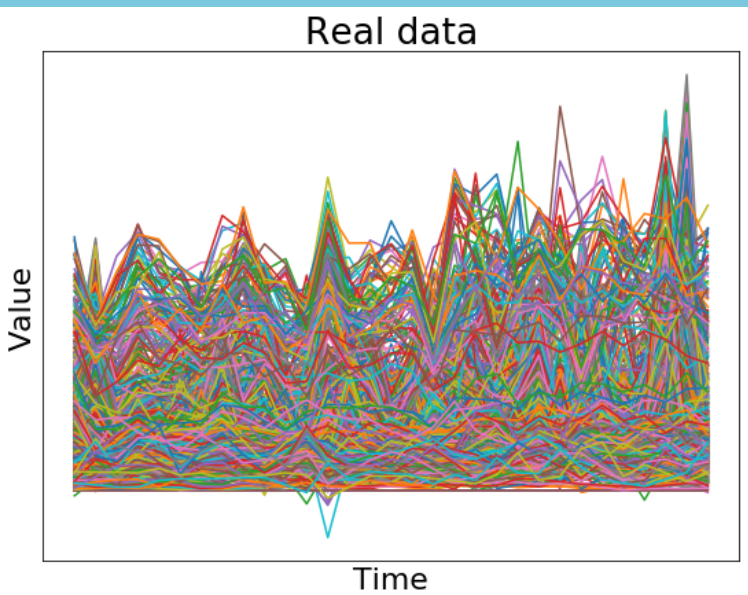
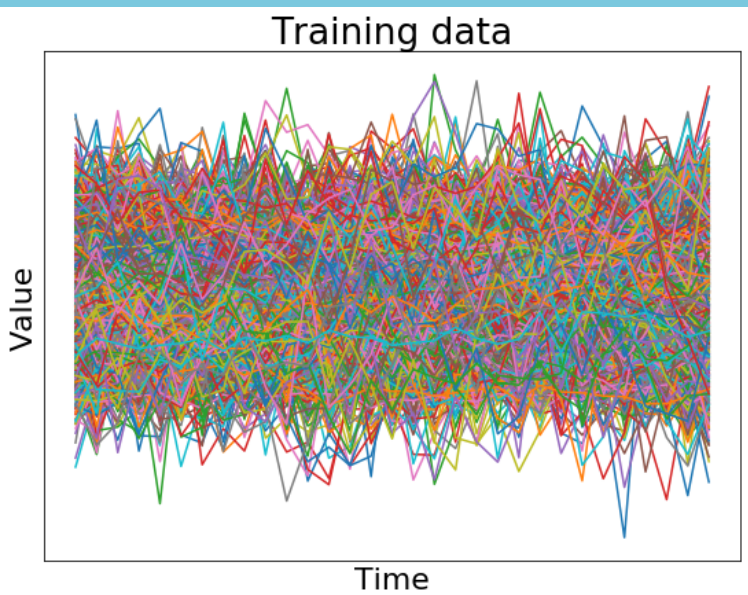
Breakpoint detection through neural nets

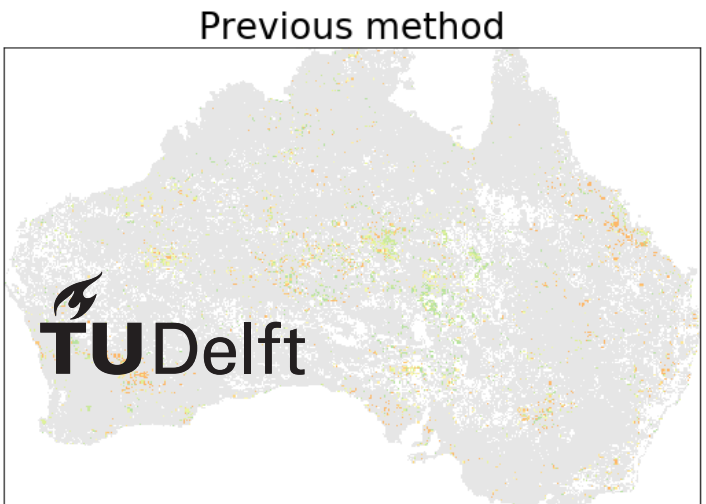
A feasibility study

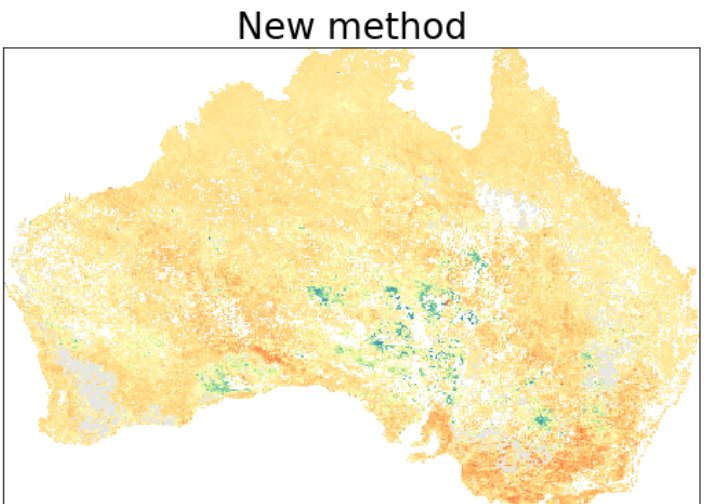
Master Thesis
Fokke Johannes Dijkstra

Supervisors: Dr. S.L.M Lhermitte (GRS, CitG)
Dr. I.V. Smal (GRS, CitG)
Prof. Dr. ir. S. Steele-Dunne (WM, CitG)









Preface

Dear reader,

At the moment, you are looking at my master's thesis. It is part of my track 'Geoscience and Remote sensing' from the Delft University of Technology. Initially wanting to become a doctor, with a bachelor's degree in mining engineering to my name, I never would have thought to finish my master's degree in remote sensing on a subject closely related to computer sciences.

Personally motivated to further the analysis of movement of deserts, this thesis started with the intention to assess desertification processes. Interested in the developing field of deep learning, I sought to combine 'deserts' with 'deep learning'. During my literature study, a problem in a contemporary analysis tool seemed present. It became the main subject of this thesis. Hopefully, peers of mine can draw inspiration from this work as help for their own work, or to find their own thesis subject of interest.

This research shows how easily good ánd bad algorithms can be produced. What struck me most, is how common assumptions can falter, while rules of thumb are confirmed time and again. By producing a 'black box' and showing it 'everything', some surprising performances could be achieved. These performances could not directly be translated to the real world, though. Looking back, the whole process was similar to my study path: it changed course over time.

I wish you, my dear reader, most pleasure reading through my thesis.

Fokke JohannesDijkstra

Delft, March 2020

"In science, you have to work hard,
but you also have to be lucky enough
to have the pieces fall into place
at the right time" - Stef Lhermitte

Abstract

A variety of statistical methods are available to detect sudden changes, or breakpoints, in time series when used as multi-temporal change detection technique. However, these methods are unreliable in the presence of noise. Neural nets might detect breakpoints better. These deep learning models are able to generalize and optimize well, even in the presence of noise. This research tests the feasibility of different neural net architectures to detect breakpoints in generic linear time series. Two relatively simple neural nets are proposed, combined with four different descriptions of breakpoint, and trained on synthetic data. The neural nets are tested on two datasets: On a separate synthetic dataset and on Australian rain-use-efficieny (RUE) time series, a surrogate for dryland ecosystem functioning. Some of the neural nets built performed exceptionally well on synthetic data, outperforming a benchmark statistical method with margin. The direct translation to RUE time series was less successful. The results shows great promise for the use of neural nets in change detection. A generalist change detection approach by use of neural nets is likely not optimal. Current developments in deep learning, as well as choosing the right user-case, show great promise to unlock the full potential of neural nets in time series analysis.

Key words: breakpoint detection, deep learning, neural nets, multi-temporal change detection

Contents

List of Figures

List of Tables

1 Introduction

1.1. Introduction to the problem

breakpoint *noun* 1: a convenient point at which to make a change, interruption etc. (Dictionary.com)

Time series analysis is increasingly important in a diverse range fields (Aminikhanghahi and Cook, 2017; Chandola et al., 2009). They are measurements over time of a process. Time series enable to find patterns over time, enabling to explain the past and predict the future. Processes change over time, sometimes abruptly. Such an abrupt change can wield many names: step detection, segmentation, edge detection, regime switching, structural breaks, break points, change point detection or detection of transition instants, though all have similar meaning (Aminikhanghahi and Cook, 2017). It is important to detect if and when an abrupt change happens. A sudden change in observed patterns can give valuable insights. One such moment where a sudden, unexpected change had great effect on decision maker policy, was the Nikkei stock exchange crash in 1989. In the years leading up to the crash, the Nikkei stock index behaved differently than after the crash, see figure 1.1. A breakpoint occurred. The immediate aftermath of this breakpoint included banks and investors changing strategies and governments to act (Bayoumi, 2001). Instead of leading policy, breakpoints can be used to determine quality of implemented policy. An example is wildlife conservation. Wildlife population trends are key indicators of wildlife conservation quality (Kiffner et al., 2020). Observing sudden changes in these trends can thus bring valuable insights in their functioning (Ogutu et al., 2011). A sudden drop in population of an endangered species, a breakpoint in the population trend, can indicate serious problems. Honey bees are such a species. Next to inspiring policy change or providing information on a monitored processes, breakpoints can help explain observed novelties. As of this writing, historic forest fires are devastating Australia. While there are forest fires on a yearly bases, this time the forests possibly won't recover (Nolan et al., 2020). Might there have been a breakpoint in ecosystem functioning, facilitating such fires? By observing changing patterns in temperature, precipitation or parameters, such a question might be answered. To find the cause for breakpoints, first they have to be detected. It is thus useful to know when breakpoints are present in time series, when changes in systems occur.
Breakpoints in time series can be present in various ways. Assuming a linear process, breakpoints can be present mainly in three ways, in change of: (Rohrbeck, 2013)

- Mean value
- Variability
- Regression structure

These are visualized in figure 1.2. Note that any combination of the above can also occur. A multitude of breakpoints can also appear in a single time series. Changes in the fundamental function functioning are not taken into account in these descriptions of breakpoints. Where first a linear process is observed, after a breakpoint another function description might be better to describe the behavior of a process.

A great number of generic change point detection algorithms are developed (Aminikhanghahi and Cook, 2017). They answer the need to monitor abrupt change in a consistent manner on vast amounts of data.

Figure 1.1: Nikkei stock index 1970-2005 (Colombo, 2012). After the buildup of a housing bubble in the end of 1989, the stock market collapsed. Before and after this breakpoint a different pattern is observed.

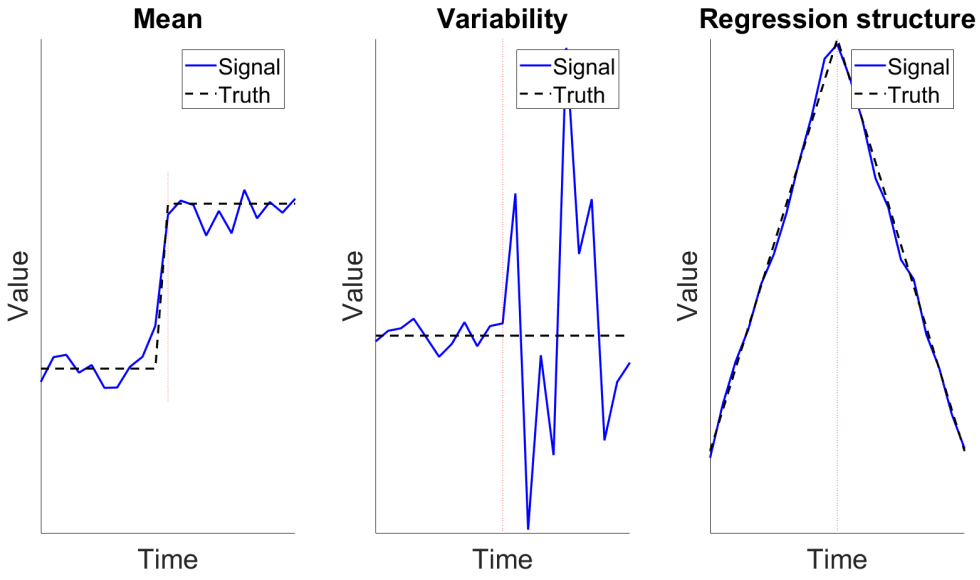


Figure 1.2: Three breakpoint types of a linearly behaving process. Above each graph the changing feature is noted. All cases show a permanent change in time series behavior and can occur in combination with one another.

Typically these algorithms assume that observations $\{y_t : t \in 1,...n\}$ have distribution F_0 for $i \leq \tau$ and a possible different distribution F_1 for $i > \tau$ (Chen et al., 2015). The parameter τ is some threshold value, used to test a hypothesis. In other words, statistical methods typically test between the null hypothesis H_0 and the alternative hypothesis H_A:

$$\delta = \begin{cases} H_A, & \text{if} > \tau \\ H_0, & \text{otherwise} \end{cases} \tag{1.1}$$

With δ as the decision rule. The null hypothesis states that no change has occurred, where the alternative states a change did occur. All of the aforementioned ways a breakpoint can occur, can, but not necessarily are, tested for using this hypothesis. For example, a sudden change in mean (μ) or variance (σ^2) would have the criteria:

$$\mu_t = \begin{cases} \mu_n, & t > \tau \\ \mu_1, & \text{otherwise} \end{cases} \quad \mu_1 \neq \mu_n$$

or

$$\sigma_t^2 = \begin{cases} \sigma_n^2, & t > \tau \\ \sigma_1^2, & \text{otherwise} \end{cases} \quad \sigma_1^2 \neq \sigma_n^2$$

These examples are in layman's terms: a change in mean larger than a certain threshold and a change in variance larger than a certain threshold. Generally, change detection algorithms are classified as either *online* or 'real-time,' or as *offline* or 'retrospective' (Aminikhanghahi and Cook, 2017; Liu et al., 2013). Online algorithms run concurrently with the process they monitor, with as goal to detect change as soon as possible (Downey, 2008). Early warning systems are online change detection algorithms which are commonplace in the financial world (Berg et al. 2005) and natural disaster management (Basher 2006; Zschau and Küppers 2013). Well known early warning systems are in place to protect against tsunamis and river floods, which rely on the sole time series of a seismograph and precipitation rates respectively.

Offline algorithms consider the whole dataset and look back to recognize change. Aim is often to identify a sequence breakpoints. Offline change analysis can be used to improve accuracy of forecasting models (Downey, 2008). Various types of change detection algorithms are available, some non-parametric (Carlstein et al. 1994), some under specific assumptions (James et al., 1987). Aminikhanghahi and Cook, 2017 also classify all algorithms as either *supervised* or *unsupervised* algorithms.

Supervised algorithms are machine learning (ML) algorithms that learn a mapping from input data to a target attribute of the data. This usually means that ML algorithms are trained on class labels, either as binary or multi-class classifiers (Cook and Krishnan, 2015). Each segment in between change points can for example be labeled a different class, or the change points themselves might be indicated. If using an interpretable model, information can be given on the nature of detected change points. The dominant factor which caused a model to detect change could then presumably be deduced. Drawback is the need for sufficient amount of training data, with the right diversity to represent all possible scenarios. Example supervised algorithms include, but are not limited to: decision trees, nearest neighbor, hidden Markov models, Gaussian mixture models and Naive Bayes.

Unsupervised learning algorithms segment time series data, based on statistical features of the data (Aminikhanghahi and Cook, 2017). An advantage of such algorithms is that they handle a multitude of situations without prior training. Unsupervised algorithms often check time series in iterative fashion, checking at each possible interval whether or not a test fulfills the null hypothesis. This hypothesis test can be likelihood-based, probability distribution-, kernel-based, graph- or state space-based. Bayesian change detection is an example of a probabilistic unsupervised change detection method. Change detection algorithms either monitor change directly or detect change in the past. In all cases, they should be able to scale to large data sizes (Aminikhanghahi and Cook, 2017; Chandola and Vatsavai, 2011).

A field of study for which this is especially true, is remote sensing (RS). RS deals with imaging the Earth's surface with satellites. The nature and utilization RS related products differs, as each individual satellite has different orbits and/or measurement equipment. This results in a massive, continuous data stream from orbit to Earth. From this data, valuable information can be extracted through change detection. Detection of changes over time in RS is used to aid better decision making in the fields of agriculture (Khabbazan et al., 2019; Mulla, 2013; Steele-Dunne et al., 2017), sea level monitoring (Rose et al., 2019; van der Burgt et al., 2016; Vermeer and Rahmstorf, 2009), ice cap melt (Hall, 2012; Spreen et al., 2008; Van Angelen et al., 2012), forestry (Darmawan and Sofan, 2012; Housman et al., 2018; Masek and Collatz, 2006; Rödig et al., 2017), dryland conservation (Burrell et al., 2017; de Jong et al., 2012; Negri Bernardino et al., 2018; Washington-Allen et al., 2008) and more. To extract the desired information from these large data streams, many change detecting

algorithms have been developed specifically for remotely sensed data (Coppin et al., 2004; Hussain et al., 2013; Lu et al., 2004).

These approaches can be either pixel or object based, derived by use of either statistical methods or machine learning approaches. Breakpoint detection algorithms in RS detect changes in the time series of pixels, thus can be considered pixel based change detection methods. Many methods are *bi-temporal*, which means they focus on the use of 2-5 images only (Coppin et al., 2004; Lu et al., 2004). *Multi-temporal* change detection methods are limited in number. As multi-temporal change detection technique, time series analyses are commonly used. Some time series analysis used in RS applications are: principal component analysis (PCA, Crist and Cicone, 1984), wavelet decomposition (Anyamba and Eastman, 1996), Fourier analysis (Azzali and Menenti, 2000), Chance Vector Analysis (CVA, Lambin and Strahlers, 1994) and the Breaks For Additive Seasonal and Trend (`bfast`) model (de Jong et al., 2012; Verbesselt et al., 2010). `bfast` is created in response to two shortcomings in the other methods, because 1) the other methods show difficulty in labeling the change components and because 2) they depend on specific thresholds or change directories. Verbesselt et al., 2010 also mention the shortcomings of the other methods in their ability to deal with seasonal patterns and state the need "for an unsupervised, more generic, change detection approach independent of the data type and change trajectory."

`bfast` fits a piecewise linear and trend model to the data, after which a statistical test is used for the null hypothesis. Out of this test, the amount of change is quantified and two new piecewise models are fitted. This is iterated until no more breaks are detected. `bfast` is available as a package for the programming language R and is mainly used currently in tropical rainforest change (Darmawan and Sofan, 2012) and detecting dryland degradation (Burrell et al., 2017; de Jong et al., 2012). The algorithm decomposes the signal in seasonal, linear and noise components. It fits a general model of the form:

$$y_t = \alpha_1 + \alpha_2 t + \sum_{j=1}^{k} \gamma_j sin(\frac{2\pi j t}{f} + \delta_j) + \epsilon_t \tag{1.2}$$

For linear time series, without seasonal component, this simplifies to:

$$y_t = \alpha_1 + \alpha_2 t + \epsilon_t \tag{1.3}$$

The `bfast` function decomposes the signal (y) into linear trend (α), seasonal amplitude (γ), seasonal phase (δ) and remainder components (ϵ), for a yearly measurement frequency (f) (De Jong et al., 2013; Verbesselt et al., 2010). j in equation 1.2 indicates the corresponding segment of the time series. Note that changes in α_1 and α_2 are direct measures of the change in mean and regression structure, respectively, as depicted in figure 1.2. There is the option to limit the function to remove either the trend or seasonal components, which is mathematically presented in equation 1.3. A special case of `bfast`, `bfast01`, is used to detect at most a single breakpoint. In its original presentation, the methodology is validated both with simulated time series and real NDVI time series. NDVI, or Normalized Difference Vegetation Index, is the normalized difference of two spectral bands (Tucker, 1978). The spectral bands are the red and infrared spectral bands, collected by various satellites. The spectral signatures of vegetation and bare ground are very different in these bands, which causes NDVI to be commonly used in ecosystem monitoring. The simulated NDVI time series to test `bfast` on are made by generating time series following the model represented by equation 1.2. figure 1.3 shows how three breakpoints in a simulated time series are detected. All parameters to generate the time series in figure 1.3, are estimated from known typical values in NDVI time series.

Not tested in the original presentation of `bfast` is the sensitivity of `bfast` to noise. All newly acquired datasets contain unknown levels of noise. Figure 1.4 shows two time series with identical signal, but different degrees of noise. By eye, the breakpoint in the top image is easily observed, while from the data in the bottom image no real judgement on breakpoint behavior can be made. When the range of these values is increased by the presence of noise, a null hypothesis becomes less reliable. This is the case for `bfast`, but this theoretically also poses a problem for other statistical change detection systems. Few naturally occurring processes act linearly and observations are (almost) never deterministic, meaning that effects such as instrument errors, detection thresholds and resolution need to be captured by the uncertainty of the observed value. This raises the question: **How can breakpoints in time series reliably be detected in large, noisy datasets?**

There are two boundary conditions to be met in this question. First is the notion of large and noisy datasets. Any breakpoint detection method should be able to handle real-world datasets. Second is the notion of reliability. This is something subjective, as for every problem statement a different reliability can be

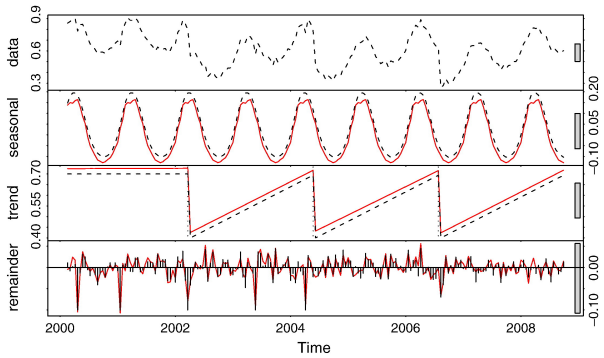


Figure 1.3: `bfast` applied to a simulated NDVI-time series (Verbesselt et al., 2010). The dotted line indicates simulated values, red lines indicates the fit made with `bfast`. From top to bottom: the original signal, the seasonal-, the trend- and the remainder components. In this case, `bfast` found the three breakpoints correctly. The breakpoints in this figure are characterized as sudden changes in mean, with also a change in regression structure for the first breakpoint. This is visible in the trend component. The y-axis changes in scale for every decomposition component.

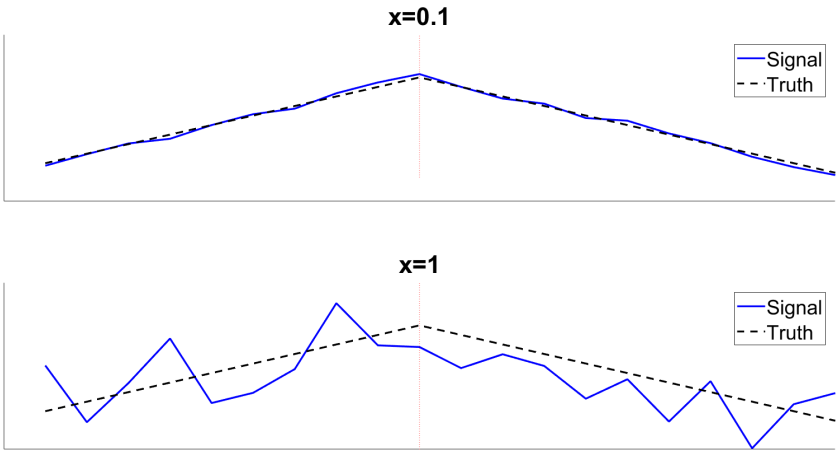


Figure 1.4: Two example time series with different degrees of noise. Noise of the form $N(\mu = 0, \sigma = x * R)$ is added, where R refers to the range of the time series and x is mentioned above each graph. Both time series have an identical base signal with breakpoint in the middle, but the above graph has contains less noise. In the bottom graph, the change point cannot reliably be observed visually.

required. Given these boundary conditions, a new ML approach might lead to something useful. As mentioned earlier, ML approaches have been developed for breakpoint detection. However, no deep learning approach has been proposed yet.

LeCun et al., 2015 state that: "Deep-learning methods are representation-learning methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level." Highly non-linear functions can be approximated by these representations. These methods work notoriously well, even when trained on noisy datasets (Lee et al., 2016; Rolnick et al., 2017; Sukhbaatar and Fergus, 2014). Deep learning is a catalyst in breakthroughs in the fields of speech recognition (Deng et al., 2013; Wu et al., 2016), vision recognition (Dong et al., 2016; Rastegari et al., 2016), drug discovery (Ramsundar et al., 2015; Wallach et al., 2015), particle tracking (Chenouard et al., 2014; Meijering et al., 2012) and more (LeCun et al., 2015). The ML method can be applied in both a supervised and unsupervised manners, though supervised methods are at the moment more widely used (LeCun et al., 2015). The application of deep learning is done through neural networks or neural nets.

A major advantage of neural nets is their versatility. They can be applied to a multitude of problems. The networks can be tweaked and adjusted to optimize for ones' wishes. Statistical methods can only tackle the problem they are designed to solve. While this also is true for a deep learning approach, the neural nets presented in this thesis are easily adjusted to incorporate more data. This can be spatial relations in the case of RS data, or a multitude of data sources.

A major drawback of neural nets is their 'black box' nature. The inner workings of neural nets are often unknown or uninterpretable. Some call this the 'Achilles heel' of deep learning (Zhang and Zhu, 2018). It is semantically useful to understand how a neural net functions (Bau et al., 2017; Zhang and Zhu, 2018; Zhang et al., 2018b). In other words, it is useful to know why a model does what it does. Another drawback of machine learning approaches is the requirement of lots of training data to work properly. There are limited annotated databases for breakpoints detection, because the creation of such a dataset requires enormous expertise in that field of study (Aminikhanghahi and Cook, 2017).

In many fields of study, the 'best guess' of the functioning of a process is often to assume linearity (Bonacich and Kirby, 1976; Koch et al., 2009). When considering possible breakpoints in time series of linearly behaving processes, they can occur in limited ways. The earlier mentioned change in mean, variability and regression structure, cover a great deal of possible scenarios. By training a ML approach on all possible scenarios a breakpoint can occur, it should theoretically be possible to apply such an approach on time series in general.

The use of synthetic training data is already shown successfully in other deep learning problems. Barbosa et al., 2018 trained face recognition software by training on computer generated faces. The idea in this case is to show all possible time series to the neural nets, so resulting networks can be generally applied in breakpoint detection in time series with assumed linearity. The synthetic data is made with a few assumptions in mind. First and foremost, is the assumption that the signal function is linear. Second is the assumption that only a single breakpoint occurs in a time series. These keep the problem relatively simple. Stated earlier is the rare presence of a linear process in real-world applications. The user-case on which to apply all models should thus be highly linear in nature. A third assumption, is the assumption that a breakpoint does not occur by change in variability. To investigate model performances, they are compared per noise level. The variability is considered as the noise in this research, hence it is not changed by a possible breakpoint. Only noise is considered in the form of $N(\mu = 0, \sigma = x * R)$, with range R and multiplication factor x. The synthetic time series will be randomly generated. Fourth assumption is the absence of gradual change. A last assumption is that all possible time series will be presented to the algorithms.

1.2. Research objective

This research investigates how supervised deep learning can best be applied to breakpoint detection. It is a feasibility study, where model performances will be expressed per noise level. This can give insights into their workings and reliability when presented a new dataset. The complexity of the used neural nets is kept to a minimum, which allows for a good understanding of the various working parts. The simplicity of models can also be indicative whether this application is even possible with these models. To check whether resulting models have a good performance, resulting deep learning approaches need to be working comparably or better than current breakpoint detection systems (Makridakis et al., 2018). This is why all neural net models are compared to `bfast01`, the version of `bfast` optimized to detect a single breakpoint in a time series. `bfast01` is a good choice of comparison. It should work optimally on the synthetic dataset, because the

function it fits is of the same form as the actual functions used to generate the data, see also equation 1.3. The performance of the breakpoint detection systems are compared by both looking at the accuracy in breakpoint detection and accuracy in breakpoint timing. All neural nets created and `bfast01` are compared both on synthetically created data and a user-case.

Rain-use-efficiency (RUE) is chosen as the user-case, which is a measure of dryland ecosystem functioning based on remotely sensed data (Fensholt and Rasmussen, 2011). Abrupt changes in RUE mean that the ecosystem inherently acts differently. Drylands are degrading more than they are improving, which can cause water shortages, food security issues and loss of biodiversity (UN 2010). Displacement of people can give a variety of other problems (Hillier and Dempsey, 2012). Early warning systems thus are necessary to provide local authorities ample time to respond in a proper manner (Hillier and Dempsey, 2012). A good working breakpoint detection system can thus add value in this context. RUE is by the ratio between Net Primary Production (NPP) and precipitation, see equation 1.4 (Fensholt and Rasmussen, 2011; LeHouerou, 1984; Negri Bernardino et al., 2018). RUE is useful in describing dryland ecosystem functioning, as it is a measure for the response of plants to precipitation (Negri Bernardino et al., 2018). RUE therefore normalizes for inter-annual rainfall variability. As a proxy for NPP, the sum of NDVI over the growing season is used.

$$RUE = \frac{NPP}{Precipitation} \approx \frac{\sum NDVI_{Growingseason}}{\sum Precipitation_{Growingseason}} \tag{1.4}$$

More specifically, RUE time series for Austalia are chosen to apply breakpoint detection methods on. These time series are chosen as user-case for three main reasons. Firstly, it is assumed that RUE changes monotonically over time, to be a linear process (De Jong et al., 2013; Negri Bernardino et al., 2018). This is mandatory to have a neural net based model, trained on synthetic data, be properly applied to RUE. Secondly, RUE is a RS-derived product. It gives a large data set to work with. Stated earlier is the need of breakpoint detection systems for large datasets, thus all developed algorithms can be applied on such a dataset. Thirdly, `bfast01` is already applied to this type of data in earlier research (Negri Bernardino et al., 2018). Dryland ecosystem functioning of Australia specifically is investigated in more ways (Burrell et al., 2017, 2018; Negri Bernardino et al., 2018, giving way to visually compare results of different researches.

1.3. Research questions

The research question posed is:

How can neural networks best be used in order to detect breakpoints in linear time series?

This question contains different elements. Time series are considered linear, which reduces the variety in breakpoint types possibly present. 'Best' indicates there is a fixed framework to compare different breakpoint detection algorithms in. This does not exist (Aminikhanghahi and Cook, 2017), thus such a framework will be formulated. To give a clear answer to this questions and subdivide the problem statement, the research will answer the following sub-questions:

1. How do different neural net architectures compare in the detection of breakpoints?
2. Can reliability of neural nets in breakpoint detection be quantified?
3. How do neural nets compare to `bfast01`?

To find what neural net architecture is best suited for the problem at hand, some fundamental neural net architectures are compared. If breakpoint detection is even feasible, it should definitely be possible if the simplest of models can do it.

The resulting models should be applicable to real-world user-cases, which require a certainty measure. For every architecture tested, a hypothesis is made and tested to figure out which parts tell something about the model's reliability.

Last, to put resulting model performances in context, it should be compared with a current statistical method. The compared method of choice is `bfast01`, for reasons outlined above.

1.4. Thesis outline

The fundamental knowledge behind the concepts of neural nets, the metrics coined to judge breakpoint detection and some staple multi-temporal change detection algorithms in RS are found in Chapter 2. The

methodology and conducted experiments are described in Chapter 3, followed by their results in Chapter 4. The user-case, its history, application and test results can be found in Chapter 5. The discussion in Chapter 6 focuses on the practical implications ensuing all results, puts them into context. This thesis finalizes with a conclusion and recommendations for further research in Chapter 7 and 8, respectively.

2

Background and important concepts

This chapter covers the necessary information to grasp what is coming later. Breakpoint detection is dealt with in this thesis, thus some qualitative parameters are coined in order to describe the quality of breakpoint detection. Deep learning is used to detect breakpoints. What actually is deep learning? This is also covered in this chapter: The terminology and workings the neural nets is briefly explained, to provide a basic understanding. Lastly, the mechanics behind some widely used breakpoint detection algorithms for remote sensing are explained, one of which will be the baseline to compare deep learning approaches with.

2.1. Breakpoint metrics

How breakpoints can occur in linear processes, is presented in Chapter 1. To assess the performances of breakpoint detection systems, a consistent framework to compare them in should be present. Accuracy, precision and signal of breakpoints should be taken into account. These terms are coined and described in this section.

2.1.1. Signal of breakpoints

There currently is no description of the signal of breakpoints. It is useful to define this signal, because not all breakpoints are detected equally easily. A breakpoint detection algorithm needs to distinguish at least the more clear occurrences of breakpoints. A strong breakpoint signal should indicate that a breakpoint is easily observed. A breakpoint in a time series is more easily observed when it occurs in the middle of said time series. On the other hand, a breakpoint close to the start or end of a time series is harder to distinguish. Similarly, a breakpoint is more easily detected if the change is significant. Only linear signals are used in this thesis. There are thus two things important in observing change easier: the nature of the change and the time at which change occurs. Combining this into a definition, the signal of a breakpoint is hereby defined as: *the minimal detectable change between the current situation and a situation where no breakpoint had occurred.* This signal is depicted in figure 2.1 and mathematically described in equation 2.1.

$$Signal_{\text{Relative}} = min(|\Delta x| * T_{bp-edge_{\text{start}}}, |\Delta x| * T_{bp-edge_{\text{end}}}) + B \tag{2.1}$$

With Δx as the change in slope from before to after the breakpoint and B the abrupt change at a breakpoint. $T_{bp-edge_{\text{start}}}$ is the time from the breakpoint to the start of the sequence, $T_{bp-edge_{\text{end}}}$ is the time between breakpoint and the end of the sequence. Note that the signal of a breakpoint differs from the signal of the underlying process. In figure 2.1 an arbitrary signal is plotted with high noise, with a breakpoint in the middle, in the x-direction. The underlying linear process cannot be clearly distinguished by eye, the data shows high noise. The breakpoint, however, is somewhere in the middle of the graph the change in regression structure can be envisioned visually. In cases there is no breakpoint, there cannot be signal of a breakpoint. Noise of a breakpoint is similarly undefined. Henceforth, no signal to noise ratio in breakpoint detection can be defined. However, the relation between breakpoint signal, observed noise and breakpoint detection accuracy and precision can be investigated. Such a relation is good to investigate, because from the original data noise estimates can be determined, which in turn estimates the expected model performance.

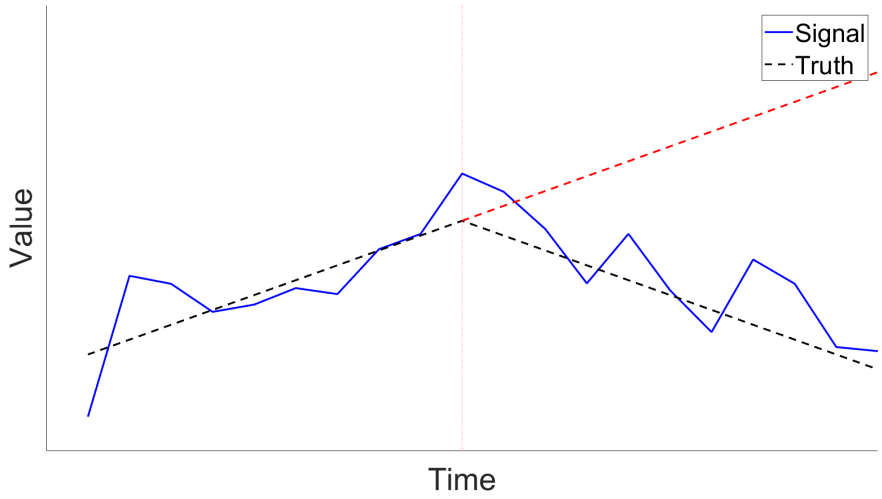


Figure 2.1: A randomly generated signal with high noise and observable breakpoint in the middle. The breakpoint can still be observed because it has a good breakpoint signal. The red line indicates what would have happened if no breakpoint occurred. The difference between the red line and the truth is considered the breakpoint signal

Table 2.1: Confusion matrix. Small and capital p and n reflect the positive and negative case, respectively. Positive in the context of breakpoint detection means a breakpoint is present, negative means no breakpoint is present.

