Deep Active Learning with Artificial Neural Networks for Automatic Detection of Mercury's Bow Shock and Magnetopause Crossing Signatures in NASA's MESSENGER Magnetometer Observations

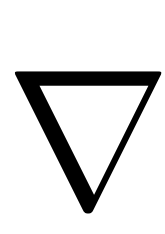
Bachelor's Thesis by

Nikolas Kirschstein

January 28, 2022

Advisor	Primary Examiner	Secondary Examiner
Sahib Julka	Prof. Dr. Michael Granitzer	Prof. Dr. Tomas Sauer
Chair of Data Science	*Chair of Data Science*	*Chair of Digital Image Processing*

Abstract

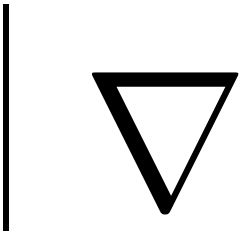


Between 2011 and 2015, NASA[1]'s MESSENGER[2] spacecraft orbited Mercury, where it collected magnetic measurements through the onboard magnetometer. With these, several studies attempted to geometrically model the exact shape of the planet's bow shock and magnetopause boundaries. However, despite being highly complex, these static models struggle to adapt to changing environments. In cases like this, where it is necessary to capture fine structures in discrete signals, deep learning has been able to outperform traditional modeling in various applications over the last decade. Hence, we devise a deep neural network that identifies the spacecraft's bow shock and magnetopause crossings within a local time window of measurements and predicts upcoming crossings beyond that window. The model achieves an overall macro F1 of 0.82 and accuracies of 80 % and 88 % on the bow shock and magnetopause crossings, respectively. Furthermore, we employ an active learning paradigm to determine how many Mercury years' worth of observations are required for a representative model. We find that two Mercury years' worth of measurements are sufficient for a satisfactory performance, which is only about a tenth of the entire data. This work may be relevant to future research concerning the *BepiColombo* mission by ESA[3] and JAXA[4], whose space probes will enter orbit around the planet in December 2025.

Deutsche Kurzfassung: Von 2011 bis 2015 befand sich die NASA-Raumsonde MESSENGER auf einer Umlaufbahn um den Merkur, wo sie mit ihrem Magnetometer Messdaten sammelte. Auf deren Grundlage versuchten einige Studien, die Bereiche der Bugstoßwellen und Magnetopause geometrisch zu modellieren. Trotz ihrer Komplexität können diese statischen Modelle nur schwer auf sich verändernde Umgebungen adaptiert werden. In solchen Fällen, wo feine Strukturen in diskreten Signalen erfasst werden müssen, konnte Deep Learning in den vergangenen Jahren klassische Modellierungsmethoden in verschiedenen Anwendungen übertreffen. Daher entwickeln wir ein tiefes neuronales Netzwerk, das Überquerungen von Bugstoßwellen und der Magnetopause durch das Raumfahrzeug innnerhalb eines lokalen Zeitfensters an Messdaten identifiziert und bevorstehende Überquerungen außerhalb dieses Fensters vorhersagt. Das Modell erreicht einen Makro F1-Wert von 0.82 sowie eine Korrektklassifikationsrate von 80 % bzw. 88 % für die Bugstoßwellen bzw. Magnetopause. Darüber hinaus verwenden wir ein aktives Lernparadigma, um zu bestimmen, wie viele Merkurjahre an Daten für ein repräsentatives Modell erforderlich sind. Wir stellen fest, dass die Messungen von zwei Merkurjahren für eine zufriedenstellende Leistung ausreichen, was nur etwa einem Zehntel der gesamten Daten entspricht. Diese Arbeit könnte für künftige Forschungen im Zusammenhang mit der BepiColombo-Mission von ESA und JAXA relevant sein, deren Raumsonden im Dezember 2025 in Umlaufbahnen um den Planeten eintreten werden.

[1] **N**ational **A**eronautics and **S**pace **A**dministration
[2] **ME**rcury **S**urface, **S**pace **EN**vironment, **GE**ochemistry, and **R**anging
[3] **E**uropean **S**pace **A**gency
[4] **J**apan **A**erospace E**X**ploration **A**gency

Acknowledgements



First and foremost, I am grateful to my advisor Sahib Julka for letting me participate in his research, for providing helpful advice in our weekly meetings, for his remarkable responsiveness to late-night e-mails, and for his thorough feedback on the thesis drafts.

Next, I want to thank my examiners, Prof. Granitzer and Prof. Sauer, for approving the thesis topic. Moreover, I thank Prof. Granitzer for allowing me to use the staff computing cluster `drogon` and Prof. Sauer for an enlightening discussion about measures of uncertainty and information theory in general.

Huge gratitude is due to my flatmates and friends Ben and Kassian for providing me with mental support and enduring vociferous singing throughout the writing process. Thank you to Ben, who simply lights up the day with witty comments, and thanks to Kassian, who ensured a never-ending supply chain of self-made apple rings during the final stage of writing.

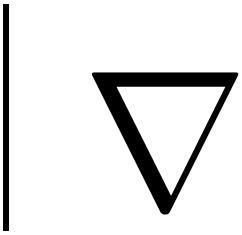
The same goes for my parents: Thank you for your unconditional support throughout my Bachelor's, especially towards the end!

Furthermore, I want to thank some friends in particular: Firstly Peter, for a bike-rack discussion about AI in our second semester which fostered my interest for the area. Secondly Nico, who gave general advice on thesis writing and the registration modalities. Thirdly Laurin, a.k.a. Clemens, who was always patient when I showed up late for lunch in the cafeteria because I "only had to start three more runs on the cluster".

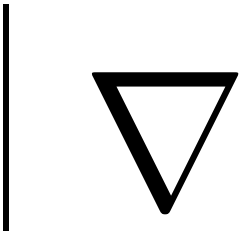
Thanks to all the diligent reviewers of my thesis drafts: Karla, Kassian, Laurin, Manu, Nico, Tobi, and Vogti. They invested a lot of their free time in spotting the subtlest mistakes, even if they appeared in footnotes. On that score, also kudos to the developers of *Grammarly*, an exquisite language checker for non-native speakers.

Finally, my thoughts are with Martin, Peter, and Vali, who will submit their theses soon. Hang in there, you are almost done!

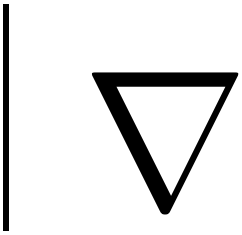
Contents



List of Figures



List of Tables



Chapter

1 Introduction

Mercury is the only planet in the inner solar system besides Earth with a large-scale magnetic field. The *magnetosphere* of a planet is defined as the surrounding area where its magnetic field constitutes the dominating force on charged particles. Mercury's magnetosphere differs from Earth's in two key aspects: Firstly, Mercury's magnetic field is only around 1 % as strong as that of Earth [Nes+74; And+10]. Secondly, since Mercury resides in close proximity to the Sun, its magnetosphere is highly influenced by solar weather conditions [KR95]. As a result, Mercury's magnetosphere is comparatively small and highly dynamic.

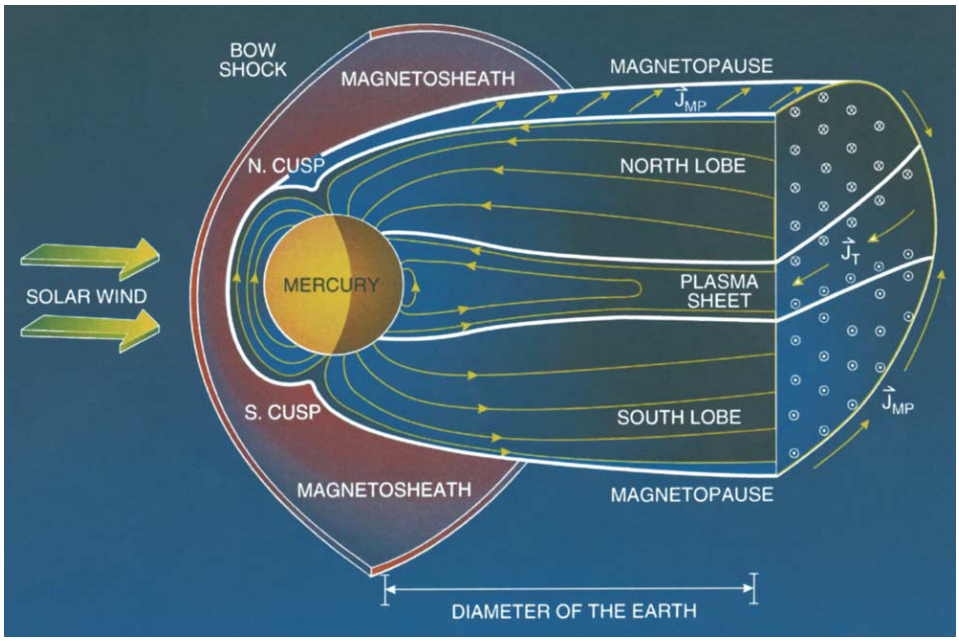


Figure 1.1: Schematic view of Mercury's magnetic conditions. Taken from [Sla04].

Figure 1.1 shows a sketch of the physical mechanisms that are at play: Starting from the upper atmosphere of the Sun, a plasma of charged particles called *solar wind* sets off toward the planets. The solar wind consists mainly of electrons and protons from ionized hydrogen as well as alpha particles, i.e., helium nuclei. Initially, the plasma flow is supersonic and therefore propagates faster than sound waves. As the solar wind approaches Mercury, it finally reaches a point where magnetic interaction with Mercury's magnetic field becomes significant. At this point, a shock wave arises that slows down

the solar wind to subsonic speeds and deflects it around the planet. This encounter is called the *bow shock* since the solar wind is displaced by Mercury's magnetic field like water by the bow of a ship. The region where the subsonic solar wind flows sideways past Mercury is called *magnetosheath*. It forms an intermediate space between the bow shock and the *magnetopause*, being the boundary of the magnetosphere. The described processes and notions apply to any inner planet of our solar system [KR95]. Still, since the solar wind weakens with the square of the distance from the Sun, they most prominently take effect around the innermost planet, Mercury.

It has long been of scientific interest in the planetary science community to study the bow shock and magnetopause signatures of Mercury. To this end, comprehensive empiric measurements were required. Thus, NASA launched the *Mariner 10* space probe in 1973 that encountered Mercury three times during the following two years. While these fly-bys yielded first clues regarding the magnetic dynamics, the insights were very limited. Hence, NASA decided to insert a spacecraft into orbit around Mercury for long-term empirical study [Pea00].

Dubbed MESSENGER as a tribute to the ancient Roman god lending the planet its name, the spacecraft launched on 3 August 2004. After several fly-bys, it entered orbit around Mercury on 11 March 2011. On 30 April 2015, the spacecraft crashed into Mercury's surface as it ran out of fuel [NAS19]. By then, MESSENGER had completed over 4000 orbits, yielding a vast amount of data from the onboard magnetometer [And+07]. During each orbit, the spacecraft usually passed from the interplanetary magnetic field through bow shock, magnetosheath, magnetopause and magnetosphere regions of Mercury and thereupon underwent the same sequence in reverse, as indicated in Figure 1.2. Hence, the spacecraft provides measurement series from more than 8000 incidences of bow shock and magnetopause crossings.

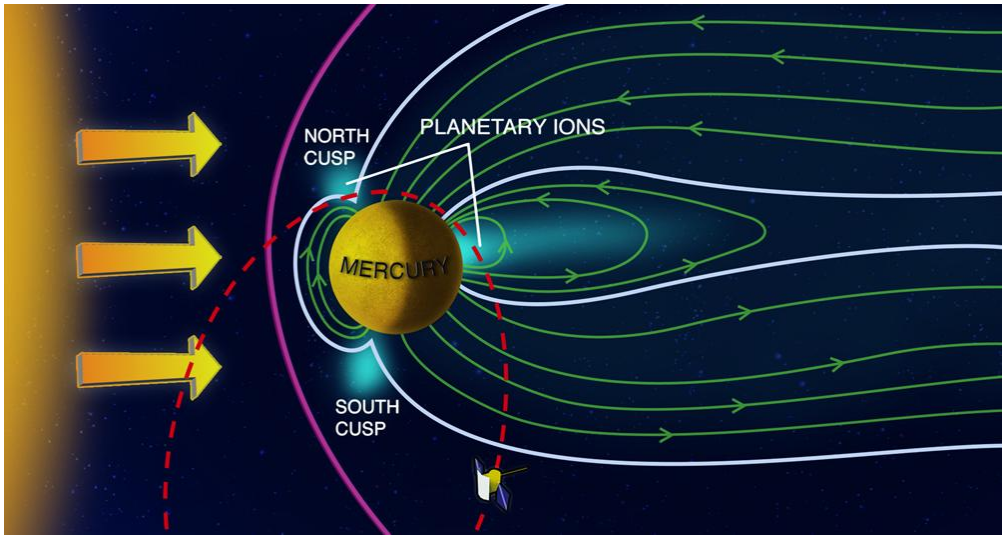


Figure 1.2: Schematic view of a typical MESSENGER orbit (red). Taken from [Zur+11].

Based on the abundance of data from the MESSENGER mission, several studies propose geometric models of Mercury's magnetosphere [Joh+12; Win+13; Zho+15; Phi+20a]. However, due to their global and static nature, they only provide an average shape of the bow shock and magnetopause crossings. The respective authors find the models struggle to capture the many fluctuations and details present in the time series data. For recognizing intricate patterns, *deep learning* has proven itself as a powerful modeling technique in recent years. Hence, Lavrukhin et al. [Lav+20; Lav+21a; Lav+21b] are currently working on modeling the crossings with a deep neural network, for which they provide a prepared dataset [Par+21].

We follow the direction of Lavrukhin et al. by devising a deep model for Mercury's magnetodynamics. It processes orbits *locally* on short time windows on the order of seconds or minutes. The model then predicts a magnetic region for each time step in a window. To give a complete picture for an entire orbit, the local predictions for all windows within an orbit can be integrated into a global one. The labels required for supervised training come from Philpott et al., who have analyzed all orbits for bow shock and magnetopause crossings [Phi+20a]. For an exemplary orbit, its magnetic flux data along with the crossing annotations are shown in Figure 1.3.

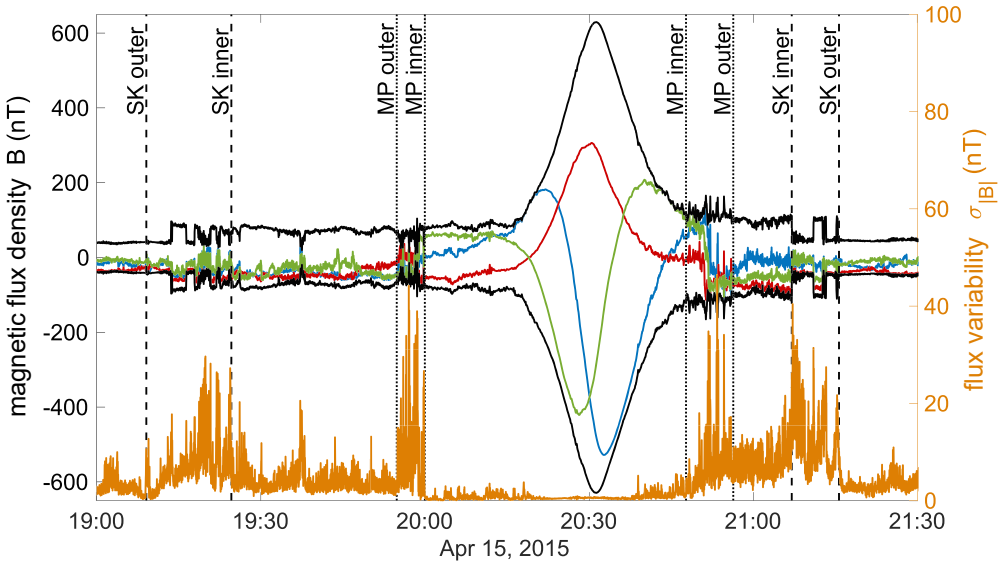


Figure 1.3: Magnetic flux density spatial components (RGB), ±magnitude (black) and variability (orange) for a typical MESSENGER orbit with bow shock (SK) and magnetopause (MP) annotations from Philpott et al. [Phi+20a].

Different orbits exhibit similarities in the magnetic field structure. We thus conjecture that not all orbits' data are required to obtain a representative model. It seems reasonable that at least the orbits from one complete Mercury year are necessary to ensure most conditions throughout the planet's journey about the Sun are present at least once. Since a Mercury year takes only 88 Earth days, the MESSENGER data comprise tens of Mercury years. As the magnetic conditions should be similar at the same location in

different Mercury years, it seems plausible that using only a few Mercury years' worth of orbits suffices to match the performance of a fully trained model. The notion of *active learning*, borrowed from educational science, allows for adding samples to the training process incrementally by their informativeness. In this vein, we examine how the model performance scales with available data on orbit-level.

Next to correctly identifying bow shock and magnetopause crossings, we also deem it desirable to predict if such a crossing is coming up in the near future when considering a given time window. We extend our deep model's classification task by a few future time steps for which the model receives no measurements. This may be of scientific interest when the next Mercury Mission, *BepiColombo* [Ben+10], initiates orbit entry in 2025: Comparing live model predictions with the new insights obtained from the mission in order to find out where the present conception of Mercury's magnetosphere matches reality and where it deviates significantly. After all, the data from BepiColombo will be of higher quality than the MESSENGER data and also cover regions of the magnetosphere that the latter did not reach. Our active learning approach might then become useful for model retraining with new data obtained from the mission.

Overall, our contributions may be summarized as follows:

1. We devise an end-to-end discriminative deep learning model capable of detecting current and predicting soon-to-be-expected bow shock and magnetopause crossings around Mercury from raw measurement data.
2. We investigate how the performance of a deep model scales with the number of orbits used for training. Specifically, we explore how many Mercury years' worth of data are required for a sufficiently representative model.
3. We provide a high-quality codebase that may be used as a framework for further studies on the topic. The source code is publicly available online.[1]

The remainder of this thesis is structured as follows: Chapter 2 discusses related work and techniques useful for our endeavors. Chapter 3 describes the MESSENGER magnetometer dataset, the crossing annotations, and our preprocessing pipeline. Based on the dataset, Chapter 4 explains in-depth how we formulate the deep learning model and the active learning algorithm. Employing this methodology, Chapter 5 presents the central experimental results and discusses their significance. Finally, Chapter 6 concludes with a summary of our findings and an outlook on future work. Appendix A extensively explains the mathematical background of our deep learning approach.

[1] `https://freddie.kirschstein.io`

Chapter

2 Related Work

Our work aligns with the recent research direction initiated by Lavrukhin et al. [Lav+20; Lav+21a; Lav+21b] who model Mercury's bow shock and magnetopause boundaries with deep learning. In particular, they employ a standard convolutional neural network for classifying time steps in the MESSENGER time series. However, they shared their progress so far only in the form of abstracts and short presentations, which denies us further insights into their methodology and results. Luckily, they provide an enhanced version of the original MESSENGER magnetometer measurements, which are grouped into orbits and augmented by additional data fields [Par+21]. We employ an earlier version of their prepared dataset for this work. To our knowledge, there do not exist any further deep learning approaches for modeling Mercury's magnetodynamics. We thus broaden the scope by referring to other modeling approaches based on the MESSENGER magnetometer data and research on deep learning for related time series tasks. In addition, we briefly review active learning literature.

Once the first MESSENGER measurements became available, several exact geometric models of Mercury's magnetic regions were proposed. Johnson et al. [Joh+12] use the measurements of MESSENGER's first three Mercury years in orbit to geometrically model the magnetic field inside the magnetosphere, assuming a paraboloid of revolution. Winslow et al. [Win+13] determine the average shape and location of the bow shock and magnetopause by fitting empirical models to the MESSENGER magnetometer data. They find that figures of revolution can best represent both boundaries, in particular, a hyperboloid and a figure similar to the Earth's magnetopause shape, respectively. Zhong et al. [Zho+15] use MESSENGER data from 24 March 2011 to 17 March 2014 to identify circa 5700 magnetopause crossings. They model the magnetopause as a three-dimensional non-axially symmetric shape. Finally, Philpott et al. [Phi+20a] extend the aforementioned studies by identifying bow shock and magnetopause crossings for the complete orbital mission and modeling the average boundary shape. They find that Mercury's average magnetopause is well modeled by both an axisymmetric shape and a three-dimensional shape with indentations. All these approaches share the drawback of considering static models that cannot capture changing conditions in the environment since they propose a fixed geometric shape cemented for all times. The only way in which we rely on them is by utilizing the crossing annotations provided by Philpott et al. [Phi+20b] for supervised deep learning.

The MESSENGER measurements are time series data, a domain where deep learning has been successful for some time now. This is evident from the many applications employing neural networks [Lai+18; LZ21; Ngu+18]. In time series and computer vision tasks, it is common to process not the entire input but apportion it with a sliding window algorithm [Die02; SL11; Sel+17]. We employ this approach as well to achieve a feasible model formulation. For our specific task of time series classification with neural networks, Ismail-Fawaz et al. [Ism+19] provide a recent survey. In particular, they conduct an empirical study of deep neural network architectures commonly used for time series classification. We leverage their findings for constructing our architectures, refined to the task at hand.

Regarding the notion of active learning, Burr Settles [Set09] conducted the first large-scale survey of existing literature. He summarizes various scenarios and creates a taxonomy of approaches that is now considered to be standard. Later, he extended this survey to a complete book [Set12] about active learning that includes more theoretical background and discussion of methods. We take advantage of Settles' work for pinning down an active learning strategy appropriate to our setting. Ren et al. [Ren+20] survey active learning in the more specific context of deep learning. There exist several active learning approaches explicitly designed for time series classification [He+15a; PLN17]. However, none of these fits the hierarchical two-layer structure of our setting, where the samples are on window-level, but active learning shall operate on orbit-level. While the notion of multiple-instance active learning [SCR07] addresses group-wise increment, it requires each sample in a group to have the same label. By and large, as no existing approach applies to our scenario, we must devise a custom framework.

Chapter

3 Dataset

Foundational to our study is the data provided by the MESSENGER mission. Hence, we first need to investigate the available data and prepare it for use in a machine learning model. Section 3.1 describes the overall structure of the MESSENGER magnetometer observations. Section 3.2 is concerned with the existing bow shock and magnetopause crossing annotations. Finally, section 3.3 explains all preprocessing steps that we employ to ensure a training-ready dataset.

3.1 MESSENGER Observations

The MESSENGER measurements used in this work stem from a previous version of the published dataset by Parunakian et al. [Par+21]. They comprise 4082 orbits of the spacecraft around Mercury in total. A single orbit contains time steps from one apoapsis[2] to the next on the resolution of one second. The number of time steps within an orbit thus hardly fluctuates. However, on 16 April 2012, the MESSENGER spacecraft accelerated to change from 12-hour orbits to 8-hour orbits. Hence the orbit lengths drop from around 42000 seconds to about 29000 seconds from that date onwards.

We summarize the available measurements for each time step in Table 3.1. Their vast majority can be grouped into triples of vector-valued numerical quantities. For some of them, their euclidian norm is already precalculated. The only categorical feature is a marker that indicates certain extrema of the spacecraft's position throughout the orbit. The vectorial quantities are highly redundant and mostly differ in the choice of coordinate system. For instance, the spacecraft position comes in four different formulations. However, the magnetic flux density measurement error is only given in the most widely used coordinate system: The *Mercury Solar Orbital* (MSO) coordinates specify the spacecraft's position in terms of its distance to Mercury's center. The x-axis points to the sun, the y-axis in the opposite direction of the spacecraft's orbital movement and the z-axis points northwards to complete the right-handed system.

For an exemplary orbit, Figures 3.1 and 3.2 plot the magnetic flux density and spacecraft position in MSO coordinates, respectively. One clearly sees the increase in flux density magnitude as the spacecraft approaches the planet towards the center of the plots.

