Predicting Tidal Reflection

Dmitrijs Strizevskis

Master of Science

Computer Science

School of Informatics

University of Edinburgh

2020

Abstract

Tidal reflections are a persistent problem in over-water radio transmission, where the changing sea level causes the signal that is reflected off the water surface to shift in respect to the line-of-sight signal, periodically destructively interfering with it, thus causing a drop in the overall signal strength. This issue is also known as tidal fading, and, although it is a largely understudied phenomenon, there are known methods for mitigating it, such as spatial and frequency diversity. To work well, these methods require a means of predicting performance of tide-affected radio links, which is highly non-trivial since the tidal height in itself can be notoriously difficult to predict because of its dependence on atmospheric conditions. This work addresses this problem by designing deep learning models for forecasting performance of links affected by tidal fading. Experiments carried out on three months of real data from an over-water wireless network demonstrate that deep learning models are fit for this task, showing good results in spite of the modestly sized dataset. Attempts to build general models that predict performance of unseen before links proved to be unsuccessful, but the results also suggest that there is still untapped potential in training the models on larger amounts of data. In addition to the practical part of this dissertation, it also features a comprehensive theoretical interpretation of the phenomenon of tidal fading, which has been lacking from the literature.

Declaration

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Dmitrijs Strizevskis*)

Acknowledgements

First of all, I would like to thank my supervisor Peter Buneman for introducing me to this incredibly interesting topic and guiding me through this project from very early on. I would also like to express my gratitude to William Waites and Mahesh K. Marina for co-supervising the project.

I also wish to offer my special thanks to Finlay MacKenzie for setting up a UNMS account, Mark de Vries for his valuable advice and Martin Davies for setting up Air-Control which has proved crucial for this work.

Table of Contents

1 Introduction **1**

2 Background **3**

2.1 Tidal fading 3

2.2 Time series classification and forecasting 4

3 Physics of tidal fading **6**

4 Data **11**

4.1 Contents of the dataset 11

4.1.1 Signal and bandwidth 11

4.1.2 Sea level 13

4.1.3 Weather 14

4.1.4 Tidal fading onset markers 14

4.2 Data pipeline 16

4.2.1 Data sources 16

4.2.2 Cleaning and labelling 17

4.3 Data analysis 18

5 Methods **19**

5.1 Model design 19

5.1.1 CNN 21

5.1.2 LSTM 21

5.2 Baselines 22

5.3 Making the most of data 23

5.4 Hyperparameter tuning 25

5.5 Training and evaluation 26

6 Experiments **28**
6.1 Architecture choice . . . 28
6.1.1 Classification . . . 28
6.1.2 Regression . . . 29
6.2 Ablation study . . . 32
6.3 Generalisation across multiple links . . . 34
6.4 Impact of dataset size . . . 36
6.5 Test set evaluation . . . 38

7 Conclusions **39**

Bibliography **41**

A Equations **45**
A.1 Fresnel equations . . . 45
A.2 Path length difference in static water . . . 45
A.3 Path length difference in tidal water . . . 45
A.4 Angle approximations . . . 46

B Screenshots **47**

C Tools **49**
C.1 Using unms-stats . . . 49
C.1.1 Setting up a bash script run.sh . . . 49
C.1.2 Setting up a cronjob . . . 49
C.2 Fecthing data from AirControl . . . 50
C.3 GUI for labelling . . . 50

D Architectures **52**
D.1 CNN . . . 52
D.1.1 Diagram . . . 52
D.1.2 Tuned hyperparameters . . . 52
D.1.3 Tuning results . . . 53
D.2 LSTM . . . 53
D.2.1 Diagram . . . 53
D.2.2 Tuned hyperparameters . . . 54
D.2.3 Tuning results . . . 54

Chapter 1

Introduction

There are places in the world with limited access to broadband infrastructure due to difficult terrain. Examples of such places are the Scottish Highlands and Islands, where the combination of mountain ranges, straits, lakes and low population density often render wired broadband communications unaffordable. This gave an impetus to start the Tegola project – a high-speed wireless network covering rural areas of Scotland [2]. Today the Tegola network provides many communities with broadband access while running on highly affordable commodity hardware. However, wireless networks' inherent sensitivity to the environment, sometimes in unexpected ways, proved to be a challenge. In particular, some of the Tegola's over-water links that span sea inlets are affected by what has become known as *tidal fading* [18].

Tidal fading describes a situation where an over-water radio link experiences cyclic drops in the signal strength caused by two signals – direct and reflected from tidal water – cancelling each other out, with the phase of the reflected signal changing in time along with the tide. This can result in substantially lower quality of service, severely limiting the throughput for a considerable time and causing connection failures.

There are ways of mitigating tidal fading. For example, carefully installing an additional antenna at a different height can ensure that there is always at least one properly functioning link at any given point in time. This approach is commonly known as *spatial diversity*. The alternative approach is *frequency diversity*, where one can perform a timely switch to a different frequency channel, altering the phase difference between the direct and the reflected signals [18]. However, switching to a different frequency channel or antenna can incur a downtime, especially on cheap hardware. This indicates the need for a method for forecasting performance of links affected by tidal fading. This would allow a network technician or an automated algorithm to an-

ticipate approaching onsets of tidal fading and plan for them accordingly, scheduling well-timed switches and avoiding false positive ones.

However, predicting tidal fading is not a trivial task. Although signal strength is highly dependent on the sea level, knowing the sea level at a remote gauging station, let alone relying on imprecise tide tables, does not explain all of the variance of signal strength. This is because the propagation of tides depends on weather conditions such as wind speed and air pressure [9]. Moreover, even if the sea level at the reflection point were precisely known, reflectance can further be impacted by surface roughness and, to a lesser extent, air humidity [19]. With so many intricate interdependencies, manually modelling this process is highly challenging. However, the ability to collect the relevant data enables us to train deep learning models, which are capable of modelling otherwise intractable dependencies given enough data.

With the ultimate goal of contributing to solving the problem of tidal fading, this work carries several purposes. The first is collecting a dataset and then using it to design, train and evaluate deep learning models that can be of practical benefit in the Tegola network. The second is being a reference that can assist in training these models in the future when more data becomes available, along with the dataset, tooling, code and practical tips. Last but not least, it attempts to close the apparent gap in the literature with regards to tidal fading by providing a comprehensive theoretical explanation of the phenomenon.

The experimental findings of this work show that deep learning models are indeed fit for predicting performance of tide-affected links, outperforming simpler baselines despite seeing less than 200 tidal cycles in the training set. Although generalisation between different links was not observed, there are indications that the models are far from reaching their full potential and, as it stands, the size of the dataset appears to be the largest bottleneck. On such scales, weather data was found to be a useless addition to the set of input features, unlike tidal height data which was found to be a beneficial input feature.

The structure of the dissertation is as follows: §2 covers the background material relevant to tidal fading and deep learning for time series forecasting and classification, §3 gives a comprehensive overview of the phenomenon of tidal fading, §4 describes in detail the steps behind acquiring and processing of the data used in this work, §5 gives an overview of machine learning methods used in the experiments, which are then presented in §6. Final conclusions with a brief summary of the results is provided in §7.

Chapter 2

Background

2.1 Tidal fading

Despite tidal fading being a known phenomenon for a long time, the literature on the topic remains sparse. This section gives an overview of the limited information that can be found on the subject whilst the later chapter (§3) attempts to give a more complete explanation of tidal fading to provide the reader with the bigger picture.

The book "Essentials of radio wave propagation" has a section on microwave propagation over water that also concerns tidal reflections. The author suggests spatial diversity as a solution to the problem and derives equations for estimating the best separation of antennae to make sure that when either one of the antennae is in a signal strength trough (also commonly referred to as *null*), the other one is at the peak. It does not go into details of the mechanism by which tidal fading occurs or the effect that weather can have on it [12]. S. Mason studies the effect that meteorological conditions have on a radio link where the transmitter is installed on a floating dock that moves with the tide. He finds that the impact of the height variation on the signal strength dwarfs all other environmental factors [19]. However, despite the involvement of tides, this case is not directly related to the actual tidal fading, through which the weather can have an indirect effect on the transmission.

A paper that came out of the Tegola project in 2010 proposed frequency diversity as a means of mitigating tidal fading, which is a cheaper alternative to spatial diversity since it can be implemented purely in software with frequency hopping. Authors suggested several strategies for deciding when to switch frequency channels – one of them involving predictions of link's performance to guide the decision [18]. Such predictive models are the focus of this dissertation.

M. Pereira provides a more in-depth physical model of fading of electromagnetic waves over conductive surfaces, such as seawater. He demonstrates how to work out signal power attenuation using Fresnel equations as well as briefly describes different strategies of mitigating fading [21], including those already mentioned.

In a recent paper, Gaitan et al. ran simulations to determine the influence that different settings of antennae heights and polarisation have on the signal strength in over-water links affected by tides. As before, it does not feature a comprehensive explanation of the phenomenon of tidal fading and the authors raise the fact that the problem has received much less attention than it deserves given its significance [8].

2.2 Time series classification and forecasting

To my knowledge, tidal fading forecasting has never been addressed with machine learning, let alone deep learning. Nevertheless, time series classification and forecasting using deep learning are very active areas of research. Although this work concerns a new domain in those areas, there are many conceptual similarities with other machine learning problems where data comes in the form of time series.

For example, financial time series prediction is one such area that has received a great deal of research attention in recent years, especially with the advance of deep learning. A systematic review of publications in that field by Sezer et al. concludes that deep learning models generally outperform more traditional machine learning methods. Authors find that recurrent neural networks (RNN), in particular, Long Short-term Memory (LSTM) are the most commonly used architectures for time series forecasting, whilst Convolutional Neural Networks (CNN) are usually the architecture of choice for classification tasks [10].

Wang et al. propose a modification to the CNN architecture, called the Fully Convolutional Network (FCN), which achieves highly competitive results across many different time series classification problems without tailoring to each problem individually, making it a strong general baseline [32]. In a recent review, Fawaz et al. do a comprehensive empirical study of different neural network architectures for time series classification. They also find that FCNs and Residual Networks (ResNets) achieve state-of-the-art results for different tasks. Moreover, the authors published implementations of studied models on their GitHub [7].

Zhao et al. use LSTM networks for forecasting road traffic. LSTM networks demonstrated better performance than classical models and simpler neural network

architectures. However, authors also find that the accuracy of predictions quickly deteriorates with longer forecast times [34]. Selvin et al. analyse the performance of CNNs and LSTM for univariate stock price predictions, using overlapping windows of data for training. They identify the CNN as the best architecture for the task [28].

Cerqueira et al. study different approaches to evaluation of performance of time series forecasting models, including different variants of cross-validation techniques [4]. Klimberg et al. do a breakdown of various forecasting performance measures and give practical interpretations [16]. Note that in this work I refer to what they call Mean Absolute Deviation (MAD) as Mean Absolute Error (MAE), which is another common name for it.

Chapter 3

Physics of tidal fading

A two-ray ground-reflection model is a common way to analyse tidal fading whereby we consider two components of the signal – the direct line of sight (LOS) ray between the antennae as well as a secondary ray that is reflected off water.

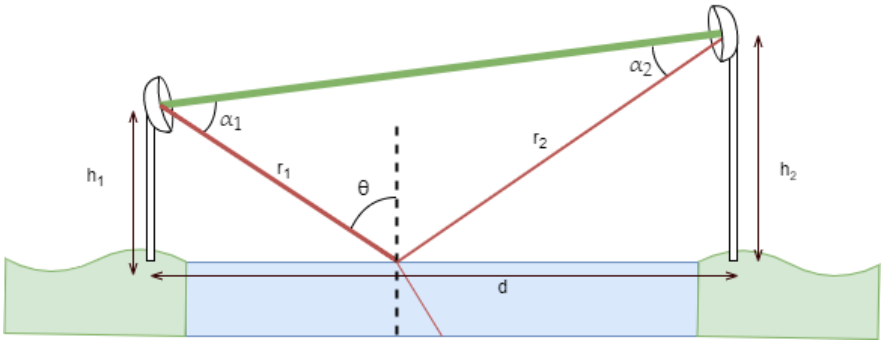
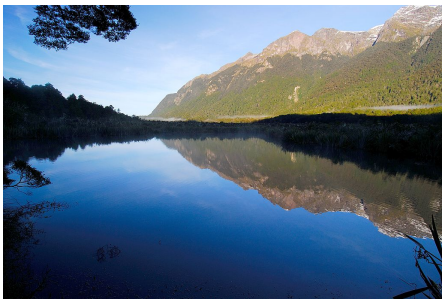


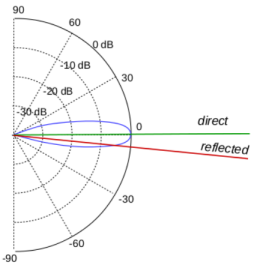
Figure 3.1: The two-ray model.

