled better. However, this implementation is harder to generalize for other climate models that only uses the climate adjusted time series, for which the anomalies T and W would have to be extracted. An alternative could be to retrain the network composition that uses the climate adjusted weather series $\tilde{pr}_{w,i,y}$ and $\tilde{tas}_{t,i,y}$ (eq. 2.2.1) (in accordance to the initial attempt 7.2) as well as takes e.g. the averages of $\tilde{pr}_{w,i,y}$ and $\tilde{tas}_{t,i,y}$, instead of T and W, in the separate branch with C and N.

That being said, the feature importance alaysis clarified how the inputs impacted the outputs and resulted in a suitable set of inputs. Though soil and latitude seemed to be of less importance for predicting annual spring wheat, their inclusion did improve the estimates.

The investigation of input variable importance was conducted when the seasonal output variables still was included in the multitask models and were hence also accounted for in the final selection. On the other hand, the selection was mainly based on the input effects on spring wheat estimates. Further, the network that was used for feature permutation was defined for the climate adjusted time series, where only the climate variables C and N was passed in a separate branch. But this may not be

that big of a concern since the input selection also was supported by the random forest and general data analysis.

If one were to re-do the analysis of input importance, the seven time series with monthly summary statistics defined in 6.4 could also be revised. Though extreme temperatures can be seen by shifts in the daily mean-values, the shifts might not be fully representative of how extreme the temperature actually were, since the effect of an extreme event can be enhanced, reduced or even canceled out by other daily events. For example, a rise in daily average temperature could be caused by an overall warm day or be the effect of a high peak in temperature on an otherwise normal day. In retrospect, I think more useful and diverse information could be gained from the lower quantile of $min(t_{day})$ and upper quantile of $max(t_{day})$ instead of quantiles of average temperature $mean(t_{day})$ in one day - or at least give a better representation of the extreme events like high temperature assiciated with drought.

Alternative Network Architectures

The climate variable related dependencies in residuals needs to be better managed in the model. As mentioned in the outline under A.7, when an extra layer was added before the output branching, the model residuals tended to be less correlated with the climate variables. However, this did not uniformly improve model score.

An alternative approach could be to pass the climate variables through one or several dense layers before connecting them with the other network branches, instead of directly concatenating the climate variables with the other three branches. Such a network does not necessarily require more model parameters, since it could result in fewer nodes needed in the joint dense layer. An example of such model is presented in Appendix B.1. Though it is only an extension of the above models - and not fully optimised in terms of hyperparameters - it shows that an extra dense layer in the CTWN-input-branch can remove some of the dependencies in the residuals.

Sample correlations could also be accounted for in the network, rather than assuming them all to be independent. A *Recurrent Neural Network* (RNN) can achieve just that, if the input and output space is ordered correctly. Different types of RNNs that can handle serial correlations are discussed in appendix B.2.

Importance of Sample Selection

The dependencies between the samples may also be reduced by an appropriately designed sample selection. The importance of *sample selection* was supported by the reduced overfitting and improved performance that resulted from the random exclusion years [1]. Such sample selection could be extended to the climate variable dimensions and possibly further reduce overfitting. Instead of using an initial sample selection, a sequential design could be implemented, in which the sample selection can be optimised iteratively along with model reruns. *Bayesian optimisation* is often used on black-box surrogate modeling and is explained in B.3.

Aggregation

Annual spring wheat in the dry climate region B was much harder to model than those in the tropical (A), temperate (C) and continental (D) regions. The most probable cause for this is the large variation of samples. In fact, climate class B is composed by samples (locations) that satisfy the conditions for class belonging of A, C and D, reps (which are solely defined by temperature statistics), but that also satisfy the extra conditions for being dry (defined by both temperature and precipitation statistics). Though perhaps fewer distinguishable sample characteristics can be found in class B, than in the whole sample domain, the samples in B are evidently too diverse for the network.

If appropriately aggregated, the separate models would not have to learn as many varying features and require less parameters, which in turn even could result in fewer parameters, than if the whole domain (as any other domain containing significant diversities) were to be modeled by one network. This effect could be seen when models were trained for the separate climate classes derived from the climate adjusted time series $\tilde{pr}$ and $\tilde{tas}$ (defined in 2.2.1), mentioned in the introduction of

[1] i.e the exclusion of three quarters of the years along the time dimension the sample domain, drawn independently for every respective location and climate scenario

section 7.1.1. Fewer model parameters and features could in turn facilitate interpretation of what is learned, especially if the model domain is aggregated in a comprehensive matter.

The Köppen Climate Classification system is advantageous in that the classes are easily interpreted, but as it fails to cluster the data into adequate classes. If it is to be used for emulation of the data considered, some adjustments are in order. Since all scenarios in B are just dry locations in class A, C and D, respectively, it may be sufficient to exclude B and distribute the samples among only these classes. If the dry climate samples differ to much from the samples within A, C or B, an alternative could be to also consider aggregating the grid cells into even smaller classes using the sub-classes of A, B,C and D, which would result in classes that all are defined by both temperature and precipitation statistics.[2]

Another option is to perform unsupervised classification based on explanatory features. There exists several such classification methods that can be implemented in an automated fashion. Three viable candidates are the briefly mentioned *Bayesian classification methods* conditioned with various assumptions about the data. For more information see B.4.

As was mentioned in 7.1.1, if the Köppen climate classification proves to cluster the samples in an appropriate fashion - even if slightly adjusted according to e.g. above suggested configurations - it has the advantage over other automated classification methods (like those based on Bayesian inference) in that it is easily comprehensible to the user.

Whether or not the Köppen climate classification system is to prefer, temperature and precipitation proved to of help for clustering of crop responses. Perhaps also the seasonal output variables also could provide information to base the clustering on. Despite being excluded from the model, the seasonal variables do correlate with the crop responses and they did in fact help the network learn, at least in some cases. Hence, they could be of use in aggregation of the model domain, e.g. in the just mentioned Bayesian aggregation methods.

Notwithstanding the room for improvement, the neural network approach can evidently be used for emulation, especially if sample domain classification is complemented with some *sequential sample design*, which most likely will increase the probability of success for separate branched multitask CNNs combined into an integrated composite emulator.

[2]This however is not true for the polar climate class E, which only is defined by low temperatures, so if there are any samples that do belong to class E one could just remove the lower temperature threshold in the other climate class conditions that distinguishes them from the polar climate.

Appendices

A

Extensions

A.1 Pseudocode for Adam

Algorithm 1: The Adam Algorithm

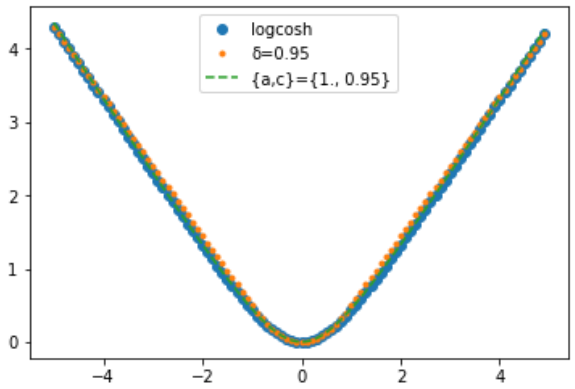
Require: Stepsize lr (Keras default: 0.001)
Require: Exponential decay rate β_1, $\beta_2 \in [0, 1)$ for 1^{st} and 2^{nd} moment estimates (Keras default: 0.9, 0.999 resp.)
Require: A small constant for numerical stability ϵ (Keras default: 1e-07)
Require: Stochastic objective function $f(\theta)$
Require: Initial parameter vector θ_0
$m_0 \leftarrow 0$ (Initialize 1^{st} moment vector)
$v_0 \leftarrow 0$ (Initialize 2^{nd} moment vector)
$t \leftarrow 0$ (Initialize time step)
while θ_t *not converged* **do**
  $t \leftarrow t + 1$;
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients);
  $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased 1^{st} moment);
  $v_t \leftarrow \beta_1 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t \odot g_t$ (Update biased 2^{nd} moment);
  $\hat{m}_t \leftarrow \frac{m_t}{(1-\beta_1^t)}$ (Bias correction in 1^{st} moment estimate);
  $\hat{v}_t \leftarrow \frac{v_t}{(1-\beta_2^t)}$ (Bias correction in 2^{nd} moment estimate);
  $\hat{\theta}_t \leftarrow \theta_{t-1} - lr \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}$ (Update parameters)
end

A.2 The General And Adaptive Robust Loss Function

The general robust loss function can be seen as smooth approximation of the Mean Absolute Loss function for $a = 1$ which also resembles the Huber Loss, where the c - like the Huber Loss parameter δ determines the width of the less robust convex part. When Huber Loss and the generalized function (with a equal to 1) both having 0.95 as their shape parameter they looks just like the log-cosh loss function.

When a approaches 2, the loss function start to behave like the Mean Squared Error Loss function. Note that the general loss equation is not defined when a is 2. Nor is it defined for when a is 0 or infinite. When implemented, the neural networks will make use of the adjusted version of the general loss function presented in "A General And Adaptive Robust Loss Function" Barron 2017, p. 10, that guard against the removable singularities of the general loss function and numerically instability for when alpha close to 0 and 2.

The "*adaptive*" part of this loss function was not used in this study. This generalized loss function was in fact designed for *Variational Autoencoding* (VAE) of images (where the loss function is that of the input data and the encoded-decoded data), in which the adaptive part comes in naturally in the intermediate sample generation from the encoded latent space. What distinguishes this general and adaptive function from other loss functions is the additional shape parameter α. VAEs are



further explained under section *VAE for Compression of Weather Series* B.5, that focuses on how VAE can be applied using this generalized and adaptive robust loss in general and in the surrogate modeling of crop responses to climate change in particular.

A.3 Adaptive Upper Threshold in Final Wrapper

The adaptive version of generalized robust loss function (presented in 5.1.1) penalizes negative and positive residuals equally (see extended explanation in B.6). However, the distribution of annual spring wheat is heavily right tailed at most locations, where some potentially influential, but likely miss-representative, outliers are far higher than other values. It could therefore be of interest to introduce an outlier-limit preventing the final network estimates to grow larger than a given threshold. If applied, the best estimates for samples beyond the threshold would be the actual threshold and hence restrict the related losses from never decreasing further than the difference between the threshold and the actual target. When prohibited from improving estimates of the outliers (beyond the given threshold) in this manner, the network can only reduce the loss through betterment of the other estimates (below the given threshold).

If the distribution of samples varies between groups and all samples do not seem to belong to the same population, the outlier limit can not be uniquely determined. As for the scale and shape parameters in the adaptive generalized loss function mentioned above, this threshold could be allowed to take different values for different samples. An adaptive threshold could be suitable for the type of data considered here, which can be blocked into subgroups where the variation is smaller within than between groups.

By dividing all training samples into groups by similarity (e.g. by location and climate scenario) an outlier threshold can be calculated for every sample group, according to a chosen formula, e.g. 6.1, and stored in a vector. The adaptable threshold can then be implemented by constructing a threshold vector as an input of the same shape as the target tensor. So every entity in that input tensor holds the sample specific threshold based on the distribution to which the output, in the corresponding entity in the target tensor, belongs. Note that the test-set is assumed to be unknown, which precludes calculations of a test-sample specific threshold. Hence this threshold should only be applied during training. The thresholds could also be learned during training (like the adaptive α vector in the above mentioned PReLU activation function).

There exist no such activation layer in Keras, but it can implemented using the `Lambda` layer that allows for custom function transformations. (*Keras, Core Layers*).

A.4 Köppen Climate Classification System

Köppen climate classification system is by the Encyclopædia Britannica described as a *"widely used, vegetation-based, empirical climate classification system developed by German botanist-climatologist Wladimir Köppen. His aim was to devise formulas that*

Class	Conditions
A (tropical)	$min(T_y) \geq 18°$
C (temperate)	$-3° < min(T_y) \leq 18°$ AND $max(T_y) \geq 10°$
D (continental)	$min(T_y) \leq -3°$ AND $max(T_y) \geq 10°$
E (polar)	$max(T_y) \leq 10°$
B (dry)	$0.7 \cdot \sum P_y \leq \sum P_s$ AND $\bar{P}_y < 20 \cdot \bar{T}_y + 280$
OR	$0.7 \cdot \sum P_y \leq \sum P_w$ AND $\bar{P}_y < 20 \cdot \bar{T}_y$
OR	$0.7 \cdot \sum P_y > \sum P_s$ AND $0.7 \cdot \sum P_y > \sum P_w$ AND $\bar{P}_y < 20 \cdot \bar{T}_y + 140$

Table A.1: Köppen climate classification conditions for main classes A (tropical), B (dry), C (temperate), D (continental) and E (polar).[a]

[a]Clarification of B-conditions: The first condition for B is stating that 70% or more of annual precipitation falls in the summer half of the year; the first condition in the second alternative (second row in class B) states *70% or more of annual precipitation falls in the winter half of the year*; and the first two condition in the third alternative means *neither half of the year has 70% or more of annual precipitation.*

would define climatic boundaries in such a way as to correspond to those of the vegetation zones (biomes) that were being mapped for the first time during his lifetime." (Arnfield 2020).

It divides the world into five main climate classes: tropical (A), dry (B), temperate (C), continental (D) and polar (E). Their distribution is shown in figure 6.1.[1] The class conditions for A, B, C, D and E are defined using temperature and precipitation summary statistics that are either monthly, annual, taken from the summer months or winter months. Here the Kö summary statistics are based on the observed LPJ-GUESS input time series tas and pr between the years 1981 and 2009, using the conditions defined in the Encyclopædia Britannica (Arnfield 2020).[2]

The conditions for these main five categories are specified in table A.1. T stands for *air temperature* and P for *precipitation*. The subscripts $_y$, $_s$ and $_w$ denote that T or P is a vector with monthly averages over a year (y), a vector with monthly averages during the summer half of the year (s) and a vector with monthly averages during the winter half of the year (w), respectively. The summer months and winter months are interchanged between the Northern Hemisphere (NH) and Southern Hemisphere (SH): April-September are the summer months in NH and the winter months in SH; October-March are the winter months in NH and summer months in SH.

The conditions for class-belonging to A, C, D and E are only based on temperature, whereas the class dry B is defined by both precipitation and temperature. Note that the conditions for A, C, D and E defines four disjoint sets that make up the whole globe, so all locations that also satisfy the conditions for the dry class B will be classified as such. In other words, locations in class B will also satisfy one of the other classes conditions (solely based on temperature).

Non of the locations, at least not the 192 considered here, are classified as polar climate E - neither for when the observed time series of precipitation and temperature nor for when the climate adjusted time series are used.

Algorithm 2: Sample Selection

Require: Sample set $\mathcal{A} = \{(c,t,w,n,l,y)_{all} : c \in [360,510,660,810], t \in [-1,0,1,2,3,4,6], w \in [-50,-30,-20,-10,0,10,20,30], n \in [10,60,200], l = 1,2,...,L, y = 1982,1983,...,2009\}$

for $(c,t,w,n,l,:)_{all} \in \mathcal{A}$ **do**

  Draw a set of 7 unique random years $\{\tilde{y}_1, \tilde{y}_2, ..., \tilde{y}_7\}$ from $\{1982,1983,...,2009\}$;

  $(c,t,w,n,l,1)_{subset}, ..., (c,t,w,n,l,7)_{subset} \leftarrow (c,t,w,n,l,\tilde{y}_1)_{all}, ..., (c,t,w,n,l,\tilde{y}_7)_{all}$

end

A.5 Pseudocode For the Blockwise Sample Selection

A.6 Analysis of Input Variable Importance

This section exemplifies how the analysis of input importance was conducted for climate class A.

Pearson Correlation and Data Analysis

The Pearson correlation matrix A.1 show that yield and above ground biomass resp. maturity season and pre-anthesis phase are strongly correlated, but also that yield and biomass are somewhat correlated to the season length variables.

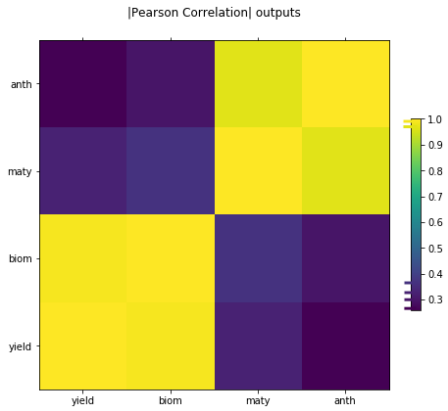


Figure A.1: Person Correlation of outputs.

One can also see that the absolute Pearson correlation coefficient matrices A.2 of the monthly weather series corresponding to yield and that corresponding to biomass are very similar. As for the case of the season outputs. Initially the network was separated into four output branches, but these can support the argument for having an initial separation into two branches - one shared by yield and biomass and the other by the season outputs - before separating the pairs at the final step, i.e. that yield and biomass resp. the seasonal outputs, share the same networks until the final layer.

It can also be seen that can see that mean R correlation coefficients, corresponding to $\bar{R}_{i,y,m}$ are the smallest for all outputs and that $P^{(\%0)_{i,y,m}}$ (%W==0) has largest correlation.

Another thing to notis is perhaps the similarities between the Pearson coefficients corresponding to $\bar{T}_{i,y,m}$ (mean T) and $T^{(Q_{10})}_{i,y,m}$ (Tql0), resp. $\bar{P}_{i,y,m}$ (mean W) and $P^{(Q_{90})}0_{i,y,m}$ (Wq90), suggesting it may be redundant to include all of these

[1]The Köppen climate classification system can also divide the five climate groups A, B, C, D and E into smaller sub-groups, but those will not be used here (Arnfield 2020).

[2]The Köppen climate classification system has been updated several times. A new version with conditions adjusted for the second half of the $20th$ century was presented in the paper *"World Map of the Köppen-Geiger climate classification updated"* published in *Meteorologische Zeitschrift* (2006).