[2] The apoapsis of an elliptic orbit is the point farthest away from the planet.

Column	Type	Description
DATE	datetime64	ISO 8601 formatted timestamp
X_MSO	float64	Spacecraft position x in MSO coord system [km]
Y_MSO	float64	Spacecraft position y in MSO coord system [km]
Z_MSO	float64	Spacecraft position z in MSO coord system [km]
BX_MSO	float64	Magnetic flux density x component in MSO coord system [nT]
BY_MSO	float64	Magnetic flux density y component in MSO coord system [nT]
BZ_MSO	float64	Magnetic flux density z component in MSO coord system [nT]
DBX_MSO	float64	Magnetic field measurement error x component in MSO coords [nT]
DBY_MSO	float64	Magnetic field measurement error y component in MSO coords [nT]
DBZ_MSO	float64	Magnetic field measurement error z component in MSO coords [nT]
RHO_DIPOLE	float64	Dipole-centric spacecraft distance [km]
PHI_DIPOLE	float64	Spacecraft magnetic azimuth [rad]
THETA_DIPOLE	float64	Spacecraft magnetic latitude [rad]
BX_DIPOLE	float64	Planetary model dipole magnetic field x component in MSO [nT]
BY_DIPOLE	float64	Planetary model dipole magnetic field y component in MSO [nT]
BZ_DIPOLE	float64	Planetary model dipole magnetic field z component in MSO [nT]
BABS_DIPOLE	float64	Planetary model dipole total magnetic field magnitude [nT]
RHO	float64	Planetocentric spacecraft distance [km]
RXY	float64	Spacecraft distance to the MSO z axis [km]
X	float64	Mercury heliocentric position x in SE coord system [km]
Y	float64	Mercury heliocentric position y in SE coord system [km]
Z	float64	Mercury heliocentric position z in SE coord system [km]
D	float64	Mercury heliocentric distance [km]
VX	float64	Mercury orbital velocity x component in SE coord system [km/sec]
VY	float64	Mercury orbital velocity y component in SE coord system [km/sec]
VZ	float64	Mercury orbital velocity z component in SE coord system [km/sec]
VABS	float64	Mercury orbital velocity magnitude [km/sec]
COSALPHA	float64	Cosine of Mercury azimuth angle in SE coord system
EXTREMA	int64	Marker indicating extreme positions: $\begin{cases} 2 \text{ apoapsis} \\ -2 \text{ periapsis} \\ 1 \text{ max planet distance} \\ -1 \text{ min planet distance} \\ 0 \text{ otherwise} \end{cases}$

Table 3.1: Overview of all features in the MESSENGER magnetometer dataset.

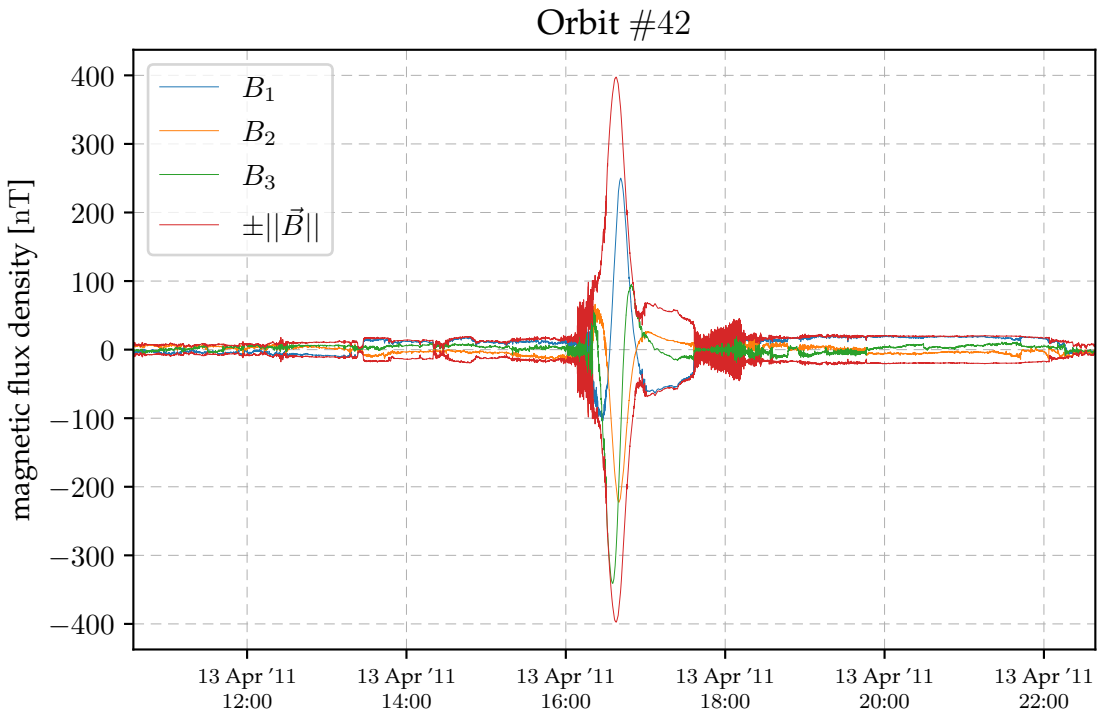


Figure 3.1: Magnetic flux density MSO components and norm for an exemplary orbit.

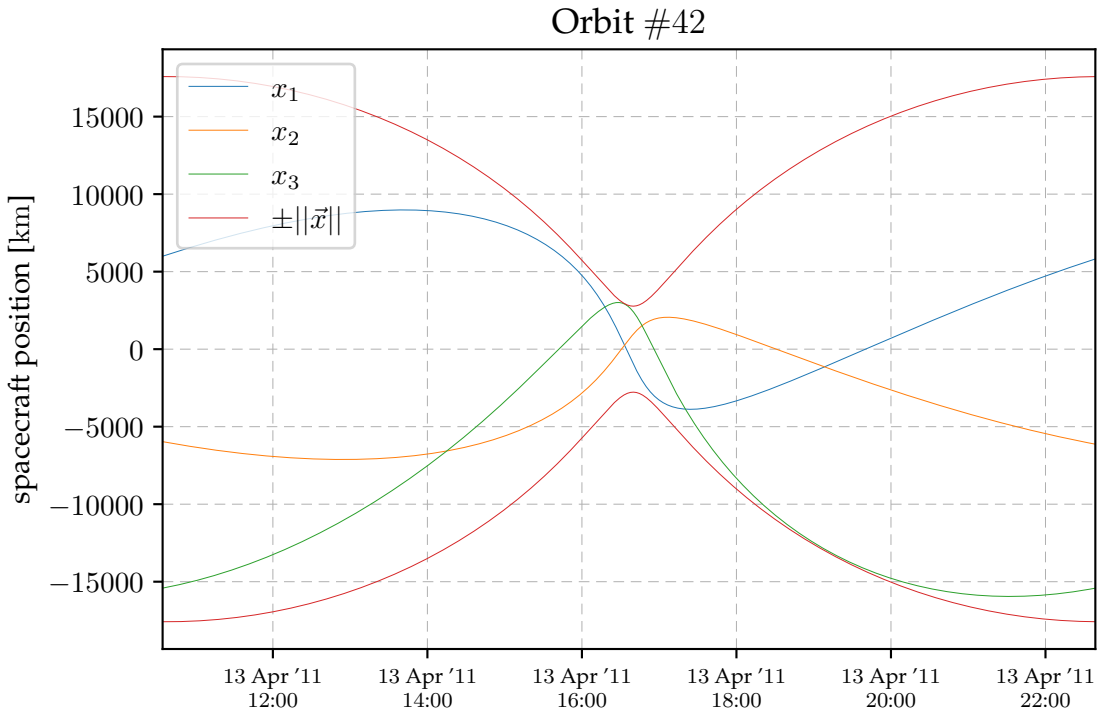


Figure 3.2: Spacecraft MSO position components and norm for an exemplary orbit.

3.2 Crossing Labels

As mentioned in Chapter 1, we take advantage of the work from Philpott et al. [Phi+20b] who localized bow shock and magnetopause crossings for the entire MESSENGER mission. In particular, they determined the times of spacecraft entry and exit for each crossing. With 4019 out of 4082, almost all orbits receive a complete set of those crossing annotations. The remainder is only partially annotated due to special magnetic conditions where the crossings did not happen in the usual succession.

For details on how the annotations were obtained, we refer to their publication and only stress two particular points: Because the bow shock and magnetopause crossings moved rapidly relative to the spacecraft, they were typically crossed multiple times in a row. Philpott et al. decided not to label these successive crossings individually but to *combine* them into a single larger crossing. Furthermore, the boundaries could not be clearly identified in some cases due to low magnetic flux variability. In these cases, the boundaries were chosen *conservatively*, potentially overestimating their extent.

Figure 3.3 shows our exemplary orbit #42 from the previous section supplemented with its bow shock (SK) and magnetopause (MP) crossings as well as the resulting regions of interplanetary magnetic field (IMF), magnetosphere (MSp) and the gray area of magnetosheath (MSh). We will use these abbreviations and the plot shading colors consistently throughout this work.

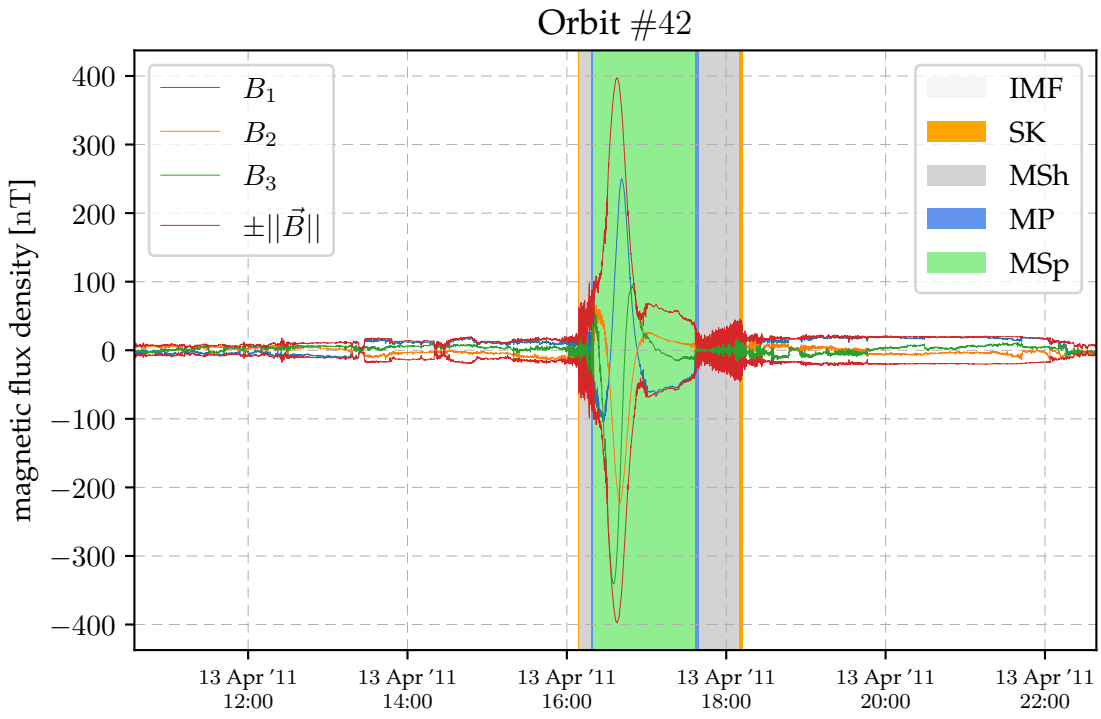


Figure 3.3: Magnetic regions for an exemplary orbit.

In order to build a *local* deep learning model, we leverage the global crossing annotations to label each time step on every orbit with an ordinal from 0 to 4 indicating the associated magnetic region, which will yield a five-class classification problem. Table 3.2 displays the resulting distribution of labels over the entire dataset. It is intrinsically *unbalanced*, since the bow shock and magnetopause crossings usually last for only several seconds or minutes while the spacecraft spends hours far away from Mercury in the interplanetary magnetic field. This imbalance is also evident from Figure 3.3.

label	magnetic region	share
0	interplanetary magnetic field (IMF)	$65.4\,\%$
1	bow shock crossing (SK)	$3.7\,\%$
2	magnetosheath (MSh)	$14.5\,\%$
3	magnetopause crossing (MP)	$2.3\,\%$
4	magnetosphere (MSp)	$14.1\,\%$

Table 3.2: Labels for a single time step and their frequency.

3.3 Preprocessing

For a machine learning model to be able to learn from the labeled MESSENGER observations, we employ several preprocessing steps. In particular, they comprise cleaning the data, putting aside holdout sets for validation, and statistical normalization.

3.3.1 Faulty Orbit Removal

Not all labeled orbits are immediately usable for our purposes. A significant number of them exhibit unwanted properties, wherefore we call such orbits *faulty*. We rigorously remove all those orbits from the productive dataset. In total, we identify four properties qualifying an orbit as faulty:

NaN Values. Some orbits do not provide measurements for all time steps or all features. Especially in the beginning and the end of the MESSENGER mission, but also during occasional phases in between, a lack of measurements occurs. This filtering step affects the most orbits, with a total of 462.

Missing Time Steps. Some orbits contain fewer time steps than the difference of the last and first time step in seconds would imply. Hence, steps are missing in between. For instance, orbit $\#3565$ consists of only three and $\#3686$ of only 1641 seconds, whereas the usual length is around thirty and forty thousand seconds. This is unpleasant for two reasons: Firstly, such orbits are in no way comparable to the other, complete ones. Secondly, our machine learning model requires a contiguous sequence of measurements without jumps to learn from. This filtering step affects a total of 290 orbits.

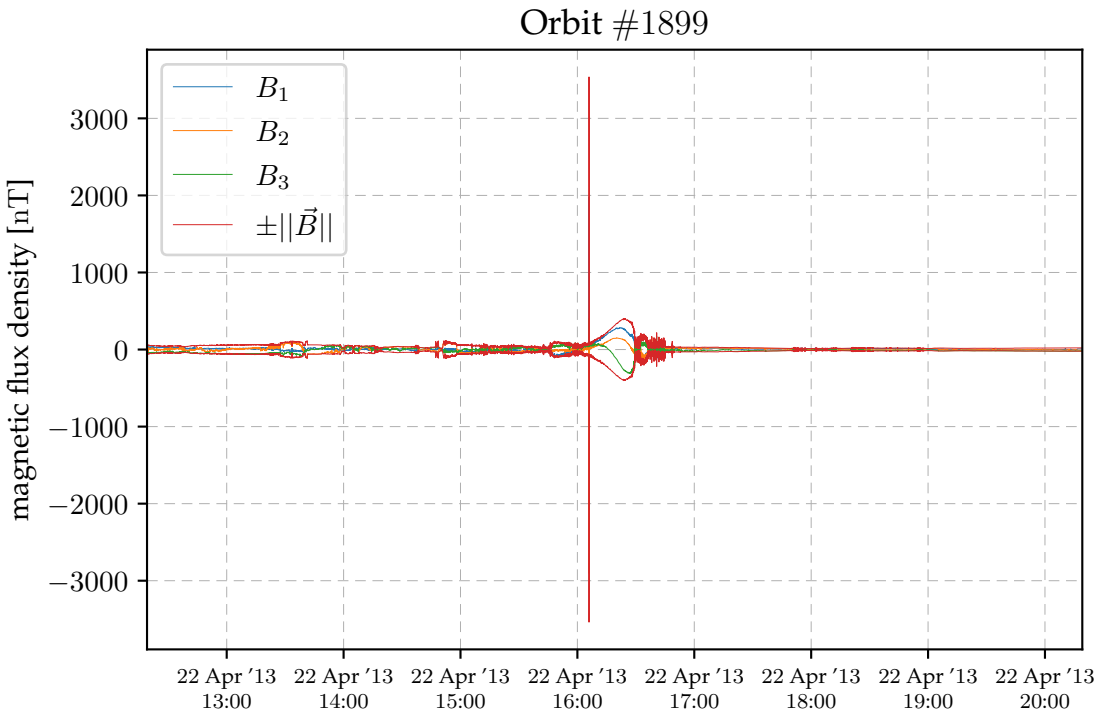


Figure 3.4: An orbit with significant outliers due to measurement errors.

Flux Outliers. Some orbits contain time steps for which the measured flux density magnitude is extreme. Figure 3.4 exemplifies this problem for an orbit where the norm of flux density exceeds the usual magnitude by several orders. As such outliers can utterly impair the results and performance of machine learning models, we apply the *three-sigma rule* to detect orbits with outliers as follows: For each orbit $o \in \mathbb{N}$, we determine the maximum occurring flux density magnitude $B_{\text{max}}^{(o)}$. Now consider the continuous random variable B_{max} of maximum flux norms. We regard o as outlier orbit whenever its maximum flux exceeds the empirical mean by three empirical standard deviations, i.e., $B_{\text{max}}^{(o)} \geq \mu + 3\sigma$ where $\mu \approx \mathbb{E}[\mathrm{B}_{\text{max}}]$ and $\sigma \approx \sqrt{\mathrm{Var}[\mathrm{B}_{\text{max}}]}$. A total of 10 orbits turn out as outliers by this method.[3]

Overhanging Crossings. Some orbits exhibit a crossing anomaly where their crossings extend into neighboring orbits or vice versa. Often in these cases, the crossings also cover an extreme time range, presumably resulting from Philpott et al.'s conservative annotation approach. We give an example of this "neighbor intrusion" in Figure 3.5. It affects a total of 437 orbits.

Although we could recover some faulty orbits by interpolation methods, we decide to rigorously remove all of them. After all filtering steps, 3151 orbits still remain out of the 4019 labeled ones. Note that some orbits exhibit multiple faults.

[3] An expectable order of magnitude: If $\mathrm{B}_{\text{max}} \sim \mathcal{N}(\mu, \sigma^2)$ were normally distributed, the probability of being below the three-sigma threshold would be $\mathbb{P}[\mathrm{B}_{\text{max}} \leq \mu + 3\sigma] = \mathbb{P}[\frac{\mathrm{B}_{\text{max}} - \mu}{\sigma} \leq 3] = \Phi(3) \approx 99.87\,\%$. Applied to 4019 overall labeled orbits, this would give a bit over 5 outliers.

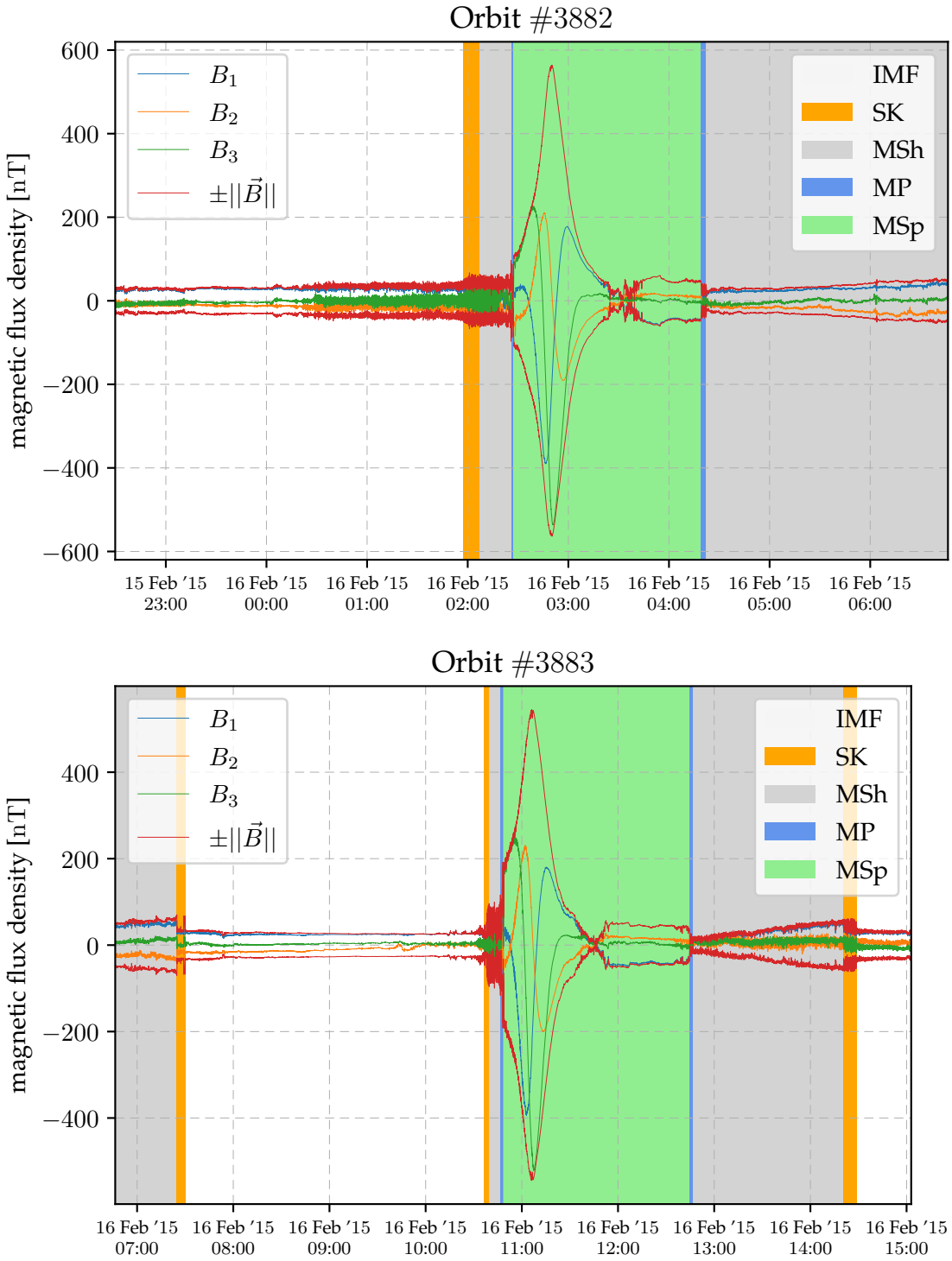


Figure 3.5: Example of an overhanging bow shock crossing. The outer bow shock crossing of the orbit at the top extends beyond the apoapsis into the orbit at the bottom. This is presumably due to special solar wind conditions.

3.3.2 Dataset Partitioning

To assess the performance of our machine learning model, we need to hold out some part of the data that will never be used for learning. As is the undisputed standard in machine learning, we employ a random split into three disjoint partitions: A *training*, *evaluation* and *test* set, respectively. Since, unfortunately, machine learning literature disagrees upon the meaning of the terms evaluation and test set, we define explicitly how we use these terms in this work:

- **Training set (70 %, 2207 orbits):** The subset used for actually training the machine learning model.
- **Evaluation set (20 %, 630 orbits):** The subset used for validating the performance of models during the internal development process.
- **Test set (10 %, 316 orbits):** The subset used for validating the performance of a final model at the end of development.

Throughout this work, we use these terms exactly in the sense given above.

3.3.3 Feature Normalization

Due to their astronomical nature, the features in the MESSENGER observations vastly differ in their value range. Having such differently-scaled input features poses a major problem for deep model training: Since almost all transformations in a neural network model are linear combinations of weights and the input, the gradient with respect to the parameters is larger in dimensions of the feature space with larger scales. Hence, the steps do not proceed straight towards a local minimum but exhibit a zig-zag behavior. This slows down the overall convergence of gradient descent [Cur44; LeC+98].

Furthermore, differently-scaled features cause a neural network to put different importance on them. Although it could *in theory* learn to undo the scale difference, this does not happen in practice [Hua+20]. We do not want to introduce a prior belief about the relative importance of the features since it contradicts the end-to-end notion. Leaving the features unnormalized would effectively be implicit feature engineering.

To get rid of scale differences, we employ *z-normalization*: Let x be a continuous random variable describing one of the input features with empiric mean $\mu \approx \mathbb{E}[\mathrm{x}]$ and empiric standard deviation $\sigma \approx \mathrm{Var}[\mathrm{x}]$. Then we feed the model not the original feature x, but its standardization $\frac{\mathrm{x}-\mu}{\sigma}$ with approximately zero mean and unit variance. Normalization techniques like this have been shown to have huge effects on the learning capability of deep models [Zha+21].