		Predicted outcome		
		p	**n**	**total**
Actual value	**p′**	True Positive	False Negative	P′
	n′	False Positive	True Negative	N′
	total	P	N	

2.1.2. Accuracy in breakpoint detection

The performance measures commonly used to judge change detection algorithms are accuracy, false positive ratio and false negative ratio (Aminikhanghahi and Cook 2017; Radke et al. 2005). Table 2.1 shows a confusion matrix, which aids to explain these terms. False positive and negative ratio are the ratio of time series erroneously labeled as having a breakpoint and having no breakpoint, respectively. Accuracy is derived by the sum of the ratio of true positives and negatives, see equation 2.2. Accuracy is chosen over other metrics often used to distinguish breakpoint detection performance, such as F1-scores or recall, because it gives equal weighting to detected positives and negatives.

$$Accuracy = \frac{True_{\text{Positives}} + True_{\text{Negatives}}}{Samples_{\text{total}}} \tag{2.2}$$

Accuracy is considered as the most important indicator of a well performing model. It is subjectively considered more useful to reliably know that a system behaves systematically different, than precisely knowing when a systemic change is initiated. In a different context this priority might be set different. False positive and false negative behavior can be used to find shortcomings in considered algorithms. If false positive ratio is zero, it means an algorithm can always be trusted when it predicts a breakpoint. If false negative ratio is zero, the opposite holds. These metrics tell nothing on whether a detected breakpoint is placed at the correct timing, though.

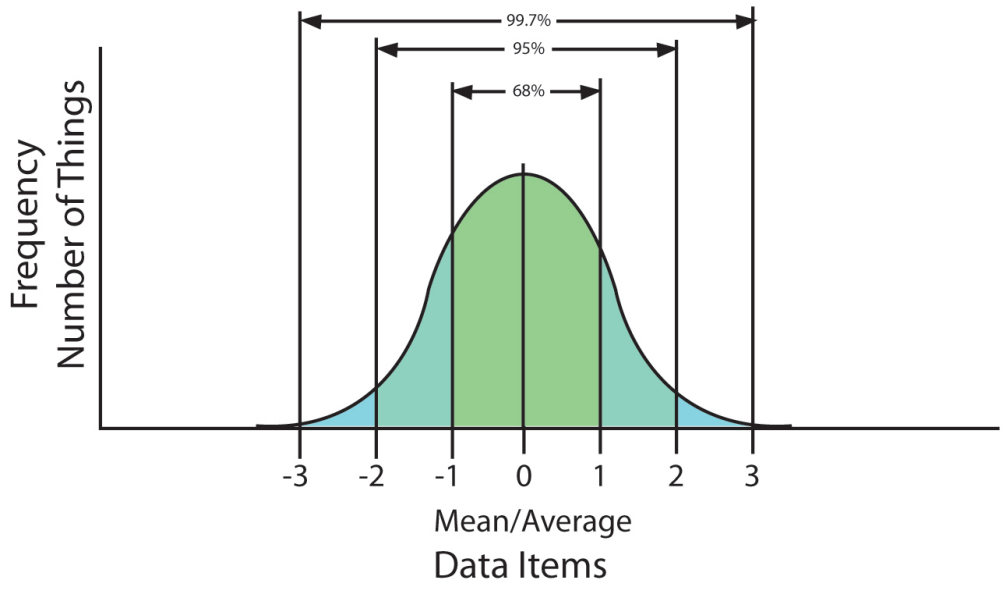


Figure 2.2: Certainty bounds from standard deviation visualized. The x-axis shows multiples of the standard deviation (σ_{std}), the percentages the chance that a value falls within that segment.

2.1.3. Precision

The second metric to distinguish is the quality of a detected breakpoint, the correctness of detected breakpoint time. The precision is coined hereby as: *a measure with which detected breakpoint times differ from the true breakpoint time.* To express precision in a single number, it will be calculated as the standard deviation of the offset between true and detected breakpoint. A good precision is thus a low standard deviation from the truth. This is mathematically presented in equation 2.3, with standard deviation σ_{std}, true and predicted breakpoint times μ and N samples.

$$\sigma_{\text{std}} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(\mu_i^{\text{true}} - \mu_i^{\text{pred}})^2} \tag{2.3}$$

The precision of change detection algorithms is considered separately, because different change detection algorithms give different types of certainty bounds for found breakpoints. By having a generic, easy to derive quantity describing the offset between detected and true breakpoints, different algorithms and methods can be compared in a consistent manner. The use of standard deviation to express variation in a set values is common practice among statisticians (Bland and Altman, 1996; Leys et al., 2013). Other measures of precision could be forms of mean-square-error (Aminikhanghahi and Cook, 2017), but standard deviation is chosen in this research. Standard deviation then indicates the possible range of offsets which can be expected, if these offsets are normally distributed. This is visualized in figure 2.2 and means that any detected offsets from the true breakpoint should be tested for normality.

2.2. Deep learning

Deep learning algorithms consist of neural nets, how they work is explained in this section. First a very brief history of deep learning is given. When discussing neural nets there is a lot of terminology involved, which is still unsettled in the machine learning community (Lipton and Steinhardt 2018). Therefore, a short description of neural nets and terms used are explained. A lot of terms will be introduced, which will all be listed at the end of Section 2.2.2. What follows are clarifications on some critical concepts appearing in any neural net architecture, which are in turn needed to understand the reasoning behind and the functioning of the proposed models in Chapter 3. The section is ended with a description of the 2 main network architectures used in this research: The multilayer perceptron and the Long Short-Term Memory recurrent neural net. The third neural net architecture, the convolutional neural net, is not used, for reasons explained in appendix 10.1. If not specified, information in this section is taken from Goodfellow et al. 2016.

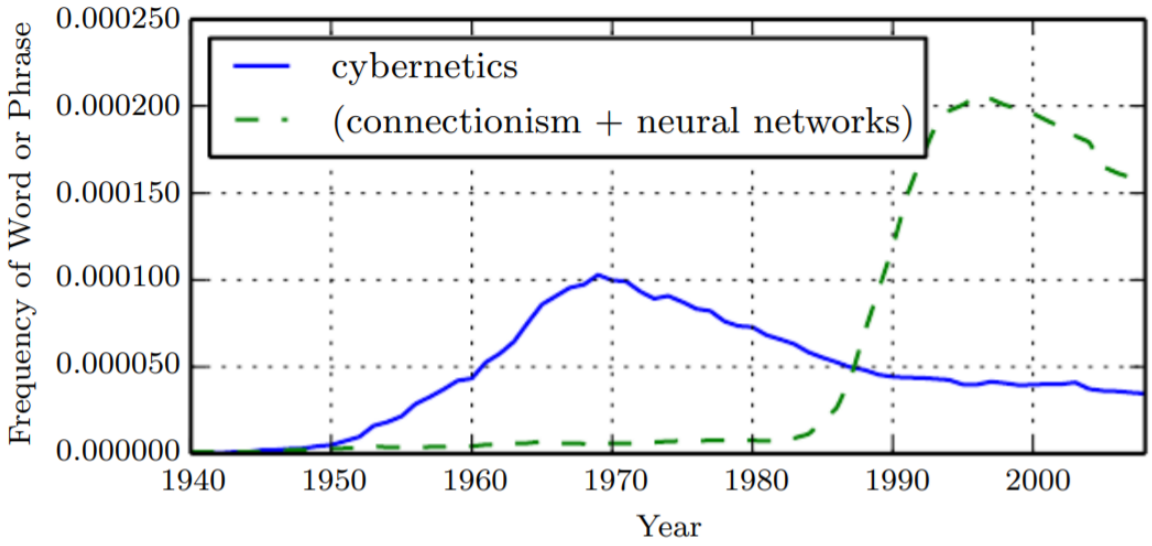


Figure 2.3: The historic waves of neural net research (Goodfellow et al. 2016). The first two waves in which neural network research was popularized are as of this writing behind us. The third wave is initiated around 2006, which is not represented in the figure. The number of phrases is measured from the frequency of the phrases 'cybernetics', 'connectionism' or 'neural networks', according to Google Books.

2.2.1. The history of deep learning

AI, neural nets and deep learning are popular terms in the media today. Deep learning is around for some time, though. It is conceptually around since the 1940's, but under different names. Be it under the name of *cybernetics*, *connectionism* or *artificial neural networks*, deep learning appeals to learning multiple levels of composition. This concept can also be applied to machine learning frameworks which are not necessarily neurally inspired. The evolution of deep learning can be explained in three waves, see also figure 2.3. Where the first wave (1940's) dealt with linear prediction problems, the second wave was sparked by the cognitive sciences (1980's). Cognitive science is an interdisciplinary approach to understanding the mind. The central idea back then was that a large number of simple computational units can achieve intelligent behavior when networked together. Many concepts which were coined during these first two waves are still around today. The third, current, wave is occurring since 2006. It is sparked by neural nets outperforming other machine learning methods, as well as various hand-designed functionalities. This third wave dawned with unsupervised learning techniques, though the current abundance of large datasets redirects interest to older, supervised learning algorithms (LeCun et al., 2015). In this thesis only supervised learning algorithms are considered.

In machine learning, supervised learning is the most common learning type (LeCun et al., 2015). It means that a machine learning model, be it a deep learning approach or not, is trained on a labelled dataset. Because the optimal output is known, the model error can be calculated, after which the trainable parameters of a machine learning model can be adjusted to minimize the calculated error. Supervised learning algorithms are limited in the things that they can learn. Take labeling of images as an example. First pictures of classes 'houses', 'cars' and 'people' are collected. An algorithm is designed to calculate the probability of an image being a house, car or man, with the highest probability giving a class prediction. The model makes mistakes, for which the error is computed, and the 'knobs' of the model are updated to limit the error. However, when an image of a 'cat' is presented, the model will still provide an output of either 'houses,' 'cars' or 'people.' It is thus important in supervised learning problems to present all possible scenarios during training, as not learned features cannot be extracted or processed reliably.

Unsupervised learning is somewhat inspired by observing learning in animals (Barlow, 1989; LeCun et al., 2015). The brain makes sense of its surroundings without any associated reward or punishments (Barlow, 1989), seen as key in the self-learning framework (Le, 2013). Unsupervised deep learning algorithms extract features from the input data by itself, without the need to label data. Le, 2013 show that such techniques can work to extract high-level features from unlabeled Youtube images, like human faces, the heads of cats and human bodies. Unsupervised learning methods do not see widespread use, but might get more common than supervised learning methods in the future (LeCun et al., 2015).

2.2.2. The concepts behind and terminology of neural nets

Neural nets or neural networks are machine learning algorithms, which try to approximate a function by a collection of smaller, simpler functions. They get their name from the possibility to schematically draw them like a network, loosely resembling the structure of neurons in nervous systems. What follows is a description of neural nets using all actual terminology, with a large term density to the lay person. This section is ended with an overview of all newly introduced terms in Table 2.2, which aids as reference in the subsequent sections.

The networks are comprised of a variety of layers, wherein the smaller functions are applied. The location of such a smaller function is called a *neuron* or *node*. Every layer has a *width*, the number of functions included in the layer, where *depth* refers to the number of layers. Wide layers thus contain many neurons, narrow layers contain less neurons, whereas shallow networks contain few layers, deep networks contain many layers. From this terminology the name *deep learning* is arisen, though what constitutes 'wide' or 'shallow' is subjective. Neural nets always have a *input layer*, the original data, and an *output layer*, the prediction. The zero or more *hidden layers* are called hidden, because they don't give a visible output. The presence of sometimes many hidden layers causes the 'black box' nature of neural nets. Connections between the various layers are called *synapses*, the name of which is inspired by neurons in the brain (Karnin 1990). The model *architecture* refers to the whole neural net: the input, hidden and output layers, as well as the nature of the synapses and all functions at every node. An arbitrary example visualization of a model architecture is presented in figure 2.4. Along every synapse, a *weight* and *bias* is given to the information it retrieves. At the receiving node, all inputs from all synapses are added together to a value, which in turn is passed on through an *activation function* to new synapses. Activation functions are the 'simple' functions mentioned earlier and are described in paragraph 2.2.4. The application of weights and biases is expressed in equation 2.4.

$$f = x * w + b \tag{2.4}$$

With weight w, bias b, applied to value x resulting in new value f. For a fresh, untrained network, these weights and biases have to be optimized for to get a working model. To accomplish this, data is first fed through the layers, the *forward pass*, while the neural net has small randomized weights and biases. This is why every time the same neural net is trained, slight changes in resulting model can be observed. The output of the last layer are then subjected to a *loss*, or *cost*, function. Loss is determined over all training samples. How this loss is computed, depends on the problem statement, what the model should predict. Through a process called *back-propagation* or *backprop*, calculated is how the weights connecting the layers should be changed in order to minimize the average loss for *batch size* number of samples. With supervised learning, during training the desired output is known, thus the optimal weights and biases of the nodes can be back-calculated by use of *stochastic gradient*. It is called stochastic, because the subset of samples used to calculate loss gives an estimate of overall model performance. It is called a gradient, as the gradient to the optimal solution is back-calculated. There is actually more behind back-propagation, but more is not necessary to know to understand this research. For a more thorough explanation, the reader is kindly directed to Goodfellow et al. 2016, pages 204-219. The training data is fed through the neural net *epochs* times, which means that a neural net model can learn something from a training sample epoch number of times. Every epoch and for every batch, the same loss function is applied. Batch size and number of epochs can be of great influence on under- and overfitting of the networks. At increasing number of epochs, loss can still drop, while the model actually can overfit on the training data (Weigend 1994). To avoid overfitting, a good number of epochs and the right batch size should be estimated by checking the loss for a subset of data, the validation data. Increasing loss on validation data and reducing training loss at the same time during training is a major indication of overfitting. Training thus should be aborted before overfitting.

2.2.3. Common loss functions

Loss, cost or objective functions are the functions used to test model performance. They determine how 'wrong' a model is. This assessment is used to determine the stochastic gradient to the hidden layer, previous to the output layer. Two very common loss functions are mean squared error (MSE) and binary cross-entropy (BCE), mathematically expressed in equation 2.5 and equation 2.6. Mean square error is the typical loss function in problems where quantities are predicted, also called regression problems. Binary cross-entropy is the typical loss function for binary classification problems. Binary classification problems are problems where either class '0' or class '1' are assigned. Where model architecture is described, in paragraph 3.3.1, elaborated will be why and where these loss functions are used.

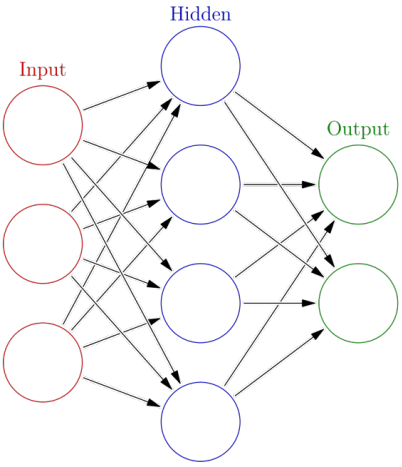


Figure 2.4: An example visualization of a neural net architecture (Albright, 2016). Circles resemble nodes and arrows synapses. At the nodes non-linear functions are applied to its input. Along synapses, a linear transformation is applied to the output of nodes. The input layer is the original data, with every node in the input layer resembling the value of a time series at a particular time. The output layer is also called the prediction. The input and output are visible to the user, which is why the hidden layer(s) are called 'hidden.'

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2.5}$$

$$BCE = -\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i log(p(y_i)) + (1-\hat{y}_i) log(1-p(y_i))) \tag{2.6}$$

With N the number of samples, y the predicted outcome, $\hat{y}$ the truth and $p(y)$ the probability of a point to be labelled '1'.

2.2.4. Common activation functions

Neural nets attempt to approximate a function by use of smaller functions. When passing information through synapses a linear combination of the output of the previous layer is made. This limits the predicted function approximation to be of linear nature. The linear combinations used as input of a node is therefore the input for an activation function, a function with beneficial properties. These activation functions introduce non-linearity to the approximated function. Depending on the circumstances, different activation function can be useful. The three most widely used activation functions are the sigmoid, the hyperbolic tangent, or tanh, and the rectified linear unit, ReLU, functions. These are also the only activation functions used in this thesis. Their respective mathematical representation are given in equations 2.7, 2.8 and 2.9, they are visualized in figure 2.5. Note that when no activation function is selected, the linear combination itself can be passed on through the network.

$$h_{\text{sigmoid}}(x) = \frac{e^x}{e^x + 1} \tag{2.7}$$

$$h_{\tanh}(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{2.8}$$

$$h_{\text{ReLU}}(x) = max(x, 0) \tag{2.9}$$

Important feature in the sigmoid and tanh functions is the presence of asymptotes. This guarantees that any continuous function can be approximated (Leshno et al. 1993). Both the tanh and sigmoid functions reach (close to) their asymptotes with relatively small input. The beneficial property of the sigmoid function is that it squashes the local output between 0 and 1, where the tanh functions squashes local output to between -1 and 1. This limits the local output at different nodes, which is useful. In a binary classification exercise, where output should be either 0 or 1, a sigmoid function limits the output to be within the desired range. The ReLU function causes output to be positive. In cases where no negative output is desired, this is practical. When predicting a number of tangible objects, number of apples in a basket for example, no negative amount of objects can be predicted, no negative amount of apples can be in a basket. The ReLU helps in these cases.

Term	Description
Neuron	Place where in neural nets a small function is applied to the previous
Node	See neuron
Width	Number of neurons per layer in neural net
Depth	Number of layers in neural net
Synapse	Connection between neurons, place where weight and bias are applied
Weight	Multiplication factor along synapse. Learnable parameter
Bias	Addition along synapse. Learnable parameter
Architecture	Name for specific collective configuration of a neural net, which includes all layers, loss and activation functions
Back-propagation	Algorithm used to determine gradient to correct weights and biases in backwards direction based on measured loss after the forward pass
Backprop	See back-propagation
Forward pass	Usage of an un-optimized neural net, in order to start back-propagation process
Loss function	Function which quantifies the prediction error made by a neural net
Cost function	See loss function
Activation function	Function applied to previous at a neuron, introduces non-linearity to the model
Input layer	Layer in neural net which takes in original data
Output layer	Layer in neural net which provides the output
Hidden layer	Layer in neural net unseen to the user
Stochastic gradient	Algorithm used to update weights and biases, using gradient from back-propagation
Batch size	Number of samples used during backprop process

Table 2.2: List of used deep learning-related terms

2.2.5. The multilayer perceptron

The multilayer perceptron consists of fully connected layers, also called dense layers, and is conceptually the simplest of neural net. It consists of nodes or neurons, each of which is connected to all nodes in the previous layer. Hence, when the first hidden layer is fully connected, that means every node in that layer is connected to every datapoint used as input to the network. The width of a dense layer is the amount of nodes in that layer. At each node, a weighed sum is made of the input, which often is passed through a nonlinear function, an activation function (Lippmann 1987), see Section 2.2.4.

In multidimensional space, the number of weights to be calculated increase exponentially with each dimension for fully connected layers. As more and more connections per node are made, more weights and biases need to be determined. Within the context of analyzing short time series, a 1D problem, this poses no problem. With every layer added, an abstraction of the previous is made. For example, consider a multilayer perceptron with two dense layers with width 2. Input is a time series and output the likelihood of a peak in the middle. The two local outputs of the first dense layer might represent the derivative of the first half and the second half of said time series. If one is positive and the other negative, this information can then be combined in the second layer to indicate the possible presence of a peak in the middle of said time series. Admittedly, abstractions made by a neural net are not humanly explainable in a majority of cases, when number of layers and interactions are large.

2.2.6. Recurrent neural nets

Recurrent neural nets specialize in processing sequential data. They are widely used in speech recognition and language translation (Mikolov et al. 2010; Zaremba et al. 2014). Their performance in speech recognition is unrivaled. Recurrent neural nets are particularly effective on speech recognition tasks, because the complex sequential nature in language. An adjective might refer to a word elsewhere in a sentence, be placed in front or after a word. Different languages might use as many, less or more words to directly translate into one another. Multiple words can mean the same. Context matters in speech and language, thus processing sentences as sequences makes sense. As recurrent neural networks specialize in processing sequential data their use is explored in this research. They process a sequence of vectors by applying a recurrence formula at every

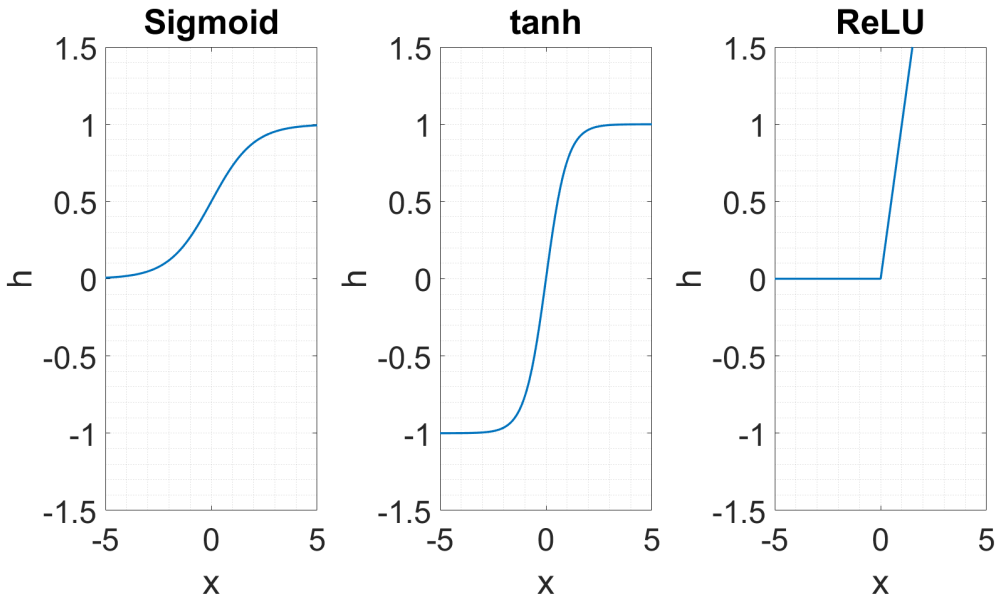


Figure 2.5: sigmoid, tanh and ReLU functions visualized. These are widely used activation functions applied at neurons. All of them introduce non-linearity. Beneficial properties of the sigmoid and tanh functions are their asymptotes at $h = 0, 1$ and $h = -1, 1$, respectively. This causes local output to be within these ranges. The ReLU function causes output to always remain positive, which is beneficial when an output must be positive.

timestep, as described in equation 2.10.

$$h_t = f_W(h_{t-1}, x_t) \tag{2.10}$$

With some function (f) using learnable parameters (W), using the old state (h_{t-1}) and the current input vector (x_t) to determine the new state (h_t). The same function and same set of parameters is used at every timestep, which limits the number of parameters to learn considerably. With increasing input size, the amount of learnable parameters does not increase. Another advantage is that no fixed sequence length is required as input. Contrary to the fully connected layer in the multilayer perceptron, not all timesteps have to be fed to a recurrent layer at once. Disadvantage is that usage of recurrent layers is sequential, increasing computational time. The whole recurrent formula has to be applied to every timestep, which hampers the possibility to parallelize the process. Often a recurrent layer consists of a single tanh activation function, see figure 2.6. Another disadvantage of simple recurrent neural nets, its their inability to remember past events. The state of the model gets updated every timestep, which adjusts the 'memory' of earlier events. For this reason the Long Short-Term Memory recurrent neural net (LSTM) is created.

The LSTM is the recurrent neural net of choice in this research. A LSTM works as follows. Information is stored in a state vector, which is represented by the horizontal uppermost arrow in figure 2.6. Input is the current new information combined with previous output. Some information from this combination is multiplied with the state vector. Because sigmoid functions give numbers between zero and one, selected is whether and by how much the current input multiplies with the state vector through the left-most sigmoid function. For the similar reasons are the other sigmoid functions present; they are also called 'forget' gates (Graves and Schmidhuber 2005). Part of the input then is put through a tanh function and added to the state vector. This state vector is passed on to the next timestep. The output, the new state, is then determined by multiplying the current state vector, put through a tanh function, with the current input. This output is also passed on to the next iteration.

The LSTM is chosen as recurrent neural net of choice, because breakpoints are more easily observed after multiple timesteps, see Section 2.1. A recurrent neural net should thus 'remember' an earlier state, to more easily correctly distinguish a sudden change.

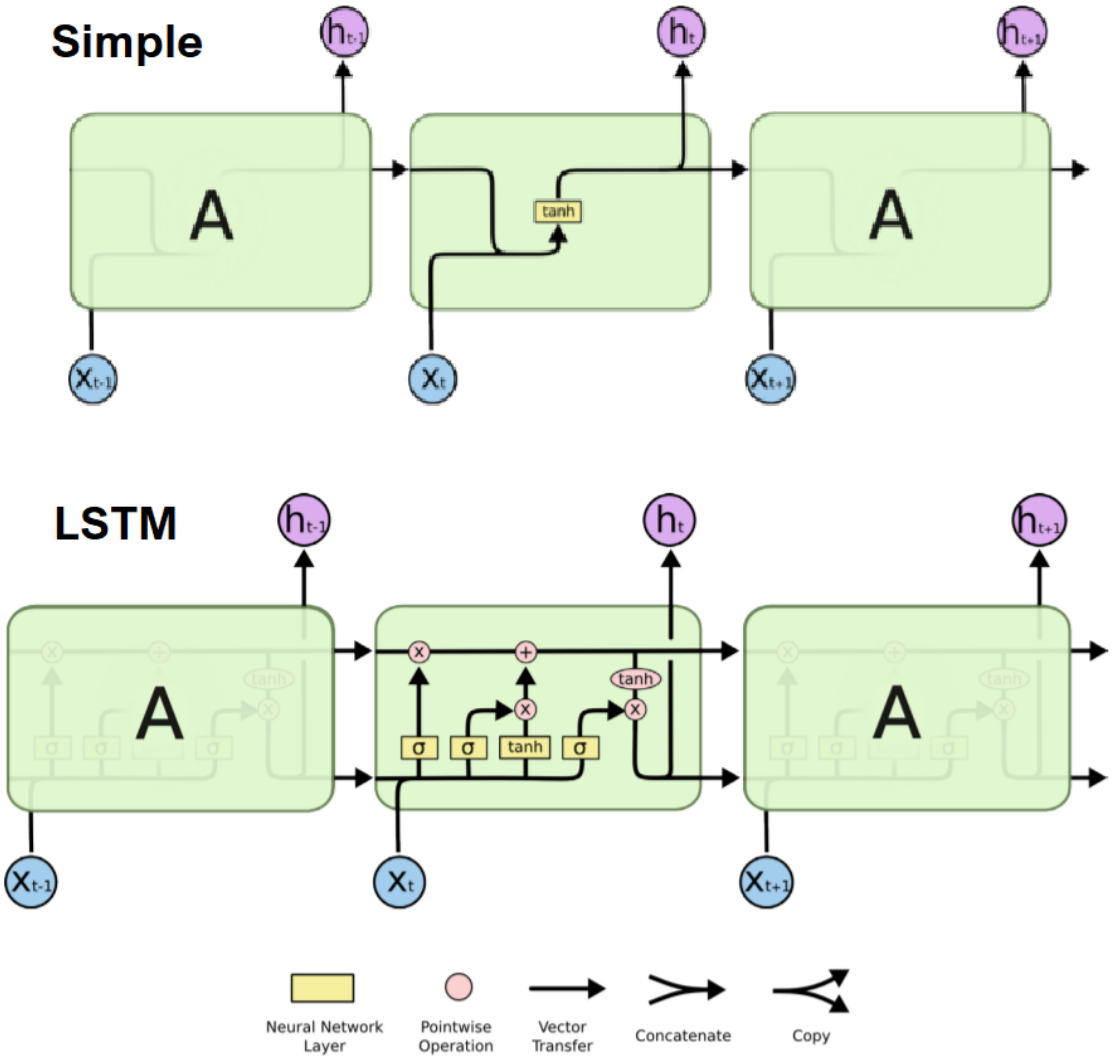


Figure 2.6: Graphical representation simple and LSTM recurrent layers. The 'Neural Network Layers' describe activation functions, described in Section 2.2.5. The recurrent neural nets (RNNs) apply the same operation repeatedly. The simple RNN applies a tanh function to the input data at time t (X_t) and to output of the previous step (h_{t-1}). The Long Short-Term Memory RNN is based on using the same function, but its memory is separate from its output. The upper horizontal arrow is the state vector ('conveyor belt'), along which memory is passed on to the next timestep. The different sigmoid functions ('forget gates') are used to select which parts of the previous state vector are used to produce an output, which parts of the new data X_t added to the new state vector and which parts of the new data are used to produce the output for the current timestep (h_t).

2.3. Change detection in remote sensing

Change detection in remotely sensed (RS) data products is done by use of many different methods. This section explains a few of them, with the aim of providing some context on the available change detection algorithms in the field of RS. More bi-temporal methods are readily available in RS than multi-temporal methods (Coppin et al., 2004; Lu et al., 2004). Dealt is with breakpoint detection in time series in this thesis. For this reason, some very common multi-temporal RS change detection algorithms will be explored: `bfast`, `PCA` and `CVA`. `bfast` is the benchmark time series change detection method used in this research. `PCA` is a transformational technique (Lu et al., 2004) used in a number of fields of study. `CVA` is an algebraic technique (Lu et al., 2004), which captures both the direction and magnitude of change. `PCA` and `CVA` are chosen, because they form the basis of many more analysis methods. `CVA`, for example, is nowadays used as a basis in development of unsupervised change detection in RS (Liu et al., 2017; Saha et al., 2019; Thonfeld et al., 2016; Zhuang et al., 2016). `PCA` and `CVA` are also indicative of the need in RS to have a skilled user (Coppin et al., 2004; Lu et al., 2004; Singh and Talwar, 2013; Verbesselt et al., 2010).

2.3.1. Principal component analysis

Principal component analysis (`PCA`) forms the basis for many multi-variate data analyses (Wold et al., 1987). `PCA` is used in many different fields. In the field of RS, it is used in forestry (Crist and Cicone, 1984), urban management (Li and Yeh, 1998), general change detection in spectral bands (Loughlin, 1991), unsupervised classification exercises (Fauvel et al., 2009; Rodarmel and Shan, 2002), chlorofyll estimates (Laliberté et al., 2018), water management (Farhan et al., 2016) and more. Similarly, the use of `PCA` has different goals. These can be, but are not limited to: Problem simplification, data reduction, variable selection, unmixing, classification, outlier detection and prediction (Ringnér, 2008; Wold et al., 1987).

`PCA` does this by approximating a data matrix X as the product between smaller matrices T and P'. Then T represents the dominant 'object patterns' of X, where P' gives the 'variable patterns' (Wold et al., 1987). In other words, every number in T is indicative of the variability in a direction. Richards, 1984 describe the procedure to generate T and P' from X as follows:

1. Derive the variance-covariance matrix
2. Compute eigenvectors
3. Apply a linear transformation on the dataset

Given an N-dimensional variable P' with mean vector M and variance-covariance matrix C_P, C_P can be estimated:

$$C_P = \frac{1}{N-1}\sum_{i=1}^{N}(P'_i - M)(P'_i - M)^T \tag{2.11}$$

With N the number of pixels, in the context of RS. These pixels should be from the same image (Fung and LeDrew, 1987). Each principal component then is described as:

$$X_j = t_{1j}P'_1 + t_{2j}P'_2 + ... + t_{Kj}P'_N = t_j^T P' \tag{2.12}$$

Where t_j^T is the transpose of the normalized eigenvectors of C_P. This summarizes to:

$$X = TP' + E \tag{2.13}$$

With E being a matrix containing the residuals, the part not captured in TP'. Values in E are ideally as small as possible. Simply by setting the size of T, the number of dimensions in which variability is sought can be set. T is thus the transpose of the normalized eigenvectors of the variance-covariance matrix C_P, such that the covariance matrix of X:

$$C_X = TC_P T^T \tag{2.14}$$

The diagonal elements of C_X are the eigenvalues λ:

$$C_X = \begin{bmatrix} \lambda_1 & . & . & 0 \\ . & \lambda_2 & . & . \\ . & . & . & . \\ 0 & . & . & \lambda_N \end{bmatrix} (\lambda_1 > \lambda_2 > ... > \lambda_N) \tag{2.15}$$

Useful information can be extracted from these eigenvalues, as they indicate the directions wherein most variability is detected. C_X may be computed by parts of images, which enables for example classifying different land-cover types as having a specific set of eigenvalues (Anuta et al., 1984). Change can then be detected by observing eigenvalues change over time. These eigenvalues are subject to change by atmospheric corrections and sun angles (Fung and LeDrew, 1987). Fung and LeDrew, 1987 note a variety of studies showing the need of observing a place with high correlation between neighbouring pixels to observe changes over longer periods of time. A high correlation in this case means no change (Coppin et al., 2004; Fung and LeDrew, 1987).

PCA is a technique that relies on the need for setting thresholds (in λ) to interpret, requires high correlation between different images and is scene dependent, which is why it requires a lot of skill of the user (Coppin et al., 2004; Lu et al., 2004). This means that research done in one region, where perfect thresholds to distinguish land cover change might be established, does not translate easily to another region.

2.3.2. Change vector analysis

Change vector analysis (CVA) compares parameters over two or more successive time periods (Lambin and Strahlers, 1994; Malila, 1980). Both the magnitude and direction of change are provided by this algebraic method (Coppin et al., 2004; Lu et al., 2004; Malila, 1980). CVA is employed in forestry (Malila, 1980; Nackaerts et al., 2005), desertification research (Bayarjargal et al., 2006), assessing wetlands (Baker et al., 2007), land-cover classification (Chen et al., 2003; Lambin and Strahlers, 1994) and more.

By combining multiple images, the time-trajectory of a pixels' parameter can be compared over different epochs. The change vector herein is the vector difference of the multidimensional parameter vectors constructed for the different epochs (Lambin and Strahlers, 1994; Malila, 1980). The parameter vector contains often a multitude of spectral bands (Singh and Talwar, 2013). The multitemporal parameter vector p contains different parameters I at time t during epoch y:

$$p(i,y) = \begin{bmatrix} I(t_1) \\ I(t_2) \\ ... \\ I(t_N) \end{bmatrix} \tag{2.16}$$

Where N is the number of timesteps during each epoch and i the pixel observed. The change vector c between two epochs then simply becomes the change in two parameter vectors:

$$c(i) = p(i,y_1) - p(i,y_2) \tag{2.17}$$

By observing change over multiple epochs, CVA becomes a multi-temporal change detection. Figure 2.7 visualizes how a change vector can be used to distinguish between desert and savanna. The desert and savanna can be differentiated by comparing the winter vegetation with the summer vegetation. The desert does not have vegetation, thus shows no difference in vegetation activity between winter and summer. The savanna shows more vegetation activity during summer. The change vector to indicate typical savanna behavior can be constructed from these observations. In this way, different change vectors can be created to label different land-cover types. A change in change vector can then be coupled to changes in the environment.

Setting suitable thresholds to distinguish between change and noise takes a lot of expertise (Lu et al., 2004). These thresholds are empirically estimated, thus completely depend on the skill of the user and might even be influenced by external factors (Singh and Talwar, 2013). Coppin et al., 2004 also mention that CVA requires meticulous data handling, as slight artefacts in images can change the change direction. A good understanding of the observed process (Singh and Talwar, 2013) is also necessary. An advantage of this technique is that all remotely sensed data can be incorporated, somethings bfast is not inherently able to do.