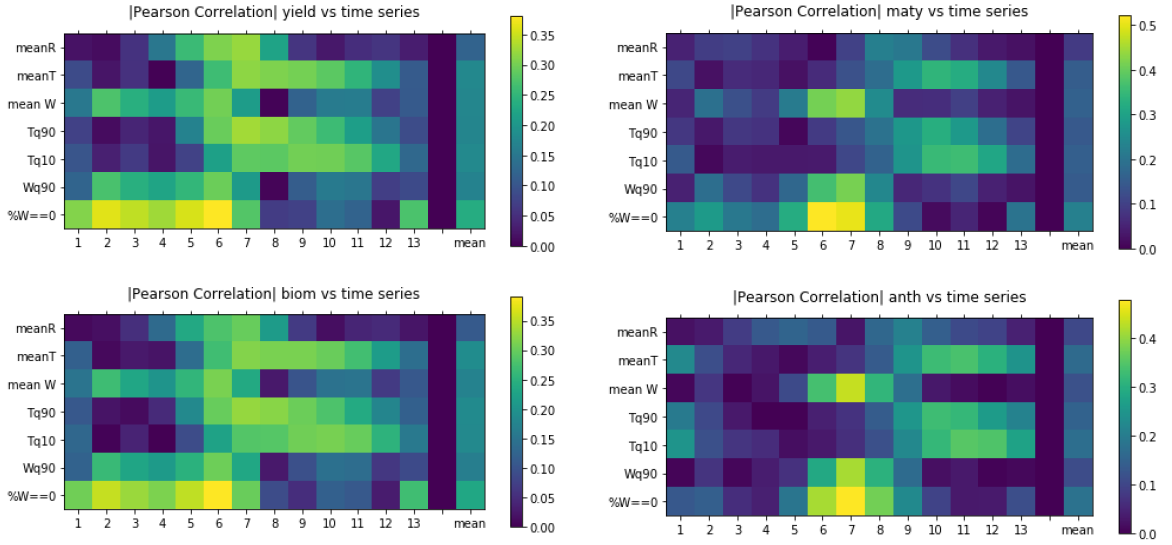


Figure A.2: Absolute Pearson correlation coefficient matrices of the time series for the four separate outputs.

variables.

The same thing could be concluded from the plot and histograms in A.3 of the three T-variables and the three W-variables (all normalized). The W- and T- quantiles seem to follow similar distributions as their mean-variables and %W==0 instead mirrors the behavior of the mean.

Permutation Feature Importance

Permutation Feature Importance was conducted according to the description in section 6.5 and the main results are presented below. The tables in A.4 display how much the performance was reduced for different permuted features.

The metrics used for importance compaison of the respective inputs where R^2, the root mean squared error (RMSE), as well as the permutation feature importance measures F_{diff} and F_{ratio} that compares the residuals $Error(Y, f(X_{perm}))$, from the network in which one input in X have been swapped (X_{perm}), with the residuals $Error(Y, f(X))$, when all inputs where accounted for. These are defined below.

$$F_{diff} = Error(Y, f(X_{perm}) - Error(Y, f(X)) \tag{A.1}$$

$$F_{ratio} = \frac{Error(Y, f(X_{perm})}{Error(Y, f(X))} \tag{A.2}$$

The tables A.4 suggest that %W==0 is the most important variable and that R never is the most important variable, but neither the least important. This might be due to it being the only variable representing radiation - in comparison to W and T having three variables each. Also the T quantiles seem to be of importance (though Tq10 less so for yield and biomass) while mean T seems to be of more importance.

The permutation results also show similarities between yield and biomass and between maturity season and pre-anthesis phase. For example, in the second table it can be seen that the permutation that has the largest effect on the season outputs is that of the latitude dummy variables (i.e. in which latitudinal interval the observation lie in), but seems to have quite little effect on the yield and biomass estimates.

Further, permutation of N and C have a very large effect on both yield and biomass, in fact the largest effect among all variable permutations (but no effect on the model estimates of maturity and anthesis due to them not being climate adjusted)

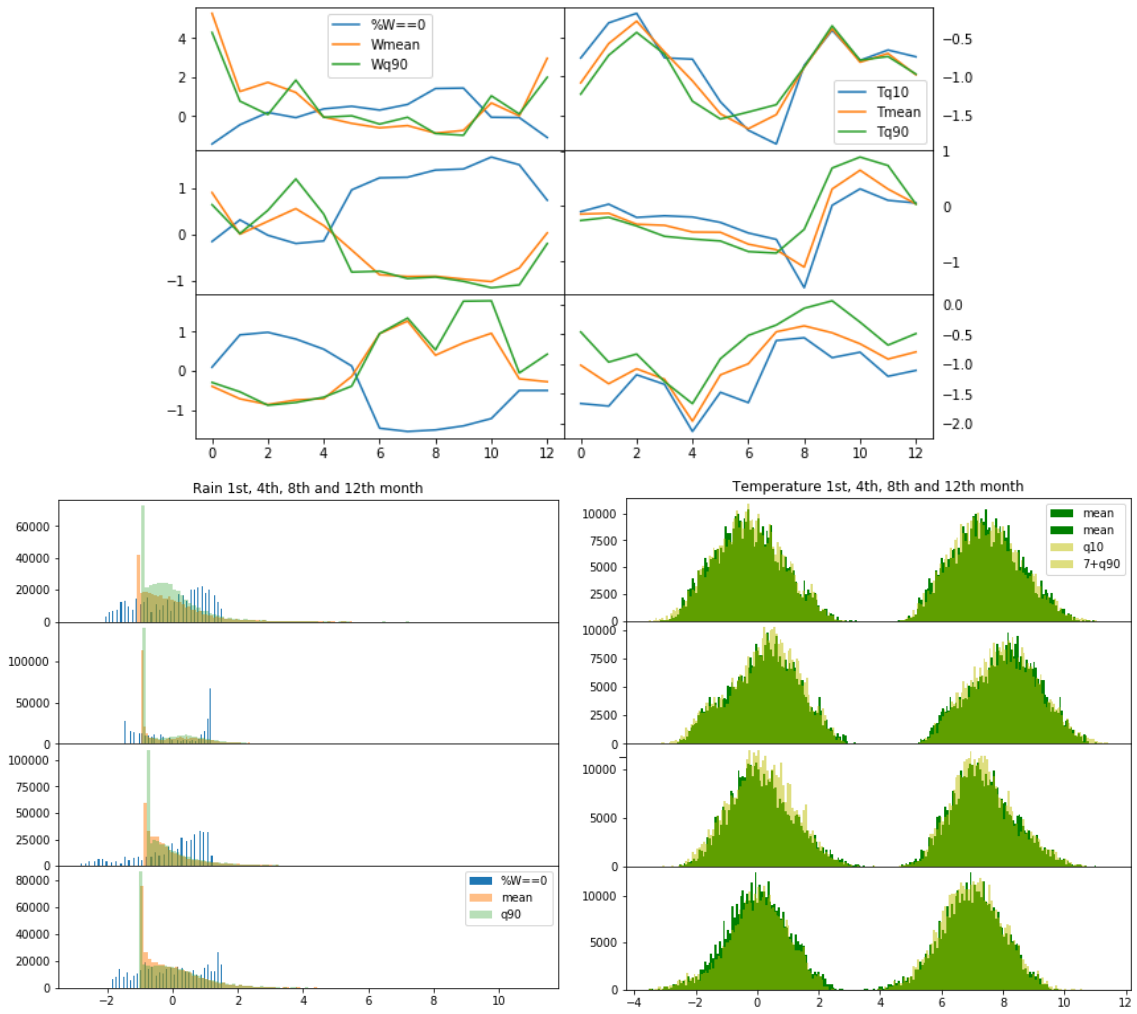


Figure A.3

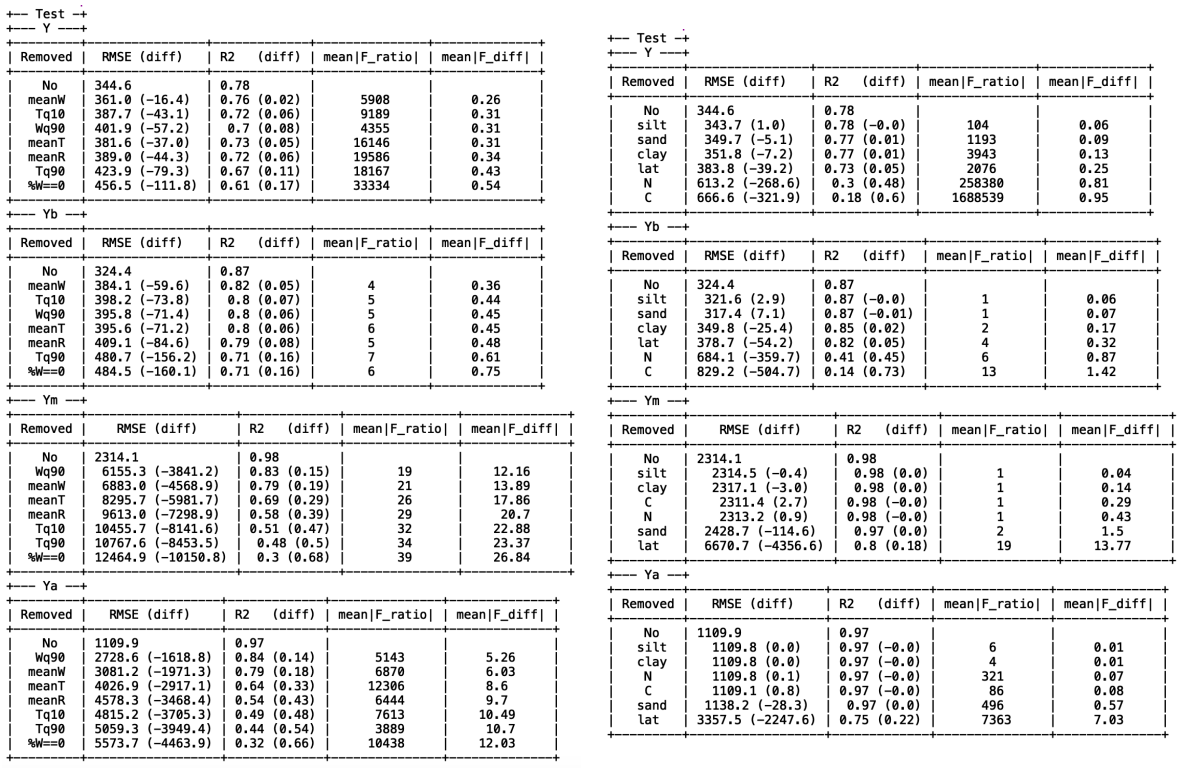
Gradient Boosting Tree

The feature importance according to a gradient boosted tree regressor trained to estimate annual spring wheat.

In the second figure focusing only on the time series could tell us which months in the time series that are of most importance. If compared with the Pearson correlation matrices we can see that the months the regression tree finds most important also are highly correlated with yield (e.g. W-variables: more importance and correlation around the end of the first half, T-variables: more importance and correlation around the beginning of the second half).

The gradient boost importance plot also show that %W==0 is most important. It is also the variable that correlates the most with the output values as well as the one that seemed to be the most importance according to permutation feature importance test.

The radiation entities in R also seem to be of importance - perhaps more so than expected in the the Pearson correlation analysis showing very small correlation between radiation and the output variables. But this result is in line with the permutation feature importance test. There might thus exist an important relation between R and the outputs - though not a simple linear correlation. Besides, in contrast to temperature and precipitation, only one input sequence provides information about radiation.



+-- Test --+
+--- Y ---+

Removed	RMSE (diff)	R2 (diff)	mean|F_ratio|	mean|F_diff|
No	344.6	0.78		
meanW	361.0 (−16.4)	0.76 (0.02)	5908	0.26
Tq10	387.7 (−43.1)	0.72 (0.06)	9189	0.31
Wq90	401.9 (−57.2)	0.7 (0.08)	4355	0.31
meanT	381.6 (−37.0)	0.73 (0.05)	16146	0.31
meanR	389.0 (−44.3)	0.72 (0.06)	19586	0.34
Tq90	423.9 (−79.3)	0.67 (0.11)	18167	0.43
%W==0	456.5 (−111.8)	0.61 (0.17)	33334	0.54

+--- Yb ---+

Removed	RMSE (diff)	R2 (diff)	mean|F_ratio|	mean|F_diff|
No	324.4	0.87		
meanW	384.1 (−59.6)	0.82 (0.05)	4	0.36
Tq10	398.2 (−73.8)	0.8 (0.07)	5	0.44
Wq90	395.8 (−71.4)	0.8 (0.06)	5	0.45
meanT	395.6 (−71.2)	0.8 (0.06)	6	0.45
meanR	409.1 (−84.6)	0.79 (0.08)	5	0.48
Tq90	480.7 (−156.2)	0.71 (0.16)	7	0.61
%W==0	484.5 (−160.1)	0.71 (0.16)	6	0.75

+--- Ym ---+

Removed	RMSE (diff)	R2 (diff)	mean|F_ratio|	mean|F_diff|
No	2314.1	0.98		
Wq90	6155.3 (−3841.2)	0.83 (0.15)	19	12.16
meanW	6883.0 (−4568.9)	0.79 (0.19)	21	13.89
meanT	8295.7 (−5981.7)	0.69 (0.29)	26	17.86
meanR	9613.0 (−7298.9)	0.58 (0.39)	29	20.7
Tq10	10455.7 (−8141.6)	0.51 (0.47)	32	22.88
Tq90	10767.6 (−8453.5)	0.48 (0.5)	34	23.37
%W==0	12464.9 (−10150.8)	0.3 (0.68)	39	26.84

+--- Ya ---+

Removed	RMSE (diff)	R2 (diff)	mean|F_ratio|	mean|F_diff|
No	1109.9	0.97		
Wq90	2728.6 (−1618.8)	0.84 (0.14)	5143	5.26
meanW	3081.2 (−1971.3)	0.79 (0.18)	6870	6.03
meanT	4026.9 (−2917.1)	0.64 (0.33)	12306	8.6
meanR	4578.3 (−3468.4)	0.54 (0.43)	6444	9.7
Tq10	4815.2 (−3705.3)	0.49 (0.48)	7613	10.49
Tq90	5059.3 (−3949.4)	0.44 (0.54)	3889	10.7
%W==0	5573.7 (−4463.9)	0.32 (0.66)	10438	12.03

+-- Test --+
+--- Y ---+

Removed	RMSE (diff)	R2 (diff)	mean|F_ratio|	mean|F_diff|
No	344.6	0.78		
silt	343.7 (1.0)	0.78 (−0.0)	104	0.06
sand	349.7 (−5.1)	0.77 (0.01)	1193	0.09
clay	351.8 (−7.2)	0.77 (0.01)	3943	0.13
lat	383.8 (−39.2)	0.73 (0.05)	2076	0.25
N	613.2 (−268.6)	0.3 (0.48)	258380	0.81
C	666.6 (−321.9)	0.18 (0.6)	1688539	0.95

+--- Yb ---+

Removed	RMSE (diff)	R2 (diff)	mean|F_ratio|	mean|F_diff|
No	324.4	0.87		
silt	321.6 (2.9)	0.87 (−0.0)	1	0.06
sand	317.4 (7.1)	0.87 (−0.01)	1	0.07
clay	349.8 (−25.4)	0.85 (0.02)	2	0.17
lat	378.7 (−54.2)	0.82 (0.05)	4	0.32
N	684.1 (−359.7)	0.41 (0.45)	6	0.87
C	829.2 (−504.7)	0.14 (0.73)	13	1.42

+--- Ym ---+

Removed	RMSE (diff)	R2 (diff)	mean|F_ratio|	mean|F_diff|
No	2314.1	0.98		
silt	2314.5 (−0.4)	0.98 (0.0)	1	0.04
clay	2317.1 (−3.0)	0.98 (0.0)	1	0.14
C	2311.4 (2.7)	0.98 (−0.0)	1	0.29
N	2313.2 (0.9)	0.98 (−0.0)	1	0.43
sand	2428.7 (−114.6)	0.97 (0.0)	2	1.5
lat	6670.7 (−4356.6)	0.8 (0.18)	19	13.77

+--- Ya ---+

Removed	RMSE (diff)	R2 (diff)	mean|F_ratio|	mean|F_diff|
No	1109.9	0.97		
silt	1109.8 (0.0)	0.97 (−0.0)	6	0.01
clay	1109.8 (0.0)	0.97 (−0.0)	4	0.01
N	1109.8 (0.1)	0.97 (−0.0)	321	0.07
C	1109.1 (0.8)	0.97 (−0.0)	86	0.08
sand	1138.2 (−28.3)	0.97 (0.0)	496	0.57
lat	3357.5 (−2247.6)	0.75 (0.22)	7363	7.03

Figure A.4: Results from permutation of every weather sequence and their the corresponding average-pooled version, one (pair) at a time in one table and the effects of permuting the other input variables separately.

A.7 Outline of Model Search

This section presents a detailed outline of model search, including the decisions made, considered problem formulations and re-formulations as well as the reasoning behind.

Initial Model Search

I started of by training a **branched CNN** on all samples. The network took the climate variables C and N trough one branch and scanned the climate adjusted weather time series in a separate branch, containing one convolution layer - where precipitation (pr) and temperature (tas) was adjusted according to the shifts in their respective climate variables (W) and (T) in accordance to description 2.2.1. Initially a large model was tested, with many nodes, and was tuned through several smaller grid searches, comparing different three-way-combinations of network hyperparameter.

When no good model could be found I decided to make use of more of the LPJ-GUESS inputs: **latitude and the soil data**. Tried models where latitude only was range-scaled, with the range-scaled absolute value of latitude (due to the almost symmetric relation between latitude and spring wheat) as well as with latitude dummy variables defined in section *Latitude as Indicator Function* 6.4. I investigated the new models in a similar fashion, but also those left much to be desired.

The problem with fitting extremes and distinguishing between different modes (which had been discernible also in the bachelor's thesis, where spring wheat at the considered locations were modeled for only one climate scenario - even when using different kinds robust resp. sensitive versions of the generalized loss function) was yet to be handled. So I searched for strategies to overcome multi-modal and long-tail problems.

