University of Amsterdam

MSc Artificial Intelligence
Master Thesis

Detecting Exoplanet Transit Signals in Light Curves Using Recurrent Neural Networks

---

by
Yke Jan Rusticus
11306386

July 23, 2021

48 ECTS
October 2020 – July 2021

Supervisor:
MSc D. Ruhe

External co-supervisors:
Prof. Dr. B. H. Foing
Dr. A. M. Heras

Assessor:
Dr. P. D. Forré

European Space Agency

[TODO: write Acknowledgements and Abstract]

Contents

Chapter 1

Introduction

The discovery of exoplanets is the first step in answering a broad range of fundamental questions in astronomy. Is our solar system one of a kind? Is there life elsewhere in the universe? Exoplanets, short for extrasolar planets, are planets orbiting stars other than our Sun. The problem of detecting exoplanets is an active region of science and is described in more detail in the first section of this chapter. In Section 1.2, we state our general research aims. Commonly used approaches and their limitations are briefly discussed in Section 1.3, after which we outline our proposed method to address these limitations in Section 1.4. The structure of this thesis is described in Section 1.6.

1.1 Problem

Different approaches to exoplanet discovery exist. This work focuses on the transit method, the most successful method in terms of number of discoveries. The transit method relies on the rare event that a planet moves in front of its host star in our line of sight, i.e. the so-called *transit*. Transits express themselves in the form of periodic dips in the observed brightness of a star over time. The detection of these dips in stellar data is therefore essential for the discovery of new exoplanets.

The observed brightness, or flux, of a star over time is referred to as the star's *light curve*. Generally, transit signals have a duration in the order of hours and light curves have a time span in the order of days or months. A transit signal may repeat itself if the orbital period of the exoplanet is shorter than the time span of the light curve. In case the orbital period longer, a transit signal may occur only once, in which case we speak of a *monotransit*. Moreover, multiple planets might orbit the same star, in which case we can have multiple transit signals from different planets in the same light curve.

The physics of transits is well-understood, but the detection of their signatures in light curves is hindered by stellar and spacecraft induced noise. In addition, transit signals may vary in depth, duration and general shape depending on stellar and planetary parameters. Furthermore, transit signals are in general weak and sparse, so the chance of any given light curve to contain detectable transit signals is low. A detailed description of the challenges and their origins is given in Section 2.3.

The problem we address is thus the detection of dips in light curves of stars that could indicate the presence of an orbiting exoplanet. In literature, however, some ambiguity exists in the naming of different steps in the process of discovering transiting exoplanets. In the *detection* step, i.e. the focus of this thesis, we search for potential signals in a light curve, and provide parameters such as the epoch t_0 and orbital period P of potential signals. The epoch is a reference time, e.g. the time of the first event, which together with P allows for retrieving the signal that was found. Detection should not be confused with *identification* or *vetting*, the step in which the detected signals are classified as false positive, planet candidate, or candidate of another type of object.

1.2 Research aims

The field of exoplanet science has seen a rapid increase in the amount of data over the last decades. For this reason, artificial intelligence (AI) has come to play a more important role in the discovery of exoplanets. Most applications of AI, however, focus on the identification step, e.g. *Autovetter* (Catanzarite, 2015), *Robovetter* (Coughlin, 2017) and *AstroNet* (Shallue and Vanderburg, 2018). Few works exist on the application of AI for the detection of transit signals, an example of which is the work of Pearson et al. (2018). Following their line of research, we aim at contributing to the development of AI-based algorithms for transit signal detection, with the goal to overcome limitations encountered in previous approaches.

1.3 Common approaches and limitations

Most commonly used in the task of transit detection is the Box Least Squares (BLS) algorithm (Kovács et al., 2002). This algorithm uses a box-shaped model of a strictly periodic signal. In order to search for transits, the model is fitted to the data at different signal durations, periods and epochs. Variants to this algorithm exist, for example Transit Least Squares (TLS) (Hippke and Heller, 2019), or Sparse Box Least Squares (SBLS) (Panahi and Zucker, 2021).

One limitation of these methods, is that they require a detrended input light curve. Detrending is the process of removing irrelevant time-dependent noise from the light curve. In the process, however, information is altered at short time scales, so the signal that we wish to find may get reduced or altered (Hippke et al., 2019). Alternatives have therefore been proposed, for example simultaneously modeling the background while searching for transit signals Foreman-Mackey et al. (2015).

However, what these methods still have in common is their brute-force approach to searching for signals. They rely on iterating through trial configurations of parameters that define the signal's shape and timing. In particular for transit models more realistic that the box-function, this iteration over potential signals can be costly. Furthermore, in case the search is directed towards multiple planets in the same light curve, this process has to be repeated several times, each time masking the previous signal that was found. Lastly, the parameter configurations to be iterated over are often specified by the user, which might cause the search to be biased towards specific signal shapes.

An approach based on AI may overcome several of these limitations, as shown by Pearson et al. (2018). In their work, the use of a one-dimensional convolutional neural network (CNN) is proposed for the task of transit detection. On the one hand, neural network outputs can be more difficult to interpret than outputs of box-fitting algorithms, because a neural network can have complex dynamics which make it function like a "black box". On the other hand, they come with several benefits. The CNN does not iterate over transit shapes, and detrending the input light curve is not necessary. In the proposed method, however, the CNN is only capable of providing a single output for an input light curve segment of fixed size. This means that it cannot directly be applied to a full-length light curve of arbitrary size, and we need to take additional steps to determine the timing of potential signals. One option is to apply the CNN to overlapping segments in the light curve to obtain outputs at each time step. However, this approach makes the CNN less intuitive and efficient in its use for detection.

In the task of transit signal detection, the use of recurrent neural networks (RNNs) remains unexplored. The question therefore remains whether RNNs can provide an efficient and competitive alternative to classical transit detection methods, and improve over CNNs in terms of their applicability. Additionally, we raise the question of whether the outputs of RNNs can be made more intuitive for the task of detection. For example, one could estimate the confidence over its outputs, or utilize hidden representations of detected signals in an attempt to make the black box more interpretable.

1.4 Proposed method

We evaluate the use of RNNs for the task transit signal detection. As opposed to CNNs, RNNs can (i) handle arbitrary input sizes and (ii) provide an output at every time step. We make use of property (i) by training the network on light curve segments and applying it to full-length light curves for detection. We make use of property (ii) by training the model to identify exactly those time steps that are part of a transit signal. In other words, the RNN is trained to classify each data point as signal or non-signal.

An important factor in the network's performance is the way in which the input data is preprocessed. Therefore, the question arises which preprocessing steps can or should be considered when using RNNs for this task, and how they affect the model's performance. For example, to address the problem of non-uniform time intervals between measurements, we explore the use of a generative RNN, which is trained to fill in data gaps as it processes through a light curve.

In experiments using simulated data, we assess whether an RNN-based detection algorithm is a viable and competitive alternative to existing approaches. In the task of monotransit detection, we investigate the question of whether the outputs of the RNN can be made more intuitive, in order to set better thresholds and reduce the amount of false detections. To do so, we extend the network to produce confidence outputs in addition to the standard outputs at each time step, as proposed by DeVries and Taylor (2018). The confidence outputs are supposed to indicate the confidence over the corresponding standard outputs at each time step.

In the case of detecting repeating signals, additional steps are necessary to determine the periodicity. Two algorithms are compared in this work, the first of which is inspired by the work by Pearson et al. (2018). Similar to Pearson et al. (2018), we will refer to the network outputs over time as probability time series (PTS). The first algorithm relies on peaks in the PTS, which indicate the potential detections of individual transit events. It uses the average distance between subsequent peaks as a period estimate of the signal. In the search for multiple

planets, one could apply this method iteratively, each time removing the most prominent peaks from the PTS. However, ambiguities may arise if it is not clear which peaks in the PTS belong to which signal. To address this problem, we explore the use of learned signal representations by the RNN, which could help resolve such ambiguities and allow for simultaneous detection of multiple different signals at once. The second algorithm is similar to many algorithms in that it folds the PTS over a set of trial periods. For each trial period, a score is given based on the aggregated values within the folded PTS, and the best scoring value for the period is used as a period estimate.

1.5 Summary of contributions

This work reports the first attempt in literature of using RNNs for the task of transit detection. The research presented revolves around the question of whether and how RNNs could be applied to compete with conventional transit detection algorithms. We provide a detailed analysis of what preprocessing steps could be considered when applying this network to light curves to search for signals. Successes and failure cases are identified for specific design choices, and the general performance is evaluated in comparison to existing approaches. Our approach does not require detrending of input light curves as opposed to commonly used methods. During search, the RNN does not iterate over transit durations, depths or other parameters that define the signal's shape, since the model is implicitly trained to recognize these features prior to application.

In short, we found the RNN to show most potential in the task of monotransit detection, because it does not rely on periodicities of signals and it can be applied highly efficiently. It outperformed the box-fitting algorithm of Foreman-Mackey et al. (2016) in the task of retrieving single transit events in simulated light curves. In the case of multiple transit signals in a single light curve, we found the BLS algorithm to benefit more from the periodicity of the signal, as it outperformed our RNN-based algorithms. In both cases however, the algorithms tested were able to detect signals that the other one was not able to find. In other words, the methods performed complementary. This result suggests that an RNN-based transit detection method could open the door to detecting exoplanets that would otherwise be overlooked by conventional algorithms.

1.6 Outline

This thesis is structured as follows. In Chapter 2, the necessary background is given, for example on the transit method and on existing methods to detect transit signals in light curves. The details of our methods are described in Chapter 3. In Chapter 4, we present the experiments and results, which are discussed in Chapter 5. Finally, the work is summarized and concluded in Chapter 6. Appendix A can be referred to for additional information, for example on conference contributions that were based on this work.

Chapter 2

Background

[TODO: short chapter introduction]

2.1 Transit method

From all exoplanet discovery methods, the transit method has most exoplanet discoveries on its name. At the time of writing, over 75% (3343 in total) of the known exoplanets (4424 in total) were discovered using this method. Of the other methods, the radial velocity method is the most successful and covers about 20% of the exoplanet discoveries. The radial velocity method works by measuring the shifts in the spectrum of a star which are due to the gravitational pull of an orbiting planet. While this used to be the prominent method for exoplanet discovery, nowadays it is mostly used to complement the transit method as a means of confirming exoplanet candidates, or characterizing planetary systems. Anther method closely related to the transit method uses transit timing variations (TTVs) of transiting planets to infer the presence of an additional planet in the same system. TTVs are deviations from the expected periodicity of a signal, which could be caused by a disturbing planet in a different orbit around the same star. Other methods to exoplanet discovery include for example gravitational microlensing and direct imaging, but for the purpose of this thesis we do not go into depth for these methods.

Exoplanets are far away and form a compact system together with their host star. This makes direct imaging difficult. Therefore, most methods rely on indirect measurements of potential exoplanets. This is also the case for the transit method, because only the brightness of a star is monitored over time. Dips in the observed brightness can indicate an exoplanet passing in front of the stellar disk. A requirement for this method is therefore that the system is seen edge-on.

From the shape of the transit signal we can learn several parameters about the planet and its host star. For example, the depth of the signal relates to the planet size (depth $= R^2_{\text{planet}}/R^2_{\text{star}}$). The duration relates to the period of the signal, which in turn relates to the distance of the planet from its host star.

[TODO: explain phase folded light curve and how it increases SNR]

[TODO: describe ingress, egress, mid-transit, etc.]

[TODO:

$$P^2 = \frac{4\pi^2}{GM} a^3. \tag{2.1}$$

]

2.2 Exoplanet-hunting missions

Since the first exoplanet discovery, the rate at which exoplanets are discovered has grown exponentially. Ground-based surveys such as WASP, HAT, TRAPPIST, KELT [TODO: references] are responsible for the discovery of several exoplanets. However, the major turning point was when the search for transiting exoplanets was taken into space.

CoRoT was the first space-based mission designed to search for transiting exoplanets and added 33 exoplanets to the list. Kepler and its continued mission K2 hold the discovery of over half of the known exoplanets (> 2500). Kepler observed a small patch of the sky and monitored over 500,000 stars. Most targets were observed with an observation interval of 30 minutes, i.e. a 30-minute cadence. TESS, Kepler's successor, monitors brighter and closer stars in a full-sky survey. After each 27.4-day sector, the telescope points at a different patch of sky. Midway each sector, the spacecraft transmits data back to earth, during which no new data is gathered. With over 200,000 stars observed and "only" 131 exoplanet discovered, thousands discoveries more are expected.

The amount of data in the field will only increase more rapidly, as future telescopes will get bigger and better. Planned for launch in 2026, PLATO will monitor the brightness of around up to a million stars for the detection and characterization of transiting exoplanets.

2.3 Challenges in detecting transit signals

In the task of transit detection, the presence of an exoplanet is not known beforehand, so we need to search the light curves for potential signals. However, many noise sources hinder the search and could trigger false detections. This and other challenges are described in the following subsections.

2.3.1 Stellar variability

Stars do not emit photons at a constant rate. A star's surface is dominated by cell-like structures called granules. Energy transport within the star causes hotter and brighter cells of plasma to rise, and cooler and dimmer plasma to descent. This granulation effect leaves the stellar surface constantly changing, and results in fluctuations in the star's light curve. [TODO: specify time scale]. At larger time scales, starspots could leave their imprint on a light curve. These visually dark regions on the star's surface are relatively low in temperature, and thus decrease the flux. As the star rotates, the starspot could come in and out of view, which results in large scale fluctuations in the star's light curve. This effect is called rotation modulation. Additionally, other phenomena could be at play such as stellar oscillations or sudden outbursts of energy called flares. In summary, a light curve may exhibit quasi-periodic patterns caused by complex mechanisms.