2.3.3. bfast

bfast is created to deal with the various types of change which occur in RS data (Verbesselt et al., 2010). In RS data, changes in pixels' time series can in most occassions be changes in trend, or changes in seasonal

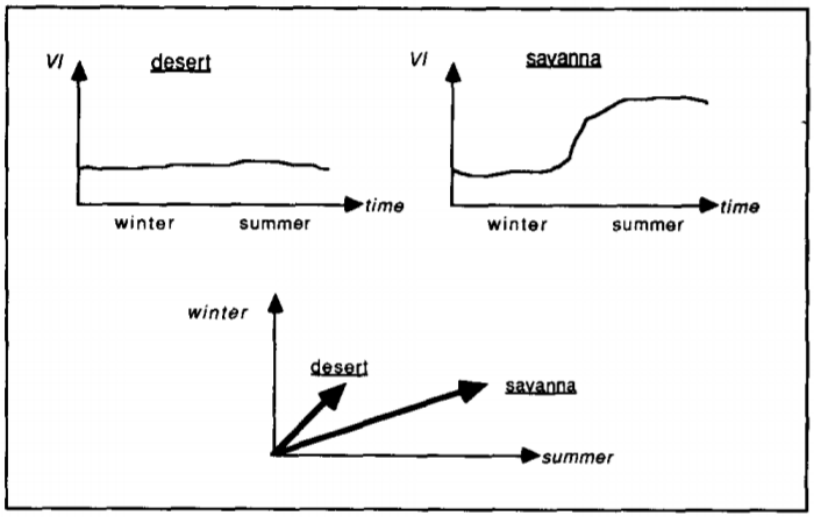


Figure 2.7: Example change vectors for two simulated land-cover types (Lambin and Strahlers, 1994). The top graphs show the development of some parameter over time in two biomes. Similar behavior is observed during winter and summer for the desert, thus its change vector follows a 1:1 trajectory. The savanna shows increased values during summer, thus its change vector is larger and skewed towards higher values in the summer. The nature of this change can then be rationalized. In this case, the parameter can be a NDVI time series or any vegetation indicator.

parameters, both of which can indicate disturbances (Verbesselt et al., 2010). It is thus useful to distinguish a seasonal and a trend component to these time series. `bfast` fits a season-trend model to time series.

The `bfast` package can be used to estimate the time and number of abrupt changes within time series (Verbesselt et al. 2012). It characterizes breakpoints by their magnitude and direction. The package contains multiple functions, with at its core the `bfast` function. In this research an adjusted `bfast` function is used: `bfast01`. This function works similar to `bfast`, but is optimized for detecting the most influential trend shift in the time series (De Jong et al. 2013). The season-trend model used in `bfast` is described as:

$$y_t = \alpha_1 + \alpha_2 t + \sum_{j=1}^{k} \gamma_j sin(\frac{2\pi j t}{f} + \delta_j) + \epsilon_t \tag{2.18}$$

The `bfast` function decomposes the signal (y) into linear trend (α), seasonal amplitude (γ), seasonal phase (δ) and remainder components (ϵ), for a yearly measurement frequency (f) (De Jong et al., 2013; Verbesselt et al., 2010). The seasonal components of the signal can be ignored for RUE time series, due to its annual signal, see also Chapter 5. Equation 2.18 can then be rewritten to equation 2.19.

$$y_t = \alpha_1 + \alpha_2 t + \epsilon_t \tag{2.19}$$

An iterative procedure using an ordinary least squares (OLS) residuals-based MOving SUM (MOSUM) test is performed until number and position of the breakpoints are unchanged. The `bfast01` function uses the same OLS-MOSUM test, though it considers only the possibility of a single breakpoint. The MOSUM test is described as:

$$M_t = \frac{1}{\hat{\sigma}\sqrt{n}} \sum_{(t-h+1)}^{t} \hat{\epsilon}_s \tag{2.20}$$

With bandwidth h, measured standard deviation $\hat{\sigma}$ and time series length n. M_t should be close to zero and fluctuate randomly in a stable model. If structural change occurs it will deviate systematically from zero. When the maximum absolute value $max_t|M_t|$ exceeded its 5% asymptotic critical value, a breakpoint is declared (Chu et al. 1995). In simpler terms, this means that if the sum of the residuals deviates exceeds the expected values, which are based on the measured standard deviation of a segment, a breakpoint is declared in the middle of assessed segment. This means that for a greater standard deviation in a part of the time series causes less sensitivity to breakpoints. Both this asymptotic critical value and bandwidth can be set to custom values. Two models are fitted, where the breakpoint timing is estimated by minimizing the OLS residuals. A

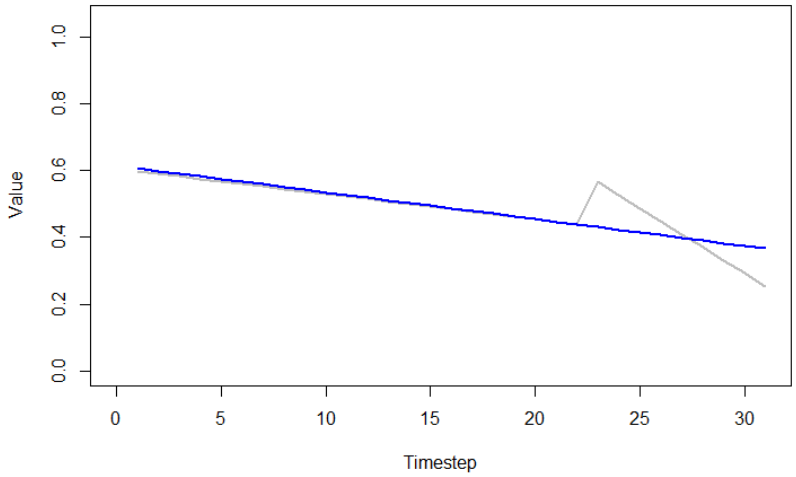


Figure 2.8: Example in breakpoint timing prediction of `bfast01` on noise-free data. In blue the fitted timeseries is presented, with in grey the original signal. No breakpoint is detected. This is caused by the large bandwidth used in the `bfast01` settings, which removes the ability of `bfast01` to detect breakpoints close to the edges of the time series

certainty bound also is calculated. An example how a fitted breakpoint is predicted by `bfast01` is shown in figure 2.8. What is evident from this example, is that the bandwidth used can greatly influence the quality of a detected breakpoint.

3 Method

This research investigates how breakpoints can best be detected using deep learning models. Any deep learning approach should outperform current breakpoint detection systems. This is why a statistical breakpoint detection system is compared to the deep learning approaches tested. The statistical method of choice is the `bfast` package for R (Verbesselt et al. 2012), `bfast01` more specifically. They are compared by means of a synthetic dataset, as well as on a user-case: Rain-use-efficiency (RUE). `bfast01` is chosen as comparison, as it should function optimally on the generated data. It also already is in earlier research on RUE. The basic concepts behind `bfast01` and neural nets are explained in Chapter 2. What RUE is, how it is created and how models perform on these time series, is described in Chapter 5. In this chapter the general workflow is described, followed by a thorough description of the created synthetic data and the proposed neural net architectures.

3.1. Workflow

To train a neural net to real-world data, a lot of labelled data is required. In any real-world example, whether and when a breakpoint occurred can almost never be known for certain (Wei and Keogh, 2006). This absence of training data in real-world cases is overcome by the use of synthetic data. Compared to real data, various advantages are obtained by usage of synthetic time series:

1. Both the accuracy and precision of a breakpoint can objectively be determined
2. Optimal settings for both `bfast01` and neural networks can be determined
3. Enough labelled training data can be generated

By simulating all possible time series, the neural nets developed are supposed to be generically usable for breakpoint detection in problems which behave similar to the generated time series. Neural nets are expected to generalize well (El-Sharkawi, 1996; Goodfellow et al., 2016; LeCun et al., 2015), which is why they should work well even if the training data is slightly different from the real data.

First step is the generation of time series. There are more ways to utilize deep learning methods in breakpoint detection problems, thus second is the definition of all neural nets used. The performance of all used models, including the benchmark `bfast01`, is determined by observing accuracy and precision (as described in Chapter 2) of all models. This workflow is visualized in figure 3.1. Figure 3.1 also signifies which concepts are taken from or inspired by literature.

To assess which models work best to detect breakpoints in time series, they should be compared in a consistent framework. They should be judged based on the same performance metrics. The performance metrics selected are generally applicable, which gives way to assess all possible breakpoint detection systems' performance similarly. Makridakis et al., 2018 state the need to compare neural net performances with statistical methods. `bfast01` is chosen as comparison of choice.

Subjectively considered most important performance metric is the model accuracy. Accuracy is a measure of how well the occurrence of a breakpoint is (not) detected, which is deemed most useful in practical applications. Next to accuracy, also the false positive and false negative ratios are measured, as these can give valuable insights. Second metric to assess is the precision. Precision is quantified in the form of the standard

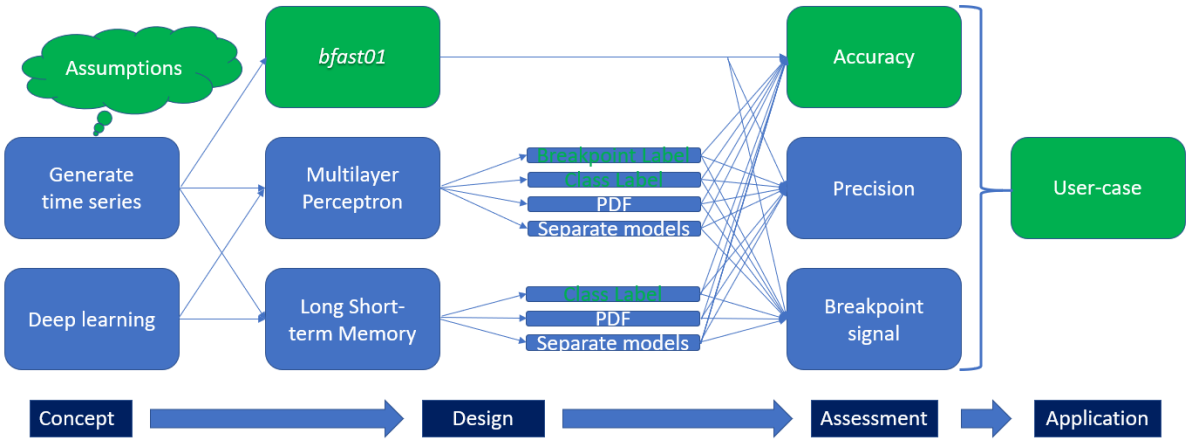


Figure 3.1: Schematic overview workflow. This research can be separated in four stages, which are in consecutive order: the stage of design, model training, model assessment and application to a user-case. These roughly correspond with Chapters 2 & 3, Chapter 3, Chapter 4 and 5, respectively. The green boxes indicate concepts which are taken from literature; Assumptions are loosely based on the user-case, `bfast01` is not developed in this thesis (Verbesselt et al., 2010) and the accuracy metric is widely used in change detection Aminikhanghahi and Cook, 2017). The green letters indicate concepts which are used in other ML change detection methods as well (Aminikhanghahi and Cook, 2017).

deviation, the deviation of the detected breakpoint time from the true breakpoint time. Standard deviation is a concept which is used in many different contexts. If the offset between predicted and true breakpoint times is normally distributed, the standard deviation can directly be translated to some a certainty (see also Section 2.1). The best performing models are investigated more thoroughly. Two models are defined (see Section 3.3), of which at least a single model ideally is tested more thoroughly. For these best working models in terms of accuracy and precision, normality in precision is tested for using a visual and a statistical normality. The visual normality test is a quantile-quantile plot. A quantile-quantile plot is a method to compare two data distributions, by plotting their respective quantile values against each other (Gnanadesikan and Wilk, 1968). The statistical test chosen is the Shapiro-Wilk test, the most powerful normality test (Razali et al. 2011). This test is implemented by utilizing the SciPy library in Python. The accuracy and precision are measured as a function of noise in the dataset. This allows to identify when a model 'breaks'; it gives insight in its sensitivity. A third metric to assess the quality of a detected breakpoint, is by comparing the breakpoint signal with the model certainty. This gives some insight in the relation between the 'observability' of breakpoints and their detection.

The synthetic data should resemble real data. However, because the aim is to determine how deep learning methods can best be used for breakpoint detection, simple time series are used. This limits any direct application of produced models to real-world problems. That is why any model trained on such simple time series should only be applied to real-world problems where linearity in change is expected. The user-case used in this thesis has but two influences on the methodology: It sets the length of created time series and aids in selecting at what noise level models performance should be optimal. Time series length can be set arbitrarily, so that has little to no influence on any outcomes in this research. Optimizing model performance for a specific noise level is also somewhat arbitrary. The noise levels at which to critically judge these performances, are the noise levels observed in the real-world data. While this may sound of large importance, it had minor influence to the choices made in this methodology.

3.2. Synthetic data specifications

The synthetic time series are produced in four steps, which are visualized in figure 3.3. Firstly, some points are randomly generated: a breakpoint time, breakpoint value, start value and end value. The time series are of equal length as the user-case time series: 31. In the user-case, this means a value per year, for 31 years. The two limits of these generated points are their range, between 0 and 1, and the breakpoint is not generated to be at the start nor at the end of the time series. This is because no algorithm is expected to determine such a breakpoint, as no breakpoint signal would be present in the time series. In cases no breakpoint is present, only a start and end value are generated. To take into account a sudden change in the environment, after

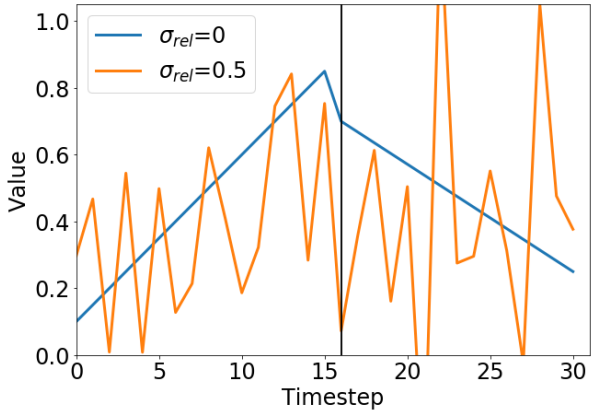


Figure 3.2: Two synthetic time series with identical signal, but varying noise. The black line indicates the breakpoint. Noise of the form $N(\mu = 0, \sigma = x * R)$ is added, where R refers to the range of the time series and x is mentioned as σ_{rel}. The blue line has noise equal to the lowest noise levels used in the synthetically produced data, where the yellow line has noise equal to the highest noise levels used. In the case of maximum noise, no breakpoint features are visually detectable.

which a system changes permanently, a change in mean is added at the breakpoint. This change in mean is randomly generated, normally distributed, and divided by 4. This division factor is inspired by looking at the time series of the user-case, though testing showed it has negligible influence.

The second step is the linear interpolation of all points in the time series, which then have values assigned. This corresponds to the second image from above in figure 3.3.

The third step is the addition of noise, which is assumed to be white. The white noise is of the form $N(\mu = 0, \sigma = x * R)$, with R the range of the time series and a multiplication factor. This multiplication factor is coined hereby as the the *relative noise*, or σ_{rel}. σ_{rel} from 0 to 0.5 is added to the time series. Thus, a σ_{rel} of 0 indicates no noise, a σ_{rel} of 1 indicates white noise with a standard deviation as big as the range of the signal. At a σ_{rel} of 0.5, the signal can already almost not be recognized, see also figure 3.2. Note that the assumption that noise is white should be checked in any real-world application.

After the noise is added, the last step is normalization. All time series are normalized to lie in between 0 and 1. Normalizing the input data is beneficial for the training of the neural nets (Ioffe and Szegedy 2015). It increases learning rates during training. Normalization is done by use of the minimum and maximum values of the dataset, not by use of the range of a particular time series. These minimum and maximum value can have outliers in the positive and negative direction. Therefore, this normalization process gives a bias in the synthetic data towards a value of 0.5, 'the middle'.

The synthetic data consists of 110000 time series, half of which contains a single breakpoint. Single breakpoints are added, as opposed to multiple, because this study is a feasibility study of sorts. This study aimis to check whether deep learning problems optimize properly in breakpoint detection. That means they should definitely be able to detect them in the simplest of cases.

Chosen is to have half the dataset contain a breakpoint, as not necessarily a breakpoint occurs in real data. The number of time series containing breakpoints during training influences the sensitivity of neural nets to breakpoints. Thus when the algorithms are 'shown' more data with breakpoints, it becomes more sensitive to them.

The 110000 time series are separated into smaller datasets with varying degrees of σ_{rel}. Data is created with a σ_{rel} from 0 to 0.5, with the arbitrary step size of 0.05. At a relative noise level of 0.5, signals are already almost impossible to distinguish, see figure 3.2. This noise range is assumed to incorporate all possible real-world noise levels. Note that high σ_{rel} does not automatically entail a bad breakpoint signal. As described in 2.1, the signal of a breakpoint differs from the signal of the process at hand. The signal of a breakpoint is not inversely correlated with the noise in a time series. To put these noise levels into context, the σ_{rel} of the user-case can be measured: By taking the σ_{rel} of 103843 Australian pixels' RUE time series, it is found that the user-case σ_{rel} lies in between 0.05 and 0.4, with an average and a median value of 0.20. This means that a relative noise range in between 0 and 0.5 incorporates at least all possible noise for the user-case.

A separate, similarly created dataset is created as training dataset. The total number of time series required is assessed in scales of 10 for MLP_{BP} (see Section 3.3 for a description of this neural net model). No significant increase in model accuracy is observed when the model is trained on more than 10000 time series

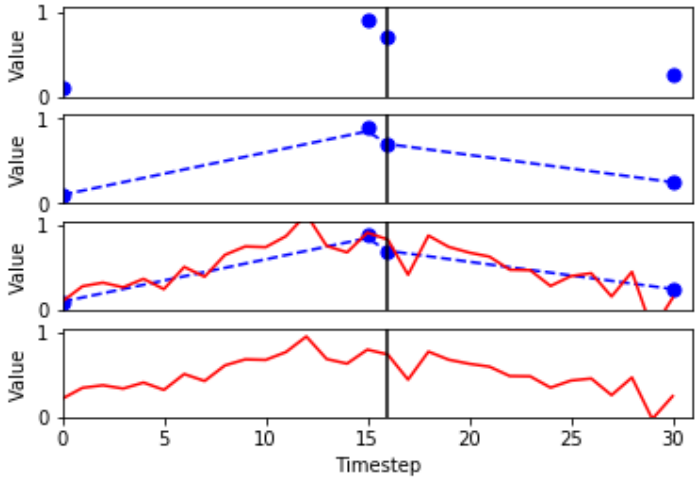
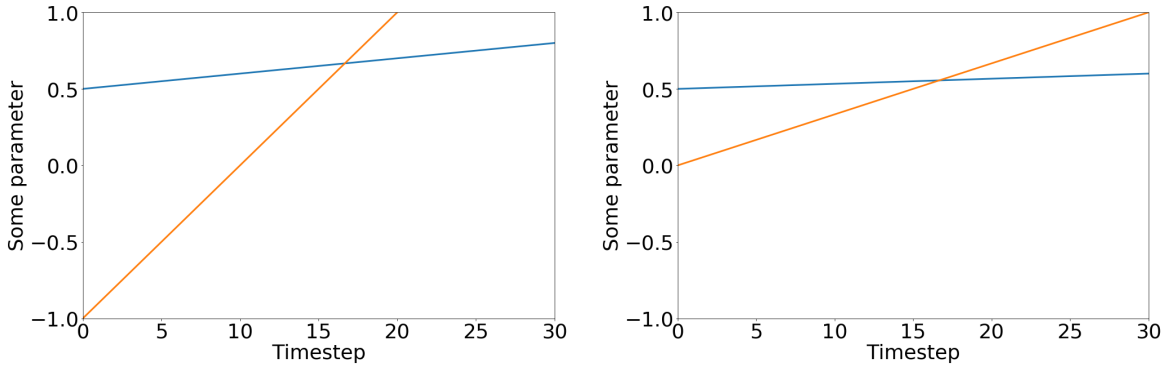


Figure 3.3: The steps taken to produce synthetic time series. From top to bottom: The randomly generated points used to describe the signal, where both a change in regression structure and change in mean are added. These points are linearly interpolated. Third is the addition of noise, which is of the form $N(\mu = 0, \sigma = x * R)$, with R the range of the time series. Last is the normalization of all time series. In case no breakpoint is present, the only difference is a reduction in number of generated points to define the signal during the first step.



(a) Two lines before normalization

(b) Two lines after normalization

Figure 3.4: An artefact of normalizing data. The yellow line has a much larger minimum and maximum than the blue line, which causes the blue lines values to be pressed towards a value of 0.5

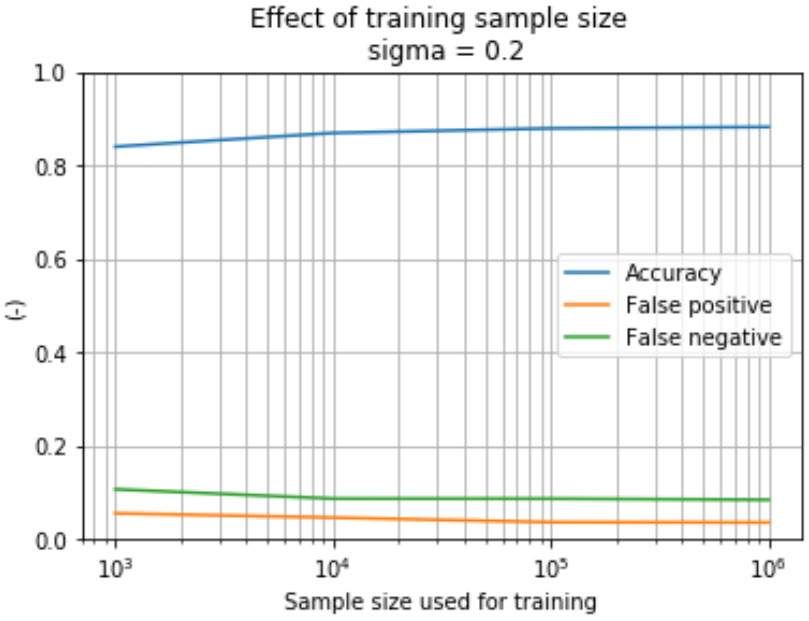


Figure 3.5: The observed effect of changing the number of training samples per noise level for MLP_{BP} accuracy. All accuracies are measured at the same relative noise level. From a sample size of 10000 per noise level, no significant increase in accuracy is observed.

per noise level, see figure 3.5. A dataset containing 10000 samples per noise level thus is deemed large enough for training purposes.

Performance estimations will be linked to the observed σ_{rel}. This will provide insight into how well various models handle noise, when a model 'breaks.' Such a link is useful, because the noise present in a dataset can have major influence on the performance of breakpoint detection methods. σ_{rel} can be estimated for the original data. When no breakpoint is present, σ_{rel} can be estimated quite closely. In time series with a breakpoint σ_{rel} will be overestimated in most cases, because a single linear fit is sub-optimal to describe the signal. Figure 3.6 provides an example to show this tendency for linear fits to underestimate σ_{rel} in breakpoint containing time series. The expected breakpoint detection performance when applied to an unseen dataset will thus likely underestimate the true model performance, when making use of a link between σ_{rel} and model performance. Accuracy and precision of breakpoint detection systems are expected to decrease with increasing σ_{rel}.

3.3. Proposed neural net architectures

Two types of neural net models are proposed: The multilayer perceptron (MLP-model) and a purely LSTM-based neural net (LSTM-model). Chapter 2 described what their fundamental properties are. These two models are kept very simple, which gives way to reasonably judge what type of architecture is best suited for breakpoint detection in time series.

Most important in any deep learning problem is the question: What to look for? To argue why a neural net architecture is designed as it is, it is useful to know beforehand what the neural net architecture should predict. This is why first some ways to describe breakpoints will be described. These are the targets to 'hit' for the proposed neural nets. A different loss function should be used per breakpoint definition, which is noted with the breakpoint definitions themselves. Afterwards, both types of network layouts will be elaborated. Note that the use of convolutional neural nets, as the third neural net architecture, is not explored for reasons described in appendix 10.1.

The naming convention for neural nets is $\textit{Model type}_{\textit{Loss type}}$, where both *Model type* and *Loss type* are abbreviations of their name. These abbreviations are mentioned in brackets above the following sections.

3.3.1. Breakpoint definitions and loss functions

There is no perfect way to describe a breakpoint. In order to find breakpoints in time series, a variety of descriptions for breakpoints are used. These descriptions are what the neural net models should predict. With

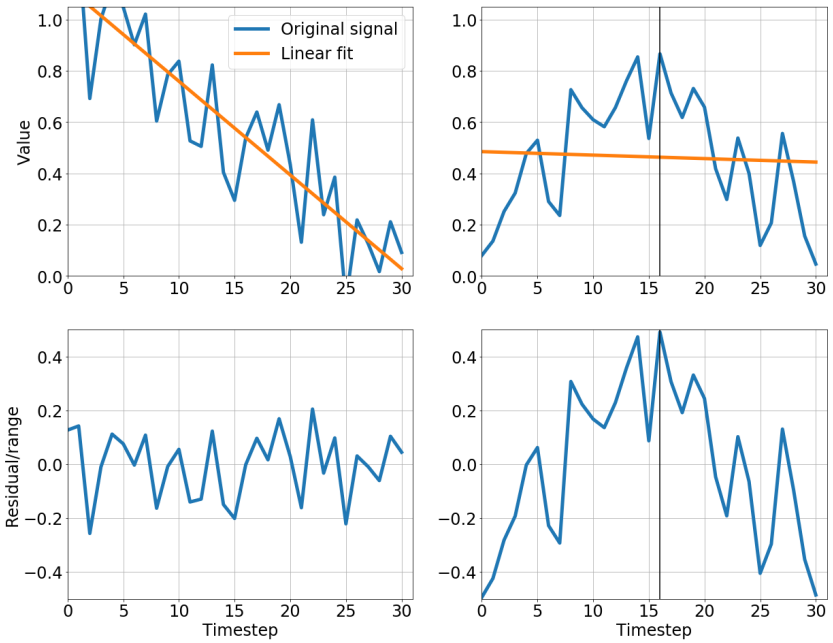


Figure 3.6: Two time series for which σ_{rel} is determined with a linear fit. Vertical black lines indicate the breakpoint times. The upper images show the original data and the linear fit, the lower images the corresponding residuals, normalized by the range of the sample. On the right is a time series with a breakpoint, the left image does not contain a breakpoint. Both lines have the same σ_{rel}: 0.2. It is observed that in both cases the true σ_{rel} can be estimated through a linear fit, where in the case of a breakpoint σ_{rel} gets overestimated.

the presentation of the network architectures (Section 3.3.2), hypotheses are drawn as to what model would work well with which breakpoint definition. With every breakpoint description, it is briefly discussed how to interpret that breakpoint definition and what inherent performance indicators are present. Each breakpoint definition 'searches' for something different, thus a different loss function should be applied to each of them. With their presentation, the proper loss function to use is described. The loss functions *Mean-Square-Error* (MSE) and *Binary Cross-Entropy* (BCE) are described in Section 2.2.3. The shortcomings of the breakpoint definitions used is also briefly mentioned. Four ways to differentiate breakpoints are tested:

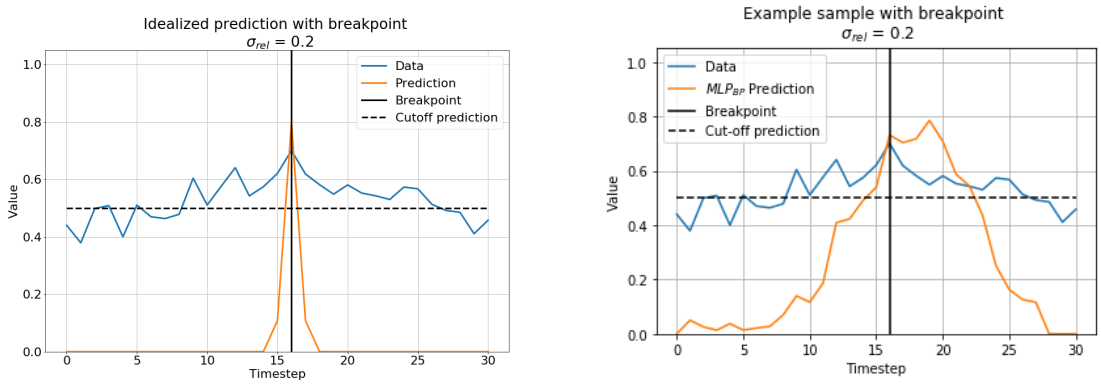
1. Breakpoint label (BP)
2. Class labels (CL)
3. Probability density function (PDF)
4. Separate models (SM)

Each breakpoint definition is visualized, where an ideal and an applied case are shown. An applied case means that a proposed model is applied to a time series created as described in Section 3.2. In real-world application, multiple breakpoints can occur. Hence after these visualizations, a short comment is made on how such a breakpoint definition would fare in a situation with multiple breakpoints. This section is ended with a summary through an example time series with breakpoint, where the applied output in every case is given.

Breakpoint label (BP)

Breakpoint label means the time series are labelled as '0' where there is no breakpoint and '1' where there is one. All points in the time series thus are labelled. This is the most intuitive of all the approaches, as a model will automatically optimize to find the area of interest: the breakpoint itself.

Roughly 1.6% of the data thus is labelled as a '1'. A neural net thus predicts numbers between 0 and 1 for every timestep, where a prediction above 0.5 is considered a breakpoint. However, if trained on these labels, the "optimal" result is to label all points as a '0', or below 0.5. In order for the neural net to actually recognize breakpoints, an increased weight is given to the points labelled as '1'. This weight should be chosen with care. Too high of a weight results in the net optimizing to labelling all points as 1. Too low of a weight results in



(a) Example idealized prediction

(b) Example actual prediction MLP_{BP}

Figure 3.7: Examples 'Breakpoint label' prediction. Vertical black lines indicate the breakpoint times. Left is the ideal model response to the this time series with breakpoint in the middle. If the prediction is above the cutoff value of 0.5, it is considered a breakpoint. Right is the actual prediction of one of the models described in the text. The shape of the prediction in the right image comes close to a probability density function, which means the model works as intended.

the earlier mentioned labelling of everything as 0. A good balance seems to be a weight of half the time series length: 15.5. The breakpoint constitutes roughly a third of what in total can be "learned" from a single time series with a breakpoint. Higher weight gives trouble interpreting resulting predictions, lower weight does not provide enough sensitivity to breakpoints.

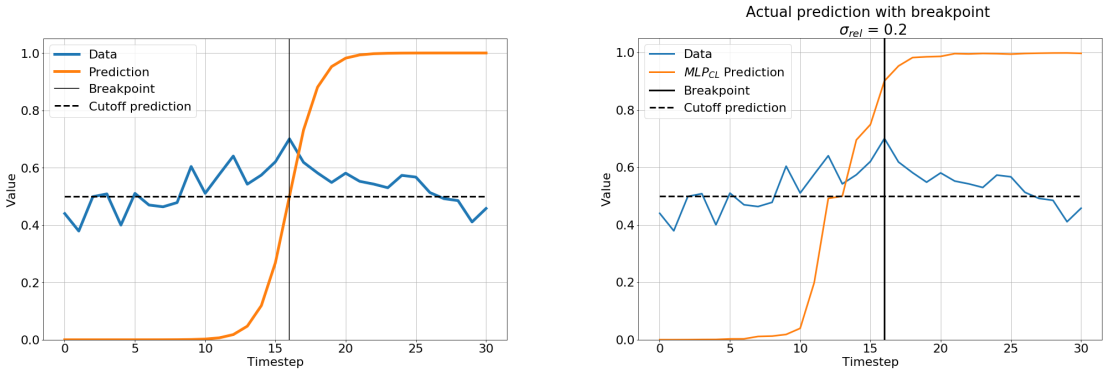
The loss function is chosen accordingly; BCE is used as loss function, as this is a binary classification exercise. The output is a prediction between 0 and 1, with a lower predicted value where there is no breakpoint. This means that a higher prediction should correspond with a higher certainty of a breakpoint and vice versa. An idealized prediction is presented in figure 3.7a. A strong correlation between the maximum predicted value of a time series and accuracy estimate is expected, the maximum value of a predicted time series can be used to provide accuracy per relative noise level. Given the restrictions of the models, multiple points surrounding the predicted breakpoint will be predicted to be a 1 in the case of a highly certain breakpoint. The 'mean predicted breakpoint' is considered the predicted breakpoint, as the 'maximum predicted value' is often not at the actual breakpoint time. An example of this phenomenon is visualized in figure 3.7b. The 'mean predicted breakpoint' is the weighted average using the predicted values above 0.5. The weight is based on how much the prediction peaks above a value of 0.5 at a point. This means that the 'certainty' of the model is taken into account.

This weighted average makes no sense if multiple breakpoints are incorporated into the problem. However, if wanting to detect multiple breakpoints, more points can be labelled as a '1' and the weighting of such points can simply be adjusted to a lower amount. Breakpoints close to one another will be hard to detect though, because

Class labels (CL)

Class labels mean that all points in a sequence are classified as '0' before the breakpoint or '1' after the breakpoint. This also is an intuitive way to describe a breakpoint: Humans could describe a line with breakpoint as a line with two segments. In this case BCE is used as loss function, as this also is a binary classification exercise. An ideal prediction is shown in figure 3.8a. Drawback of defining a breakpoint in this manner, is that the interpretation of results gets complicated. There is no restriction applied to force only sequences of 1's or 0's. For example a sequence which should be labelled [0,0,0,1,1] might be incorrectly classified as [0,0,1,0,1], causing problems in interpreting model results; What is the breakpoint time in this case? A [0,1] transition, or the average of such transitions? This problem is also visualized for a user-case example in figure 3.9. A-posteriori interpretation of the output is done as follows. To check the occurrence of a breakpoint, the last element in the prediction is inspected. The last element should theoretically always be labelled a '1', if a breakpoint is detected using this breakpoint definition. Breakpoint time is determined by averaging all [0,1] sections in the predicted samples.

The prediction increases ideally over time, with a prediction over 0.5 after the breakpoint has occurred. Similar to the 'Breakpoint label' breakpoint definition, the magnitude of values predicted can provide an indication of how 'sure' the model is a breakpoint has occurred.



(a) Example idealized prediction

(b) Example actual prediction MLP_{CL}

Figure 3.8: Examples 'Class label' prediction Vertical black lines indicate the breakpoint times. Left is the ideal model response to the this time series with breakpoint in the middle. If the prediction is above the cutoff value of 0.5, it is considered after the breakpoint. Right is the actual prediction of one of the models described in the text. The shape of the prediction in the right image comes close to the idealized prediction, which means the model works as intended. The model used to construct figure 3.8b is MLP_{CL}, which is one of the neural nets defined in Section 3.3.

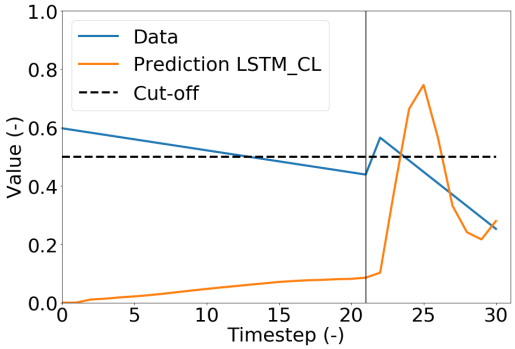


Figure 3.9: Example problem in usage of 'Class Label' breakpoint definition. The vertical black line indicate the breakpoint time. If the prediction is above the cutoff value of 0.5, it is considered after the breakpoint. The shape of the prediction is dissimilar from the expected shape, as shown in figure 3.8a, because the prediction drops after it has predicted a breakpoint. This causes causing interpreting where a potential detected breakpoint is placed. The model used to construct the figure is LSTM_{CL}, which is one of the neural nets defined in Section 3.3.

By only recognizing a single breakpoint, this breakpoint definition is limited to detecting single breakpoints only. To increase the number of possible breakpoints detected, more classes could be added, where class '2' could indicate the segment after the second breakpoint etc. Another way to enhance this breakpoint definition to incorporate multiple breakpoints, is by classifying in a different nature. Segments with a specific regression structure could be labelled, for example.