One idea was to model mean annual spring wheat and shift form mean separately. The shift from mean was also very skewed and their distributions displeasing, form a modeling perspective, so I figured it was better to use only one model to

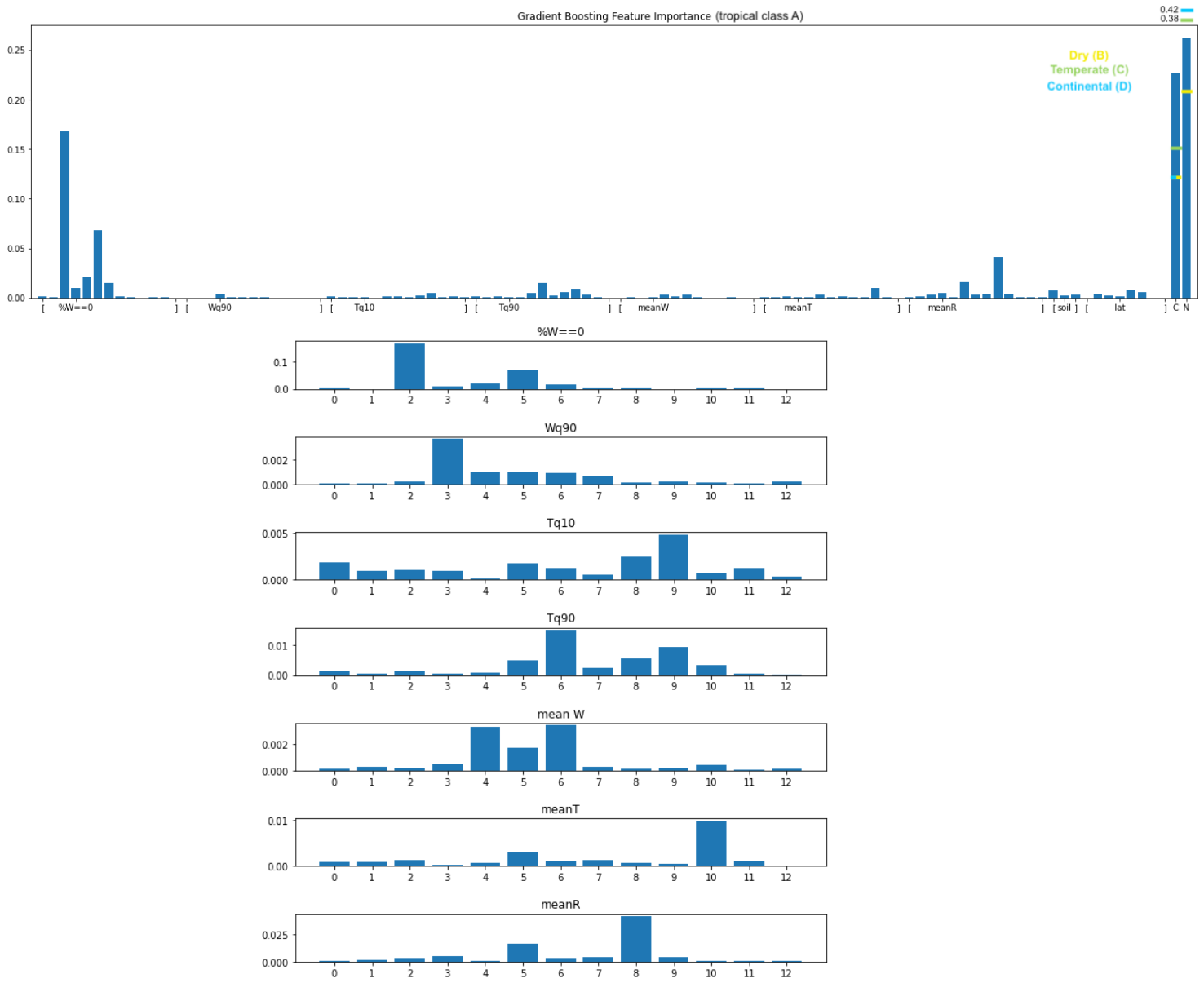


Figure A.5: Input feature importance according to a Gradient Boosted Random Forest. First show that of all variables and the second includes only the importance of the time series

predict yield on the original form, instead of using two models which would require more storage. The above mentioned difficulties eventuated in elaboration of various inputs as described in the following section.

Elaboration of Various Inputs

The Köppen climate classification system, I later found, is often used in crop related problems (see A.4). Having seen that the first, second and third level conditions in Köppen climate classification system could divide the samples into smaller groups of similarly behaving ones - which could be seen by e.g. plotting their distribution annual spring wheat as well as when investigating their relation to their corresponding inputs - I decided to make use of the definitions and summary statistics conditioned on in the classification system. I started off by taking the following 12 summary statistics:

$Rain_{(scenario,location,year)}$:

- Total rain during summer resp. winter
- Driest month in summer resp. winter
- Wettest month in summer resp. winter

$Rain_{(scenario,location)}$, same for all years:

- Expected total rain during a year (sum of rain in a year, averaged over 28 years)

$Temperature_{(scenario,location,year)}$:

- Coldest month in summer resp. winter half of the year
- Warmest month in summer resp. winter half of the year

$Temperature_{(scenario,location)}$, same for all years:

- Expected monthly average temperature (temp averaged over 28 years)

where summer and winter is defined as in section 6.2

The networks tested cared less about the daily time series and those of 5-, 10- and 15-daily averages when they also were passed some of these weather summary statistics (together with the previously mentioned inputs). As soon as the nodes and activations in the network were assigned small L1-penalties, most, if not all, weights connected to to the time series were set to zero in almost all of the networks trained.

The monthly time series presented in 6.4, stretching over almost the whole month before planting and the following year (so that it covered the longest growing season), was interchanged with the Köppen-variables and the shorter time series. Notwithstanding the prof the weather statistics being useful, they were derived from Köppen climate conditions, hence perhaps not the optimal inputs. Monthly time series on the other hand allow the network to automatically decide which features that are of importance.

The reason for including some days before planting and the vegetation state (instead of letting the time series span over the longest observed anthesis period as before) was to make use of the information that in the previously tested networks showed to be of importance as well as to get some insight in how the importance of extracted features from the time series varied over time and between different periods (like before and after planting, before and after heading, etc.).

Advantages of using larger averages over smaller: the climate adjusted daily series were shifted and scaled uniformly over the days according to the assumed annual change and are perhaps not fully representative of how the actual weather would behave. By taking larger averages of these possible unrealistic transformed inputs, the resolution is reduced and thus also the impact of any small scale deviations from reality. Another perk of lower resolution is that sample specific noise (e.g. for a specific year and/or location) can be hidden, or averaged out, in the monthly summary statistics, which can prevent the network form overfitting.

Additional Outputs

Instead of re-doing the time consuming hyperparameter optimisation of the network taking the newly defined inputs (in A.7), which even then probably could be further improved, if not completely unsatisfactory - I tested adding more outputs as well.

In light of what **multitask networks** can do for training and generalization purposes, as explained under section 4.4, as well as that the modeler can decide how much that the network should account for the separate tasks - I figured I could expand the network with the new inputs to a multi-task network before optimising it and fully evaluating the new inputs. Initially the idea was to train a multi-task neural network to model yield as well as classifying it to sub-intervals:

$$\{y : y = 0\}, \quad \{y : y \in (0, y_{extreme})\}, \quad \{y : y \in [y_{extreme}, \infty)\} \tag{A.3}$$

but that would not solve the long-tail problem. An option could be to use more intervals (however, the first would still contain the vast majority of the samples, which may or may not be a problem - at least the other sub-intervals will differ less in size,

but probably still be larger than the one containing the extremes)

$$\{0\}, \quad (0, y_1], \quad (y_1, y_2], \ldots, \quad (y_{n-1}, y_n = y_{extreme}), \quad [y_{extreme}, \infty) \tag{A.4}$$

Such network would perform both regression (of LPJ-GUESS yield simulations) as well as classification, where the first mentioned task is to model one continuous output and the latter is to predict probabilities of class-belonging.

Instead chose to make use of the lower prioritized simulator outputs that actually are of interest - *Total above ground biomass, Anthesis date, Maturity date*. These can all be formulated as a regression problem.

Annual spring wheat, total above ground biomass, anthesis date **and** ***maturity date*** are clearly related (both in relation to each other and input-wise) but are measured in two different units. Yield and biomass are measured in $ton/(ha \cdot year)$ and the other in number of days, hence opening for analysis of how the network benefits from modeling outputs with shared metrics and outputs measured in different units. Further, yield and biomass are similarly distributed - and the same applies to the other outputs. The distribution of one type of output looks like a shifted version of the other measured in the same unit and their min-max-range transformations are almost equal in distribution.

Model Composition of Kppen Climate Models and Input Selection

Investigation of input importance was conducted according to section 6.5 with separate sample analysis for the four climate classes A, B, C and D derived from the climate adjusted and .

Discussions regarding **aggregation the model domain**, mentioned in 7.1 fed the idea of designing an the emulator as a composition of domain specific sub-models. The previous analysis involving the Köppen climate classification system showing that the yield samples could be divided into smaller groups of similarly behaving samples lead to some testing of separate models predicting crop for the different climate classes - and they proved to be much better at producing good estimates.

The aggregation strategy used for those models tested were the one allowing for different climate scenario samples at the same location to be separated into different climate classes (among A, B, C and D), mentioned in section 6.2. If these four model domains were modeled separately, the total number of network parameters required for these climate-models together to achieve good model scores, were about the same as the number of parameters needed for one network used for the whole domain to get as good of a model score. When the different network estimates where visualized together with the targets, it became however apparent that the estimates produced by the large network were much worse, due to it not being able to e.g. distinguish between modes. The residual analysis also supported that when the different climate-class locations were modeled separately, much better estimates could be produced.

I thereafter decided to focus on creating four different **multi-output network, one for each of the main climate classes** A, B, C and D (not E because no samples was satisfied those conditions), but using the other classification strategy based on only the observed precipitation and temperature (and not the climate adjusted versions) that sets all samples related to a specific location in the same climate-class.

When I trained models for each separate climate class, I only included the most important features (according to previous data and feature analysis) for the climate class considered - which worked fine in terms of model performance - but for simplicity I later had the four models take the same inputs

Final Changes and Hyperparameter optimisation

The above reformulations and data analysis narrowed down the options to the set of architectures for branched multitask neural networks, given that the emulator were to be constructed as a model composition of four models with disjoint climate class domains. Continued model analysis and tuning though hyperparameter optimisation lead to the decisions and considerations below.

- **Removed the date-outputs**. For the reasons mentioned at the end of section 7.2.

- **Change of inputs**. Initially I chose to introduce the change in T and W by shifting the time series according to 2.2.1, but no tested network were able to properly model the crop response to changes along the T- and W-dimension. The network proved to predict better across the shifts in climate if the shifts of T and W were introduced separately, as the climate variables C and N and only the observed weather series were used as inputs and - instead of using the climate adjusted time series.
- **He uniform weight initializers**. The networks tested sometimes stopped learning, especially when extra layers were added. He weight initialization, both for when assuming normal and uniform distribution, solved this problem. He initializers are described in section 5.2.3.
- **Hyperparameter optimisation using** using first probabilistic search in Talos of several network structures. Tested using several dense layers after the input concatenation with different distribution of the node - equally many nodes in all layers, nodes descent in consecutive layers, to different extents. The results suggested networks with either a single layer or with several layers with *brick-shape*, i.e. with equally many nodes in each layer.

  The narrowed down set of networks where then compared through a grid search. Among the considered hyperparameters one to four number of dense layer before splitting into two output branches was compared, with several combinations of number of nodes (mainly brick-shape, but also with some node decent in following layers). Also here brick-shape tended to be best. Often one single dense layer could yield equally good or even better model scores than the networks with several layers - however, when the residuals were analysed, the multiple-dense-layered networks (with approximately the same number of model parameters) often seemed to account for more climate-variable dependencies, even when they had lower overall model scores. For simplicity, only a one such dense layer was used in the climate-networks.
- **Adaptive upper threshold in final ReLU wrapper**. I constructed an adaptive threshold based based on the quantile threshold, as described in section 5.2.1, for the simulated annual yield between 1982 and 2009, at each location and climate scenario, separately. Then included optimisation of the threshold parameter c (together with larger grid search considering other parameters as well), that involved a c-parameter sweep between 0.5 and 3 as well as 0 (no upper threshold). Sometimes inclusion of threshold lead to better model scores (mainly for. B-climate class models when tested), and sometimes not. Better results could perhaps have been attained if other thresholds (than the Tukey) had been defined. I did not investigate this any further, due to time restrictions and no obvious improvement, and choose to go on without the threshold.
- **Exclusion of some yearly observations**. To prevent overfitting I randomly selected yearly observations from every respective location and climate scenario, as described in section 6.3. This exclusion of random samples removed much of the sequential dependence between the consecutive samples and really improved the generalization.
- **8000 samples in one batch**. Batch size turned out to not be of that great importance in terms of performance (it has of course an effect of the training time). The batch size was set to 8000 for all climate classes. Smaller batches often resulted in longer training and a loss trajectory that never stopped oscillating (though, as expected, less and less for each epoch). A smaller batch size (of around 2000) sometimes resulted in improved overall model scores, though not by much.
- **Pseudo Huber loss with $\delta = 0.3$ and handling of learning rate**. Grid search evaluating networks using the losses *MSE*, *log-cosh* and *pseudo Huber* with $\delta = 0.3$, for different *initial learning rates* and *factors of how much they should be reduced* with when the loss has converged on a plateau (see the learning rate section 5.2.7). Having seen the typical trajectory of losses computed after every epoch from previous runs (using a batch size of 8000), where the validation loss often started to increase if the network were not stopped soon enough (see *Early Stopping* 5.2.6), reduction of learning rate when the loss has not changed (in any direction, i.e. not even oscillated) was set to be allowed after four epochs and network training was set to be stopped when the loss had not been reduced for 15 epochs.

B

Suggested Methods for Further Research

This section proposes alternative methods that can be of use for surrogate modelling of annual crop responses to changes in climate. The first section B.1 presents results from, in the mentioned *Conclusions and Further Research*9 mentioned, extended model accounting for relations between the climate variables C,T,W and N, in the CTWN-branch. The second section B.2 describes another type of neural network than the one used here, that better handles adjacent dependencies and possibly could make use of the structural dependencies in the sample domain. Surrogate modeling can be comprised of more than just model design, like sample selection and experimental design. Methods surrounding these other parts of surrogate modeling are presented in section B.3 and B.4. The final three section B.5, B.6 and **??**regards the network training and presents methods that can be of use for especially this emulation problem.

B.1 Model Climate Variables Separately Before Concatenation With Other Inputs

These models are based the models presented in the result section and have been re-trained but not fully optimised in terms of hyperparameters. Only the hyperparameters L2-penalty and number of nodes in final dense layer before separation into two output branches and the new hyperparameter number of nodes in the additional dense layer in the CTWN-branch, have been optimised through a small grid search. That is, the models contain the optimal values of the hyperparameters just mentioned, given the other hyperparameter values, like e.g. number of nodes in the dense layer in the convolution branch, soil-branch and latitude-branch. Thus, there may exist better hyperparameter combinations and these could be found through optimisation of all hyperparameters (including both the network hyperparameters and training parameters as loss function and initial learning rate) simultaneously.

Class	Param	$Param_{ext}$	$Param - Param_{ext}$	R^2	R^2_{ext}
A	374	368	6	0.84	0.85
B	404	319	85	0.66	0.71
C	374	388	-14	0.73	0.77
D	344	370	-26	0.86	0.84
all	1496	1445	51	0.8	0.82

Table B.1: Comparison of number of parameters and metrics

The two models are compared in table B.1. Not that the new model composition extended with an extra dense layer in the nodes actually contains fewer model parameters than the presented one. The improvement isn't that great and is hardly visible in the violin plots comparing the two models' KDE residual distributions grouped by the climate classes C.3 and by location and year C.25, respectively. But then again, the extended models did improve despite not being fully optimised in terms of hyperparameters.

B.2 Multi-Directional Recurrent Neural Networks

One should perhaps account for sample correlation, rather than assuming them all to be independent, as was done here. One could for example use a type of *Recurrent Neural Network* (RNN) called *Long Short Term Memory network* (LSTM), to account

for the serial correlation between the consecutive annual yield samples.

An alternative is to to construct a network similarly designed as a bi-directional recurrent neural network for consecutive samples, but extended to operate across more dimensions - i.e. some kind of *Multi-Directional Recurrent Neural Network* (M-RNN). Such a network could operate across e.g. the climate variable axes. But it could also operate across latitude and time, or perhaps extend it to a longitude dimension and in that way make use of the dependence along these dimensions. The many possible blocking factors that could be used for this data, enables several possible M-RNN set-ups.

B.3 Bayesian optimisation

A wise sample selection may help overcome the difficulty of modeling the data considered - in which there are dependencies between samples, multiple modes in the distribution of all spring wheat simulations and varying levels of skewness. As was mentioned in chapter 3, *Bayesian optimisation* is a sequential design approach often used in global optimisation of black box functions, where only the inputs and outputs are assumed to be known - as assumed here for the simulator LPJ-GUESS.

The modeling difficulties of the unbalanced and seemingly i.i.d. data considered in this project was mentioned in chapter 7. I.e. that neural networks are heavily reliant on the data used during training and though regularization methods can prevent it from overfitting, they tend do focus on the the normally behaving samples that make up the majority of the training set and takes less notice of rare behaviours and relations. Further, the statistical power is reduced due to the lower number of examples and the rare samples are also less accounted for during training, since their related errors easily can be evened out in the cumulative loss and merely be read as noise. For the same reason, intra diversity within minority sample groups may be even harder to detect.

Bayesian optimisation can trough certain *acquisition functions* account for both *exploitation* of samples proven to be awarding and *exploration* of uncertain samples. The *acquisition function* is optimised over the emulator to get the next query point i.e. for which point in the domain we should evaluate next. This sample selection altered with emulator re-training can iterates until for as long as needed.

For more information see *A Tutorial on Bayesian optimisation of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning* (Brochu and Freitas 2010)

B.4 Bayesian Classification of Model Domain

Unsupervised Bayesian classification methods can be used for aggregation of the model domain. Below are three such methods presented in the order of growing complexity: *K-means*, *Gaussian Mixed Models* and finally a *Markov Random Field model* (MRF).

B.4.1 K-means

Given a set of explanatory variables for each sample and the specified number of classes to be created, *K-means* will try to cluster the samples into groups s.t. the within-cluster sum of squares, i.e. within-group variance, is as small as possible. (Kriegel, Schubert, and Zimek 2017, ch. 3.1.2)

K-mean assumes

- a latent field of class belongings, $z_i \in \{1, \dots, K\}$
- spatial independence, $p(z_1, z_2, ..., z_K) = \prod_{k=1}^{K} p(z_k)$
- equal abundance of each class, i.e. that $p(z_i = k) = \pi_k = \frac{1}{K}$ is constant,
- Gaussian data distribution, $y_i | z_i = k \sim N(\mu, \Sigma)$

The unsupervised clustering is done by repeatedly assigning classes using maximum a posteriori classification, that is where convergence is reached when assignments no longer change.

Require: initial k-means, or cluster centres (could be chosen randomly)
while *not converged* **do**
  Assign all data points to their nearest cluster centre;
  Update the cluster centers to the calculated mean within each cluster.
end

B.4.2 Gaussian Mixed Models

Gaussian Mixed Models also assumes spatial independence, bug are slightly more advanced in that they allow for different class properties, namely:

- different abundance of each class, $p(z_i = k) = \pi_k$
- different parameters for the data distribution of each class, $y_i|z_i = k \sim N(\mu_k, \Sigma_k)$
- spatial independence, $p(z_1, z_2, ..., z_K) = \prod_{k=1}^{K} p(z_k)$

Determining the class belongings and parameter estimates given observations, $\mathbf{z}, \boldsymbol{\theta}|\mathbf{y}$, is hard to do by a maximum likelihood, due to a complicated posterior distribution. Instead, one uses Gibbs sampling, which is Markov Chain Monte Carlo technique. The idea behind an MCMC is to sample a Markov Chain with the desired distribution as its stationary distribution. It is even more convenient to sample from the conditional posteriors, $p(z_i = k|\boldsymbol{\theta}, y_i), p(\boldsymbol{\theta}|z_i = k, y_i)$, which is exactly what is done in Gibbs sampling. The algorithm can be summarized in the following steps

1. Sample the class belongings, given parameters, $p(z_i = k|\boldsymbol{\theta}, y_i)$
2. Sample the class proportions, $\boldsymbol{\pi}|\mathbf{z}$
3. Sample the class parameters (given class belongings and class proportions), $p(\mu_k, \Sigma_k|z_i = k, \pi_k, y_i)$

B.4.3 Markov Random Field Models

Markov Random Field models (MRF) also takes spatial structure into account. It is a complex model that involves different sampling schemes and will not be explained in detail here.