2.3.2 Instrumental and photon noise

All data used in this work is assumed to be collected by space-based observatories. Although space offers a good environment for astronomical observations, the instruments still suffer from imperfections. For example, temperature changes in the spacecraft might alter the pointing of the telescope, which could cause jumps or drifts in the observed flux from stars in the field of view. Incorporating information on the telescope's pointing, or centroids, in addition to the flux values seems beneficial in some exoplanet-related tasks [TODO: cite centroid examples].

Stray light, e.g. reflected off the moon, might influence the quality of observations. Poor quality data may flagged by the spacecraft's pipeline, which we could filter out. However, this leaves missing data points in the light curve, which might pose a problem to certain processing algorithms. Another source of noise, although not due to instrument imperfections, is photon noise. This noise originates from the arrival of different amounts of photons at the detector, between different observations. In general, this type of noise can be seen as the time independent Gaussian-distributed noise present in the data.

2.3.3 Transit signal shapes

The basic shape of a transit signal is determined by, among others, the planet's orbital period, radius relative to its host star, distance from its host star, and inclination and eccentricity of its orbit. Each parameter affects the shape of a transit differently. It is not the task of the detection algorithm to determine all these parameters, but the algorithm should be robust against different signal shapes.

The stellar surface also plays a role in the shape of the signal. The limbs of the star seem darker than its central region. This is because at the central region, we look deeper into the star, where temperatures are higher. When a planet passes in front of the star, it therefore blocks a varying portion of the star's total emitted light. For this reason, a box-shaped transit model is not accurate. More realistic transit models take this limb darkening effect into account. Other effects can cause transit depths to vary over time, for example when a planet passes in front of starspots during transit.

2.3.4 Weak and sparse signals in big data

As mentioned, a requirement for the transit to work is that the planetary system is seen edge-on. However, nothing restricts a system from being oriented randomly. For this reason, even if all stars would host exoplanets, a large portion would still go undetected. Moreover, the farther the planet from the host star, the smaller the chance that the planet transits the star.

For example, the chance of earth to transit the sun if it were observed from outside the solar system is only 0.47%. Even if it does, Earth transits only once per year and leaves dip of 80 ppm in the Sun's observed flux for 13 hours. In other words, transit signals are weak and sparse, especially those we are most interested in.

The reason why still many exoplanets are discovered by the transit method, is mainly due to large amount of available data. For example, TESS alone collects over 20 GB every day [TODO: cite TESS data amount].

2.3.5 Astrophysical false positives

Sometimes, a false positive detection is not due to stellar or spacecraft induced noise, but due to other phenomena. For example, a large portion of the stars exists in a binary configuration. In case an eclipsing binary (EB) system is seen edge-on from Earth, the stars transit each other just like an exoplanet would transit its host star. Generally, EB signals will be larger than for exoplanets, but if the stars have grazing transits, the signals can become smaller and more similar to exoplanet transits.

EB systems lurking in the background of an observed star can also produce disturbing signals. These background EBs (BEBs) are much farther away than the star under consideration, but contribute to the star's observed light curve. The BEB transit signals will thus also be smaller and could mimic exoplanet transit signals in the light curve of the foreground star.

In the detection step, however, it is not the main goal to discern (B)EBs from exoplanet transit signals. Preferably, most EB signals are filtered out from the start, but restricting our detection algorithm too much on what kind of signals it is allowed to pick up, may exclude various interesting signals from being detected. For example, the detection step could return a signal that is in fact caused by a large exocomet or asteroid, which would be of great astrophysical value to find and study. Therefore, in this work we focus our detection efforts on potential transit signals, which may include astrophysical false positives. Astrophysical false positives can be further categorized or filtered out in the identification step. [TODO: revise]

2.4 Transit detection methods in literature

[TODO: short section introduction]

2.4.1 Box and Transit Least Squares

The Box Least Squares (BLS) algorithm, proposed by Kovács et al. (2002), is widely adopted in the task of transit signal detection [TODO: cite examples]. Its simplified model of the transit signal makes it relatively efficient, because the model is only parametrized by the transit depth, duration and timing. The model is fitted to the data by minimizing the squared error, or similarly, maximizing the log likelihood of the model with respect to the transit depth δ for a given configuration of orbital period P, transit duration τ and epoch t_0[1]. The log likelihood at different values for P, maximized over (δ, τ, t_0) is proportional to the Signal Residue (SR) as defined by Kovács et al. (2002). The SR at each given trial period defines the BLS periodogram, which can directly be used for detection. For example, a peak in the periodogram at a given period P could indicate the detection of a signal with corresponding parameters (δ, τ, t_0, P).

[TODO: show periodogram example and explain harmonics].

Several different detection thresholds can be set using this periodogram. For example, a detection threshold could be set on the signal-to-noise ratio (SNR) of transit candidates. Ignoring the presence of time-dependend noise, the SNR can be defined as $\delta/\sigma_w \cdot \sqrt{n_t}$ with $n_t \propto \tau$ the number of measurements that belong to the transit signal and σ_w the estimated white noise in the detrended light curve. However, Pont et al. (2006) show that a detection threshold for BLS can be set better if time-dependend "red" noise is accounted for in the approximation of the SNR. This is likely because background patterns with time scales similar to the transit duration could remain intact after detrending.

More commonly, a detection threshold is set on the Signal Detection Efficiency, which is defined as (Kovács et al., 2002):

$$SDE = \frac{SR_{\text{peak}} - mean\,(SR)}{std\,(SR)}, \tag{2.2}$$

where SR_{peak} is the maximum SR in the periodogram, and $mean\,(SR)$ and $std\,(SR)$ are the mean and standard deviation of the periodogram respectively.

Several variants to BLS exist. For example, Carter and Agol (2013) relax the assumption of strictly periodic signals, and use a quasi-periodic transit model that is fitted to the data. Foreman-Mackey et al. (2016) omit the periodicity assumption altogether and use an algorithm similar to BLS to search for monotransits. In their work, a box function is fitted to the data at multiple points in time, and a detection threshold for single events is set on the SNR above the noise floor determined by a sliding median filter.

More recently, the Transit Least Squares (TLS) algorithm was proposed by Hippke and Heller (2019). As opposed to BLS, TLS takes into account the effect of limb darkening on the transit shape, and is therefore more expressive than BLS. For this reason, TLS generally performs better at detecting transit signals than BLS, however at the cost of longer computation times.

[1] https://docs.astropy.org/en/stable/timeseries/bls.html

The performance of BLS and TLS largely depends on the way the light curve is detrended prior to the search. Hippke et al. (2019) compare a wide range of detrending methods, before applying TLS to search for transits. They evaluate each method by the extend to which known transit signals are retrieved by the TLS algorithm, after applying the detrending method. The methods evaluated include sliding mean and median filters, Gaussian processes, the Savitzky-Golay filter, and more. The results showed that detrending could significantly alter or reduce the transit signals, for some methods more than others. A time-windowed median filter performed reasonably considering its computational efficiency. Furthermore, for window-based filters, it was found that a window size of three times the transit duration is the best choice for detection. The problem is however, that the presence of a transit is not known beforehand in the task of detection, let aside its duration.

2.4.2 Simultaneously modelling background and signal

In order to avoid the risks that are involved with detrending, one could simultaneously model background stellar activity with the transit signals. This approach was taken by Foreman-Mackey et al. (2015), where each light curve was described as a combination of the 150 most representative components of the principle component analysis (PCA) of K2, while simultaneously fitting for a box-shaped transit model. Since the background is modelled side-to-side with the transit signal, the background model is less prone to overfit to the transit signal, and the transit signal is therefore expected to stay better intact.

On the contrary, Kovács et al. (2016) argue that with simultaneous modelling, there are more degrees of freedom and thus more chance of false positives. They found better results by first detrending the light curve, and subsequently searching for transit signals. This approach was also found to be considerably more efficient.

2.4.3 Kepler and TESS pipeline

The pipelines of Kepler and TESS cover the process of reducing the raw images of stars to systematics corrected light curves and detecting potential signals. The TESS pipeline is largely based on the Kepler pipeline (Jenkins et al., 2016), so we base our description of the transit search in both pipelines on the Transiting Planet Search (TPS) module as described in the Kepler Data Processing Handbook Jenkins et al. (2017).

In the TPS module, the input light curve first undergoes a series of preprocessing steps, e.g. stitching sectors together and filling data gaps. Subsequently, a time-varying whitening filter is applied, so the light curve is transformed to the time-frequency domain. This pre-whitening filter is supposed to clear the light curve of irrelevent time dependend noise. The use of a pre-whitening filter for transit search has been adopted more often, e.g. Carpano et al. (2003), as it is considered to be the optimal detector in combination with a simple matched filter in the case for colored Gaussian noise, as explained in Jenkins (2002). However, Rodenbeck et al. (2018) note that a pre-whitening filter can introduce features that could be misinterpreted as signal. The pipeline therefore removes positive flux outliers which could introduce transit-like features in the whitened flux. Since the whitening filter can also alter transit signals, the same whitening is applied to the transit pulse train that is used as transit model. After single events have been determined, a grid of periods, durations and epochs is searched through to find a least squares fit to each potential signal. A representative limb darkening model is used to fit with the data.

Although the pipeline is far in its development and is constantly being improved, it still relies on heavy preprocessing and an extensive search through parameter configurations of potential signals. Furthermore, the TPS module in the pipeline is designed as general algorithm to retrieve most of the hidden transit signals. Since this algorithm might still miss signals, especially in special cases, it is worth investigating different methods for transit detection that are not previously considered.

2.4.4 Other classical detection methods

In cases where a periodic signal resembles a sine function, the Fourier transform (FT) provides a good way to detect it. For a transit signal this is in general not the case as its duration is short relative to its period. For ultra-short-period (USP, < 1 day) planets on the other hand, the duration becomes larger relative to the period, and the FT will have similar detection performance as the BLS algorithm as shown by Sanchis-Ojeda et al. (2014). In their work, an FT-based detection method is applied to detect transits from USP planets in Kepler data. They argue that the FT produces less disturbing harmonics in its resulting spectrum compared to BLS, but BLS is more effective for longer period planets.

Another approach to transit detection is by using the dispersion of the phase folded light curve. Plavchan et al. (2008) fold a given light curve over several trial periods, each time evaluating the difference between the folded light curve and its boxcar-smoothed counterpart. A small set of periods for which the folded curve corresponds best to the smoothed phase curve, is further analysed for transits.

Wheeler and Kipping (2019) also propose a method which aims to minimize the phase dispersion by folding the light curve at different trial periods. Since this method does not assume any specific signal shape, it is

sensitive to strictly periodic signals of arbitrary shape. Their method was applied by Chakraborty et al. (2020) to find 377 previously unreported signals in the first year of TESS observations.

2.4.5 "Intelligent" algorithms

The human eye can function as an excellent tool for pattern recognition and can exceed algorithms in complex ways. For this reason, the citizen science project Planet Hunters was launched. Participants, or users, are presented with light curves from Kepler and asked to flag parts of the light curve that resemble transit signals. Six months after the launch, millions of classifications were made and later Fischer et al. (2012) reported the detection of the first two planet candidates that were flagged by participating volunteers.

This process can be viewed as a detection algorithm. The user is in this case the detector, which utilizes prior knowledge about the shapes of different transit signals to flag potential signals. Without requiring the light curve to be detrended or folded, individual events are flagged by the detector. If there is enough confidence about a certain signal, i.e. enough users have flagged the same events, the 'detection' is passed to the vetting stage. For Planet Hunters, this last stage consists of experts in the field who rule out clear false positives and conduct follow up observations to confirm the planets.

Similarly, one could use artificial intelligence for the task of transit detection. Pearson et al. (2018) trained a one dimensional CNN to classify light curve segments as signal or non-signal. [TODO: explain CNN]. The CNN lends itself well for identification and forms the basis of several recent works in the field such as *AstroNet* (Shallue and Vanderburg, 2018) and variants (Ansdell et al., 2018; Dattilo et al., 2019; Koning et al., 2019; Yu et al., 2019; Osborn et al., 2020). For detection, however, the model is limited by the fact that is only allows relatively small inputs of fixed size. Therefore, Pearson et al. (2018) propose two approaches to use the CNN for detection. In the first approach, the network is applied to overlapping segments in the light curve to obtain outputs at each point in time, which are referred to as probability time series (PTS). Subsequently, the average distance between peaks in the PTS can give an estimate of the periodicity of the signal. In the second approach, the light curve is folded over given trial periods, each time applying the CNN to overlapping segments in the the resulting phase curve. A detection in the phase curve directly gives an estimate of the periodicity, but this approach requires the same parts of the light curve to be evaluated many times by the CNN. Zucker and Giryes (2018) tested the feasibility of using CNNs for detection in comparison with the BLS algorithm. They simulated light curves with Gaussian processes to account for stellar variability. Their task was to output a binary classification (signal or non-signal) for a given input light curve.