Probability density function (PDF)

Breakpoint time (μ) and uncertainty (σ_{PDF}) can be used to define a Gaussian distribution, which can represent the signal to find: the breakpoint. μ and σ_{PDF} can be fed to the normal probability density function (PDF, equation 3.1) to obtain the Gaussian describing the certainty of a breakpoint at any given time (x).

$$PDF(x,\mu,\sigma_{\text{PDF}}) = \frac{1}{\sigma_{\text{PDF}}\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2_{\text{PDF}}}} \tag{3.1}$$

μ is defined as the breakpoint time, defined at random in case there is no breakpoint. The use of random breakpoints as surrogate in non-breakpoint time series causes the model to have a slight bias in μ towards the middle timestep, as that is the best guess, minimal loss, in case no breakpoint signal is present. For this reason a high number of training iterations is run, with every iteration a new random μ for time series without breakpoint. With more data fed to the network, less biased breakpoint times should be predicted. σ_{PDF} is the arbitrary uncertainty of a breakpoint. σ_{PDF} is defined as 0.5 for samples with a breakpoint and 10 for samples without a breakpoint. These values are specified for their well looking PDF output, as depicted in figure 3.10. In a best case scenario, the PDFs constructed by use of predicted μ and σ_{PDF} have a shape similar to the ones trained for. One can argue that σ_{PDF} should be correlated to breakpoint signal strength (see Section 2.1) or relative noise level (see Section 3.2), because these inherently change the ease with which σ_{PDF} is determined. Chosen is to set σ_{PDF} to two values, because the contrary would introduce even more subjectivity to setting a σ_{PDF}.

Predicted σ_{PDF} distribution is expected to optimize into two peaks: one near 0.5 and one near 10. Somewhere in between a cut-off σ_{PDF} needs to be distinguished. Deemed most important in breakpoint detection is accuracy, see also Section 3.1, which is why the cut-off is determined by checking cut-off numbers from 0 to 12. The cut-off then is the cut-off which provides the highest accuracy, where accuracy is summed over all noise levels. A round cut-off number is defined. Every time a neural net is trained, results change slightly due to the random initializing of the weights and biases. This change with each training session causes the cut-off to shift by decimals, hence the choice to define a single, round cut-off.

Advantage of using this target output is that automatically a PDF can be produced. Output is also easy to interpret, as both μ and σ_{PDF} are directly related to precision and accuracy, respectively. This forces the model to be limited to detect only a single breakpoint, which can be considered a disadvantage. To translate this breakpoint definition into one which takes into account multiple breakpoints, a creative solution should be sought for. Suggestions for a solution to this problem are given in Chapter 6. Another disadvantage is that the presence of a breakpoint needs to be read from σ_{PDF}, which is a fictitious parameter with no real link to the real world.

The models are supposed to optimize for the shape of the PDF, not μ and σ_{PDF} necessarily. Thus to determine loss in this case, a loss function other than MSE or BCE should be defined. The Kullback Leibler (KL) distance is used to determine the difference between the true and predicted PDF (Goldberger et al., 2003; Hershey and Olsen, 2007; Van Erven and Harremos, 2014). It expresses the dissimilarities in two Gaussians, or in this case the true PDF (p) and predicted PDF (q, Goldberger et al. 2003). The KL distance for continuous functions is provided in equation 3.2, which is simplified to loss function 3.3 over N samples. The derivation from equation 3.2 to equation 3.3 is given in appendix 10.2.

$$KL(p|q) = \int p * log(\frac{p}{q}) \tag{3.2}$$

$$Loss_{KL} = \frac{1}{N}\sum_{i=1}^{N}\left(\log(\frac{\sigma^2_{i,pred}}{\sigma^2_{i,true}}) + \frac{\sigma^2_{i,true} + (\mu_{i,true} - \mu_{i,pred})^2}{2\sigma^2_{i,pred}} - \frac{1}{2}\right) \tag{3.3}$$

Alternative approaches to determine loss could be to maximize the overlap in p and q or to optimize for predicted μ and σ_{PDF} separately. The KL divergence is best to use as loss in this case, because by using this type of loss the shape of p and q are compared. The shape of the predicted PDF should resemble the 'actual' PDF. The 'actual' PDF is trained for with an arbitrary σ_{PDF}, thus should still be interpreted with care.

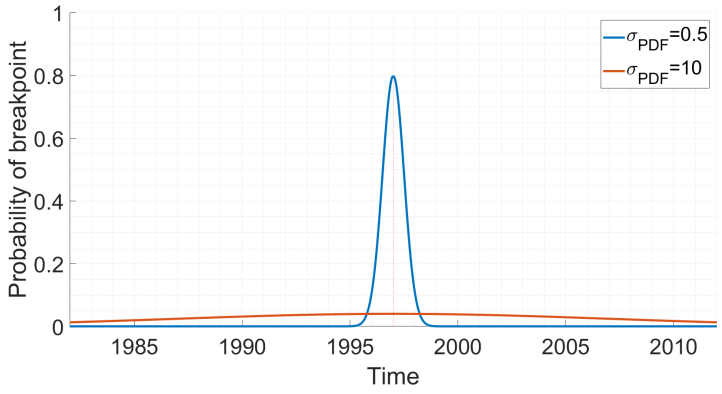


Figure 3.10: Example probability density function with and without breakpoint, the target to optimize for by from a model predicting 'Probability Density Function' breakpoint definition. μ is identical in both time series, σ_{PDF} is either 0.5 or 10. In the case of no breakpoint, for a large σ_{PDF} (yellow line), no peak is generated. In the case a breakpoint is present, for a small σ_{PDF} (blue line), it is clear where the breakpoint likely is. Note that in the y-axis the 'probability of breakpoint' is not guaranteed, as σ_{PDF} is set artificially.

Separate models (SM)

'Separate models' refers to using two separate models sequentially: first a model is trained to distinguish whether there is a breakpoint present, then a second model is used for the positives to find the breakpoint time. The first model is a binary classification problem, where output is either a 0 or 1, for no and a breakpoint, respectively. Hence BCE is used as loss function for the first model. To train for the breakpoint time, trained is with samples with a breakpoint only. For this second model, MSE loss is used. Training two separate models gives way to determine perhaps an optimal network strategy combining the benefits of both the MLP-model and the LSTM-model.

Summary all used breakpoint detection forms

All four mentioned breakpoint definitions to trained for are visualized in figure 3.11, to have an overview in how different breakpoint definitions stack up to one another. The `bfast01` methodology is also included for a comparison.

3.3.2. Network layouts and activation functions

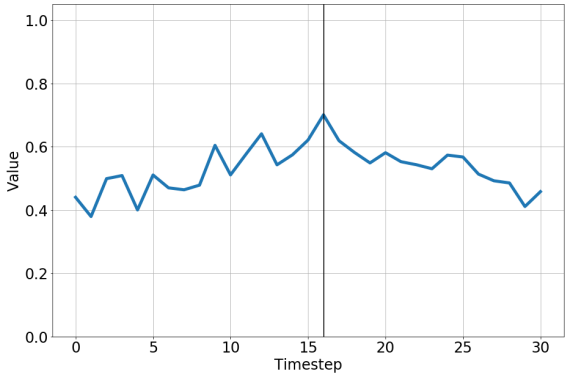
In this section is described how the two tested models are built. First their general layout is described, with the rationale behind it. Their internal 'wiring,' the connections between the layers in every model, are mentioned second. Then a short rundown is given of their expected performance on the breakpoint definitions introduced in Section 3.3.1. The description of their inherent strengths and weaknesses in breakpoint detection are mentioned last.

Multilayer perceptron-model (MLP)

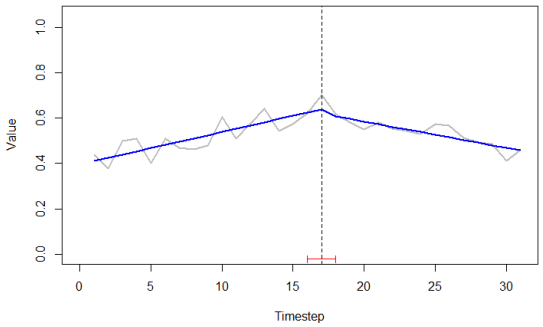
The two hidden layers of the MLP-model have 200 neurons a piece. The model is graphically represented in figure 3.12.

To avoid overfitting on the training data due to redundancy in neurons, the 'right' amount of neurons is empirically estimated. The width of the hidden layers is tested for in multiples of 50. These tests are performed with these multiples, because a rough estimate on the proper number of neurons is required. Every time a neural net is trained, it optimizes to slightly different weights due to the random initialization of the weights. It is thus not as useful to try to obtain the 'optimal' number of neurons per layers, as the variability in results caused by addition of single neurons can be captured in the variability caused by the neural net weights initialization. With less than 200 neurons per layer, accuracy on the test data dropped considerably, while a higher number of neurons marginally improves or reduces accuracy. Adding more depth also did not cause model accuracy to improve.

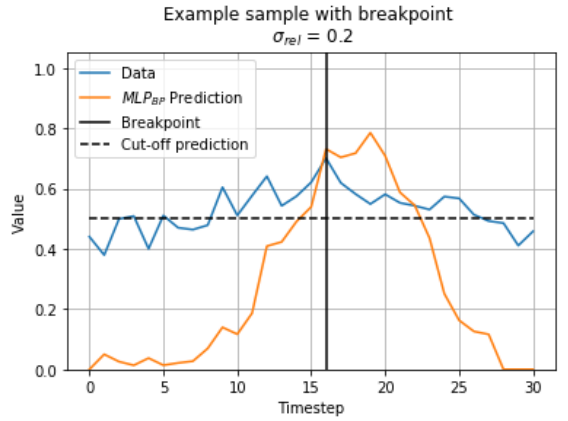
The hidden layers in the MLP-model have tanh activation functions, except for the output layer. The idea is that the first hidden layer is able create a derivative-like product from the input, where the second layer can combine this information to a second order derivative, where in turn the breakpoint parameters can be estimated from. In the binary classification exercises a maximum predicted value of 1 and minimum predicted value of 0 are required per timestep. Thus at the output layer a sigmoid activation function is



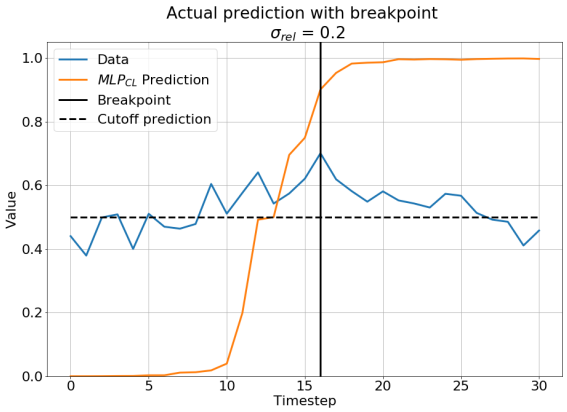
(a) The original signal



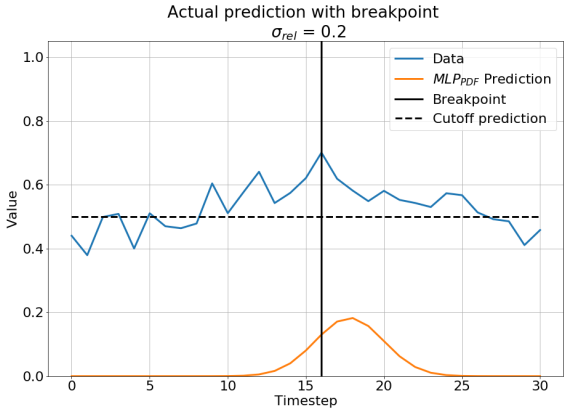
(b) `bfast01` breakpoint detection



(c) BP-type breakpoint detection



(d) CL-type breakpoint detection



(e) PDF-type breakpoint detection

Figure 3.11: All breakpoint detection methods visualized. The vertical black line indicates the true breakpoint time. In all cases, the prediction is interpreted differently. 'Separate Models' breakpoint definition is not visualized.

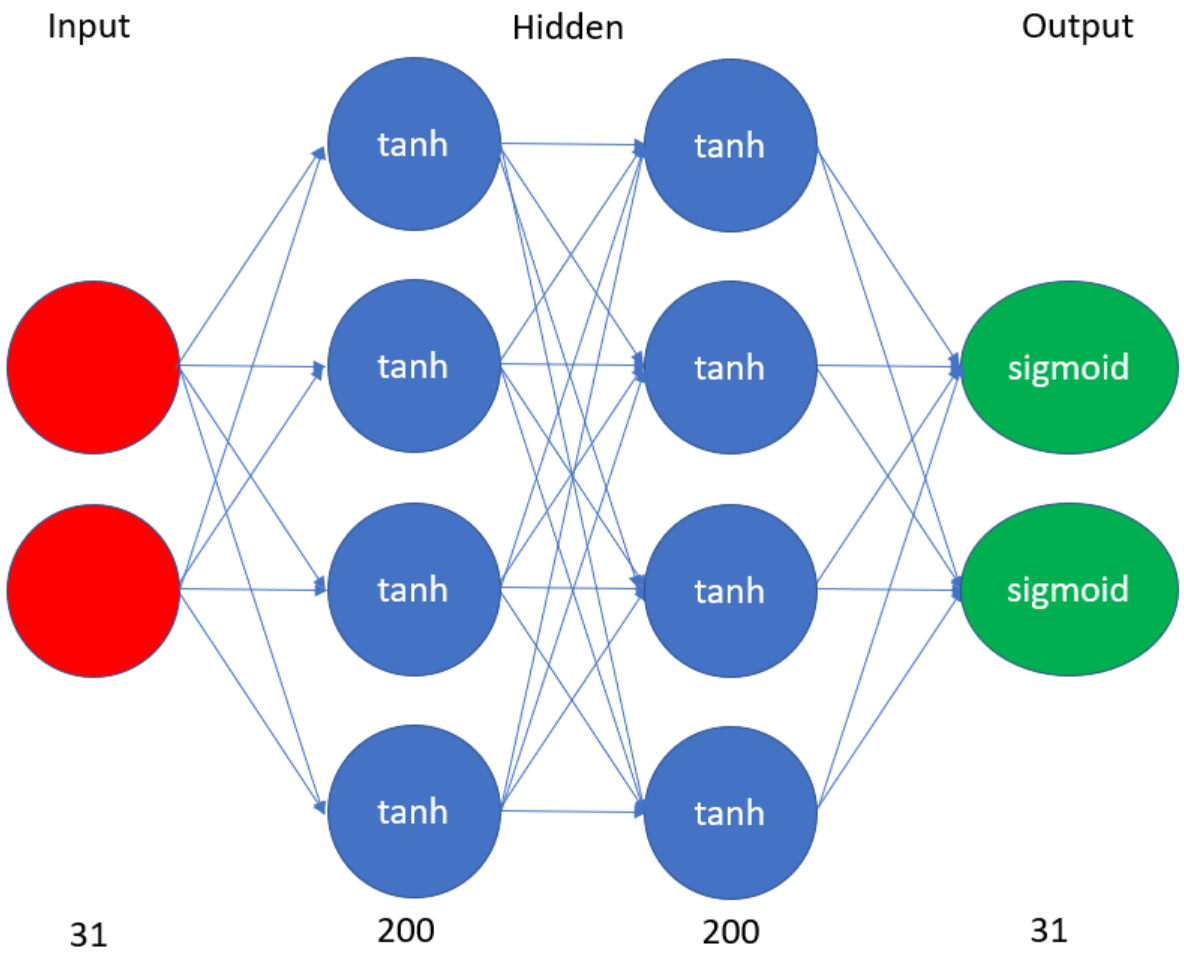


Figure 3.12: Graphical representation of the multilayer perceptron model used. All layers are fully connected, meaning all nodes of a layer are connected with all nodes from the previous and next layer. Activation functions are mentioned in this neurons. Above the layer the layer name is presented, below is the number of neurons within that layer. The output has either one, two or 31 nodes, depending on the breakpoint definition selected. Activation function for the output can also differ dependent on the breakpoint definition, see text.

chosen, which guarantees such an output. A positive output is necessary for the other types of loss, which is why the ReLU activation function is used at the output layer in those cases.

The MLP-model is suited to be coupled with all breakpoint definitions from Section 3.3.1). Because the activation function in all nodes introduce some sort of non-linearity, all types breakpoint definitions should provide a prediction as intended.

Advantage of the MLP-model is that all outputs one can come up with are possible to use. It is relatively simple type of network, making it easy for the layman to build. In this context of detecting breakpoints in time series, it is still possible to grasp layer functionality. A second advantage is that all information is incorporated in providing an output. By observing the whole time series at once, noise effects are likely not very impactful, as the model produce an output based on 'the bigger picture.' Disadvantage of the MLP-model is that no gaps in the data can be present. As all layers are fully connected, gaps in the dataset must be dealt with. In the context of a linear process, this is done by linear inter- or extrapolation, though any gaps in the data should be treated with care. Replacing them requires a thorough understanding of the background process in any user-case. Another major disadvantage are the parameters to optimize. With increasing time series length, or when incorporating a multiple of datasets, the amount of parameters to tune increase exponentially. When this becomes a problem in training or applying the model, suggested would be to use a multilayer convolutional neural net instead of the multilayer perceptron, see also appendix 10.1, to reduce the number of parameters to optimize. Another suggested solution for the 'curse of dimensionality' would be to limit the input of the MLP-model to a fixed length, after which parts of the time series are individually assessed for breakpoints.

LSTM-model

The LSTM-model consists of a single LSTM layer, connected to a single node. The single node function similarly as the nodes in a MLP. A LSTM layer is a recurrent neural layer, which applies its function to the input of the current timestep, as well as combining this with the state vector of the previous timestep. See also Chapter 2.2 for a more thorough description of the workings of a LSTM. Chosen is to use a LSTM over a simpler recurrent neural net type, because LSTMs can 'remember' information over longer times. This is beneficial, because before and after a breakpoint the signal changes behavior. To remember past behavior over more timesteps, the 'memory' of the LSTM is needed.

The desired output is made by combining the state vector of the LSTM layer into a single number, using said node. Depending on the defined loss, the LSTM layer provides either an output per timestep or for the sequence as a whole. Both these versions are graphically represented in figure 3.13. The state vector is of size 200, which is redundantly large. Note that all LSTM-models are trained bidirectionally, to prevent favoring accuracy in a single direction.

The 'Class label' (see Section 3.3.1) breakpoint definition is the only case where an output per timestep is required, as no good weighting could be distinguished for the 'Breakpoint label' breakpoint definition. Hence, for the 'Class label' and the first part of the 'Separate models' case, the output node has a sigmoid activation function, where it otherwise has a ReLU activation function.

The reason no 'Breakpoint label' could be assigned, is the instantaneous nature of the breakpoint in this particular case. When evaluating the time series, the LSTM-model passes through the time series one step at a time. It has to assign a label based on the input of the current timestep evaluated. This means that in this case in the 'eyes' of a LSTM-model, it has to evaluate each timestep whether a breakpoint is on the outer 'end' of the time series. The breakpoint signal is still non-existent at this moment, it is 'overshadowed' by the variability due to noise, thus the breakpoint definition 'Breakpoint label' is theoretically nigh impossible to assign for the LSTM-model.

A breakpoint can be present close to the start or end of the time series, for which the LSTM-model might defect. In the case of 'Class labels,' for example, it must be able to instantly decide whether or not a breakpoint is present. For this reason, the predicted value is likely close to 0.5 in the case of 'Class labels,' which can cause many transitions from labelling a section as 'before' or 'after' the breakpoint. For the other breakpoint definitions, where single outputs are concerned for whole sequences, the LSTM-model should be able to optimize correctly. The state vector redundant size provides ample channels to 'remember' features observed in the past, which can be combined to come up with a correct prediciton.

Advantage of the LSTM-model is the sequential nature of the model, as it deals with an arbitrary input length and any gaps in the data pose no problem. At the same time, disadvantage of the LSTM-model is the sequential nature of the model. Training and application times are much larger then its MLP-model counterpart. These computational speeds are much lower, because a LSTM-model needs to be applied in sequential fashion. Another disadvantage this sequential processing is, compared to the MLP-model, that not the whole time series can be evaluated at once.

3.4. Neural net and bfast01 settings

Some specifications on how bfast01 and the neural nets are used and made, respectively, are required to reproduce this thesis. bfast01 is used similarly to Negri Bernardino et al. 2018: The formula for the regression function is set to only look for the linear trend: "response~trend". This means the formula to be fitted by bfast01 reduces to equation 3.4.

$$y_t = \alpha_1 + \alpha_2 t + +\epsilon_t \tag{3.4}$$

The generated synthetic dataset follows equation 3.4 exactly, which is why bfast01 should perform optimally on that dataset. An important parameter to set in bfast01 is the bandwidth in the MOSUM test. Optimal bandwidth for the MOSUM test is found to be half the time series length. This is determined by viewing accuracy for bfast01 for various bandwidth on the synthetic dataset. Figure 3.14 shows this. With more noise, bfast01 has worse accuracy. Accuracy drops least for bandwidths roughly half the time series length, which is 31.

To create the neural nets, Keras library version 2.2.4 is used. No adjustments in learning rate or other hyperparameters are made. An Adam optimizer is used in all cases (Kingma and Ba, 2014). This optimizer is a well-known, widely used algorithm to adjust the weights in the training, the back-propagation process. All neural nets described are trained on a 50000 sample synthetic dataset with a relative noise up to 0.2. This training data is fed to the algorithms in sets of 10000 in order of increasing noise, to let them optimize best

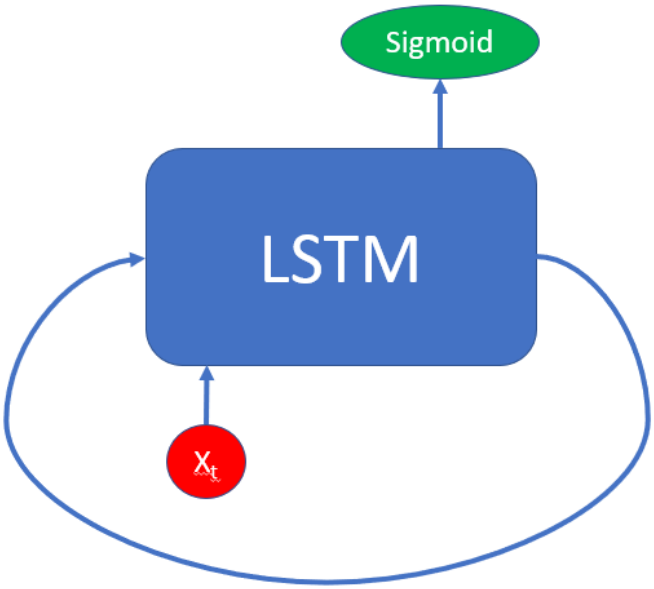


Figure 3.13: Graphical representation of the LSTM-model used. The small circle above the LSTM unit resembles a single node with its activation function. Each LSTM cell is identical, as well as the connection to the output. Every input (X_t) represents the value of a single timestep. Dependent on the breakpoint definition choice, output might be given per timestep or after assessing a whole time series.

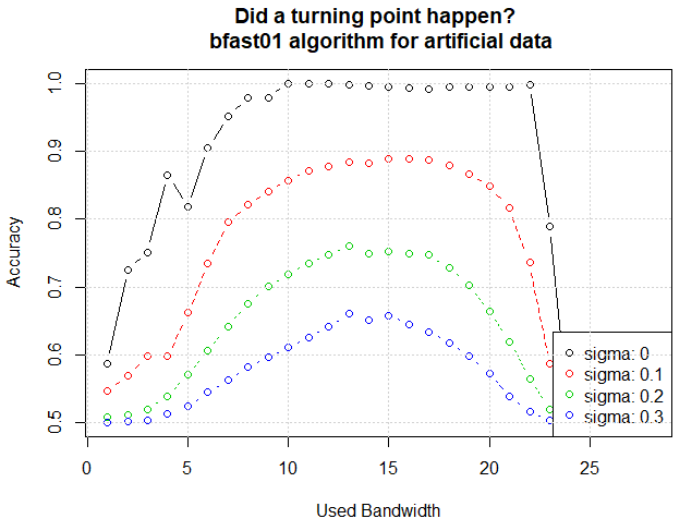


Figure 3.14: Accuracy `bfast01` for different bandwidths used and for data with various noise levels. The time series are of length 31. 'sigma' indicates the noise level in the time series used. Noise of the form $N(\mu = 0, \sigma = sigma * R)$ is used, with R the range of a time series. With increasing noise, accuracy of `bfast01` drops considerably. Breakpoint detection is most unreliable with small and high bandwidths. Accuracy drops least with noise and is highest for bandwidths roughly half the total time series.

for RUE data. The reason this works best, is because by fitting the weights and biases first on the noisefree data, overfitting is reduced significantly (Neelakantan et al. 2015). In other words, all weights and biases, the 'valves and settings' within the neural nets, get to be initialized on the perfect condition, after which slight changes can be made to deal with increasing noise levels.

Batch size is set to 16 and number of epochs is 10 for the MLP-models. A rough estimation on the proper amount of epochs is made by checking the loss during training of a validation subset. If the validation loss increased and loss still decreased, training is aborted to avoid overfitting. The batch size for the MLP-models is chosen for no particular reason, as it affected performance of the MLP-models noticeably only when changing the batch size in orders of magnitude. Training of the MLP-models takes less than a minute in all cases. Number of epochs and batch size are both set to 4 for the LSTM-models. The number of epochs is chosen in a similar way as described above. The batch size is kept relatively small, as otherwise the LSTM-models often optimized for the 'best guess.' This means that to learn properly, not too many time series should be used in the backprop process at once. Training times for the LSTM-models are significantly longer: training took approximately four to five minutes. Batch size and epochs is set fixed for the two model types, because this gives a level 'playing field' when comparing the neural net models with one another. Chapter 4 continues with this comparison.

4

Results

First on the menu is a performance comparison of all mentioned model architectures from Chapter 3.3 and `bfast01`. These comparisons are in measures accuracy and precision in breakpoint detection, which are defined in Chapter 2.1. Distinguished is which models perform best in this context. Some time series with corresponding model predictions are shown, to grasp better how they work. A more thorough analysis of the best performing models is then presented.

4.1. Accuracy, false positive and false negative ratio

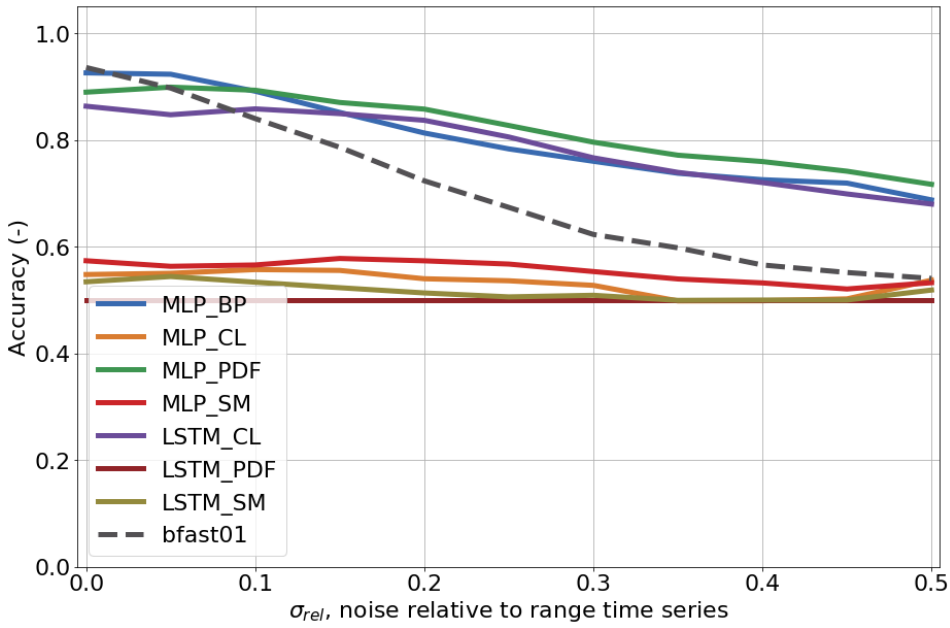
Accuracy, false positive and false negative ratios per relative noise level for all tested models are are shown in figure 4.1. From figure 4.1a it is clear that some models outperform `bfast01` in accuracy. MLP_{BP}, MLP_{PDF} and LSTM_{CL} to be specific. All models show decreasing accuracy with increasing relative noise, which is to be expected. No model correctly detects all breakpoints on noise free data. This is likely resulting from the lack of constraints for the data used: Time series with breakpoints can resemble time series having no breakpoint by random chance. Peculiar is that most models do not have their optimal accuracy when the data is noise free. This is likely an artefact from the training order (see 3.4), as all neural net models are optimized for the last data they are fed. By comparing figures 4.1b and 4.1c, concluded can be that this not-optimal accuracy at $\sigma_{\text{rel}} = 0$ is caused by false negatives. What this means, is that the neural net models function similar to a statistical test of sorts, as a certain threshold for change must be observed before labelling a breakpoint. If the neural nets are trained in order of decreasing noise levels, most of the models would have similarly shaped false positive and false negative curves as `bfast01`.

With increasing noise, false positive ratios increase more than their false negative counterpart, except for `bfast01`. More noise causes the neural nets to indicate more breakpoints, which can indicate that the neural nets are generally more sensitive to detect breakpoints than `bfast01`. For `bfast01`, the detected number of breakpoints drops with increasing noise. The false positive and false negative ratios of MLP_{BP} and MLP_{PDF} are parallel, which might indicate that the functioning of their hidden layers behave similarly. This is explained by their similar loss functions, as MLP_{BP} is optimized to provides something looking similar to a probability density function, the function MLP_{PDF} should optimized for.

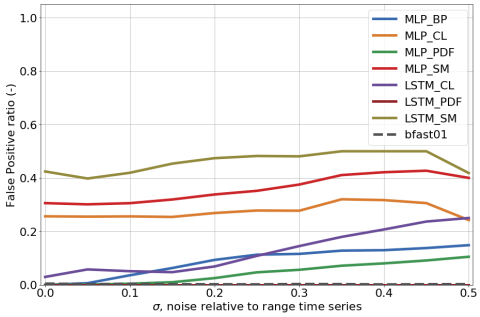
Striking is the lower accuracy of all other models. MLP_{CL} gets outperformed by its LSTM counterpart by a large margin. This indicates that the labelling of two sections in a time series is done much better by a sequential model than a MLP. This might be inferred by the MLP being less restricted to predict separate sections of '1's or '0's, as it does not take sequential relations in mind necessarily. The separate models might be dysfunctional because of the loss function used. By simply predicting a '0' or '1' for a whole sequence, the models are likely having a hard time recognizing the feature to optimize for. Increasing the number of epochs during training did not improve performance of these models. In RUE time series, relative noise ranges from 0.05 to 0.40. Based on accuracy it can be said that MLP_{BP}, MLP_{PDF} and LSTM_{CL} outperform all other models in accuracy, especially for the relative noise range observed in RUE time series.

4.2. Precision comparison

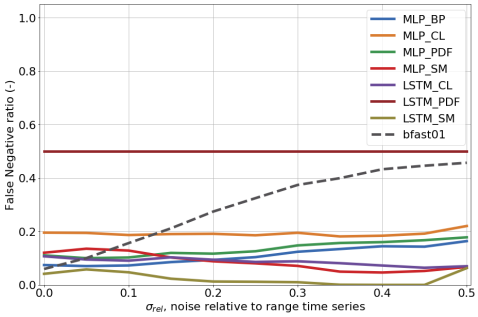
The standard deviation of the predicted breakpoints from the true breakpoints for all models is presented in figure 4.2a. `bfast01` detects much less breakpoints correctly than some of the neural nets, which is why



(a) Accuracy all tested models

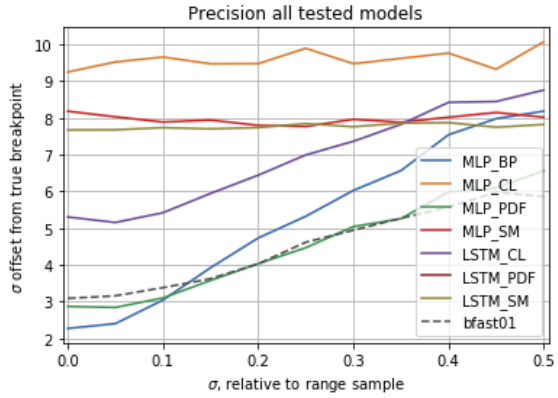


(b) False positive ratio all tested models

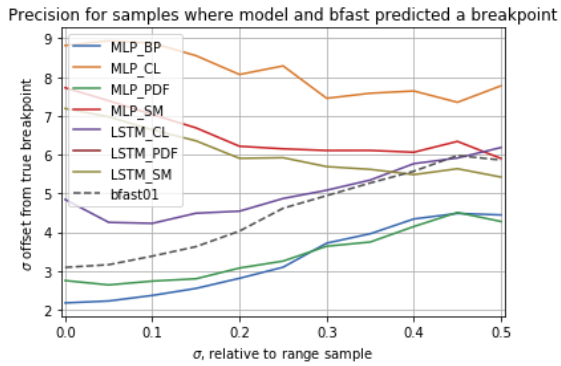


(c) False negative ratio all tested models

Figure 4.1: Accuracy, false positive and false negative ratio for all tested models. On the x-axis the relative noise is expressed, on the y-axis the ratio is represented. Good performing models show high accuracy and low false positive and false negative ratios. `bfast01` performs relatively well for low relative noise, though false negative ratio increases rapidly with increasing noise. MLP_{CL}, MLP_{SM}, LSTM_{BP}, LSTM_{PDF} and LSTM_{SM} perform worst in accuracy by a significant margin. The best performing models on the basis of accuracy are MLP_{BP}, MLP_{PDF} and LSTM_{CL}. Notable is that their false negative ratios increase only slightly with more noise, whereas their false positive ratio increases with more noise.

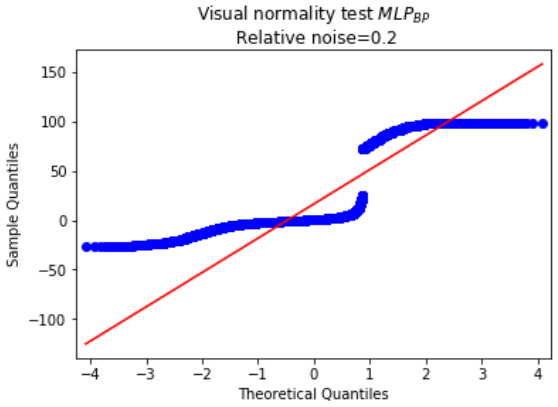


(a) Precision for all tested models, for all true positives

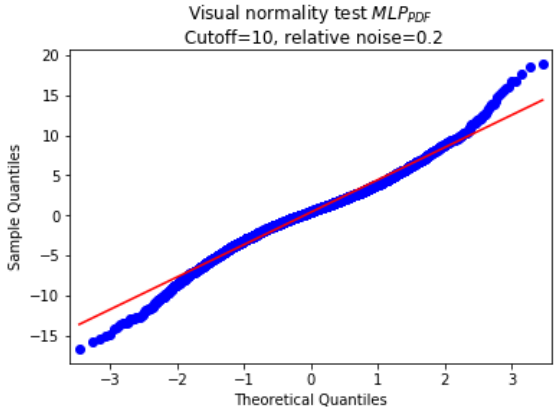


(b) Precision of all samples where both `bfast01` and neural nets correctly predicted a breakpoint

Figure 4.2: Overall precision and precision when compared to `bfast01`. Low σ_{std} means good model performance. The well performing models in terms of accuracy (MLP_{BP}, MLP_{PDF} and $LSTM_{CL}$) show worse precision with increasing noise. Only MLP_{PDF} shows similar precision as `bfast01`. If comparing only the samples where both neural nets and `bfast01` predict a breakpoint, it is observed in the lower σ_{std}, that MLP_{BP} and MLP_{PDF} have considerably better precision than `bfast01`.



(a) Quantile-quantile plot precision MLP_{BP}



(b) Quantile-quantile plot precision MLP_{PDF}

Figure 4.3: Visual normality tests of precision for two models. Based on these quantile-quantile plots, it is concluded that the offset between true and predicted breakpoint of both MLP_{BP} and MLP_{PDF} are not normally distributed. This means that the precision per noise level shown in figure 4.2a should not directly be translated to a certainty bound for detected breakpoints.

another comparison is made: the σ_{std} for all samples where both a neural net and `bfast01` detected a breakpoint correctly. This is visualized in figure 4.2b.