Chapter

4 Methodology

Having investigated and prepared the MESSENGER observations, we now describe our approach in detail. Section 4.1 establishes the central machine learning task formulation that underlies this work. Next, Section 4.2 specifies the employed neural network architectures which we compare empirically later. Finally, Section 4.3 explains how we incorporate active learning into the system.

4.1 The Freddie Task

To build a deep learning model, we first need to consider what exact task it is supposed to solve. In particular, we need to specify the expected input and output of the neural network. Naively, we would simply feed an entire orbit's time series as input to the model. However, there are several drawbacks to this approach:

1. As pointed out in Section 3.1, orbits vary greatly in length. Alas, not all neural network architectures support variably sized inputs. In principle, we could cope with different lengths by cropping each time series to the minimum occurring length. However, in this vein, we would discard a fourth of the information in the earlier orbits where MESSENGER had a lower velocity.

2. The absolute length of orbits with tens of thousands of time steps is relatively high for machine learning standards. Therefore, even if we only employed architectures that allow for variable input sizes, the processing and memory burden they brought along would be infeasibly high.

3. Processing an entire orbit at once would counteract our goal of a *local* model. As outlined in Chapter 1, we focus on exploiting local structures to help better understand the bow shock and magnetopause boundaries. Feeding the model with an entire orbit might provide it with too much information to be useful, since we cannot figure out from where in the orbit the model draws its conclusions.

For the reasons given above, we present only short, contiguous subsequences of an input series to the model. This is done with the *sliding window algorithm* [Die02] outlined in Algorithm 4.1: A window covering $w \in \mathbb{N}$ time steps slides across the time series $\boldsymbol{T} \in \mathbb{R}^{d \times n}$ in chronological order with stride $s \in \mathbb{N}$. Each position of the window yields a slice of the time series that constitutes an input for the model.

```
/* extracts sliding windows from the given time series        */
sliding_window(T ∈ ℝ^{d×n} : time series, w ∈ ℕ : window size, s ∈ ℕ : stride):
    W := ∅  // contains extracted windows
    i := 0  // index of first step in window
    while i + w ≤ n :
        W := W ∪ {T_{:,i:(i+w−1)}}  // add next window
        i := i + s
    return W
```

Algorithm 4.1: Conceptual overview of the window sliding strategy.

Besides solving the discussed issues, using overlapping slices of the time series brings forth another potential: With a sufficiently small stride, each time step of the original series is contained in multiple windows and thus reused, but in different arrangements. This essentially constitutes a form of data augmentation which dramatically increases the number of training samples the model can learn from. To get the most out of the MESSENGER time series, we choose a stride of $s = 1$. Furthermore, this choice ensures the model gets to see all bow shock and magnetopause crossings in every possible position within a time window. If we chose $s = w$ instead and were unlucky, the crossings might happen to always appear at the same position within a window, such that the model would not be incentivized to develop translation-equivariance.

To summarize, the model's input is a window of $w \in \mathbb{N}$ successive time steps. Each of these time steps has dimensionality $d \in \mathbb{N}$, which is the number of used scalar features. Each of the features has been normalized according to Section 3.3.3. We will leave the exact choice of features unspecified for now, since their selection is carried out empirically in Section 5.1. Instead, we abstractly consider a formal window:

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}^{(1)} & \boldsymbol{x}^{(2)} & \cdots & \boldsymbol{x}^{(w)} \end{bmatrix} \in \mathbb{R}^{d \times w}.$$

Let us now turn to the output specification. We desire a discriminative model, classifying each time step with the magnetic region in which the spacecraft was at that moment. Recall from Section 3.2 that there are exactly five mutually exclusive regions. However, directly predicting the identifier number of a class would be a flawed specification since it would assume a linear ordering of the classes which does not exist.[4] Therefore, we provide the expected output to the model per time step as *one-hot* vectors, or simply standard basis vectors. These take the form $[0, \ldots, 0, 1, 0, \ldots, 0]^{\mathrm{T}}$ with the position of 1 indicating the class. Since we expect the model to predict a class per time step, we pack multiple of these one-hot vectors next to each other into a matrix.

[4] In fact, they have a cyclic ordering, the future exploitation of which we discuss in Chapter 6.

We may think of the last time step $\boldsymbol{x}^{(w)}$ in a window as representing the "present". Instead of merely classifying the "past" time steps within a window, we deem it interesting to also let the model predict the magnetic region for $f \in \mathbb{N}$ future time steps. Thus, the expected output matrix of one-hot vectors is $\boldsymbol{Y} \in \mathbb{R}^{5\times(w+f)}$. This essentially imposes two tasks on the neural network: Classifying available time steps and classifying future time steps for which measurements are not provided. We conjecture that this *multi-task* approach might benefit generalization by forcing the model to learn shared representations that foster both tasks. In this work, we fix the window size to be 2 minutes, i.e., $w = 120$ seconds, and the future size to be $f = 20$ seconds.

Formally, the task could be termed multi-dimensional multi-class classification with a future component. We dub it the Freddie task: Given the window $\boldsymbol{X}$, predict a sequence of magnetic region probabilities, where each column sums up to one:

$$\hat{\boldsymbol{Y}} \coloneqq \left[\begin{array}{ccc|ccc} p_{1,1} & \cdots & p_{1,w} & p_{1,w+1} & \cdots & p_{1,w+f} \\ \vdots & & \vdots & \vdots & & \vdots \\ p_{5,1} & \cdots & p_{5,w} & p_{5,w+1} & \cdots & p_{5,w+f} \end{array}\right] \in [0,1]^{5\times(w+f)}$$

To measure the error between $\hat{\boldsymbol{Y}}$ and the ground-truth label $\boldsymbol{Y}$, we employ the cross-entropy loss as derived in Section A.1. Ours is the special case of *categorical cross-entropy*, since the ground-truth one-hot output constitutes a Dirac distribution. However, we do not use vanilla cross-entropy, but a *weighted* version of it. The reason is the considerable class imbalance discussed in Section 3.2. If we were to weight each class the same, the minority classes of bow shock and magnetopause crossings would be immensely discriminated by the discriminator, or worse, even completely ignored. To counteract this, we give each time step's loss a weight inversely proportional to its class' frequency $f_c \in \mathbb{N}$ in the dataset:

$$w_c \coloneqq \frac{\sum_{i=1}^{5} f_i}{f_c}$$

The resulting weighted loss for a single time step j in a window is then:

$$\mathcal{L}_j(\hat{\boldsymbol{Y}}, \boldsymbol{Y}) \coloneqq -\sum_{i=1}^{5} w_i \boldsymbol{Y}_{ij} \log(\hat{\boldsymbol{Y}}_{ij})$$

Note that the sum is actually a fake sum, since only exactly one of the $\boldsymbol{Y}_{ij}$ is non-zero. By averaging across all time steps, we straightforwardly obtain the window loss:

$$\mathcal{L}(\hat{\boldsymbol{Y}}, \boldsymbol{Y}) \coloneqq \frac{1}{w+f} \sum_{j=1}^{w+f} \mathcal{L}_j(\hat{\boldsymbol{Y}}, \boldsymbol{Y}) = -\frac{1}{w+f} \sum_{i=1}^{5} w_i \sum_{j=1}^{w+f} \boldsymbol{Y}_{ij} \log(\hat{\boldsymbol{Y}}_{ij})$$

The expected loss over all windows extracted from the training set then forms the total loss we seek to optimize.

4.2 Model Architectures

With the task definition consolidated, we seek a suitable neural network model for it. We consider a total of six architecture categories, of which we experimentally determine the best instances. For clearness, we present only the final architectures for each category. All models use the rectified linear unit (ReLU) as activation due to its advantages described in Section A.3.2. Their input has the window size $w = 120$ and the future size $f = 20$ that we fixed in Section 4.1. Furthermore, the channel dimensionality $d = 9$ is a result of the empirical feature selection undertaken in Section 5.1. All six model architectures are outlined in Figures 4.1 and 4.2.

MLP For a solid baseline, we employ the simplest possible architecture: A *multi-layer perceptron* (*MLP*) as described in Section A.3.1. It consists of two dense hidden layers with 128 neurons each and solely operates on the flattened window. Therefore, the MLP is completely agnostic to the structure of the input, which consists of a time dimension and a channel dimension.

CNN To make use of the time series structure, we proceed with a classical *convolutional neural network* (*CNN*). We explain the involved components comprehensively in Section A.3.3. First, the input passes through three successive one-dimensional convolutional layers with subsequent max-pooling. The 1D convolutions all employ "same" padding and can also be thought of as 2D convolutions with full channel width. As often in deep time series classification [Ism+19] we find that increasing the convolution kernel sizes with deeper layers works well; in our case, the succession is 3-5-7. The convolutional block's result is then flattened and processed by a single dense layer to yield the required output shape. The dense layer is the reason for employing max-pooling, since the latter reduces the parameter number in this final layer, which however is still rather high.

FCNN A downside to the vanilla CNN is its final dense layer having a lot of trainable parameters. This happens because ours is a sequence-to-sequence task, while CNNs were originally developed for summary tasks where the output size is much smaller than the input size. We circumvent this issue with a slightly modified version that discards the dense layer. For this reason, the result is often called a *fully-connected convolutional neural network* (*FCNN*) [Ism+19]. The dense layer's substitute is two-fold: First, a singleton convolution converts the channel size to the required flattened output size. Due to the kernel size of one, this is equivalent to a point-wise dense layer. Second, a global average pooling layer reduces each resulting channel across the time dimension to a single value. The FCNN has no choice but to learn to use different channels for different features, since different time steps within the same channel will eventually be combined. The FCNN learns entirely through convolutional layers, which also speeds up training.

RNN Both the CNN and FCNN exploit local patterns in the time series. Their convolutional layers may thus be thought of as feature extractors. However, they do not make use of temporal relationships that might follow a systematic rule. This is what *recurrent neural networks* (*RNN*) as described in Section A.3.4 achieve. Hence, we add an RNN to the competition, consisting of three stacked LSTMs. Since our task definition allows the model to utilize the entire input window without restrictions, all LSTMs are bi-directional. As RNNs are designed for sequence modeling, we require only a few additional components. Before the LSTMs, we add zero padding that extends the input series by the desired amount of additional future steps to predict. In the end, a point-wise dense layer reduces the dimensionality from the internal LSTM state size to the number of classes.

CRNN To get the best of both worlds, we combine the feature extraction capabilities of convolutional components with the temporal aspect addressed by recurrent components into a single architecture. The resulting crossover is often called a *convolutional recurrent neural network* (*CRNN*) and was successfully employed in several time series tasks [ZPT17; Kao+18; ZD20]. We thus expect an improvement over a plain CNN or RNN on the Freddie task as well. In this architecture, a convolutional block is followed by a recurrent stack. Therefore, we dispense with the pooling layers of the CNN and FCNN. Firstly, there is no memory-related need for reducing the sequence length since RNNs are agnostic to it, applying the same transformation to as many time steps as there are. Secondly, without pooling, the convolutional block "only" extracts features while preserving the original sequence length, making the recurrent operation more meaningful. Besides, if we employed pooling, we would have to upsample after the recurrent stack regardless, so we avoid a sequence length reduction in the first place.

CANN As discussed in Section A.3.5, recurrent networks are firstly slow to train and, secondly, often fail to capture long-term dependencies, even with LSTMs. Hence, we try an experimental architecture, replacing the second, recurrent part of the CRNN with attention mechanisms. Deliberately alluding to CRNNs, we dub this model a *convolutional attentional neural network* (*CANN*). Loosely following the structure of a Transformer encoder [Vas+17], it employs several multi-head self-attention layers separated by point-wise dense layers.

Admittedly, our architecture search space was biased towards small models with fewer parameters due to resource and time limitations. We expect them to trivially increase in performance by simply scaling them up and adding regularization. However, we are only interested in the *relative* performance of the architectures, wherefore smaller ones suffice. We evaluate all of the above architectures in Section 5.2 to find the "best" one. The winner then forms the basis of all further investigations.

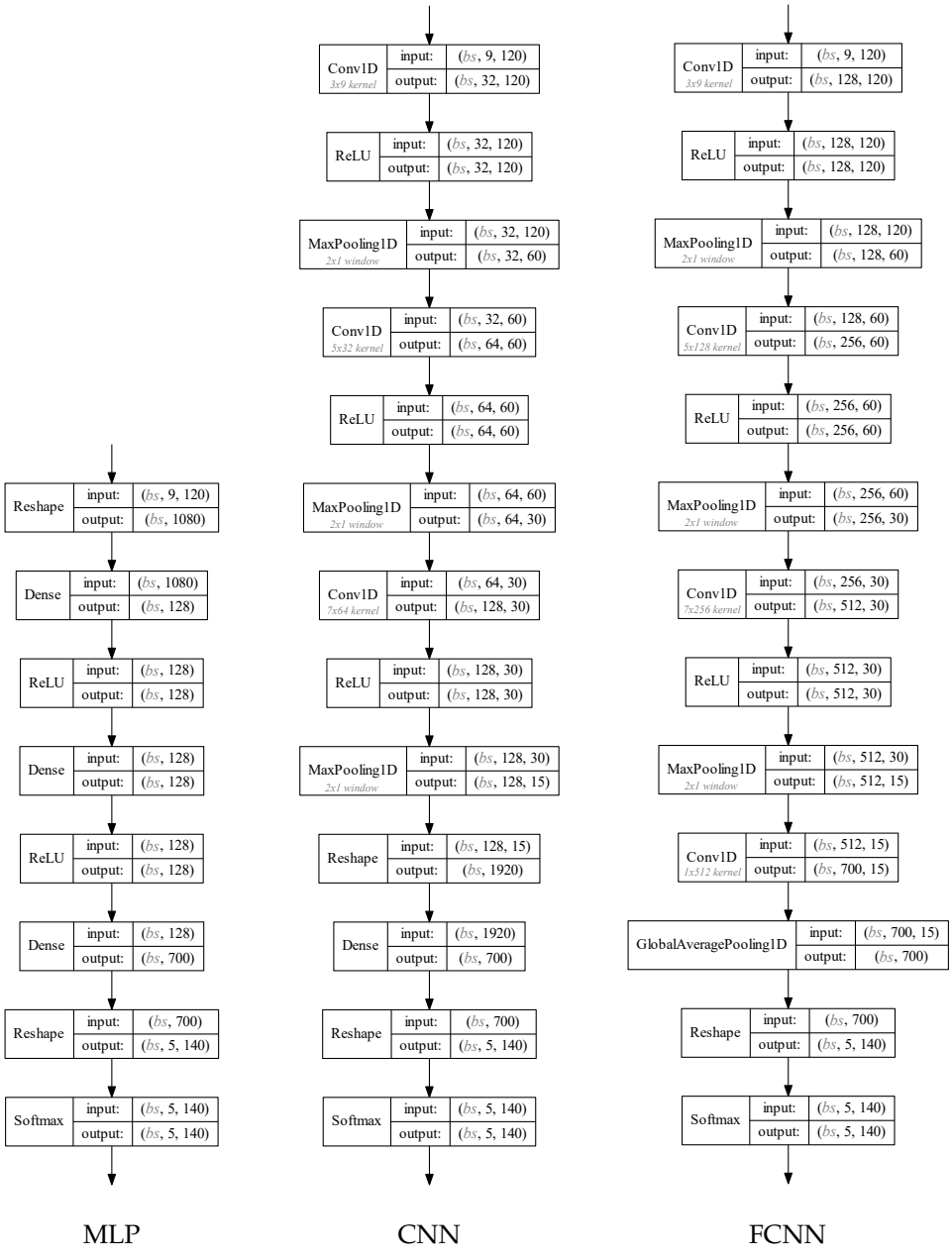


Figure 4.1: Model Architecture Diagrams, Part 1. The concrete input and output sizes result from the choice of window size $w = 120$, future size $f = 20$ and number of features $d = 9$. The placeholder *bs* stands for the batch size.

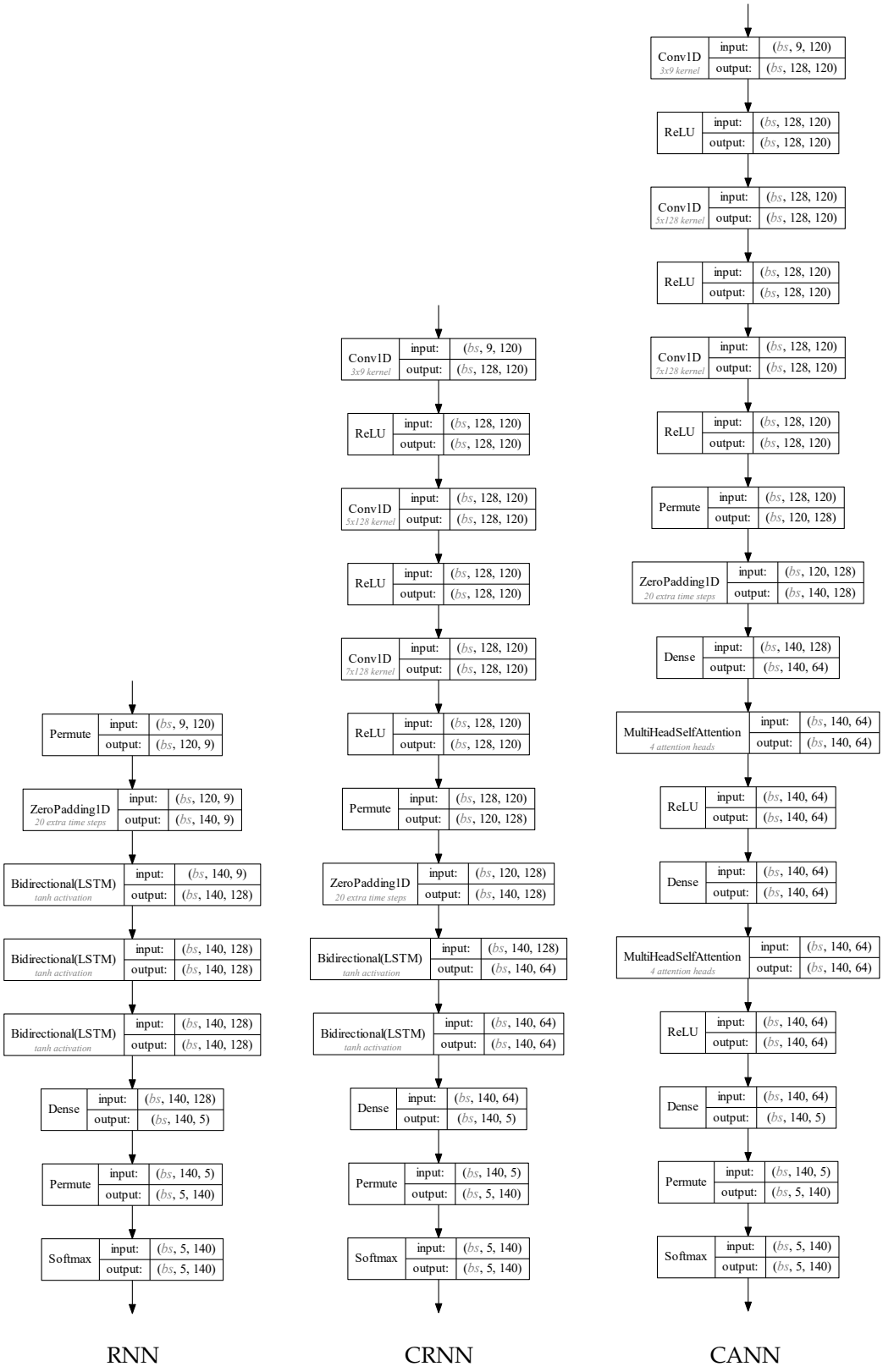


Figure 4.2: Model Architecture Diagrams, Part 2. The concrete input and output sizes result from the choice of window size $w = 120$, future size $f = 20$ and number of features $d = 9$. The placeholder *bs* stands for the batch size.

4.3 Active Learning

Once a final model architecture is established via experimentation, we can turn to our ultimate goal: Determining the minimum number of orbits required for a representative model of Mercury's bow shock and magnetopause boundaries. As mentioned in Chapter 1, we approach this question using the notion of *active learning*. Originally, this is an instructional technique from education literature, where, instead of passively digesting presented information, students actively participate in the learning process.

By viewing the neural network model as the learner, the active learning concept can be transferred to a machine learning setting. Out of the three major active learning scenarios considered in the literature [Set09; Set12], ours is an instance of *pool-based active learning* as illustrated in Figure 4.3. Initially untrained, the model repeatedly selects samples from a pool of yet unlabeled samples. These receive corresponding labels by an oracle, which could be a human annotator or, as in our case, simply the act of providing the held-back labels to the model. The newly obtained samples are finally added to the training set, on which the model is retrained. This retraining comes in two flavors: Either the model is trained from scratch on the enlarged training set, or the training continues from the previously found parameters.

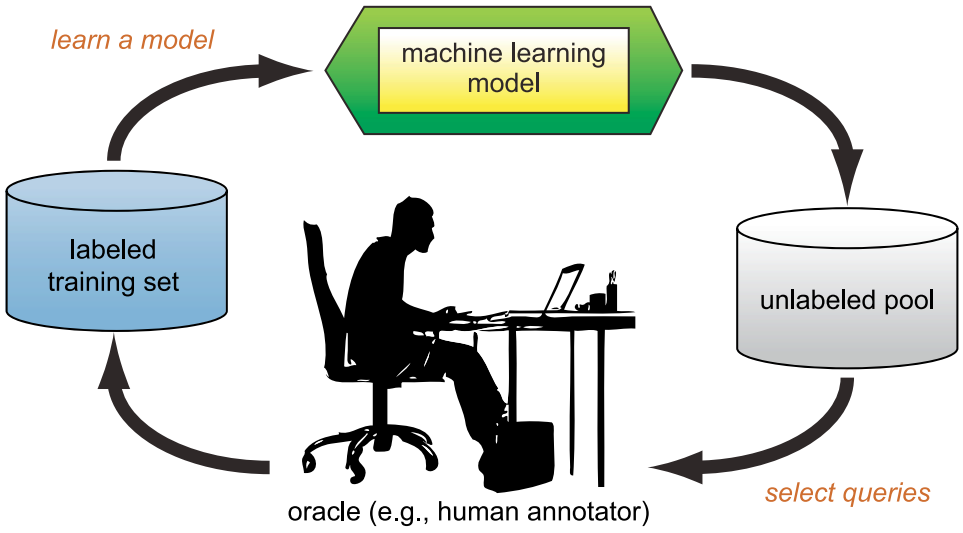


Figure 4.3: Illustration of pool-based active learning. Adapted from [Set09].

The active learning cycle was initially introduced into machine learning to reduce the required amount of labeled samples. This becomes crucial when sample annotation is costly, like for medical images. It has been shown that an effective active learning algorithm can, in theory, accelerate labeling efficiency in an exponential manner [BBL09]. We may thus argue that, through active learning, the model can get the most out of the available data, which leads to a nearly optimal learning curve.

To address our performance scaling question, we increment the training set not by individual windows but on the level of entire orbits. In order to choose the next orbit(s) to add, we need to rank all yet unused orbits according to an *informativeness* measure. To this end, there exist several strategies [Set09], with increasing computational cost:

- *Uncertainty Sampling:* Select the samples for which the model prediction is least certain according to an adequate uncertainty measure for the output distribution.
- *Query by Committee:* Train a committee of multiple models throughout the active learning procedure, representing competing hypotheses, and select the samples about which the committee disagrees most.
- *Expected Model Change:* Select the samples which would lead to the gradients of highest magnitude if they were added to the training set. If their labels are not known, the new gradient has to be estimated by an expectation over the output probability distribution.
- *Estimated Error Reduction:* This approach is similar to the expected model change strategy. Instead of maximizing the (estimated) future gradient, it minimizes the (estimated) future loss when adding samples to the training set.