Figure 3.1 illustrates the two-ray model. The direct signal path l is represented as the green line, while the reflected signal paths r_1 and r_2 are in red. Line thickness represents signal strength. The surface of water is not a perfect conductor, which means that a portion of the secondary ray is refracted into the water according to the Fresnel equations (A.1) [21], making the reflected beam lose part of its power, hence r_1 is thicker than r_2. The smaller the angle of incidence of the reflected beam, the higher the proportion of the refracted signal. We observe an equivalent effect with visible light when we look at water and see a reflection that gets gradually darker towards us (Figure 3.2a). The diagram also represents l as a thicker line than r_1; this is for a reason. Not all signals are emitted (or received) equal – the power of the transmitted beam depends on which part of the transmitting antenna it is emitted from and where it enters the receiver. Antennae have radiations patterns which determine

the amplification of an outgoing/incoming signal based on how the signal is directed through the pattern. Figure 3.2b shows the main lobe of an antenna's radiation pattern; the green signal goes through the area of maximum amplification, whereas the red signal is transmitted at an angle with lower power. Figure 3.1 illustrates the same situation – the two antennae are pointed at each other so that the direct signal takes full advantage of amplification.



(a) Reflection in a lake. Photo by Jenny / CC BY 2.0



(b) Antenna lobe (in blue)

Figure 3.2

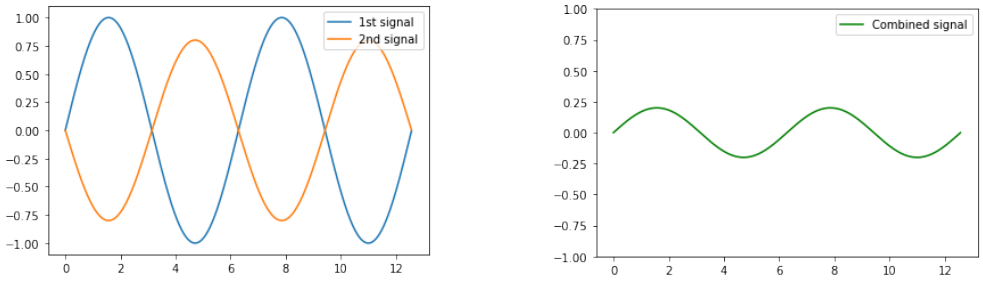
The receiver combines all incoming signals by adding them together, akin to wave interference. Therefore, it may happen that, when the two signals are in anti-phase with each other, the aggregate signal fades (is at a null). Figure 3.3 shows a toy example of such destructive interference, where one signal has an amplitude of 1, the second signal has an amplitude of 0.8 and they are in antiphase with each other. When the reflection surface is still, one can fix this by tuning antenna heights until the phases of the two signals shift relative to each other by half a wavelength so that they are aligned in constructive rather than destructive interference.

$$\delta = l - (r_1 + r_2) \approx \frac{2(h_1 - t)(h_2 - t)}{d}, \tag{3.1}$$

where t is the height of the tide at the reflection point, which we set to 0 in static water.

Equation 3.1, which gives the length difference of LOS and reflection paths, can be used to work out the necessary change in antenna heights for achieving this (derivations in A.2, A.3). However, this situation gets more complicated in tidal waters since antenna heights change along with the tide, causing the relative phase shift of the two signal to drift, once in a while resulting in nulls. This is called tidal fading.

There are several ways one can approach this problem. In some situations, switching to a lower frequency may result in a wavelength large enough that it dwarfs the



(a) (b)

Figure 3.3: Destructive interference of two signals in antiphase (a). The 2nd signal has a slightly lower amplitude, hence the resulting signal (b) is not completely flat.

phase shift caused by tides. For example, in a setup where $h_1 = 20m$, $h_2 = 40m$, $d = 8km$ and the sea level varies (Δt) by $5m$, which is common in Scotland, the phase shifts by at most 0.08m (see equations 3.1 and A.3). In that case, for example, switching to a 0.5GHz radio, which has a wavelength of 0.5m, would mean that, if tuned properly, the antennae will never end up at a null. In fact, if the maximum phase shift is below one-quarter of the wavelength, such setup should never experience any destructive interference at all. Alternatively, instead of increasing wavelength, one can attempt to constrain the phase shift. As equations suggest, this can be done either by sufficiently lowering the dishes or increasing the distance between them. Often this is not feasible and, depending on the setup, can even be harmful because lowering an antenna runs the risk of obstacles, e.g. passing ships, getting in the way of the beam, and it can have an amplifying effect on the reflected signal. In fact, sometimes the opposite is done.

$$\alpha_2 \approx \frac{2h_1}{d}, \quad \alpha_1 \approx \frac{2h_2}{d} \tag{3.2}$$

Raising antennae can help in two ways. First, it increases the angle of incidence (θ) of the reflected signal, increasing the amount of power that gets absorbed by water. Second, it also increases the angle between the reflected ray and the axis of maximum amplification of the main lobes of the antennae (equations 3.2, see the derivation in A.4), reducing the amplification of the reflected signal. Note that the shape of the lobe (Figure 3.2b) is such that amplification falls more rapidly as the angle gets farther from 0 degrees. This reveals yet another potential solution to the problem, which is often done in practice. One can tilt one of the antennae up, which corresponds to a clockwise rotation of beams on Figure 3.2b. Although this causes a slight misalignment of the

LOS path, due to the shape of the lobe, it loses less power than the reflected path. Equations show that the lower antenna will have the larger angle between the reflected path and the axis of maximum amplification, making it the best candidate for tilting since this will result in more power attenuation per degree than with the other antenna.

The aforementioned methods have two common drawbacks. First and foremost, sometimes they are simply not enough to counteract tidal fading: admissible antennae heights are subject to landscape features and resources (building a tall mast or moving an antenna uphill is expensive), low frequency radios are not suited for high-bandwidth channels [11]; with some exceptions, the distance between the antennae can hardly be changed, as it is dictated by where they are needed, and sometimes tilting an antenna will not be enough. Second, since these methods are aimed at making the reflected signal weaker, not only do they reduce the effects of destructive interference, but they also do the same to constructive interference, sacrificing sometimes better performance for consistency. This is where active fading mitigation strategies can come into play.

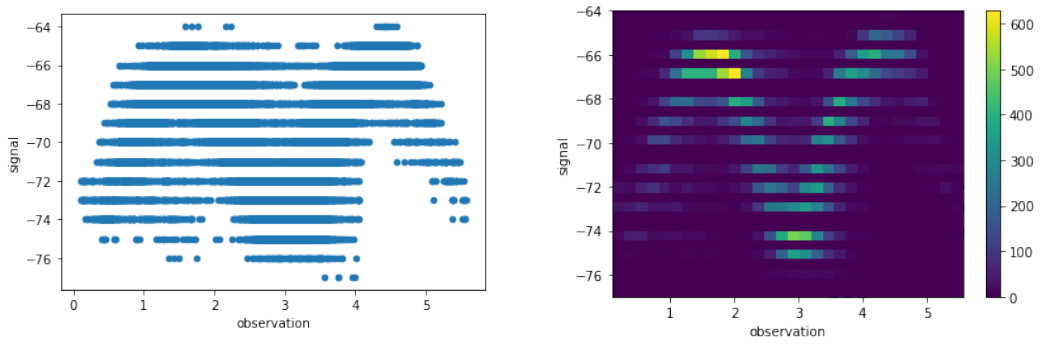
We have seen that the antenna heights and the radio frequency determine the nominal relative phase shift of the two signals, i.e. if the sea level were fixed. This means that switching to an antenna at a different height or hopping to different frequency channel will alter the phase shift. The former is called spatial diversity and the latter – frequency diversity [18]. Spatial diversity is more expensive as it requires additional hardware. Frequency diversity, however, is subject to availability of the required frequency band. With spatial diversity, the trick is then to place one additional antenna at such elevation that, when one antenna is at a null – the other one is at peak performance. Similarly, for frequency diversity, one can calculate the necessary frequency separation between two channels to ensure that when the two signals are shifted by 180° in one channel, they are perfectly aligned in the other channel [18]. Then, by timely switching between antennae or channels, one can guarantee that at least one link is not affected by destructive interference, hence avoiding nulls. In fact, one can use more than two spaced out antennae/frequency channels to get a better guarantee for the best performing link at any given time. However, there is yet another pitfall.

Switching can be costly, especially, on cheap hardware where it can result in downtime on the order of seconds. This is why it is important to switch at the right moment, avoiding false positives calls. Unfortunately, detecting “the right moment” is not a trivial task. One could, for example, simply monitor signal strength on the active link and switch whenever it falls below a certain threshold, but this can be highly unreliable in noisy conditions; for example, a drop in the signal strength can be caused by a pass-

ing ship or a bird landing on the antenna, thus, triggering a false alarm. Alternatively, one could time switches based on the tidal schedule. However, tidal height is hard to predict precisely as it depends on atmospheric conditions.

Atmospheric pressure can have a strong effect on tides. High pressure prevents tides from growing large; conversely, low atmospheric pressure, e.g. during hurricanes, is fraught with tides significantly larger than usual. Strong wind can also have an amplifying effect on tides in the direction where it blows [9]. Other, less influential factors, which do not affect the sea level but can affect links' performance, include roughness of the sea and air humidity [19] – the former can affect reflectance for radio waves with wavelengths comparable to the size of sea waves, and the latter can slightly weaken both signals as well as change the refractive index of air [20], affecting the amount of the secondary signal that gets absorbed by water.

One could measure the sea level directly, avoiding the need for predicting it; however, it is very expensive, requiring large investments in hardware and maintenance. Alternatively, measurements from gauging stations can sometimes be obtained from 3rd party sources, like the National Tidal and Sea Level Facility (NTSLF) in UK. Of course, the farther the gauging station from some point, the less accurately its measurements reflect the sea level at that point. For example, tidal height data from the Tobermory gauging station was used in this work; the station is roughly 50km away from the reflection points of links in question. Figure 3.4 demonstrates that, whilst the signal strength strongly depends on the sea level at the remote station, its variance is nowhere near fully explained, especially between peaks and troughs (a difference of 3 dBm corresponds to an approximately two-fold increase/decrease of signal power). This work is an attempt at predicting it better by training deep learning models to take all those different variables into account.



(a) Scatter plot

(b) 2D Histogram

Figure 3.4: Signal strength (dBm) of a link vs sea level (m) at a remote gauging station

Chapter 4

Data

Data plays the key role in the process of designing machine learning algorithms. This chapter describes the design process of the dataset for the task, including acquisition, preprocessing and analysis of various forms of data.

4.1 Contents of the dataset

The full dataset consists of three datasets – one per radio link – each consisting of time series with a 5-minute step for the time interval from the 8th of March till the 25th of June. The time series are:

- Signal and bandwidth time series for the particular link
- Sea level time series taken at the nearest gauging station
- Weather time series at a location near the radio links: wind speed (both magnitude and degree), air pressure and air humidity
- Markers indicating onsets of tidal fading. These are not part of raw data and are artificially created during preprocessing for use in classification (see §4.1.4).

4.1.1 Signal and bandwidth

The primary form of data in the dataset is signal strength and bandwidth of tide-affected radio links, which are key in ascertaining whether the link is undergoing tidal fading – when they are high, there is clearly no significant fading. Moreover, the curvature of the time series can hint at the stage of the fading cycle. Although signal

strength and bandwidth are closely related, the former is a relatively low-level measure reflecting a physical property of the aggregate received signal, whereas the latter is a high-level measure that goes through many layers of hardware processing and adjustment. For example, whilst the signal level time series is relatively smooth, the bandwidth time series can have plateaus where it caps or where the device takes steps to stabilise it. On the other hand, bandwidth is perhaps more important as a prediction target because it is more easily interpretable – reporting a decrease in bandwidth from 100 to 25 Mbps is more telling than an arbitrary decrease from -68 to -74 dBm of signal strength. For the same reason, it proved useful as a reference for creating markers during preprocessing (§4.1.4).

The Tegola network runs on Ubiquiti devices, which report both signal strength and bandwidth as integers, with the signal being reported in decibel-milliwatts (dBm) and bandwidth (both download and upload) in Mbps. This makes the range of signal level values rather sparse due to the logarithmic scale of dBm. To accumulate enough data for the purpose of machine learning, it was decided to start collecting it early in March. Measurements were taken at 5 minutes intervals for three links that exhibited the most apparent tidal fading. Although higher resolution (on the order of 5-20 seconds) is possible (see section §4.2 for more details), it would make these time series significantly out-of-scale with the rest of the data. The final range of signal and bandwidth time series is from the 8th of March till the 25th of June. With a lunar day lasting 24 hours and 50 minutes and two tidal cycles per lunar day [24, 31], this works out to roughly 200 tidal cycles across 31k data points per link.

Preliminary data analysis revealed some unfavourable aspects leading to additional preprocessing steps. On one of the links, the signal time series had clear outliers, with sudden single spikes up to -20 or down to -100 dBm from within the normal range of -50 – -80 dBm. Such exponential jumps can be attributed to something passing between the antennae, e.g. a ferry, or an antenna swaying in the wind etc. Because these spikes were few and far between, keeping them in the dataset to build up robustness to this kind of noise was pointless; it could, however, be useful in a larger dataset – one could even artificially augment the dataset with such artefacts – but for a dataset as small as ours that would mean needlessly losing much signal-to-noise ratio. Hence, a preprocessing step was added that removes all points below or above a certain threshold and replaces them with the average of the neighbouring points.