Extending on this work, Chintarungruangchai and Jiang (2019) evaluated the use of two-dimensional CNNs. Instead of folding the light curve to one dimension, they fold the light curve such that each folded segment is stacked on top of each other. Subsequently, they train the CNN to classify these two-dimensional "images" as signal or non-signal. Although multiple trial periods still need to be evaluated to search for a potential signal, this method is found to be more robust against deviations from the true period than the method proposed by Pearson et al. (2018). However, the problem remains that the methods evaluated by Chintarungruangchai and Jiang (2019) and Zucker and Giryes (2018) only provide a binary output for a large input, and do not allow to determine the exact timing of the individual signals.

2.5 Recurrent neural network (RNN)

2.5.1 Theory

The field of machine learning covers a broad range of techniques and methods, e.g. k-nearest neighbours, decision trees, linear regression. A subfield of machine learning is deep learning, which is based on artificial neural networks (NNs) such as the RNN and CNN. A simple NN architecture is the multilayer perceptron (MLP), which is a network consisting of fully-connected (FC) layers of computational units called neurons. The output at each layer $h_i \in \mathbb{R}^n$ in an MLP can be obtained by transforming the preceding layer $h_{i-1} \in \mathbb{R}^m$ as:

$$h_i = \phi(W_{i-1,i} h_{i-1} + b_i), \tag{2.3}$$

where $W_{i-1,i} \in \mathbb{R}^{n \times m}$ is the weight matrix between layers $i-1$ and i and $b_i \in \mathbb{R}^n$ the bias vector. $\phi(\cdot)$ is the activation function, which is used to introduce non-linearities in the network. Between layers, $\phi(\cdot)$ is often chosen to be the ReLU activation function, i.e. $\phi(a) = \text{ReLU}(a)$, where

$$\text{ReLU}(a) = \max(0, a). \tag{2.4}$$

The network's output y for a given input $X \in \mathbb{R}^N$ is given by the function $y = g(X, W)$, which represents the neural network. In this notation, we use W to refer to the collection of all weight and bias parameters in the network. For example, for an MLP with one hidden layer we have

$$g(X, W) = \psi(W_{1,2}(\phi(W_{0,1} X + b_0)) + b_1), \tag{2.5}$$

where, for example, in the case of binary classification, the output $y \in \mathbb{R}$ is mapped between 0 and 1 by choosing $\psi(a) = \sigma(a)$, where $\sigma(a)$ is the logistic sigmoid function given by

$$\sigma(a) = \frac{1}{1 + \exp(-a)}. \tag{2.6}$$

For an RNN, the network function, say $f(X, W)$, becomes more complicated. This is because an RNN uses each element x_j in X to update its hidden state recursively. Consider X to be a time series, so we can speak of individual time steps j with corresponding data points x_j. The hidden state $h_j \in \mathbb{R}^K$ of the "vanilla"-RNN at time step j is given by

$$h_j = \phi(Uh_{j-1} + Vx_j + b), \tag{2.7}$$

where $U \in \mathbb{R}^{K \times K}$ and $V \in \mathbb{R}^{K \times N}$ are weight matrices and $b \in \mathbb{R}^K$ the bias vector, which are the same for each time step. An output for each time step can be obtained by applying some function to h_j. For example, we can apply Equation 2.5 to the hidden states of the recurrent layer to obtain an output y_j for each time step j, i.e. $y_j = g(h_j, W)$. The complete RNN then outputs a vector $Y \in \mathbb{R}^N$ with elements y_j for an input sequence X, i.e. $Y = f(X, W)$.

Equation 2.7 represents the update rule for a unidirectional RNN, i.e. an RNN which only propagates through time in one direction. However, in some applications it is beneficial to utilize information form both directions in time. This can be achieved by concatenating the hidden states of two RNN components, propagating though the input in opposite directions:

$$H_j = \begin{bmatrix} h_j \\ h'_j \end{bmatrix} = \begin{bmatrix} \phi(Uh_{j-1} + Vx_j + b) \\ \phi(U'h'_{j+1} + V'x_j + b') \end{bmatrix}. \tag{2.8}$$

h'_j is of the same dimensionality as h_j and is obtained using weight matrices V' and W' and bias vector b', which are of the same dimensionality as V, W, b. The result, $H_j \in \mathbb{R}^{2K}$, can be used in the same way as before, e.g. $y_j = g(H_j, W)$.

The key to a well-performing network is not only to have an appropriate network architecture, but also to have a good training procedure. A network is trained by updating its trainable parameters W to optimize for a given loss function $\mathcal{L}$. To do so, generally the derivative of $\mathcal{L}$ is taken with respect to W with a process called "backpropagation", which efficiently computes the gradient at each layer using the chain rule. The derivative can be used to update W in the direction of a lower loss value, for example using (stochastic) gradient descent or more advanced optimization algorithms such as Adam (Kingma and Ba, 2014).

A problem may occur, however, if the network has many layers. Since the gradient of the first layer in the network is computed using the chain rule, i.e. by multiplying the gradients of all subsequent layers, we may have that the gradients quickly explode or vanish. For a vanilla-RNN this is problematic, because the gradient is propagated through time, in which we treat each time step in the RNN as a separate layer. Suppose that the input X consists of 1000 data points. Even if the gradients of each individual layer lie around 0.9, then the gradients of the loss with respect to the weights of the first layer can get as low as $0.9^{1000} \approx 10^{-46}$. This means that earlier time steps will contribute very little to the weight update, and the RNN will not be able to learn long term dependencies.

For this reason, different RNN architectures have been proposed. Several proposed RNN architectures use gates to better control the flow of information. One of these is the long short-term memory (LSTM, Hochreiter and Schmidhuber (1997)) RNN. For LSTM, Equation 2.7 can be replaced by the following[2]:

$$i_j = \sigma(W_{ii}x_j + b_{ii} + W_{hi}h_{j-1} + b_{hi}) \tag{2.9}$$
$$f_j = \sigma(W_{if}x_t + b_{if} + W_{hf}h_{j-1} + b_{hf}) \tag{2.10}$$
$$g_j = \tanh(W_{ig}x_j + b_{ig} + W_{hg}h_{j-1} + b_{hg}) \tag{2.11}$$
$$o_j = \sigma(W_{io}x_j + b_{io} + W_{ho}h_{j-1} + b_{ho}) \tag{2.12}$$
$$c_j = f_j \odot c_{j-1} + i_j \odot g_j \tag{2.13}$$
$$h_j = o_j \odot \tanh(c_j), \tag{2.14}$$

where $\odot$ is the element-wise product. The gates i_j, f_j, g_j, o_j are known as the input, forget, cell and output gates respectively, and c_j is known as the cell state.

Another, yet similar, architecture is the gated recurrent unit (GRU, Cho et al. (2014)) RNN. For GRU,

[2] https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html

Equation 2.7 can be replaced by the following[3]:

$$r_j = \sigma(W_{ir}x_j + b_{ir} + W_{hr}h_{j-1} + b_{hr}) \tag{2.15}$$
$$z_j = \sigma(W_{iz}x_j + b_{iz} + W_{hz}h_{j-1} + b_{hz}) \tag{2.16}$$
$$n_j = \tanh(W_{in}x_j + b_{in} + r_t \odot (W_{hn}h_{j-1} + b_{hn})) \tag{2.17}$$
$$h_j = (1 - z_j) \odot n_j + z_j \odot h_{j-1}, \tag{2.18}$$

where the gates r_j, z_j, n_j are known as reset, update and new gates respectively. GRU has fewer gates and thus fewer parameters than LSTM.

2.5.2 Applications

RNNs seem to be missing from literature concerning transit detection. They have been applied in related tasks, for example to predict stellar parameters or the number of potential transit signals, given a light curve (Hinners et al., 2018). In the latter, however, the bidirectional LSTM (bi-LSTM) only achieved near-random results, which the authors attributed to the imbalance in the data. We note that it might also have been due to the sparse learning signal that is given to the RNN. Namely, only after having seen about 7000 input data points, the RNN outputs a single classification vector or value, for which a learning signal is given. This way, figuring out where transit signals are present in a given light curve can only be learned implicitly by the model, even though we have the possibility to provide this information as learning signal at every time step.

Another application of the RNN is that of detrending light curves (Morvan et al., 2020). An LSTM was trained to learn out-of-transit patterns, and interpolate these within the duration of a given transit signal. By doing so, the authors show that the background of a transit signal can be better subtracted, which allows for better characterization of the exoplanet. Results improved if centroid data was included.

Other applications of RNN mostly involve the classification of light curves as a whole, for example for the classification of variable stars. Jamal and Bloom (2020) compare different neural network architecture for this task, including dilated temporal convolutional networks (dTCNs), LSTMs, GRUs, temporal convolutional NNs (tCNNs) and encoder-decoder models. The LSTM and tCNN were found to be performing best, with the latter having the benefit of shorter computation times.

In the context of variable star classification, the problem of irregularly time series in addressed by Naul et al. (2018). To deal with this problem, time intervals between measurements are used as additional input to the RNN. Becker et al. (2020) also experiment with time intervals between measurements as inputs to their model, and additionally use differences between flux values as inputs instead of their raw values. They used GRU over LSTM, because of its roughly similar performance, but lower computational requirements. Within machine learning literature, other methods have been proposed to deal with irregularly sampled time series. One of these methods is the ODE-RNN (Rubanova et al., 2019; Chen et al., 2018), which is an RNN based on ordinary differential equations, which allow time to be treated as continuous instead of discrete. Although this architecture seems beneficial for our purposes, it comes at the cost of altering of the training process as opposed to using a standard RNN, in which batched training does not come straightforward. Moreover, the benefits of the ODE-RNN mostly come forward if the sampled input data is sparse. In the task of transit detection, however, this is not the case as we generally have transit durations in the order of hours and an observation cadence in the order of minutes.

Unrelated to exoplanet science or light curves, several works investigate the use of RNNs for the detection of signals in time series. For example, LSTMs have been used to detect anomalies in time series in an unsupervised learning approach (Malhotra et al., 2015). The RNN is trained to predict subsequent steps of "normal" data, which is clear of anomalies, and a deviation from the predictions is considered as anomaly. The same approach can be used to detect collective anomalies (Bontemps et al., 2016), i.e. anomalies that cover multiple data points. Transit signals can be seen as collective anomalies in time series dominated by "normal" stellar activity. However, as noted by Cherdo et al. (2020), this "unsupervised" approach still requires knowing which light curves are clear of anomalies. We may never know for sure if a light curve is clear of transit signals, but in some cases we do know for sure whether a transit signal is present. Therefore, we expect a supervised learning approach to be better suited for our task. [TODO: confidence estimation]

[3] https://pytorch.org/docs/stable/generated/torch.nn.GRU.html

Chapter 3

Methodology

3.1 Network

The basis network architecture, training objective and procedure are described in the following sections. The main function of the network is to classify individual data points as signal or non-signal, where every data point that falls within the duration of a transit is considered as part of the signal. Extensions, such as the prediction of flux values within data gaps, are described in a separate sections, as these require slightly different network architectures and training. All of the following was implemented using PyTorch (version 1.8.1).

3.1.1 Architecture

The RNN forms the basis of the detection algorithm. As light curves may contain many data points, we avoid the "vanilla" RNN, which is known to suffer from vanishing gradients. Other recurrent cells considered were GRU and LSTM. We found GRU to produce better and more stable results than LSTM, which is why we used GRU in our final RNN design.

Both directions in time are relevant for the classification of a data point. For example, both ingress and egress are strong features of a transit signal. This means that at mid-transit, knowing information from both directions could make it easier to classify the data points, compared to only knowing the past. For this reason, we make use of a bidirectional RNN.

In order to classify each time step as signal or non-signal, the output of the network should be a single value at each time step. To achieve this, we apply fully connected (FC) layers to the outputs of the recurrent layers, which project the hidden representation at each time step down to a single node. The result is passed through the sigmoid activation function, so the outputs lie between 0 and 1. Between fully connected layers, the ReLU activation function is applied to introduce non-linearities.

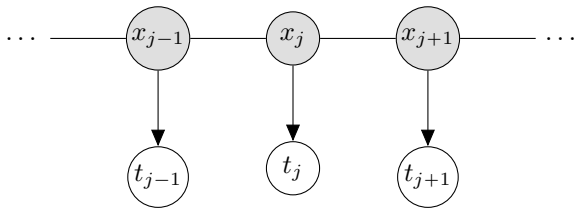
[TODO: describe number of nodes and layers, and tuning process]

3.1.2 Training

In a supervised fashion, we train the network to classify data points as signal or non-signal. This requires knowing the ground truth of the data. In our case we use simulated data, so we have access to the ground truth. Alternatively, one could use known (non-)transit signals in real-world light curves as ground-truth, or inject simulated transit signals in known transit-less light curves.

Typical light curves, however, contain too many data points to be used for training directly. First of all, the training procedure requires storing the gradients at each time step which would quickly overflow memory. Second of all, the RNN cannot well learn dependencies over thousands of time steps, so using full-length light curves for training would be unnecessarily inefficient. Therefore we assume that the data used for training consists of light curve segments, obtained by splitting the original light curve in segments of equal length N. N should be chosen larger than the typical transit duration ($\sim$6 hours) so sufficient background activity is included in the segments, but small enough to allow for efficient training. This work uses $N = 1500$, which corresponds to 50 hours for a cadence of 2 minutes, unless stated otherwise.