All of these sampling strategies can further be extended by *density-weighting*. For each unlabeled sample, this method weights the base strategy's informativeness score by the average similarity of that sample to all samples in the dataset according to some similarity measure. Density weighting ensures that selected samples are not only highly informative, but also representative of the input distribution.

Due to limited computing resources, we decide for an uncertainty sampling strategy. Nowadays in deep active learning, solely relying on the top uncertain samples is discouraged because they are most likely similar and lead to overfitting [Ren+20]. However, since we always add entire orbits' worth of samples, this is no issue in our study! We automatically get diversity and representativeness from the orbit-leveled processing. For this reason, we also do not require density-weighting.

Since the Freddie task has a series of Multinoulli distributions as output, we may measure uncertainty as a function of the output probabilities. As discussed in Section A.1.1, *Shannon entropy* is a mathematically well-funded measure of uncertainty in a probability distribution. Hence, we choose entropy as the basis of our active learning endeavor: Consider the training set $\mathcal{D} \subseteq \mathbb{R}^{d\times w} \times \{\text{IMF}, \text{SK}, \text{MSh}, \text{MP}, \text{MSp}\}^{w+f}$ with number of features $d \in \mathbb{N}$, window size $w \in \mathbb{N}$ and future size $f \in \mathbb{N}$. Given a model prediction $\hat{\boldsymbol{Y}} = [\hat{\boldsymbol{y}}^{(1)}, \dots, \hat{\boldsymbol{y}}^{(w+f)}] \in [0,1]^{5\times(w+f)}$, we define its uncertainty as

$$\mathfrak{u}(\hat{\boldsymbol{Y}}) \coloneqq \max_j H(\hat{\boldsymbol{y}}^{(j)}) = -\min_j \sum_{i=1}^{5} y_i^{(j)} \log(y_i^{(j)}),$$

where $H : \triangle^4 \to \mathbb{R}$ is the Shannon entropy formulation on the standard 4-simplex.

Since we intend to conduct active learning on the orbit level, we need to measure the uncertainty on an entire orbit. To this end, we must reduce the individual window uncertainties to a single orbit score. We are particularly interested in the bow shock and magnetopause crossings, and the evaluation in Section A.4 shows that a deep model confidently detects the other regions. Thus, we argue that the most uncertain windows of an orbit will usually overlap with a crossing region. For this reason and to lessen computational cost, we only consider the uncertainty of such windows for the overall orbit uncertainty. Let $\mathcal{D}_o \subseteq \mathcal{D}$ be the windows belonging to the orbit $o \in \mathbb{N}$ and

$$\widetilde{\mathcal{D}}_o := \{(\boldsymbol{X}, \boldsymbol{y}) \in \mathcal{D}_o \mid \boldsymbol{y} \cap \{\text{SK}, \text{MP}\} \neq \emptyset\}$$

be only those samples that overlap with a bow shock or magnetopause boundary region. The average uncertainty over these windows defines the integrated orbit uncertainty of a model $\hat{f}_{\boldsymbol{\theta}} : \mathbb{R}^{d \times w} \to \mathbb{R}^{d \times (w+f)}$ for the FREDDIE task:

$$\mathfrak{U}_{\hat{f}_{\boldsymbol{\theta}}}(\widetilde{\mathcal{D}}_o) := \frac{1}{|\widetilde{\mathcal{D}}_o|} \sum_{(\boldsymbol{X}, \boldsymbol{y}) \in \widetilde{\mathcal{D}}_o} \mathfrak{u}(\hat{f}_{\boldsymbol{\theta}}(\boldsymbol{X})).$$

Using this uncertainty measure, we formulate our active learning procedure in Algorithm 4.2. Instead of strictly adding orbits one-by-one, we more generally allow for an *increment function* $\blacktriangle : \mathbb{N}_0 \to \mathbb{N}$ that yields the number of most uncertain orbits to add, depending on the number of already seen orbits. Out of the two retraining strategies described before, we decide not to retrain the model from scratch in each iteration but continue learning with the previous parameter values to save computation time.

```
/* actively trains the given model on an incrementally growing
   subset of the training data                                    */
active_learning(f̂_θ : ℝ^{d×w} → ℝ^{d×(w+f)} : model,
                Ω ⊆ P(D) : set of all training orbits,
                ▲ : ℕ_0 → ℕ : increment function):
    T := ∅  // current training orbits
    while |T| < |Ω| :
        U := hash_table()  // empty hash map
        for D_o ∈ Ω \ T do
            U[D_o] := 𝔘_{f̂_θ}(D̃_o)  // determine orbit uncertainty
        T := T ∪ top_k(U, ▲(|T|))  // add ▲(|T|) most uncertain orbits
        f̂_θ := train(f̂_θ, T)  // retrain model on updated set
    return f̂_θ
```

Algorithm 4.2: Our active learning scheme.

Chapter 5
Experiments

With the scene set, we can now conduct empirical investigations. Section 5.1 explores suitable input features from the MESSENGER data. Then, Section 5.2 describes our architecture search and compares the final models. Consequently, section 5.3 extensively evaluates the best model from the architecture comparison. Finally, Section 5.4 employs our active learning strategy to analyze how model performance scales with available data to address our central research question.

Training Setup: We use the PyTorch library [Pas+19] on a cluster with eighty Intel® Xeon® Gold 6248 CPUs @ 2.50GHz, eight Nvidia® GeForce RTX 2080 Ti GPUs and 512 GB RAM. As mentioned in Section 4.1, we fix the window size and future size to be $w = 120$ and $f = 20$, respectively, and use a weighted categorical cross-entropy loss. We minimize this loss with the Adam optimizer [KB15], having an initial learning rate of $\alpha = 10^{-5}$ and the default first and second moment running average coefficients of $\beta_1 = 0.9$ and $\beta_2 = 0.99$. The minibatch size is fixed to 1024 samples as we observed this value to yield stable and efficient training. All weights and biases are randomly initialized according to the PyTorch defaults, and we initialize all pseudozrandom number generators with seed 42. The training's termination criterion is early stopping on evaluation loss with a patience of three, i.e., once the loss on the evaluation set increases for three epochs in a row, training terminates. All performance metrics reported in this section apply to a model's state in the epoch of lowest evaluation loss.

5.1 Feature Selection

Prior to all further experimentation, we need to pin down the remaining variable that we left unspecified in Chapter 4, namely which of the available features in Table 3.1 to provide as input to the model. To begin with, it is reasonable not to tear apart groups of semantically related features like the three spacecraft position coordinates. While in principle we could use any of the groups, we should not provide all of them to the model as input. For one, the hand-crafted `EXTREMA` feature leaks global information about a time step. If its value is 2, for instance, the model can immediately draw from this that the spacecraft is farthest away from Mercury on its orbit and thus most probably in

the interplanetary medium. Since our modeling approach is time-local and supposed to only operate on actual measurements, we exclude EXTREMA from the game. Likewise, the manually added B[X|Y|Z|ABS]_DIPOLE feature group drops out since it does not contain measurements, but predictions of a model for Mercury's dipole [Par+21]. Furthermore, recalling Section 3.1, the MESSENGER dataset contains multiple correlated feature groups due to the spacecraft position being reformulated in different coordinate systems. However, feeding highly correlated features to a machine learning model in general does not improve performance but only increases input dimensionality and thus computational cost. Hence, Occam's Razor suggests we decide for an uncorrelated subset of the features.

The only a priori indispensable feature is the magnetic flux density measurements, for we are concerned with the bow shock and magnetopause crossings after all. Since it is only available in MSO coordinates, our desired subset contains at least BX_MSO, BY_MSO and BZ_MSO. Starting from there, we experimentally evaluate the performance of a small MLP in rounds where one feature group at a time is included as additional input. We train and evaluate these models only on 10% of the training and evaluation sets for time considerations. Table 5.1 shows the results, reporting the overall accuracy and macro F1 scores as defined in Section A.4, calculated over all individual time steps.

provided features	**macro F1**	**accuracy**
B[X|Y|Z]_MSO	53.38 %	71.00 %
B[X|Y|Z]_MSO, [X|Y|Z]_MSO	**67.71 %**	**84.07 %**
B[X|Y|Z]_MSO, DB[X|Y|Z]_MSO	64.30 %	81.32 %
B[X|Y|Z]_MSO, [RHO|PHI|THETA]_DIPOLE	59.35 %	76.32 %
B[X|Y|Z]_MSO, RHO, RXY	56.81 %	73.38 %
B[X|Y|Z]_MSO, X, Y, Z, D	48.19 %	65.34 %
B[X|Y|Z]_MSO, V[X|Y|Z|ABS]	49.30 %	67.42 %
B[X|Y|Z]_MSO, COSALPHA	54.10 %	71.30 %

Table 5.1: Contribution of different feature groups to MLP performance.

We observe that the other two feature groups in MSO coordinates significantly improve the model performance over the baseline of using just the magnetic flux density. All remaining features do not have an equally great impact on model performance, changing the macro F1 by at most five percentage points in either direction. Since the MSO spacecraft position achieves the largest margin, we include it into our subset of features and repeat the process. The next round shows that none of the feature groups achieves a noteworthy improvement except for the flux measurement error. Upon inclusion, it bumps up the macro F1 to just over 72 %, suggesting that we should add this feature group as well. We hence select {B[X|Y|Z]_MSO, [X|Y|Z]_MSO, DB[X|Y|Z]_MSO} as our final feature subset to train all further models on. Each having three spatial components, we get $d = 9$ as input time step dimensionality for our model architectures.

5.2 Architecture Comparison

The previous experiment fixes the input shape of our model architectures defined in Section 4.2. As mentioned there, the presented architectures result from an architecture search within each of the six categories MLP, CNN, FCNN, RNN, CRNN, and CANN. An initial broad hyperparameter optimization with random search hinted at rough regions of the parameter space to investigate further. In these regions, we continued to optimize manually. We do not present results from this search phase since it is highly iterative and training runs often only differ in a single detail.

Having a "worthy" representative of each category at hand, we finally compare their classification performance on the evaluation set across the categories. To this end, we employ the following metrics, discussed in Section A.4: Macro F1, overall accuracy, and the class-wise accuracies for the critical bow shock and magnetopause classes, respectively. Table 5.2 gives the results for all models, along with their respective number of trainable parameters as an indicator of their size.

model	macro F1	accuracy	SK accur.	MP accur.	# params
MLP	74.73 %	86.60 %	73.87 %	84.05 %	245180
CNN	77.80 %	89.29 %	74.75 %	84.62 %	1413372
FCNN	78.97 %	90.88 %	78.83 %	89.08 %	1444796
RNN	79.93 %	92.03 %	**81.50** %	91.75 %	237701
CRNN	**81.21 %**	**93.04 %**	79.22 %	**92.22 %**	267333
CANN	80.20%	92.46 %	81.30 %	**92.23 %**	246469

Table 5.2: Comparison of the model architectures.

We are aware that, in machine learning, conducting a fair comparison is delicately hard. For one, it is not even clear what "fair" is supposed to mean exactly. The simplest yardstick arguably is the number of trainable parameters in a model, for it directly correlates with its learning capacity. As Table 5.2 shows, we therefore kept a weather eye on ensuring that the parameter numbers between most categories are similar enough to be comparable. Alas, the CNN and FCNN constitute an exception to this. The CNN inevitably receives a high number of parameters from its final dense layer, and in preliminary experiments the FCNN required relatively large channel sizes to perform reasonably.

Keeping that in mind, we now discuss the results. As expected, the baseline MLP achieves the lowest performance. The CNN improves upon this, but at the cost of significantly more parameters. Likewise, the FCNN manages to go yet one better. As matches the widespread experience in deep learning with time series, convolutions thus are a beneficial component for our models. However, recurrent components seem to be even more gainful, as the simple RNN architecture outperforms the previous three and even has the highest bow shock accuracy. Hence, the CNN and FCNN gain no real benefit

from their capacity advantage. We see a clear margin between the first and second triple of models. Like we conjectured before, the combination of convolutional and recurrent blocks leads to yet another improvement, although not by a large margin. The CRNN achieves the highest overall scores and the highest magnetopause accuracy. Our experimental CANN accomplishes almost the same magnetopause performance but lags slightly behind on the overall metrics. Although the CANN achieves a higher bow shock accuracy than the CRNN, we continue our experiments with the latter as it has the best overall performance and is an established architecture.

5.3 Best Model Evaluation

With the CRNN found as the best model as measured by metrics over the evaluation set, we now assess its "real" performance on the test set. Not that the test set did not come into play before since it shall not influence the model development process. We first evaluate the CRNN's overall performance. Then we investigate the performance on classifying the past time steps only and integrate the window classifications to conduct inference on entire orbits. Finally, we isolate the future classification performance.

5.3.1 Overall Performance

To get a first impression, we recalculate the metrics we already used for the architecture comparison in Section 5.2 on the test set. Table 5.3 compares them with the results on the evaluation set.

set	macro F1	accuracy	SK accur.	MP accur.
eval	81.21 %	93.04 %	79.22 %	92.22 %
test	81.95 %	93.13 %	79.93 %	87.51 %

Table 5.3: CRNN performance on the test versus evaluation set.

Remarkably, the overall performance on the test set slightly surpasses that on the evaluation set. However, the bow shock accuracy is almost equal and the magnetopause accuracy significantly worse. This implies that the model performs better on the "easier" classes of interplanetary magnetic field, magnetosheath and magnetosphere. Since the architecture search proceeded on the evaluation set with special attention to the crossing classes, it seems plausible that the resulting model relatively overfits on those.

For a deeper insight into the model's classification behavior and the mistakes it makes, we consider the confusion matrix as defined in Section A.4. However, since the dataset contains a large number of windowed samples, in Figure 5.1 the absolute frequencies are normalized to percentages that sum up to one. By and large, the model gets most classes right. In particular, the two critical crossing classes are almost never confused for one another. However, what also catches the eye is that about half of the time steps predicted

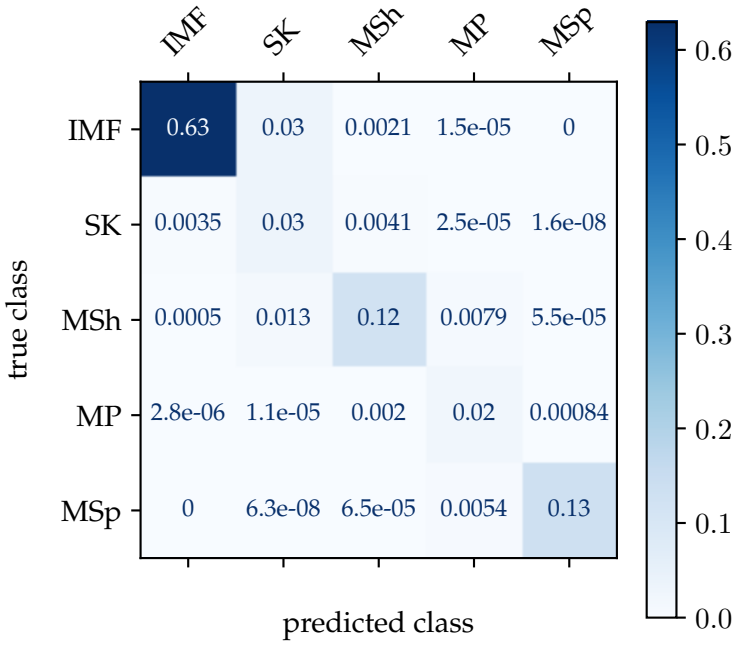


Figure 5.1: Normalized confusion matrix for the CRNN.

as bow shock incidence, in fact, belong to the interplanetary magnetic field region. In other words, the bow shock precision is pretty bad. Figure 5.2 highlights this more by showing the confusion matrix normalized in a precision-oriented way, i.e., dividing each column by its sum. In this vein, the diagonals contain exactly the precision values for the respective class as defined in Section A.4. Thanks to this different perspective, we also notice that the model confuses a considerable share of magnetosheath time steps with magnetopause crossings.

As precision is only one side of the coin, Figure 5.3 shows the analogous recall-oriented confusion matrix obtained by dividing each row by its sum, so that the diagonal contains the class recall scores as defined in Section A.4. Unfortunately, this is very often referred to as the "normalized" confusion matrix, although it is not the true normalized matrix. We find this terminology utterly *confusing*[5] and hence strictly distinguish between the *normalized*, the *precision-oriented* and the *recall-oriented* confusion matrix. Besides being confusing, reporting only the recall-oriented matrix can be misleading. In our case, for instance, it looks far better than the precision-oriented one. With 80 %, the worst recall score for bow shock is quite acceptable. By looking at the whole picture, we nevertheless know that precision is not.

[5] Actually, this pun was unintended.

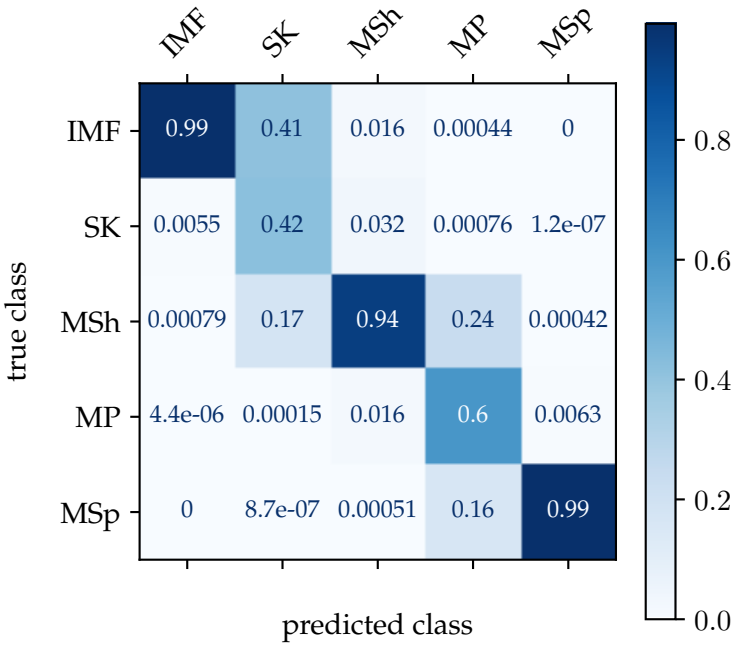


Figure 5.2: Precision-oriented confusion matrix for the CRNN. Each columns sums up to one.

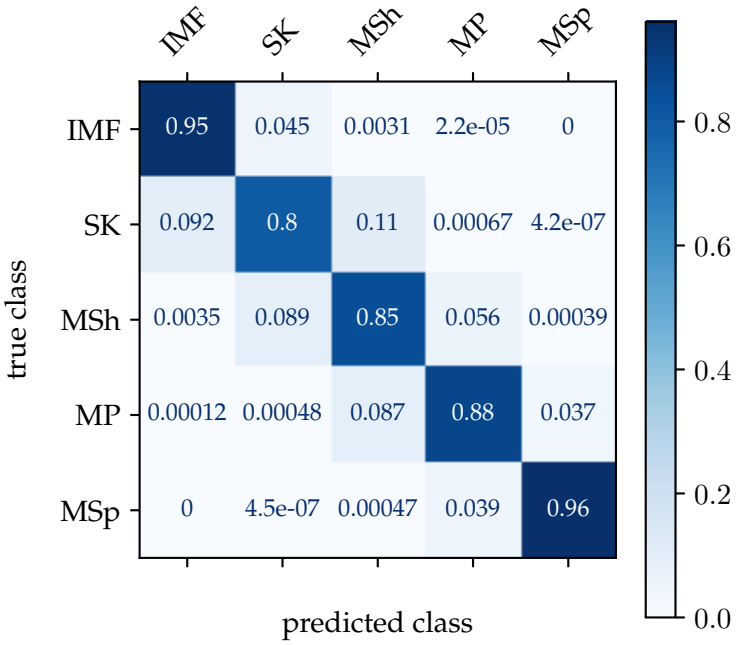


Figure 5.3: Recall-oriented confusion matrix for the CRNN. Each row sums up to one.

Our results so far show that the CRNN's performance on the magnetopause crossings is considerably better than on the bow shock crossings. We analyze this further by plotting *precision-recall curves* (*PRC*) for the two classes. A PRC for a class results from binarizing the classification task as "this class versus the rest", which means the probability outputs of the model for the other classes are summed together. Now, we could say that the model's decision for the class in question is positive if its respective probability is greater than 0.5. However, this threshold value is not mandatory. In fact, we can declare any value between zero and one as the decision threshold. Hereby, the precision and recall metrics will change in opposite directions. If the threshold increases, it is harder to hit this class, so precision will increase, but recall will decrease. By calculating precision and recall for many different threshold values in $[0, 1]$ and plotting the image of the resulting parametric curve, the PRCs in Figure 5.4 emerge. They display the tradeoff between precision and recall.

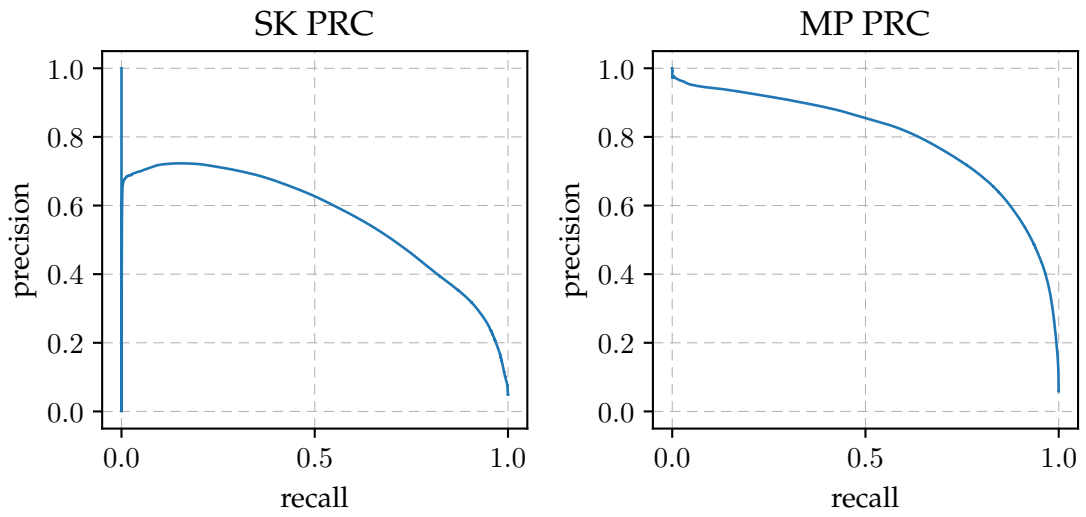


Figure 5.4: Precision-recall curves for the bow shock and magnetopause classes.

The PRCs again confirm our repeated observation that the model systematically handles the magnetopause class better. Regardless the threshold, the magnetopause curve is well above the bow shock curve. In case you are wondering about the sudden jumps in the beginning of the SK PRC: This happens whenever the decision threshold becomes so high that no true positives exist, but recall is still positive. Once recall becomes zero as well, the curve jumps to one due to the convention of $\frac{0}{0} := 1$. This artifact not appearing in the MP PRC is yet another indicator that the model is more confident on the magnetopause crossings.

5.3.2 Past-only Performance

In the previous Section 5.3.1, we evaluated the CRNN model using its entire output, i.e., class probabilities for the window time steps and future steps. In the following, we chop off the future predictions and repeat the entire evaluation machinery on the test set with the past time steps only. Table 5.4 shows the overall metrics on the test set and Figures 5.5, 5.6 and 5.7 provide the normalized, precision-oriented and recall-oriented confusion matrices, respectively.

macro F1	accuracy	SK accur.	MP accur.
82.00 %	93.09 %	79.64 %	87.77 %

Table 5.4: CRNN past classification performance on the test set.