Another observation was that the Rx and the Tx bandwidth curves, otherwise behaving in almost the same way, sometimes would not always both plateau at the same

level. For example, while the Tx curve is capped at around 150 Mbps and appears flat, the Rx curve can fluctuate some 10s of Mbps above this flat line. Due to their similarity, keeping both statistics in the dataset was excessive, but, for example, taking the average of the two would mean that a sharp drop from the plateau, which could potentially be a useful signal to models, would be smoothed out by the second curve. Instead, the minimum of the two curves was taken, which preserved the flatness (see the orange line on Figure 4.3 for an example).

Finally, it was also noted that bandwidth was capped at different levels on different links. On a dataset of this size, this could hinder generalisation between several links. To address this, bandwidth was scaled by dividing it by the value at which it caps. This value is set manually in the GUI labelling tool (§4.2), but one could determine it automatically. These reference values were stored as part of the dataset so that true bandwidth could be restored by multiplying by the corresponding reference value. The relative scale of bandwidth with respect to the cap also served as a useful general heuristic in determining onsets of tidal fading (§4.1.4).

4.1.2 Sea level

Sea level is the second most important candidate to be added into the dataset. As shown in §3 (Figure 3.4), there is strong dependence between signal strength/bandwidth of a link and the stage of the tide. However, acquiring accurate sea level data is difficult. Without large investments into one's own tide gauges, one has to rely on third party sources. In the UK, National Sea Level and Tide Facility (NTSLF) and British Oceanographic Data Centre (BODC) provide real-time tidal data as well as historic records. BODC publishes monthly data from 43 gauging stations across the UK. The gauging station in Tobermory is closest to the three radio links (see B.1 for the locations on the map), so it was chosen as the source of the tidal data for March through June. The data has a machine-readable format and the tidal height (relative to the admiralty chart datum) is reported in metres every 15 minutes. During preprocessing, this data was linearly interpolated to match the resolution of the signal/bandwidth time series.

An important criterion when considering potential sources of data was the ability to acquire it in near real-time with the intent to be able to apply the resulting models "in production". This is complicated by the fact that, currently, NTSLF do not publish real-time tidal height data in machine-readable formats. Instead, it is displayed as a raster image of a chart (see Figure B.2 for an example). In the event that raw real-

time data remains inaccessible, one could write a script to read off the measurements from the chart. Additionally, to account for the cases when sea level data is completely unavailable for any reason, an ablation study was performed (6.2) where tidal data was completely removed from the input features.

4.1.3 Weather

Chapter §3 explains the effects that weather can have on tidal fading. Since in our case tidal height measurements are taken at a remote location, there is a lag between the measurement and the actual sea level at the reflection points. This lag can potentially be further aggravated by high air pressure and wind. Although it was unclear whether the effects would be noticeable in this particular geography on a relatively small dataset, incorporating weather data was simple enough to test it in the ablation study (6.2). I chose OpenWeatherMap for the weather API provider because they featured historical weather data for any coordinate on the map. The data has an hourly step and, amongst other parameters, includes wind speed provided in degrees and magnitude (m/s), air pressure (hPa at sea level) and humidity (%).

Similar to the sea level, weather data was interpolated to 5-minute steps; however, special care was taken with respect to wind speed. Whilst air pressure and humidity are smoothly behaving scalars, wind speed, which is a vector quantity, can be restricted by the geographical features. An extreme example would be a narrow fjord where wind can blow only in either of two directions. In that case, linear interpolation would be misleading. Instead, I chose nearest-neighbour interpolation for the wind speed time series.

4.1.4 Tidal fading onset markers

To train a supervised model to predict “onsets of tidal fading”, one needs a definition for such onsets. As seen on Figure 4.1, signal strength on a tide-affected link changes smoothly with time, without abrupt “drops”. This means that “an onset of tidal fading” is rather vague – at which point of the smooth curve does it start? One way to define it could be the peak of the signal curve followed by a slow but steady drop. However, because of the logarithmic scale as well as the rounding that is applied to the signal strength values on the hardware end, the top of the curve looks rather flat, making it hard to pinpoint the peak. Moreover, it can take on the order of hours until the signal drops from the peak to the point where it noticeably affects bandwidth, making this

definition not very practical. The first attempt was to instead manually mark all points where it was apparent that the drop in signal strength could not be attributed to noise and that further descent was imminent. With the help of the developed GUI tool (§4.2) it was not very time consuming on a dataset of this length; however, on a dataset with a timespan on the order of years it would be infeasible for a solo effort. Nevertheless, a larger issue with this approach turned out to be the difficulty of keeping consistency when deciding where to put a marker or whether to put it at all. This was especially problematic in highly noisy regions or when the amplitude of the tide was large enough that there were two peaks of signal strength per one peak of sea level, as seen on Figure 4.1.

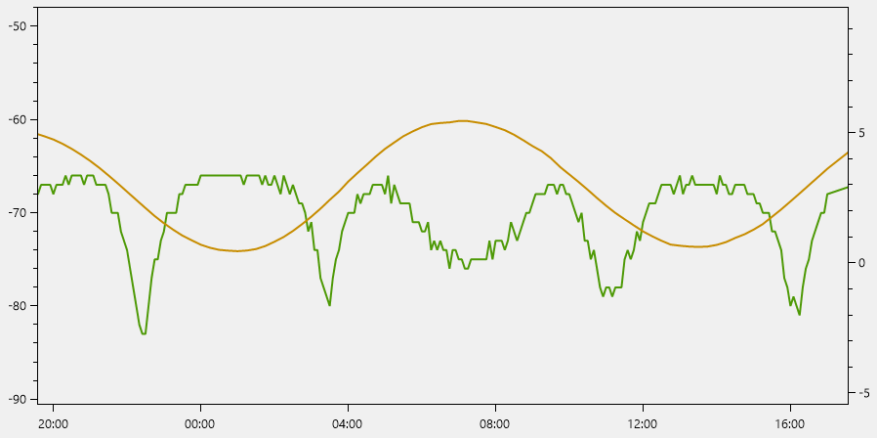


Figure 4.1: Screenshot from the GUI tool for labelling, showing two peaks of signal strength (green line, left axis) per one peak of tidal height (orange line, right axis). This happens when the tide is high enough that the phase shift exceeds the point where the two signals are in phase.

The second and final attempt involved strictly defining the onset of tidal fading in terms of loss of bandwidth relative to the value at which it caps, which, after scaling (see §4.1.1), is exactly 1.0 on all links. Such a definition enables simple automatic labelling. After many experiments with different parameters, the following criteria for an onset gave visually consistent results that also made practical sense: a point constitutes an onset of tidal fading if it marks the moment where bandwidth falls below a 72% threshold after staying above it for at least 20 minutes and then stays below it for at least 30 minutes. These settings required little to no manual corrections to achieve seemingly consistent labelling across all three links. The caps of the three links range from 140 to 170 Mbps, so the thresholds roughly fall in the range of 100 to 120 Mbps. It is unclear whether the same approach would work well for links

with drastically larger capacity or even for hardware from other vendors that may handle bandwidth differently. Therefore, this step must be performed with special care – adjusting heuristics on a case-by-case basis – if one intends to train a model that generalises well across multiple links.

4.2 Data pipeline

This section covers the the details of the tools that were used in acquiring, processing and labelling of the dataset so that the experiments can be reproduced in the future on a larger dataset or for a different network.

4.2.1 Data sources

The Tegola network is centrally managed via the Ubiquiti Network Management System (UNMS) and AirControl (predecessor to UNMS). They allow for monitoring statistics such as signal strength and bandwidth via a GUI in the browser, but UNMS also exposes an API through which one can programmatically query statistics of the devices. The web GUI is helpful in determining links of interest – significant fading will result in a highly periodic wavy signal strength curve with an amplitude of upwards of 10 dBm (e.g. as seen on Figure 4.1). After identifying the affected devices, one can periodically poll them using the API to retrieve recent statistics (e.g. signal strength, bandwidth) with a relatively short timestep – 5–20 seconds. In March I set up a simple Python tool *unms-stats* to retrieve signal strength statistics from the specified list of devices and store them in a SQLite database. This command-line tool can be used both for accumulating data at a higher resolution than what is available in UNMS or AirControl and for retrieving data in real-time for use in production. I published the code for unms-stats on GitHub[1], and the appendix (C.1.2) includes a sample cronjob setup for continuous monitoring. However, soon after setting up custom monitoring, I got access to AirControl, which had been set up to monitor many more links. Amongst other statistics, AirControl also collected bandwidth data, which unms-stats was not set up to collect at the time since the benefits of bandwidth data had not yet become apparent. Thus, it was decided to keep unms-stats for redundancy as a back-up alternative and use the AirControl data instead. The data stored in AirControl has lower resolution with 5-minute steps, which is still fine-grained enough to capture important

[1] https://github.com/swifteg/unms-stats

features of the signal curve and it is on the same scale as the rest of the data, i.e. the sea level and weather time series. Unfortunately, AirControl does not feature a simple way of downloading all of the persisted data with the shortest possible timestep – see the appendix for a work-around (C.2). In future work, it is recommended to use unms-stats, both for a higher resolution, which can always be downsampled if needed, and for easier real-time use.

As mentioned in §4.1.3, weather data for a particular geographical coordinate can be acquired from 3rd party services in a variety of machine-readable formats. The same is true for tidal height data in UK, which can be requested from BODC. However, real-time tidal height data is not easily accessible, and obtaining it will either involve parsing an image of a chart from NTSLF or relying on other 3rd party providers.

4.2.2 Cleaning and labelling

When data from all sources is collected, it goes through preprocessing and labelling steps (Figure 4.2). This is repeated per link. First, a script bundles signal, bandwidth and sea-level time series into one, interpolating the sea level to match the 5-minute timestep. If the radio link ever lost signal it would appear as *null* in the data. These points are dropped from the dataset, which results in "tears" in time series that are carefully handled before feeding the data to machine learning algorithms (see §5.3).

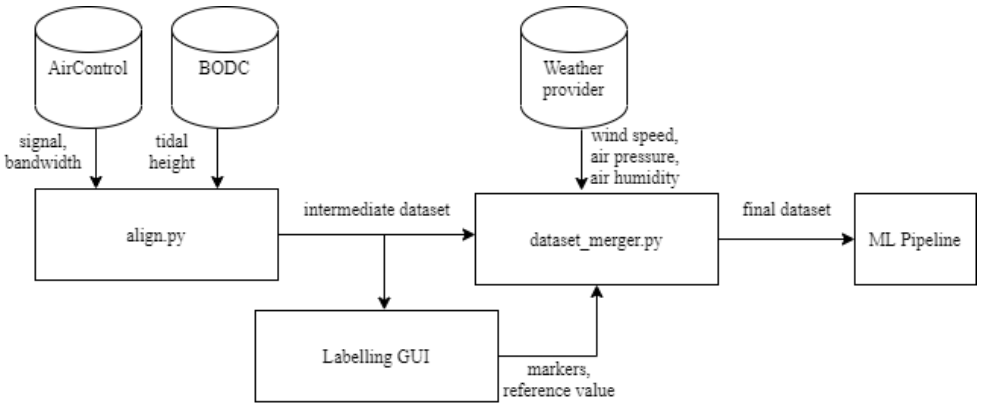


Figure 4.2: Data processing pipeline

The output of the first step is then passed on for labelling, i.e. markers of tidal onsets are added to the dataset. This is performed in a semi-automatic fashion using a developed GUI tool (Figure 4.3). The tool supports editing markers manually, but it also implements automatic labelling based on the criteria described in §4.1.4. The appendix has a more detailed description of its features and controls (C.3).

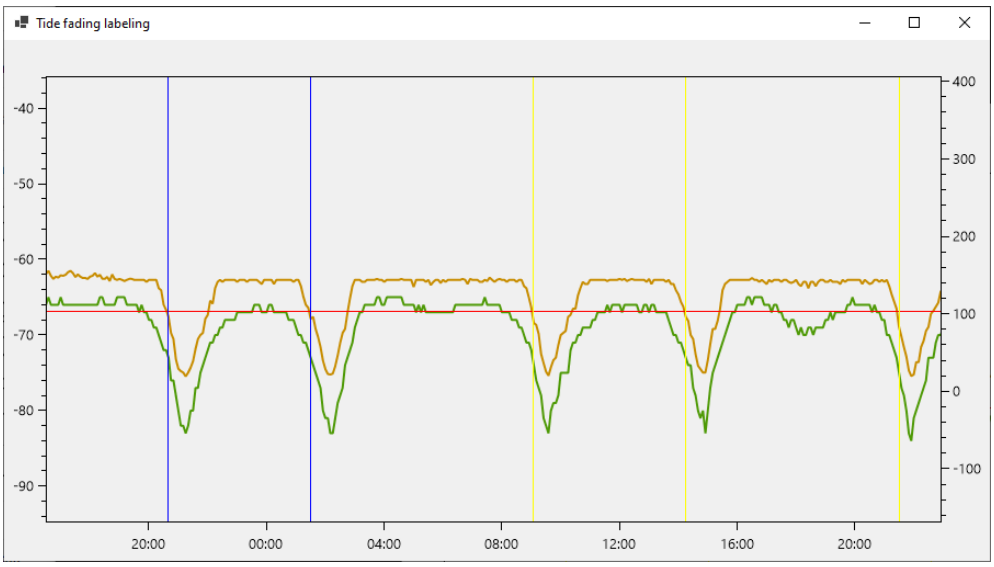


Figure 4.3: GUI tool for dataset labelling

The third step bundles the output of the first step with markers and weather data, interpolates the weather data, transforms bandwidth based on the reference value (see §4.1.1) set in the labelling tool, removes outliers, trims any excess data and serialises the dataset. At this point, the data is clean and ready for statistical work. The further processing steps are more narrowly tailored to machine learning algorithms and are described in chapter §5.