Note that the standard deviations presented in figure 4.2 are **not** Gaussian, as the offsets from the truth are not normally distributed. This is visually checked in a quantile-quantile plot (figure 4.3) and rejections of the Shapiro-Wilk test (not presented). This means that predicted σ_{std} should be not be directly linked with a certainty bound.

The in accuracy well-performing models MLP_{BP}, MLP_{PDF} and $LSTM_{CL}$ also show relatively good precision. What stands out is that `bfast01` together with MLP_{PDF} got similar and the best precision out of all models. When taking in mind that some neural nets detect a lot more breakpoints in total (see figure 4.1a) than `bfast01`, it can be said that MLP_{PDF} shows best performance in both precision ánd accuracy. This also indicates that in time series where `bfast01` detects a breakpoint, a higher degree of certainty in breakpoint detection can be expected from the neural nets. From figure 4.2 it seems that $LSTM_{CL}$ is less suited for breakpoint detection, as its precision is at best similar to `bfast01`. Because MLP_{BP}, MLP_{PDF} show very similar accuracy and precision, these models are selected for a more thorough analysis. $LSTM_{CL}$ is still selected, as the best performing LSTM-model, to show some predictions for. This is useful to judge why this model shows worse functioning than MLP_{BP} and MLP_{PDF}.

4.3. Inspection of neural net predictions

To investigate how the different models function it is useful to check some individual predictions. The predictions of the neural nets MLP_{BP}, MLP_{PDF} and $LSTM_{CL}$ are the main focus. These models perform best and at least both a MLP and LSTM-model is presented. Chosen is to show three comparisons in particular:

1. A noiseless time series with breakpoint, where `bfast01` does not detect a breakpoint
2. A noisy time series, where neural nets provided the breakpoint correctly
3. A noisy time series without breakpoint, where neural nets provided a breakpoint

The first comparison is shown in figure 4.4g. It can be observed that all neural nets provide some prediction of a breakpoint, where `bfast01` does not. The nature of the breakpoint, where a change in regression structure is in opposite direction to the change in mean, is the reason `bfast01` is insensitive in this case. MLP_{CL} does not work as intended, the lack of bounds to its prediction is the reason for this. $LSTM_{CL}$ provides a label by sequentially going through the data, which is observable in figure 4.4f. After the breakpoint is passed, the prediction shoots up. However, after few timesteps it 'reconsiders;' the predicted value goes down again. Reason is likely the order in training, which is mentioned in Section 4.1. The model is optimized to recognize breakpoints in noisier situations. The prediction goes up at the breakpoint, because a change in behavior is detected. After some timesteps, this change does not look systemic, thus the prediction goes down again. This means the LSTM-model is optimized to recognize a line segment, instead of a sudden change. MLP_{BP} and MLP_{PDF} both give predictions which are similar to expectations. Both these models provide a reasonable breakpoint time prediction, with a good certainty there is a breakpoint present.

The second example selected shows the neural nets' ability to detect breakpoints in very noisy data. `bfast01` is unable to recognize any breakpoint signal from the data. MLP_{BP} can notice something happened, though the breakpoint time is not correctly predicted. Both MLP_{PDF} and $LSTM_{CL}$ give remarkably good predictions in this case: they are just one timestep off-target, with high confidence.

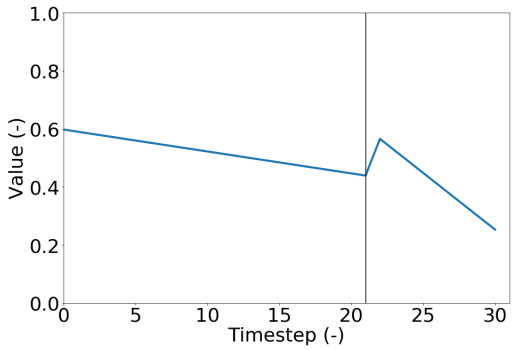
The last example is to show when the neural nets are wrong, see figure 4.6. The neural nets detect breakpoints where there are none, some with more confidence than others. This goes to show that while some of the neural nets outperform `bfast01` in terms of accuracy and precision, an indication of the certainty of different models is necessary for their actual use. Based on the examples depicted, $LSTM_{CL}$ shows a very comparable quality in breakpoint detection as its MLP counterparts. The choice made in Section 3.3.1 to label something a breakpoint based on the last predicted element, might be an erroneous choice. However, no good alternative is present. Predictions like the one in shown in figure 4.4f, where segments labelled as '0' and '1' are separated, are plentiful, causing difficulty interpreting results.

4.4. MLP_{BP} stability analysis

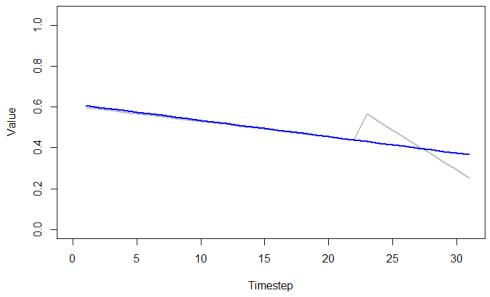
MLP_{BP} is a multilayer perceptron, that predicts a breakpoint by labelling a time series as '1' (>0.5) where there is a breakpoint. As laid out in paragraph 3.3.1, the maximum predicted value can be used to determine the certainty in breakpoint detection. The distribution of all predictions confirms this, which is presented in figure 4.7.

Figure 4.7 shows the MLP_{BP} distribution of all correct and incorrectly classified breakpoints, for all noise levels. Two clear peaks are observed in the distribution: at lowest prediction values and at highest predicted values. At around a prediction of 0.5, the 'true' and 'false' histograms meet. This means that at the boundary of what is considered a breakpoint, the model has a 50% chance of being correct, mimicking a coin toss or a random guess. The model does not know whether a breakpoint is present or not. That this place is at 0.5, is to be expected, which shows the model works as intended. The predicted values and the breakpoint signal are compared, shown in figure 4.8a.

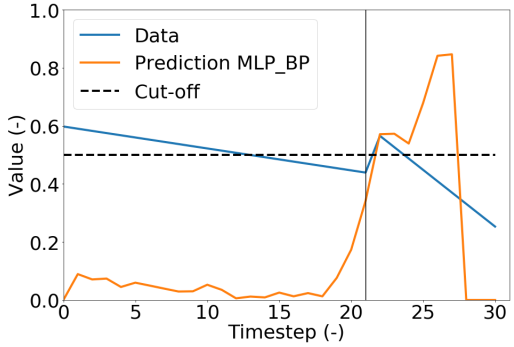
Figure 4.8a indicates that the model indeed predicts a higher maximum value for the more 'obvious' or clear breakpoints. The inverse is also true, generally; With a very small breakpoint signal, less breakpoints are detected (prediction<0.5). This is conform expectations and can explain the low and constant false negative ratio for this model. The cases where there is negligible breakpoint signal, the models do not recognize the breakpoint. These time series are present at all noise levels, in similar number per noise level. The certainty bounds of `bfast01` are also compared with the maximum predicted value in figure 4.8b. From this figure, it can be concluded that where `bfast01` has a low uncertainty bound, where `bfast01` has high certainty of a detected breakpoint, MLP_{BP} predicts nigh only 'maximum' values (see figure 4.7). This indicates that where `bfast01` detects a breakpoint with 'good' certainty, the MLP_{BP} indicates a breakpoint with 'very high'



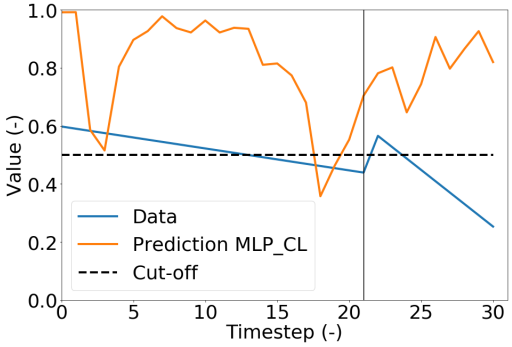
(a) The original signal



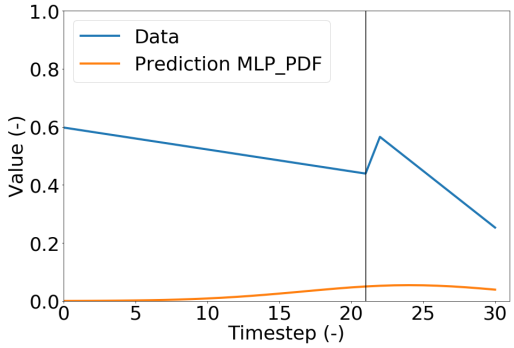
(b) `bfast01`



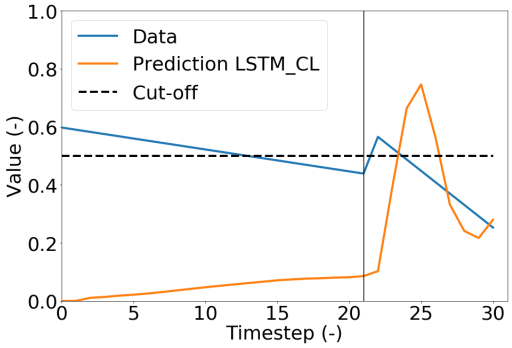
(c) MLP_{BP}



(d) MLP_{CL}

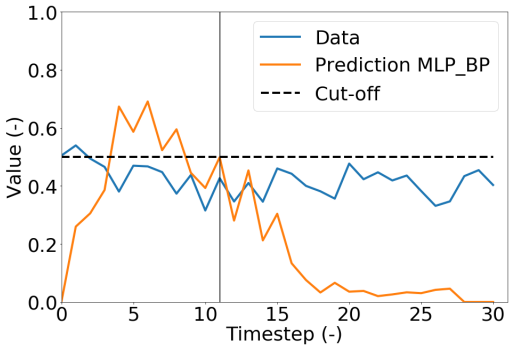


(e) MLP_{PDF}

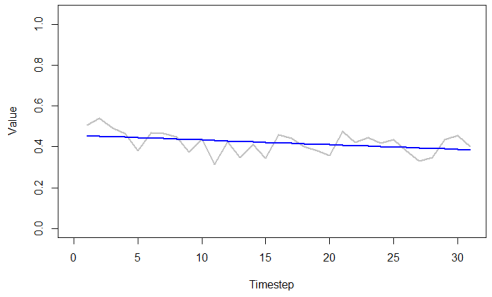


(f) $LSTM_{CL}$

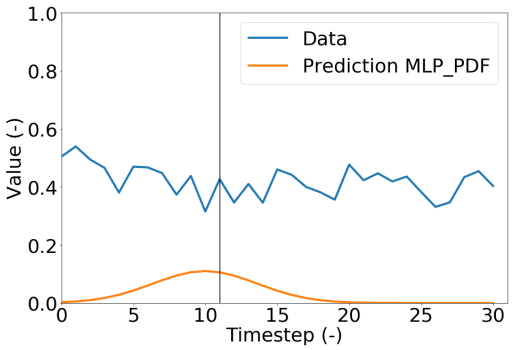
(g) Example predictions for a noise free signal. Black lines indicate the breakpoint, dotted lines the value above which a prediction is considered detecting a breakpoint. bfast01 does not recognize the breakpoint, because it is close to the end of the time series. The change in regression structure and change in mean also contradict each other, which causes problems in testing the null hypothesis. The neural nets generally have difficulty in determining the correct breakpoint time. MLP_{CL} is included to show the contrast with $LSTM_{CL}$, of which the latter provides a prediction in the expected form. MLP_{CL} is less constricted to provide predictions of the desired form, which can explain its unstructural behavior. Due to its sequential nature, $LSTM_{CL}$ can be 'read' from left to right: it detects the breakpoint when it passes, increasing predictions at after that timestep, but when considering the later timesteps gets more 'uncertain' whether a breakpoint actually occurred, lowering the prediction. MLP_{PDF} predicts a σ_{PDF} of 7.6, which means a breakpoint is detected, see also Section 4.5.



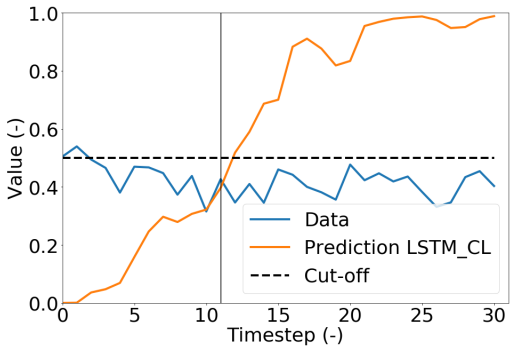
(a) MLP_{BP}



(b) `bfast01`

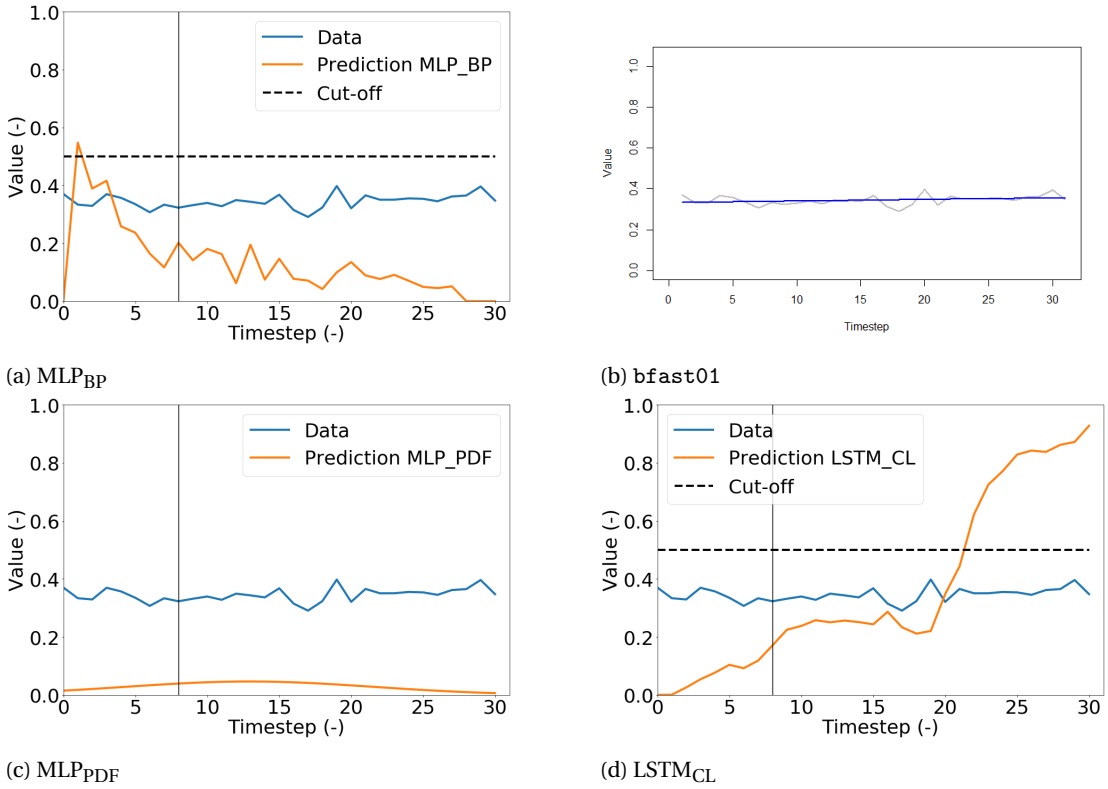


(c) MLP_{PDF}



(d) LSTM_{CL}

Figure 4.5: Example correct predictions for noisy data. Black lines indicate the breakpoint, dotted lines the value above which a prediction is considered detecting a breakpoint. σ_{rel} is maximal: 0.5. `bfast01` is unable to recognize anything in the signal, where MLP_{PDF} and LSTM_{CL} give a remarkably accurate prediction. σ_{PDF} is 3.6, indicating high confidence in the detection of a breakpoint, see also Section 4.5. MLP_{BP} notices a change occurred, but the allocated breakpoint time is quite far from the true breakpoint time



(a) MLP_{BP}

(b) `bfast01`

(c) MLP_{PDF}

(d) $LSTM_{CL}$

Figure 4.6: Example erroneous predictions for noisy data. No breakpoint is present, the dotted lines are the value above which a prediction is considered detecting a breakpoint. σ_{rel} is maximal: 0.5. `bfast01` is unable to recognize anything in the signal. MLP_{PDF} detects a breakpoint: σ_{PDF} is 9.2, indicating very low confidence in the detection of a breakpoint, see also Section 4.5. MLP_{BP} notices a change occurred, but with extremely low confidence at a particularly wrong time, see also Section 4.4. $LSTM_{CL}$ seems fairly confident in a detected change, though for all the wrong reasons. These features indicate high sensitivity of all models to possible breakpoints.

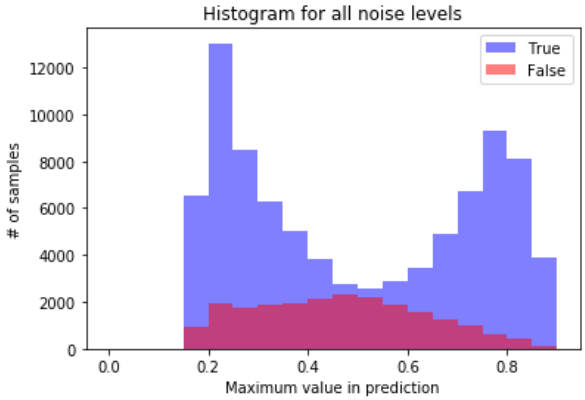
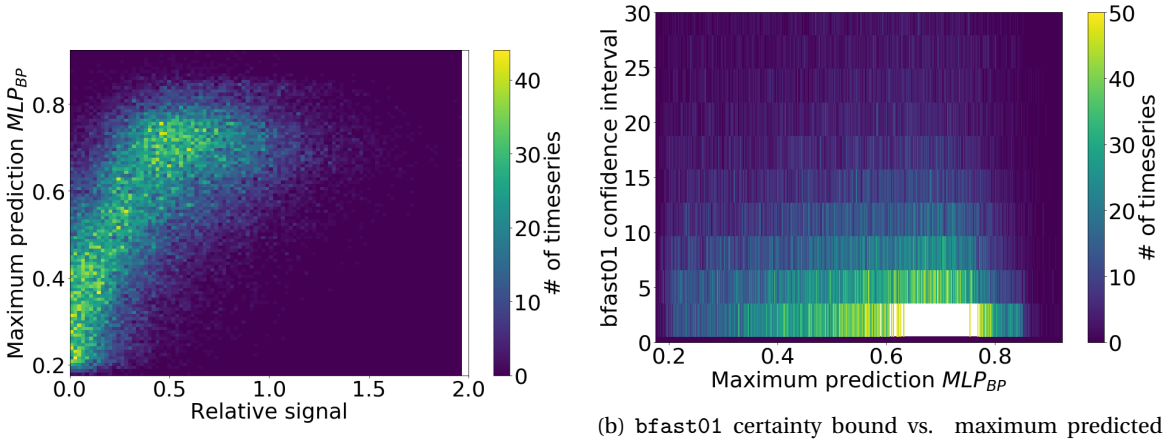


Figure 4.7: Distribution MLP_{BP}. Note that for a prediction above 0.5, a breakpoint is considered detected. 'True' indicates true positives, as well as true negatives. 'False' indicates false positives, as well as false negatives. From this distribution, it is clear that a lower or higher predicted maximum value correlates with a greater degree of certainty of no breakpoint or a breakpoint detection, respectively. At the transition of what is considered a breakpoint, a maximum predicted value of 0.5, MLP_{BP} has roughly 50% chance of being correct, which is an intuitive outcome. This shows that the model works as intended.



(a) Relative signal vs. maximum prediction

(b) `bfast01` certainty bound vs. maximum predicted value

Figure 4.8: MLP_{BP} maximum prediction compared to breakpoint signal and `bfast01` confidence interval. Relative signal is the dimensionless breakpoint signal defined in Section 2.1. For a relative breakpoint signal close to 0, a lower value is generally predicted. This indicates that the maximum predicted value can be used as a certainty measure, because its value is higher for signals with 'clear' breakpoints. The comparison with `bfast01` shows that where `bfast01` has a low confidence interval (is 'certain' of a breakpoint), MLP_{BP} has a very high maximum predicted value (is 'very certain' of a breakpoint).

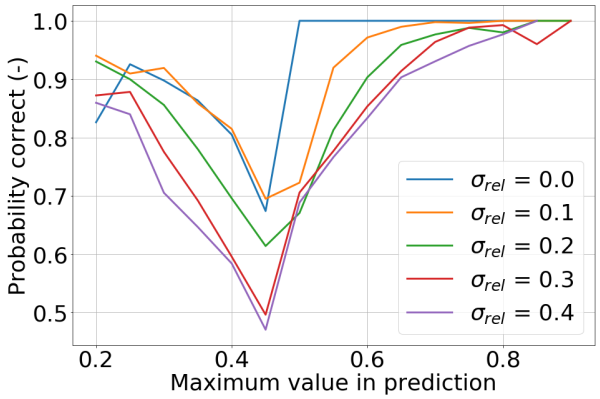


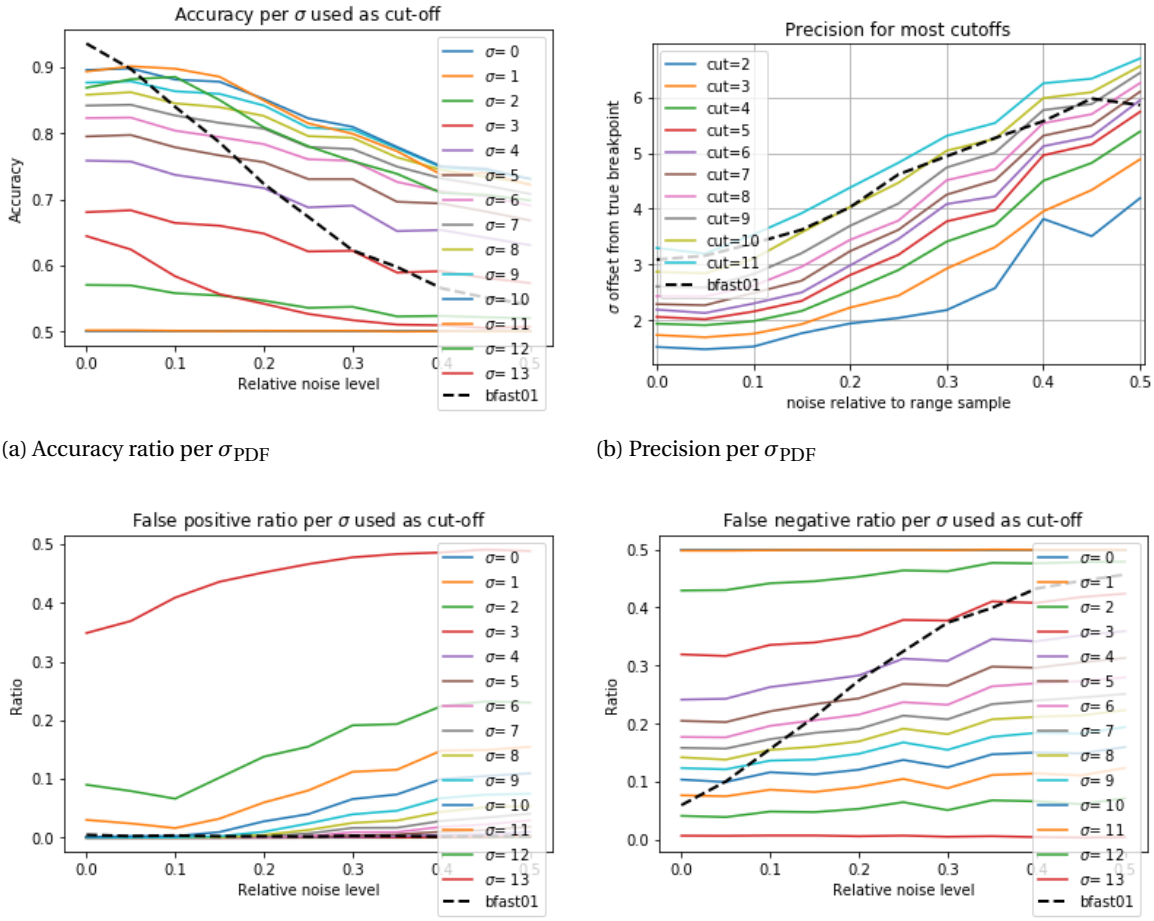
Figure 4.9: Likelihood of a trustworthy MLP_{BP} breakpoint detection, in relation to the maximum predicted value. A maximum predicted value above 0.5 indicates a breakpoint has been detected. In noise free data (σ_{rel}=0), no false negatives are present, which is indicated by a perfect reliability of the model above a maximum predicted value of 0.5. Generally, reliability is higher for lower and higher maximum predicted values, which indicates that the maximum predicted value indeed can be used as a performance indicator.

certainty, confirming this expectation from Section 4.2. This also helps explain the precision trends observed earlier in figure 4.2, where MLP_{BP} showed similar precision as `bfast01` in cases both these models detected a breakpoint.

By use of histograms of 'maximum predicted value' per relative noise levels, breakpoint detection certainty can be estimated per relative noise level, resulting in figure 4.9. From figure 4.9 it is shown that certainty in the (un)detected breakpoints is lowest at around a maximum predicted value of 0.5. At a maximum predicted value of 0.5 the 'True' and 'False' parts of the graph meet. This behavior holds for all relative noise levels present in the data. With increasing noise levels, a higher degree of uncertainty is observed as all lines 'move' downward. These graphs show that a degree of certainty can be given to a detected breakpoint based on the maximum value in the predicted time series. Only above a relative noise level of 0.3 does reliability of MLP_{BP} drop significantly at most common maximum predicted values 0.2 and 0.9. This means the model likely performs well on the user-case next chapter, the RUE time series.

4.5. MLP_{PDF} stability analysis

A major advantage of MLP_{PDF} is that the breakpoint time and a certainty measure are automatically provided in μ and σ_{PDF}, respectively. However, as σ_{PDF} is set to arbitrary values of 0.5 and 10 for time series with and without breakpoint, respectively. The range in predicted σ_{PDF} is from 2 to 13, which is outside of the



(a) Accuracy ratio per σ_{PDF}

(b) Precision per σ_{PDF}

(c) False positive ratio per σ_{PDF}

(d) False negative ratio per σ_{PDF}

Figure 4.10: Accuracy, precision, false positive and false negative ratios for every cutoff for model MLP$_{PDF}$. From these figures it is clear that σ_{PDF} is a strong indicator of model performance. If cutoff σ_{PDF} is selected more 'strict,' selected lower, precision improves and false positive ratio drops. With higher σ_{PDF}, false positive ratio increases, but false negative ratio drops. Note that for every cutoff used, false negative ratio does not vary much with noise observed in the time series.

provided range in training. The optimal cutoff is checked by testing all round numbers from minimum to the maximum σ_{PDF} and looking at accuracy and precision behaviors. Accuracy, precision, false positive ratio and false negative ratio for all cutoffs are shown in figure 4.10.

It is quickly observed in figure 4.10 that the sensitivity of MLP$_{PDF}$ is highly dependent on the cutoff used to distinguish whether breakpoints are detected or not. With a higher σ_{PDF} selected as cutoff, more samples will be considered a breakpoint, thus a larger cutoff increases false positive ratio significantly. The inverse is also true for false negative ratios, as less time series containing no breakpoint are labelled as having one. Observed in figure 4.10c is that the false positive ratios for MLP$_{PDF}$ are extremely low for low noise levels for most selected cutoffs; for a cutoff lower than σ_{PDF}=8, false positive ratio is below 0.05 for noise levels below σ_{rel}=0.35. As set out in paragraph 3.4, the cutoff for the optimal accuracy should be selected for, which in this case is a σ_{PDF} of 10. That the predicted σ_{PDF} is a direct indication of quality of the predicted breakpoint, is visible in figure 4.10b: With increasing cutoff, precision drops. By observing the noise in a time series, expected model performance can be deduced by viewing predicted σ_{PDF}.

Another concern is the tendency of MLP$_{PDF}$ to predict breakpoints in the middle of the time series. With a higher and higher cutoff, the predicted breakpoint time becomes more and more the 'average.' This tendency is visualized for all samples where MLP$_{PDF}$ predicted a σ_{PDF} over the cutoff of 10 in figure 4.11. What this means, is that for every sample where there is no breakpoint detected, the 'average' prediction is given. This is a logical result of randomizing the breakpoint in cases there is no breakpoint signal to be detected. To check whether the model has a bias towards the middle of a given time series, the histograms in predicted μ for two different cutoffs are given in figure 4.12. It is observed that for lower cutoff σ_{PDF}, the model has a

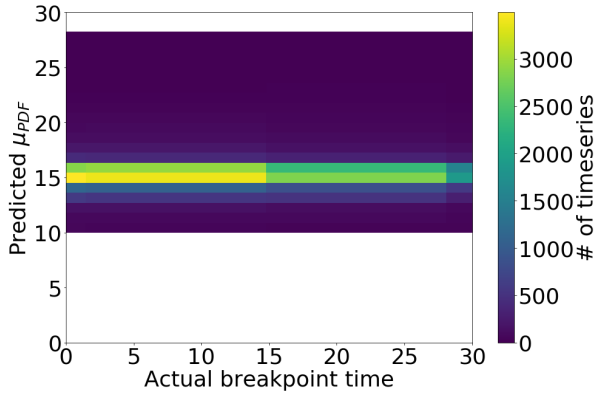
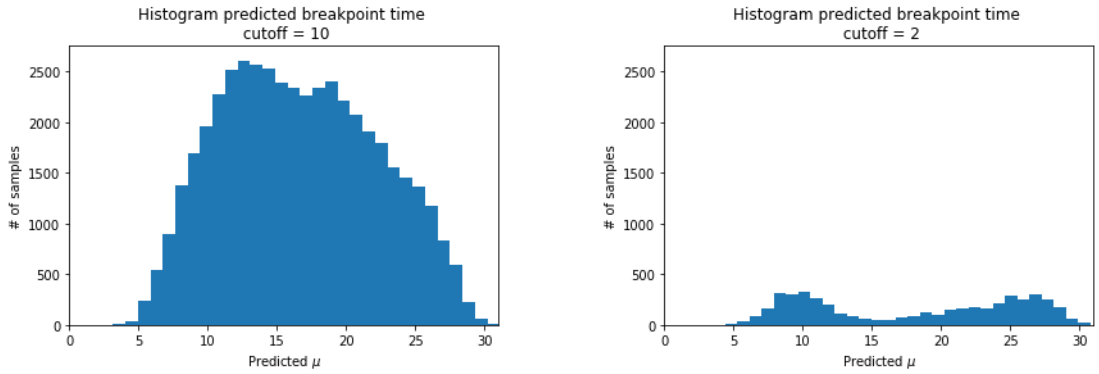


Figure 4.11: Predicted versus true breakpoint for samples where MLP_{PDF} detected no breakpoint (cutoff=10). This graph indicates that in cases there is no breakpoint detected, MLP_{PDF} gives the 'average' prediction, meaning a bias is present in the predicted breakpoint times towards the middle timestep. This is a training artefact of attributing random breakpoint times to time series without breakpoint.



(a) Distribution predicted μ by MLP_{PDF} for cutoff = 10

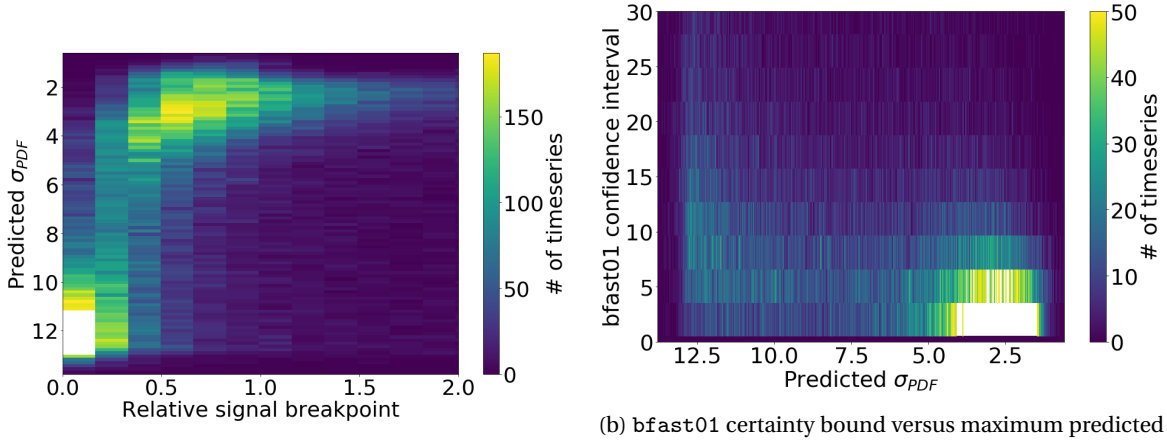
(b) Distribution predicted μ by MLP_{PDF} for cutoff = 2

Figure 4.12: Distribution predicted μ, MLP_{PDF} for two cutoffs on synthetic data. The peaks present when using a lower cutoff indicate there is a correlation between predicted μ and σ_{PDF}. When the model is 'very certain' of a breakpoint (low σ_{PDF}), it is more likely to predict a breakpoint close to the ends of the time series. The reason why this happens is unclear.

bias away from the middle, while the inverse is true for higher σ_{PDF} as cutoff. The strong correlation between predicted σ_{PDF} and μ is present because of unknown reasons. They should be uncorrelated. One explanation is that they are both abstractions from the same layer. As they have similar input values, they might behave similarly. Another explanation might be that σ_{PDF} eventually dominated the optimizing process, as μ was provided randomly for half of the samples.

By comparing the predicted σ_{PDF} with the breakpoint signal and the `bfast01` confidence interval, see figure 4.13, an eerily similar pattern is observed as for the MLP_{BP} model, see figure 4.8. This is a strong indicator that the hideen layers within both the MLP_{BP} and the MLP_{PDF} model function similarly. The same conclusions can be drawn from figure 4.13 as figure 4.8: Where `bfast01` has a high certainty of a detected breakpoint, MLP_{PDF} predicts high σ_{PDF}. This indicates that where `bfast01` detects a breakpoint with 'good' certainty, the MLP_{PDF} neural net indicates a breakpoint with 'very high' certainty.

What this also means, is that there should be a considerable overlap between `bfast01` with both MLP_{BP} and MLP_{PDF} predictions for a user-case. It is time to introduce rain-use-efficiency into the story, in Chapter 5.



(a) Relative signal versus maximum prediction

(b) `bfast01` certainty bound versus maximum predicted value

Figure 4.13: MLP$_{\text{PDF}}$ prediction σ_{PDF} compared to breakpoint signal and `bfast01` confidence interval. Relative signal is the dimensionless breakpoint signal defined in Section 2.1. For a relative breakpoint signal close to 0, a higher σ_{PDF} is generally predicted. This indicates that the σ_{PDF} can indeed be used as a certainty measure, because its value is lower for signals with 'clear' breakpoints. The comparison with `bfast01` shows that where `bfast01` has a low confidence interval (is 'certain' of a breakpoint), MLP$_{\text{PDF}}$ has a very low σ_{PDF} (is 'very certain' of a breakpoint).

5

Rain-use-efficiency

Next to synthetic data, the generated neural net models and `bfast01` should be compared in a user-case. Chosen is to use rain-use-efficiency (RUE), because it has a few beneficial qualities set forth in the introduction: It behaves linearly over time (Negri Bernardino et al., 2018) and the time series are constructed using remotely sensed data. This gives way to a lot of time series to be analyzed, it is a large dataset. The `bfast` package is commonly used in desertification research (Burrell et al. 2017, 2018; de Jong et al. 2012; Negri Bernardino et al. 2018), giving way to compare results of various researches. The chapter will continue as follows. First, the relevance of RUE in dryland assessment is briefly discussed. Second and third are the description of the study area and RUE data collection, respectively. The application of `bfast01` and the neural nets MLP_{BP} and MLP_{PDF} to the RUE time series follows. Some individual results are then picked and compared. The chapter is closed by a brief synthesis of the predicted breakpoints in RUE time series. These results are briefly discussed in this chapter, their implications to change detection more broadly and the comparison to earlier results from Chapter 4 are discussed more thoroughly in the next chapter, in Chapter 6.