We would expect the model to do slightly better on the past classification as compared to the entire "past+future" prediction. While this difference indeed turns up, it is almost negligible, and the bow shock accuracy of past-only classification is, in fact, a bit worse. Thus, we may say that the total performance is essentially equal. The confusion matrices support this conclusion, being almost identical to the ones in the previous subsection. Their lower left corners imply that the past-only part of the model does not confuse magnetopause nor magnetosphere with interplanetary magnetic field or bow shock. Furthermore, we observe that the class precision for magnetopause is a bit better and for bow shock a bit worse, matching the accuracy observation.

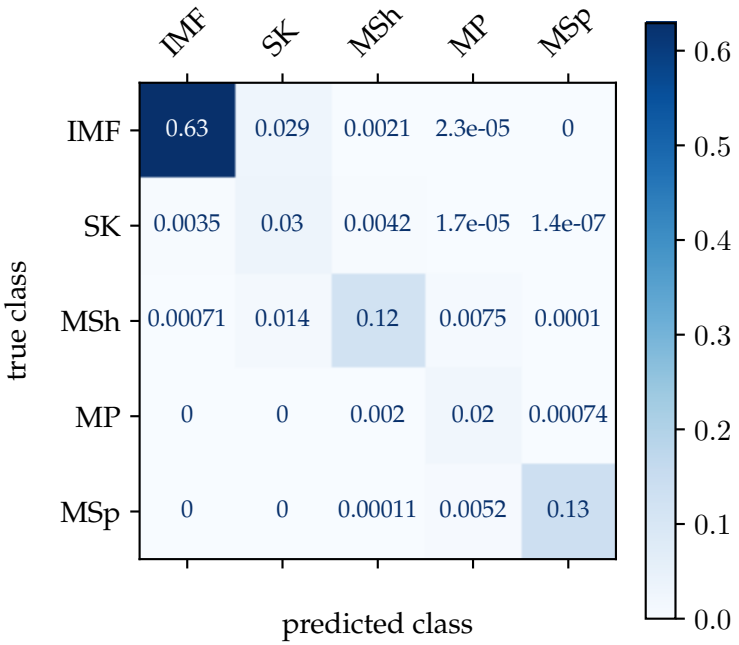


Figure 5.5: Normalized confusion matrix for the CRNN's past classifications.

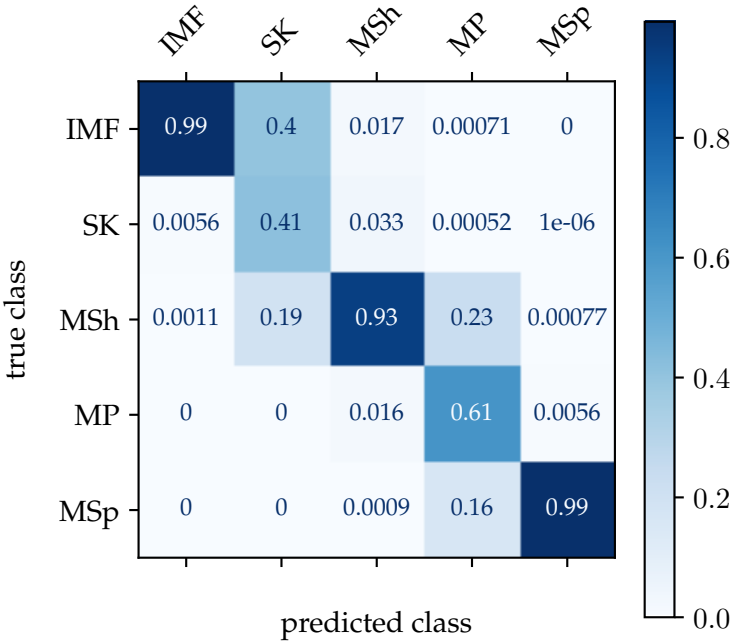


Figure 5.6: Precision-oriented confusion matrix for the CRNN's past classifications. Each column sums up to one.

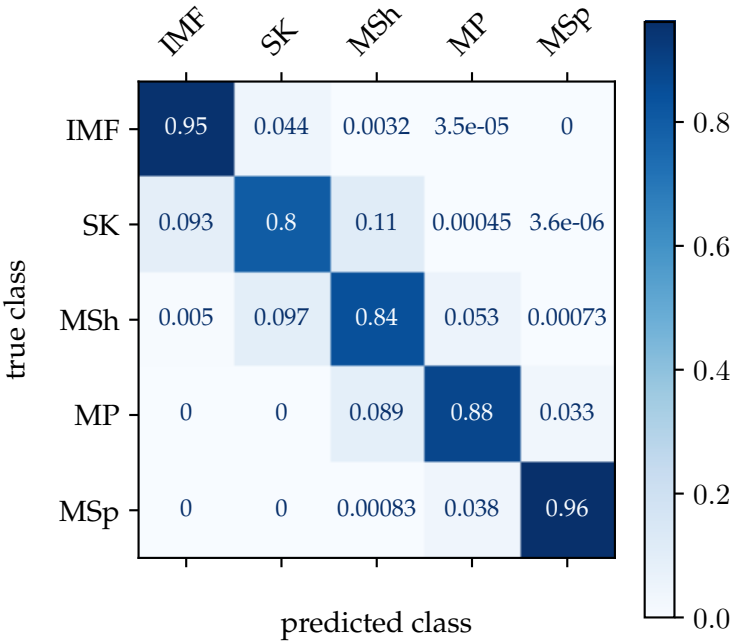


Figure 5.7: Recall-oriented confusion matrix for the CRNN's past classifications. Each row sums up to one.

So far, our evaluation attested good recall scores on the bow shock and magnetopause crossings but lousy precision scores, suggesting that the model is overconfident on these classes. We now want to confirm this insight gained from quantitative analysis with a qualitative inspection. To this end, we utilize the model's past-only classifications in a window to infer predictions for an entire orbit.

Given any orbit's measurements, we generate windows of the same size w used for training the model with Algorithm 4.1. Then, we let the model classify all time steps in each window. In this manner, almost every time step of the underlying orbit receives distinct predictions from w different windows in which it is contained. The exception are the first $w - 1$ time steps of the orbit, which do not have a sufficiently long past. To integrate the predictions for a single time step, we average the w different probabilities for each of the five classes, again yielding probabilities that sum up to one. Finally, a softmax on the integrated probability distribution gives a class prediction for the time step. We do this for all orbits in the test set and plot their magnetic flux density along with the predictions.[6]

We visually inspected each of the 316 orbits in the test set. The model mostly predicts contiguous magnetic regions and almost always in the correct order. Nevertheless, we identify four major qualitative issues:

- *Crossing Overconfidence:* On virtually all orbits, the extent of the model's predicted crossings is too large. Like in Figure 5.8, the model predicts the bow shock and magnetopause regions to be wider than they actually are. Particularly the predicted bow shock crossings tend to steal from the neighboring regions. Underconfidence, on the other hand, rarely happens, matching our quantitative analysis.
- *Scattered SK crossings:* In addition to being too wide, bow shock crossings are frequently torn apart into several crossings like in Figure 5.9. The model seems rather sensitive to relative changes in the magnetic flux.
- *Scattered MP crossings:* Analogous to bow shock crossings, magnetopause crossings are also sometimes dispersed into separate ones like in Figure 5.10. However, this happens significantly less often than with bow shock crossings, matching our finding that magnetopause performance is generally better.
- *Random MSh spikes:* Very rarely, the model hallucinates small magnetosheath regions in unexpected places like in Figure 5.11.

On a handful of orbits, like that in Figure 5.12, all of the above problems appear at the same time. For future work to tackle these challenges, we propose some countermeasures in Chapter 6. In conclusion, we confirmed what the quantitative metrics suggested, and additionally gained more insight through visual inspection.

[6] All plots for the entire test set reside in the code repository accompanying this work, see Chapter 1.

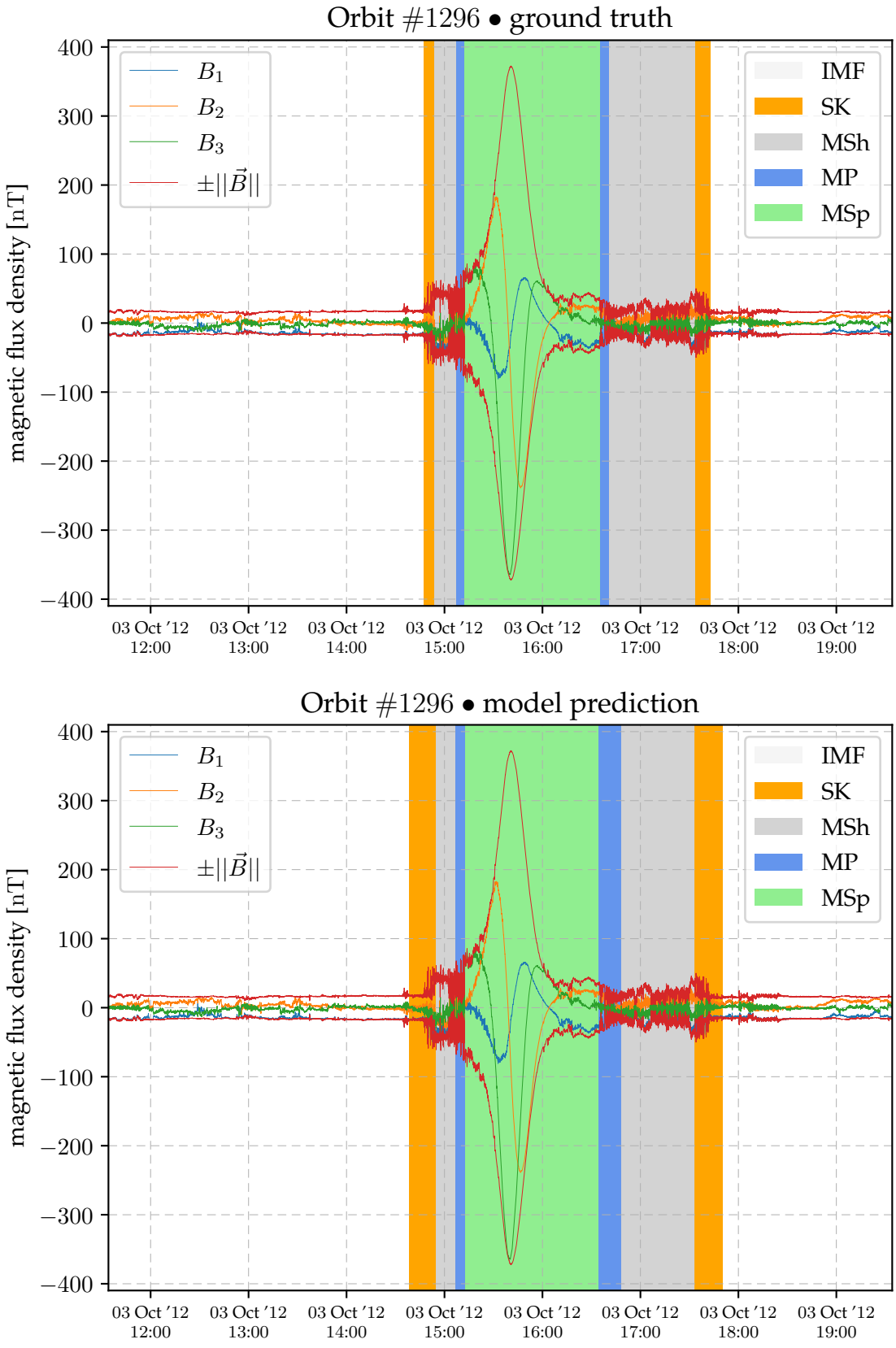


Figure 5.8: Inference on an orbit where predictions are overconfident on crossings.

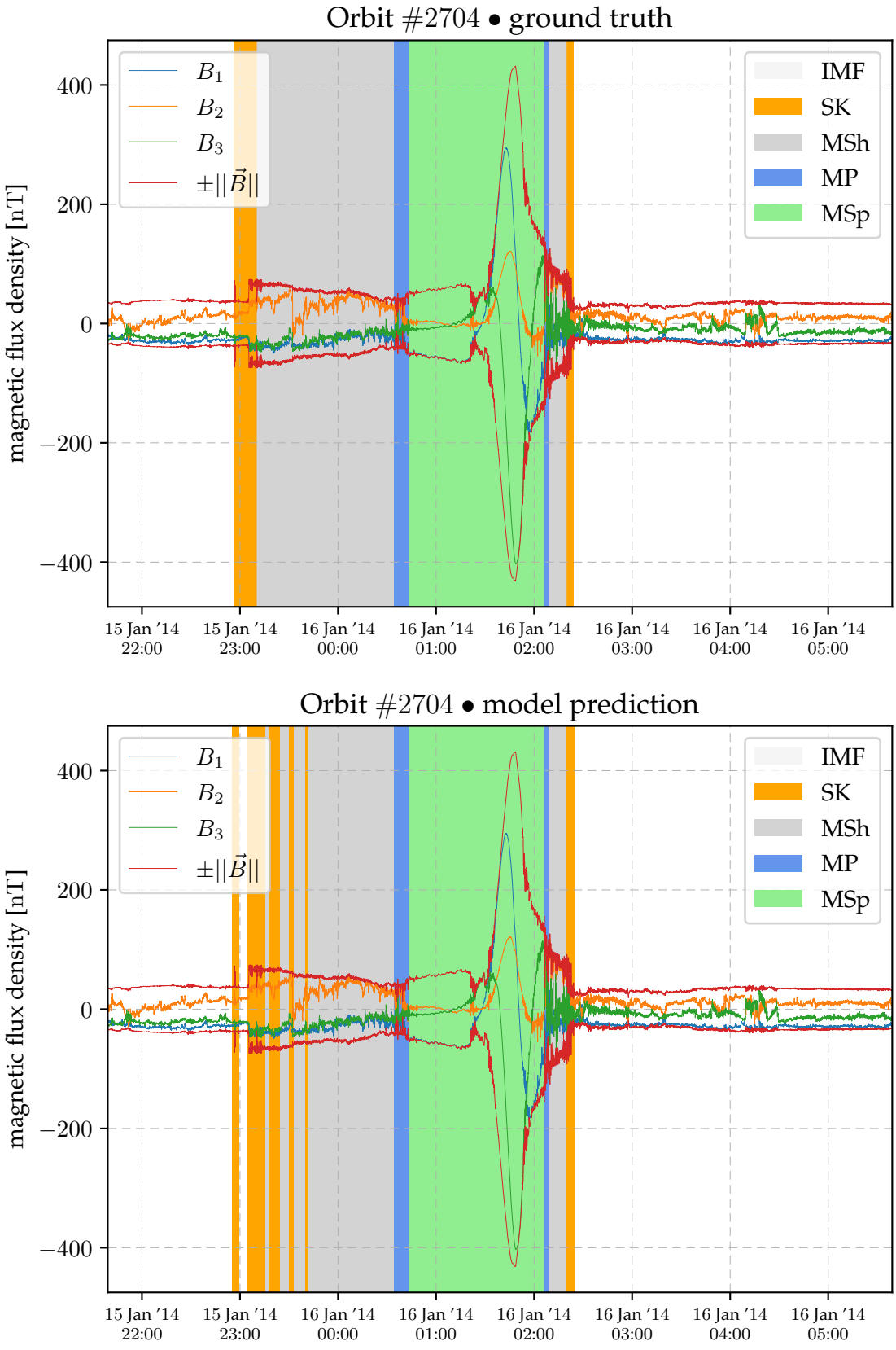


Figure 5.9: Inference on an orbit where predictions scatter a bow shock crossing.

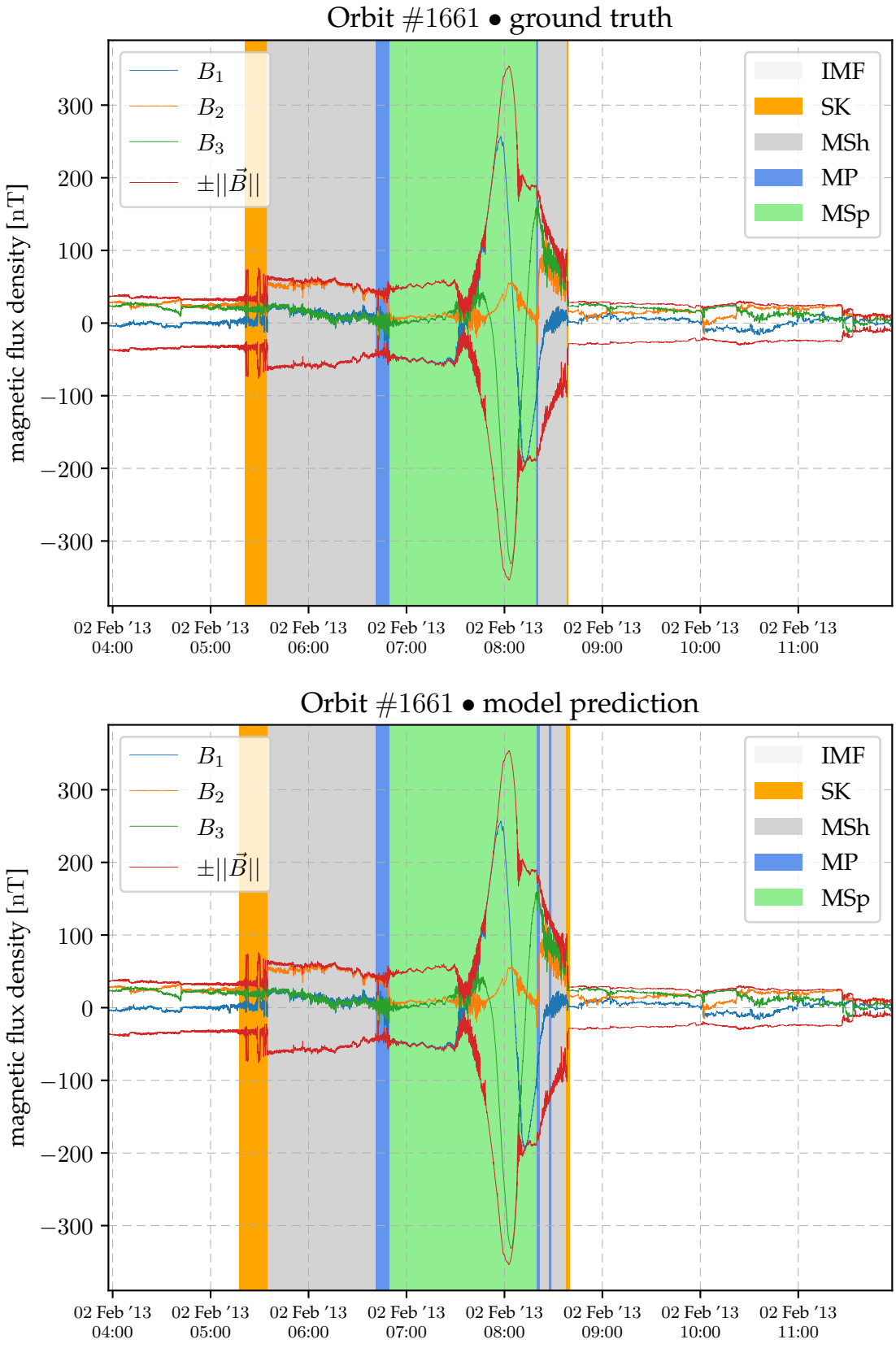


Figure 5.10: Inference on an orbit where predictions scatter a magnetopause crossing.

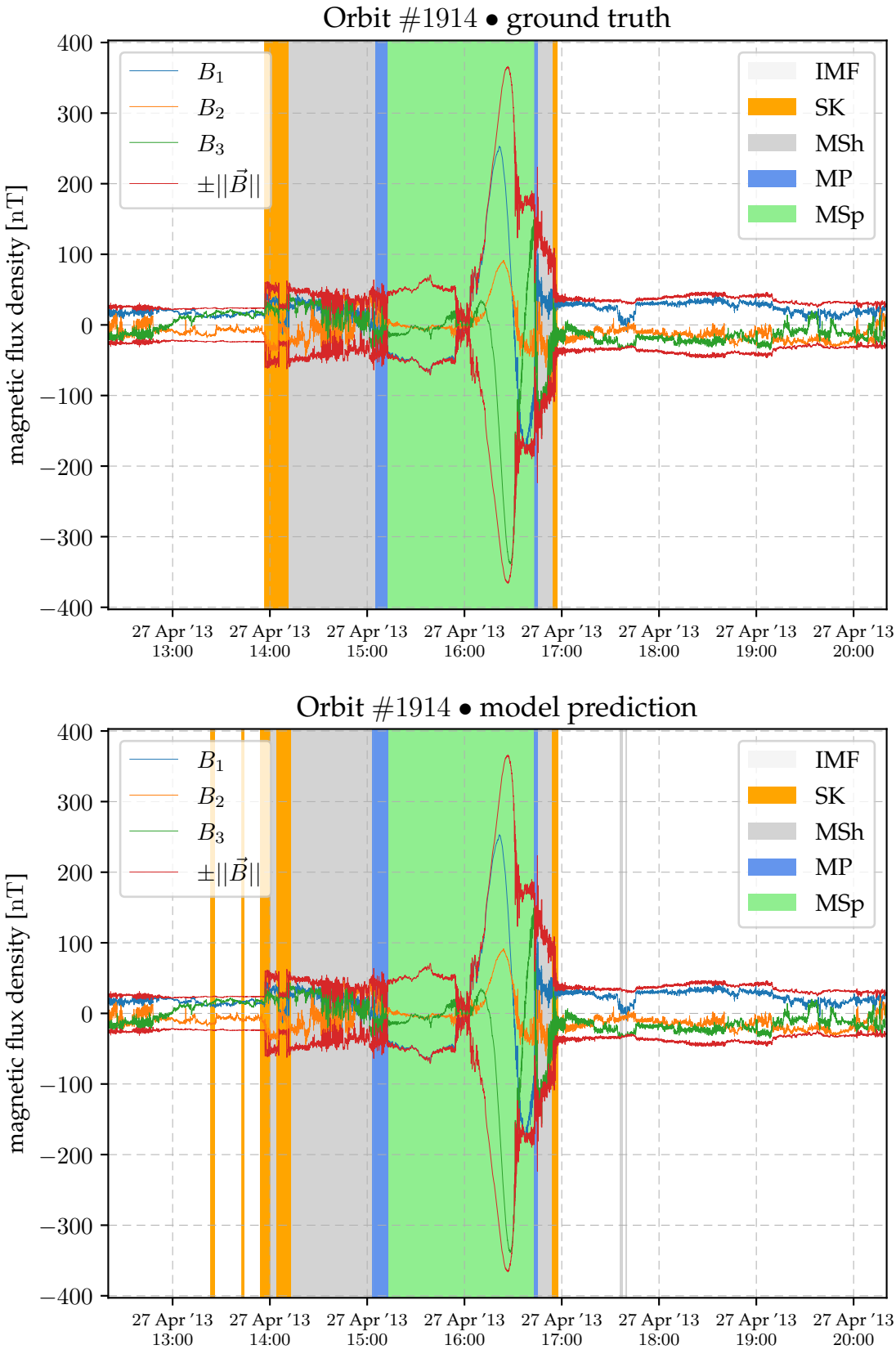


Figure 5.11: Inference on an orbit with additional predicted magnetosheath regions.