We want to model $p(T|X)$, where T represents the targets $\{t_j\}_{j=1}^{N}$ for each data point $\{x_j\}_{j=1}^{N}$ in a given light curve X. A target t_j is 1 if x_j is part of a transit signal, otherwise it is 0. We assume that the target t_j depends on all flux values through x_j, which is dependent on its neighbours $x_{j\pm1}$, which are dependent on their neighbours, and so on. This can be visualized by:



Since we observe X, the targets become separated in the graphical model. Therefore we model them as independent, given X, i.e.:

$$p(T|X) = p(t_1, \dots, t_N|X) = \prod_{j=1}^{N} p(t_j|X) \tag{3.1}$$

Our network outputs a value y_j for each x_j, using the function $Y = y(X, W)$, where W represents the weights of the network. Y represents the collection of outputs $\{y_j\}_{j=1}^N$. We interpret y_j as the conditional probability $y_j = p(t_j = 1|X, W)$, such that $p(t_j = 0|X, W)$ is given by $1 - y_j$. The conditional distribution of targets at time step j is then given by

$$p(t_j|X, W) = y_j^{t_j}(1 - y_j)^{1-t_j} \tag{3.2}$$

Given a set of independent observations $\{T_i\}_{i=1}^M$ for light curves $\{X_i\}_{i=1}^M$, we aim to maximize the likelihood $p(T|X, W)$ by updating the weights W of the network. Equivalently, we can maximize the log likelihood, which is given by

$$\log p(T|X, W) = \sum_{i=1}^{M} \log p(T_i|X_i, W) \tag{3.3}$$

$$= \sum_{i=1}^{M} \log \prod_{j=1}^{N} p(t_{ij}|X_i, W) \tag{3.4}$$

$$= \sum_{i=1}^{M} \sum_{j=1}^{N} \log[y_{ij}^{t_{ij}}(1 - y_{ij})^{1-t_{ij}}] \tag{3.5}$$

$$= \sum_{i=1}^{M} \sum_{j=1}^{N} t_{ij} \log y_{ij} + (1 - t_{ij}) \log(1 - y_{ij}), \tag{3.6}$$

where t_{ij} represents the target at time step j in light curve i and $y_{ij} = y(X_i, W)_j$.

To define our loss function, which we aim to minimize, we take the negative of $\log p(T|X, W)$, averaged over the number of light curves M and the number of data points per light curve N, i.e. for a single light curve i we have the loss:

$$\mathcal{L}_{\text{BCE},i} = \frac{1}{N} \sum_{j=1}^{N} l_{ij} + (1 - t_{ij}) \log(1 - y_{ij}), \tag{3.7}$$

where we used $l_{ij} = t_{ij} \log(y_{ij})$, and the subscript BCE because this is known as the binary cross-entropy loss. Using this loss function for a batched training with batch sizes of M, i.e. M light curve segments per batch, we get

$$\mathcal{L}_{\text{BCE}} = \frac{1}{M} \sum_{i=1}^{M} \mathcal{L}_{\text{BCE},i} \tag{3.8}$$

In terms of individual data points, however, we are dealing with a highly unbalanced data set, because there are many more non-signal points than signal points. In order to deal with unbalances, we use a different definition of l_{ij}:

$$l_{ij} = p_t w_{ij} t_{ij} \log(y_{ij}). \tag{3.9}$$

[TODO: references of loss weighting] Here p_t is the positive weight, which is used to tune the weight of positive samples, i.e. the data points which are part of a transit signal. If $p_t > 1$, then the predictions over signal data points will be given extra weight in the loss function. Increasing p_t will increase the recall, but will also lower the precision. In addition to the weight p_t, which is the same for all positive samples, we explore the

use transit-specific weighting. It might be that the data set contains more shallow than deep transit signals, in which case we may choose to increase the weight of data points belonging to deep transit signals in the data set, and vice versa. This can be done by letting the weight w_{ij} depend on the depth δ_{ij} of the transit signal at time step j. For example, we can set $w_{ij} = \delta_{ij}/\sigma_i$, where σ_i is the estimated level of white noise in light curve i, so w_{ij} depends on the relative transit depth rather than absolute transit depth. In this case, predictions over data points belonging to a transit signal which is twice as deep as another signal in the same light curve, will get twice the weight in the loss function. In case both p_t and w_{ij} are used, a small adjustment to p_t needs to made for maintaining consistent results between different choices for w_{ij}. The new weight for signal data points during training becomes $p'_t = p_t \cdot \sum_{ij} t_{ij} / \sum_{ij} w_{ij}$, where for $w_{ij} = t_{ij}$ we have $p'_t = p_t$.

[TODO: add weight decay]

3.2 Detection algorithm

In case the search for signals is directed towards monotransits, the network outputs for a given input light curve can directly be used for detection. For example, one can set a threshold on peaks in the PTS, which can be considered as candidate detections. In case we wish to search for repeating signals, the RNN outputs alone are not enough. In order to compare the performance of an RNN-based detection algorithm with conventional methods such as BLS, we also need to determine the period and epoch of the signal. Two different algorithms are tested to do so, which only make use of the RNN outputs, i.e. the PTS, for a given input light curve.

3.2.1 PTS-Peak

As proposed by Pearson et al. (2018), one could use the distances between peaks in the PTS to determine the period of a repeating signal. Here, we implement this idea and refer to the algorithm as PTS-Peak. In order for this algorithm to work well, we need to take into account the presence of potential false detections in the PTS, or multiple detections of signals belonging to different planets. Therefore we adopt several rules and filtering steps to ensure consistency between matched peaks and resulting parameter estimates. The algorithm used is as follows:

[TODO: algorithm description]

3.2.2 PTS-Fold

It could be that individual transit events are missed by the RNN, or that peaks in the PTS are too weak to be taken into account by PTS-Peak. To be less dependent on distinguishable peaks in the PTS, but more on overall response to transit signals, we define another algorithm which we refer to as PTS-Fold. This algorithm is similar to many detection algorithms (e.g. phase dispersion minimization or BLS) in that it folds the input time series over a set of trial periods. However, whereas other algorithms fold the raw light curve over trial periods, PTS-Fold only folds the PTS. Since each value the PTS indicates the extend to which a transit signal might be present at the corresponding time step, we can efficiently compute a candidate detection score for repeating signals by aggregating overlapping data points in the folded PTS. For clarification, we describe the algorithm in steps in the following:

[TODO: algorithm description]

3.3 Handling data gaps

If we naively pass a light curve with missing values through the RNN, then varying time intervals will be ignored and false detections could be triggered around gaps. One way to deal with this problem is to feed the time differences between measurements to the RNN. However, this extra stream of data might increase the chance of overfitting [TODO: elaborate]. Other methods include data imputation, for example by filling missing data with linearly interpolated values.

Since the RNN offers the possibility to make predictions at every time step, we explore the ability of the RNN to predict missing flux values when applied to a light curve with gaps. In other words, we extend the RNN and train it to predict subsequent flux values, in parallel to classifying data points as signal or non-signal. To do so, we apply a second FC-network to the outputs of the recurrent layers, to produce an output $\hat{x}_{ij}$ at each data point x_{ij}. We add an additional loss term, so the the generative network is trained to improve its predictions such that $\hat{x}_{ij}$ gets closer to x_{ij}:

$$\mathcal{L}_{\text{MSE},i} = \frac{1}{N} \sum_{j=0}^{N-1} (x_{ij} - \hat{x}_{ij})^2. \tag{3.10}$$

The loss over a light curve i for the generative RNN then becomes:

$$\mathcal{L}_i = \mathcal{L}_{\text{BCE},i} + \lambda \mathcal{L}_{\text{MSE},i} \tag{3.11}$$

[TODO: add weight decay]
When this network is applied to a light curve with missing data, we can replace a missing value $x_{ij} = \text{NaN}$ with the predicted value $\hat{x}_{ij}$ and use it as input to the network instead.

3.4 Increasing interpretability with confidence estimation

Although the network outputs are conveniently bounded between 0 and 1, they should not be confused with probabilities. For example, when two peaks in the PTS have the same height, we cannot say that they are equally likely to correspond to true signals. This is because the input data might be slightly different than the data seen during training, which could cause the outputs of the RNN to be less reliable. In addition to the standard outputs, it would therefore be helpful to have an estimation of confidence over the predictions to know when the network is more certain of its predictions.

To this end, we make use of a simple approach proposed by DeVries and Taylor (2018). We extend the network from 3.1.1 by applying additional FC-layers to the outputs of the recurrent cells, which output a scalar c_{ij} for each time step j in light curve i. This value, which is mapped between 0 and 1 by the sigmoid function, represents the confidence of the network over the prediction y_{ij}. Since c_{ij} is still the output of a neural network, it should not be interpreted as confidence in the statistical sense, but rather as indication of confidence relative to other predictions.

c_{ij} is used in the loss function by replacing y_{ij} with $y'_{ij} = c_{ij} y_{ij} + (1 - c_{ij} t_{ij})$ in Equation **??**. This means that for low confidence, i.e. $c_{ij} \approx 0$, we get that the network prediction during training becomes the same as the target, $y'_{ij} \approx t_{ij}$. For high confidence, $c_{ij} \approx 1$, the network uses its own prediction in the loss function, $y'_{ij} \approx y_{ij}$. To prevent the network from always returning low confidence values, an additional term is added to the loss function:

$$\mathcal{L}_{\text{conf},i} = \frac{1}{N} \sum_{j=0}^{N-1} -\log(c_{ij}), \tag{3.12}$$

So the loss over a given light curve i for the confidence RNN becomes:

$$\mathcal{L}_i = \mathcal{L}_{\text{BCE},i} + \lambda \mathcal{L}_{\text{conf},i}, \tag{3.13}$$

where λ is used as a weighting parameter. In line with DeVries and Taylor (2018), we adopt a budget parameter β which is used during training to adjust λ. If $\mathcal{L}_{\text{conf}} > \beta$, then we increase λ, and if $\mathcal{L}_{\text{conf}} < \beta$, then we decrease λ. Lastly, only for half the samples we use y'_{ij} instead of y_{ij}, in order to encourage the network more to learn meaningful decision boundaries.

3.5 Separating signals from different planets

Another way to increase the interpretability over the network outputs is to utilize the network's representations of each data point. This comes in helpful if there are only a few potential transit signals in a given light curve. Based on the PTS only, we cannot tell whether the individual signals appear different from each other. The detection algorithm might thus match the wrong signals together, leading to an incorrect detection which may been prevented if signal shapes are taken into account.

To illustrate how one could use an RNN and still take into account different signal shapes, we extend the network to learn representations of signals, such that signals from different planets are represented differently. The network from 3.1.1i s extended with FC-layers layers which are applied to the outputs of the recurrent cells. These layers project the representations of each time step down to D dimensions into a vector R_{ij} for time step j in light curve i. For different candidate transit signals, say, A and B, covering time steps $\{k, \ldots, l\}$ and $\{m, \ldots, n\}$ respectively, the network outputs representations $\{R_{ik}, \ldots, R_{il}\}$ and $\{R_{im}, \ldots, R_{in}\}$. We can average both sets of representations to obtain R_{A} and R_{B}, which correspond to the aggregated representations of signal A and signal B respectively.

In the supervised learning approach, we know whether A and B are signals caused by the same planet or not. Assuming that both A and B are true detections, we call A and B a positive pair if they are caused by the same planet, and a negative pair if they are signals from different planets. Actually, the network outputs a representation for each time step, regardless of whether a signal was detected at that time step. Therefore, R_{A} and R_{B} may as well be the aggregated representations of any two disjunct parts of the input light curve. A

and B are thus also called a positive pair if they both cover a non-signal part of the light curve, and they are also called a negative pair if only one of the two corresponds to a transit signal and the other does not.

The network is trained to represent positive pairs similarly, and negative pairs differently. This is achieved by adding a score of similarity between R_{A} and R_{B} to the loss function. The score of similarity used in this work is the cosine of the angle $\theta_{\mathrm{A,B}}$ between R_{A} and R_{B}: [TODO: add cosine similarity source]

$$\mathrm{sim}(\mathrm{A},\mathrm{B}) = \frac{R_{\mathrm{A}} \cdot R_{\mathrm{B}}}{\|R_{\mathrm{A}}\|\|R_{\mathrm{B}}\|} = \cos(\theta_{\mathrm{A,B}}). \tag{3.14}$$

For smaller angles between the representation vectors, A and B are considered to be more similar.

During training, we present the network with pairs of light curves with the same stellar background, and pre-select a region A from the first light curve and a region B from the second. These could be true transit signals, but could also be background noise. The parameter $p_{\mathrm{A,B}}$ indicates whether A and B are a positive or a negative pair. The network is then trained to minimize the representation loss term given by

$$\mathcal{L}_{\mathrm{repr},j} = (\mathrm{sim}(\mathrm{A},\mathrm{B}) \cdot (1 - 2 \cdot p_{\mathrm{A,B}}) + 1)/2. \tag{3.15}$$

The factor including $p_{\mathrm{A,B}}$ within brackets is to ensure that a positive pair with a high similarity score results in a low loss, a negative pair with high similarity results in high loss, and vice versa. The addition of 1 and subsequent division by 2 is to ensure that this loss term always has a value between 0 and 1. The combined loss for a light curve pair i is then

$$\mathcal{L}_i = \mathcal{L}_{\mathrm{BCE},i} + \lambda \mathcal{L}_{\mathrm{repr},i}, \tag{3.16}$$

where λ is used to weigh the representation term. In this case $\mathcal{L}_{\mathrm{BCE},i}$ is the average of the BCE loss term over both light curves in the input pair.