5.1. The context of rain-use-efficiency

Assessing land degradation in drylands is important, as 40% of the world's landmass accounts for this biome and over a third of the world population live in such ecosystems (Negri Bernardino et al. 2018; UN 2010), see figure 5.1. An abrupt change, a breakpoint, in ecosystem functioning (EF) in drylands can have severe consequences (Foley et al., 2005), as poverty and food scarcity might increase following the breakpoint (Berdugo et al., 2017). "Ecosystem functioning" refers to "some state or trajectory of the system under consideration and to the sum of those processes that sustain the system" (Jax, 2005). Estimating breakpoints in EF for the drylands of the world can thus provide useful insights in resource allocation (Hillier and Dempsey 2012).

Rain-use-efficiency (RUE), first coined by LeHouerou, 1984, is a measure for assessing land degradation in arid/semi-arid areas (Fensholt and Rasmussen 2011). RUE is a surrogate for ecosystem functioning in these

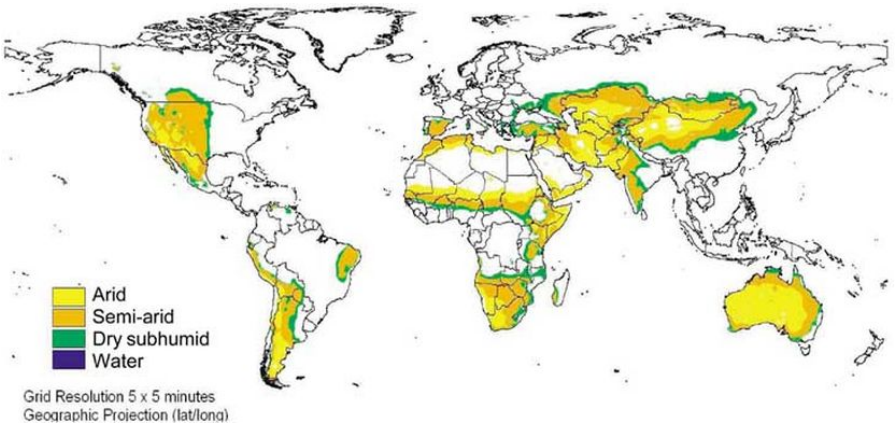


Figure 5.1: Global dryland distribution ('Atom'). Drylands cover over 40% of the world's landmass.

drylands and is defined by dividing Net Primary Production (NPP) by precipitation (Fensholt et al. 2013; Negri Bernardino et al. 2018), see the middle part of equation 5.1. NPP is a quantity describing the energy input into an ecosystem, proposed as a variable to describe the whole of ecosystem functioning (Sala and Austin, 2000). The definition of RUE normalizes NPP for inter-annual rainfall variability, which is important. Conventional RS-derived NPP proxy indicators, such as NDVI time series (Tucker, 1978), do not take precipitation patterns into account, which makes them susceptible to simply reflecting inter-annual rainfall variability instead of true EF.

$$RUE = \frac{NPP}{Precipitation} \approx \frac{\sum NDVI_{\text{Growing season}}}{\sum Precipitation_{\text{Growing season}}} \tag{5.1}$$

As a practical proxy for NPP, the growing season NDVI small integral is used. The NDVI small integral is the integral over growing season NDVI from a base value (Eklundha and Jönssonb 2017; Jonsson and Eklundh 2002; Jönsson and Eklundh 2004). NDVI, or Normalized Difference Vegetation Index, is the normalized difference of two spectral bands (Tucker, 1978). The spectral bands are the red and infrared spectral bands, collected by various satellites. The spectral signatures of vegetation and bare ground are very different in these bands, which causes NDVI to be commonly used in ecosystem monitoring. Vegetation is easily distinguished from its surroundings by use of NDVI. RUE can thus be described as the growing season NDVI, normalized by the precipitation over the same perios. This statement is mathematically included as the last part of equation 5.1.

In earlier research, changes in RUE or NDVI time series are detected with the `bfast` algorithm (Burrell et al., 2017; de Jong et al., 2012; Verbesselt et al., 2010). `bfast01` detects only a single, major breakpoint. There are three reasons to use `bfast01` over `bfast` for RUE time series. First reason is the RUE time series provide a single value per year. RUE time series' time resolution is too coarse to reliably consider short-term processes, like forest fires and floods. These can, for example, be detected with bimonthly NDVI time series (Verbesselt et al. 2010). Second reason is the particular interest in long-term change, thus more appropriate is to identify a single "most important" break (De Jong et al. 2013). Third is the ability of `bfast01` to provide a typology of the detected change, whether change accelerated, decelerated, whether change is complete or transitional, etc. (De Jong et al., 2013; Negri Bernardino et al., 2018). This third attribute of `bfast01` is not utilized in this thesis, as the neural net models do not provide such a characterization.

5.2. Study area

One region is targeted to test `bfast01` and the neural net models: Australia. The target region is visualized in figure 5.2. Australia is selected for its vastness, as well as it being well documented in desertification research. The vastness of the landmass gives way to see patterns in breakpoint analyses. Breakpoint analyses in Australia using different statistical techniques on RUE and NDVI time series give very different breakpoint patterns over the region (Burrell et al. 2017; Negri Bernardino et al. 2018). Applying a machine learning approach to characterize breakpoints in the drylands of Australia can give good insights in how neural net analyses compare to other methods and models. It is also selected to limit the scope of this test, as analysis of the whole world would make interpretation and comparison of results unnecessary complex.

5.3. RUE data collection

RUE is derived from the same data and in similar fashion as defined by Negri Bernardino et al. 2018. NDVI time series from the third generation GIMMS NDVI from AVHRR sensors version 1 (NDVI3g.v1) are used to determine the NDVI small integral. This dataset provides bi-monthly data at 0.083° spatial resolution from January 1982 to December 2013. This timerange is slightly shorter than Negri Bernardino et al. 2018, because the data is extracted from Google Earth Engine, where the NDVI3g.v1 dataset is available up to 2013. The second half of 1981 is not used, to allow for easier usage of TIMESAT. This slight artificial shortening of the time series is assumed to not influence detection of breakpoints. TIMESAT 3.3 is the matlab package used to determine the NDVI small integral from the NDVI time series. A mask for areas with lower median NDVI value of 0.1 is used to remove areas with sparse vegetation (De Jong et al., 2013). For precipitation, the Climate Hazards Group InfraRed Precipitation with Station data (CHIRPS) dataset is used. The CHIRPS dataset provides five measurements per month, at a 0.05° spatial resolution. For ease of combining both the datasets, the maximum NDVI value per month is used and the CHIRPS dataset is summed per month and resampled to match NDVI3g.v1 spatial resolution using the nearest neighbor method. These datasets are practical to use in ecosystem research, because they have a good temporal range.

Figure 5.2: Selected study area to assess RUE time series for

Regions with double growing seasons are not included by considering two seperate NDVI small integrals. Their infrequent presence in the study area (less than 1% of over 100.000 pixels showed a double season) caused suspicion as to whether actually a double season is present anywhere in Australia. The start and end time of every season is given by TIMESAT, as well as the NDVI small integral. This integral is used in further computations, as compared to manually calculating the NDVI small integral using the dates provided by TIMESAT. This is consistent with Negri Bernardino et al. 2018. The precipitation sums are made by taking the precipitation sum over the same time period, plus some months before the growing season starts. This is done to take into account the water uptake of an area before the actual growing season, as delayed responses to precipitation might occur in dry ecosystems (Horion et al., 2013; Vicente-Serrano et al., 2013). The number of months added is 0 to 11, which is determined per pixel by taking the precipitation sum with highest correlation to the NDVI small integral. Gaps in the data are filled with a linear inter- or extrapolation of surrounding points, as the underlying processes is assumed to be linear. An example of a few RUE time series is presented in figure 5.3. In this example it is quickly observed that the four time series presented show high correlation, which helps visualize the assumption of high spatial correlation between close pixels. It also shows the typical values in which RUE is measured.

5.4. Expected breakpoint patterns

As `bfast01` detects only the most clear of breakpoints (see Chapter 4), expected is that neural nets predict breakpoints with high certainty where `bfast01` detects breakpoints. Thus it is expected that neural nets predict breakpoints in at least the areas where `bfast01` predicts them. If pockets of breakpoints are detected, highest certainty in found breakpoints should be observed in the middle, with certainty dissipating to edges of said pockets. In any given pocket of breakpoints, observed breakpoint time should be similar. Pixels close to one another are assumed to be highly correlated. This high expected spatial correlation is assumed based on inspection of raw RUE time series, for example the ones depicted in figure 5.3.

5.5. Breakpoints in Australian rain-use-efficiency

Breakpoints in RUE as predicted by MLP_{BP} and MLP_{PDF} are shown in figure 5.4 and figure 5.5, respectively. Their `bfast01` counterpart is shown in figure 5.6. In the centre of Australia both MLP-models predict similar breakpoint times, where results diverge everywhere else. Comparing figures 5.4, 5.5 and 5.6, it is evident that the MLP-models detect more breakpoints over all of Australia than `bfast01`. All models visually show very dissimilar breakpoint patterns over Australia.

5.5.1. MLP_{BP} predictions

The breakpoints in RUE time series detected by MLP_{BP} (figure 5.4a) show a curious pattern. Where in the middle of Australia later breakpoints are observed, closer to the coasts of Australia earlier breakpoints are found.

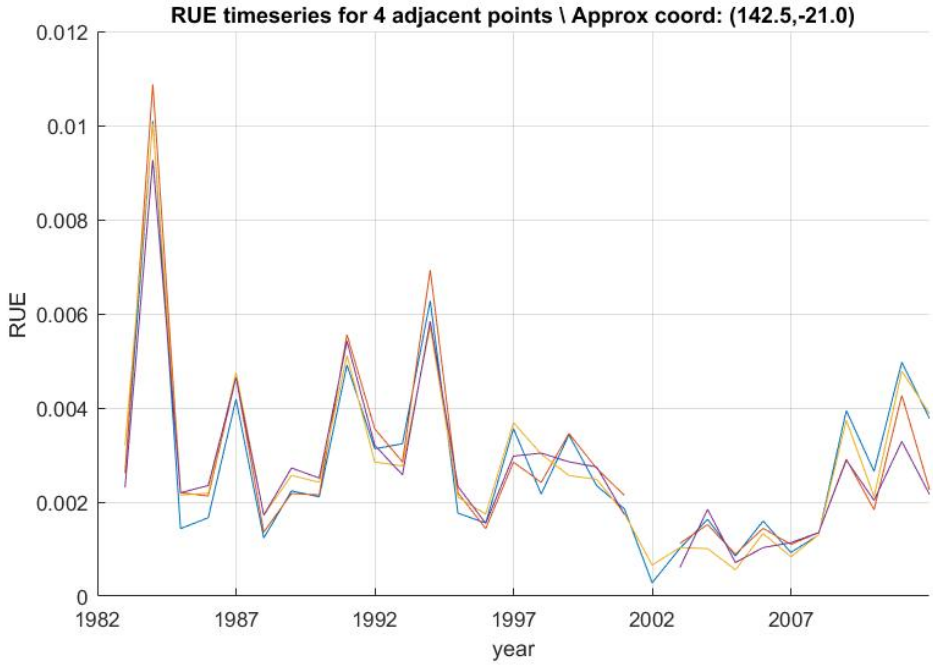
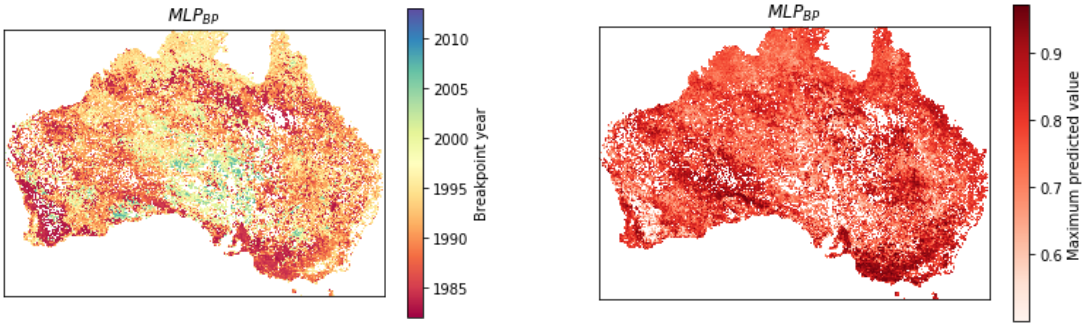


Figure 5.3: Four examples of RUE time series. They are from adjacent pixels, located in North West Australia. This graph shows that points spatially close to one another are highly correlated, as well as some of the typical values observed in RUE time series.



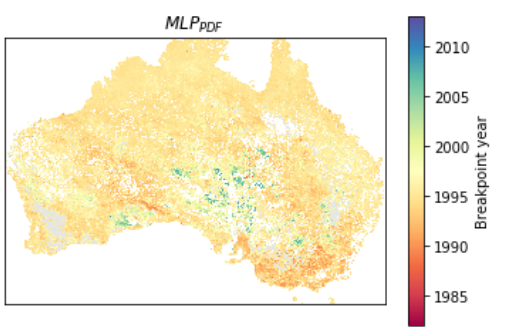
(a) Predicted breakpoint times in Australia RUE time series by MLP_{BP}

(b) Predicted maximum value for detected breakpoints by MLP_{PDF} in Australia RUE time series

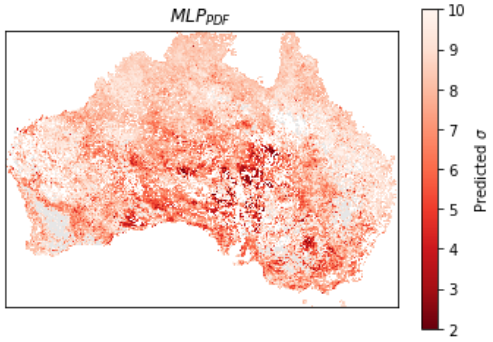
Figure 5.4: Predicted breakpoints in RUE time series in Australia by MLP_{BP}. In most pixels a breakpoint is detected. These breakpoints tend to be later in central Australia. Based on these images, there seems that there is a relation between predicted maximum value and predicted breakpoint year; where predicted maximum value is highest, the breakpoint year tends to be earlier.

Table 5.1: Number of breakpoints detected compared for different models

Model	**Breakpoints Australia**	**As % of total time series**
`bfast01`	3732	4.3%
MLP_{BP}	74659	86.4%
MLP_{PDF}	71269	82.5%

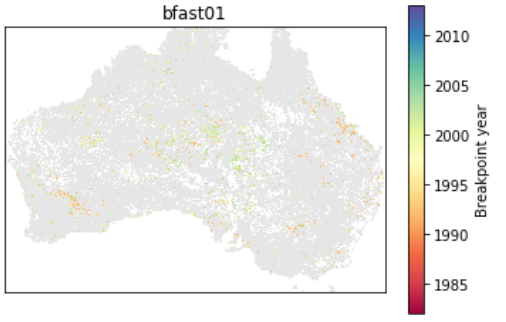


(a) Predicted breakpoint times in Australia RUE time series by MLP_{PDF}.

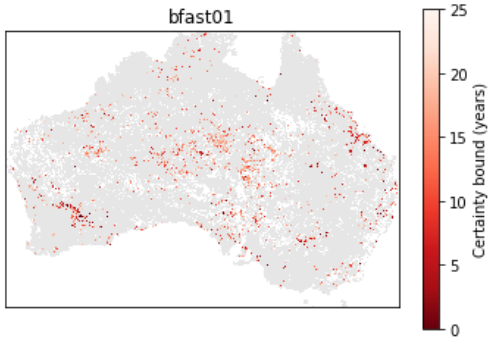


(b) Predicted σ_{PDF} for detected breakpoints by MLP_{PDF} in Australia RUE time series

Figure 5.5: Predicted breakpoints in RUE time series in Australia by MLP_{PDF}. In most pixels a breakpoint is detected. These breakpoints tend to be later in central Australia. Based on these images, there seems that there is a relation between predicted σ_{PDF} and predicted breakpoint year; where predicted σ_{PDF} is lowest, the breakpoint year tends to be later.

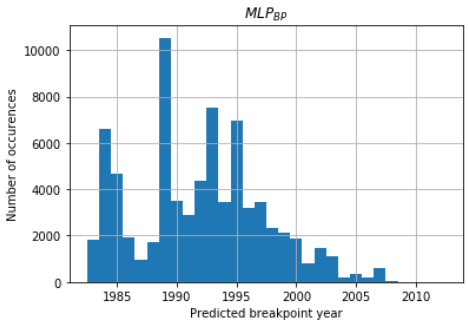


(a) Predicted breakpoint times in Australia RUE time series by `bfast01`.

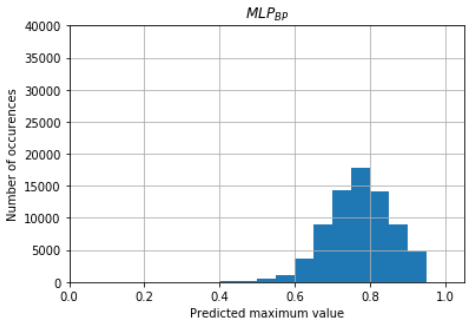


(b) Predicted certainty bounds for breakpoint detection by MLP_{PDF} in Australia RUE time series

Figure 5.6: Predicted breakpoints in RUE time series in Australia by `bfast01`. No clear patterns nor pockets of breakpoints can be observed.

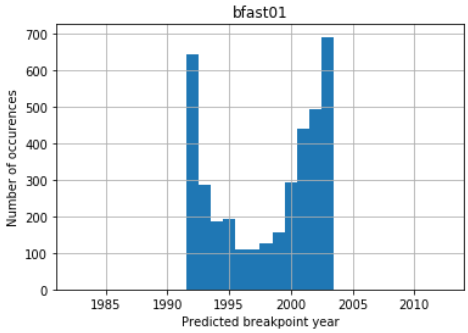


(a) Distribution MLP_{BP} predicted breakpoint time

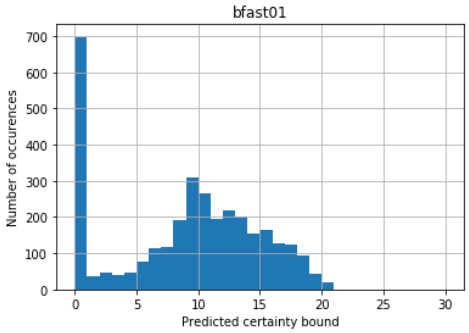


(b) Distribution MLP_{BP} predicted maximum value.

Figure 5.7: Histograms showing the predicted breakpoint year and the predicted breakpoint uncertainty for MLP_{PDF}. A maximum value above 0.5 indicates a breakpoint is present, thus almost all RUE time series in Australia are deemed to have a breakpoint according to this model. The relatively high predicted maximum values indicate a high certainty in detected breakpoints. The model does not seem to favor a particular breakpoint time.



(a) Distribution bfast01 predicted breakpoint time

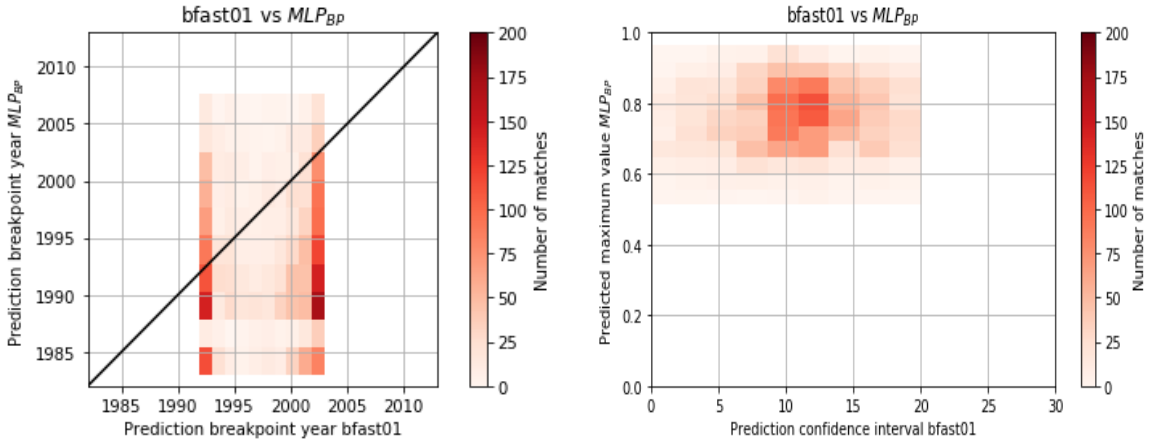


(b) Distribution bfast01 predicted certainty bound value.

Figure 5.8: Histograms showing the predicted breakpoint year and the predicted breakpoint uncertainty for bfast01. bfast01 does not predict breakpoints before 1993 or after 2003, which is an artefact of the chosen bandwidth in bfast01 application. The uncertainty bound is 0 in most cases, indicating high degrees of certainty for most detected breakpoints. This interpretation should be treated with care.

These earlier detected breakpoints coincide generally with a higher degree of detection certainty. There is clustering of breakpoint times in various regions, though in some Southern places there are abrupt changes in detected breakpoint time. The certainty of detected breakpoints is highest in the middle of pockets of breakpoints. These features indicate the model meets the standards set forth in Section 5.4 to some degree, with the expected breakpoint certainty patterns looking better than the breakpoint time patterns. Note that almost all RUE time series are predicted to contain breakpoints; 86.4% of all RUE time series are predicted to contain a breakpoint. This can either indicate MLP_{BP} is oversensitive to breakpoints or that Australia underwent drastic change in past decades. Oversensitivity to breakpoints can have various origins.

The distribution of detected breakpoint times and the maximum predicted values, shown in figure 5.7, indicate that MLP_{BP} generally predicts somewhat earlier breakpoints in the RUE time series. Figure 5.9 shows a comparison in breakpoint time and certainty for MLP_{BP} and `bfast01` in cases where both detected a breakpoint. No clear correlation in breakpoint time can be observed. This can mainly be explained by `bfast01` predicting by far the most breakpoints at around 1992 and 2003. `bfast01` is thus severely limited in which years it detects breakpoints by the chosen bandwidth, see also Section 3.4. Based on figure 5.8b, it again is observed that in cases where both bfast01 and MLP_{BP} detect breakpoints, MLP_{BP} is relatively 'certain' of the occurrence of breakpoints. Figure 5.8 shows the distribution of breakpoint time and certainties for `bfast`. A lot of 'highly certain' breakpoints, with uncertainty bound 0, cause suspicion. Reason for this suspicion is the overlap in detected breakpoints. The overlap in detected breakpoints for `bfast` and MLP_{BP} is almost every `bfast01` detected breakpoint, except for the points where `bfast01` provides a certainty bound 0.



(a) Predicted breakpoint times compared

(b) Predicted breakpoint certainty compared

Figure 5.9: `bfast01` and MLP_{BP} predicted breakpoint and uncertainties compared for pixels where both models predicted breakpoints. The black line indicates perfect similarity. For both the predicted breakpoint year and breakpoint detection certainty, very dissimilar results are produced. There is no correlation in predicted breakpoint years visible, though the same relation in certainty measure is observed as for the synthetic time series (Section 4.5).

5.5.2. MLP_{PDF} predictions

MLP_{PDF} detects breakpoints with highest certainty in the centre of Australia, where clusters of later breakpoints are predicted, similar to MLP_{BP}. These clusters are an indication of a well-performing model, as laid out in paragraph 5.4. There is an extreme difference in number of detected breakpoints between MLP_{PDF} and bfast01, which is summarized in table 5.1. The overlap is 3375 time series, or 90% of the total bfast01 detected breakpoints. This overlap is compared in figure 5.11. `bfast01` has a tendency to predict breakpoints at around either 1993 or 2003, where MLP_{PDF} predicts mainly breakpoints around 1995. The pattern in `bfast01` can be explained by the chosen bandwidth to apply `bfast01` with. The breakpoints around 1995 of MLP_{PDF} might be an artefact of the bias in the model. The homogeneity of detected breakpoint times can have as cause the bias described in Chapter 4, or a sudden change in the period around 1995. The strong correlation between predicted μ and σ_{PDF} found last chapter, indicates less certainty to the latter though. In figure 5.5 there again seems that there is a correlation present between μ and σ_{PDF}; where the breakpoint year gets predicted later, σ_{PDF} is lower, more certain. There is no clear correlation between MLP_{PDF} and `bfast01` predicted breakpoint times present. The same holds for the uncertainty measures between the two models. Some of the `bfast01` uncertainty bounds are negative or zero, which causes problems interpreting `bfast01` results. In figure 5.11b, a similar pattern as in figure 5.9b can be observed. The mediocre confidence interval of `bfast01` between roughly 5 and 15 coincides with the bulk of overlapping points. Biggest difference between figures 5.11b and 5.9b is that MLP_{PDF} also has overlap in many time series where the bfast01 certainty bound is 0. This indicates the similar internal workings to determine the maximum value in MLP_{BP} and σ_{PDF} in MLP_{PDF}, which was also observed in Chapter 4.

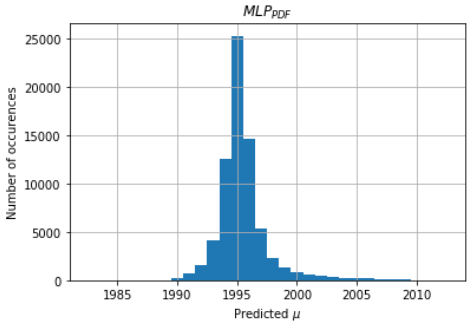
5.6. Individual comparisons

Some example time series with their respective predictions can be compared, giving extra insight into the functioning of the different algorithms. Note that interpretations are solely based on the time series as presented, ecological features of the time series or other contextual information is not taken into account. Visual interpretation is subjective. Three examples are picked, where:

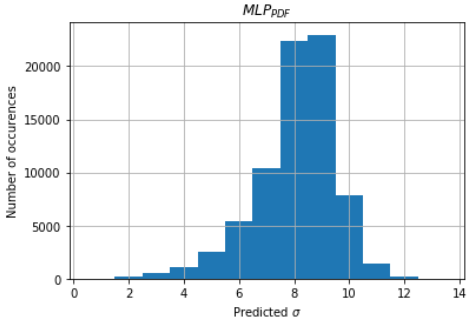
1. All models predict a breakpoint. Visually, one can be expected.
2. None of the models predict a breakpoint. Visually, none can be expected.
3. `bfast01` predicts no breakpoint, but the neural nets do. Visually, one can be expected.

All models predict a breakpoint

Figure 5.12 shows the prediction for one of the RUE time series. In the middle of the time series, a change in regression curve can be observed. `bfast01` predicts a breakpoint at this point. MLP_{BP} shows erratic be-

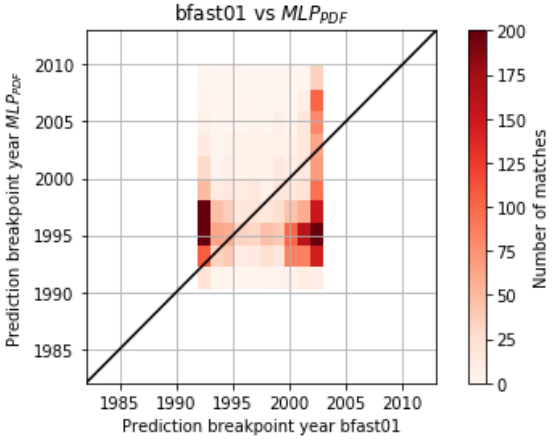


(a) Distribution MLP_{PDF} predicted breakpoint time

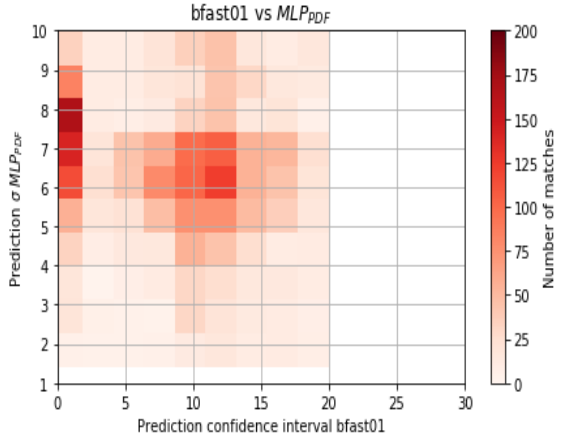


(b) Distribution MLP_{PDF} predicted σ_{PDF}

Figure 5.10: Histograms showing the predicted breakpoint year and the predicted breakpoint uncertainty for MLP_{PDF}. MLP_{PDF} favors predicting breakpoints in the middle of time series, which is again visible in the distribution. Predicted σ_{PDF} indicates MLP_{PDF} is not as certain of predicted breakpoints, because most detected breakpoints have a corresponding σ_{PDF} closer to 10.

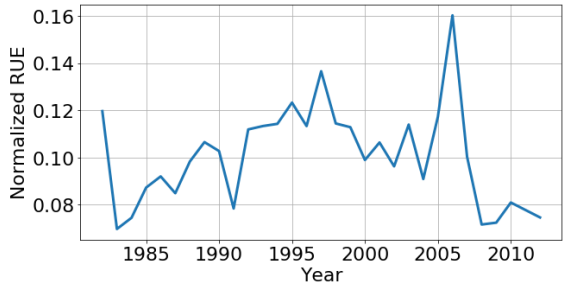


(a) Predicted breakpoint times compared

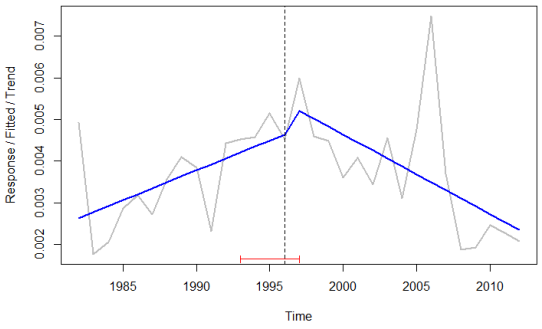


(b) Predicted certainty compared

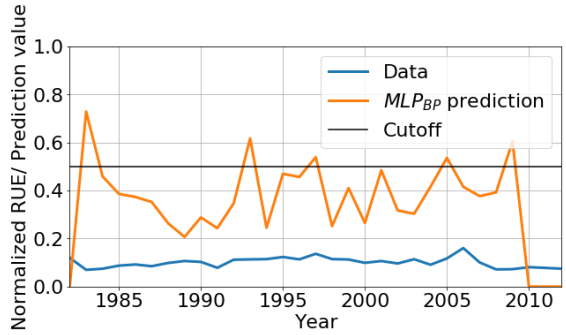
Figure 5.11: `bfast01` and MLP_{PDF} predicted breakpoint and uncertainties compared for pixels where both models predicted breakpoints. The black line indicates perfect similarity. For both the predicted breakpoint year and breakpoint detection certainty, very dissimilar results are produced. There is no correlation in predicted breakpoint years visible. The relation in certainty measure is also different from the relation observed for the synthetic time series (Section 4.5).



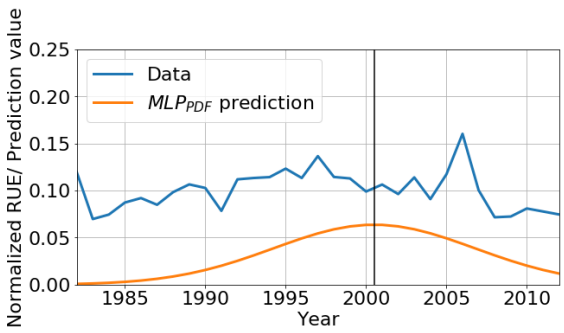
(a) The normalized RUE time series



(b) `bfast01` prediction



(c) MLP_{BP} prediction



(d) MLP_{PDF} prediction

Figure 5.12: The first selected example RUE time series and predicted breakpoints. For every graph, a different scale on the y-axis is present. Visually, a breakpoint could be present around 1996. All models predict a breakpoint, with `bfast01` predicting it exactly where it visually is expected. MLP_{BP} shows erratic behavior, causing trouble in interpreting where it actually predicts a breakpoint. MLP_{PDF} predicts the breakpoint between an outlier and the expected breakpoint. The vertical black line indicates the predicted breakpoint time.

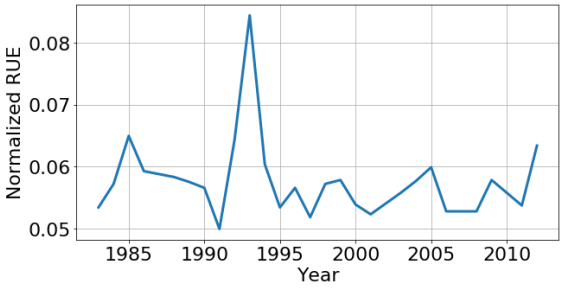
havior, which can be caused for a multitude of reasons. All places where a prediction larger than the cutoff is given, a peak or outlier is present in the time series. It likely 'sees' breakpoints in various parts of the time series, causing peaks in different places. MLP_{BP} is thus relatively sensitive to breakpoints. MLP_{PDF} predicts its breakpoint in between the the largest outlier and the middle, visually expected breakpoint. The placement can indicate that MLP_{PDF} gives a 'weighted average' of the breakpoints it detects, because it is forced to provide only a single breakpoint.

None of the models predict a breakpoint

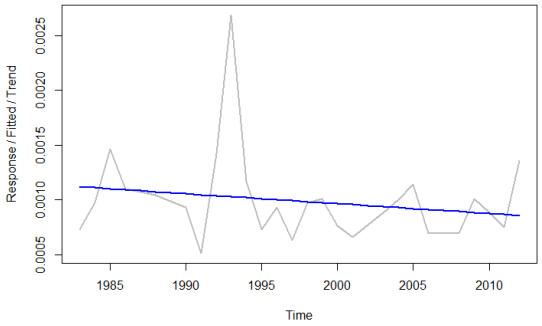
Figure 5.13 shows the prediction for one of the RUE time series. Around 1993, a large outlier is visible. No breakpoint detection method predicted a breakpoint. This shows that all methods generalize well, as an outlier does not necessarily forces a breakpoint to be predicted in any case.

The neural nets predict a breakpoint, where one is expected

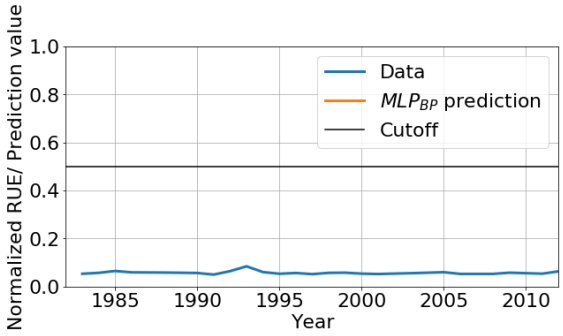
Figure 5.14 shows the prediction for one of the RUE time series. In the middle of the time series, a change in mean can be observed. `bfast01` predicts no breakpoint at this point, likely the result of the large variance in the time series. MLP_{BP} again shows erratic behavior; it provides essentially two breakpoints: one early on and one at the expected breakpoint time. No clear reason can be given why the early peak in prediction is present. A high predicted value at the start of the time series is also predicted in many cases where no breakpoint is predicted. This indicates that MLP_{BP} is slightly more sensitive to predicting breakpoints at the start of time series. A shape similar to a probability density function is not observed in figure 5.12c, nor in figure 5.14c. This indicates the model does not work as intended. MLP_{PDF} predicts its breakpoint in the middle, at the visually expected breakpoint. MLP_{PDF} seems to work as intended, as σ_{PDF} is higher in cases a breakpoint visually is expected. The breakpoint detection in figure **??** indicates that MLP_{PDF} is more sensitive than `bfast01` to possible breakpoints.