They assumes a latent Markov structure

$$p(x_i|\{x_j, j \neq i\}) = p(x_i|\{x_j, j \in \mathcal{N}_i\})$$

i.e dependence between a point and its neighbours.

In short, inference for an MRF is obtained through

1. sampling from the posterior MRF given parameters, $p(x_i|x_j, j \in \mathcal{N}_i, \mathbf{y}, \theta)$
2. sampling of the class parameters, μ_k, Σ_k
3. sampling of the field parameters, α, β_k

and the model allows for investigation of the effects of

- the choice of neighbourhood, $\mathcal{N}_i$
- the effect of a constant β v.s. a class-varying β_k

where β provides information of how similar (or different) neighbouring pixels tend to be. E.g. $\beta > 0$ would mean that neighbours strive to be equal.

B.5 VAE for Compression of Weather Series

Variational Auto Encoders (VAEs) can pass the inputs through an encoder that produces new feature representations of the inputs, which in turn gets passed into a decoder that preforms the reversed process. If used for compression of the weather time series *pr*, *tas* and *rsds*, it could probably remove statistically redundant information. It could either be implemented as complementary pre-processing step before convolution layer scanning or completely substitute the convolution branch.

It is statistically solid and becomes especially interesting together with the *Generalized Adaptive Robust Loss Function*, presented in section 5.1.1. As was briefly mentioned in section A.2, this loss function was created for VAE of images and has tunable shape and scale parameters, α and c that can be optimised during training.

Usually VAEs assumes the entities in the inputs (pixels in images) to be normally distributed in which the MSE is assumed to be the the negative log of the univariate probability (of inputs) that is to maximized (see 5.1). Then the scale parameter (variance) can be adaptively selected during training. However, instead of assuming the input entities (pixels in images) to be normally distributed, we can assume them to follow this generalized distribution derived from the adaptive loss function (see 5.1.1). That is, the generalized adaptive loss function is set to the negative log of the univariate probability instead. Then both the shape and scale parameter of the VAE's posterior distribution over the entities (or pixels), can be adaptively selected during training.

B.6 Varying Robustness in Loss Function

The generalized robust loss function has tunable shape and scale parameter α and c, where the scale parameter c determines when a residuals should be considered as an outlier, in accordance to how δ determines the robustness limit in the Huber loss. Through extra tunable shape parameter α in the generalized loss', also the sensitivity to outliers can be determined.

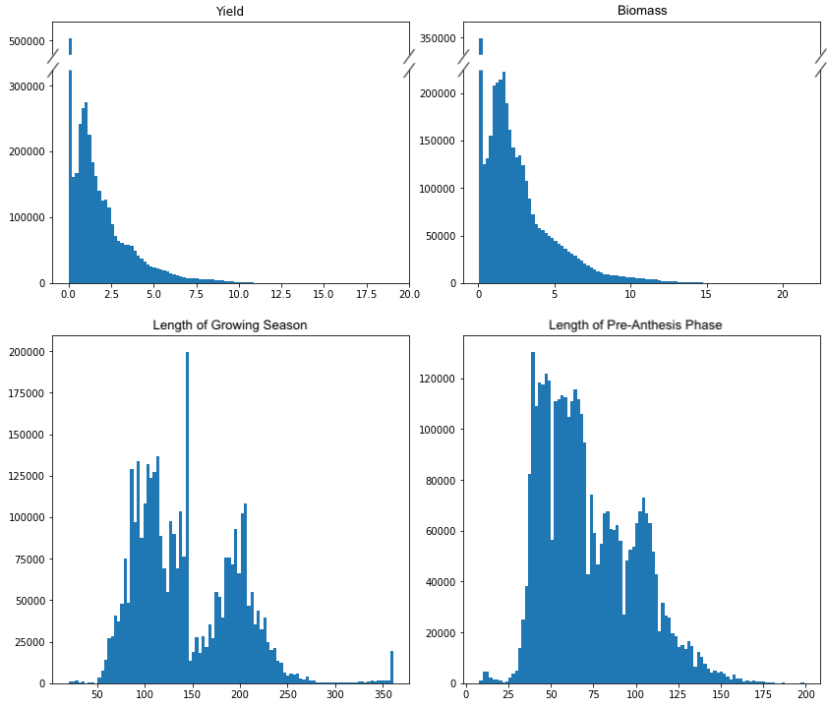
The (adaptive) generalized loss function presented in section 5.1.1 can be linked to the target distribution, in similarity to how the MSE loss is derived from normal assumptions (see 5.1). The scale parameter in the generalized loss function determines when a residuals should be treated as one corresponding to a target outlier, but since the targets are not identically distributed, it is probably hard to find a single scale parameter value suitable for all samples. Better results could perhaps be attained if the scale parameter were allowed to vary among the targets. Each scale parameter value could for example be based on the sample distribution to which the current evaluated target belongs. This in accordance with how the upper threshold in the final ReLU-wappers was determined in section 5.2.1. That is, the limit for when a loss should be considered as an outlier could be pre-defined individually for each sample group at every single location and climate scenario. This could be done by e.g. calculating standard deviation in each group or some other outlier-limit, to finally incorporate it in the loss function used during the network training.

This can also be implemented similarly to the adaptive threshold used in the final wrapper (5.2.1). That is, determining a scale (and shape if desired) for every sample group used during training - and constructing a vector of the same shape as the target where every entity holds the sample specific scale value based on the sample distribution to which the sample output in the corresponding entity (in the target tensor) belongs.

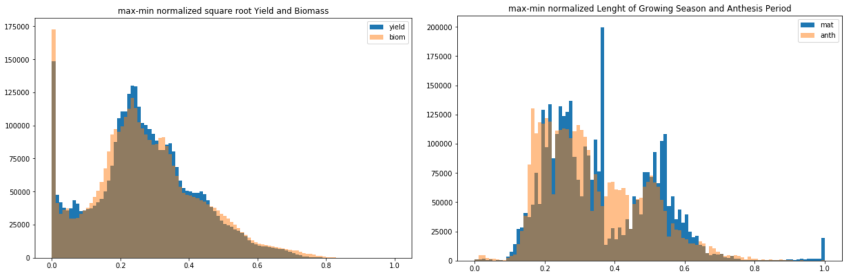
C

Figures

C.1 Data

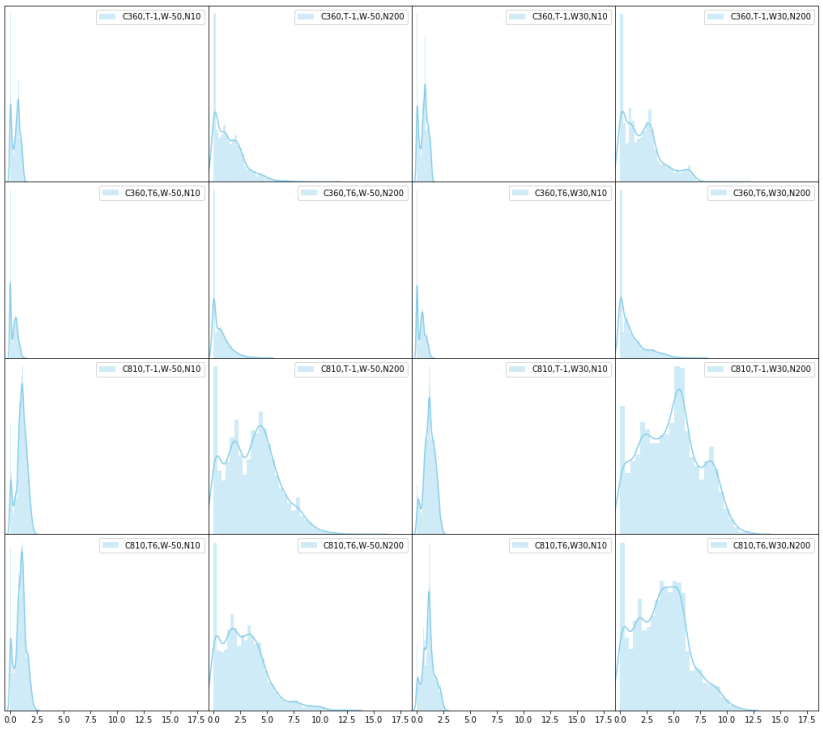


(a) Distribution of simulated annual spring wheat and above ground biomass responses to change in climate on the top. Distribution of number of days before maturity and anthesis, respectively, on the bottom.

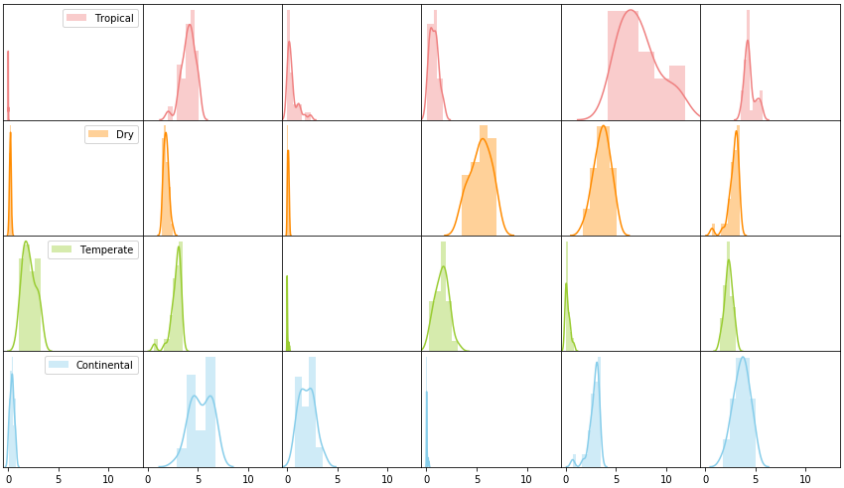


(b) Min-max range scaled outputs plotted together in pairs.

Figure C.1



(a) Global annual crop responses to different climate settings.



(b) Annual yield in base scenario $C_{360}T_0W_0N_{200}$ at some random locations, in four different climates.

Figure C.3: Distribution of simulated annual spring wheat within different sample blocks

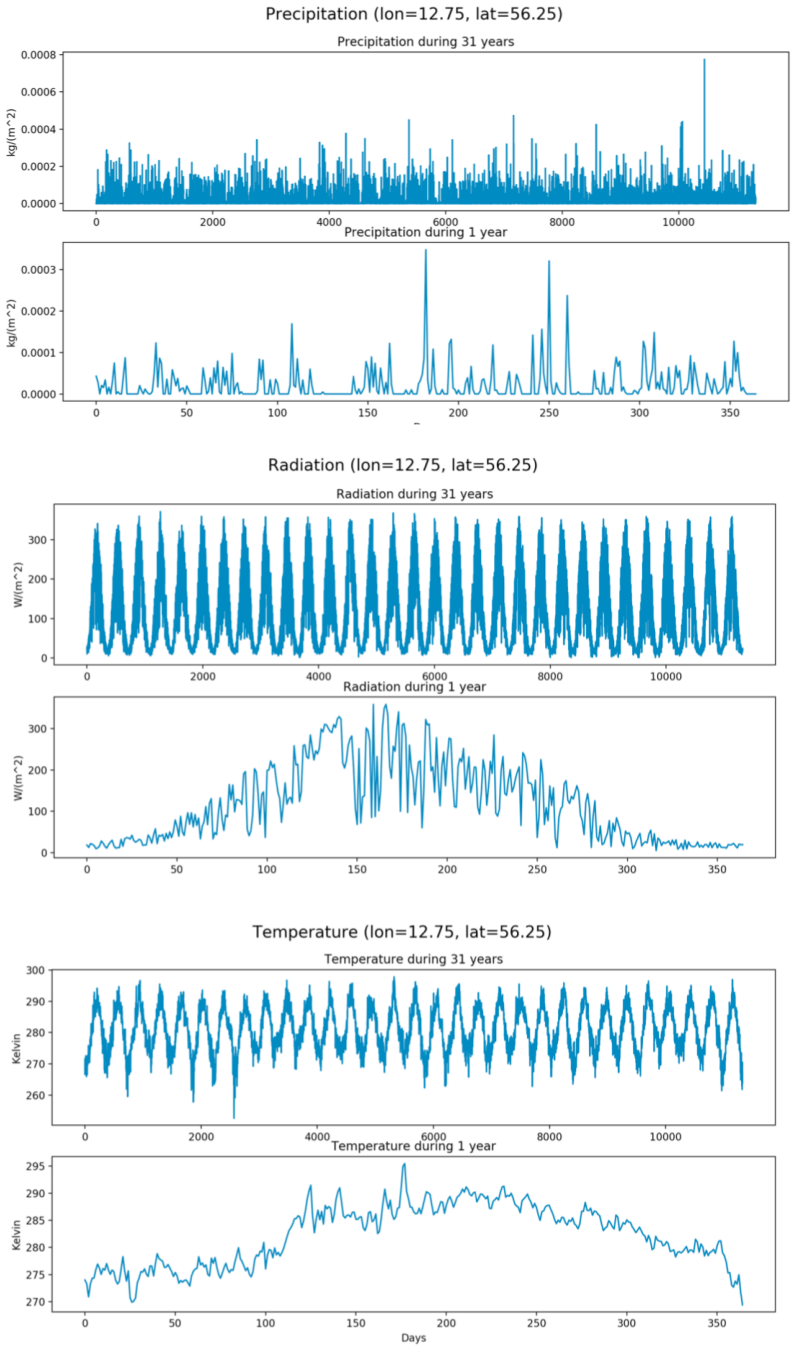


Figure C.5: Precipitation, radiation and temperature at lon=12.75, lat=56.25 between 1980 and 2010 as well as during year 2000 only (Olsson 2017).

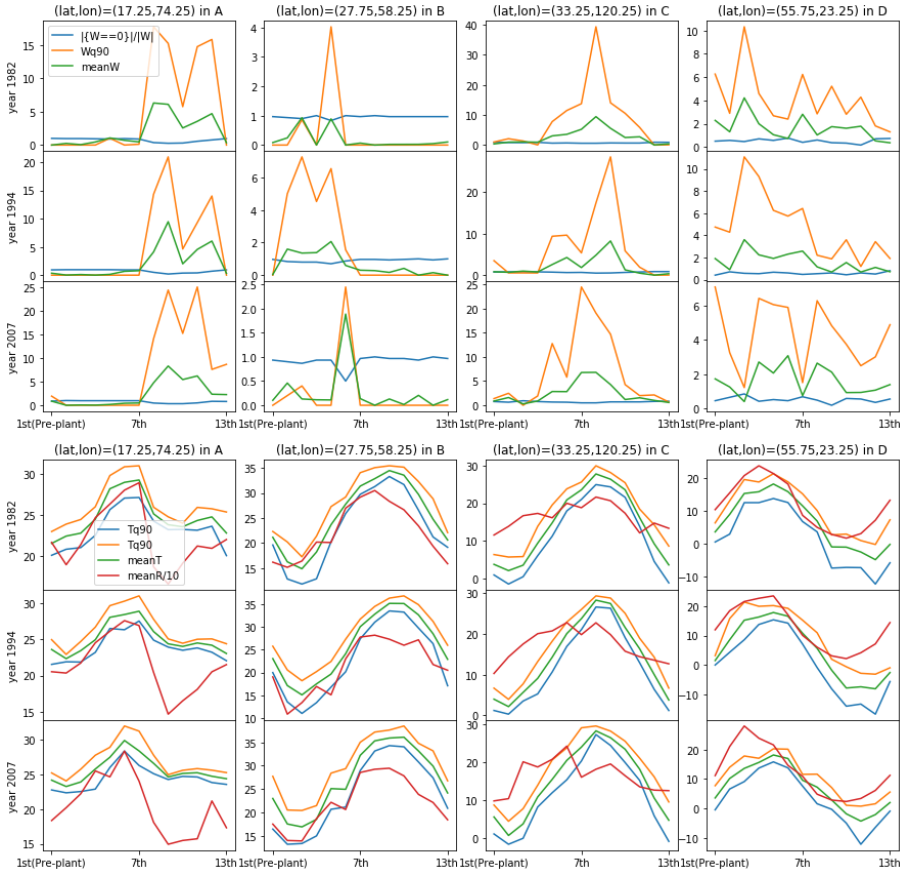


Figure C.6: The sequences of months defined in 6.4 for different locations within each climate region in 1982, 1994 and 2007. In these figures average radiation plotted in red is divided by 10.

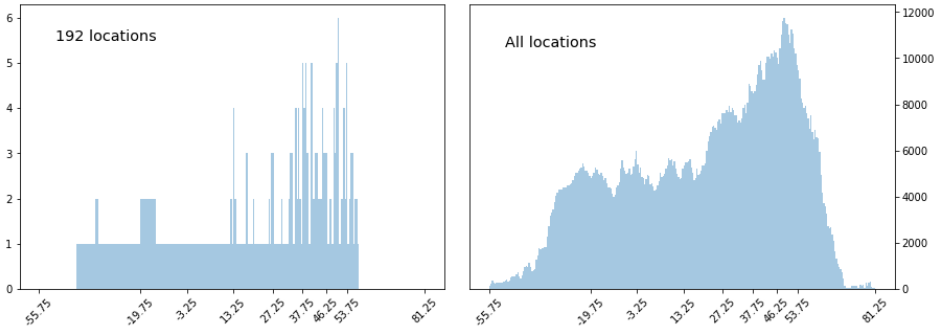


Figure C.7: Number of locations within latitudinal bands of width 0.5 for the 192 locations in used for training and testing of emulation approach (left) and the corresponding fractions among all locations considered in the GGCMI study (right). [a]

[a]The ticks on the x-axis mark the limits of the eight intervals, each of which containing around 12.5% ($\frac{1}{8}^{th}$) of all the locations.