3.6 Data simulations

For the development and evaluation of our RNN-based algorithm we used simulated data. This is because simulated data comes with an absolute ground-truth of the hidden signals, and for the development of the algorithm it showed useful to be in control over the parameters of the input light curves. Two sources of data were used. First, we made use of a light curve simulator that was specifically designed for this work. We refer to this simulator and its produced data as LCSim. Second, we made use of simulated light curves in the Lilith-4 data set, which was produced by the Lilith data simulator of the TESS pipeline. The following subsections describe the details of the simulations and some general preprocessing of the data.

3.6.1 LCSim

In line with TESS data, we adopt a 2-minute cadence for all simulations in this work. The simulator, made available here[1], may also be used to generate 30-minute cadence light curves, to obtain light curves that are closer to Kepler data.

Stellar variability is simulated using Gaussian process (GPs). GPs have been used several times in literature to account for background activity in light curves, both for characterizing and simulating stellar activity (Barros et al., 2020; Zucker and Giryes, 2018). GPs are defined by a mean and covariance function, or kernel. In our case, the mean function is always 1, so the behaviour is fully determined by the kernel, which should therefore hold all of the star's relevant properties. An instance of the function representing stellar variability can be obtained by sampling from the multi-dimensional Gaussian distribution that is defined by the mean and covariance function. However, if we use the same kernel for each sample, then each light curve have the same underlying stellar properties. Therefore, we construct a kernel for each light curve. For the construction of kernels that are able to simulate quasi-periodic behaviour as observed in stars, we make use of the Python library `celerite` (Foreman-Mackey et al., 2017). `celerite` has built-in kernels to simulate stellar granulation and rotation modulation. [TODO: briefly describe parameters and what they mean]

To simulate photon noise, we simply sample from a Gaussian distribution $\epsilon_{ij} \sim \mathcal{N}(0, \sigma_i)$ and add ϵ_{ij} to the corresponding flux at time step j in light curve i. The standard deviation σ_i defines the level of time independent noise in the light curve, and is specified for each light curve separately.

For the simulation of transit signals, we made use of the Python library `batman` (Kreidberg, 2015), which is based on the equations from Mandel and Agol (2002) which describe the physics of transits. Several parameters need to be specified to obtain the transit signals of a single planets, see Table **??**. [TODO: add table of parameter distributions]. We approximate the stellar limb darkening effect with a quadratic function, parametrized by u_1 and u_2. [TODO: give quadratic function, and constraint sampling distributions]. We constrained the orbital

[1] https://github.com/ykerus/transit-detection-rnn15

period P of a planet and its semi-major axis a according to Kepler's Third Law (see Section **??**). However, in order to do so, the stellar mass M and radius R also need to specified so the result is compatible with the input requirements of `batman`. The orbital inclination i is assumed to be 90° for all planets we simulate, to avoid problems with non-existent transit signals, or transit depths that are more difficult to predict. Basically, setting $i = 90^\circ$ ensures that a simulated planet moves in front of the stellar disk during transit.

In case a light curve is required to have transit signals from two different planets, the transit simulator is simply called twice with different parameters for the planets while maintaining all the parameters belonging to the star. The result might be that the two planets coincidentally have the same distance from their host star, which would be unrealistic, but does not pose a problem for the aims of this thesis. Overlapping transit signals, on the other hand, could make the process of developing and evaluating our detection algorithm more difficult. If overlapping signals have been found, one needs to make sure which signal triggered a detection: it could be one of the two, or both. Since overlapping signals are far less common in real-world data than non-overlapping signals, we only simulate non-overlapping transit signals in this work to avoid confusion.

[TODO: describe parameter sampling] [TODO: describe data sets used in experiments and preprocessing]

3.6.2 Lilith-4

The Lilith-4 data set comprises four sectors of simulated TESS data, and takes into account readout errors, spacecraft jitter, focus errors, diffuse light, cosmic rays, stellar variability, transiting exoplanets, eclipsing binary stars, and more (Osborn et al., 2020). We use this data for several reasons. Firstly, it functions as a test for both our algorithm and the LCSim simulator. Unexpected results can lead us to believe that either our algorithm is too dependent on the data that is used, or that LCSim data is too unrealistic. Secondly, it allows for the evaluation of certain preprocessing steps in combination with our algorithm, that would be necessary in the case of using real-world data. Lilith-4 also includes information about the pointing of the telescope, the centroid data, which can be used by our algorithm to potentially benefit from.

Chapter 4

Experiments and results

4.1 Preprocessing and performance

Preprocessing is a vital step of the algorithm, so we first evaluate the effect of different preprocessing steps on the performance of the RNN. Often in classification tasks, the accuracy is used as a measure of the model's performance. However, since we are dealing with an imbalanced data set, the accuracy might give a wrong impression, because classifying each data point as non-signal would already result in high accuracy. Therefore we primarily look at the precision and recall, which are given by:

$$\text{precision} = \frac{tp}{tp + fp} \tag{4.1}$$

$$\text{recall} = \frac{tp}{tp + fn}, \tag{4.2}$$

where tp is the number of true positive classifications, fp the number of false positives and fn the number of false negatives. Since precision and recall, and accuracy for that matter, are dependent on the classification threshold, we use area under the precision-recall (PR) curve as a measure that is independent of the threshold. The area under the PR-curve is also known as average precision (AP).

In the following, we assume all "raw" light curves to be median normalized, and each light curve segment to be obtained from splitting this median normalized light curve into parts. The first preprocessing step we take is centering the light curve segments by subtracting 1. Additionally, we investigate the effect of scaling the light curves by their estimated noise level σ, so the network performance becomes less dependent on this noise and instead more on the transit depth relative to this noise. We will refer to this as sigma scaling.

Still, the range of flux values might greatly vary between two separate light curve segments, as illustrated in Figure 4.1. This is understood by realizing that the original light curve may contain large-scale fluctuations due to stellar rotation modulation. The network might interpret these range differences as indicators of the presence of a transit signal, in particular if there is little data available for a certain range of flux values. This is not what we want, because a transit event can occur at any given time, independent from the activity on the stellar surface. Applying median normalization again to the segments is not a solution, because then we prevent the network to learn to deal with input ranges it would normally encounter in full-length light curves. An alternative which avoids this problem, is to take the derivative of the input light curve and feed it to the network instead of the absolute values.

Figure 4.2 shows the effect of these different preprocessing steps on the AP of the RNN in the task of classifying individual data points as signal or non-signal. In this figure, we can observe a small difference between using no scaling and using sigma scaling, in favor of no scaling. We found that using derivatives instead of absolutes prevents the network from learning[1].

To deal with gaps, we compare two gap-filling approaches and the use of the generative RNN. The gap-filling approaches are simple and are not aimed to reconstruct the original light curve, but rather produce an input that the RNN can handle well. Figure 4.3 visualizes how each approach works, and Figure 4.4 visualizes the predictions of the generative RNN applied to the same example. Since we use a bidirectional RNN, we get two streams of predictions. We can see from the figures that this creates some unrealistic artefacts in the predicted curve, which is likely the cause for the bumped peak in the PTS of the generative RNN. Therefore unexpectedly, we found that the generative network slightly outperformed the other two approaches when applied to a larger data set, especially for the shallowest transit signals. The results are plotted in Figure 4.5. The second best

[1] In an earlier iteration of this work when less complex data was used (e.g. with less stellar variability), the derivative inputs worked well. Only with increased complexity the model started to fail.

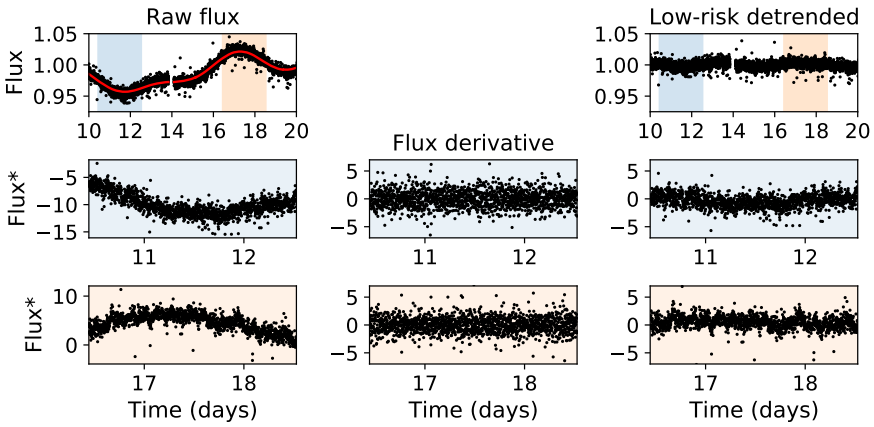


Figure 4.1: (Left) Two segments are taken from a "raw" median normalized light curve, which have considerably different flux ranges after preprocessing. (Middle) Taking the derivative solves the problem of different input ranges, while conserving all relevant information. (Right) With loss of information, one could carefully remove large-scale trends in the light curve, using the red trend line in the top-left panel, to achieve a similar result.

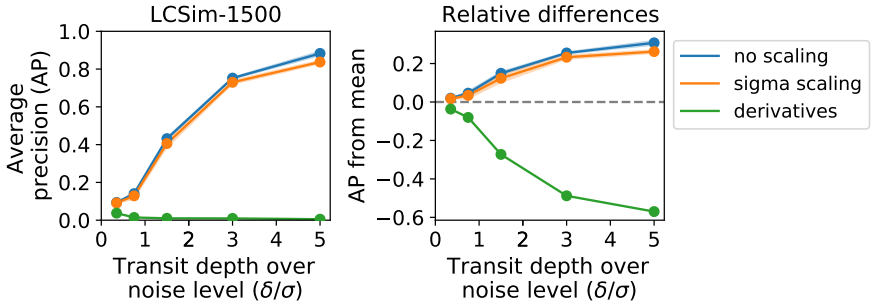


Figure 4.2: The average precision of the RNN evaluated at different transit depth bins for different basic preprocessing steps. At larger depths, the AP is expected to be higher. The right figure shows the deviations from the mean of all curves. Filled regions, which are narrow in this figure, show one standard deviation over the results of three differently initialized networks applied to the LCSim-1500 test split.

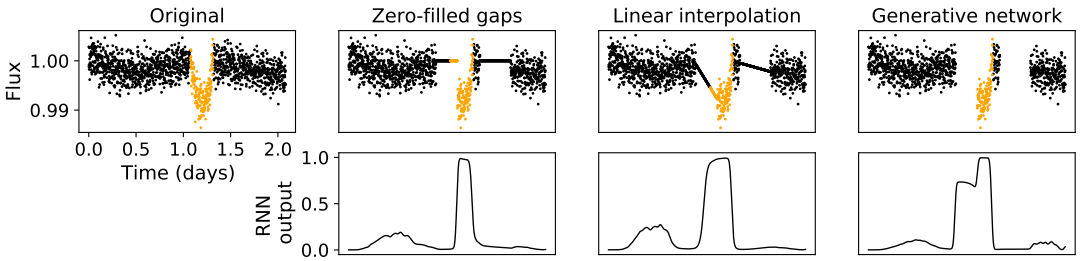


Figure 4.3: An example from the LCSim-1500-Gap validation split, with different approaches to dealing with gaps and the corresponding PTS of the RNN. The signal data points are indicated by orange. Zero-filling in this example translates to one-filling, because the light curve segments are centered around zero before passing them to the RNN.

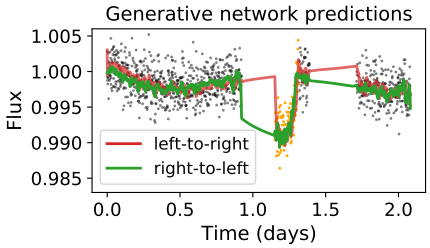


Figure 4.4: The predicted flux values in red and green from the bidirectional generative RNN corresponding to the right-most figure in Figure 4.3. The network seems to follow an expected trend in both directions, however, the gaps are not filled in a satisfactory manner.

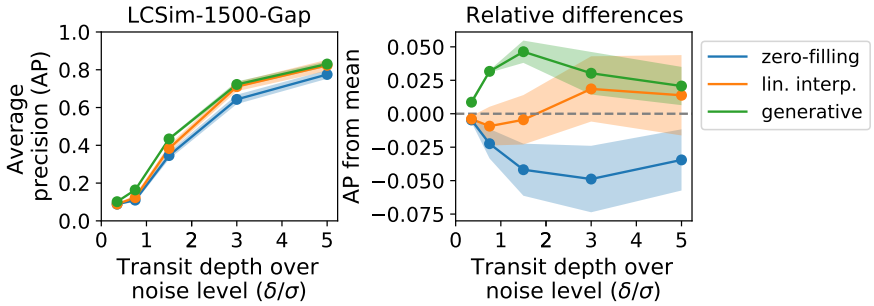


Figure 4.5: The average precision for varying transit depths, with filled regions indicating the standard deviation of three runs on the test split of LCSim-1500-Gap with differently initialized networks. The comparison includes different methods to deal with data gaps: filling gaps with zeros, linear interpolation, or using predictions from the generative RNN.

approach tested was linear interpolation, and the worst was zero-filling. In terms of efficiency, however, linear interpolation is preferred over the generative RNN. This is because during training, the generative RNN needs to evaluate at each time step whether the data point is missing, and if so, replace it with its own prediction before it proceeds to the next time step. This, in combination with the fact that we could not use pre-implemented RNN architectures in PyTorch for this purpose, resulted in an increase of training time of more than five times compared to the standard RNN applied to linearly interpolated data.