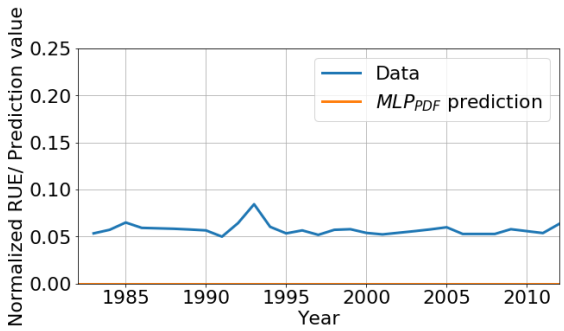
(a) The normalized RUE time series



(b) `bfast01` prediction

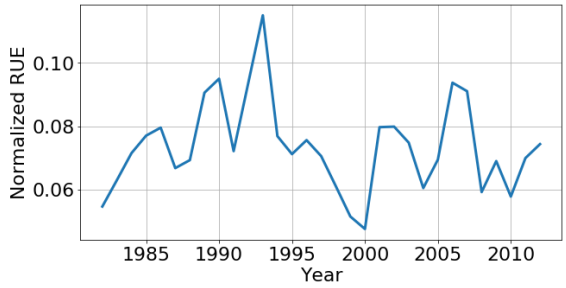


(c) MLP_{BP} prediction

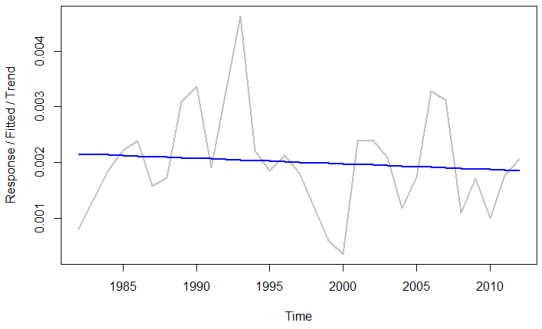


(d) MLP_{PDF} prediction

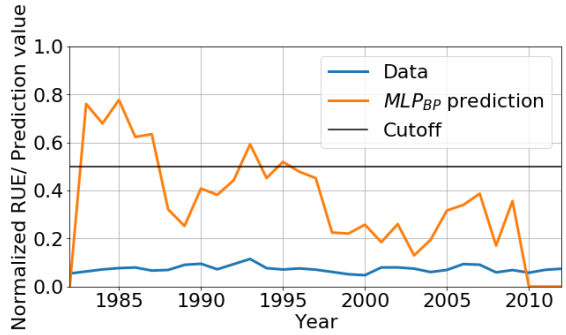
Figure 5.13: The second selected example RUE time series and predicted breakpoints. For every graph, a different scale on the y-axis is present. Visually, no breakpoint is expected. An outlier around the year 1993 can be observed. This indicates that all used models generalize quite well, as outliers do not necessarily corrupt the prediction.



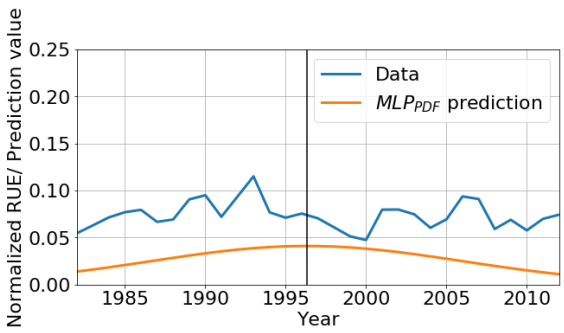
(a) The normalized RUE time series



(b) `bfast01` prediction

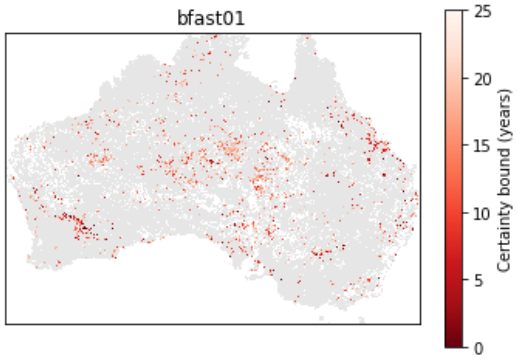


(c) MLP_{BP} prediction

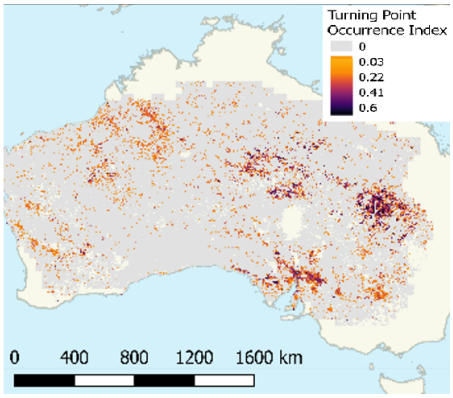


(d) MLP_{PDF} prediction

Figure 5.14: The third selected example RUE time series and predicted breakpoints. For every graph, a different scale on the y-axis is present. Visually, a breakpoint could be present around 1996. The regression structure seems the same, with a sharp change in mean. The neural nets predict a breakpoint. MLP_{BP} shows erratic behavior, causing trouble in interpreting where it actually predicts a breakpoint. MLP_{PDF} predicts the breakpoint at the visually expected breakpoint. The vertical black line indicates the predicted breakpoint time.



(a) Predicted breakpoint certainty bounds in Australia RUE time series by `bfast01`



(b) Turning Point Occurrence Index (TPOI) map. Darker colors represent areas with a higher certainty of breakpoint in ecosystem functioning occurrence and associated timing. From Negri Bernardino et al. 2018.

Figure 5.15: Australian RUE breakpoint patterns, determined by bfast01 in this and another study. The dissimilarities in these patterns raises questions in the quality of reproduced RUE time series. Main difference between the two studies are the bfast01 settings: In this study, a single bandwidth for the bfast MOSUM test is set, while Negri Bernardino et al., 2018 used all possible bandwidths to detect change points. The TPOI is then defined as: (Number of bandwidths detecting breakpoints) / (Total bandwidths tested) However, because the bandwidth in this study should be most sensitive to breakpoints, the patterns in both these images should be similar, while that is not the case.

5.7. Brief synthesis

The MLP methods detected many more breakpoints in Australian RUE time series than `bfast01`. This can indicate two things: Either Australia has changed much more significantly than previously expected or the MLP methods are much too sensitive to change. Assuming the latter, it can be the result of the sensitivity set during training or by having training data too dissimilar to the RUE time series. The limitations of all methods to detect the time of a breakpoint with high degrees of certainty causes concern for their real-world use. The real-world process of ecosystem change likely occurs non-linear or not abruptly, which is not taken into account in any method used in this thesis. This is a major concern in the analysis of RUE time series.

All methods showed limitations in the possible range of predicted breakpoint times. MLP_{BP}, less restrained to produce a single breakpoint, showed to be highly sensitive to breakpoints, though with an expected bias towards earlier breakpoints. In case a multiple of breakpoints could be present, MLP_{PDF} seems to average between the options detected.

The RUE time series used in this research are likely to vary from the ones determined by Negri Bernardino et al., 2018. This expectation is drawn from the dissimilarity in Australian breakpoint patterns observed in figure 5.6a and in Negri Bernardino et al., 2018. This is depicted in figure 5.15. Where in Negri Bernardino et al., 2018 two major pockets in breakpoints are found, one in South Australia and one in East Australia. In this research, none such pockets of breakpoints is detected. The detected breakpoints are much more spread out over the continent. Note that in both cases the occurrence of a breakpoint is determined differently. Negri Bernardino et al., 2018 used all possible bandwidths in the MOSUM test in `bfast01` to determine a 'certainty' measure of detected breakpoints. However, the empirically established optimal bandwidth is used in the application of `bfast01`. Thus where a breakpoint is detected by `bfast01` in this research, there should at least be a 'TPOI' larger than zero following the methodology of Negri Bernardino et al., 2018, see also figure 5.15b.

All models in this research are tested on the same time series, though. A similar relation between the models was expected for synthetic data and RUE time series, as the synthetic data should resemble the RUE time series. Especially as MLP_{BP} and MLP_{PDF} are expected to function similarly. The hypothesis that model performances compare similarly for both RUE and synthetic time series does not hold.

One of the reasons that the synthetic data does not look similar to the RUE data, is because of the normalisation process. Given the assumption that RUE behaves linearly, two things were assumed earlier, but can be verified: The noise pattern and the value distribution.

The assumption of white noise can be checked by linearly detrending the RUE time series, after which the residuals can be translated to the frequency domain. The summed amplitude spectrum for all time series

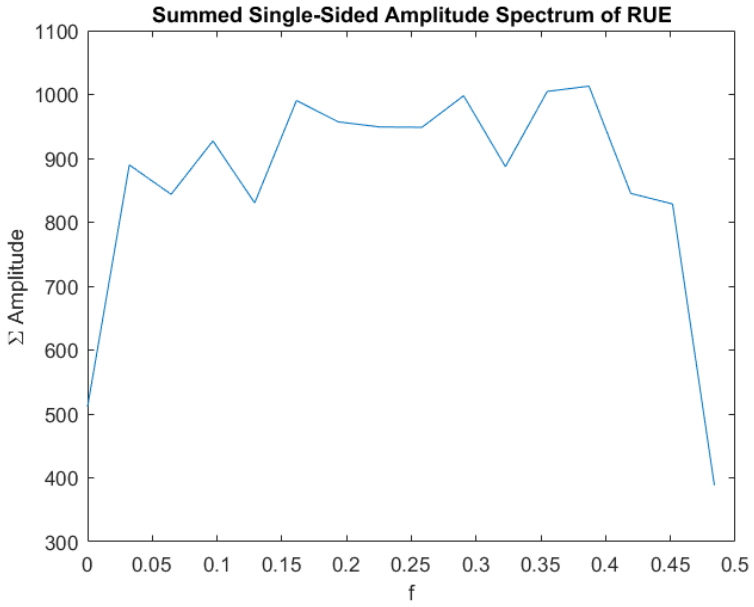
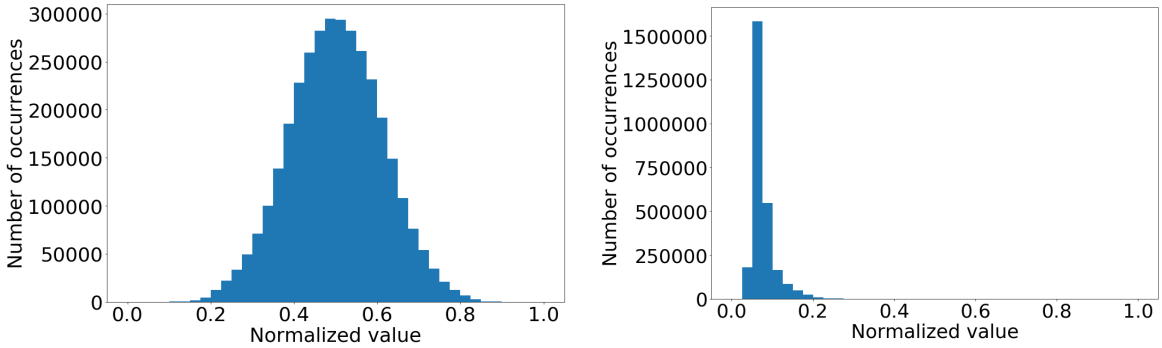


Figure 5.16: The summed amplitude spectrum of detrended RUE time series. Individual time series show similar patterns. At no single frequency, a peak in peak amplitude is observed, indicating the signal has a variability which can be represented by white noise. Individual time series showed similarly shaped amplitude spectra.

is shown in figure 5.16. Individual time series show similar patterns. From the amplitude spectrum it is quickly observed that no frequency shows a dominant amplitude, which indicates white noise is likely a good representation of the noise pattern in the data.

The normalized RUE value distribution is shown in figure 5.17. This normalized value distribution is different from the synthetic time series. The abundance of lower normalized RUE values is of concern, because randomly generated synthetic time series have most energy around a normalized value of 0.5. Their randomly generated nature causes outliers in the positive and negative direction, which cause the whole synthetic dataset to be normalized, 'squashed', to 0.5, the 'middle'. This is thus not the case for actual RUE time series. RUE values are dominated by lower values, with few very large outliers in positive direction. What this means is that the in Chapter 4 presented accuracy of MLP-models cannot be guaranteed for the application of these specific models on RUE time series. This does not invalidate the methodology applied to construct these MLP-models necessarily, or disproves their possible potency. The outliers did not seem to affect model behavior. Even when outliers in the data are manually removed, by removing values outside the range of $Median +/- 3 * \sigma_{std}$ (a robust outlier removal procedure according to Leys et al., 2013), do the earlier shown patterns not change. The distribution of normalized RUE values then looks much more normal, see figure 5.18, though still very different from the synthetic data. Two indicators that the neural nets and `bfast01` generalize well are observed. Firstly, the methods do not seem to recognize outliers as breakpoints, see also figure 5.14. Secondly, the exact same results are acquired when the outliers are removed (not presented). Thus the same results are obtained with a data distribution as presented in figure 5.18 and a data distribution as presented in figure **??**. These data distributions are easily distinguishable. If all possible RUE time series are captured in the training distribution (figure **??**), the MLP-models can be reliable. A method should be used to ensure the similarity of the training data to real-world data, a subject which will be expanded on next chapter.



(a) Distribution of normalized values synthetic data

(b) Distribution of normalized values RUE data

Figure 5.17: Distribution of normalized values compared. RUE values are not normally distributed, like the randomly generated synthetic data. This indicates large dissimilarities can be expected between the RUE and synthetic time series.

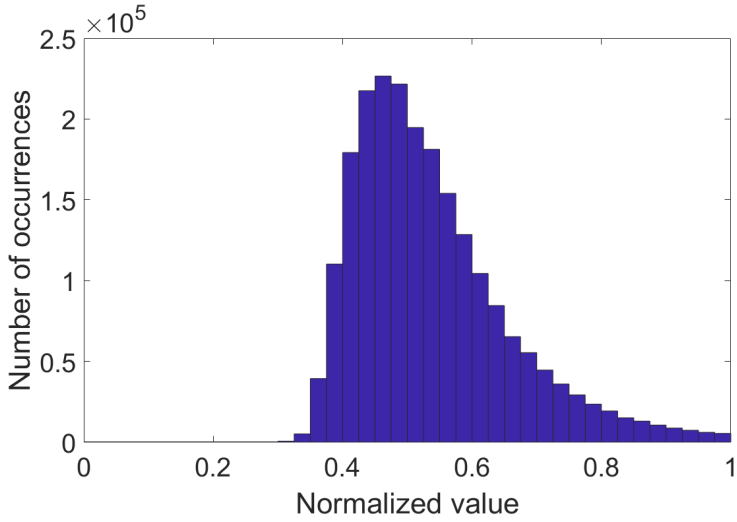


Figure 5.18: Distribution RUE time series when outliers are removed, following the procedure outlined in Leys et al., 2013. The distribution in this image is much more similar to a normal distribution, but still shows large discrepancies.

6 Discussion

Where earlier chapters focused on describing the preparation and execution of an experiment, in this chapter critique is given on some factors influencing the results. Some arguments are touched on in other chapters, but have also found a home in this chapter. This is where practical implications of neural nets in breakpoint detection are discussed as well.

6.1. Considerations with respect to the realism of used synthetic data

6.1.1. Assuming linearity in time series

Deep learning algorithms cannot detect things they are not trained for. Synthetic training data must resemble real data, which is often non-linear in nature (De Jong et al. 2013; Koch et al. 2009). In the context of change detection, expert knowledge on the real world processes is required to label raw data (Wei and Keogh, 2006). To create a proper training dataset for a supervised machine learning approach, some parameters must be taken into account. The nature of the process, what type of behavior is to be expected before and after breakpoints, the type and magnitude of noise. Global sea levels for example, show logarithmic increases (Vermeer and Rahmstorf 2009). Assuming linearity in the process of global sea level change is thus counterproductive. In the context of ecosystem functioning specifically, changes can occur rapidly after critical thresholds are reached (Koch et al., 2009; Negri Bernardino et al., 2018), which introduces inherent non-linearity of the problem. Linearity is often the first guess of a process' functioning, which is often a wrong one (Koch et al., 2009).

Even where linearity is expected, a large discrepancy between real and simulated data is observed. This is the main reason doubt is cast over the change patterns in RUE found in Chapter 5. RUE time series are expected to change monotonically, behave linearly over time (De Jong et al. 2013; Negri Bernardino et al. 2018). The subsequent hypothesis that models performances compare similarly for both RUE and synthetic time series does not hold, however. The synthetic data used in this research is dissimilar from RUE time series in two ways.

First is the distribution of values. This is captured in the range of RUE values after normalization and the noise present in RUE time series. No outliers are simulated, where in some RUE time series they were present, to an extreme degree sometimes. RUE time series also showed only outliers in the positive direction, with many values within a certain bandwidth. The normalized RUE time series value distribution is different from the simulated time series value distribution. The assumption of white noise seemed to hold, though.

Second is any non-linear pattern, which might be present in the form of decadal fluctuations or non-monotonic RUE behavior. El Nino effects in Australia can be of huge influence on precipitation patterns and thus on RUE time series, for example (Chiew et al. 1998). Such environmental patterns over longer times are not captured by statistical methods either, though. These fluctuations over longer times also cast doubt as to whether assuming linearity in RUE time series was right in the first place.

Thus even in a situation where all expert assumptions (linearity, behavior of breakpoint) are used to generate realistic synthetic time series, time series on which deep learning approaches detect change better than widely used methods, the neural net models cannot be blindly trusted. A guarantee is required to have the distribution of real time series be contained within the synthetic time series distribution.

6.1.2. A way forward

Argued is for the use of synthetic data on the basis of three reasons: Enough training data can be generated, it enables comparisons between models under perfect conditions and the models' optimal settings could be established. So far, it is found that the generated time series are too dissimilar from the 'real' time series to guarantee proper neural net functioning. The generated time series are of a particular form.

A workaround which comes to mind is to test for linearity in the target dataset, to search the perfect 'problem.' Linearity of the target dataset can be tested in a variety of ways (Hansen, 1999; Kroll and Emancipator, 1993), but such a workaround would mean to search for a problem which does not necessarily need a solution. This is deemed counterproductive. One way to solve the problem of dissimilarity in training data is presented in this section: Apply a GAN algorithm for data augmentation.

As mentioned in Chapter 1, Chapter 3, there is no large annotated database for change points. Also, the assumed degree of time series containing breakpoints is of influence to the sensitivity of the neural net models (Chapter 3, Chandola et al., 2009). By taking a well known annotated dataset and applying a data augmentation method on it, a sufficiently large dataset might be obtained. Data augmentation is a process whereby the training data is tampered with, to reduce overfitting and enlarge datasets (Perez and Wang, 2017; Van Dyk and Meng, 2001).

Generative Adversarial Networks

Generative Adversarial Networks (GANs) are networks which are data generators (Goodfellow et al., 2014; Perez and Wang, 2017; Salimans et al., 2016). They are a the class of unsupervised deep learning methods. GANs consist of a generative network (G) and a discriminator network (D).

The goal of G is to let D accept its output. G generates altered data based on a noise vector z, which must look like the original data distribution $p_{data}(x)$, D is trained to distinguish whether its input is from the original data distribution. Thus G is trained to generate data in the distribution $p_{model}(x)$ along:

$$p_{model}(x) \approx p_{data}(x) \tag{6.1}$$

GANs are still newly in development (Goodfellow et al., 2014; Radford et al., 2015), first proposed in 2014. They show great promise in various fields of study (Miyato et al., 2018), but are so far mainly applied in classification exercises on various of the large image classification datasets (ImageNet, CIFAR10, Mao et al., 2017; Miyato et al., 2018; Radford et al., 2015; Zhang et al., 2018a).

In lay terms, GANs are designed to produce data which is similar to the original data. Thus the manual assessment in Chapter 5 of dissimilarity between the synthetic and RUE time series, can in some ways deemed to be a failure of G to produce $p_{model}(x)$, the synthetic data, to resemble RUE time series $p_{data}(x)$. D in this case was a visual comparison.

GANs are inherently able to augment existing datasets, breakpoint labelled time series too in theory. D does not require the input data to be labelled, which enables to train a discriminator on any large dataset for which some labelled data exists. $p_{data}(x)$ is then fully utilized. G can then adjust any and all labelled data to 'fool' D. These adjustments should follow a procedure, where labelling of the data is taken into account. This procedure should take expert knowledge of the envisioned application in mind. The labelling of segments or breakpoints should be taken into account, because otherwise breakpoints still cannot be determined, they would 'get lost in translation.' Then the generative models should be able to inversely detect breakpoints in the original data. Such an approach could mean G can incorporate one of the breakpoint detection models proposed in this thesis, though such an approach definitely requires extensive research. In short, GANs show promise to solve for any data dissimilarities found in this research, though such practice has not been developed yet (Mao et al., 2017; Miyato et al., 2018; Zhang et al., 2018a).

6.2. Influences on neural net performance

Mentioned throughout Chapter 3 are the rationale behind the choices made in model training and architecture development. In Chapter 4 and 5, information is gathered on the performance of the different tested neural net architectures. MLP_{PDF} showed high performance changes based their a-posteriori interpretation. What follows are some explanations and suggestions what can be done to the neural net models to enhance performance for various circumstances. When, for example, higher sensitivity in breakpoint detection is warranted. These suggestions are relevant for processes following similar data distribution as the synthetic data used in this research.

6.2.1. Loss functions

Loss functions, or how breakpoints are defined, have major influence on the performance of constructed models. Dependent on the model type and hidden layer layout, the effectiveness of different proposed loss functions varied. Both the simplest (BP) and the most elegant (PDF) breakpoint definition showed best results in combination with a simple model (MLP-model). The LSTM-model tested worked best by defining the breakpoint as a line segment. This works best, because this breakpoint definition favors the use of sequential relations.

The lack of information provided to a model through the PDF loss type showed that models simply optimize for the average if nothing is learned through the loss function. In other words, if the model is 'told' it is wrong, but not why it is is wrong, it tends to make a best guess. Neural nets should thus be designed congruent with the loss function in mind.

The neural net models fitting a PDF as loss function, posed another training difficulty. They optimized to '*divide by zero*' losses eventually, thus could not handle too many iterations. Based on the initialization of weights and biases, a model could be trained. This proved to be possible in about 30% of the training attempts made. What can be learned from this, is that cases can optimize in a sub-optimal direction. When applying a custom or more complex loss function, it is necessary what the limiting constraints of the used loss function are. One can imagine that by using a very complex loss function, it gets real hard real quickly to determine what the bottleneck is in the optimization process.

Both MLP_{BP} and MLP_{PDF} showed that a-posteriori accuracy estimates were possible to create. This happened similarly to expectation and provided an output similar to a Bayesian probability. This is a useful trait if these models are incorporated in larger, more complex models (Richard and Lippmann, 1991). The model certainties can be utilized in both these cases to adjust for model sensitivity. For example, the cut-off σ_{PDF} used to distinguish between the presence or absence of breakpoints can be set differently. In the case of MLP_{BP} a higher threshold can be used to distinguish a breakpoint, which automatically gives the model a greater reliability in detected breakpoints. Such actions are easy ways to change the reliability of the models slightly.

6.2.2. Training order

The order in which training data is fed to a model is of importance, as all weights and biases are optimized best for the last data they encounter during training. For the neural net models as presented, they generally optimize best by 'seeing' less noisy data first. Neelakantan et al. 2015 is proven right herein, this indeed gave best test results, albeit in the order of single percentage points. In addition to this statement, it can be advised to train a neural net on a dataset containing noise, up to a certain noise level.

6.2.3. Interpretability

Any neural net should be interpretable, because otherwise they can be rendered useless in many cases (Zhang et al., 2018b). Both MLP_{BP} and MLP_{PDF} loss functions showed to have interpretable results, as both these models worked according to expectations. Because they showed to function very similarly as well, expected is that the MLP-model layers worked as intended. The LSTM_{CL} model performed best out of the LSTM-models, but when its output differed from expectations no real interpretation could be given to the output. Forcing the presence of only two segments of the same class was attempted in a variety of ways, but could not be achieved. An a-posteriori processing method might prove useful to extract more information from a LSTM-model. Zhang et al., 2018b mention simulatable models as the most transparent, interpretable models. Simulatable means that an entire model can be contemplated at once, the model is simple. In both the MLP- and LSTM-models, layers are simple enough to grasp their decision making process. Post hoc interpretation is done through showing examples, from which useful information could be deduced as well. The best performing model architecture presented, were the most intuitive, simple ones.

6.3. Neural nets and bfast01: A direct comparison

A direct comparison of bfast01 with neural nets is made in three categories: Reliability, execution speed and flexibility. All of these are of importance for practical use of either model, especially in the analysis of large, complex datasets, or when incorporated in rapid-response applications.

6.3.1. Reliability

The reliability discrepancy between `bfast01` and neural nets is circumstantial. Based on tests with the synthetic datasets, `bfast01` can be assumed to not give any false positives. The neural nets proved to be more sensitive to breakpoints. This is reflected in the discrepancy in detected breakpoints in RUE time series in Australia. There is also a strong correlation presented between `bfast01` prediction and reliability of breakpoint detection by neural nets. Where `bfast01` predicted a breakpoint, neural nets were generally 'really sure' a breakpoint was present. The overlap between the `bfast01` and neural net detected breakpoint was just 90%, however. This can indicate that neural net models 'missed' 10% of the 'easy' breakpoints. `bfast01` and neural nets can be used with different settings, influencing sensitivity to breakpoint. Not investigated is the influence of the threshold value in the MOSUM-test, for example.

The neural net models could not reliably be translated to the real-world application of synthetic data, which makes them fundamentally less reliable for now. This is even the case when taking expert assumptions in mind. A solution to cope with this problems is mentioned in Section 6.1.

A practical use a combination of models can be used to detect breakpoints. Which to use or combine is dependent on the circumstances. If a high sensitivity must be warranted, MLP_{PDF} with a high cutoff σ_{PDF} might suffice. If reliability is prioritized, the use of `bfast01` in combination with a sensitive MLP_{BP} model might be used.

6.3.2. Computational speed

Computational speed must be kept to a minimum in many application, albeit for ones' sanity. It is crucial in cases time is of the essence. In natural disaster management or high frequency stock trading are examples of such cases (Chandola and Vatsavai, 2011; Kirilenko et al., 2017), though examples are ample. Admittedly, the methods and models proposed in this thesis deal with offline breakpoint detection only. The models are used as an analysis tool, where time is not a constraint necessarily. The bottleneck in the user-case of Chapter 5 was determining the RUE values, not applying the breakpoint detection systems. However, comparing computational speed is relevant, as these breakpoint detection techniques can help improve online breakpoint detection (Downey, 2008). Another useful thing to mention is that in the analysis of large datasets, computational speed is of importance, even for the sole reason that some algorithms can take years to optimize in their tasks Urban et al., 2016.

There is a significant reduction in computational effort when using neural nets over `bfast01`. Both neural nets and `bfast01` can assess time series in parallel (El-Sharkawi, 1996; Goodfellow et al., 2016; Verbesselt et al., 2010). However, when tested on a single sample, a large difference in computational speed is observed. Executing a single time series takes the models:

- MLP: 0.001 seconds (shortest detectable by python)
- LSTM: 0.004 seconds
- `bfast01`: 0.018 seconds

This means neural nets are approximately a factor 4 to 18 faster. Both methods are parallizable, though it still means neural nets are preferred over `bfast01` based on computational speed. `bfast01` is applied in R and the neural nets are applied in python, which influence execution speeds.

Training the neural nets did not take long. The LSTM-models took far longer to train than the MLP-models. The longest training time was for MLP_{BP} using a million training samples, which took less than half an hour. The models presented in Chapter 4 all took less than five minutes to train. These training times are negligible in most applications, as a neural net needs to be trained only once. Note that missing input data to `bfast01` is internally dealt with through linear interpolation (Verbesselt et al. 2010). It can thus be said that neural nets are much faster than `bfast01` when applied to complete datasets.

Not tested is how these computational speeds differ when increasing the time series length. The computational effort of `bfast01` is expected to increase linearly with increasing time series length, as the amount of MOSUM tests increase linearly with time series length. The MLP-models do not necessarily have to get computationally slower with larger datasets, when the same number of nodes used. These can be computed in parallel. The LSTM-models will also decrease in speed when time series get longer, as with every timestep the LSTM-unit has to be applied once more.

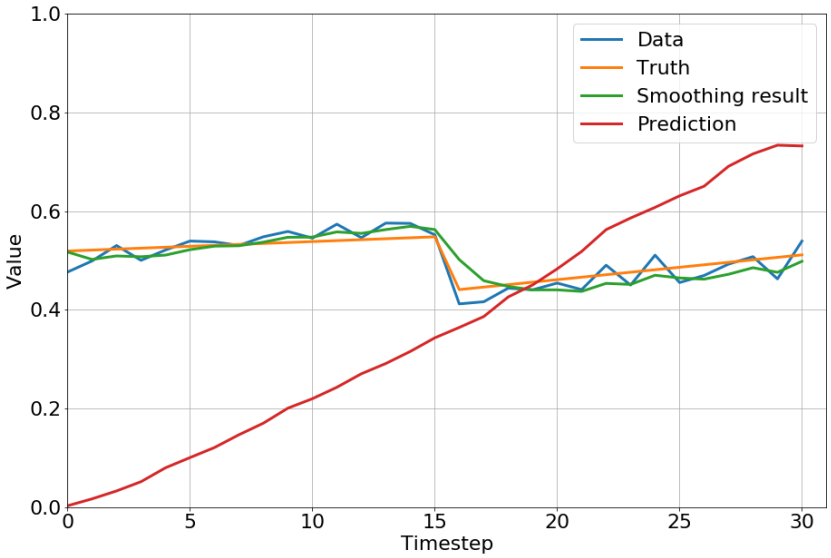


Figure 6.1: An example more complex neural net. The truth is base process to be determined, before noise is added. A LSTM-layer is used to reduce noise from the original data, where a MLP_{CL} model provides a prediction based on this modified time series. For all time series, the MLP_{CL} part of the model gave the exact same output. The noise reduction is applied as expected. This serves as an example how extra features can be added to the neural net models.

6.3.3. Versatility

Time series don't happen in a vacuum. Remotely sensed time series, for example, can be spatially related (Coppin et al., 2004; Lu et al., 2004). If a process is present in one place, a similar process is more likely occurring nearby. This is observed in RUE time series. Multiple parameters, multiple data sources, can also be correlated. For example, the normalized burn index, used to investigate forest fire severity (Cocke et al., 2005; Escuin et al., 2008; Miller and Thode, 2007), can be highly correlated to changing precipitation patterns. Where incorporating such relations with `bfast01` is not trivial, this can be accomplished easier with neural nets.

Using a similar method as outlined in Chapter 3, the neural net models can be adjusted to take in a multiple of data sources or relations. Only single time series were used as input in this thesis. By training on a multitude of correlated synthetic time series, the neural nets can optimize to utilize known relations in the data. The input layer in the neural nets looks a bit different, but the extra information can likely already be mined. Swath of unexpected consequences will likely occur. This flexibility of neural nets over `bfast01` gives room for improvement their time series analysis.

As an example, a different neural net is tested, but not presented yet. It is shows how the versatility of neural nets can easily be used. The model is built out two layers: A LSTM-layer and a fully connected layer. The LSTM-layer is trained to predict the original signal, where the fully connected layer is trained to function similarly as the MLP_{CL} model. Loss is defined as the multiplication of the loss of both layers. This loss is not recommended, as one loss can dominate the other. While this model did not detect breakpoints well at all, an example time series is depicted in figure 6.1. The noise is reduced as intended, but the model to recognize breakpoints failed in all cases. This example does not work well, but still shows the ease with which extra features can be added to the neural net models.

Not investigated in this thesis is the possible presence of multiple breakpoints. The `bfast` algorithm (instead of `bfast01`) can detect multiple breakpoints. It uses the same theoretical basis, thus reliability is not in jeopardy. To distinguish a multiple of breakpoints through neural nets, two solutions are readily available: adjusting loss functions or adding various 'branches' to the models, where each branch tests for a null hypothesis. For the loss functions, 'Breakpoint Label' can be utilized or more classes can be added to the 'Class Label' breakpoint definition. By adding various 'branches,' the idea is to train the same model on time series with different amounts of breakpoints. Then a probability of how many breakpoints are present can be determined by inspecting the performance measure of the different generated models. Both these 'solutions' will have unexpected consequences. Adjusting the neural net models to take into account multiple breakpoints is not trivial. All of the aforementioned adjustments in neural net architecture should be trained on data similar to the dataset on which they will be applied, as outlined in Section 6.1.

7

Conclusion

Change detection algorithms are used in many scientific and industrial branches and come in many shapes and sizes. Monitoring of processes becomes more important as time progresses. Breakpoint detection algorithms for time series can be supervised or unsupervised, offline or online. They all need to process larger and larger amounts of data. Online change detection algorithms can benefit greatly from better offline change detection algorithms. These offline change detection algorithms generally perform worse when the data contains more noise. This drop in performance appears, because often a null hypothesis is tested for in part of the time series. The possible range of values in the tested part has a major influence on the sensitivity of the algorithm to breakpoints.

A field of study where change detection in large datasets is commonly assessed is remote sensing. Remote sensing provides invaluable data, aiding a great many of industries. Many bi-temporal change detection methods exist in remote sensing, limited multi-temporal ones are used. Multi-temporal change detection methods are either very user-dependent or limited in versatility.

A generic deep learning approach for breakpoint detection has been tested in this research to solve both these issues. Deep learning has shown on many occasions to generalize well and to deal with noise efficiently. No general purpose deep learning breakpoint detection algorithm for time series is proposed yet.

The designed neural net architectures were relatively elementary: a double layer perceptron (MLP) and a single layer recurrent neural net (LSTM), combined with four and three different descriptions of breakpoints, respectively. Their elementary nature allowed for a feasibility study, to check whether neural nets can be utilized to detect breakpoints in generic linear time series. These neural nets were trained on a a synthetic dataset, designed to capture most of the variability possible in generic linear time series. This should allow application of such a model in a variety of fields of study.

The neural nets are compare with a staple current breakpoint detection system: `bfast01`. They are compared on both a synthetic dataset and a user-case: Rain-use-effiency (RUE). The synthetic dataset is used to test both neural nets and `bfast01` in a level playing field. RUE is a remote-sensing derived product to assess dryland functioning. It is chosen for the availability of data and assumed linearity in its process. Metrics to determine detected breakpoint quality are established: Accuracy and precision, the occurrence of a breakpoint and the timing of a breakpoint, respectively. Accuracy was decided to be the most important metric in determining model performance. Their performance is assessed in a novel way; performance is assessed by linking performance to the noise observed in the data. This allowed to link model performances with a measurable quantity in any time series.

The main question posed is:

How can neural networks best be used in order to detect breakpoints in time series?

Which is dissected into the sub-questions:

1. How do different neural net architectures compare in the detection of breakpoints?
2. Can reliability of neural nets in breakpoint detection be quantified?
3. How do neural nets compare to `bfast01`?

Both MLP and LSTM based models could achieve high accuracy, even when noisy data is presented to them. The best performing LSTM-model was the model where sequential relations were present in the breakpoint definition. The output of this model was hard to interpret sometimes. The best performing MLP-models showed promising results. In combination with the simplest and the most elegant of chosen breakpoint definitions (defining the breakpoint as a labelled point in a time series (BP) and defining the breakpoint as a Gaussian distribution (PDF)), the MLP-model performed best. Applied to RUE time series, both gave somewhat similar patterns in breakpoints, though there were indicators that both had biases in their predicted breakpoint time.

The MLP_{BP} and MLP_{PDF} models are investigated more closely. Both have a performance indicator in their output: the maximum predicted value and predicted σ_{PDF}, respectively. They show the exact same relation between the observability of breakpoints and their performance indicator. When breakpoints are 'easier' to spot, they detect with large confidence and vice versa. They also have the same relation with `bfast01`: If `bfast01` is certain of a detected breakpoint, the MLP-models are 'very certain'. Applied to RUE time series, the MLP-models detected breakpoints in over 85% of the pixels. Both the patterns in detected breakpoint time and certainty differed vastly, except for a few pockets of later breakpoints.