4.3 Data analysis

A closer look at the datasets for the three links (from now on referred to as links #1, #2 and #3) revealed some differences which could hinder generalisation. Link #1 is the more consistent, behaving similarly during different tidal seasons (the amplitude of the tide changes all the time from around $1m$ to $5m$ with a period of roughly 2 weeks). Amplitudes larger than $4.5m$ would result in 2 peaks of signal strength per high tide, whereas amplitudes of around $3.2m$ would result in a highly spiky signal curve. Nevertheless, the labelling for this link is more or less uniform across the entire range. Link #2 is less affected by tidal fading in general, and there appear to be “islands of stability” where no significant fading occurs at all, making the labelling more sparse. Link #3 only ever experiences tidal fading when the amplitude of the tide is above $2m$, so the labelling also is not uniform. See the appendix for supporting figures (B.3).

Chapter 5

Methods

There are several approaches to forecasting performance of radio links affected by tidal fading. One approach is to frame it as a classification problem and train a model to predict whether or not there will be an onset of tidal fading in the near future. The main advantage of this approach is that the output is easily interpretable – it is informative even to a casual user and can be used as part of an automated fading mitigation procedure, e.g. slow frequency hopping [18]. On the other hand, the disadvantages are that onsets of tidal fading are hard to define, which is elaborated in §4.1.4, making it difficult to have a consistent, generalisable and, at the same time, useful labelling across data from different radio links. It also lacks context – a network technician may prefer a more detailed picture of the link's performance to a binary indicator. A different approach is an autoregressive model that outputs a continuation of performance curves, i.e. signal and bandwidth, for some number of timesteps into the future after the last input. Relating such output to tidal fading is not as straightforward as a binary indicator, but it provides a bigger picture, and, with additional postprocessing steps, one can infer the same kind of labels as the output of a classifier. Whilst the classifier model is more true to the topic of predicting tidal fading end-to-end, the regression model perhaps has more practical use. That is why in this work I considered and evaluated both approaches.

5.1 Model design

The first step in designing models was to decide on the format of input and output to the models. Doing that, it was important to strike a balance between practicality and efficient use of the dataset. For example, doing a prediction for one future timestep

while conditioning on a single past timestep would have little practical use, but would result in a large number of independent input-output pairs for training. The other extreme would be feeding the model the widest possible window of past time series, requiring a more complex model, but, at the same time, making the training set too sparse for the model.

A 6-hour window of most recent data was chosen for the input because this roughly corresponds to a half-period of the tide, allowing the model to witness the last peak or trough. Given a 5-minute timestep, this corresponds to a 72-point long input sequence. With the help of some extra steps described in §5.3, splitting the dataset into such samples still resulted in a fairly sized training set. The output of the classification model is a one-hot encoded vector, where classes correspond to disjoint future intervals that an onset of tidal fading may fall into. To account for predictions getting less accurate the further the targets are, these intervals are sized exponentially: 0–15 minutes, 15-60 minutes, 60-120 minutes, 120-240 minutes and from 240 minutes to infinity. Although it would be useful to have an even shorter first interval, any less than 15 minutes would result in training samples being too noisy because of imperfect onset marking (§4.1.4). For the regression case, the output is 3-hour long multivariate time series of future bandwidth and signal strength, which is twice as short as the input data but is still long enough to schedule fading mitigation routines well in advance.

The next step was deciding on architectures for deep learning models. For experimenting with deep learning models I extensively used the Keras library [5] with Tensorflow back-end which allowed for quick prototyping, evaluation and hyper-parameter tuning. Although Keras does not feature all of the latest state-of-the-art architectures, the modest size of the dataset is probably the most dominant performance bottleneck, rather than a suboptimal choice of an architecture. Nevertheless, there are important architectural choices that may affect both the validation performance and computational efficiency.

Both in the classification and regression modes, the input to a model is a multivariate time series. A good architecture will, therefore, be suited for extracting features from temporal sequences. LSTMs and 1-dimensional CNNs, which are widely used for time series classification and forecasting (see §2.2), are first-class citizens in Keras. Because I highly relied on automatic hyperparameter tuning (see §5.4), I kept the architectures conceptually simple to avoid a combinatorial explosion of different hyper-parameter combinations. The following two subsections describe the skeletons of the tested CNN and LSTM models. For classification, I also compared them against

more involved architectures – Residual Networks (ResNet) and Fully Convolutional Networks (FCN) – that showed the best performance in a study of different architectures for time series forecasting [7]. These architectures, implemented in Keras, were taken as is from the authors' public repository.

5.1.1 CNN

Convolutional neural networks have highly parallelisable computation graphs [23] at the expense of not intrinsically modelling time dependence, unlike LSTMs. This makes CNNs more lightweight in comparison, allowing for deeper models or faster compute times. The very first experiments were done with the most basic "vanilla" CNN architectures, with a number of equally-sized 1-D convolutional layers stacked on top of each other without pooling layers, followed by two fully connected layers – a hidden layer with ReLU activations and the final output layer with either linear activation in the regression case, or softmax activation in the classification case. Early hyperparameter tuning attempts showed that models with a relatively high number of layers (around 8) and few filters (also around 10) performed the best. Deep networks and deep CNNs, in particular, can suffer from the *vanishing gradients* problem, when, due to repeated multiplication, gradients from the input layer have hard time propagating through many hidden layers [7]. Because raw input features can be important, especially to the autoregressive model, with inspiration from the ResNet architecture [7], I added a skip connection from the input to the end of the stack of 1-D convolutions. Unlike in ResNet, the skip connection performs concatenation (rather than elementwise sum) and there are no skip connections in-between convolutional layers, nor are there batch normalization layers. The filter width was fixed at 3 and the multivariate time series is passed as multiple channels. Refer to the appendix (D.2) for the diagram of the architecture.

5.1.2 LSTM

LSTM, a type of recurrent neural networks, processes timesteps one after another, which makes it impossible to parallelise computations in the time dimension, rendering LSTMs computationally more expensive than CNNs, especially for long input or output sequences; however, this naturally captures time dependence of time series.

For the classifier I chose the most basic skeleton – a number of LSTM cells stacked on top of each other, with the final output of the top cell connected to a hidden dense

ReLU layer, followed by the output layer with softmax activation. The autoregressive model is different – it employs the encoder-decoder architecture where, first, the encoder compresses the input into hidden latent space and then the decoder decodes it into the output sequence. Because the decoder can have a different input format to the encoder, this configuration allows to train our model using *teacher forcing*.

When training the model with teacher forcing, the decoder gets passed the i-th ground truth value before predicting the $i+1$-th step. During prediction, ground truths values are unavailable, so instead the output of the decoder is passed as the next input [33]. Teacher forcing speeds up training of deep learning models and is widely used in autoregressive models, especially in the field of natural language processing [13]. Nevertheless, there are conflicting views on the effect that this procedure has on generalisation and whether it should be avoided. The concerns stem from the fact that the decoder generates output from different distributions during training and inference – the former distribution is a conditional on the ground-truth prefix and the latter is a conditional on the previous outputs of the model. The disparity between the two is commonly known as *exposure bias*, and its impact on generalisation is debated [13, 25, 17, 27].

To contrast the CNN-based model (§5.1.1), which was trained conventionally, I used teacher forcing for the LSTM-based regression model to test whether it would be beneficial in our scenario. Both the encoder and the decoder are made up of stacks of LSTM cells of the same height. Once the last input time step is encoded, the state of the topmost cell is passed to the first level of the decoder. The first input to the decoder is the last input to the encoder, except that only the signal and bandwidth values are kept to match the format of the output. All following inputs are passed in accordance with teacher forcing. Each output of the final LSTM layer of the decoder is passed to a fully connected layer with ReLU activation, followed by a fully connected linear layer with two output nodes – signal and bandwidth respectively. Refer to the appendix (D.2) for the diagram of the architectures.

5.2 Baselines

To get a good sense of their comparative performance, deep learning classification and regression models (§5.1) were compared against simpler baselines. A natural choice for classification was multi-class logistic regression. Similarly, I picked linear regression for the regression case. However, because of highly periodic data, another very

strong candidate for a regression baseline was the *seasonal naive* method.

A seasonal naive model simply makes a forecast that is equal to the value observed at the same stage of the last season. In our case, the season is the tidal day, which lasts 24 hours and 50 minutes. For example, if the model were to make a prediction for the time interval 13:00 – 16:00 on the 2nd of January, it would output the historical values for 12:10 – 15:10 of the 1st of January, without regard for the inputs to the model. Although conceptually simple, there were several corner cases that had to be handled with care for the sake of fair evaluation.

For example, the model can be asked to make a forecast for a range that is at the beginning of the training data, in which case it has nowhere to look up the value from the last season. To avoid errors, one can fall back on the *naive* method that simply outputs the mean of the input time series. However, this can result in skewed validation results for sensitive metrics, like the mean squared error. To avoid this, samples corresponding to the earliest tidal day were dropped from the validation set.

Another corner case happens when the last season has missing timesteps (see §4.2.2). Dropping all such samples from the validation set, like in the previous case, would mean losing too much validation data. Instead, these missing values are interpolated on the fly. The reason why these interpolated values are not part of the dataset is that these "tears" can be many timesteps long – keeping them would add too much noise to the training data. At the same time, it is only fair that the seasonal naive model is validated on such samples since they occur in practice.

5.3 Making the most of data

The relatively small size of the dataset necessitates the use of special techniques to take full advantage of it. The situation is made worse by the fact that we are dealing with time series, therefore temporal ordering of points must be respected. It is further complicated by the presence of missing values or "tears" in the time series (see §4.2.2). This section goes into details of how these issues were handled.

A straightforward way to split the dataset would be to partition it into disjoint 72-steps long samples. Given a 31k-point dataset (for a single link) this would result in a mere 430 samples, even without accounting for samples that are thrown out because of missing values. After setting aside sizeable chunks of samples for validation and test sets, the training set would remain unacceptably small. This approach would work perfectly well for a dataset orders of magnitude larger than ours. For our case, a much

more involved pipeline was necessary.

At the start, a dataset (for a single link) is split into two: the last 10% are held out for the test set, and the first 90% are allocated for training and validation. The test set is kept until the end to test how the final model would perform in practice. Both the training/validation and the test sets are then processed independently. Each (part of) dataset is split into continuous intervals with no missing values. Then, to generate more samples, each interval is split into overlapping samples with a specified stride. This technique lowers the information density of the dataset since any individual sample now carries less unique information; however, it increases the total amount of information in the dataset as the model does get some new insight from samples that are made up of two disjoint samples. The lower the stride, the less information-dense the dataset becomes. I found that, in our case, lowering the stride beyond 3 gives virtually no performance improvements while substantially increasing the size of the dataset. It was therefore kept at 3 for all of the experiments.

Unfortunately, using overlapping data ensued in yet another challenge. To be able to ensure proper generalisation of a model, the validation set must come from the same distribution as the training set [14]. In the simple case of the disjoint dataset partitioning described above, this is easily achievable by randomly selecting some proportion of samples for the validation set. However, in the case of overlapping samples, this would result in significant overlap between training and validation sets, making estimates of the generalisation error highly biased. One could instead allocate the tail of the dataset for validation purposes, but that would then result in a disparity between the validation and training data distributions, especially for seasonal data. A way to address this is to use k-fold cross-validation (CV), where the dataset is split into k equal continuous parts (k is usually between 5 and 10). The model is then fit k times, each time with a different fold for the validation set and the other $k-1$ folds as the training set. The final generalisation error is then estimated as the mean of k validation errors of the k runs. This gives a better estimate than only using the tail because the model is evaluated on different parts of the dataset, some of which may be "harder" than orders. It is important to note that there is still some overlap in-between bordering folds. Since these overlaps are now constrained in small regions, unlike when samples were shuffled, they can now be removed without losing as much data as before. This variant of CV is known as *hv-block* CV [22]. Figure 5.1 illustrates the described process. Once the overlapping regions are removed, each fold can be internally shuffled without breaking any temporal dependencies or introducing new overlaps. When using

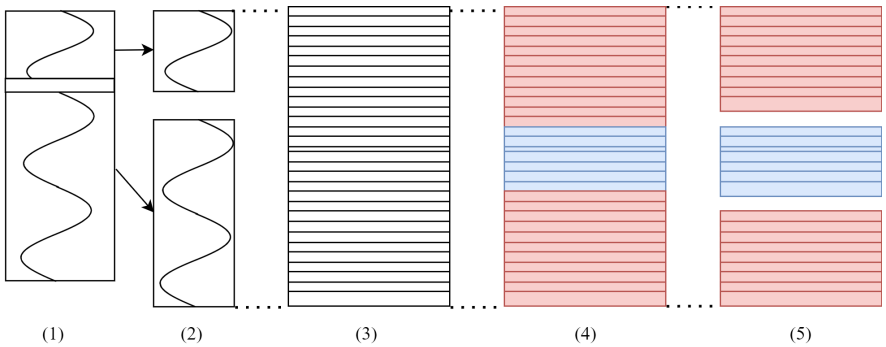


Figure 5.1: (1) Time series with a missing value (tear) (2) gets split into continuous intervals (3) intervals are split into overlapping samples (4) $1/k$ of continuous samples are selected for validation, the rest for training (each run of CV, a different disjoint part is selected) (5) overlap between the validation fold and the training folds is removed

data from multiple links, I ran hv-block CV on each dataset independently and then merged every i-th block with the other i-th blocks from other datasets.