In Figure 4.6, we evaluate a subset of the same preprocessing steps applied to the Lilith-1500 data set. In the shallow transit range, each network's AP lies relatively close to each other. Towards the deepest transits, it seems that sigma scaling is beneficial for this data set. From the figure, however, it also seems that zero-filling is the better approach to deal with data gaps, in contrast to our earlier results.

Lastly, we evaluate additional preprocessing steps applied to Lilith light curves. First of all, to address the problem illustrated in Figure 4.3, we detrend the original light curves (before segmenting) with a filter that only removes the largest trends. We refer to this as low-risk detrending (LRD). For this, a Savitzky-Golay filter was used with a polynomial order of 2 and a window length of 2881, i.e. ∼4 days (see Hippke et al. (2019) for

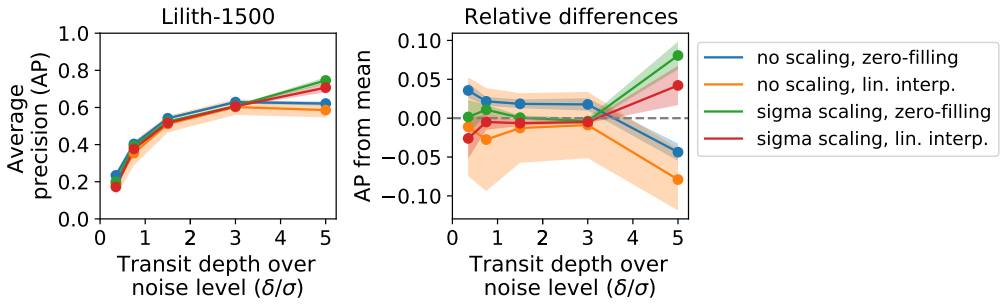


Figure 4.6: The effect of different basic preprocessing steps on the performance of the RNN applied to Lilith light curve segments. Filled regions indicate the standard deviation over three runs using differently initialized models that were applied to the test split after training.

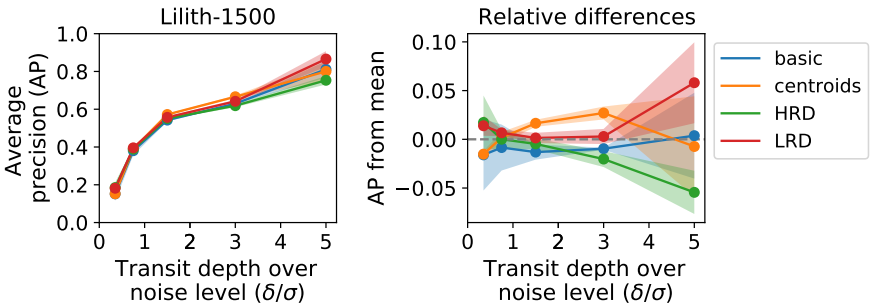


Figure 4.7: [TODO: outliers] The effect of high- and low-risk detrending, and centroid inputs on the AP of the RNN, in comparison with only using basic preprocessing, i.e. sigma scaling, linearly interpolating data gaps. The results are obtained on the test set of Lilith-1500, with standard deviations over three runs indicated by the filled regions.

details on the Savitzky-Golay filter). We also test the effect of detrending the original light curve with a filter that would more commonly be used in combination with other search algorithms. We refer to this as high-risk detrending (HRD). For this, a time-windowed sliding median filter was used with a window length of 12 hours. Finally, since Lilith provides centroid data, we investigate the effect of using the centroids as additional input to the network at each time step. The results shown in Figure 4.7 suggest that both LRD and centroid inputs are beneficial to the network, although there is some overlap between different runs. In the mid-range of transit depths, LRD and centroid inputs seem to be consistently better than other methods. It should be marked that using an HRD filter did not improve the results, especially in the deeper transit range, which underlines our claim that the RNN does not require its inputs to be cleared of short scale time dependent noise in order to do its task.

In each of the following experiments, we make use of sigma scaling and linear interpolation to deal with gaps.

4.2 Model comparison

Although the task of our network is about the classification of individual data points, it is worth comparing the RNN's performance with the more commonly used CNN in the task for which the CNN is generally applied. The CNN proposed by Pearson et al. (2018) was evaluated on its accuracy of classifying light curve segments as signal or non-signal, which is a more balanced problem than ours. To obtain a single classification of the RNN for an input light curve segment, we have two options. Either we take the maximum value of the corresponding PTS as prediction and make no changes to the network or training, or we train the network to output its classification over the entire segment only at the last time step. The latter approach, we refer to as "naive" because we only provide a sparse learning signal to the network, even though we have the option to provide a learning signal at every time step.

Table 4.2 shows that the RNN (bi-GRU-1, i.e. single-layer bidirectional GRU) outperforms the CNN that was used in the comparison, even though it was not trained for this task. It must be noted however, that the performance of the CNN depends on many hyperparameters (number of layers and channels, kernel sizes, pooling, strides), and the one used here differs from the one used by Pearson et al. (2018). Each network architecture was obtained by changing the hyperparameters in small steps until no large improvements were observed on the validation accuracy [TODO: appendix for tuning process]. As more consistent baseline than CNN, we also include the MLP in the comparison, which depends on fewer hyperparameters. Other models, such as two-layer GRU or LSTM are also included in the comparison to motivate our choice for the bi-GRU-1. The results show that the preferred model copes well with longer input sequences, whereas the CNN is affected more strongly by the increase of data points. This is reassuring, as our network should be able to cope with long input sequences which contain more distracting background patterns than the smaller inputs the CNN is generally applied to.

To further explore the potential of the RNN in the task of transit detection at the level of individual data points, we evaluate different weighting schemes to deal with the imbalances in the data for this task. First, we vary the positive weight p_t in Figure 4.8. The results suggest that setting a positive weight $p_t > 1$, is beneficial in terms of the tradeoff that can be made between precision and recall. The weight $p_t = 12.6$ would effectively fully balance the data, because there are about 12.6 times more non-signal data points in LCSim-1500 than there are signal data points. However, as is clear from the figure, a weight this high would also require us to set a high classification threshold to obtain reasonable precision (e.g. > 0.5). Therefore a lower weight is preferred,

Model	Segment accuracy ($N = 500$)	Segment accuracy ($N = 1500$)	Data point accuracy ($N = 1500$)
MLP	0.6846 ± 0.0007	0.60 ± 0.01	
CNN	0.692 ± 0.008	0.61 ± 0.03	
GRU-1 (Naive)	0.64 ± 0.10	0.507 ± 0.006	
GRU-1	0.68 ± 0.02	0.64 ± 0.01	0.935 ± 0.001
bi-GRU-1	**0.719** ± 0.003	**0.681** ± 0.004	**0.9399** ± 0.0003
bi-GRU-2	0.712 ± 0.005	0.66 ± 0.03	0.938 ± 0.003
bi-LSTM-1	0.708 ± 0.005	0.59 ± 0.04	0.931 ± 0.003

Table 4.1: The comparison of different models applied to LCSim light curve segments of varying length, i.e. $N = 500$ (∼16.7 hours) and $N = 1500$ (50 hours). The center columns show the test accuracy of classifying entire segments as signal or non-signal, similar to Pearson et al. (2018). The last column shows the accuracy of classifying individual data points as signal or non-signal, which is high by default because of the large data imbalance. The classification threshold in each case was set at 0.5, meaning that a segment or data point is classified as signal if the network output is larger than 0.5. Each RNN uses the maximum of the PTS for the classification of the entire light curve segment. The values given are the means and standard deviations over three independent runs of each model. [TODO: network architectures in appendix].

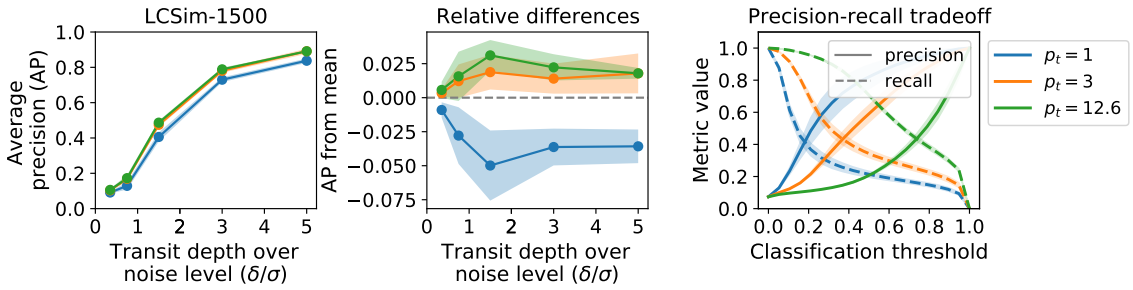


Figure 4.8: The effect of different positive weights on the AP evaluated on the test split of LCSim-1500, and the corresponding shifts in precision and recall versus the classification threshold. To clarify, a threshold of 0.25 would mean that data points are classified as signal for which the RNN outputs are above 0.25. Filled regions indicate the standard deviation over the results of three independent runs.

e.g. $p_t = 3$, because it still increases the AP compared to $p_t = 1$, and it allows for setting a more intuitive classification threshold compared to $p_t = 12.6$.

Next, we evaluate the use of the transit-specific weighting parameter w_{ij}. In the simple case we have $w_{ij} = 1$, meaning that all signal data points are weighted the same, regardless of their corresponding transit depths. If the data set contains more shallow transit signals, then using the same weight for all transits might cause the network to become biased toward only detecting shallow signals, even though we should expect the network to perform well also in the simpler case of deep transit signals. Therefore, we evaluate setting $w_{ij} = \delta_{ij}/\sigma_i$, which sets a weight on transit samples that grows linearly with the corresponding transit depth relative to the white noise. Additionally, we set $w_{ij} = \sqrt{\delta_{ij}/\sigma_i}$, which has a similar, but less strong effect. Figure 4.9 shows, as expected, that we can indeed tune the focus of the network towards specific transit samples. In our case, $w_{ij} = \sqrt{\delta_{ij}/\sigma_i}$ provides a good balance in classifying transit signals of varying depths. Similar results are observed for the different weighting schemes applied to Lilith data in Figure 4.10.

4.3 Monotransit retrieval

We apply the RNN trained with weighting parameters $p_t = 3$ and $w_{ij} = \sqrt{\delta_{ij}/\sigma_i}$, to full-length light curves and evaluate its ability of retrieving single transit events. A detection is defined by a peak in the PTS for a given input light curve above a given threshold, which visualized in Figure 4.11. In this figure where we also illustrate the use of the confidence outputs produced by the network we refer to as Conf-RNN.

As baseline we use the monotransit detection algorithm used by Foreman-Mackey et al. (2016), which is based on a box function that is fitted to the data at different points in time. The only adjustment we made was that instead of searching for transits of a single duration, we combine the detection results from using trial durations ranging from 1 to 13 hours in steps of 1 hour. The output of the algorithm is a time series of SNR values of the best-fitting box at each time step, which we will refer to as SNR-TS. We set a detection threshold

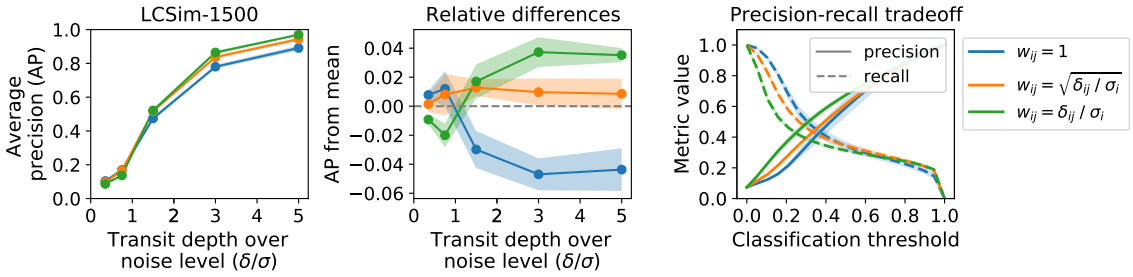


Figure 4.9: The effect of different transit-specific weights on the AP evaluated on the test split of LCSim-1500. No large shifts in precision or recall over the classification threshold are observed.

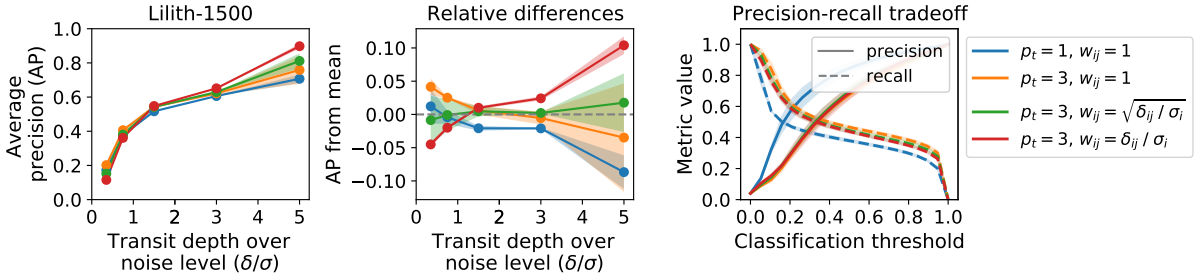


Figure 4.10: The effect of different weighting schemes on the AP evaluated on the test split of Lilith-1500. The results are similar to those obtained using LCSim-1500.