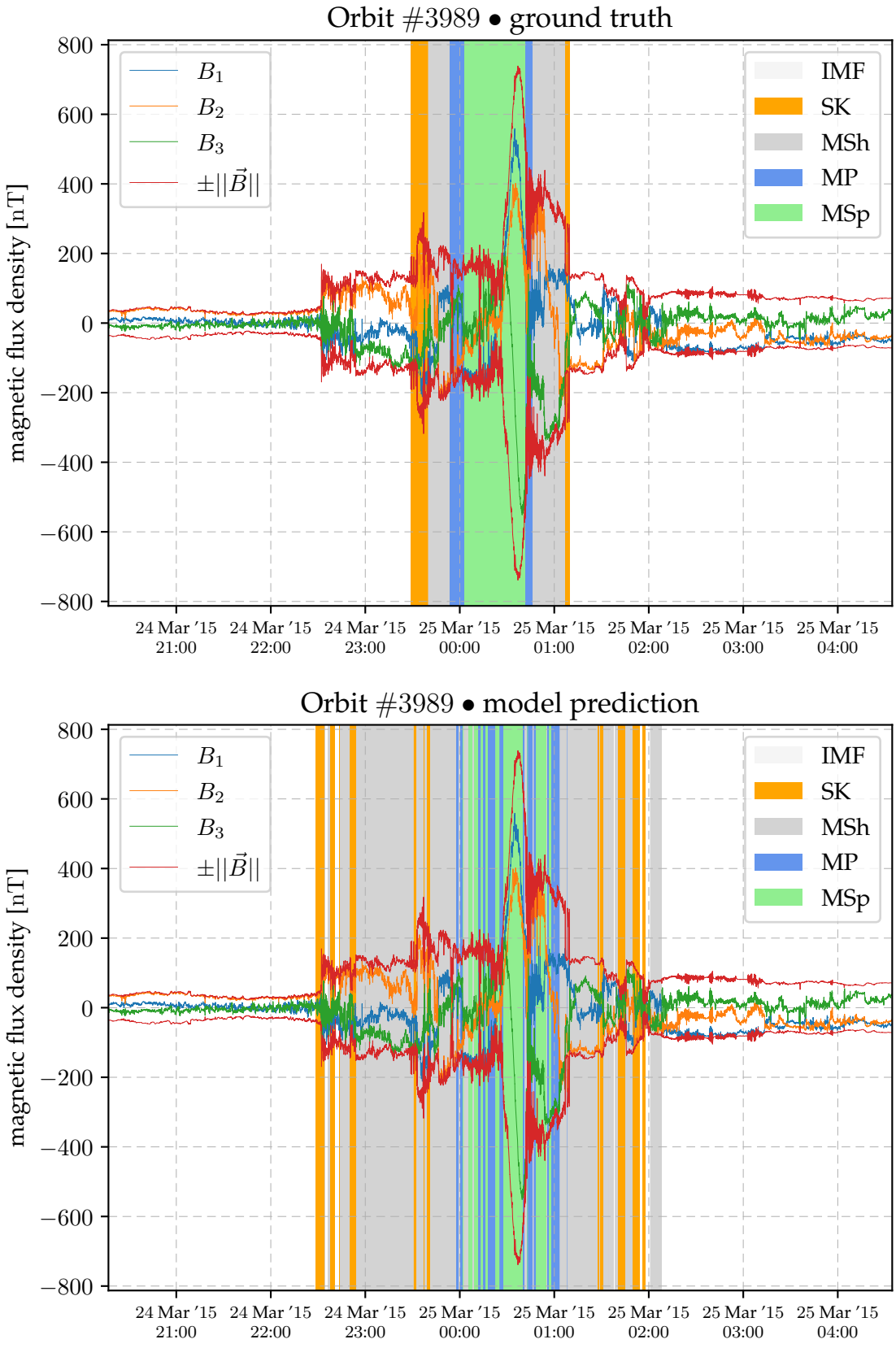


Figure 5.12: Inference on an orbit where predictions are utterly broken.

5.3.3 Future-only Performance

Analogous to the past classification, we now focus on the future classification performance of the CRNN by chopping off the predictions for the time steps contained in a window. This is particularly interesting since the model has no corresponding input for the future time steps. Table 5.5 again shows the summary metrics and Figures 5.13, 5.14 and 5.15 the confusion matrices.

macro F1	**accuracy**	**SK accur.**	**MP accur.**
81.35 %	92.78 %	78.05 %	86.44 %

Table 5.5: CRNN future classification performance on the test set.

Based on our insights about past classification in the previous subsection, we now expect the future classification performance to behave complementarily. After all, the set of all predictions is precisely the disjoint union of all past and future predictions. Indeed, comparing Figure 5.14 with Figure 5.6 yields that precision on bow shock crossings is a bit lower for future classifications. Furthermore, Figures 5.15 and 5.7 show that recall for future classification is slightly worse on both classes. However, these differences are minor, and besides that, future-only predictions seem to behave essentially the same as past-only classifications. We suspect that this negligible difference arises from the model simply predicting the same class for all future time steps as it did for the rear time steps within a window since this heuristic is correct for almost all samples. In Chapter 6, we thus propose a different task definition for future work.

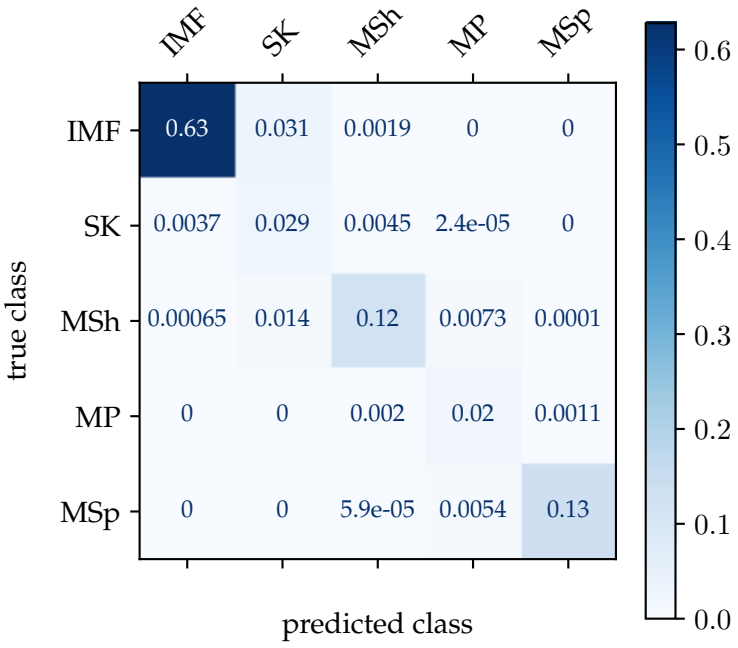


Figure 5.13: Normalized confusion matrix for the CRNN's future classifications.

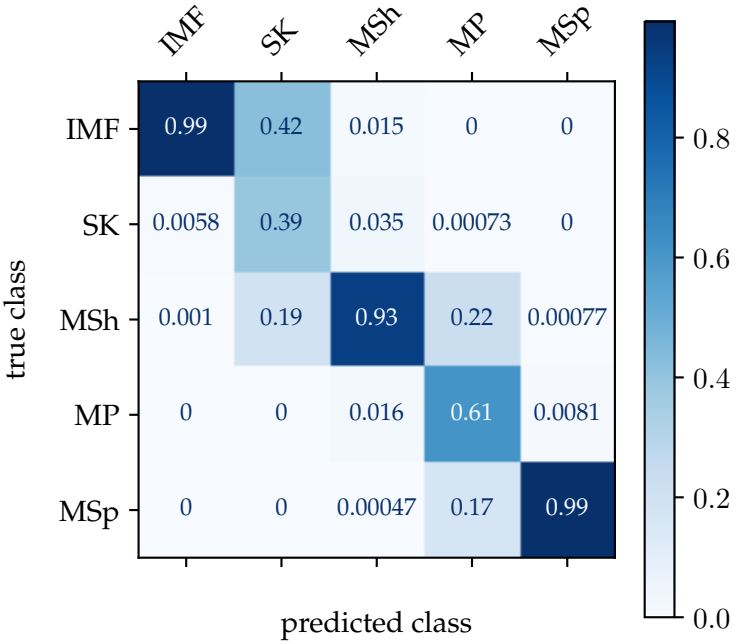


Figure 5.14: Precision-oriented confusion matrix for the CRNN's future classifications. Each column sums up to one.

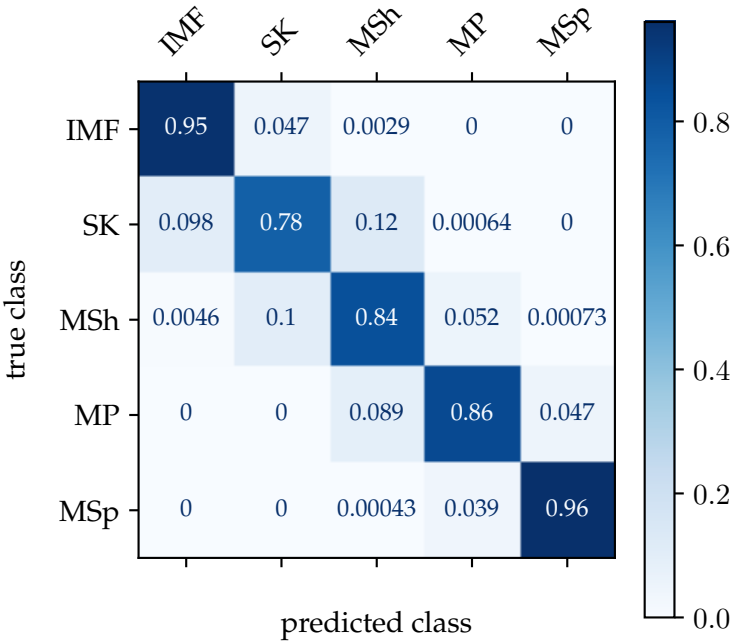


Figure 5.15: Recall-oriented confusion matrix for the CRNN's future classifications. Each row sums up to one.

5.4 Active Learning

After extensive evaluation of the CRNN model obtained by passive learning, we finally incorporate our active learning approach from Section 4.3. In particular, we run Algorithm 4.2 with two different choices for the increment function: one leading to an exponentially growing training set and one leading to a linearly growing training set. In this manner, we explore how the classification performance scales with available data and determine the order of magnitude of orbits required for a satisfactory model.

5.4.1 Linear Increment

To get a rough overview of the active learning dynamics, we run a preliminary active learning experiment with the increment function defined by $➤(n) := \max\{1, n\}$. This definition doubles the training set size in each iteration, ensuring an equal proportion between "old" and "new" orbits. Figure 5.16 plots our four evaluation metrics also used in the previous sections over time, i.e., against the number of already included orbits. It essentially constitutes a *learning curve* for the model.

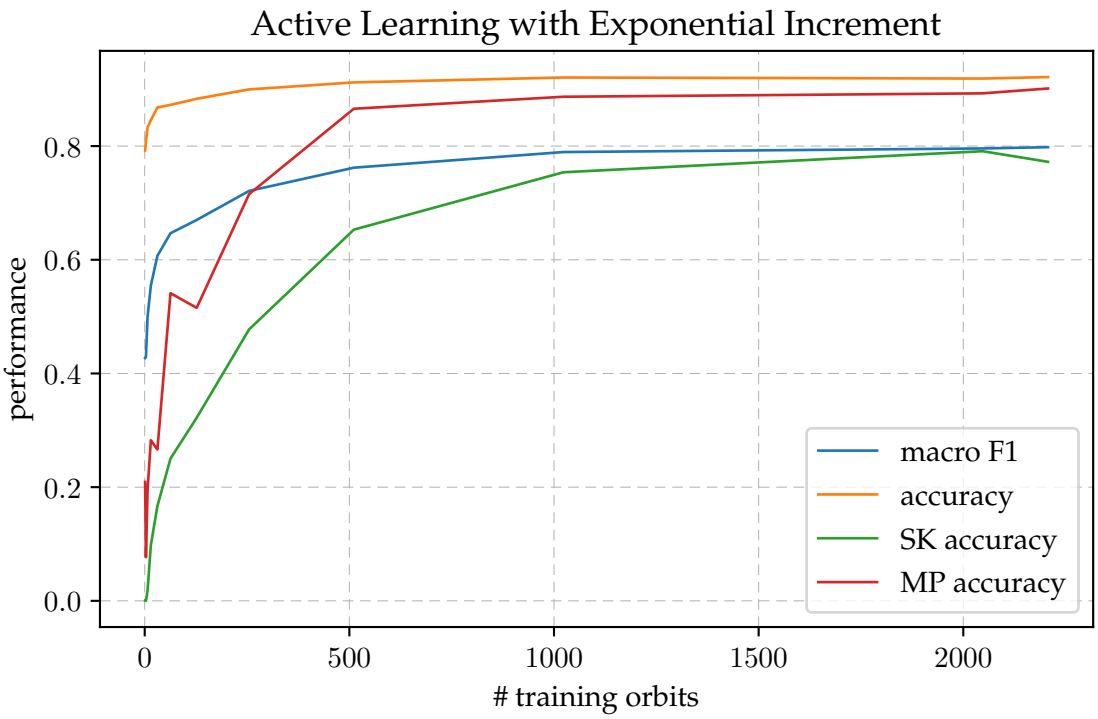


Figure 5.16: Active learning curve with exponentially growing training set.

We observe a rapid increase of all metrics in the beginning, followed by a longer period of flattening. The performance after having used all orbits is comparable to the passively trained model, but slightly worse. We suspect overfitting during the first iterations to be the culprit. Especially the very first iteration on only one orbit lasted for a multiple of the

gradient descent epochs we usually observed in passive training. With more iterations, the epoch number until early stopping decreased constantly. This might imply that the active learner homed in on the first orbits it received and afterward only occasionally learned something new. From Figure 5.16, we may tentatively derive an upper bound of around a thousand orbits on our desired quantity of orbits needed for satisfactory performance. Beyond this number, not much happens anymore.

5.4.2 Constant Increment

To get a more fine-grained development of the model's performance over time, we conduct our main experiment with a linearly growing training set, where a constant number of orbits enters the training set each iteration. Intuitively, one would simply want to add orbits one by one. However, there are two drawbacks to this approach. Firstly, adding a single orbit in each iteration will cause over 2000 iterations to take place in total, which makes the experiment last infeasibly long. Secondly, as we observed in the exponential approach, with too few orbits overfitting is a huge problem for the active learning scheme. As a reasonable choice, we thus increase the training set by ten orbits in each epoch, according to $\blacktriangle(n) := 10$. Figure 5.17 shows the resulting learning curve. We are only able to present the results for the first 1000 orbits since the experiment has only proceeded this far up to now. Nevertheless, its evolution is clearly evident from the first half of the experiment.

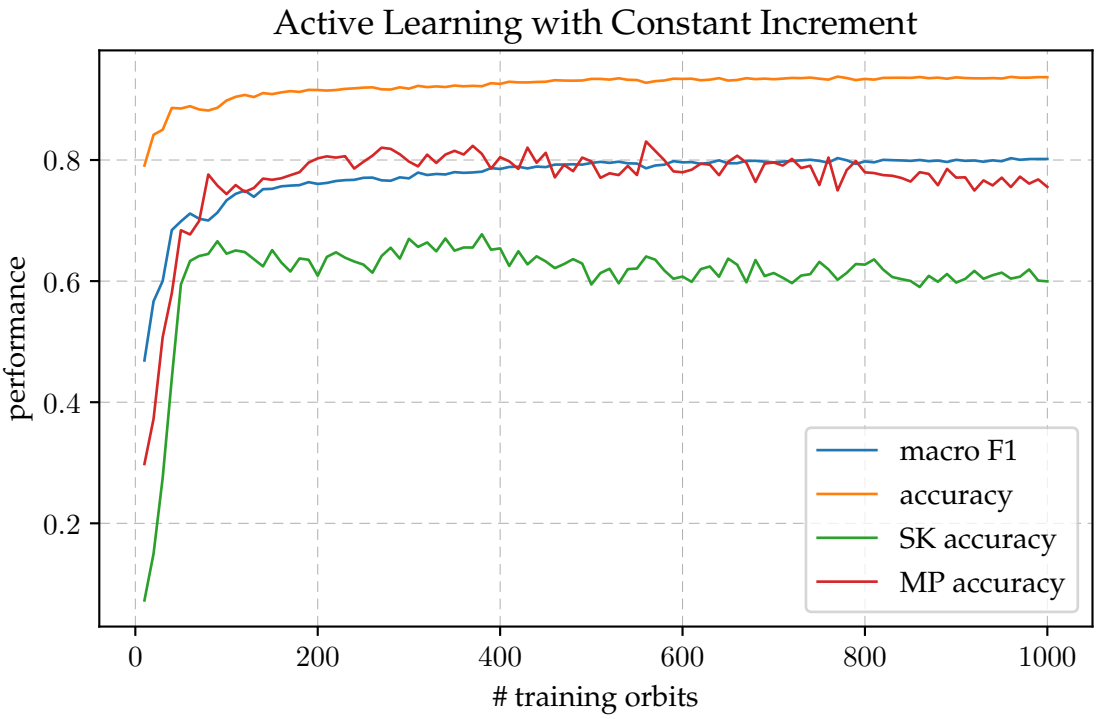


Figure 5.17: Active learning curve with linearly growing training set.

Like in the exponential setting, we observe a rapid performance increase in the beginning. However, this time the metrics seem to hit the ceiling after 100 orbits already and then continue to increase only marginally. After around 500 orbits, the class accuracies for bow shock and magnetopause even begin to decrease, while the overall metrics go further up. This divergence implies that the model focuses more on the majority classes and increasingly ignores the two we are concerned with. As a result, the performance after having trained on half of all orbits is essentially equal to the passively trained model in terms of macro F1. However, the class-wise accuracies are considerably worse, in particular, worse than was the case with a linear increment.

We suspect the constant increment of causing this mediocre development. Since the number of orbits added in each iteration does not depend on the number of already seen orbits, their relative proportion becomes more and more skewed towards the known orbits. Consequently, the model decreasingly learns new things but keeps optimizing over known orbits repeatedly. In this regard, the linear increment is superior since it ensures a constant proportion of "new" vs. "old". For future experimentation, we thus propose to employ an exponential schedule but with a smaller base so that orbits are not added too quickly. Furthermore, in the beginning, orbits should be added in groups of a specific minimum size. For instance, a choice of $\blacktriangle(n) := \max\{10, \lfloor \frac{n}{4} \rfloor\}$ seems reasonable as it ensures that a fifth of the current training set consists of unseen orbits and yet allows fine-grained investigation of the lower end of the training set size spectrum like the constant increment does.

A straightforward approach to avoid overlearning would be retraining the model from scratch instead of reusing the previous parameters. In this vein, the model simply starts fresh in each iteration. However, this sledgehammer solution does not come without drawbacks: Firstly, it increases the training time even further since the model cannot profit from parameters that already are in a suitable region of parameter space. Secondly, it conforms less to the pure concept of an active learner incrementally increasing its knowledge but as it repeatedly resets the learner's memory and gives it increasingly more to learn at once. Instead, as a less extreme remedy we propose applying a slight random noise to the model's parameters before each retraining step. This might keep the model from forever following the same descent path in the loss landscape it entered during the first iterations but still preserves the majority of what the model has already learned.

Despite the mediocre absolute performance, we may still infer an answer to our question based on the relative development. The crossing performance continues to increase until just under 500 orbits form the training set. While the crossing accuracies do not reach the performance of the passively trained model at this point, the overall accuracy and macro F1 do. It is thus worth investigating the surrounding region more closely. We do this by taking into account further aspects of the active learning process.

Besides the performance metrics, we can also consider the model's orbit uncertainty as defined in Section 4.3. After all, it is the very measure by which orbits are selected for training in the active learning scheme and indicates the model's confidence about its decisions. We are interested in the point from which on uncertainty does not significantly decrease anymore, meaning that the model has nearly saturated its learning capabilities. In a sense, the model has "seen enough" until that point. To this end, Figure 5.18 plots the worst occurring orbit uncertainty at each iteration as a function of the number of orbits included in the process.

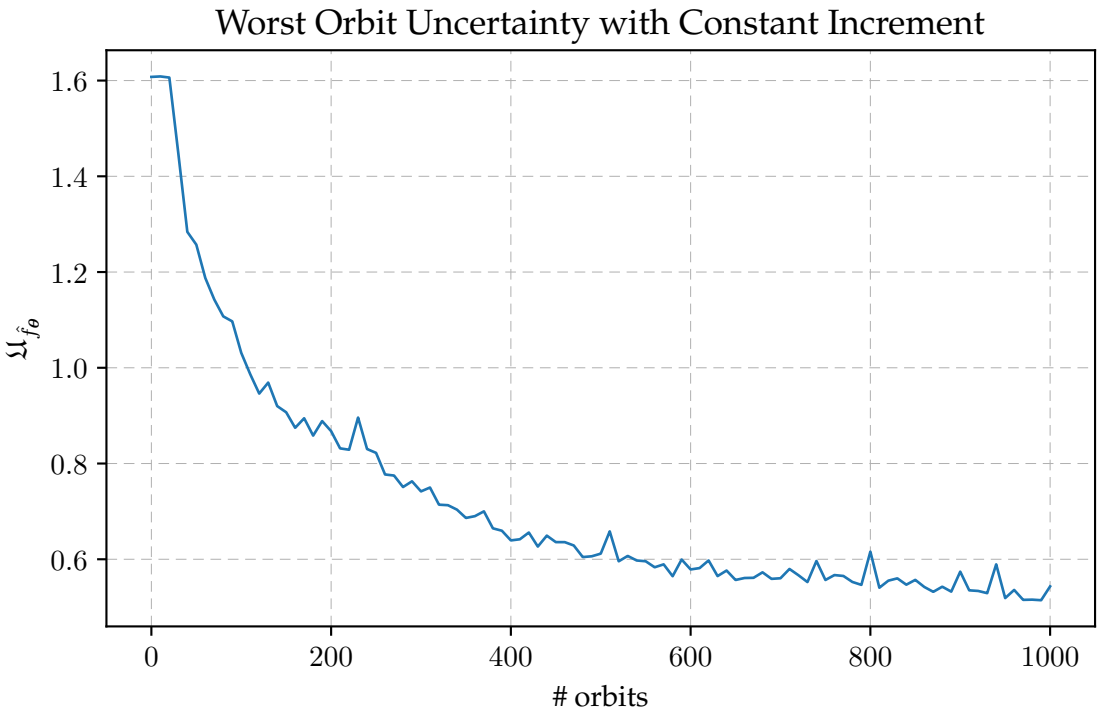


Figure 5.18: Worst orbit uncertainty development with a linearly growing training set.

In the beginning, the orbit uncertainty starts off with its global maximum of just under $\log(5) \approx 1.609$, which according to Section A.1 is the entropy of a uniform distribution on five outcomes. This is not a coincidence but results from the definition of our orbit uncertainty measure in Section 4.3 and the model's random parameter initialization. For illustration, the prediction with highest entropy for a single time step in the first iteration was

$$[0.201, 0.208, 0.187, 0.207, 0.197],$$

being indeed very close to a uniform distribution. Remarkably, the orbit uncertainty remains on this high level for two more iterations, i.e., until 30 orbits are in the training set, while the performance metrics in Figure 5.17 improve significantly. This implies that in these iterations, the output probabilities did not depart from the equilibrium but merely rearranged themselves such that their maximum is at the correct position.

After its initial plateau, the orbit uncertainty decreases rapidly from the fourth iteration onwards. Analogously to the performance metrics in Figure 5.17, the uncertainty eventually flattens out and seems to almost asymptotically approach a value of 0.5. Again, the lion's share of improvement happens during the first half until about 500 orbits are included. We provide further evidence for this claim by considering not only the worst overall orbit uncertainty. Like we did as an example for the first iteration, we also consider the worst uncertainty/entropy of a single time step prediction in an iteration. Figure 5.19 plots these worst entropies, again as a function of the training set size. While there are iterations with an outlier, the worst-case entropy decreases steadily until around 450 orbits are included. Afterward, the worst case does not improve.