A useful consequence of CV is that it allows estimating standard errors of reported metrics (eq. 5.1). Although runs of k-fold CV are not truly independent among themselves, which is a necessary condition for calculating the standard deviation of a sample statistic, for small enough k's it nonetheless provides a good approximation [1]. I used these estimates to put "errors bars" on the evaluation metrics.

$$SE(\theta) = \frac{SD(\theta)}{\sqrt{k}} \approx \sqrt{\frac{Var(CV_1(\theta), CV_2(\theta), ..., CV_k(\theta))}{k}}, \tag{5.1}$$

where θ is a statistic, $SE(\theta)$ is the standard error of θ, $SD(\theta)$ is the standard deviation of θ, k is the number of folds, Var is sample variance, $CV_i(\theta)$ is θ reported on the i-th run of CV.

5.4 Hyperparameter tuning

To pick good parameters for the models described in §5.1, I heavily relied on distributed hyperparameter tuning provided by the Google Cloud Platform (GCP) as part of their AI platform. It features Bayesian optimisation, which, rather than performing a complete grid search of the parameter space, uses information from previous runs to explore its most promising regions [30]. Because CV makes the training process roughly k times longer, when running hyperparameter optimisation, I only used validation on a single fold, which is equivalent to the "tail validation" mentioned in the

previous section. Although, this results in a suboptimal estimate of the optimisation target, this was a necessary compromise given the high cost of running full CV.

Each optimisation job was set up to run for 50 trials with the validation loss as the optimisation target. Refer to the appendix for more information on which hyperparameters were optimised for each architecture, the settings that were passed to the optimisers and the results that were obtained (D.1, D.2). Moreover, running hyperparameter tuning on the cloud required some specific "scaffolding" and reorganisation of the code. The supplied material has a working sample setup.

5.5 Training and evaluation

This section goes into detail on how the models were trained and evaluated throughout different experiments. The experiments themselves are presented in the next chapter §6.

Before training, all the input time series would get normalised to zero mean and unit variance (standardised) [29]. Categorical cross-entropy and mean squared error (MSE) loss functions were used for training multi-class classification and regression models respectively. Adam optimiser [15] was used with the learning rate of 0.001. At first, there were problems with convergence of regression models. The hypothesis is that it was caused by the two target features – signal strength and bandwidth – having significantly different scales, causing MSE to perform badly as it is scale-dependent [3]. In fact, standardising the target sequences resolved the issue.

When reporting validation MSE, one could rescale the outputs and then report MSE on unscaled signal and bandwidth. However, because signal strength takes orders of magnitude larger absolute values than relative bandwidth, MSE would be dominated by signal residuals, with bandwidth residuals having little to no effect on the measure. Hence, I did not apply inverse normalisation before calculating validation MSE. The resulting metric is useful for comparing models against each other, but it is otherwise difficult to meaningfully interpret it. To better understand the results of regression models, I also calculated the mean absolute percentage error (MAPE), which has a clear interpretation – the average relative deviation of predictions from the true value.

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{y_i - \hat{y}_i}{y_i}\right|; \qquad MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|, \tag{5.2}$$

where y_i is the i-th target value, $\hat{y}_i$ is the i-th prediction, N is the number of samples.

MAPE is scale-invariant, but not shift-invariant, e.g. targets cannot be zero [3], so the standardised outputs needed to be rescaled before calculating the measure. I calculated the average MAPE per target feature across all timesteps and validation/test samples, as well as the average MAPE per timestep to see how performance varies between earlier and later predicted timesteps. MAE was also reported for the evaluation on the test set.

Classification performance was evaluated using the F_1 score [26], which, for the multi-class case, is calculated for each class separately. An important characteristic of F_1, is that it takes into account both precision and recall, highly penalising the score if either one is low. This is especially advantageous for our case since the classes are highly imbalanced. In addition to calculating F_1 scores per class, I also calculated the mean F_1 score of all classes. To prioritise underrepresented classes during training, class weights were passed to the optimiser. A class's weight is the inverse proportion of the number of its occurrences to the total number of samples.

For evaluation, I trained the models with the best-found hyperparameters (see the section on hyperparameter tuning §5.4) using cross-validation with 5 folds. Because of the size of the dataset, training was rather quick. With GPU acceleration it would take at most 5 minutes for the slowest model to converge (on a single fold). To have visual hints of performance of regression models during training, after each fold, two validation samples would get displayed (akin to Figure 6.1): one random sample and the sample that had the highest validation error. I also used early stopping to interrupt training when validation loss would not improve for 5 epochs, restoring to the best-recorded weights.

Special care was taken when implementing experiments that involved removing parts of data from the dataset. For example, when testing the impact of the number of samples in the training set on the validation error, first the k-fold partitioning was performed, then the training procedure dropped the training samples that were farthest from the validation fold. Removing samples before the k-fold split would result in different validation folds every time, making a comparison between different training runs incorrect.

Finally, at the very end, the best performing model was evaluated on the held out test set (see §5.3) to assess how it would perform in production. In this case, like when running hyperparameter optimisation, only the last fold, which is chronologically closest to the test set, was used for validation to perform a timely stop of training.

Chapter 6

Experiments

This chapter describes in detail the experiments that were carried out in this work as well as presents and interprets the results.

6.1 Architecture choice

The first experiment was set up to evaluate different neural network architectures (described in §5.1), comparing them among each other and against baselines (described in §5.2). For this experiment, the models were trained on a full dataset for a single link (link #1). Preliminary hyperparameter tuning was performed to pick optimal hyperparameters for this configuration (refer to the appendix D.1, D.2). The following experiments were based on the architecture that had shown the best results in this experiment. Whilst it is conceivable that runner-up architectures could do better in other experiments, I relied on this heuristic due to the time and resource constraints of this work.

6.1.1 Classification

Table 6.1 summarises results of various classification models, including the logistic regression baseline, tuned CNN and LSTM models, as well as ResNet and FCN architectures as provided by Fawaz et al. [7]. FCN, in particular, was proposed by Wang et al. as a strong baseline for time series classification [32]. First attempts to train FCN were unsuccessful with the model failing to converge. I added learning rate annealing, reducing it by a factor of two when validation loss would not improve for 5 epochs. Then, after a short warm-up, the model showed convergence.

Model	0–15	15–60	60–120	120–240	240+
	0.35±0.02	0.51±0.01	0.44±0.02	0.63±0.03	0.80±0.02
	0.62±0.03	0.55±0.01	0.54±0.02	0.55±0.03	0.70±0.04
Logistic	0.44±0.02	0.53±0.01	0.49±0.02	0.58±0.03	0.74±0.03
	0.54±0.01	0.66±0.02	0.61±0.02	0.74±0.03	0.82±0.02
	0.80±0.02	0.60±0.02	0.60±0.02	0.68±0.02	0.85±0.01
LSTM	0.64±0.01	0.63±0.02	0.60±0.02	0.70±0.02	0.84±0.01
	0.53±0.04	0.65±0.02	0.62±0.02	0.74±0.02	0.85±0.01
	0.75±0.03	0.64±0.05	0.61±0.02	0.73±0.01	0.81±0.02
CNN	0.62±0.03	0.64±0.03	0.61±0.01	0.73±0.01	0.83±0.01
	0.58±0.04	0.66±0.04	0.60±0.02	0.70±0.02	0.83±0.02
	0.79±0.02	0.58±0.03	0.56±0.04	0.71±0.03	0.83±0.02
FCN	0.66±0.03	0.61±0.02	0.58±0.02	0.71±0.03	0.83±0.02
	0.52±0.04	0.69±0.02	0.62±0.04	0.73±0.02	0.82±0.03
	0.82±0.04	0.62±0.05	0.59±0.06	0.66±0.04	0.84±0.02
ResNet	0.64±0.03	0.65±0.03	0.59±0.03	0.69±0.02	0.83±0.01

Table 6.1: Results. Each column corresponds to a class that covers the specified interval after the last input timestep. Each cell has a column of tree measures: precision, recall and F_1 score respectively. Higher is better.

All neural network models substantially outperformed the logistic baseline but had very little difference among themselves. This reinforces the expectation that the biggest bottleneck would be the dataset, to the extent that the choice of an architecture would not play a major role. The LSTM model showed slightly better consistency with a lower spread of values.

A remarkable result is that all models showed relatively low precision with high recall for the first class, which means that there was a high number of false positives. This suggests that the model is convinced that an onset of tidal fading is coming soon but often misses the precise timing, which is unsurprising given that the first bin is only three timesteps long. Possibly, using higher-resolution data or making the first bin longer could improve the precision. Even then, the results already show promise and can provide useful insight – that an onset of tidal fading is imminent or that it is still a long way off, with the most uncertain region in-between the two extremes.

6.1.2 Regression

The results of evaluating regression models are presented in Table 6.2. Both the CNN and the LSTM architectures outperformed both baselines across all metrics. CNN

showed by far the best MSE and better signal MAPE than LSTM, whilst the latter had marginally better bandwidth MAPE.

Model	MSE	MAPE (bw)	MAPE (sig)
Linear	0.674 ± 0.062	0.486 ± 0.019	0.0476 ± 0.0024
Seasonal naive	0.649 ± 0.040	0.383 ± 0.016	0.0419 ± 0.0011
LSTM	0.472 ± 0.054	$\mathbf{0.286 \pm 0.020}$	0.0372 ± 0.0027
CNN	$\mathbf{0.359 \pm 0.044}$	0.328 ± 0.018	$\mathbf{0.0331 \pm 0.0018}$

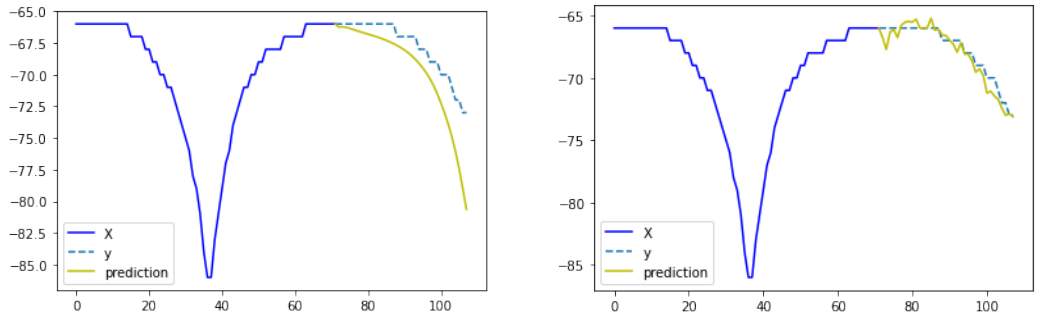
Table 6.2: Regression results. Lower is better. “bw” and “sig” stand for bandwidth and signal respectively.

Interestingly, LSTM outputs were much smoother compared to CNN outputs. Figures 6.1a and 6.1b show the same random sample predicted by LSTM and CNN models respectively – the CNN model captured jaggedness of the signal curve and the LSTM model did not. A possible explanation could be that the LSTM model is shallow and the only recurrent layer has a smooth activation function (*tanh*), while the CNN network has 12 convolutional layers with *ReLU* activations, which are inherently jagged.

Despite a much larger number of layers and a larger number of parameters (148k parameters against LSTM’s 13k), the CNN model was substantially faster to train. This comes as no surprise since the CNN model is parallelisable in the time domain, whereas the LSTM model requires at least 72 sequential steps (the number of input timesteps) to compute the output. Because in our case the convolutional layers themselves were relatively compact, even when training on a CPU, one epoch would take a couple of seconds to complete, and it would take an average of 30 epochs to converge. Conversely, without a GPU, training the LSTM for one epoch would take tens of seconds, and it would take around 30-50 epochs to converge. Whilst on a GPU the difference in training time becomes negligible, this is an important advantage of the CNN model because it can be efficiently run on cheap server hardware.