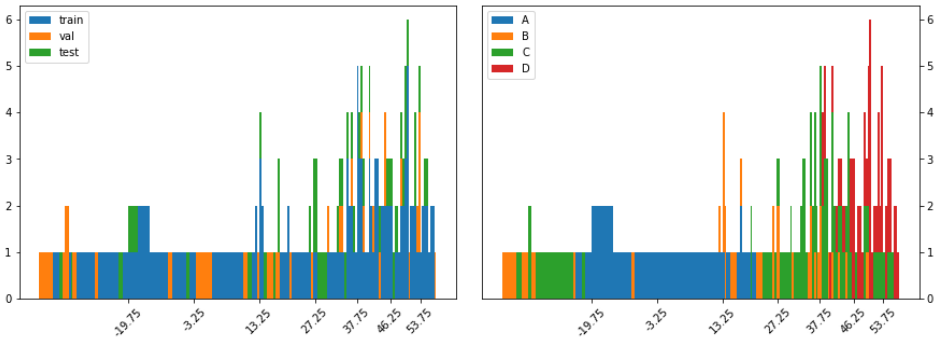


Figure C.8: Stacked number of locations within latitudinal bands of width 0.5 for the 192 for the training, validation and test set (left) as well as the climate classes (right). [a]

[a]The ticks on the x-axis mark the limits of the eight intervals, each of which containing around 12.5% ($\frac{1}{8}^{th}$) of all the locations considered in the GGCMI study.

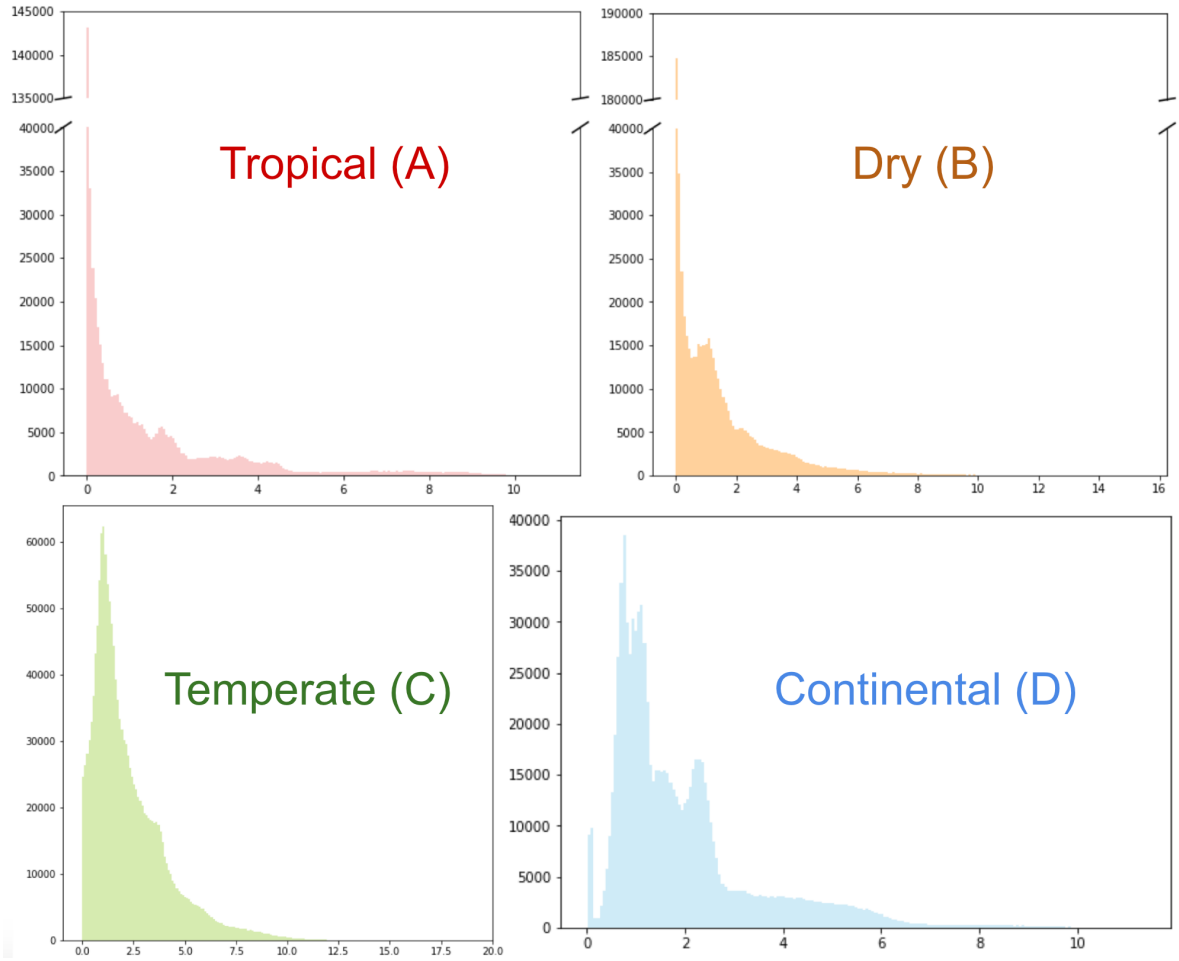


Figure C.9: The distribution of annual spring wheat in the considered Köppen climate regions.

C.2 Results

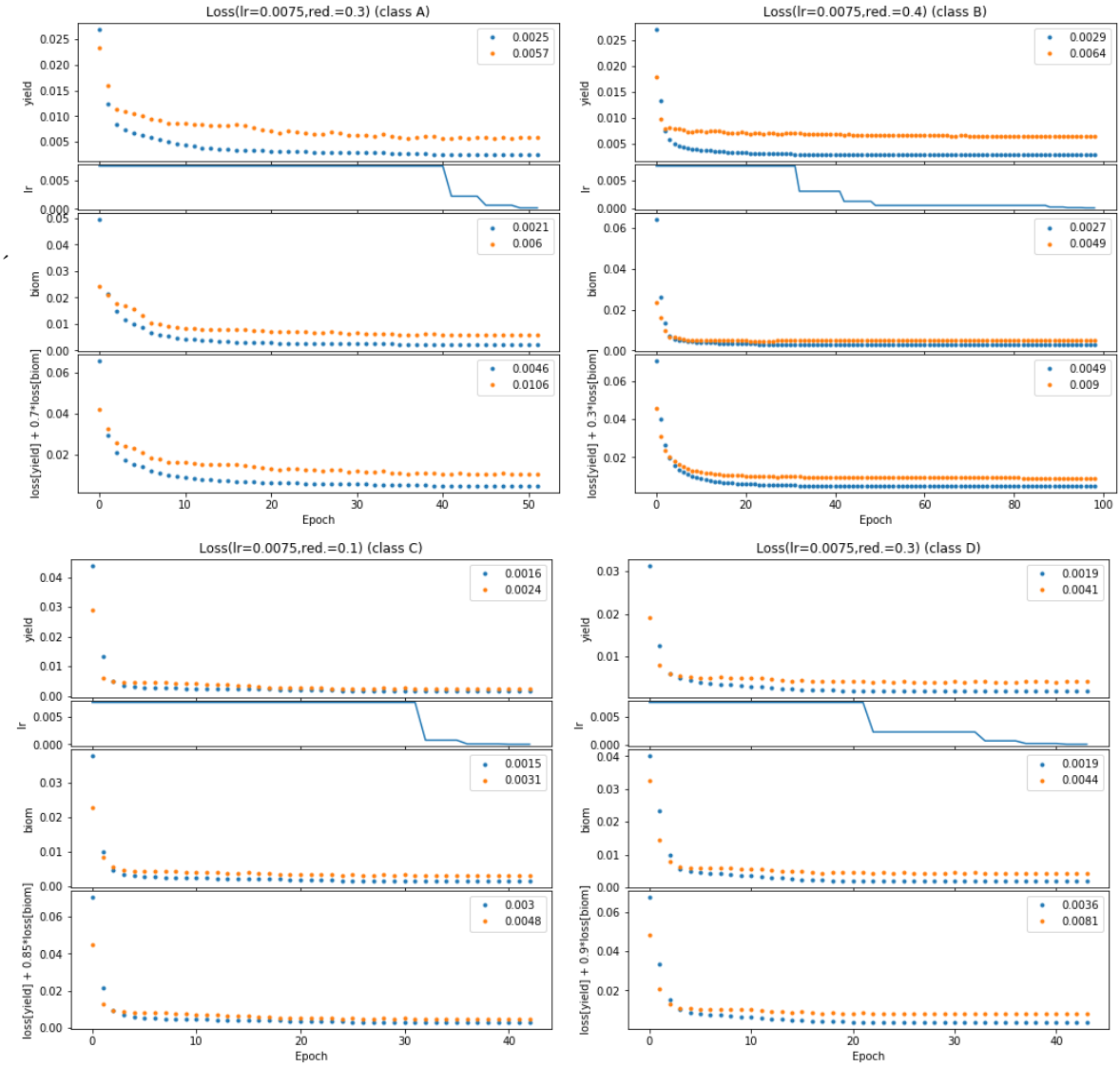


Figure C.10: The trajectories of the training and validation losses for yield, biomass and their weighted sum, for all separate climate models.[a]

[a]Also displayed is the initial learning rates, the factors of how much they should be reduced with when the loss has converged on a plateau (see the learning rate section 5.2.7). The blue solid line displays the reduction of the learning rate, The learning rate was set to be reduced at stagnation after 4 epochs and the network training was stopped early when the of the validation loss of the high priority output yield had not decreased for 15 epochs. See 5.2.6.

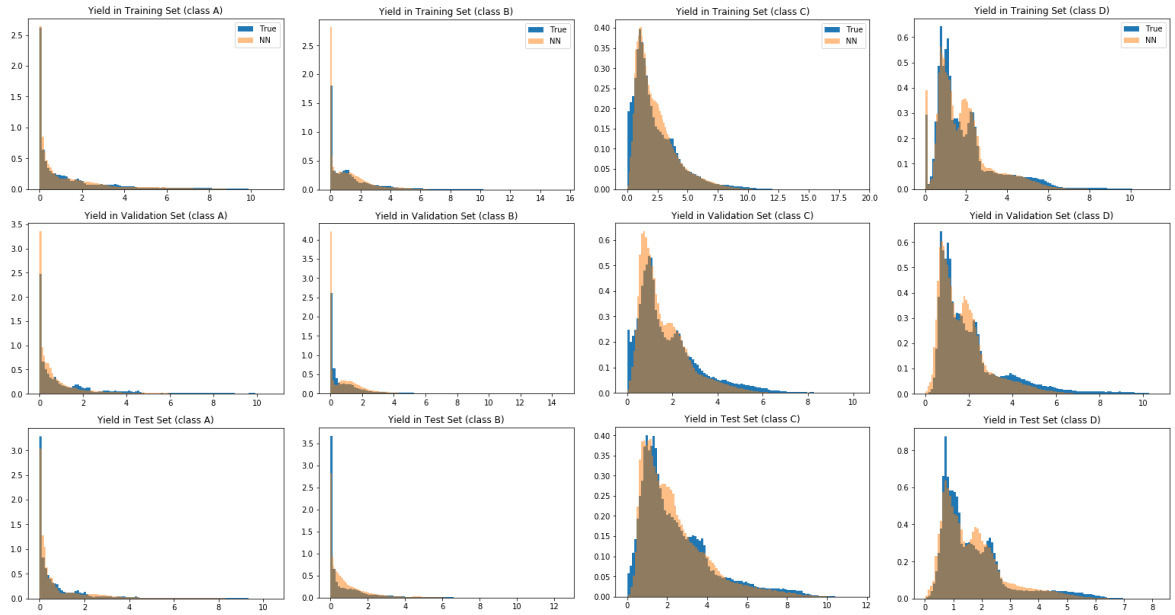


Figure C.11: The distribution of the (true) annual spring wheat simulations (measured in $ton/(ha \cdot year)$) in the test, training and validation set, respectively, together with the inverse-transformed NN estimates (measured in $\frac{ton}{(ha \cdot year}$), for the separate climate classes A, B, C and D.

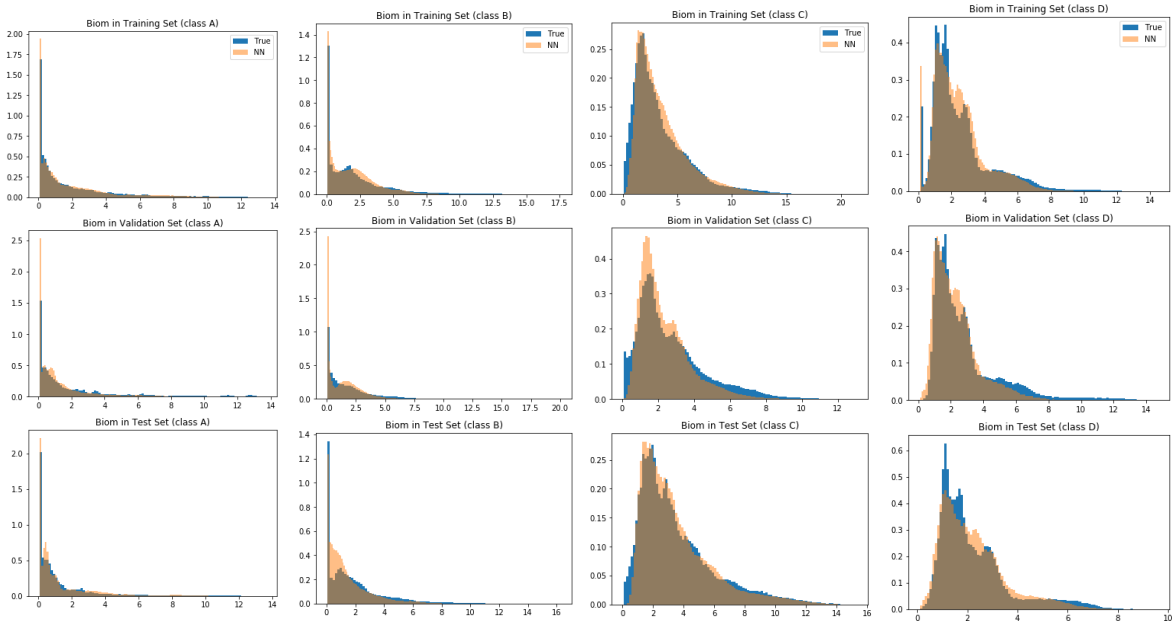


Figure C.12: The distribution of the (true) total above ground biomass simulations (measured in $ton/(ha \cdot year)$) in the test, training and validation set, respectively, together with the inverse-transformed NN estimates (measured in $ton/(ha \cdot year)$), for the separate climate classes A, B, C and D.

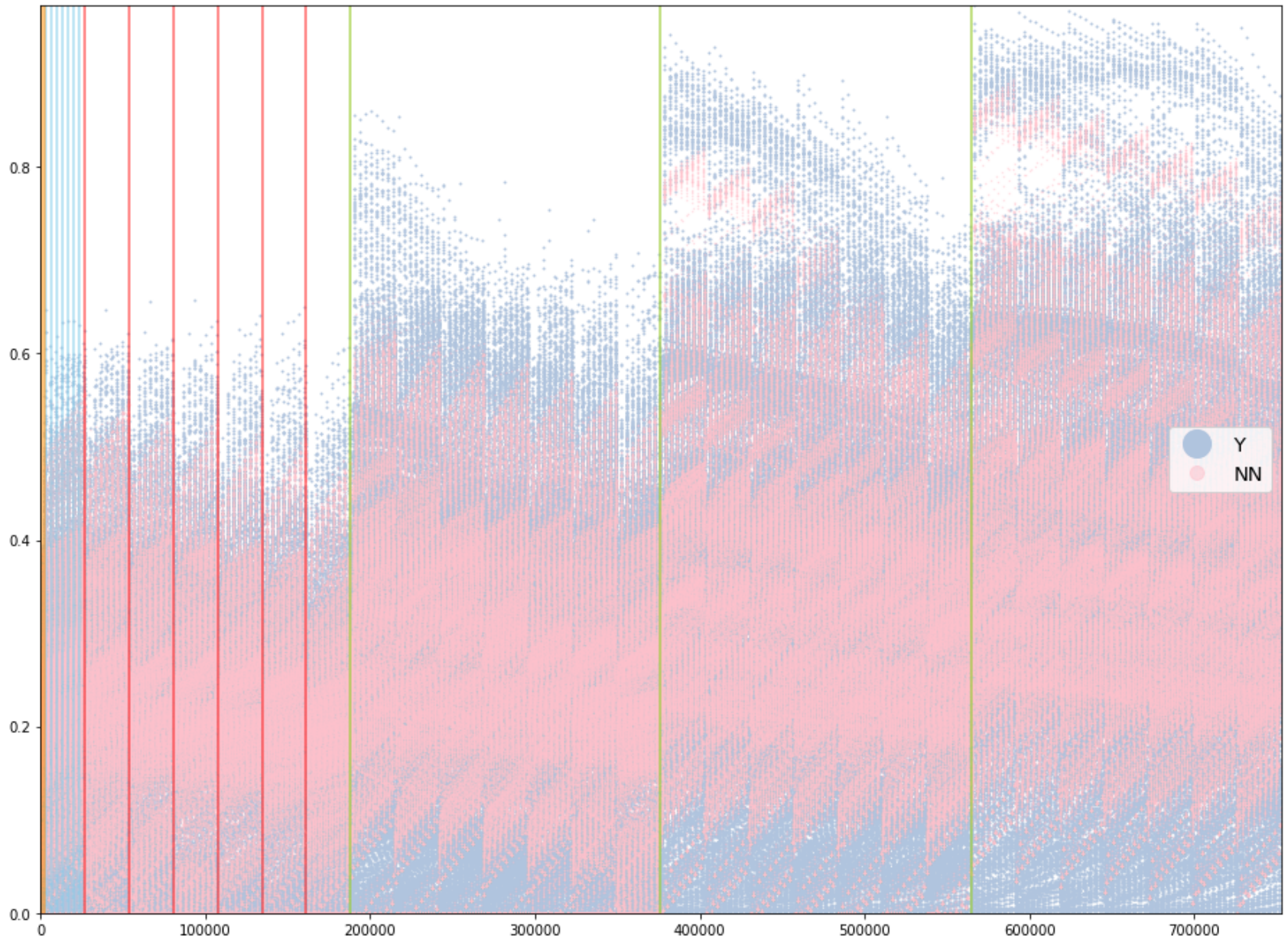


Figure C.13: The targets (Y) plotted together with the network estimates (NN) against raveled index - corresponding to a point $C_*T_*W_*N_*loc_*year_*$ in the multidimensional domain spanned by $C \in \{360, 510, 660, 810\}$, $T \in \{-1, 0, 1, 2, 3, 4, 6\}$, $W \in \{-50, -30, -20, -10, 0, 10, 20, 30\}$, $N \in \{10, 60, 200\}$, test location index $loc \in \{1, 2, ..., 40\}$ and $year \in \{1982, 1983, ..., 2009\}$ (see footnote C.14). The vertical lines marks where the climate variables changes: *green lines* marks when C changes from 360 to 510, from 510 to 660 and from 660 to 810; *red lines* when T changes to the next level for the first time (the later level changes are not visualized); *blue lines* when W changes for the first time; *orange lines* when N changes for the first time. See fractions of this plot below to for more details