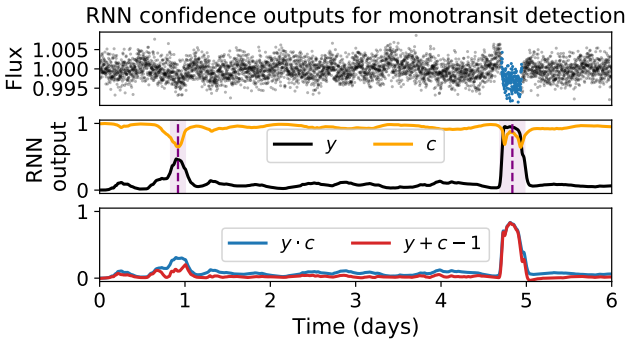


Figure 4.11: An example of how the RNN outputs could be used for monotransit detection. The standard outputs y define the PTS for the given input, which can directly be used the set a threshold on candidate detections (e.g. purple shows candidate detections for $y > 0.25$). In case the confidence RNN is used we also get c, which we use in two ways in combination with y to define an alternative PTS with the aim of reducing peaks of which the RNN is less certain. These curves, as shown in the bottom panel, can similarly be used to set detection thresholds.

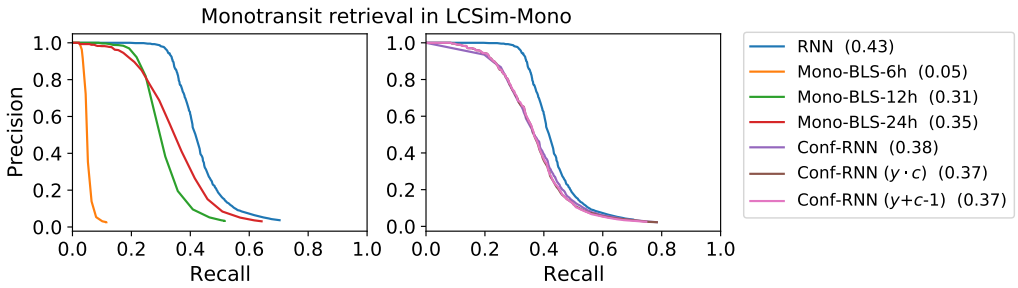


Figure 4.12: Precision-recall (PR) curves of different methods in the task of monotransit detection, divided over two figures to avoid clutter. The area under the curve, or average precision (AP), is given between brackets for each method. The RNN is the single-layer bi-GRU which only outputs the PTS for a given input light curve. Conf-RNN is a separately trained network which outputs confidence values in addition to the standard PTS, which performed worse than the standard RNN, even if the confidence values are not used.

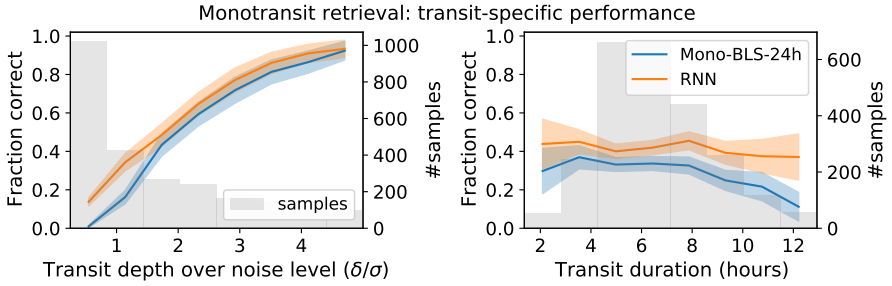


Figure 4.13: The ability of the presented methods to retrieve specific transit samples, e.g. if the fraction correct is 0.5, then half of the planets were found for the corresponding parameter bin. Underlying data distributions are shown in gray which also indicate the bins used in the evaluation. The detection threshold for both methods was set closest to a corresponding detection precision of 0.5. Filled regions shows the Wilson score interval (Wilson, 1927) which approximates the 95% confidence interval of the performance within each bin.

on the SNR above the noise floor in the SNR-TS, which is computed using a sliding median filter. Before applying this method, we detrend the input light curves with a sliding median filter with varying window sizes, i.e. 6, 12, and 24 hours. Since the algorithm is similar to BLS, we refer to it as Mono-BLS-{6h, 12h, 24h}, where the extension (e.g. "6h") refers to the window length of the detrending filter.

Figure 4.12 shows the PR-curves of the different detection algorithms. In this experiment, the RNN outperformed Mono-BLS, which is also made clear in Figure 4.13 where we take a closer look at the performance of both methods for specific transit samples. Furthermore, the Conf-RNN performed similar to Mono-BLS and thus worse than the standard RNN, even the Conf-RNN that did not make use of its confidence outputs. Table 4.3 presents the exact number of planets retrieved by the best performing methods at a precision of 0.5, including the number of planets found by one method that were not found by the other.

In terms of computational efficiency, the RNN greatly outperformed (our implementation of) Mono-BLS. For the search for monotransits in 5000 light curves, Mono-BLS took about 10 hours on a CPU, while the RNN only took a few minutes on a GPU. If no GPU is available, the RNN would have taken about 2 hours for this task. For Mono-BLS, the computation time could probably be reduced to a few hours by using less trial durations, using a lower time resolution for the search, or using a faster programming language for the implementation. However, since the RNN can easily be run on a GPU with a library such as PyTorch, and pre-implemented algorithms such as (Mono-)BLS do not naturally come with GPU compatibility, the RNN is the preferred choice in terms of efficiency.

4.4 Single planet retrieval

For the evaluation of the ability of the RNN-based algorithms to retrieve single planets with multiple transit signals, we apply the RNN from previous sections to light curves in the LCSim-Single data set. Subsequently, we use PTS-Fold and PTS-Peak to determine the period and epoch of a maximum of three candidate signals per light curve. Both algorithm give each candidate a corresponding score, as described in Section 3.2, which is used to set a detection threshold. A detection is counted as correct if the estimated period is correct with a 1% error margin, and the estimated epoch is within the duration of the first transit in the light curve.

	RNN		Mono-BLS-24h	
	1042	(1.00, 0.42)	818	(1.00, 0.33)
not RNN	-		74	(0.09, 0.03)
not Mono-BLS-24h	298	(0.29, 0.12)	-	

Table 4.2: The absolute and relative number of correct detections in the task of retrieving monotransits, where the detection thresholds are set closest to a corresponding detection precision of 0.5. An element, say with labels "A" in the corresponding column and "not B" in the row, corresponds to the number of detected planets by A that are not detected by B. Values between brackets show the fraction with respect to the total number of detections of A, and the fraction with respect to the total amount of planets in the data (i.e. 2500) respectively.

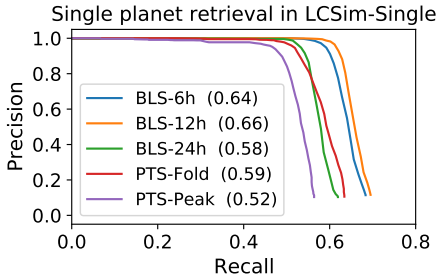


Figure 4.14: Precision-recall curves for different methods in the task of detecting repeating transit signals of a single planet. The average precision is given between brackets.

As baseline we use the standard BLS algorithm, as implemented in the `astropy` package[2]. We set a detection threshold on the SDE of each candidate in the BLS periodogram, and recursively apply the method to search for a maximum of three potential signals per light curve, each time masking the previous signal that was found. The algorithm is applied to detrended light curves using sliding median filters of varying lengths (6, 12, and 24 hours).

Figure 4.14 shows that a 12-hour window works best for detrending in combination with BLS for this task, which is in line with the results from Hippke et al. (2019). Moreover, the results show that the BLS algorithm outperformed the RNN-based detection algorithms in this experiment. Among the RNN-based algorithms tested, PTS-Fold performed best, which was as expected because it does not rely on distinct peaks in the PTS as opposed to PTS-Peak. The performance of the best performing algorithms can is further inspected in Figure 4.15. This figure shows, in line with our monotransit experiment, that towards larger periods, i.e. fewer transit signals, the performance of BLS drops faster than that of the RNN-based algorithm. This suggests that if there is periodicity in the signal, BLS remains dominant over the RNN. However, if the periodicity is lacking, the RNN is the preferred method. Table 4.4 presents the number of planets retrieved by the methods, including the number of planets that were not found by one method and not by the other.

The RNN-based algorithm required a longer computation time in this case than in the monotransit case. To obtain the PTS of each of the 5000 light curves still required only a few minutes on a GPU, but the determination of the periodicity of potential signals added more to the necessary computation time. PTS-Peak required only required 5 minutes on a CPU. PTS-Fold on the other hand, took about 1.5 hours. Still, PTS-Fold was faster than BLS with over 3 hours of computation time. The computation times are in line with the relative performances of each method in this task.

	PTS-Fold		PTS-Peak		BLS-12h	
	1480	(1.00, 0.59)	1333	(1.00, 0.53)	1655	(1.00, 0.66)
not PTS-Fold	-		7	(0.01, 0.00)	224	(0.14, 0.09)
not PTS-Peak	154	(0.10, 0.06)	-		342	(0.21, 0.14)
not BLS-12h	49	(0.03, 0.02)	20	(0.02, 0.01)	-	

Table 4.3: The absolute and relative number of correct detections in the task of retrieving planets with multiple transit signals, where the detection thresholds are set closest to a corresponding detection precision of 0.5. The data contained a total of 2500 planets.

[2] https://docs.astropy.org/en/stable/timeseries/bls.html

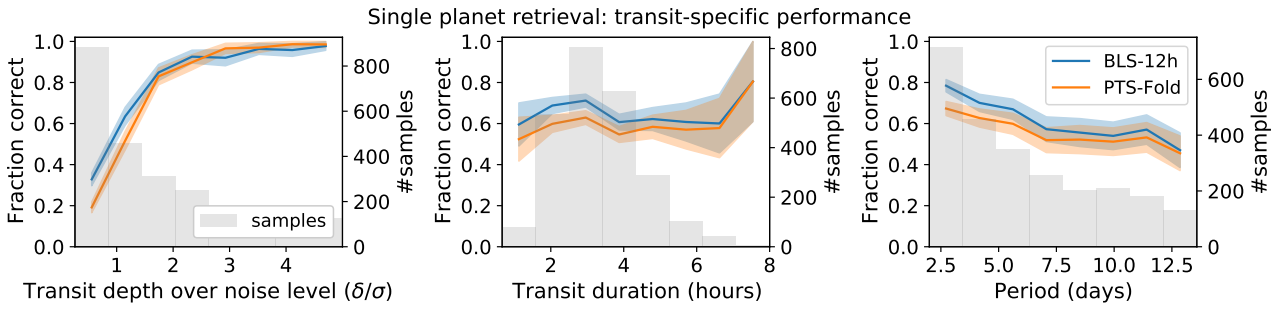


Figure 4.15: The ability of the presented methods to retrieve planets within the given parameter ranges. A detection threshold was set closest to a corresponding detection precision of 0.5. PTS-Peak is not shown because it behaves similarly but worse than PTS-Fold and would therefore only clutter the image. Filled regions show the Wilson score interval (Wilson, 1927), which approximates the 95% confidence interval of the performance per bin.

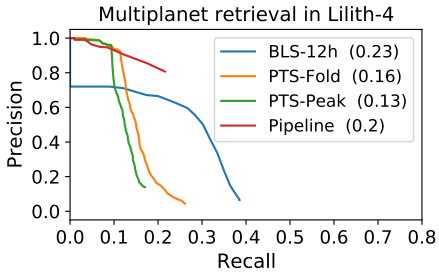


Figure 4.16: Precision-recall (PR) curves of different methods in the task of detecting repeating transit signals in light curves in Lilith-4. The PR-curve for the pipeline could not be drawn fully, due to limited available data.

4.5 Multiplanet retrieval

Similar to the single planet case, we compare the RNN-based detection algorithms with BLS in the task of retrieving planets with multiple transit signals. In this case we use Lilith data, which might contain transit signals from different planets within the same light curve. Again we restrict both methods to only output three candidate detections per light curve, to save on computation time.

In this experiment we also include the TESS pipeline detections for the given data set. However, since a threshold has already been set on the pipeline detections, we only have limited data, and the PR-curve for the pipeline therefore cannot be drawn fully. For the pipeline's PR-curve shown in Figure 4.16, we set a threshold on the SNR of the detections that are provided in the DV files. It can be seen from this figure, that the pipeline's PR-curve steadily decreases, suggesting that a high AP could be obtained if we had access to all the detections below the pre-set threshold. In contrast, the RNN-based algorithms drop down steeply. Nevertheless, the RNN-based methods seem to be able to maintain high precision for the detection of about 10% of all the planets. BLS on the other hand, starts off with relatively low precision, but is able to retrieve far more of the total amount planets. In terms of average precision, it therefore outperformed the other methods. Figure 4.17 shows the detection results over the distribution of transit samples for different parameters. We can see that each method behaves similarly, but at different levels of performance. Table **??** presents the number of planets retrieved by each method, including the number of planets that were found by one method and not by the other. Table **??** is provided to give a closer view on the ability of PTS-Fold and BLS-12h to retrieve multiple planets in a single light curve. From this table we can see that PTS-Fold is able to detect multiple planets in the same light curve, but there is room for improvement, compared to BLS-12h.