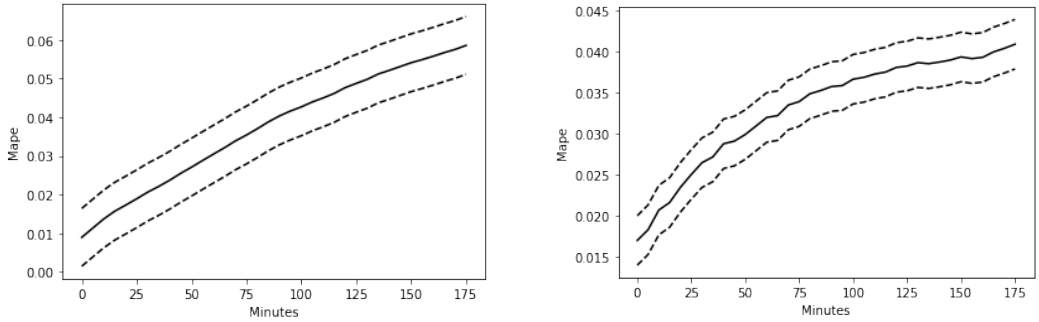
Figures 6.2a and 6.2b show signal MAPE for all timesteps individually. The charts suggest that LSTM performs slightly better for immediate predictions, but falls behind for farther timesteps. On the other hand, it consistently showed slightly better results in predicting bandwidth (see Figure 6.3).

Expectedly, the seasonal naive baseline performed very well in noiseless regions, often much better than any other model with almost perfect fits in situations that mirror



(a) LSTM (b) CNN

Figure 6.1: Signal predictions of a random validation sample.

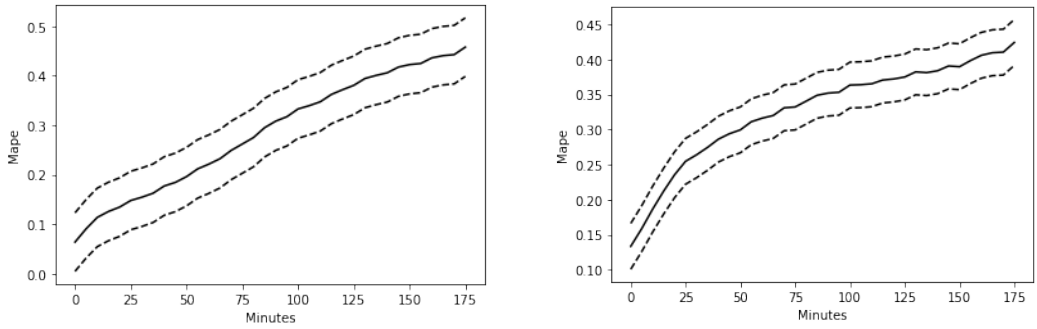


(a) LSTM (b) CNN

Figure 6.2: Stepwise signal MAPE for LSTM and CNN. Dashed lines denote standard errors in particular timesteps.

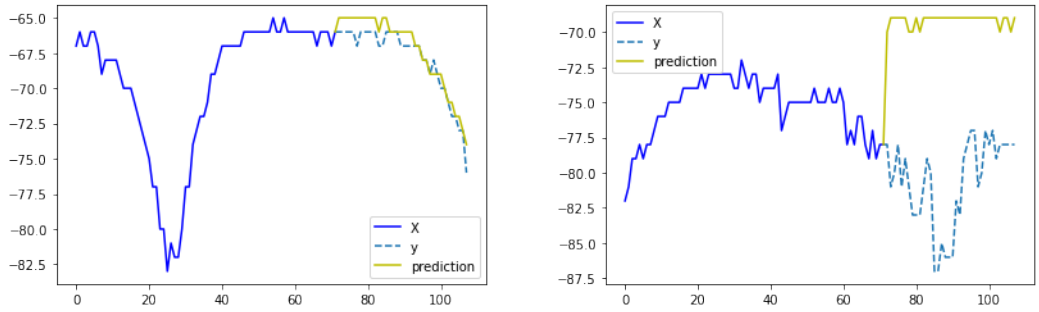
the last tidal day (Figure 6.4a), but then it would completely fail in irregular situations (Figure 6.4b). As can be seen from the results, these situations happen often enough that the seasonal naive model lost to the neural network models by a significant margin. Another disadvantage of the seasonal model is that it is not adaptive – it will not change its predictions upon seeing new data, unlike the other models.

Neural networks show promising results having witnessed only, roughly, 200 tidal cycles. Of course, the obvious drawback is the necessity to collect a dataset for the link to be able to train a model tailored specifically for it. One of the following experiments (§6.3) tests whether these models are able to generalise to other radio links on a dataset as small as ours.



(a) LSTM (b) CNN

Figure 6.3: Stepwise bandwidth MAPE for LSTM and CNN. Dashed lines denote standard errors in particular timesteps.



(a) Good fit in a repeating situation (b) Bad fit in a changing situation

Figure 6.4: Seasonal naive predictions of two samples.

6.2 Ablation study

In this experiment, different sets of input features were removed from the dataset to see how it would affect validation performance. For the regression test, I used the CNN model from the architectural experiment (§6.1) because it had shown the best performance across almost all measures and it was substantially faster to train than the LSTM model. Similarly, for the classification test, I used the CNN classifier – although all classification models showed practically the same results, the CNN model was the easiest to train. To reuse previous results, the model was trained on the same dataset for the same link as in §6.1. First, all weather features were stripped from the dataset, then also the sea level time series. For the classification test, I also tried keeping only signal or bandwidth in the dataset. This was not done for regression because the output includes both signal and bandwidth. The results of the regression experiment are shown in Table 6.3.

Features	MSE	MAPE (bw)	MAPE (sig)
{s,b,w,t}	0.359± 0.044	0.328±0.018	0.0331± 0.0018
{s,b,t}	**0.349± 0.044**	**0.320±0.020**	**0.0312±0.0020**
{s,b}	0.413± 0.038	0.374± 0.020	0.0344 ± 0.0016

Table 6.3: Regression results. Lower is better. "s,b,w,t" denote signal, bandwidth, weather data, tidal height respectively. "bw" and "sig" denote bandwidth and signal respectively.

The results suggest that weather data had no significant impact on validation performance for the regression. In fact, the model trained on everything except weather data showed slightly better average performance than the one trained on full data, though the results are well within standard error. This can be attributed to training variance or to a slight overfit of the complete model to weather data. Removing tidal data does have a significant adverse effect on all metrics. This conforms with prior analysis which showed that there was relatively strong dependence between tidal height and signal (Figure 3.4).

Features	0–15	15–60	60–120	120–240	240+
	0.53±0.04	0.65±0.02	0.62±0.02	0.74±0.02	0.85±0.01
	0.75±0.03	0.64±0.05	0.61±0.02	0.73±0.01	0.81±0.02
{s,b,w,t}	0.62±0.03	0.64±0.03	0.61±0.01	**0.73±0.01**	**0.83±0.01**
	0.67±0.04	0.75±0.01	0.63±0.03	0.72±0.02	0.83±0.02
	0.81±0.04	0.72±0.02	0.68±0.01	0.70±0.02	0.81±0.03
{s,b,t}	**0.73±0.03**	**0.73±0.01**	**0.65±0.02**	**0.71±0.02**	**0.82±0.02**
	0.62±0.04	0.64±0.02	0.51±0.02	0.65±0.02	0.81±0.01
	0.78±0.04	0.62±0.01	0.58±0.03	0.60±0.03	0.80±0.02
{s,b}	0.69±0.03	0.63±0.06	0.54±0.02	0.62±0.02	0.80±0.01
	0.54±0.02	0.62±0.02	0.49±0.01	0.62±0.02	0.81±0.01
	0.75±0.06	0.56±0.02	0.58±0.02	0.58±0.03	0.77±0.02
{s}	0.63±0.03	0.59±0.02	0.53±0.02	0.60±0.02	0.79±0.01
	0.67±0.04	0.52±0.02	0.42±0.03	0.62±0.02	0.81±0.01
	0.79±0.03	0.50±0.04	0.52±0.03	0.55±0.02	0.80±0.01
{b}	0.72±0.03	0.50±0.03	0.46±0.03	0.58±0.02	0.80±0.01

Table 6.4: Classification results. Each column corresponds to a class that covers the specified interval after the last input timestep. Each cell has a column of three measures (from top to bottom): precision, recall and F_1 score. Higher is better.

The classification results, shown in table 6.4, reveal interesting insight. First of all, the model trained on the complete dataset shows clear signs of overfitting to the weather data. This conclusion can be made from the significantly lower performance of the full model for the first two classes. Hypothetically, this model could have learned to ignore the weather inputs to perform at least as well as the model that was trained on everything except weather; instead, it showed worse validation performance. Another observation is that, similar to the regression case, tidal height data provides useful information. Here we also see that it mostly helps with predictions of the middle bins – from 15 to 240 minutes. Same can be said about signal strength, which seems to explain these classes better than bandwidth. Bandwidth, on the other hand, proved to be key in predicting the first class. This is probably because the model detects the sharp drop from the plateau (see §4.1.1). Consequently, the model that combines the three features – signal, bandwidth and sea level – shows the best results across the board.

Whilst these results suggest that weather data is detrimental in a dataset of this size, one cannot rule out the possibility that it would prove useful on much larger scales, where the weather effects become more apparent. Therefore, it is worth repeating the experiment when more data becomes available.

6.3 Generalisation across multiple links

A significant disadvantage of previously tested models is they are tailored to a specific radio link. One has to first gather a dataset for a particular link, then train a model specific to that link. It took three and a half months of monitoring to achieve the above results. For some situations, this may be too much, and it would be useful to have a general model that can work for multiple links. In this experiment, I tested whether our models can generalise across several links. The methodology is as follows: one model was trained on a dataset for a single link, another model was trained on two datasets of two other links, but evaluated on the same validation set as the first model. The results of the two models were then compared. The experiment was carried out using the CNN models, which showed good results in the performance experiment, trained on signal strength, bandwidth and tidal height data, which proved to be the optimal feature set in the ablation experiment.

For regression, only MAPE measures were used for comparison because, when training models on different sets of data, the outputs end up scaled differently; hence, the scale-dependent MSE would yield unfair comparison (though, one could unscale

outputs and then evaluate MSE for bandwidth and signal separately). The regression case is also tested against the naive (see §5.2) and the seasonal naive models fit on the target link.

Target link	Target-trained	General	Seasonal	Naive
#1	0.33±0.02 0.03±0.00	0.56±0.02 0.05±0.00	0.38 ± 0.02 0.04 ± 0.00	0.68 ± 0.02 0.06 ± 0.00
#2	0.10±0.01 0.02±0.00	0.13±0.00 0.03±0.00	0.12±0.01 0.03±0.00	0.15±0.02 0.04±0.00
#3	0.27±0.03 0.04±0.00	0.45±0.06 0.09±0.00	0.30±0.04 0.05±0.00	0.51±0.05 0.08±0.01

Table 6.5: Regression results. Lower is better. Each cells contains a column of two measures (from top to bottom): bandwidth and signal MAPE. The "target-trained" model was evaluated and trained on a dataset for the same link, whereas the general model was trained on data for the two other links.

Regression results, shown in table 6.5, show that whilst the general models outperform the naive model, which simply outputs the average of the input, they fall behind seasonal naive models. This makes them not very practical in the current form. Classification results appear even less successful and reveal a deeper problem (Table 6.6). Even the models that were trained on data for the target link performed poorly, with the exception of the first link. Consequently, general models showed even worse performance. Section §4.3 hints at the possible explanation for such outcome – the second and the third links exhibit more seasonal variation than the first link. As a result, these links have long regions that have less (or even zero) tidal fading onsets than others, at least according to our definition for an onset of tidal fading (see §4.1.4). It is plausible that the very small datasets that we worked with was enough for the model to learn to somewhat decently predict tidal onsets for the more consistent first link, but not the other two links.

As before, the results suggest that the best direction for improvement is a larger dataset – with more links for better generalisation and with more data per link. As it stands, our current dataset is not suitable for training practical general models. There are also techniques worth exploring that may prove useful in achieving generalisation other than gathering raw data. One could attempt augmenting the dataset by scaling, shifting, adding noise, etc., to the input timeseries to add more variety to the dataset and synthesise more training samples. One could even try to simulate non-existing

Model	0–15	15–60	60–120	120–240	240+
#1 Target	0.73±0.03	0.73±0.01	0.65±0.02	0.71±0.02	0.82±0.02
#1 General	0.54±0.04	0.48±0.05	0.39±0.03	0.42±0.04	0.65±0.02
#2 Target	0.50±0.03	0.53±0.04	0.48±0.4	0.56±0.05	0.68±0.04
#2 General	0.29±0.08	0.26±0.08	0.23±0.07	0.39±0.06	0.48±0.06
#3 Target	0.48±0.05	0.48±0.05	0.41±0.05	0.53±0.05	0.76±0.05
#3 General	0.28±0.06	0.14±0.04	0.19±0.04	0.14±0.05	0.47±0.07

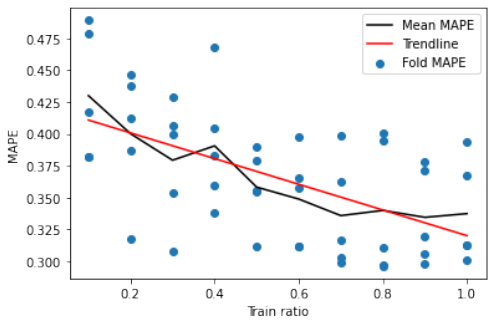
Table 6.6: Classification F-scores. Higher is better. The “target” model was evaluated and trained on a dataset for the same link, whereas the general model was trained on data for the two other links.

links by modelling the physics of tidal fading (§3). This should be attempted with care and consideration so as to not generate irrelevant data that just ends up reducing the signal-to-noise ratio of the dataset. Hence, this topic requires its own dedicated research. Another potentially useful technique in the event that there is a lot of data for some links but less for others, is transfer learning – a model trained on a dataset for one link (or some links) may learn low-level features that can be reused for other links with some fine-tuning, requiring less data than when training a new model from scratch [6].