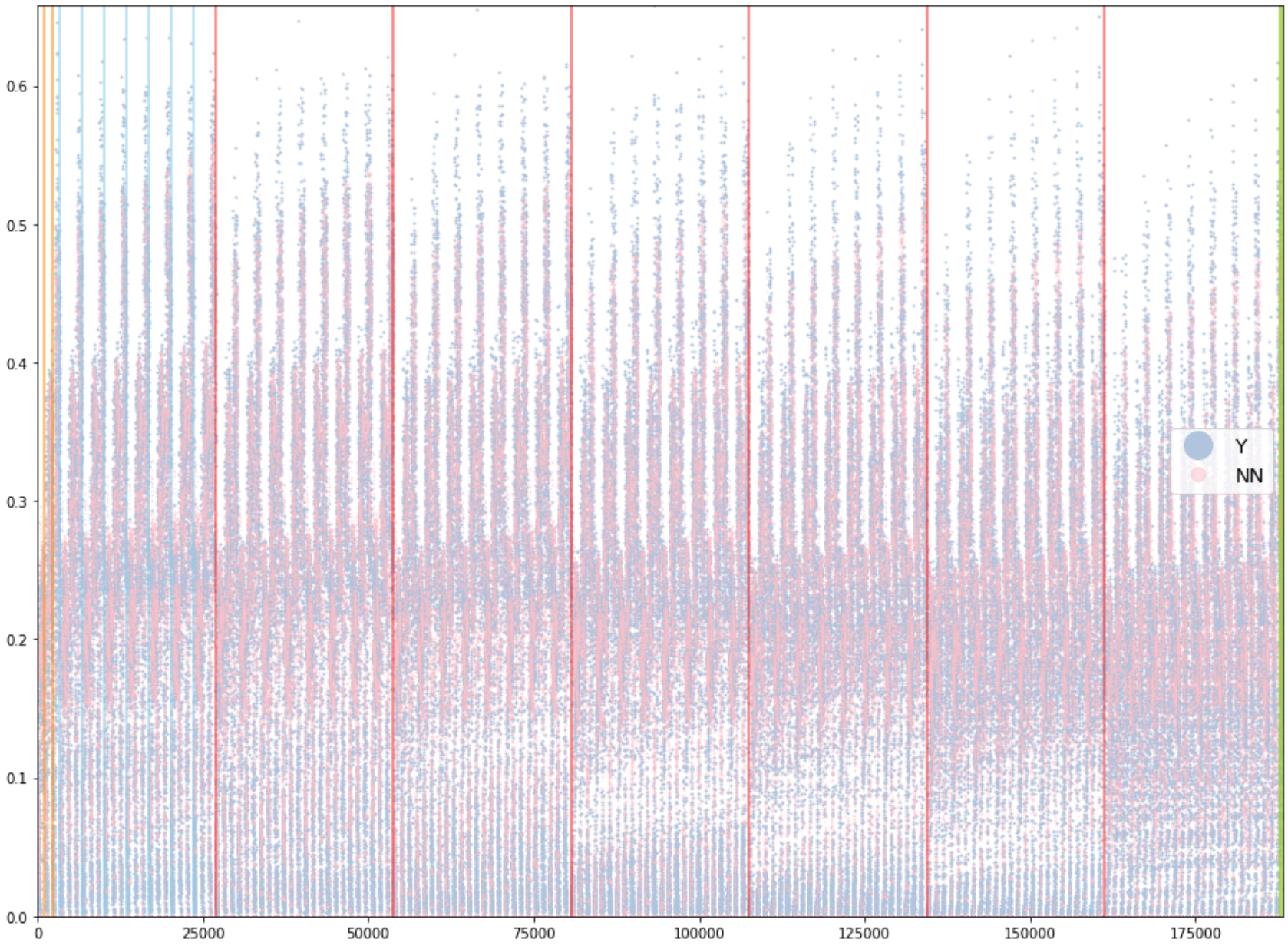


Figure C.14: Fraction of plot C.13 with the samples where $C = 360$ (the C-level in the base scenario).[a]

[a]The first index corresponds to the first level in each dimension, i.e. $C_{360}T_{-1}W_{-50}N_{10}loc_1year_{1982}$. The following indices corresponds to the following years, i.e. to the levels on the time dimension, with the other dimensions kept fixed. That is the second point is $C_{360}T_{-1}W_{-50}N_{10}loc_1year_{1983}$, ..., the 28th point is $C_{360}T_{-1}W_{-50}N_{10}loc_1year_{2009}$. The following 28 indices are the consecutive years for the next location, or *loc*-level, i.e. $C_{360}T_{-1}W_{-50}N_{10}loc_2year_{1982}$,..., the 28th point is $C_{360}T_{-1}W_{-50}N_{10}loc_2year_{2009}$ and so on, for all locations, and then repeated for all levels on the climate variable dimensions

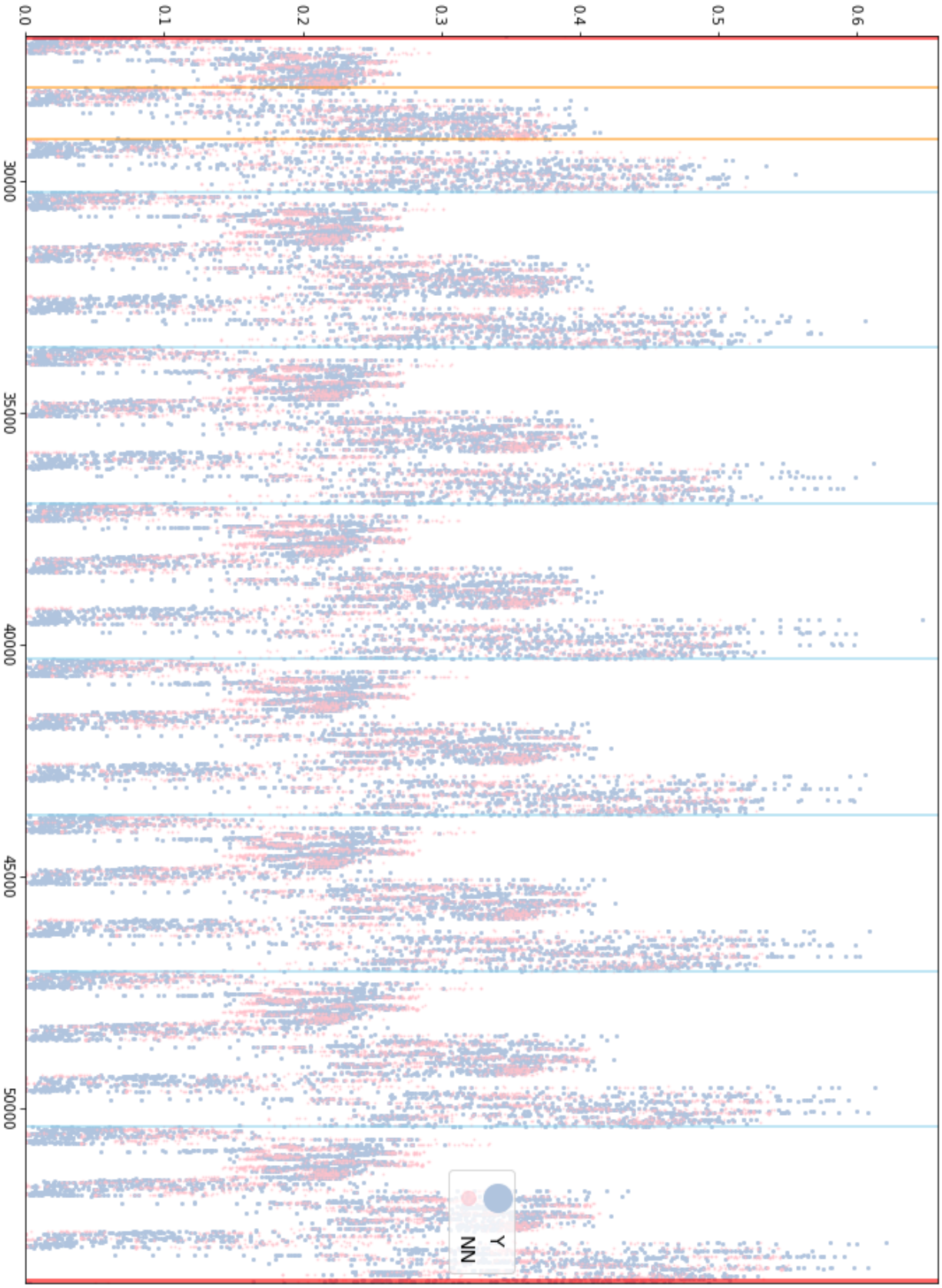


Figure C.15: Fraction of plot C.13 with the samples where $C = 360$ and $T = 0$ (the C- and T-level in the base scenario).

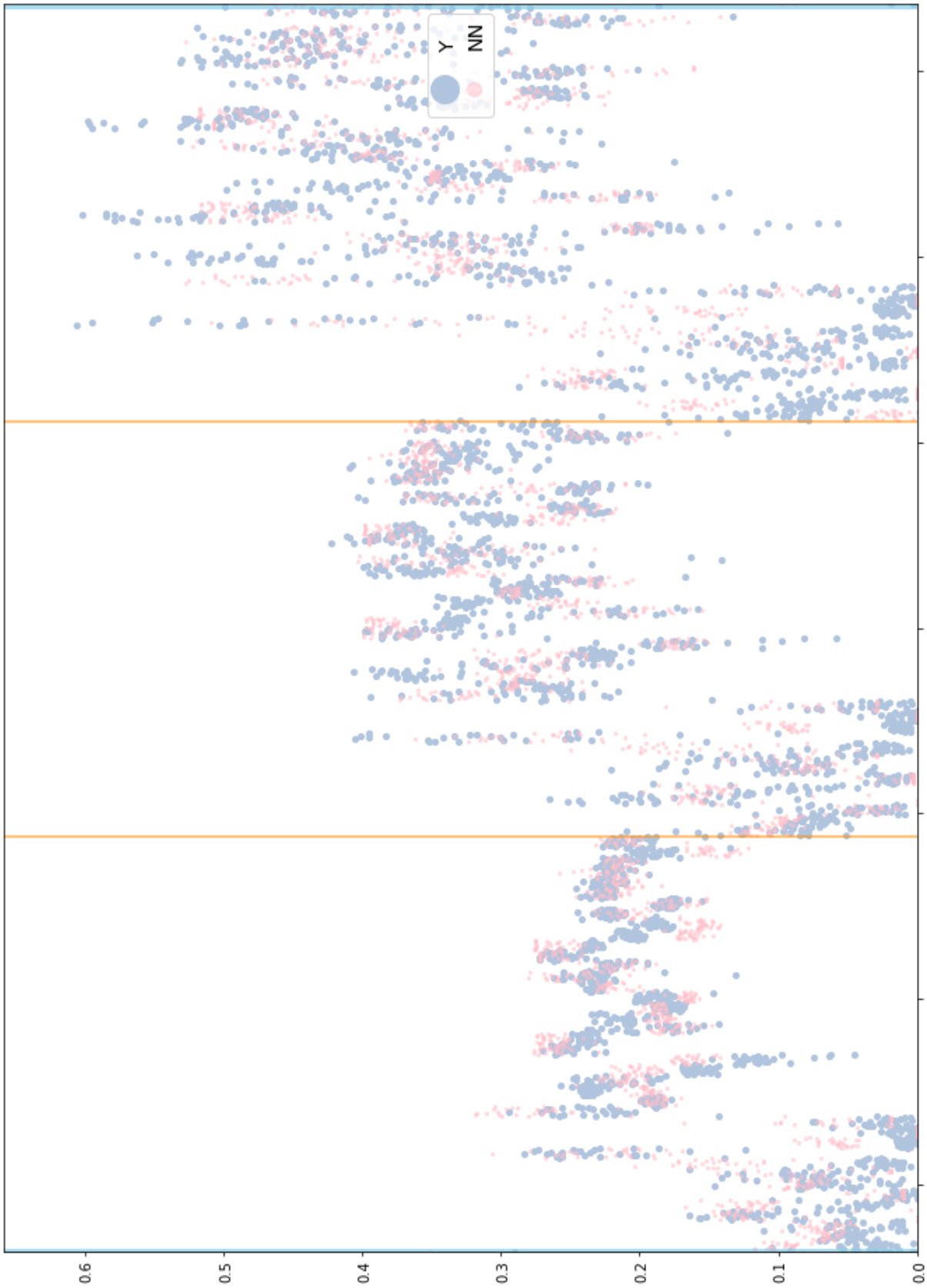


Figure C.16: Fraction of plot C.13 with the samples where $C = 360$, $T = 0$ and $W = 0$ (the C-, T- and W-level in the base scenario).

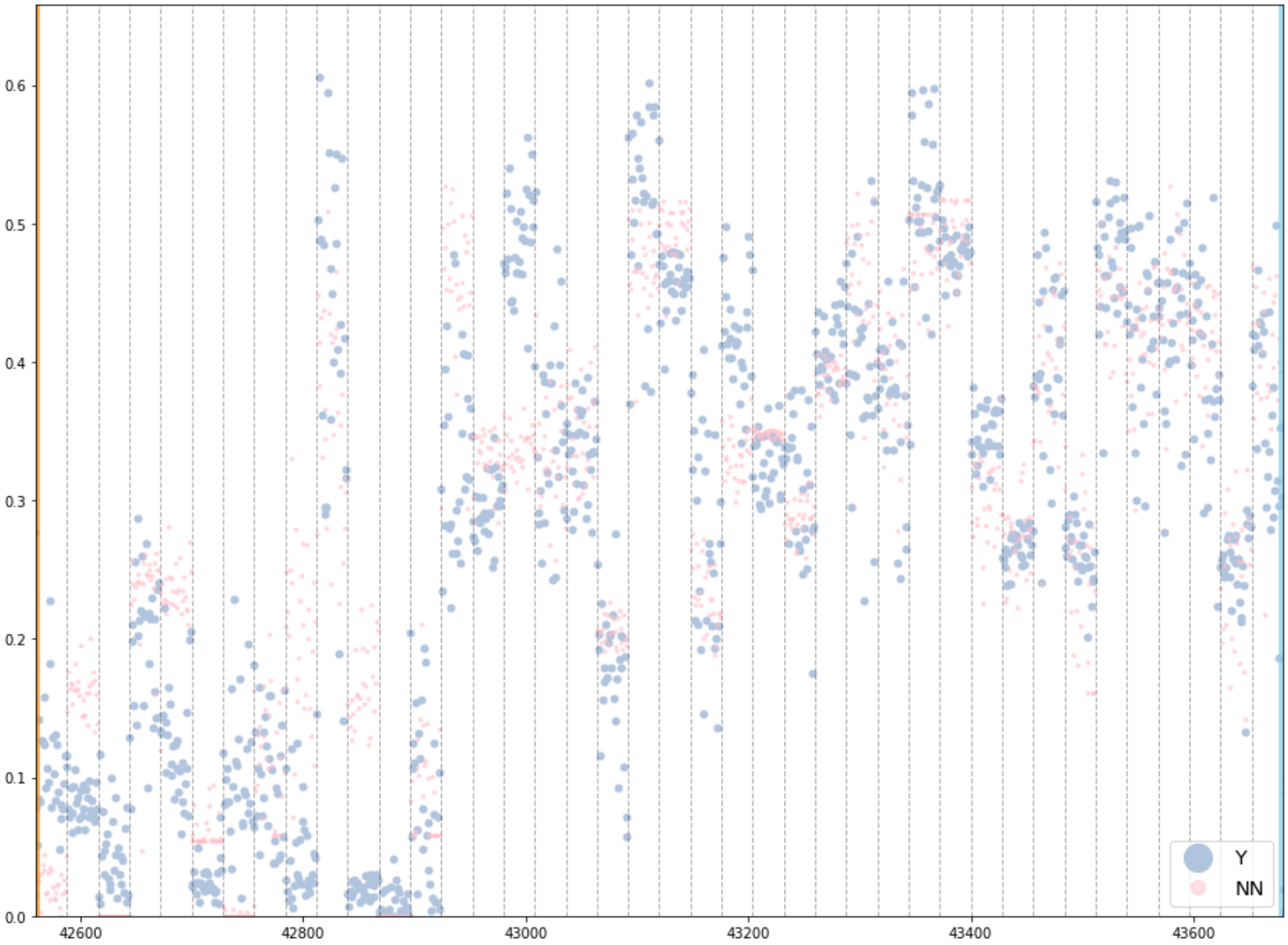


Figure C.17: Fraction of plot C.13 with the samples where $C = 360$, $T = 0$, $W = 0$ and $N = 200$ (the climate base scenario $C_{360}T_0W_0N_{200}$). The dashed lines mark the change of location, i.e. the series with annual spring wheat and the corresponding estimates lies within the dashed limits for each respective location in the test set.