4.6 Success and failure cases

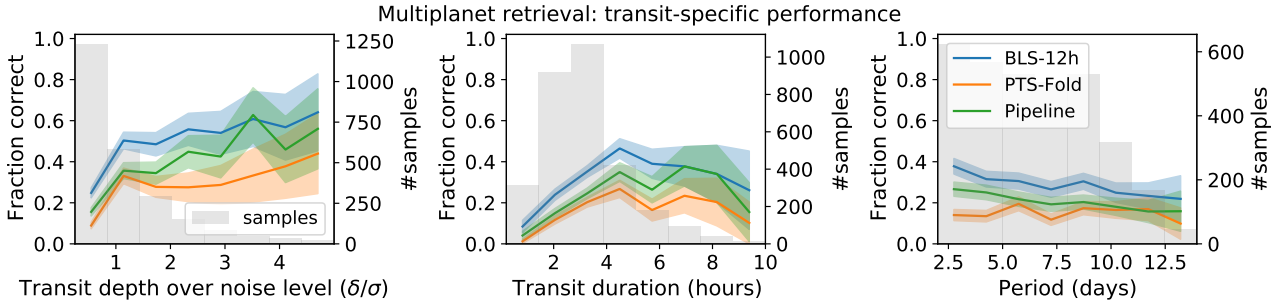


Figure 4.17: The ability of each method to retrieve planets in the given parameter ranges. A detection threshold is set closest to a corresponding detection precision of 0.5. Filled regions show the Wilson score interval Wilson (1927), which approximates the 95% confidence interval of the performance per bin.

	PTS-Fold		BLS-12h		Pipeline	
	493	(1.00, 0.15)	993	(1.00, 0.30)	710	(1.00, 0.22)
not PTS-Fold	-		522	(0.53, 0.16)	304	(0.43, 0.09)
not BLS-12h	22	(0.04, 0.01)	-		46	(0.06, 0.01)
not Pipeline	87	(0.18, 0.03)	329	(0.33, 0.10)	-	

Table 4.4: The absolute and relative number of correct detections in the task of retrieving planets with multiple transit signals, where the detection thresholds are set closest to a corresponding detection precision of 0.5. For the pipeline, the level of precision is in fact higher and the numbers presented here are relatively small, because we had no access to its lower confidence detections. The data contained a total of 3285 planets.

# planets (# light curves)	Method	Detected 1	2	3	4
1 (2130)	PTS-Fold	371			
	BLS-12h	666			
2 (474)	PTS-Fold	80	9		
	BLS-12h	120	65		
3 (57)	PTS-Fold	13	5	0	
	BLS-12h	8	20	6	
4 (9)	PTS-Fold	1	0	0	0
	BLS-12h	4	2	1	0

Table 4.5: The number of correctly detected planets per light curve with a given number of planets. For example, of the 57 light curves that contained 2 planets, PTS-Fold was able to retrieve 2 planets from 5 light curves, and only 1 planet from 13 light curves. The detection threshold was set closest to a corresponding detection precision of 0.5.

Chapter 5

Discussion

5.1 Implications of results

5.2 Limitations and suggestions

5.3 Future directions

Chapter 6

Conclusions

References

Megan Ansdell, Yani Ioannou, Hugh P Osborn, Michele Sasdelli, Jeffrey C Smith, Douglas Caldwell, Jon M Jenkins, Chedy Räissi, Daniel Angerhausen, et al. Scientific domain knowledge improves exoplanet transit classification with deep learning. *The Astrophysical journal letters*, 869(1):L7, 2018.

SCC Barros, O Demangeon, RF Díaz, J Cabrera, NC Santos, JP Faria, and F Pereira. Improving transit characterisation with gaussian process modelling of stellar variability. *Astronomy & Astrophysics*, 634:A75, 2020.

Ignacio Becker, Karim Pichara, Márcio Catelan, Pavlos Protopapas, Carlos Aguirre, and Fatemeh Nikzat. Scalable end-to-end recurrent neural network for variable star classification. *Monthly Notices of the Royal Astronomical Society*, 493(2):2981–2995, 2020.

Loic Bontemps, James McDermott, Nhien-An Le-Khac, et al. Collective anomaly detection based on long short-term memory recurrent neural networks. In *International Conference on Future Data and Security Engineering*, pages 141–152. Springer, 2016.

S Carpano, Suzanne Aigrain, and F Favata. Detecting planetary transits in the presence of stellar variability-optimal filtering and the use of colour information. *Astronomy & Astrophysics*, 401(2):743–753, 2003.

Joshua A Carter and Eric Agol. The quasiperiodic automated transit search algorithm. *The Astrophysical Journal*, 765(2):132, 2013.

Joseph H Catanzarite. Autovetter planet candidate catalog for q1-q17 data release 24. *Astronomy & Astrophysics*, 2015.

Joheen Chakraborty, Adam Wheeler, and David Kipping. Hundreds of new periodic signals detected in the first year of tess with the weirddetector. *Monthly Notices of the Royal Astronomical Society*, 499(3):4011–4023, 2020.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *arXiv preprint arXiv:1806.07366*, 2018.

Yann Cherdo, Paul de Kerret, and Renaud Pawlak. Training lstm for unsupervised anomaly detection without a priori knowledge. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4297–4301. IEEE, 2020.

Pattana Chintarungruangchai and Guey Jiang. Detecting Exoplanet Transits through Machine-learning Techniques with Convolutional Neural Networks. *Publications of the Astronomical Society of the Pacific*, 131 (1000):064502, 2019.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Jeffrey L Coughlin. Planet detection metrics: Robovetter completeness and effectiveness for data release 25. *Kepler Science Document KSCI-19114-002*, page 22, 2017.

Anne Dattilo, Andrew Vanderburg, Christopher J Shallue, Andrew W Mayo, Perry Berlind, Allyson Bieryla, Michael L Calkins, Gilbert A Esquerdo, Mark E Everett, Steve B Howell, et al. Identifying exoplanets with deep learning. ii. two new super-earths uncovered by a neural network in k2 data. *The Astronomical Journal*, 157(5):169, 2019.

Terrance DeVries and Graham W Taylor. Learning confidence for out-of-distribution detection in neural networks. *arXiv preprint arXiv:1802.04865*, 2018.

Debra A Fischer, Megan E Schwamb, Kevin Schawinski, Chris Lintott, John Brewer, Matt Giguere, Stuart Lynn, Michael Parrish, Thibault Sartori, Robert Simpson, et al. Planet hunters: the first two planet candidates identified by the public using the kepler public archive data. *Monthly Notices of the Royal Astronomical Society*, 419(4):2900–2911, 2012.

Daniel Foreman-Mackey, Benjamin T Montet, David W Hogg, Timothy D Morton, Dun Wang, and Bernhard Schölkopf. A systematic search for transiting planets in the K2 data. *The Astrophysical Journal*, 806(2):215, 2015.

Daniel Foreman-Mackey, Timothy D Morton, David W Hogg, Eric Agol, and Bernhard Schölkopf. The population of long-period transiting exoplanets. *The Astronomical Journal*, 152(6):206, 2016.

Daniel Foreman-Mackey, Eric Agol, Sivaram Ambikasaran, and Ruth Angus. Fast and scalable gaussian process modeling with applications to astronomical time series. *The Astronomical Journal*, 154(6):220, 2017.

Trisha A Hinners, Kevin Tat, and Rachel Thorp. Machine learning techniques for stellar light curve classification. *The Astronomical Journal*, 156(1):7, 2018.

Michael Hippke and René Heller. Optimized transit detection algorithm to search for periodic transits of small planets. *Astronomy & Astrophysics*, 623:A39, 2019.

Michael Hippke, Trevor J David, Gijs D Mulders, and René Heller. Wōtan: Comprehensive time-series detrending in python. *The Astronomical Journal*, 158(4):143, 2019.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Sara Jamal and Joshua S Bloom. On neural architectures for astronomical time-series classification with application to variable stars. *The Astrophysical Journal Supplement Series*, 250(2):30, 2020.

Jon M Jenkins. The impact of solar-like variability on the detectability of transiting terrestrial planets. *The Astrophysical Journal*, 575(1):493, 2002.

Jon M Jenkins, Joseph D Twicken, Sean McCauliff, Jennifer Campbell, Dwight Sanderfer, David Lung, Masoud Mansouri-Samani, Forrest Girouard, Peter Tenenbaum, Todd Klaus, et al. The TESS science processing operations center. In *Software and Cyberinfrastructure for Astronomy IV*, volume 9913, page 99133E. International Society for Optics and Photonics, 2016.

Jon M Jenkins, Peter Tenenbaum, Shawn Seader, Christopher J Burke, Sean D McCauliff, Jeffrey C Smith, Joseph D Twicken, and Hema Chandrasekaran. Kepler Data Processing Handbook: Transiting Planet Search. *ksci*, page 9, 2017.

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

Sebastiaan Koning, Caspar Greeven, and Eric Postma. Reducing artificial neural network complexity: A case study on exoplanet detection. *arXiv preprint arXiv:1902.10385*, 2019.

Geza Kovács, Shay Zucker, and Tsevi Mazeh. A box-fitting algorithm in the search for periodic transits. *Astronomy & Astrophysics*, 391(1):369–377, 2002.

Géza Kovács, Joel D Hartman, and Gáspár Á Bakos. Periodic transit and variability search with simultaneous systematics filtering: Is it worth it? *Astronomy & Astrophysics*, 585:A57, 2016.

Laura Kreidberg. batman: Basic transit model calculation in python. *Publications of the Astronomical Society of the Pacific*, 127(957):1161, 2015.

Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. Long short term memory networks for anomaly detection in time series. In *Proceedings*, volume 89, pages 89–94. Presses universitaires de Louvain, 2015.

Kaisey Mandel and Eric Agol. Analytic light curves for planetary transit searches. *The Astrophysical Journal Letters*, 580(2):L171, 2002.

Mario Morvan, Nikolaos Nikolaou, Angelos Tsiaras, and Ingo P Waldmann. Detrending exoplanetary transit light curves with long short-term memory networks. *The Astronomical Journal*, 159(3):109, 2020.

Brett Naul, Joshua S Bloom, Fernando Pérez, and Stéfan van der Walt. A recurrent neural network for classification of unevenly sampled variable stars. *Nature Astronomy*, 2(2):151–155, 2018.

Hugh P Osborn, Megan Ansdell, Yani Ioannou, Michele Sasdelli, Daniel Angerhausen, D Caldwell, Jon M Jenkins, Chedy Räissi, and Jeffrey C Smith. Rapid classification of TESS planet candidates with convolutional neural networks. *Astronomy & Astrophysics*, 633:A53, 2020.

Aviad Panahi and Shay Zucker. Sparse box-fitting least squares. *Publications of the Astronomical Society of the Pacific*, 133(1020):024502, 2021.

Kyle A Pearson, Leon Palafox, and Caitlin A Griffith. Searching for exoplanets using artificial intelligence. *Monthly Notices of the Royal Astronomical Society*, 474(1):478–491, 2018.

Peter Plavchan, M Jura, J Davy Kirkpatrick, Roc M Cutri, and SC Gallagher. Near-infrared variability in the 2mass calibration fields: A search for planetary transit candidates. *The Astrophysical Journal Supplement Series*, 175(1):191, 2008.

Frédéric Pont, Shay Zucker, and Didier Queloz. The effect of red noise on planetary transit detection. *Monthly Notices of the Royal Astronomical Society*, 373(1):231–242, 2006.

Kai Rodenbeck, René Heller, Michael Hippke, and Laurent Gizon. Revisiting the exomoon candidate signal around kepler-1625 b. *Astronomy & Astrophysics*, 617:A49, 2018.

Yulia Rubanova, Ricky TQ Chen, and David Duvenaud. Latent odes for irregularly-sampled time series. *arXiv preprint arXiv:1907.03907*, 2019.

Roberto Sanchis-Ojeda, Saul Rappaport, Joshua N Winn, Michael C Kotson, Alan Levine, and Ileyk El Mellah. A study of the shortest-period planets found with kepler. *The Astrophysical Journal*, 787(1):47, 2014.

Christopher J Shallue and Andrew Vanderburg. Identifying exoplanets with deep learning: A five-planet resonant chain around kepler-80 and an eighth planet around kepler-90. *The Astronomical Journal*, 155(2):94, 2018.

Adam Wheeler and David Kipping. The weird detector: flagging periodic, coherent signals of arbitrary shape in time-series photometry. *Monthly Notices of the Royal Astronomical Society*, 485(4):5498–5510, 2019.

Edwin B Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927.

Liang Yu, Andrew Vanderburg, Chelsea Huang, Christopher J Shallue, Ian JM Crossfield, B Scott Gaudi, Tansu Daylan, Anne Dattilo, David J Armstrong, George R Ricker, et al. Identifying exoplanets with deep learning. iii. automated triage and vetting of tess candidates. *The Astronomical Journal*, 158(1):25, 2019.

Shay Zucker and Raja Giryes. Shallow Transits—Deep Learning. I. Feasibility Study of Deep Learning to Detect Periodic Transits of Exoplanets. *The Astronomical Journal*, 155(4):147, 2018.

Appendix A

A.1 Conference and workshop contributions

- pre-recorded oral presentation at Lunar and Planetary Science Conference
- published abstract https://ui.adsabs.harvard.edu/abs/2021LPI....52.2080R/abstract
pdf: https://www.hou.usra.edu/meetings/lpsc2021/pdf/2080.pdf
- live oral presentation at European Astronomical Society congress (June 29)
- abstract [first author .. ? - author list is switched up .. all symbols are question marks ...]:
https://eas.kuoni-congress.info/2021/programme/pdf/paperToPdf.php?id=1918
- several 5-10 minute presentations at EuroMoonMars workshops