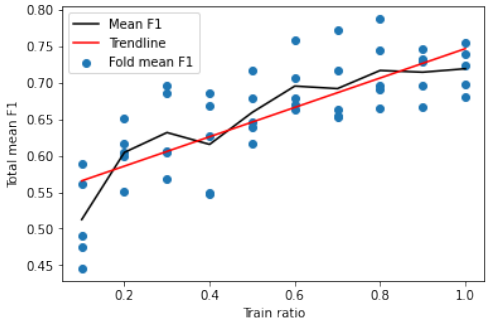
6.4 Impact of dataset size

In this experiment, I tested how the size of the training set affects the performance of the previously tested models. This is either to confirm or cast doubt on the idea that training our models on more data would significantly improve performance. Same as in the previous experiments, I used the CNN models trained on a dataset for a single link with all features except weather data. I ran full cross-validation on training sets of varying size: from 10% to 100% of training samples with a 10% step, while keeping validation folds 100% of the original size (see §5.5 for preprocessing details). Then I plotted validation metrics against percentages of the training set used to get an idea of how the two are related. Again, because different training data is used each time, the outputs get scaled differently, so plotting validation loss, which is the scale-dependent MSE, could be misleading. Instead, in the regression case, I plotted the validation

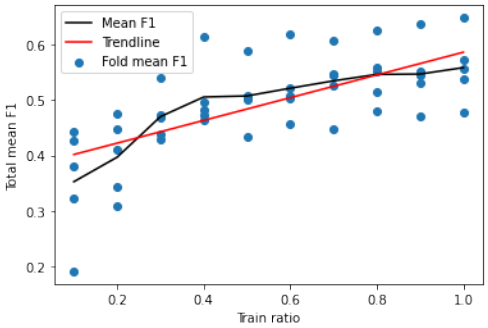
MAPE of bandwidth. For classification, I plotted the mean F_1 score of all classes, which is easier to interpret than categorical cross-entropy validation loss.



(a) Regression, Link #1



(b) Classification, Link #1



(c) Classification, Link #2

Figure 6.5: Plots of validation metrics against the ratio of the train set used.

Plots on Figure 6.5 show clear evidence of validation score improvement with adding more training data. Linear fit for the regression chart 6.5a has a slope of -0.1, indicating a 10% improvement in MAPE per tenfold increase of the size of the dataset. Of course, the underlying relationship is not linear – MAPE cannot go below 0 – and will inevitably slow down, but the plots show no signs of the model reaching its capacity – only a slight curvature at the end which cannot be distinguished from training variance. The same is true for the two classification models. Interestingly, both classification charts (6.5b, 6.5c) show a trend line with a slope of 0.2, despite the model for the 2nd link significantly underperforming, as was shown in Table 6.4. This gives hope that, although onsets of tidal fading proved to be difficult to predict for links #2 and #3, given enough data, the classifiers for those links could still catch up, indicating that the bottleneck for classification is the dataset size rather than some inherent problem with the consistency of the onset definition.

6.5 Test set evaluation

In this final test, I report how our models of choice would perform in practice, specifically, for 10 days in the middle of June, which corresponds to the last 10% of the dataset. In the previous experiments, the CNN architecture proved to be computationally efficient and showed top performance across most metrics. Meanwhile, ablation study revealed that, at least for the current dataset size and architectures, it is best not to include weather data in the training set, unlike tidal height data, which noticeably improved validation performance. Therefore, the same settings were used for this test. The evaluation was performed on the whole test set at once, so standard error estimates could not be derived. Table 6.7 summarises evaluation performance of the regression CNN model on three held out test sets for each link, matching it against the seasonal naive baseline. In addition to MAPE, I also reported the Mean Absolute Error MAE as it may have a clearer practical interpretation due to the shift invariance – the average of absolute residuals. For clarity, bandwidth MAE was rescaled by the corresponding bandwidth reference values (see §4.1.1). Table 6.8 shows the classification results. It can be noted that the test set proved to be easier than the training set for links #1 and #2.

Link	MAPE (bw)	MAE (bw)	MAPE (sig)	MAE (sig)
#1	0.342	13.59	0.025	1.66
#2	0.110	16.04	0.019	2.47
#3	0.212	15.80	0.034	2.29

Table 6.7: Regression results for each link. “bw” and “sig” stand for bandwidth and signal respectively.

Link	0–15	15–60	60–120	120–240	240+
#1	0.76	0.77	0.73	0.78	0.87
#2	0.66	0.65	0.58	0.63	0.80
#3	0.57	0.48	0.50	0.52	0.81

Table 6.8: Classification results (F_1 scores) for each link.

Chapter 7

Conclusions

This work was the first-ever attempt at forecasting performance of radio links affected by tidal fading, covering the whole process of designing and training deep learning models, including data acquisition, data processing, hyperparameter tuning and evaluation. In addition, it provided a comprehensive explanation for the phenomenon of tidal fading – an undeservedly understudied topic.

Using roughly three months worth of data for three tide-affected links, a number of different experiments were carried out to test different aspects of the proposed models. It was shown that, despite a relatively small dataset, deep learning methods were able to outperform simpler baselines. The choice of the architecture did not play a major role at this scale, but the choice of input features did – whilst sea level data proved to be a beneficial addition to the training set, the models were not able to make good use of weather data. It is entirely possible that deep learning models would find useful patterns in weather data on a sufficiently large dataset as there is strong theoretical evidence for it. Another remaining problem is training general models that can predict performance of links that were not shown to the models during training. It is expected that this is feasible with more link diversity in the dataset and more data per link. In general, accumulating more data was found to be the most promising way forward.

Other potential directions for further experiments include using transfer learning for reusing feature extraction trained on a larger dataset for links with less data as well as using data augmentation and simulations to synthesise new training data. On much greater amounts of data, some deep learning architectures might start taking lead over others, so the choice of the architecture also remains an open question.

This dissertation can be a starting point for further work in the area. The pipeline for acquiring, processing data and training deep learning models, which was developed

as part of this dissertation, can now be used to swiftly train models for use in practical applications or to carry further research on a larger dataset, when more data becomes available. The supplementary material features data, tooling and code used in this work. In the near future, I intend to put the results of this work into practice and build a tool for real-time prediction of performance of tide-affected links in the Tegola network.

Bibliography

[1] Yoshua Bengio and Yves Grandvalet. No unbiased estimator of the variance of k-fold cross-validation. *Journal of machine learning research*, 5(Sep):1089–1105, 2004.

[2] Giacomo Bernardi, Peter Buneman, and Mahesh Marina. Tegola tiered mesh network testbed in rural scotland. pages 9–16, 01 2008.

[3] Alexei Botchkarev. Performance metrics (error measures) in machine learning regression, forecasting and prognostics: Properties and typology. *arXiv preprint arXiv:1809.03006*, 2018.

[4] Vitor Cerqueira, Luis Torgo, and Igor Mozetic. Evaluating time series forecasting models: An empirical study on performance estimation methods. *arXiv preprint arXiv:1905.11744*, 2019.

[5] François Chollet et al. Keras. `https://keras.io`, 2015.

[6] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Transfer learning for time series classification. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 1367–1376. IEEE, 2018.

[7] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, 2019.

[8] Miguel Gutiérrez Gaitán, Luis Pinto, Pedro Miguel Santos, and Luís Almeida. On the two-ray model analysis for overwater links with tidal variations. In *11º Simpósio de Informática*, 2019.

[9] Ivan D. Haigh. *Tides and Water Levels*, pages 1–13. American Cancer Society, 2017.

[10] Zhongyang Han, Jun Zhao, Henry Leung, and King Ma. A review of deep learning models for time series prediction. *IEEE Sensors Journal*, PP:1–1, 06 2019.

[11] Ralph VL Hartley. Transmission of information 1. *Bell System technical journal*, 7(3):535–563, 1928.

[12] Christopher Haslett. *Essentials of radio wave propagation*, volume 91. Cambridge University Press Cambridge, 2008.

[13] Tianxing He, Jingzhao Zhang, Zhiming Zhou, and James Glass. Quantifying exposure bias for neural language generation. *arXiv preprint arXiv:1905.10617*, 2019.

[14] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer.

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

[16] Ronald Klimberg, George Sillup, Kevin Boyle, and Vinay Tavva. Forecasting performance measures - what are their practical meaning? *Advances in Business and Management Forecasting*, 7:137–147, 11 2010.

[17] Alex M Lamb, Anirudh Goyal Alias Parth Goyal, Ying Zhang, Saizheng Zhang, Aaron C Courville, and Yoshua Bengio. Professor forcing: A new algorithm for training recurrent networks. In *Advances in neural information processing systems*, pages 4601–4609, 2016.

[18] A. Macmillan, M. K. Marina, and J. T. Triana. Slow frequency hopping for mitigating tidal fading on rural long distance over-water wireless links. In *2010 INFOCOM IEEE Conference on Computer Communications Workshops*, pages 1–5, 2010.

[19] Samuel Mason. Atmospheric effects on radio frequency (rf) wave propagation in a humid, near-surface environment. page 86, 03 2010.

[20] A. Mehrabani and H. Golnabi. Investigation of humidity effect on the air refractive index using an optical fiber design. *Journal of Applied Sciences*, 11(16):3022–3027, December 2011.

[21] Miguel Pereira. Spread spectrum techniques in wireless communication part 2: Transmission issues in free space. *IEEE instrumentation & measurement magazine*, 13(1):8–14, 2010.

[22] Jeff Racine. Consistent cross-validatory model-selection for dependent data: hv-block cross-validation. *Journal of econometrics*, 99(1):39–61, 2000.

[23] Jimmy SJ Ren and Li Xu. On vectorization of deep convolutional neural networks for vision tasks. *arXiv preprint arXiv:1501.07338*, 2015.

[24] D.A. Ross. *Introduction to Oceanography.* New York, NY: HarperCollins., 1995.

[25] Matteo Sangiorgio and Fabio Dercole. Robustness of lstm neural networks for multi-step forecasting of chaotic time series. *Chaos, Solitons & Fractals*, 139:110045, 2020.

[26] Yutaka Sasaki. The truth of the f-measure. *Teach Tutor Mater*, 01 2007.

[27] Florian Schmidt. Generalization in generation: A closer look at exposure bias. *arXiv preprint arXiv:1910.00292*, 2019.

[28] Sreelekshmy Selvin, R Vinayakumar, EA Gopalakrishnan, Vijay Krishna Menon, and KP Soman. Stock price prediction using lstm, rnn and cnn-sliding window model. In *2017 international conference on advances in computing, communications and informatics (icacci)*, pages 1643–1647. IEEE, 2017.

[29] Murali Shanker, Michael Y Hu, and Ming S Hung. Effect of data standardization on neural network training. *Omega*, 24(4):385–397, 1996.

[30] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[31] H.V. Thurman. *Introductory Oceanography.* New York, NY: Macmillan., 7 edition, 1994.

[32] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International joint conference on neural networks (IJCNN)*, pages 1578–1585. IEEE, 2017.

[33] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989.

[34] Zheng Zhao, Weihai Chen, Xingming Wu, Peter CY Chen, and Jingmeng Liu. Lstm network: a deep learning approach for short-term traffic forecast. *IET Intelligent Transport Systems*, 11(2):68–75, 2017.

Appendix A

Equations

A.1 Fresnel equations

$$\Gamma_{hp} = \frac{-\varepsilon_r \sin\theta + \sqrt{\varepsilon_r + \cos^2\theta}}{\varepsilon_r \sin\theta + \sqrt{\varepsilon_r + \cos^2\theta}}; \Gamma_{vp} = \frac{\sin\theta - \sqrt{\varepsilon_r + \cos^2\theta}}{\sin\theta + \sqrt{\varepsilon_r + \cos^2\theta}} \tag{A.1}$$

where Γ_{hp} is the attenuation factor for horizontal polarization, Γ_{vp} – for vertical polarization, θ is the angle of incidence of the reflected beam and ε_r is the electrical permittivity ratio between air and water. Γ for circular polarisation can be worked out from Γ_{hp} and Γ_{vp} [21].