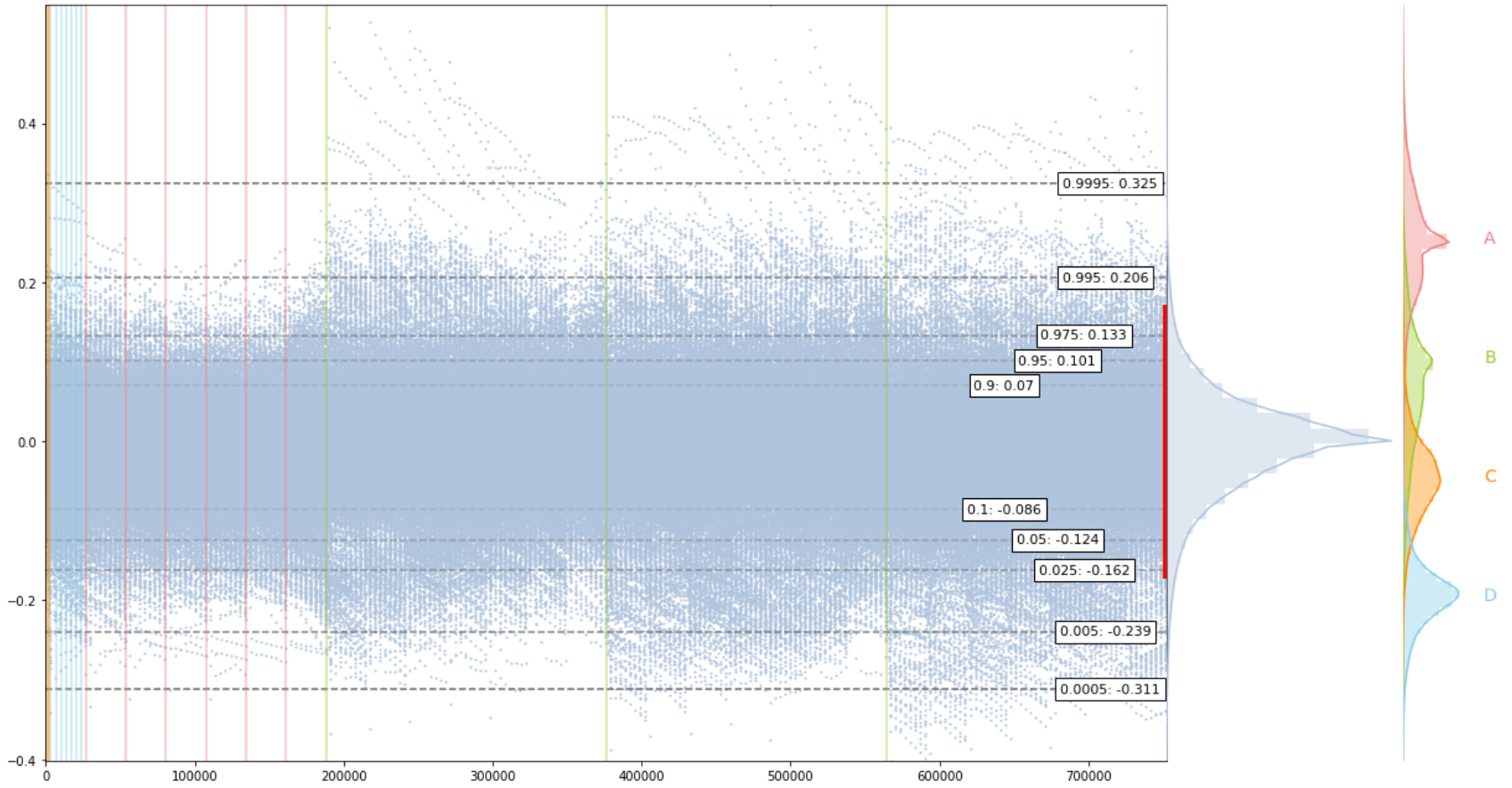


Figure C.18: The scatter plot of residuals $Y_{test} - \hat{Y}_{test}$ sorted as in figure C.13, together with the residual distribution (on the same scale) along the y-axis. The rightmost histograms are the respective residual distributions of the disjoint climate classes A, B, C and D (shifted with 0.25, 0.1, -0.05 and -0.2). [a]

[a]The *dashed horizontal lines* marks different residual percentile limits. The *red vertical line* on the axis to the right marks the interval between $[-\sqrt{Var(Y_{test})}, \sqrt{Var(Y_{test})}] \approx [-0.17, 0.17]$. The other vertical lines marks where the climate variables changes: *green lines* marks when C changes from 360 to 510, from 510 to 660 and from 660 to 810; *pink lines* when T changes to the next level for the first time (the later repeated level changes are not visualized); *blue lines* when W changes for the first time; *orange lines* when N changes for the first time.

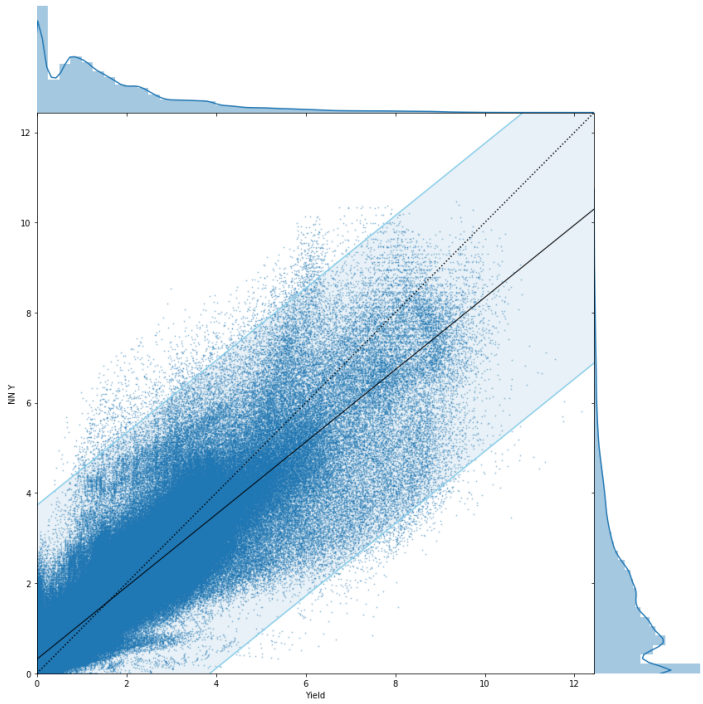


Figure C.19: The network estimated annual spring wheat plotted against the actual targets [a]

[a]Their histograms and kernel density fits along the x-axis and y-axis are. The dashed line is the perfect fit and the solid line is the actual fitted regression line. The band is a 95% confidence interval calculated for the fitted regression line (using t-student statistic with standard deviation of crop in the test set.

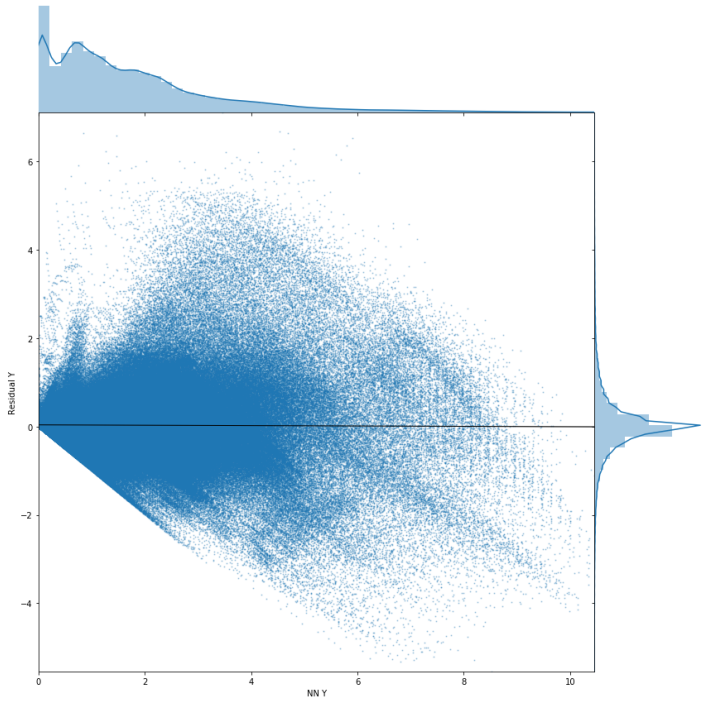
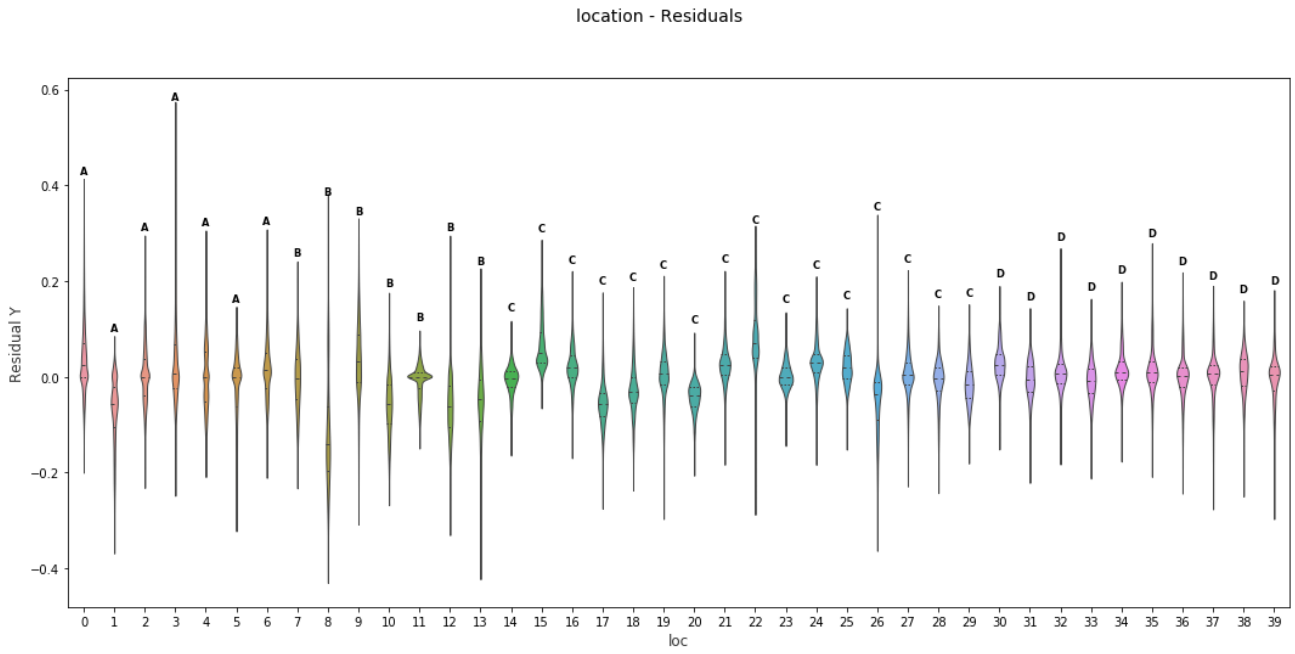
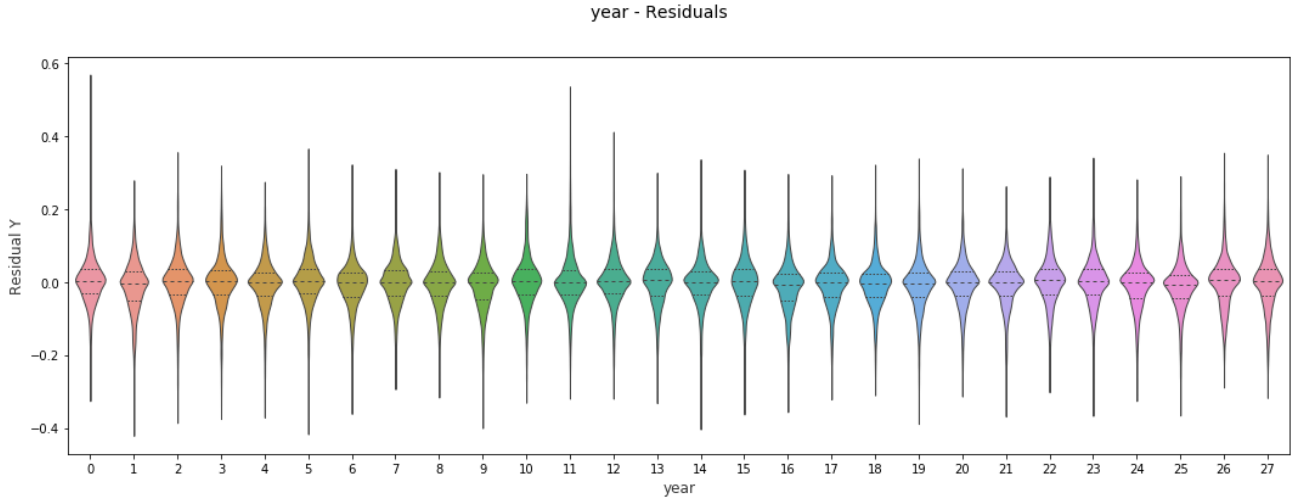


Figure C.20: The network residuals against corresponding estimates of annual spring, together with their histograms and kernel density fits. The black line is the fitted regression line.



(a) The kernel density estimation (KDE) of the underlying residual distribution at every location, marked with the climate class they belong to.



(b) The kernel density estimation (KDE) of the underlying distribution spring wheat residuals for every year between 1982 and 2009 (where 1982 is denoted with 0 in the plot, 1983 is denoted with 1, and so on).

Figure C.21: Violin plot of all spring wheat residuals in the test set grouped by location in and year respectively.

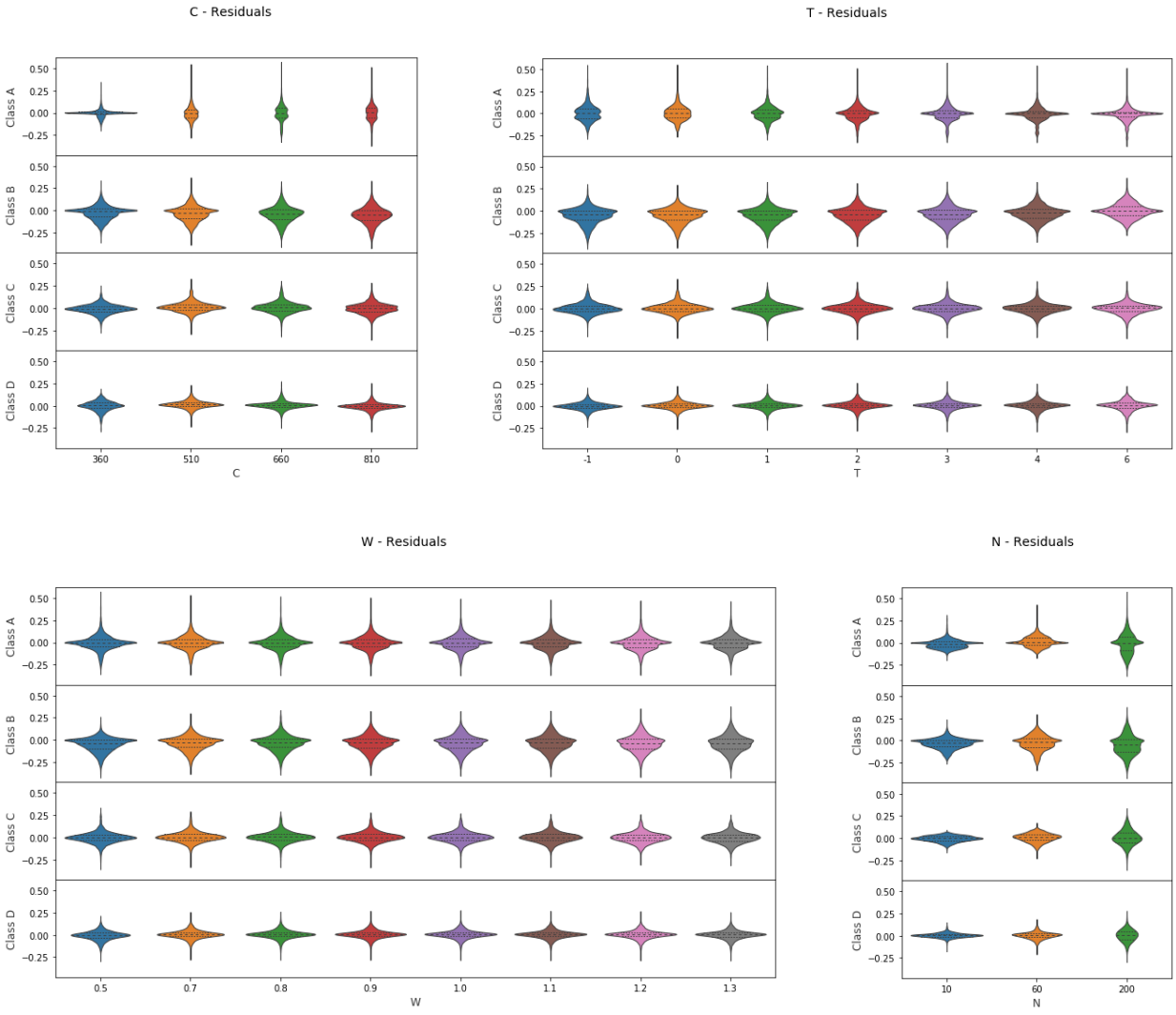


Figure C.23: Violin plots of spring wheat residuals (for the test set) in the separate climate classes A, B, C and D, showing the KDE-distribution at every factor level of the climate variables C, T, W and N, respectively.