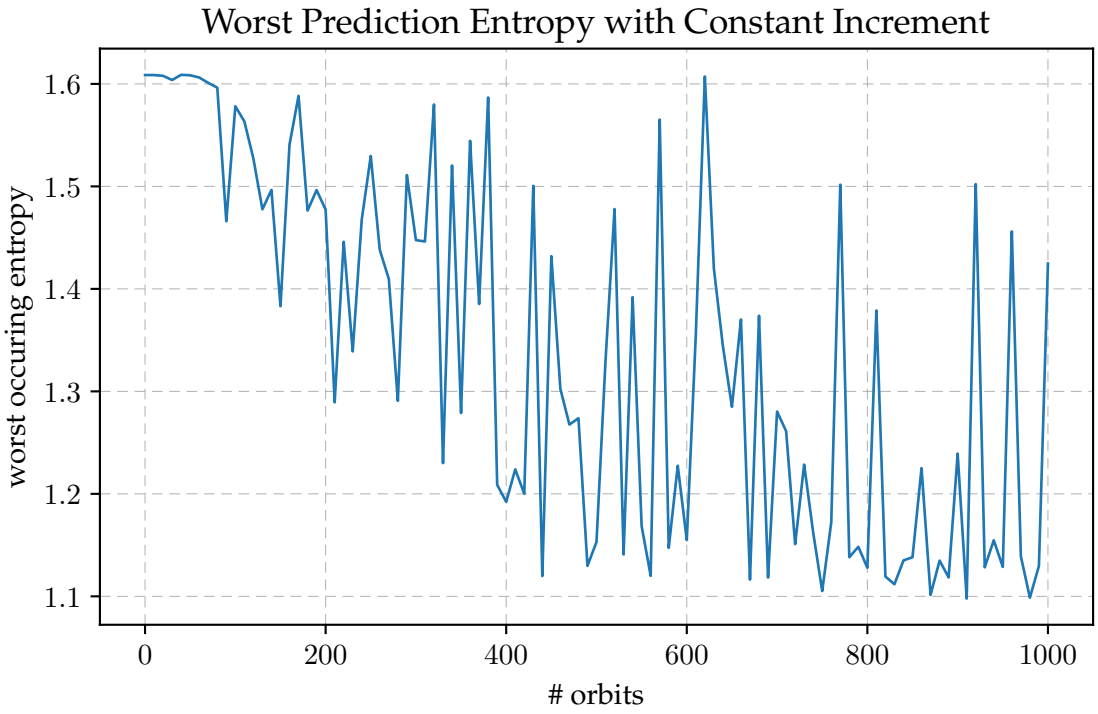


Figure 5.19: Worst single time step entropy with a linearly growing training set.

Taking all insights together, we conclude that the model's learning capacity saturates after 450 to 500 orbits. This constitutes a new upper bound for the number of orbits required for a representative model. When summing the duration spanned by the concrete orbits chosen by the model, this equates to roughly two full Mercury years' worth of MESSENGER orbits. We may therefore claim that two Mercury years make for a sufficient set of observations for the model to learn from. On the other hand, revisiting Figures 5.17, 5.18 and 5.19 suggests that one complete Mercury year (around 230 orbits in this case) is at least required. Hence, it remains for future work to explore the range in between. With the improvements we proposed before and summarize in Chapter 6, it might even be possible to lower this bound to just one Mercury year.

Chapter 6
Conclusion

In this work, we built a discriminative end-to-end deep learning model for detecting Mercury's bow shock and magnetopause crossing signatures based on raw measurements from NASA's MESSENGER mission. Moreover, we devised an active learning scheme to address the question of how many orbits' worth of measurement data is required for a representative model.

To this end, we formulated a machine learning task that, given a window of measurements, consists in predicting one of five magnetic regions for each time step in the window and even classifying several future time steps for which no measurements are provided. For this task, we devised six different neural network architectures and compared these empirically. We found that convolutional layers, performing feature extraction, followed by either recurrent (CRNN) or attention layers (CANN), addressing the temporal aspect of the data, are well suited to the task. Our best model, the CRNN, achieves a macro F1 of about 82 % and manages to detect the crossings with reasonable accuracy, with the magnetopause classification performance consistently surpassing that of the bow shock class. In particular, the recall scores of 78 % for bow shock and 86 % for magnetopause are convincing. However, the model, in turn, lacks appropriate precision, which is 39 %, and 61 %, respectively. This implies severe overconfidence, especially on bow shock crossings. We qualitatively confirm by visual inspection that indeed the predicted crossings are almost always too wide. Furthermore, we find that a crossing is frequently predicted as multiple successive ones, and sometimes unwanted artifacts of magnetosheath appear.

Based on the best model, we approached the central question underlying this work with an active learning scheme. It employs the uncertainty sampling strategy with a custom orbit-level measure based on Shannon entropy, by which we iteratively determine the next orbits to include in the training set. After a preliminary experiment with an exponentially growing training set, we conducted our main experiment with a constant increment of ten orbits per iteration. Its mediocre results imply that a constant increment is inferior to a linear one, for the latter ensures a constant portion of unseen orbits throughout all iterations. Furthermore, both increment strategies suffer from overfitting on the very first orbits included in the process. Nevertheless, we were able to derive that at least one and at most two Mercury years' worth of measurement data may be sufficient for a representative model that achieves satisfactory performance.

While our work yields comprehensive insights into the structure of the MESSENGER magnetometer data and hence the magnetic dynamics around Mercury, it can only provide a starting point. We point out possible improvements and further directions for future work to address in the following paragraphs.

A data-driven approach can only get as good as the underlying data is. Therefore, we propose to improve the dataset itself. As we discussed, the annotations from Philpott et al. have a major drawback: Whenever multiple successive crossings indeed occurred, they were combined into a single large crossing. Besides distorting reality, this might well be one cause for the crossing overconfidence of our final model since it was taught to also detect regions as crossing that, in fact, were none. As a heuristic countermeasure, future efforts should at least remove orbits with suspiciously long crossing annotations. However, it would be best to strive for new high-quality annotations as a gold standard for the MESSENGER magnetometer data. As shown, it will not be necessary to label all 4000 orbits, but only two entire Mercury years and optimally a third one for evaluation and testing. In fact, our algorithm can be modified for reannotating with active learning by inserting the human annotator into the loop. Finally, we suggest augmenting the dataset with meta-fields that for each time step indicate the Mercury year and orbit number within this Mercury year. In this vein, it becomes feasible to investigate whether an active learner indeed chooses orbits of different seasons within a Mercury year, as would seem plausible for it to gain maximum information.

Another future direction concerns the machine learning task formulation. We suspect that our definition in Section 4.1 might be too ambitious since it comprises the classification of each time step within a window individually. Instead, it suffices to let the model predict only a class for the window's ultimate "present" time step. For inference, this would result in one prediction for each time step instead of multiple votes. Likewise, the future classification output may be compressed to a single value. For instance, this could be a binary flag indicating whether the class predicted for the present time step changes in the near future. Pruning the task definition from sequence-to-sequence to sequence-to-pair would simplify model architectures and possibly improve performance. The extreme would be dividing the combined past and future classification into separate tasks altogether. Regarding the format of a class prediction, we see two possible improvements: Firstly, to tackle the model's overconfidence on the crossings, it might help to apply *label smoothing* [MKH19], i.e., not using a hard one-hot vector as the target but a slightly softened version. Secondly, one might entirely discard the categorical target, as it ignores the magnetic regions' topology. To capture their neighborhood relationships, we propose an experimental formulation of the prediction target on a unit circle in two-dimensional space. Particularly, the region labels could be the fifth roots of unity in the complex plane. An additional penalty term in the loss function could then incentivize the model's predictions to have a norm close to one.

On the modeling front, there remains an inexhaustible pool of possibilities to explore, of which we only name a few. To begin with, the CANN architecture should be investigated further since its results were comparable to the CRNN in our study. Beyond, additional model architectures like *ResNet* [He+16] or the *Time Series Transformer* [Zer+21] could be employed. In general, we expect any significantly larger model with regularization mechanisms like a parameter norm penalty, dropout [Sri+14], or batch normalization [IS15] to improve on our results in absolute terms. Our models were deliberately chosen small to reduce computational cost so that they did not require regularization but, on the contrary, might lack learning capability. Overall, a comprehensive ablation study of architectures is a to-do for future work. In particular, the contribution of the future classification part should be investigated. As a technical detail, we figure that employing *dilated convolutions* [YK16] might benefit absolute performance as well since they exponentially expand the receptive field of deeper neurons. Moreover, provided that the learning task continues to consist of combined past and future classification, the model might profit from eventually diverging representations for the two subtasks instead of sharing the same representations until the output layer. For example, in later layers, the model could branch off into two subnetworks.

Just as the possibilities for modeling are unlimited, so are those for training techniques. Firstly, instead of the Adam optimizer employed in our work, it is advisable to experiment with different options such as vanilla stochastic gradient descent or AdamW as no optimizer is fundamentally superior [Wil+17]. Secondly, even with adaptive optimizers like Adam, practitioners observed improvements by using learning rate schedules [Smi18]. This opens yet another Pandora's box, since the concrete scheduling possibilities are infinite. Usually, a simple linear day or a cyclic schedule with sawtooth or sinusoidal waveform is used. Thirdly, the inherent under-representation of the crossings may be handled in different ways. We accounted for the class imbalance by accordingly weighting the categorical cross-entropy loss function. It could be worth experimenting with the more sophisticated *focal loss* [Lin+17] that is successful in object recognition tasks. Instead of adjusting the loss, it might be even more beneficial to just under-sample the majority classes, particularly the interplanetary magnetic field region. Hereby, the class imbalance is removed on data level. However, doing this before each training epoch comes with an immense computational cost.

Ultimately, the active learning approach devised in this work is not set in stone. Firstly, we followed an uncertainty sampling strategy for its computational efficiency. Nevertheless, future research should explore other methods like query-by-committee or estimated error reduction. The latter might be implemented efficiently by using a *loss prediction module* [YK19] instead of actually calculating the expected loss. Secondly, our experiments highlight the importance of a practically constant proportion between seen and unseen orbits during the active learning process. As we outlined in Section 5.4, a

linear increment that leads to an exponential training set growth with a small basis seems promising. Thirdly, the issue of the active learner overfitting individual orbits during the first iterations needs to be addressed. As we discussed, this might be done by simply requiring a fixed lower bound of the increment function. Furthermore, overfitting could, in part, result from the model capacity being too large in relation to the training set during the first iterations. As a countermeasure, one could regularize the model with dropout [Sri+14] of rate inversely proportional to the training set size. In this vein, the model capacity effectively grows dynamically with the number of iterations, preventing the first included orbits from already affecting all parameters. The ensemble effect of dropout might benefit the active learning process even further.

By and large, this work reveals two insights on a broader level: Firstly, deep learning can be considered a suitable technique for modeling the magnetodynamics of Mercury. Secondly, active learning serves not only for enhancing labeling efficiency but also for addressing data representativeness questions. We strongly encourage future work to continue and improve our study, taking note of the suggestions made above. The outcomes might become relevant for the upcoming Mercury mission *BepiColombo* [Ben+10] by ESA and JAXA, who dispatched two space probes to presumably arrive at Mercury in December 2025.

Appendix

Deep Learning

A

Not all computational tasks can be solved with traditional deterministic algorithms in a straightforward way. As an example, consider handwritten digit recognition. How should an exact algorithm for deciding the identity of a given digit look like? What would its rules be based on? Where would it put the boundary between a one and a seven? How would it handle different viewing angles, lighting conditions and all sorts of ugly handwriting? It remains entirely unclear how to explicitly instruct a computer to accomplish an ambitious task like this.

A possible workaround is to loosen the explicitness and determinism requirements, which leads us to the field of *supervised machine learning*: Let $\mathcal{X} \subseteq \mathbb{R}^n$ be the domain and $\mathcal{Y} \subseteq \mathbb{R}^d$ the co-domain of the idealized task $f : \mathcal{X} \to \mathcal{Y}$. Instead of solving it head-on with an algorithm, which would require its designer to possess universal knowledge of the underlying laws, we let the computer *learn* a procedure like children do – from a plethora of examples! To this end, we require a *dataset*

$$\mathcal{D} = \{(x, f(x)) \mid x \in \mathcal{X}'\} \subseteq \mathcal{X} \times \mathcal{Y} \qquad \mathcal{X}' \subseteq \mathcal{X} \text{ finite}$$

of known *samples* x and their corresponding *labels* $f(x)$. Using a *model* $\hat{f}_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$ depending on *trainable parameters* $\boldsymbol{\theta}$ from a parameter space $\Theta \subseteq \mathbb{R}^m$, our goal is then to *approximate* the *target function* f as close as possible, i.e., $\hat{f}_{\boldsymbol{\theta}} \approx f$.

In order to establish a complete machine learning system, we furthermore need

- A *loss* function $\mathcal{L}_{\mathcal{D}} : \Theta \to \mathbb{R}$ that measures the approximation quality of $\hat{f}_{\boldsymbol{\theta}}$,
- an *optimization* procedure that finds optimal values for the parameters $\boldsymbol{\theta}$,
- an *architecture* that specifies the internal structure of the model $\hat{f}_{\boldsymbol{\theta}}$, and
- meaningful *metrics* for evaluating the practical generalization capability of $\hat{f}_{\boldsymbol{\theta}}$.

We address each of these ingredients one by one. Section A.1 establishes the fundamentals of information theory to arrive at a bullet-proof loss function. Section A.2 then elaborates on how this loss can be used to build an algorithm that updates the parameters $\boldsymbol{\theta}$ toward better values. Section A.3 provides insight into the concrete architecture defining $\hat{f}_{\boldsymbol{\theta}}$ in modern deep learning models. Section A.4 completes the journey with an overview of the most important evaluation metrics in practice.

A.1 Information Theory

To derive a proper loss function for our machine learning system, we take a stochastic perspective: Let $(\Omega, \mathcal{A}, \mathbb{P})$ be a probability space and $(\mathbf{x}, \mathbf{y}) : \Omega \to \mathbb{R}^n \times \mathbb{R}^d$ a random vector describing the data distribution in $\mathcal{D}$. Then we can think of the model $\hat{f}_{\boldsymbol{\theta}}$ as realizing for each $\boldsymbol{x} \in \mathcal{X}$ a conditional distribution $\mathbb{P}[\widehat{\mathbf{y}}_{\boldsymbol{\theta}} = \cdot \mid \mathbf{x} = \boldsymbol{x}]$ over a random vector $\widehat{\mathbf{y}}_{\boldsymbol{\theta}} : \Omega \to \mathbb{R}^d$ depending on the model parameters. Optimally, the model's distribution should match the data distribution as close as possible, i.e., for all $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}$ we desire $\mathbb{P}[\widehat{\mathbf{y}}_{\boldsymbol{\theta}} = \boldsymbol{y} \mid \mathbf{x} = \boldsymbol{x}] \approx \mathbb{P}[\mathbf{y} = \boldsymbol{y} \mid \mathbf{x} = \boldsymbol{x}]$. One adequate measure of dissimilarity between the model and the data distribution stems from the field of information theory, which is mainly concerned with data transmission and encoding. First formalized by Claude Shannon [Sha48], it centers around quantifying the concept of *information*.

A.1.1 Measuring Uncertainty

Imagine a weather station that periodically transmits the current weather conditions over a digital communication channel. If the station happens to be in a region where it is sunny 99% of the time, and the chance of rain is only 1%, how surprised would you be when the station transmitted "Tomorrow will be a sunny day."? Not much. However, if the message were "Tomorrow it will rain.", you would. The less likely an event is, the more surprising it is and the more new information it yields. This consideration motivates a quantitative measure of information for events in random processes.

Definition 1 (Information amount)**.** Let $\mathbf{x} : \Omega \to \mathbb{R}^d$ be a finitely supported random vector. The *information amount* $I_{\mathbf{x}} : \operatorname{supp}(\mathbf{x}) \to \mathbb{R}$ with respect to $\mathbf{x}$ is defined[7] as

$$I_{\mathbf{x}}(\boldsymbol{x}) := -\log(\mathbb{P}[\mathbf{x} = \boldsymbol{x}]) = \log \frac{1}{\mathbb{P}[\mathbf{x} = \boldsymbol{x}]}.$$

While its inverse proportionality to an event's probability is clear from the discussion above, the appearance of a logarithm might be – well – surprising. We will later see that $\log$ earns its place for deeper mathematical reasons. For now, we motivate it with the weather station example: Suppose it is in a region where the sun shines $50\,\%$ of the time and chances of rain and snow are $25\,\%$, respectively. If we wanted to find an optimal binary encoding for the status message to be transmitted, coding theory suggests we choose `0` for "sunny", `10` for "rainy" and `11` for "snowy". Observe now that when substituting the binary logarithm in Definition 1 above, the information content of a weather condition equals precisely the number of bits we need for encoding it in the optimal code. So we may think of $I_{\mathbf{x}}(\boldsymbol{x})$ as the encoded length of $\boldsymbol{x}$ in an optimized code. In addition to this interpretation, the logarithm allows for a desirable algebraic property to hold.

[7] Note that $I_{\mathbf{x}}$ is well-defined due to the definition of $\operatorname{supp}(\mathbf{x}) := \{\boldsymbol{x} \in \mathbb{R}^d \mid \mathbb{P}[\mathbf{x} = \boldsymbol{x}] > 0\}$.

Lemma 1 (Independent Additivity)**.** For two independent discrete random variables $\mathrm{x}, \mathrm{y} : \Omega \to \mathbb{R}$, the following addition law holds:

$$I_{(\mathrm{x},\mathrm{y})}(x, y) = I_{\mathrm{x}}(x) + I_{\mathrm{y}}(y). \tag{A.1}$$

Proof. The law follows from the definition of stochastic independence:

$$\begin{aligned} I_{(\mathrm{x},\mathrm{y})}(x, y) &= -\log(\mathbb{P}[(\mathrm{x}, \mathrm{y}) = (x, y)]) \\ &= -\log(\mathbb{P}[\mathrm{x} = x, \mathrm{y} = y]) \\ &= -\log(\mathbb{P}[\mathrm{x} = x] \cdot \mathbb{P}[\mathrm{y} = y]) \\ &= -\log(\mathbb{P}[\mathrm{x} = x]) - \log(\mathbb{P}[\mathrm{y} = y]) \\ &= I_{\mathrm{x}}(x) + I_{\mathrm{y}}(y) \end{aligned}$$

□

In other words: Independent information sums up. This is exactly what we expect from an intuitive measure of information. So far, we are only able to quantify the information contained in a single event. A reasonable extension to entire distributions would be the *expected* information across all outcomes. And indeed, this is the central definition of information theory.

Definition 2 (Shannon Entropy[8])**.** Let $\mathbf{x} : \Omega \to \mathbb{R}^d$ be a finitely supported random vector. We define the *entropy* of $\mathbf{x}$ as

$$H(\mathbf{x}) \coloneqq \mathbb{E}[I_{\mathbf{x}}(\mathbf{x})] = -\sum_{\boldsymbol{x} \in \mathrm{supp}(\mathbf{x})} \mathbb{P}[\mathbf{x} = \boldsymbol{x}] \log(\mathbb{P}[\mathbf{x} = \boldsymbol{x}]).$$

Being the average amount of information, we can also interpret entropy as a measure of uncertainty in a distribution, i.e., how surprised we are on average when drawing from that distribution. Furthermore, one might think of it as the average message length in an optimal code for the outcomes. Despite the definition's conceptual simplicity, we should allow ourselves some illustrative examples.

Example 1 (Entropies of Common Distributions)**.**

(a) *Bernoulli:* The entropy of $\mathrm{x}_p \sim \mathrm{B}(1, p)$, for $p \in [0, 1]$ is called the *binary entropy function*:

$$H(\mathrm{x}_p) = -p \log(p) - (1 - p) \log(1 - p)$$

Figure A.1 plots this as a function of the success probability p.

(b) The entropy of the uniformly distributed variable $\mathrm{y}_n \sim \mathrm{U}(\{1, \ldots, n\})$ for $n \in \mathbb{N}$ is

$$H(\mathrm{y}_n) = -n \tfrac{1}{n} \log(\tfrac{1}{n}) = \log(n).$$

[8] From greek *en* ≅ in and *trope* ≅ turning, translatable as "self-transformation".

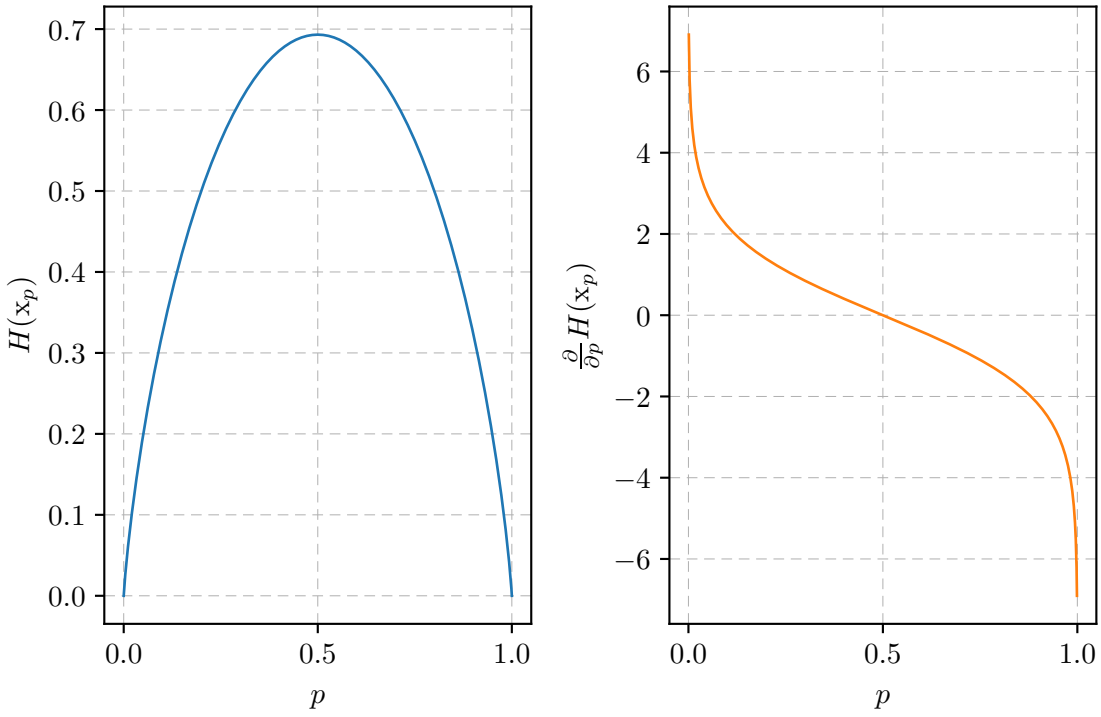


Figure A.1: *Left:* Entropy of Bernoulli variables with respect to success probability as in Example 1 *Right:* Its derivative approaching infinity at the boundaries.

The second example suggests that entropy might stand in a deeper relationship with uniform distributions. As we will see shortly, this is quite the case. Before that, we shall gather crucial properties of this fundamental information measure.

Theorem 1 (Properties of Entropy)**.** Let $\mathrm{x}, \mathrm{y} : \Omega \to \mathbb{R}$ be finitely supported random variables on a probability space $(\Omega, \mathcal{A}, \mathbb{P})$. We observe the following properties:

(a) *Non-Negativity:* $H(\mathrm{x}) \geq 0$.

(b) *Minima at Dirac Distributions:* $H(\mathrm{x})$ is minimal iff $\mathrm{x} \sim \delta(x)$ follows a Dirac distribution with $\mathrm{supp}(\mathrm{x}) = \{x\}$.

(c) *Maxima at Uniform Distributions:* $H(\mathrm{x})$ is maximal iff $\mathrm{x} \sim \mathrm{U}(\mathrm{supp}(\mathrm{x}))$ is uniformly distributed.

(d) *Independent Additivity:* If x and y are independent, then

$$H((\mathrm{x}, \mathrm{y})) = H(\mathrm{x}) + H(\mathrm{y}).$$

(e) *Uniform Monotonicity:* For uniformly distributed variables $\mathrm{x} \sim \mathrm{U}(\mathrm{supp}(\mathrm{x}))$ and $\mathrm{y} \sim \mathrm{U}(\mathrm{supp}(\mathrm{y}))$ with $|\mathrm{supp}(\mathrm{x})| \leq |\mathrm{supp}(\mathrm{y})|$, we have $H(\mathrm{x}) \leq H(\mathrm{y})$.

Proof.