A.2 Path length difference in static water

$$\begin{aligned}
&\delta = l - (r_1 + r_2) = \sqrt{d^2 + (h_1 + h_2)^2} - \sqrt{d^2 + (h_1 - h_2)^2} = \\
&= d\left(\sqrt{1 + \frac{(h_1 + h_2)^2}{d^2}} - \sqrt{1 + \frac{(h_1 - h_2)^2}{d^2}}\right) \approx \qquad (1) \\
&\approx 0.5d\left(\frac{(h_1 + h_2)^2}{d^2} - \frac{(h_1 - h_2)^2}{d^2}\right) = \frac{2h_1 h_2}{d}
\end{aligned}$$

Refer to figure 3.1 for notation. In step (1), the taylor expansion for $\sqrt{1+x}$ is used, keeping only the first two terms.

A.3 Path length difference in tidal water

We take the result from A.2 and account for water displacement t:

$$\delta \approx \frac{2(h_1 - t)(h_2 - t)}{d}, \tag{A.2}$$

If t changes from 0 (low tide) to T (high tide), then δ changes by:

$$\Delta\delta = \frac{2T(-h_1 - h_2 + T)}{d}, \tag{A.3}$$

A.4 Angle approximations

$$\alpha_1 = \arctan\left(\frac{h_1 + h_2}{d}\right) - \arctan\left(\frac{h_2 - h_1}{d}\right) \approx \frac{2h_2}{d} \tag{A.4}$$

$$\alpha_2 = \arctan\left(\frac{h_1 + h_2}{d}\right) - \arctan\left(\frac{h_1 - h_2}{d}\right) \approx \frac{2h_1}{d} \tag{A.5}$$

Here we use the fact that $tan(\alpha) \approx \alpha$ for small α and that $d \gg h_1 + h_2$.

Appendix B

Screenshots

Figure B.1: Locations of the links and the gauging station

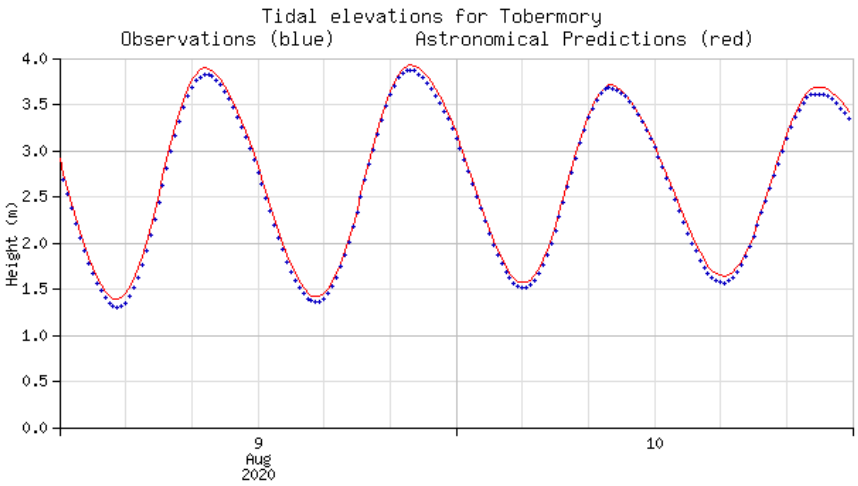
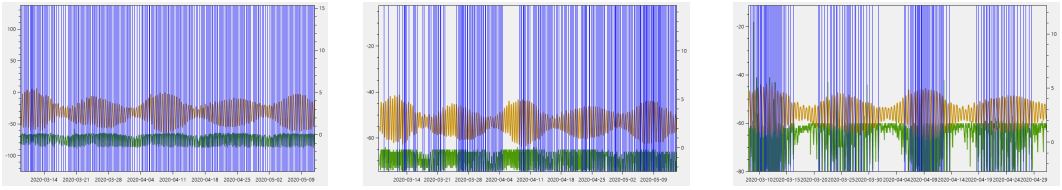


Figure B.2: Real-time tidal height observations and predictions as provided by NTSLF



(a) Link #1 (b) Link #2 (c) Link #3

Figure B.3: Effects of tidal seasonality on the labelling for the three links. Blue lines indicate onsets of tidal fading as per definition in §4.1.4.

Appendix C

Tools

C.1 Using unms-stats

C.1.1 Setting up a bash script run.sh

```
#!/bin/bash
DEVICE1="...device..id..."
DEVICE2="...device..id..."

STATS="signal uplinkCapacity downlinkCapacity"
ENDPOINT=".../nms/api/v2.1"
API_KEY="...api_key..."
DB=".../data.db" # path where to save data

python signal_logger.py -k $API_KEY \\
                        -e $ENDPOINT \\
                        -d $DEVICE1 $DEVICE2 \\
                        -s $STATS \\
                        -db $DB
```

C.1.2 Setting up a cronjob

```
*/5 * * * * some_path/run.sh > some_path/log.txt 2>&1
```

NB: It is not necessary to run the script every 5 minutes – it will fetch all statistics cached on the device. Additionally, it is recommended to periodically create a backup

of data.db using any preferred method.

C.2 Fecthing data from AirControl

To fetch 5-minute timestep data for any range from AirControl:

1. Open the UI for the corresponding link in Google Chrome.
2. Open the network tab in the Chrome's developer console.
3. Use the web UI to select the datetime range and metrics of interest.
4. "metric" entry should appear in the network tab. Right click it, hover over Copy and select copy as cURL (bash).

In the copied cURL command, under –data-binary, change the field "scale" from "hours" to "minutes" and execute the request. This will fetch all of the data in the specified range with a 5-minute step. This must be done before the cookie expires, so it is not trivial to automate reliably.

C.3 GUI for labelling

The tool is written in C# using .NET WinForms and only works in Windows. When starting the tool, it prompts to select the input dataset for labelling. Once selected, the main window with an interactive plot opens. The plot is based on OxyPlot.NET and allows the user to rescale, drag and shift any axes with the help of mouse buttons and a mouse wheel.

The controls for the application are presented in Table C.1. Frozen markers cannot be removed until unfrozen. Intersecting the plot with a horizontal line generates a marker every time when the signal line crosses the line from the top.

Key	Description
Left click	Adds a marker at the position closest to the cursor
r	Removes the closest marker to the cursor
f	Freezes all markers left of the cursor
u	Unfreezes all markers
b	Switches the secondary axis between bandwidth and sea level
]	Removes all markers
i	Intersects the signal line with a horizontal line at the cursor position
CTRL+a	Set ref value to cursor position and run the algorithm §4.1.4
l	Load labels from disk
CTRL+s	Save labels to disk

Table C.1: Control of the GUI tool

Appendix D

Architectures

D.1 CNN

D.1.1 Diagram

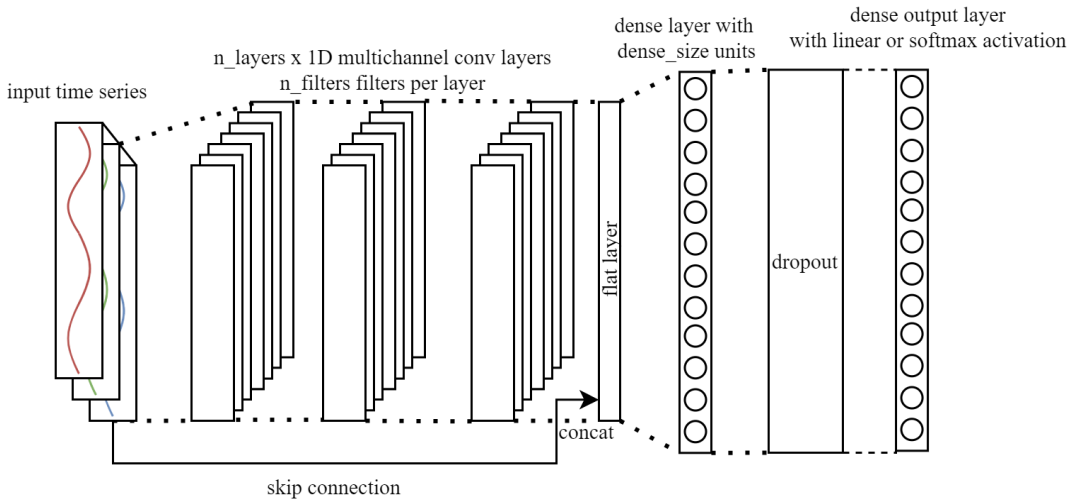


Figure D.1: Diagram of the CNN architecture

D.1.2 Tuned hyperparameters

Refer to Figure D.1 for disambiguation of parameter names.

1. n_filters from 1 to 100, log scale
2. n_layers from 1 to 12, linear scale
3. dropout from 0 to 0.7, linear scale

4. dense_size from 8 to 512, log scale

D.1.3 Tuning results

rank	val_loss	n_filters	n_layers	dropout	dense_size
1	0.819	73	3	0.5938	11
2	0.830	100	3	0.7	8
3	0.833	83	3	0.7	8

Table D.1: Top 3 results (out of 50) for classification.

rank	val_loss	n_filters	n_layers	dropout	dense_size
1	0.417	7	12	0	32
2	0.431	4	5	0	47
3	0.437	6	6	0	47

Table D.2: Top 3 results (out of 50) for regression.

D.2 LSTM

D.2.1 Diagram

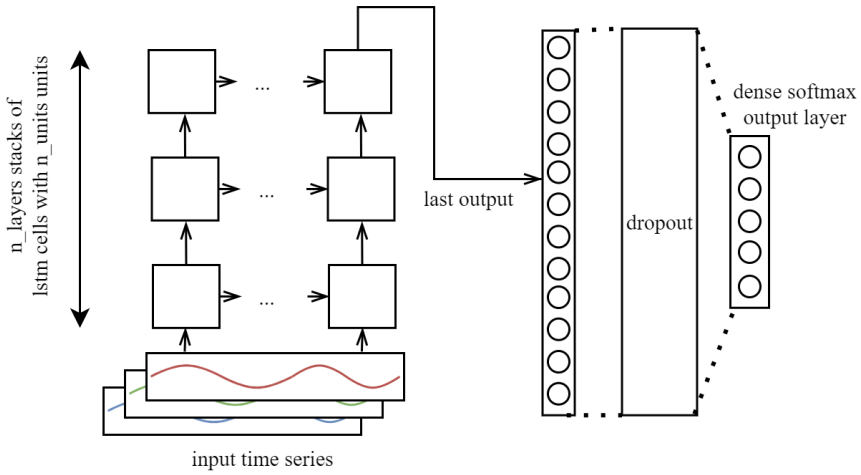


Figure D.2: Diagram of the LSTM classification architecture

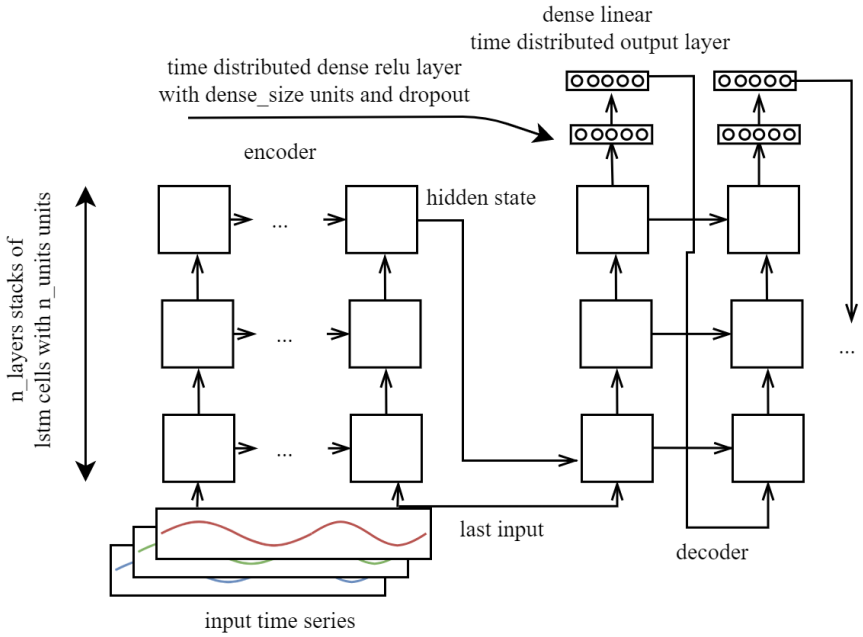


Figure D.3: Diagram of the LSTM regression architecture

D.2.2 Tuned hyperparameters

Refer to Figures D.2 and D.3 for disambiguation of parameter names.

1. n_units from 8 to 512, log scale
2. n_layers from 1 to 4, linear scale
3. dropout from 0 to 0.7, linear scale
4. dense_size from 8 to 512, log scale

D.2.3 Tuning results

rank	val_loss	n_units	n_layers	dropout	dense_size
1	0.684	8	1	0	382
2	0.759	512	4	0.107	8
3	0.760	512	4	0.437	512

Table D.3: Top 3 results (out of 50) for classification.

rank	val_loss	n_units	n_layers	dropout	dense_size
1	0.0500	21	1	0	373
2	0.0509	16	1	0	153
3	0.0512	21	1	0	373

Table D.4: Top 3 results (out of 50) for regression.