Based on the tests with synthetic data, it is observed that `bfast01` does not give any false positives, thus is deemed very reliable when detecting breakpoints in time series which behave linearly. Its accuracy drops significantly with increased noise, though. Its precision also drops significantly with increased noise levels. Most neural nets outperformed `bfast01` in terms of accuracy, though only a MLP_{PDF} model is more precise in all cases. A MLP_{BP} model also showed better precision than `bfast01` for time series where both models detected breakpoints. However, a comparison based on Australian RUE time series showed incomparable results, as well as showing a strong bias in `bfast01`, related to the settings used.

Neural networks in breakpoint detection showed some potency. A fundamental potency in their flexibility and utilization speeds, as well as better performance than a statistical method like `bfast01` in a fixed setting. Complexity might be added to the neural nets proposed in this research, by adding more data sources, by adding more functionality to the models or incorporating them in larger models. However, what they have in flexibility and speed, they lack circumstantially in reliability. Biases were present in neural net predictions and the nature of false positive and negative ratios should be taken into account with care. Applied on real data, it is essential to know the behavior of the background process.

In the user-case in this research, similarity between the synthetic and real-world data could not be guaranteed. Expert knowledge is required to create labelled datasets. New developments in generative adversarial networks might reveal new paths to guarantee data similarity. However, the user-case in this research indicate a general, deep learning based, breakpoint detection system based, trained on a truly randomized synthetic dataset, in turn based on common assumptions, does seem to generalize well when the assumptions hold.

The neural nets built in this research performed exceptionally well when presented a well-defined problem. Their predictions optimized along set expectations, certainty measures could even be read in post hoc interpretations. Even though all breakpoint detection models used showed indications of good generalization, the direct translation to a user-case was less successful. The best performing neural nets produced predictions which were dissimilar from expected patterns. They showed biases in certainty indicators and biases in predicted breakpoint times, most of which could be explained. The difference in data distribution between training and real data is attributed as the largest cause for this less reliable performance in the user-case.

What was learned is that rules of thumb should be followed: the best performing neural nets should conceptually be simple and their output easily interpretable. Their performance is tightly linked to the training data used. Thus when basing a decision made by such an algorithm, making good assumptions on your data beforehand is vital.

8

Recommendations

Using deep learning for change detection in time series can be done, though its performance cannot be guaranteed in a general sense. The main problem is to translate the approach taken in this thesis to a user-case. The main assumption made in this thesis seems inappropriate. The assumption that, given a set of assumptions on time series behavior, by randomly generating 'all possible' time series, a neural net optimizes to detect breakpoints in 'all' time series. The neural nets optimized very well in the well-defined problem, when presented 'all possible' scenarios. This means there is a great promise made by deep learning in change detection in time series. If a guarantee can be given that the data distribution used to train the neural nets (p_{train}) is similar to the data distribution used to apply them on (p_{data}), they might be a valuable next step in change detection methods. New developments in the field of machine learning in the form of generative adversarial networks can be valuable in change detection algorithm development. However, by using a well-defined problem, this might not be necessary.

The user-case used in this research, rain-use-efficiency (RUE), was likely not a strategic choice as user-case. It has three major drawbacks:

1. Major assumptions are made on its behavior
2. The time series were short
3. No real validation on resulting predictions can reliably be done

Firstly, it can be argued that the assumption of linearity in RUE time series was a bad one. For a well-defined user-case, all of the possible variability and possible time series must be simulated. A fundamental knowledge on the background process is required for this, which is not the case for RUE time series. It should both be known what type of breakpoints are to be expected in the process, and what good functions are to describe the process. The background process does not have behave linearly, because neural nets showed to generalize well. Assuming literature is right, they are able to generalize well even for non-linear processes.

The RUE time series were only of length 31, only one value is derived per year. This is relatively short, as many monitoring processes have a much higher sampling frequency. A user-case with a longer time series could provide better insight to its noise patterns and behavior. The short length of the time series also limited the possible breakpoint signal. The increased potential for a good breakpoint signal in longer time series, increases the potential to detect them. Using longer time series could also enhance comparisons with statistical methods. The bias in detected breakpoint time observed for `bfast01` is then likely to become less prevalent, for example.

Validation of breakpoints in a user-case increases confidence in model performances. This was not possible in RUE time series. Without validation of some sorts, no model can be trusted. A new user-case should therefor be tested in a known context, before applying it to a new one. This was done in this research, but the presence of breakpoints in Australian RUE time series could not be verified. Different research shows different patterns of breakpoints in RUE over Australia, which is again the case in this thesis (Burrell et al., 2017; De Jong et al., 2013; Negri Bernardino et al., 2018). This casts doubt in how to interpret the results found in this thesis.

So, when having a well-defined problem, where most of the process is understood, **what is then the added value of using deep learning?** The benefits are twofold. Firstly is the analysis quality. Analysis by use of neural

nets can be much better than previous analysis strategies, which is shown in the synthetic data comparisons. Secondly, is the possible versatility of neural nets. This versatility can open new ways to combine different datasets, to extract the most out of available data.

A possible good user-case might be NDVI time series. NDVI time series are measured by different satellites, with different temporal and spatial resolution. They are used as vegetation indicator (Tucker, 1978) and are used for some time in different contexts. The function of NDVI can be approximated pretty well, as different assumptions on its behavior are well-developed (Chen et al., 2004; de Jong et al., 2011; Hird and McDermid, 2009). When a trustworthy neural net then is developed, its versatility can then be tested by inclusion of ancillary information.

Ideally, a user-case is used for which offline breakpoint detection can aid online detection algorithms. In a world where decision making becomes more reliant on automatic change detection systems, it would be comforting to be able to trust those decisions.

9

Acknowledgements

Looking back, this thesis would not be constructed as it is, without the help of a few key individuals. A lot of open source programs and data packages. Considerable help was given in performing the research and writing the text as presented.

I would like to thank all the open source data providers, enabling so many different types of research. I would like thank the NASA GIMMS Group for the availability of the NDVI3g.v1 dataset, and the Climate Hazards Group for the availability of the CHIRPS dataset. Thanks to P. Jonsson and L. Eklundh for making the `TIMESAT` software available and to J. Verbesselt, R. Hyndman, G. Newnham and D. Culvenor for the free availability of `bfast`.

I also would like to thank my team of supervisors, each of whom gave many helpful and thoughtful comments on my work and on life generally. On a great many of occasions, they would help me with coding and writing problems alike, often at the most inconvenient of times: the weekends and evenings. Their humanity and thoughtfulness means a great deal to me.

Hereby I would like to thank Paulo Negri Bernardino and Wanda de Keersmaecker for providing me the draft version of their research paper. Their research paper was the starting point of this venture and inspired the core aspects of the research. I would like to thank Paulo in particular, for his invaluable help in reconstructing their work. Always quick to respond on any question, he made learning R and the early days of producing this thesis a breeze.

A big thanks to Kathelijne Beenen, Mark Bemelmans, my sister Lotte and my parents, who proofread for me. They were willing to think with me, provide their thoughts. This thesis would look a whole lot different without their input.

10

Appendices

10.1. Appendix A: Convolutional layers

Instead of optimizing the weights for all connections between all nodes of the first layer to all the nodes of the last layer, convolutional layers optimize a set number of kernels (Goodfellow et al., 2016). A kernel is applied separate to all nodes in the previous layer, combining information of the node with surrounding nodes into a single value.

Main advantage is that the amount of weights to optimize reduce drastically, as usually a multilayer perceptron with many nodes can be replaced by a convolutional net with a relatively low number of kernels to optimize. In an image of 512x512 pixels for example, a single fully connected layer would need to optimize $(512^2)^2$ weights and an equal number of biases, where a convolutional layer requires to optimize only 9x *Number of kernels* weights when using a small kernel. See figure 10.1 for a visual representation. Side effect is that from first to last layer, the number of nodes reduces (Krizhevsky et al., 2012). A drawback is that the function or process one would like to predict can only be approximated, where the multilayer perceptron can theoretically mimic all functions. In the context of short time series, a 1D context, using convolutional layers would mean applying various 'moving average'-like operations, which in turn are deemed non-beneficial to pursue in this research. The multilayer perceptron performs better in this context and the drawbacks mentioned do not occur here. In a context where much longer time series, or where a multitude of time series are coupled, the use of a convolutional neural net might be better suited.

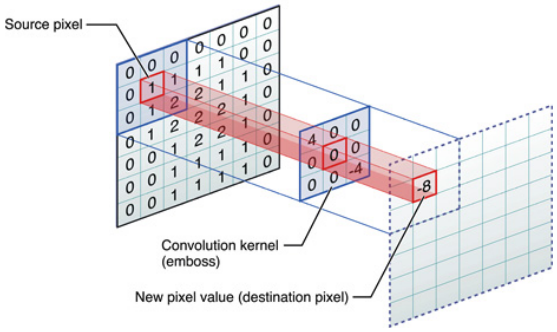


Figure 10.1: Kernels as used in convolutional layers, image from 'k0d3r'. Information from neighboring pixels is included when transferring information from one layer to another.

10.2. Kullback Leibler divergence derivation

p indicates the true value distribution, q the predicted distribution. For the Kullback Leibler divergence between two Gaussians (Allison):

$$
\begin{aligned}
KL(p|q) &= \int plog(\frac{p}{q}) \\
&= -\int p(x)\log q(x)dx + \int p(x)\log p(x)dx \\
&= \frac{1}{2}\log(2\pi\sigma^2_{pred}) + \frac{\sigma^2_{true} + (\mu_{true} - \mu_{pred})^2}{2\sigma^2_{pred}} - \frac{1}{2}(1 + \log 2\pi\sigma^2_{true}) \\
&= \log(\frac{\sigma^2_{pred}}{\sigma^2_{true}}) + \frac{\sigma^2_{true} + (\mu_{true} - \mu_{pred})^2}{2\sigma^2_{pred}} - \frac{1}{2}
\end{aligned}
$$

Bibliography

Dann Albright. What are neural networks and how do they work. https://www.makeuseof.com/tag/what-are-neural-networks/, 2016. Accessed: 2019-09-27.

L. Allison. Kullback leibler (kl) distance (divergence) of two probability density functions. http://allisons.org/ll/MML/KL/Normal/. Accessed: 2020-01-16.

Samaneh Aminikhanghahi and Diane J Cook. A survey of methods for time series change point detection. *Knowledge and information systems*, 51(2):339–367, 2017.

Paul E Anuta, Luis A Bartolucci, M Ellen Dean, D Fabian Lozano, Eeick Malaret, Clare D McGillem, Jose A Valdes, and Carlos R Valenzuela. Landsat-4 mss and thematic mapper data quality and information content analysis. *IEEE Transactions on Geoscience and Remote Sensing*, (3):222–236, 1984.

A Anyamba and JR Eastman. Interannual variability of ndvi over africa and its relation to el niño/southern oscillation. *Remote Sensing*, 17(13):2533–2548, 1996.

'Atom'. Partnership of sub saharan countries to develop green wall in sahel to face desertification and climate change. http://developmentupdates.blogspot.com/2011_11_01_archive.html. Accessed: 2019-04-30.

S Azzali and M Menenti. Mapping vegetation-soil-climate complexes in southern africa using temporal fourier analysis of noaa-avhrr ndvi data. *International Journal of Remote Sensing*, 21(5):973–996, 2000.

Corey Baker, Rick L Lawrence, Clifford Montagne, and Duncan Patten. Change detection of wetland ecosystems using landsat imagery and change vector analysis. *Wetlands*, 27(3):610, 2007.

Igor Barros Barbosa, Marco Cristani, Barbara Caputo, Aleksander Rognhaugen, and Theoharis Theoharis. Looking beyond appearances: Synthetic training data for deep cnns in re-identification. *Computer Vision and Image Understanding*, 167:50–62, 2018.

Horace B Barlow. Unsupervised learning. *Neural computation*, 1(3):295–311, 1989.

Reid Basher. Global early warning systems for natural hazards: systematic and people-centred. *Philosophical transactions of the royal society a: mathematical, physical and engineering sciences*, 364(1845):2167–2182, 2006.

David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6541–6549, 2017.

Yu Bayarjargal, A Karnieli, M Bayasgalan, S Khudulmur, C Gandush, and CJ Tucker. A comparative study of noaa–avhrr derived drought indices using change vector analysis. *Remote Sensing of Environment*, 105(1): 9–22, 2006.

Tamim Bayoumi. The morning after: explaining the slowdown in japanese growth in the 1990s. *Journal of international Economics*, 53(2):241–259, 2001.

Miguel Berdugo, Sonia Kéfi, Santiago Soliveres, and Fernando T Maestre. Plant spatial patterns identify alternative ecosystem multifunctionality states in global drylands. *Nature ecology & evolution*, 1(2):0003, 2017.

Andrew Berg, Eduardo Borensztein, and Catherine Pattillo. Assessing early warning systems: how have they worked in practice? *IMF staff papers*, 52(3):462–502, 2005.

J Martin Bland and Douglas G Altman. Statistics notes: measurement error. *Bmj*, 312(7047):1654, 1996.

Phillip Bonacich and Douglas Kirby. Using assumptions of linearity to establish a metric. *Sociological Methodology*, 7:230–249, 1976.

Arden L Burrell, Jason P Evans, and Yi Liu. Detecting dryland degradation using time series segmentation and residual trend analysis (tss-restrend). *Remote sensing of environment*, 197:43–57, 2017.

Arden L Burrell, Jason P Evans, and Yi Liu. The impact of dataset selection on land degradation assessment. *ISPRS journal of photogrammetry and remote sensing*, 146:22–37, 2018.

Edward G Carlstein, Hans-Georg Müller, and David Siegmund. Change-point problems. IMS, 1994.

Varun Chandola and Ranga Raju Vatsavai. A gaussian process based online change detection algorithm for monitoring periodic time series. In *Proceedings of the 2011 SIAM International Conference on Data Mining*, pages 95–106. SIAM, 2011.

Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

Hao Chen, Nancy Zhang, et al. Graph-based change-point detection. *The Annals of Statistics*, 43(1):139–176, 2015.

Jin Chen, Peng Gong, Chunyang He, Ruiliang Pu, and Peijun Shi. Land-use/land-cover change detection using improved change-vector analysis. *Photogrammetric Engineering & Remote Sensing*, 69(4):369–379, 2003.

Jin Chen, Per Jönsson, Masayuki Tamura, Zhihui Gu, Bunkei Matsushita, and Lars Eklundh. A simple method for reconstructing a high-quality ndvi time-series data set based on the savitzky–golay filter. *Remote sensing of Environment*, 91(3-4):332–344, 2004.

Nicolas Chenouard, Ihor Smal, Fabrice De Chaumont, Martin Maška, Ivo F Sbalzarini, Yuanhao Gong, Janick Cardinale, Craig Carthel, Stefano Coraluppi, Mark Winter, et al. Objective comparison of particle tracking methods. *Nature methods*, 11(3):281, 2014.

Francis HS Chiew, Thomas C Piechota, John A Dracup, and Thomas A McMahon. El nino/southern oscillation and australian rainfall, streamflow and drought: Links and potential for forecasting. *Journal of hydrology*, 204(1-4):138–149, 1998.

Chia-Shang J Chu, Kurt Hornik, and Chung-Ming Kaun. Mosum tests for parameter constancy. *Biometrika*, 82(3):603–617, 1995.

Allison E Cocke, Peter Z Fulé, and Joseph E Crouse. Comparison of burn severity assessments using differenced normalized burn ratio and ground data. *International Journal of Wildland Fire*, 14(2):189–198, 2005.

Jesse Colombo. Japan's bubble economy of the 1980s. `https://www.thebubblebubble.com/japan-bubble/`, 2012. Accessed: 2020-01-29.

Diane J Cook and Narayanan C Krishnan. *Activity learning: discovering, recognizing, and predicting human behavior from sensor data*. John Wiley & Sons, 2015.

Pol Coppin, Inge Jonckheere, Kristiaan Nackaerts, Bart Muys, and Eric Lambin. Review articledigital change detection methods in ecosystem monitoring: a review. *International journal of remote sensing*, 25(9):1565–1596, 2004.

Eric P Crist and Richard C Cicone. A physically-based transformation of thematic mapper data—the tm tasseled cap. *IEEE Transactions on Geoscience and Remote sensing*, (3):256–263, 1984.

Yahya Darmawan and Parwati Sofan. Comparison of the vegetation indices to detect the tropical rain forest changes using breaks for additive seasonal and trend (bfast) model. *International Journal of Remote Sensing and Earth Sciences (IJReSES)*, 9(1), 2012.

Rogier de Jong, Sytze de Bruin, Allard de Wit, Michael E Schaepman, and David L Dent. Analysis of monotonic greening and browning trends from global ndvi time-series. *Remote Sensing of Environment*, 115(2):692–702, 2011.

Rogier de Jong, Jan Verbesselt, Michael E Schaepman, and Sytze De Bruin. Trend changes in global greening and browning: contribution of short-term trends to longer-term change. *Global Change Biology*, 18(2): 642–655, 2012.

Rogier De Jong, Jan Verbesselt, Achim Zeileis, and Michael Schaepman. Shifts in global vegetation activity trends. *Remote Sensing*, 5(3):1117–1133, 2013.

Li Deng, Geoffrey Hinton, and Brian Kingsbury. New types of deep neural network learning for speech recognition and related applications: An overview. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8599–8603. IEEE, 2013.

Dictionary.com. Breakpoint. `https://www.dictionary.com/browse/breakpoint?s=t`. Accessed: 2020-01-14.

Chao Dong, Chen Change Loy, and Xiaoou Tang. Accelerating the super-resolution convolutional neural network. In *European conference on computer vision*, pages 391–407. Springer, 2016.

Allen B Downey. A novel changepoint detection algorithm. *arXiv preprint arXiv:0812.1237*, 2008.

Lars Eklundha and Per Jönssonb. Timesat 3.3 with seasonal trend decomposition and parallel processing software manual. 2017. Accessed: 2019-10-22.

MA El-Sharkawi. Neural networks' power. *IEEE potentials*, 15(5):12–15, 1996.

S Escuin, R Navarro, and P Fernandez. Fire severity assessment by using nbr (normalized burn ratio) and ndvi (normalized difference vegetation index) derived from landsat tm/etm images. *International Journal of Remote Sensing*, 29(4):1053–1073, 2008.

Yahya Farhan, Ali Anbar, Nisrin Al-Shaikh, and Rami Mousa. Prioritization of semi-arid agricultural watershed using morphometric and principal component analysis, remote sensing, and gis techniques, the zerqa river watershed, northern jordan. *Agricultural Sciences*, 8(1):113–148, 2016.

Mathieu Fauvel, Jocelyn Chanussot, and Jon Atli Benediktsson. Kernel principal component analysis for the classification of hyperspectral remote sensing data over urban areas. *EURASIP Journal on Advances in Signal Processing*, 2009(1):783194, 2009.

Rasmus Fensholt and Kjeld Rasmussen. Analysis of trends in the sahelian 'rain-use efficiency'using gimms ndvi, rfe and gpcp rainfall data. *Remote sensing of Environment*, 115(2):438–451, 2011.

Rasmus Fensholt, Kjeld Rasmussen, Per Kaspersen, Silvia Huber, Stephanie Horion, and Else Swinnen. Assessing land degradation/recovery in the african sahel from long-term earth observation based primary productivity and precipitation relationships. *Remote Sensing*, 5(2):664–686, 2013.

Jonathan A Foley, Ruth DeFries, Gregory P Asner, Carol Barford, Gordon Bonan, Stephen R Carpenter, F Stuart Chapin, Michael T Coe, Gretchen C Daily, Holly K Gibbs, et al. Global consequences of land use. *science*, 309(5734):570–574, 2005.

Tung Fung and Ellsworth LeDrew. Application of principal components analysis to change detection. *Photogrammetric engineering and remote sensing*, 53(12):1649–1658, 1987.

Ramanathan Gnanadesikan and Martin B Wilk. Probability plotting methods for the analysis of data. *Biometrika*, 55(1):1–17, 1968.

Jacob Goldberger, Shiri Gordon, and Hayit Greenspan. An efficient image similarity measure based on approximations of kl-divergence between two gaussian mixtures. In *null*, page 487. IEEE, 2003.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural networks*, 18(5-6):602–610, 2005.

Dorothy Hall. *Remote sensing of ice and snow*. Springer Science & Business Media, 2012.

Bruce Hansen. Testing for linearity. *Journal of economic surveys*, 13(5):551–576, 1999.

John R Hershey and Peder A Olsen. Approximating the kullback leibler divergence between gaussian mixture models. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, volume 4, pages IV–317. IEEE, 2007.

Debbie Hillier and Benedict Dempsey. A dangerous delay: The cost of late response to early warnings in the 2011 drought in the horn of africa. *Oxfam Policy and Practice: Agriculture, Food and Land*, 12(1):1–34, 2012.

Jennifer N Hird and Gregory J McDermid. Noise reduction of ndvi time series: An empirical comparison of selected techniques. *Remote Sensing of Environment*, 113(1):248–258, 2009.

Stéphanie Horion, Yves Cornet, Michel Erpicum, and Bernard Tychon. Studying interactions between climate variability and vegetation dynamic using a phenology based approach. *International Journal of Applied Earth Observation and Geoinformation*, 20:20–32, 2013.

Ian W Housman, Robert A Chastain, and Mark V Finco. An evaluation of forest health insect and disease survey data and satellite-based remote sensing forest change detection methods: Case studies in the united states. *Remote Sensing*, 10(8):1184, 2018.

Masroor Hussain, Dongmei Chen, Angela Cheng, Hui Wei, and David Stanley. Change detection from remotely sensed images: From pixel-based to object-based approaches. *ISPRS Journal of photogrammetry and remote sensing*, 80:91–106, 2013.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Barry James, Kang Ling James, and David Siegmund. Tests for a change-point. *Biometrika*, 74(1):71–83, 1987.

Kurt Jax. Function and "functioning" in ecology: what does it mean? *Oikos*, 111(3):641–648, 2005.

Per Jonsson and Lars Eklundh. Seasonality extraction by function fitting to time-series of satellite sensor data. *IEEE transactions on Geoscience and Remote Sensing*, 40(8):1824–1832, 2002.

Per Jönsson and Lars Eklundh. Timesat—a program for analyzing time-series of satellite sensor data. *Computers & geosciences*, 30(8):833–845, 2004.

'k0d3r'. Convolution - opencv. `https://compvisionlab.wordpress.com/2013/04/07/convolution-opencv/`. Accessed: 2019-09-26.

Ehud D Karnin. A simple procedure for pruning back-propagation trained neural networks. *IEEE transactions on neural networks*, 1(2):239–242, 1990.

Saeed Khabbazan, Paul Vermunt, Susan Steele-Dunne, Lexy Ratering Arntz, Caterina Marinetti, Dirk van der Valk, Lorenzo Iannini, Ramses Molijn, Kees Westerdijk, and Corné van der Sande. Crop monitoring using sentinel-1 data: A case study from the netherlands. *Remote Sensing*, 11(16):1887, 2019.

Christian Kiffner, Greta Binzen, Lucie Cunningham, Madison Jones, Francesca Spruiell, and John Kioko. Wildlife population trends as indicators of protected area effectiveness in northern tanzania. *Ecological Indicators*, 110:105903, 2020.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Andrei Kirilenko, Albert S Kyle, Mehrdad Samadi, and Tugkan Tuzun. The flash crash: High-frequency trading in an electronic market. *The Journal of Finance*, 72(3):967–998, 2017.

Evamaria W Koch, Edward B Barbier, Brian R Silliman, Denise J Reed, Gerardo ME Perillo, Sally D Hacker, Elise F Granek, Jurgenne H Primavera, Nyawira Muthiga, Stephen Polasky, et al. Non-linearity in ecosystem services: temporal and spatial variability in coastal protection. *Frontiers in Ecology and the Environment*, 7(1):29–37, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Martin H Kroll and Kenneth Emancipator. A theoretical evaluation of linearity. *Clinical chemistry*, 39(3): 405–413, 1993.

Julien Laliberté, Pierre Larouche, Emmanuel Devred, and Susanne Craig. Chlorophyll-a concentration retrieval in the optically complex waters of the st. lawrence estuary and gulf using principal component analysis. *Remote Sensing*, 10(2):265, 2018.

Eric F Lambin and Alan H Strahlers. Change-vector analysis in multitemporal space: a tool to detect and categorize land-cover change processes using high temporal-resolution satellite data. *Remote sensing of environment*, 48(2):231–244, 1994.

Quoc V Le. Building high-level features using large scale unsupervised learning. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8595–8598. IEEE, 2013.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

Hoyeop Lee, Youngju Kim, and Chang Ouk Kim. A deep learning model for robust wafer fault monitoring with sensor measurement noise. *IEEE Transactions on Semiconductor Manufacturing*, 30(1):23–31, 2016.

Henri N LeHouerou. Rain use efficiency: a unifying concept in arid-land ecology. *Journal of arid Environments*, 1984.

Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

Christophe Leys, Christophe Ley, Olivier Klein, Philippe Bernard, and Laurent Licata. Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. *Journal of Experimental Social Psychology*, 49(4):764–766, 2013.

Xia Li and AGO Yeh. Principal component analysis of stacked multi-temporal images for the monitoring of rapid urban expansion in the pearl river delta. *International Journal of Remote Sensing*, 19(8):1501–1518, 1998.

Richard P Lippmann. Anintroduction to computing with neural nets. *IEEE Assp magazine*, 4(2):4–22, 1987.

Zachary C Lipton and Jacob Steinhardt. Troubling trends in machine learning scholarship. *arXiv preprint arXiv:1807.03341*, 2018.

Sicong Liu, Qian Du, Xiaohua Tong, Alim Samat, Lorenzo Bruzzone, and Francesca Bovolo. Multiscale morphological compressed change vector analysis for unsupervised multiple change detection. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 10(9):4124–4137, 2017.

Song Liu, Makoto Yamada, Nigel Collier, and Masashi Sugiyama. Change-point detection in time-series data by relative density-ratio estimation. *Neural Networks*, 43:72–83, 2013.

WP Loughlin. Principal component analysis for alteration mapping. *Photogrammetric Engineering and Remote Sensing*, 57(9):1163–1169, 1991.

Dengsheng Lu, Paul Mausel, Eduardo Brondizio, and Emilio Moran. Change detection techniques. *International journal of remote sensing*, 25(12):2365–2401, 2004.

Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. Statistical and machine learning forecasting methods: Concerns and ways forward. *PloS one*, 13(3), 2018.

William A Malila. Change vector analysis: an approach for detecting forest changes with landsat. In *LARS symposia*, page 385, 1980.

Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2794–2802, 2017.

Jeffrey G Masek and G James Collatz. Estimating forest carbon fluxes in a disturbed southeastern landscape: Integration of remote sensing, forest inventory, and biogeochemical modeling. *Journal of Geophysical Research: Biogeosciences*, 111(G1), 2006.

Erik Meijering, Oleh Dzyubachyk, and Ihor Smal. Methods for cell and particle tracking. In *Methods in enzymology*, volume 504, pages 183–200. Elsevier, 2012.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*, 2010.

Jay D Miller and Andrea E Thode. Quantifying burn severity in a heterogeneous landscape with a relative version of the delta normalized burn ratio (dnbr). *Remote Sensing of Environment*, 109(1):66–80, 2007.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

David J Mulla. Twenty five years of remote sensing in precision agriculture: Key advances and remaining knowledge gaps. *Biosystems engineering*, 114(4):358–371, 2013.

Kris Nackaerts, Krist Vaesen, Bart Muys, and Pol Coppin. Comparative performance of a modified change vector analysis in forest change detection. *International Journal of Remote Sensing*, 26(5):839–852, 2005.

Arvind Neelakantan, Luke Vilnis, Quoc V Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*, 2015.

Paulo Negri Bernardino, Ben Somers, Jan Verbesselt, Wanda De Keersmaecker, Stéphanie Horion, and Rasmus Fensholt. Characterizing turning points in ecosystem functioning in global drylands. In *BEODAY 2018 2.0, Location: Barvaux-en-Condroz*, 2018.

Rachael H Nolan, Matthias M Boer, Luke Collins, Víctor Resco de Dios, Hamish Clarke, Meaghan Jenkins, Belinda Kenny, and Ross A Bradstock. Causes and consequences of eastern australia's 2019-20 season of mega-fires. *Global change biology*, 2020.

Joseph O Ogutu, Norman Owen-Smith, H-P Piepho, and Mohammed Yahya Said. Continuing wildlife population declines and range contraction in the mara region of kenya during 1977–2009. *Journal of Zoology*, 285(2):99–109, 2011.

Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*, 2017.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Richard J Radke, Srinivas Andra, Omar Al-Kofahi, and Badrinath Roysam. Image change detection algorithms: a systematic survey. *IEEE transactions on image processing*, 14(3):294–307, 2005.

Bharath Ramsundar, Steven Kearnes, Patrick Riley, Dale Webster, David Konerding, and Vijay Pande. Massively multitask networks for drug discovery. *arXiv preprint arXiv:1502.02072*, 2015.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer, 2016.

Nornadiah Mohd Razali, Yap Bee Wah, et al. Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests. *Journal of statistical modeling and analytics*, 2(1):21–33, 2011.

Michael D Richard and Richard P Lippmann. Neural network classifiers estimate bayesian a posteriori probabilities. *Neural computation*, 3(4):461–483, 1991.

JA Richards. Thematic mapping from multitemporal image data using the principal components transformation. *Remote Sensing of Environment*, 16(1):35–46, 1984.

Markus Ringnér. What is principal component analysis? *Nature biotechnology*, 26(3):303–304, 2008.

Craig Rodarmel and Jie Shan. Principal component analysis for hyperspectral image classification. *Surveying and Land Information Science*, 62(2):115–122, 2002.

Edna Rödig, Matthias Cuntz, Jens Heinke, Anja Rammig, and Andreas Huth. Spatial heterogeneity of biomass and forest structure of the amazon rain forest: Linking remote sensing, forest modelling and field inventory. *Global ecology and biogeography*, 26(11):1292–1302, 2017.

Christian Rohrbeck. Detection of changes in variance using binary segmentation and optimal partitioning, 2013.

David Rolnick, Andreas Veit, Serge Belongie, and Nir Shavit. Deep learning is robust to massive label noise. *arXiv preprint arXiv:1705.10694*, 2017.

Stine Kildegaard Rose, Ole Baltazar Andersen, Marcello Passaro, Carsten Ankjær Ludwigsen, and Christian Schwatke. Arctic ocean sea level record from the complete radar altimetry era: 1991–2018. *Remote Sensing*, 11(14):1672, 2019.

Sudipan Saha, Francesca Bovolo, and Lorenzo Bruzzone. Unsupervised deep change vector analysis for multiple-change detection in vhr images. *IEEE Transactions on Geoscience and Remote Sensing*, 57(6): 3677–3693, 2019.

Osvaldo E Sala and Amy T Austin. Methods of estimating aboveground net primary productivity. In *Methods in ecosystem science*, pages 31–43. Springer, 2000.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242, 2016.

Sartajvir Singh and Rajneesh Talwar. Review on different change vector analysis algorithms based change detection techniques. In *2013 IEEE Second International Conference on Image Information Processing (ICIIP-2013)*, pages 136–141. IEEE, 2013.

Gunnar Spreen, Lars Kaleschke, and Georg Heygster. Sea ice remote sensing using amsr-e 89-ghz channels. *Journal of Geophysical Research: Oceans*, 113(C2), 2008.

Susan C Steele-Dunne, Heather McNairn, Alejandro Monsivais-Huertero, Jasmeet Judge, Pang-Wei Liu, and Kostas Papathanassiou. Radar remote sensing of agricultural canopies: A review. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 10(5):2249–2273, 2017.

Sainbayar Sukhbaatar and Rob Fergus. Learning from noisy labels with deep neural networks. *arXiv preprint arXiv:1406.2080*, 2(3):4, 2014.

Frank Thonfeld, Hannes Feilhauer, Matthias Braun, and Gunter Menz. Robust change vector analysis (rcva) for multi-sensor very high resolution optical satellite data. *International journal of applied earth observation and geoinformation*, 50:131–140, 2016.

Compton J Tucker. Red and photographic infrared linear combinations for monitoring vegetation. 1978.

UN. Why now? `https://www.un.org/en/events/desertification_decade/whynow.shtml`, 2010. Accessed: 2019-04-29.

Gregor Urban, Krzysztof J Geras, Samira Ebrahimi Kahou, Ozlem Aslan, Shengjie Wang, Rich Caruana, Abdelrahman Mohamed, Matthai Philipose, and Matt Richardson. Do deep convolutional nets really need to be deep and convolutional? *arXiv preprint arXiv:1603.05691*, 2016.

J Van Angelen, Jan Lenaerts, Stef Lhermitte, Xavier Fettweis, P Kuipers Munneke, M Van den Broeke, and E Van Meijgaard. Sensitivity of greenland ice sheet surface mass balance to surface albedo parameterization: a study with a regional climate model. *Cryosphere*, 6:1175–1186, 2012.

FR van der Burgt, R Riva, C Slobbe, R Klees, CA Katsman, and HA Dijkstra. Remote impacts on caribbean coastal sea level variability. In *American Geophysical Union, Ocean Sciences Meeting 2016, abstract# PO52B-02*, 2016.

David A Van Dyk and Xiao-Li Meng. The art of data augmentation. *Journal of Computational and Graphical Statistics*, 10(1):1–50, 2001.

Tim Van Erven and Peter Harremos. Rényi divergence and kullback-leibler divergence. *IEEE Transactions on Information Theory*, 60(7):3797–3820, 2014.

Jan Verbesselt, Rob Hyndman, Glenn Newnham, and Darius Culvenor. Detecting trend and seasonal changes in satellite image time series. *Remote sensing of Environment*, 114(1):106–115, 2010.

Jan Verbesselt, Achim Zeileis, Rob Hyndman, and Maintainer Jan Verbesselt. Package 'bfast', 2012.

Martin Vermeer and Stefan Rahmstorf. Global sea level linked to global temperature. *Proceedings of the national academy of sciences*, 106(51):21527–21532, 2009.

Sergio M Vicente-Serrano, Célia Gouveia, Jesús Julio Camarero, Santiago Beguería, Ricardo Trigo, Juan I López-Moreno, César Azorín-Molina, Edmond Pasho, Jorge Lorenzo-Lacruz, Jesús Revuelto, et al. Response of vegetation to drought time-scales across global land biomes. *Proceedings of the National Academy of Sciences*, 110(1):52–57, 2013.

Izhar Wallach, Michael Dzamba, and Abraham Heifets. Atomnet: a deep convolutional neural network for bioactivity prediction in structure-based drug discovery. *arXiv preprint arXiv:1510.02855*, 2015.

Robert A Washington-Allen, RD Ramsey, Neil E West, and Brien E Norton. Quantification of the ecological resilience of drylands using digital remote sensing. *Ecology and Society*, 13(1), 2008.

Li Wei and Eamonn Keogh. Semi-supervised time series classification. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 748–753, 2006.

Andreas Weigend. On overfitting and the effective number of hidden units. In *Proceedings of the 1993 connectionist models summer school*, volume 1, pages 335–342, 1994.

Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

Zhizheng Wu, Oliver Watts, and Simon King. Merlin: An open source neural network speech synthesis system. In *SSW*, pages 202–207, 2016.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. *arXiv preprint arXiv:1805.08318*, 2018a.

Quan-shi Zhang and Song-Chun Zhu. Visual interpretability for deep learning: a survey. *Frontiers of Information Technology & Electronic Engineering*, 19(1):27–39, 2018.

Quanshi Zhang, Ying Nian Wu, and Song-Chun Zhu. Interpretable convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8827–8836, 2018b.

Huifu Zhuang, Kazhong Deng, Hongdong Fan, and Mei Yu. Strategies combining spectral angle mapper and change vector analysis to unsupervised change detection in multispectral images. *IEEE Geoscience and Remote Sensing Letters*, 13(5):681–685, 2016.

Jochen Zschau and Andreas N Küppers. *Early warning systems for natural disaster reduction*. Springer Science & Business Media, 2013.