C.3 Comparison Plots wih Extended Model

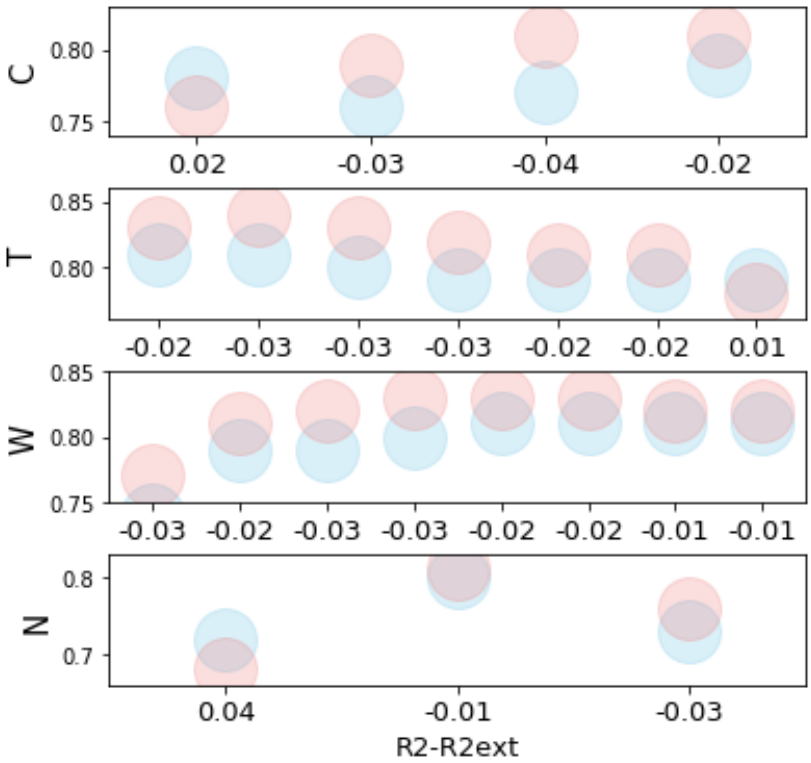
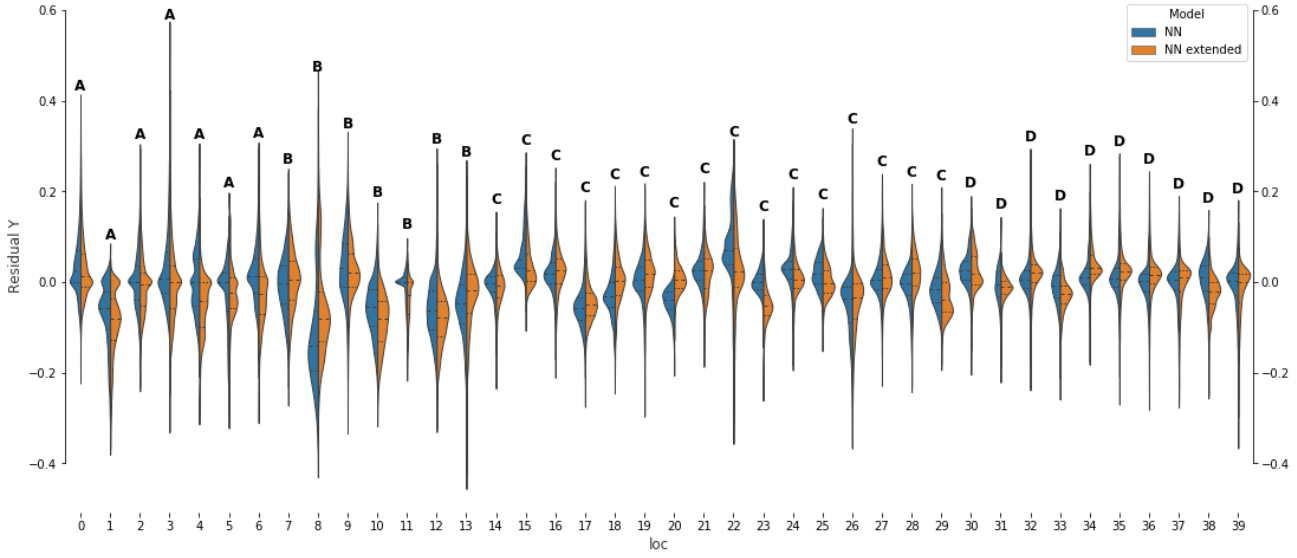
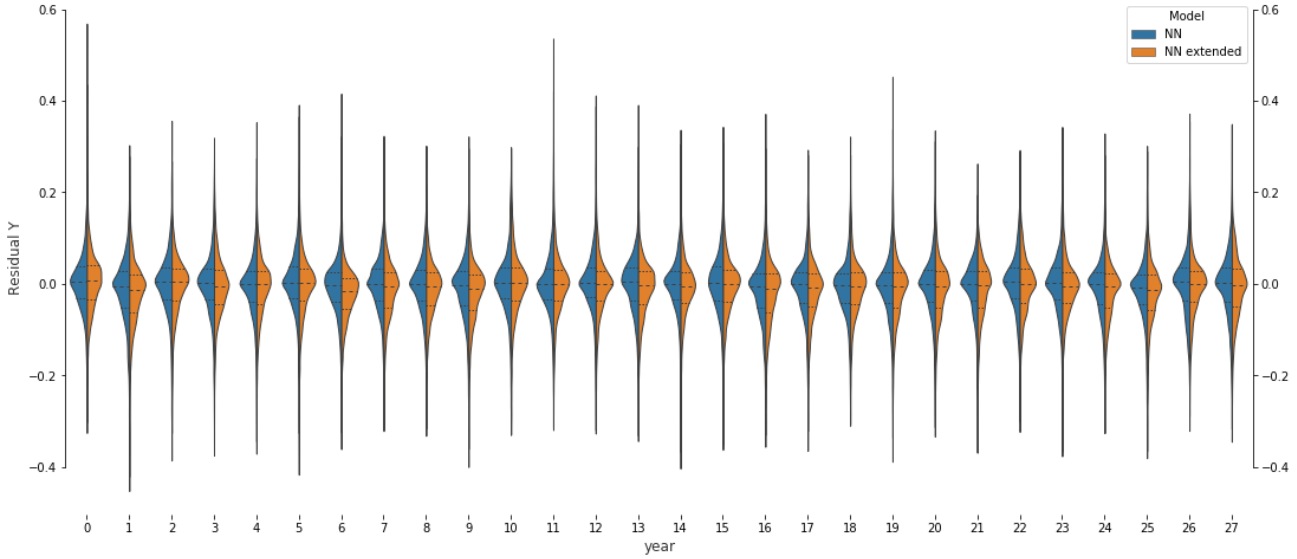


Figure C.24: R^2 for different climate variable levels



(a) The kernel density estimation (KDE) of the underlying residual distribution at every location, marked with the climate class they belong to.



(b) The kernel density estimation (KDE) of the underlying distribution spring wheat residuals for every year between 1982 and 2009 (where 1982 is denoted with 0 in the plot, 1983 is denoted with 1, and so on).

Figure C.25: Violin comparison plots of spring wheat residuals (in test set), comparing the KDE residual distributions grouped by location and year respectively, for the presented model with that of a similar extended neural network with an extra dense layer in the CTWN-branch.

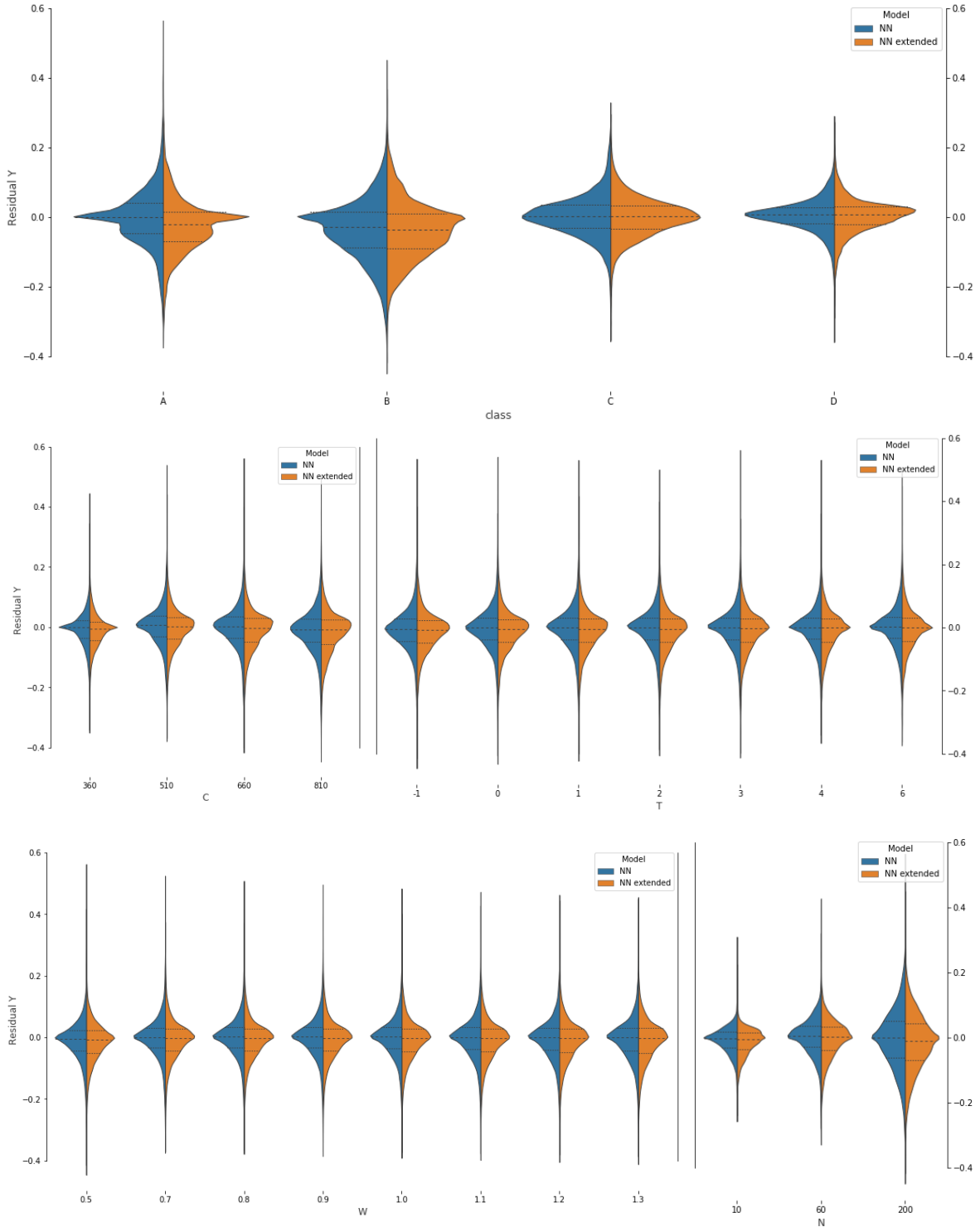


Figure C.27: Violin comparison plots of spring wheat residuals for presented vs. extended network [a].

[a]The KDE residual distribution for the presented model is compared with a similar extended neural network with an extra dense layer in the CTWN-branch. The upper violin plot compares the two models by displaying their climate class networks separately. The four plots below compares all the residuals of the respective model, the at every factor level of the climate variables C, T, W and N, respectively, for the presented model with a similar extended neural network with an extra dense layer in the CTWN-branch.

References

Abadi, Martín et al. *TensorFlow: Huber Loss*. https://www.tensorflow.org/api_docs/python/tf/losses/huber_loss. [Online; accessed 12-Sept-2019].

— (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. url: \url{https://www.tensorflow.org/}.

AgMIP. *GGCMI*. https://agmip.org/aggrid-ggcmi/. [Online; accessed 10-Sept-2019].

Arnfield, J. A. (2020). *Köppen climate classification*. https://www.britannica.com/science/Koppen-climate-classification. [Online; accessed 20-May-2020].

Balduin, S. (2018). "Surrogate models for composed simulation models in energy systems". In: 1.30. url: https://doi.org/10.1186/s42162-018-0053-z.

Barron, J. T. (2017). "A General and Adaptive Robust Loss Function". In: *CVPR*. arXiv: 1701.03077. url: http://arxiv.org/abs/1701.03077.

— (2019). *"A General and Adaptive Robust Loss Function" Jonathan T. Barron, CVPR 2019*. Youtube. url: https://youtu.be/BmNKbnF69eY.

Baxter, J. (2011). "A Model of Inductive Bias Learning". In: *CoRR* abs/1106.0245. arXiv: 1106.0245. url: http://arxiv.org/abs/1106.0245.

Biney, K. (2018). *Sentiment Analysis using 1D Convolutional Neural Networks in Keras*. https://medium.com/@romannempyre/sentiment-analysis-using-1d-convolutional-neural-networks-part-1-f8b6316489a2. [Online; accessed 15-May-2019].

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press.

— (2006). *Pattern Recognition And Machine Learning*. Springer.

Box, G. and N. Draper (1987). *Empirical Model-Building and Response Surfaces*. Wiley.

Brochu E., Cora V. M. and N Freitas (2010). "A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning". In: *CoRR* abs/1012.2599. arXiv: 1012.2599. url: http://arxiv.org/abs/1012.2599.

Chattopadhyay A., Hassanzadeh P. and S. Pasha (2018). "A test case for application of convolutional neural networks to spatio-temporal climate data: Re-identifying clustered weather patterns". In: doi: 10.31223/osf.io/q3ayz.

Chen Z., Badrinarayanan V. Lee C. and A Rabinovich (2017). *GradNorm: Gradient Normalization for Adaptive Loss Balancing in Deep Multitask Networks*. arXiv: 1711.02257 [cs.CV].

Chollet, F. (2018). *Deep Learning with Python*. Manning Publications Co.

Chollet, F. et al. *(2015) Keras*. https://keras.io.

— *Keras*. https://github.com/fchollet/keras.

— *Keras, Activation Layers*. https://keras.io/api/layers/activation_layers/. [Online; accessed 20-May-2020].

— *Keras, Core Layers*. https://keras.io/layers/core/. [Online; accessed 15-May-2019].

— *Keras, Custom Losses*. https://keras.io/api/losses/#creating-custom-losses. [Online; accessed 20-May-2020].

— *Keras: SpatialDropout1D layer*. https://keras.io/api/layers/regularization_layers/spatial_dropout1d/. [Online; accessed 20-May-2020].

Cipolla R., Gal Y. and A. Kendall (2018). "Multi-task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. doi: 10.1109/cvpr.2018.00781. url: http://dx.doi.org/10.1109/CVPR.2018.00781.

Elliott, J. et al. (2015). "The Global Gridded Crop Model Intercomparison: data and modeling protocols for Phase 1". In: *Geoscientific Model Development* (8), pp. 261–277. doi: https://doi.org/10.5194/gmd-8-261-2015.

Franke, J. et al. (2019). "The GGCMI Phase II experiment: global gridded crop model simulations under uniform changes in CO_2 temperature, water, and nitrogen levels (protocol version 1.0)". In: *Geoscientific Model Development Discussions*, pp. 1–30. doi: 10.5194/gmd-2019-237. url: https://www.geosci-model-dev-discuss.net/gmd-2019-237/.

Glorot X., Bordes A. and Y. Bengio (2011). "Deep Sparse Rectifier Neural Networks". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, pp. 315–323. url: http://proceedings.mlr.press/v15/glorot11a.html.

Glorot, X. and Y. Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, pp. 249–256. url: http://proceedings.mlr.press/v9/glorot10a.html.

Goodfellow I., Bengio Y. Courville A. (2016). *Deep Learning*. MIT Press. url: http://www.deeplearningbook.org.

Guo, M. et al. (2018). "Dynamic Task Prioritization for Multitask Learning". In: *The European Conference on Computer Vision (ECCV)*.

Hastie T., Tibshirani-R. Friedman J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction*. New York: Springer. doi: https://doi.org/10.1007/978-0-387-84858-7.

He K., Zhang-X. Ren S. and J. Sun (2015). "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *CoRR* abs/1502.01852. arXiv: 1502.01852. url: http://arxiv.org/abs/1502.01852.

Heaton, J. (2015). *Artificial Intelligence for Humans, vol 3: Deep Learning and Neural Networks*. Heaton Research, Inc.

Kingma, Diederik P. and Jimmy Ba (2014). *Adam: A Method for Stochastic Optimization*. arXiv: 1412.6980 [cs.LG].

Klein B., Wolf-B. and Y. Afek (2015). "A Dynamic Convolutional Layer for short rangeweather prediction". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. doi: https://doi.org/10.1109/CVPR.2015.7299117.

Kottek M., Grieser-J. Beck C. Rudolf B. and F. Rubel (2006). "World Map of the Köppen-Geiger climate classification updated". In: *Meteorologische Zeitschrift* 15 (3), pp. 259–263. doi: https://dx.doi.org/10.1127/0941-2948/2006/0130.

Kriegel, H. P., E. Schubert, and A. Zimek (Aug. 2017). "The (Black) Art of Runtime Evaluation: Are We Comparing Algorithms or Implementations?" In: *Knowl. Inf. Syst.* 52.2, 341–378. issn: 0219-1377. doi: 10.1007/s10115-016-1004-2. url: https://doi.org/10.1007/s10115-016-1004-2.

Larraondo P. R, Inza I. and J. A. Lozano (2017). "Automating weather forecasts based on convolutional networks". In: *Proceedings of the ICML 17 Workshop on Deep Structured Prediction* 70.

Maas, A. L. (2013). "Rectifier Nonlinearities Improve Neural Network Acoustic Models". In: *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*.

McDermid, S.P. et al. (2015). *The AgMIP Coordinated Climate-Crop Modeling Project (C3MP): Methods and Protocols, in Handbook of Climate Change and Agroecosystems: The Agricultural Model Intercomparison and Improvement Project (AgMIP) Integrated Crop and Economic Assessments — Joint Publication with the American Society of America, and Soil Science Society of America, edited by Hillel, D. and Rosenzweig, C.* London: Imperial College Press. doi: 10.1007/b98890.

Müller C. Elliott, J. Ruane A. Jägermeyr J. Balkovic J. Ciais P. Dury M. Falloon P. et al. (2019). "Understanding Crop Model Response Types in a Global Gridded Crop Model Ensemble". In: *Agriculture and Climate Change*, pp. 24–26. url: http://pure.iiasa.ac.at/id/eprint/15819/1/190324_understanding_response_types_AGRI2019.pdf.

Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. Determination Press. url: http://neuralnetworksanddeeplearning.com.

Nilsson, A. (2019). *Convolutional Neural Networks for DGVMs*. Mathematical Statistics Centre for Mathematical Sciences.

Olsson, O. (2017). *Emulators for dynamic vegetation models*. Mathematical Statistics Centre for Mathematical Sciences.

Petty, M. and E. Weisel (2019). "Chapter 4 - Model Composition and Reuse". In: *Model Engineering for Simulation*, pp. 57–85. doi: https://doi.org/10.1016/B978-0-12-813543-3.00004-4..

Rawlings J.O., Pentula S.G. and D.A Dickey (1998). *Applied regression analysis: a research tool, 2nd ed. in Springer Texts in Statistics*. New York: Springer-Verlag. doi: 10.1007/b98890.

Reed, Russell D. and Robert J. Marks (1999). *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. MIT Press.

Rosebrock, A. (2019). *Keras: Multiple Inputs and Mixed Data*. https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/. [Online; accessed 15-May-2020].

Ruder, S. (2017). "An Overview of Multi-Task Learning in Deep Neural Networks". In: *CoRR* abs/1706.05098. arXiv: 1706.05098. url: http://arxiv.org/abs/1706.05098.

Satorre E.H., Slafer-G.A. (1999). *Wheat: Ecology and Physiology of Yield Determination*. CRC Press.

Schmidhuber, J. (2014). "Deep Learning in Neural Networks: An Overview". In: *CoRR* abs/1404.7828. arXiv: 1404.7828. url: http://arxiv.org/abs/1404.7828.

Schmit, C.J. and J. R. Pritchard (2018). "Emulation of reionization simulations for Bayesian inference of astrophysics parameters using neural networks". In: *Monthly Notices of the Royal Astronomical Society* 475 (1), 1213–1223. doi: https://doi.org/10.1093/mnras/stx3292.

Smith B., Prentice-I.C. Sykes M.T. (2001). "Representation of vegetation dynamics in modelling of terrestrial ecosystems: comparing two contrasting approaches within European climate space". In: *Global Ecology Biogeography* 10 (1), pp. 621–637.

Talos, Autonomio (2019). *[Computer software]*. http://github.com/autonomio/talos. [Online; accessed 15-May-2019].

Tangirala, A.K. (2015). *Principles of System Identification - Theory and Practice*. Boca Raton: CRC Press.

Tukey, J. W. (1977). *Exploratory data analysis*. Reading, Mass, Addison-Wesley Pub. Co.

Underwood, R. (2018). *Bioprocesses in Food Industry*. Scientific e-Resources.

Wang Y., Ramanan-D. and M. Hebert (2017). "Learning to Model the Tail". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS '17)*, 7032 – 7042.

Waszczyszyn, Z. (1999). *Fundamentals of Artificial Neural Networks, 2nd ed. in Neural Networks in the Analysis and Design of Structures*.

"What Are the 5 Koppen Climate Classification Types?" (2020). [Online; accessed 20-May-2020]. Earth How. url: https://earthhow.com/koppen-climate-classification/.

Zhou, Y., Q. Hu, and Y. Wang (2018). "Deep super-class learning for long-tail distributed image classification". In: *Pattern Recognit.* 80, pp. 118–128.

Zou H., Hastie T. (2005). "Regularization and Variable Selection via the Elastic Net". In: *Journal of the Royal Statistical Society. Series B (Statistical Methodology)* 67 (2), pp. 301–320.