"(a)" The non-negativity follows immediately from the definition:

$$H(\mathrm{x}) = -\sum_{x \in \mathrm{supp}(\mathrm{x})} \underbrace{\underbrace{\mathbb{P}[\mathrm{x}=x]}_{\geq 0} \underbrace{\log(\underbrace{\mathbb{P}[\mathrm{x}=x]}_{\leq 1})}_{\leq 0}}_{\leq 0} \geq 0$$

"(b)" According to (a), the minimum must be at least zero. Considering

$$H(\mathrm{x}) = -\sum_{x \in \mathrm{supp}(\mathrm{x})} \mathbb{P}[\mathrm{x}=x] \log(\mathbb{P}[\mathrm{x}=x]) \stackrel{!}{=} 0$$

requires by non-negativity that all summands vanish. Thus, for each $x \in \mathrm{supp}(\mathrm{x})$ either $\mathbb{P}[\mathrm{x}=x]=0$ or $\mathbb{P}[\mathrm{x}=x]=1$, so x is a Dirac variable.

"(c)" Write $\mathrm{supp}(\mathrm{x}) = \{x_1, \ldots, x_n\}$ and let $p_i \coloneqq \mathbb{P}[\mathrm{x}=x_i]$. To find maxima, we consider the Lagrange function

$$L(p_1, \ldots, p_n, \lambda) \coloneqq -\sum_{i=1}^{n} p_i \log(p_i) + \lambda \left(\sum_{i=1}^{n} p_i - 1 \right)$$

Annulling the partial derivatives with respect to the probabilities

$$\frac{\partial L}{\partial p_i}(p_1, \ldots, p_n, \lambda) = -1 - \log(p_i) + \lambda \stackrel{!}{=} 0$$

requires $p_i = e^{\lambda - 1}$ for each i, so all p_i are equal. Using this in the constraint

$$\frac{\partial L}{\partial \lambda}(p_1, \ldots, p_n, \lambda) = \sum_{i=1}^{n} p_i - 1 \stackrel{!}{=} 0$$

yields $np_i = 1$, so $p_i = \frac{1}{n}$ for all i. Sufficiency is left as an exercise for the reader.

"(d)" The additivity of entropy follows from Lemma 1 and the linearity of expectation:

$$\begin{aligned} H((\mathrm{x},\mathrm{y})) &= \mathbb{E}[I_{(\mathrm{x},\mathrm{y})}((\mathrm{x},\mathrm{y}))] \\ &\stackrel{A.1}{=} \mathbb{E}[I_{\mathrm{x}}(\mathrm{x}) + I_{\mathrm{y}}(\mathrm{y})] \\ &= \mathbb{E}[I_{\mathrm{x}}(\mathrm{x})] + \mathbb{E}[I_{\mathrm{y}}(\mathrm{y})] \\ &= H(\mathrm{x}) + H(\mathrm{y}) \end{aligned}$$

"(e)" The uniform monotonicity follows from that of logarithms and Example 1b:

$$H(\mathrm{x}) = \log|\mathrm{supp}(\mathrm{x})| \leq \log|\mathrm{supp}(\mathrm{y})| = H(\mathrm{y}) \qquad \square$$

So, entropy has several properties that render it a useful measure of information or uncertainty. But aren't there also other functions satisfying these properties? Preferably functions without a bizarre $\log$ expression in it? To answer this question, we approach it axiomatically. Since information theory deals with finitely supported random variables, we reformulate the problem without loss of generality as finding an uncertainty measure $H_n : \triangle^{n-1} \to \mathbb{R}$ on the *standard n-simplex*

$$\triangle^{n-1} := \{(p_1, p_2, \ldots, p_n) \in \mathbb{R}^n \mid \forall i : p_i \geq 0, \sum_{i=1}^{n} p_i = 1\} \subseteq [0,1]^n \tag{A.2}$$

with the following properties that we consider as fundamental:

(U1) *Continuity:* It does not appear reasonable that a measure of uncertainty change abruptly upon small changes to the distribution. Hence we demand:

$$H_n \text{ is continuous on the interior of } \triangle^{n-1}.$$

(U2) *Uniform Monotonicity:* If a uniform distribution has more outcomes than another uniform distribution, it effectively contains more choices and hence more uncertainty. Thus, H should be monotonic with respect to $n \in \mathbb{N}$ in this case:

$$H_n(\tfrac{1}{n}, \ldots, \tfrac{1}{n}) \leq H_{n+1}(\tfrac{1}{n+1}, \ldots, \tfrac{1}{n+1}).$$

(U3) *Decomposability:* For any partition $\{p_1, \ldots, p_n\} = G_1 \uplus \ldots \uplus G_k \subseteq [0,1]$ of the distribution $(p_1, \ldots, p_n) \in \triangle^{n-1}$ into disjoint groups $G_i = \{g_{i1}, \ldots, g_{in_i}\}$ with group probabilities $q_i := \sum_{j=1}^{n_i} g_{ij}$, we can decompose H as

$$H_n(p_1, \ldots, p_n) = H_k(q_1, \ldots, q_k) + q_i \sum_{i=1}^{k} H_{n_i}(\tfrac{g_{i1}}{q_i}, \ldots, \tfrac{g_{in_i}}{q_i}).$$

Figure A.2 illustrates the intuitive meaning: When dividing a distribution into two successive ones, where the second is *conditioned* on the first, the overall uncertainty value should be the sum of the individual distributions' uncertainties.

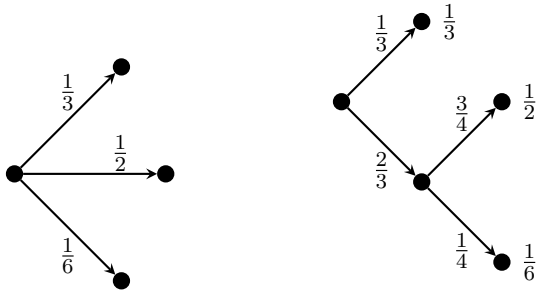


Figure A.2: Decomposition of a distribution over three probabilities.

Requiring only these three basic properties of an uncertainty measure allows us to answer the question. Remarkably, it turns out that the Shannon entropy has a unique selling point with the uncertainty properties up to a constant.

Theorem 2 (Uniqueness of Entropy)**.** If $H_n : \triangle^{n-1} \to \mathbb{R}$ satisfies the uncertainty axioms (U1), (U2) and (U3), then there is a non-negative constant $K \in \mathbb{R}_{\geq 0}$ such that

$$H_n(p_1, \ldots, p_n) = -K \sum_{i=1}^{n} p_i \log(p_i).$$

Proof. We follow Shannon's original proof [Sha48]. A more formal yet less clear proof is given by Chinčin [Chi70]. For brevity, set $\widetilde{H}(k) \coloneqq H_k(\frac{1}{k}, \ldots, \frac{1}{k})$. Suppose we have a uniform distribution of a^n outcomes for some $a, n \in \mathbb{N}_{\geq 1}$. Using (U3), we can decompose its uncertainty by grouping a outcomes together n times in a row:

$$\widetilde{H}(a^n) = \widetilde{H}(a^{n-1}) + \widetilde{H}(a) = \cdots = n\widetilde{H}(a).$$

This exponent-hopping should look familiar. Now let $b \in \mathbb{N}_{\geq 2}$ be arbitrary but fixed. For each choice of a and n, we find $m \in \mathbb{N}_{\geq 0}$ such that

$$b^m \leq a^n < b^{m+1}.$$

Logarithms are monotone and thus preserve the inequalities:

$$m\log(b) \leq n\log(a) < (m+1)\log(b) \implies \frac{m}{n} \leq \frac{\log(a)}{\log(b)} < \frac{m}{n} + \frac{1}{n} \implies \left|\frac{\log(a)}{\log(b)} - \frac{m}{n}\right| < \frac{1}{n}$$

But $\widetilde{H}$ is monotone just as well from (U2), so we get:

$$m\widetilde{H}(b) \leq n\widetilde{H}(a) < (m+1)\widetilde{H}(b) \implies \frac{m}{n} \leq \frac{\widetilde{H}(a)}{\widetilde{H}(b)} < \frac{m}{n} + \frac{1}{n} \implies \left|\frac{\widetilde{H}(a)}{\widetilde{H}(b)} - \frac{m}{n}\right| < \frac{1}{n}$$

Combining the two results and using the triangle inequality gives in the limit

$$\left|\frac{\widetilde{H}(a)}{\widetilde{H}(b)} - \frac{\log(a)}{\log(b)}\right| \leq \frac{2}{n} \xrightarrow{n\to\infty} 0 \implies \widetilde{H}(a) = \underbrace{\frac{\widetilde{H}(b)}{\log(b)}}_{=:K} \log(a) = -K \log\left(\frac{1}{a}\right)$$

with $K \geq 0$ to satisfy the monotonicity (U2). Now, consider a distribution of n outcomes with rational probabilities $(p_1, \ldots, p_n) \in \triangle^{n-1}$. By extending to a common denominator, we can assume without loss of generality that $p_i = \frac{n_i}{\Sigma n_i}$ with integers $n_i \in \mathbb{N}$ and $\Sigma n_i \coloneqq \sum_{i=1}^{n} n_i$. Using (U3) again, we can decompose the uncertainty by forming n groups of n_i equally probable outcomes with total probability p_i:

$$\widetilde{H}(\Sigma n_i) = H_n(p_1, \ldots, p_n) + \sum_{i=1}^{n} p_i \widetilde{H}(n_i)$$

In this vein, we reduce the general uncertainty to that of uniform distributions:

$$\begin{aligned} H_n(p_1,\dots,p_n) &= \widetilde{H}(\Sigma n_i) - \sum_{i=1}^{n} p_i \widetilde{H}(n_i) \\ &= K\log(\Sigma n_i) - K\sum_{i=1}^{n} p_i \log(n_i) \\ &= K\Big(\sum_{i=1}^{n} p_i \log(\Sigma n_i) - \sum_{i=1}^{n} p_i \log(n_i)\Big) \\ &= -K\sum_{i=1}^{n} p_i \log(\frac{n_i}{\Sigma n_i}) \\ &= -K\sum_{i=1}^{n} p_i \log(p_i) \end{aligned}$$

Finally, if $(p_1,\dots,p_n) \in \triangle^{n-1}$ are real-valued, the claim follows from the continuity requirement (U1). □

The theorem establishes entropy as somewhat the most mathematically sound measure of uncertainty in a distribution, with the only degree of freedom being the choice of the constant K. Returning to the optimal code analogy, the constant merely amounts to a choice of unit, since via $K = \frac{1}{\log(b)}$ for some $b \in \mathbb{R}_{>1}$ it may be integrated into the logarithmic expression, then using base b instead of e.

A.1.2 Comparing Distributions

While entropy is a powerful measure of information in a *single* random variable, it does not help us with our goal of determining the difference between *two* random variables. Or does it? When we allow the two random variable instances in Definition 2 to differ, we immediately obtain a measure for two variables.

Definition 3 (Cross-Entropy)**.** Let $\mathbf{x}, \widehat{\mathbf{x}} : \Omega \to \mathbb{R}^d$ be finitely supported random vectors with $\mathrm{supp}(\mathbf{x}) = \mathrm{supp}(\widehat{\mathbf{x}})$. The *cross-entropy* of $\mathbf{x}$ from $\widehat{\mathbf{x}}$ is

$$H(\mathbf{x} \parallel \widehat{\mathbf{x}}) \coloneqq \mathbb{E}[I_{\widehat{\mathbf{x}}}(\mathbf{x})] = -\sum_{\boldsymbol{x}\in\mathrm{supp}(\mathbf{x})} \mathbb{P}[\mathbf{x}=\boldsymbol{x}]\log(\mathbb{P}[\widehat{\mathbf{x}}=\boldsymbol{x}])$$

In the coding theory setting, we may interpret cross-entropy as the average message length required to identify outcomes of a distribution when borrowing a code optimized for *another* distribution. If this intuition is to work, surely cross-entropy from the variable optimized to the current variable must always be at least as large as the sole entropy of the current variable. Fortunately enough, it is!

Lemma 2 (Gibbs' Inequality)**.** For finitely supported random vectors $\mathbf{x}, \widehat{\mathbf{x}} : \Omega \to \mathbb{R}^d$ with $\operatorname{supp}(\mathbf{x}) = \operatorname{supp}(\widehat{\mathbf{x}})$, the following inequality holds:

$$H(\mathbf{x} \parallel \widehat{\mathbf{x}}) \geq H(\mathbf{x})$$

Furthermore, $H(\mathbf{x} \parallel \widehat{\mathbf{x}}) = H(\mathbf{x})$ if and only if $\mathbf{x} \sim \widehat{\mathbf{x}}$.

Proof. We first show that $\log(x) \leq x - 1$ holds for all $x \in \mathbb{R}_{>0}$. To this end, define

$$d : (0, \infty) \to \mathbb{R}, \quad d(x) = (x - 1) - \log(x).$$

This difference function is differentiable and

$$d'(x) = 1 - \frac{1}{x} = 0 \iff x = 1, \qquad d''(x) = \frac{1}{x^2} > 0$$

shows that it has a local minimum at $d(1) = 0$. Because of

$$\lim_{x \to 0^+} d(x) = \infty \qquad \text{and} \qquad \lim_{x \to \infty} d(x) = \infty$$

this is guaranteed to be the global minimum of d.

Now write $\operatorname{supp}(\mathbf{x}) = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ and set $p_i \coloneqq \mathbb{P}[\mathbf{x} = \boldsymbol{x}_i]$, $q_i \coloneqq \mathbb{P}[\widehat{\mathbf{x}} = \boldsymbol{x}_i]$. Then:

$$\begin{aligned}
H(\mathbf{x} \parallel \widehat{\mathbf{x}}) - H(\mathbf{x}) &= -\sum_{i=1}^{n} p_i \log(q_i) + \sum_{i=1}^{n} p_i \log(p_i) \\
&= -\sum_{i=1}^{n} p_i \log \frac{q_i}{p_i} \\
&\geq -\sum_{i=1}^{n} p_i \left(\frac{q_i}{p_i} - 1 \right) \\
&= -\sum_{i=1}^{n} (q_i - p_i) \\
&= \sum_{i=1}^{n} p_i - \sum_{i=1}^{n} q_i \\
&= 0
\end{aligned}$$

For equality, we need all $\frac{q_i}{p_i} = 1$ and hence $p_i = q_i$. □

Gibbs' Inequality confirms the intuitive notion of the difference between cross-entropy and entropy as *excess* entropy when abusing an optimized code for another distribution. Since this difference is always non-negative and zero only on distributional equality, it can servedi as dissimilarity measure between distributions. In fact, this is a widely used measure and sometimes used to define cross-entropy.

Definition 4 (Kullback-Leibler Divergence)**.** Let $\mathbf{x}, \widehat{\mathbf{x}} : \Omega \to \mathbb{R}^d$ be finitely supported random vectors with $\operatorname{supp}(\mathbf{x}) = \operatorname{supp}(\widehat{\mathbf{x}})$. The *Kullback-Leibler divergence* (*KLD*) of $\mathbf{x}$ from $\widehat{\mathbf{x}}$ is

$$D_{\mathrm{KL}}(\mathbf{x} \parallel \widehat{\mathbf{x}}) \coloneqq H(\mathbf{x} \parallel \widehat{\mathbf{x}}) - H(\mathbf{x}) = \mathbb{E}_{\boldsymbol{x} \sim \mathbf{x}} \left[\frac{\log(\mathbb{P}[\mathbf{x} = \boldsymbol{x}])}{\log(\mathbb{P}[\widehat{\mathbf{x}} = \boldsymbol{x}])} \right]$$

Note that due to the asymmetry in the definition of cross-entropy, also the KLD is in general not symmetric. Thus, it unfortunately does not fulfill all axioms of a metric. However, this is not an issue since our machine learning setting is asymmetric as well: We measure the excess entropy when utilizing the model distribution $\mathbb{P}[\widehat{\mathbf{y}}_{\boldsymbol{\theta}} = \cdot \mid \mathbf{x} = \boldsymbol{x}]$ for predicting the data distribution $\mathbb{P}[\mathbf{y} = \cdot \mid \mathbf{x} = \boldsymbol{x}]$. To this end, we require one last formality to account for the conditionality of the distributions.

Remark 1 (Conditional Variants of Information Measures)**.** Let $\mathbf{x} : \Omega \to \mathbb{R}^n$ and $\mathbf{y}, \widehat{\mathbf{y}} : \Omega \to \mathbb{R}^d$ be finitely supported random vectors with $\operatorname{supp}(\mathbf{y}) = \operatorname{supp}(\widehat{\mathbf{y}})$. The information measures discussed so far extend straighforwardly to conditional distributions by averaging over the prior:

(a) $H(\mathbf{y} \mid \mathbf{x}) \coloneqq \mathbb{E}[\mathbb{E}[I_{\mathbf{y}}(\mathbf{y}) \mid \mathbf{x}]] = -\sum_{\substack{\boldsymbol{x} \in \operatorname{supp}(\mathbf{x}) \\ \boldsymbol{y} \in \operatorname{supp}(\mathbf{y})}} \mathbb{P}[\mathbf{y} = \boldsymbol{x}, \mathbf{x} = \boldsymbol{x}] \log \mathbb{P}[\mathbf{y} = \boldsymbol{x} \mid \mathbf{x} = \boldsymbol{x}]$

(b) $H(\mathbf{y} \parallel \widehat{\mathbf{y}} \mid \mathbf{x}) \coloneqq \mathbb{E}[\mathbb{E}[I_{\widehat{\mathbf{y}}}(\mathbf{y}) \mid \mathbf{x}]] = -\sum_{\substack{\boldsymbol{x} \in \operatorname{supp}(\mathbf{x}) \\ \boldsymbol{y} \in \operatorname{supp}(\mathbf{y})}} \mathbb{P}[\mathbf{y} = \boldsymbol{x}, \mathbf{x} = \boldsymbol{x}] \log \mathbb{P}[\widehat{\mathbf{y}} = \boldsymbol{x} \mid \mathbf{x} = \boldsymbol{x}]$

(c) $D_{\mathrm{KL}}(\mathbf{y} \parallel \widehat{\mathbf{y}} \mid \mathbf{x}) \coloneqq H(\mathbf{y} \parallel \widehat{\mathbf{y}} \mid \mathbf{x}) - H(\mathbf{y} \mid \mathbf{x})$

Finally, the conditional KLD is the loss function we want to minimize. Since the entropy term does not depend on the parameters, minimizing conditional KLD in fact reduces to minimizing conditional cross-entropy:

$$\arg\min_{\boldsymbol{\theta} \in \Theta} D_{\mathrm{KL}}(\mathbf{y} \parallel \widehat{\mathbf{y}}_{\boldsymbol{\theta}} \mid \mathbf{x}) = \arg\min_{\boldsymbol{\theta} \in \Theta} H(\mathbf{y} \parallel \widehat{\mathbf{y}}_{\boldsymbol{\theta}} \mid \mathbf{x})$$

Thus, we might as well say that we employ cross-entropy as the loss function, although from a theoretical standpoint KLD is the true difference measure. Bearing in mind that the dataset $\mathcal{D}$ defines an empirical distribution with $\mathbb{P}[\mathbf{x} = \boldsymbol{x}, \mathbf{y} = \boldsymbol{y}] = \frac{1}{|\mathcal{D}|}$ for $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}$, we obtain the final, delightfully simple, loss formula:

$$\mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}) = -\frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}} \log(\mathbb{P}[\widehat{\mathbf{y}}_{\boldsymbol{\theta}} = \boldsymbol{y} \mid \mathbf{x} = \boldsymbol{x}]) \tag{A.3}$$

We are thus finally able to quantify the error of the machine learning model $\hat{f}_{\boldsymbol{\theta}}$ on the dataset $\mathcal{D}$ with the loss function $\mathcal{L}_{\mathcal{D}}$ depending on the model's parameters $\boldsymbol{\theta}$. Our next ambition will be to get this error as small as possible.

A.2 Optimization

Now that we have the loss function (A.3), we need a means to minimize it with respect to the parameters. A good starting point is the fact that at any local minimum $\boldsymbol{\theta}^* \in \Theta$ the loss must have zero gradient, i.e., satisfy $\nabla \mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}^*) = 0$. However, in general, it is impossible to solve this equation for $\boldsymbol{\theta}^*$ in closed form due to the complexity of the model $\hat{f}_\theta$ in modern architectures that define the probability terms. Therefore, we must resort to numerical methods for finding zeros of the gradient.

A.2.1 Gradient Descent

The most famous technique for finding zeros of a differentiable function is Newton's method. Unfortunately, it is not feasible for our use case since it involves calculating Jacobi matrices of the gradient, i.e., second derivatives of the loss function. For huge datasets, as they occur in practice, Newton's method thus imposes a significant computational burden that we aim to avoid. Instead, the preferred approach is to utilize only the first derivatives provided by the gradient and traverse the loss function "downhill". To formalize this, we require a measure of loss surface steepness at a point along arbitrary directions, not just the basis vectors as given by the partial derivatives.

Definition 5 (Directional Derivative)**.** Let $D \subseteq \mathbb{R}^n$ be open. The *directional derivative* of a function $f : D \to \mathbb{R}$ in the direction of $\boldsymbol{v} \in \mathbb{R}^n$ at point $\boldsymbol{x} \in D$ is given by

$$D_{\boldsymbol{v}} f(x) \coloneqq \lim_{h \to 0} \frac{f(\boldsymbol{x} + h\boldsymbol{v}) - f(\boldsymbol{x})}{h}$$

if this limit exists. In this case f is said to be *directionally differentiable*.

The directional derivative is a straightforward generalization of the partial derivatives. Since the limit expression is unpleasant to work with, we seek a better way to calculate the directional derivative. The following lemma shows that it can be reduced to an inner product of the direction vector with the gradient! Interestingly, this again means that directional derivatives are not really more general than gradients but just linear combinations of the partial derivatives according to the direction coordinates.

Lemma 3 (Directional Derivative Inner Product)**.** Let $D \subseteq \mathbb{R}^n$ be open. If $f : D \to \mathbb{R}$ is differentiable at $\boldsymbol{x} \in D$, then for any $\boldsymbol{v} \in \mathbb{R}^n$

$$D_{\boldsymbol{v}} f(x) = \boldsymbol{v}^{\mathrm{T}} \nabla f(x).$$

Proof. The set $\widetilde{D} \coloneqq \{h \in \mathbb{R} \mid \boldsymbol{x} + h\boldsymbol{v} \in D\}$ is open as preimage of a linear function. Defining on it $g : \widetilde{D} \to \mathbb{R}$ as $g(h) = f(\boldsymbol{x} + h\boldsymbol{v})$, we get using the chain rule:

$$D_{\boldsymbol{v}} f(\boldsymbol{x}) = \lim_{h \to 0} \frac{g(h) - g(0)}{h} = g'(0) = \boldsymbol{v}^{\mathrm{T}} \nabla f(\boldsymbol{x}) \qquad \square$$

At first glance, it does not seem obvious how to find a specific direction along which the gradient is smallest. With the more tractable expression for the directional derivative given by Lemma 3, we can find it by solving an optimization problem for our optimization problem. It turns out to be a delightfully simple direction.

Theorem 3 (Steepest Ascent)**.** Let $D \subseteq \mathbb{R}^n$ be open and $f : D \to \mathbb{R}$ differentiable. The gradient of f at point $\boldsymbol{x} \in D$ always points in a direction of steepest ascent:

$$\nabla f(\boldsymbol{x}) \in \operatorname*{arg\,max}_{\boldsymbol{v} \in \mathbb{R}^n, \|\boldsymbol{v}\|=1} D_{\boldsymbol{v}} f(\boldsymbol{x})$$

Likewise, the negative gradient points in a direction of steepest descent:

$$-\nabla f(\boldsymbol{x}) \in \operatorname*{arg\,min}_{\boldsymbol{v} \in \mathbb{R}^n, \|\boldsymbol{v}\|=1} D_{\boldsymbol{v}} f(\boldsymbol{x})$$

Proof. Using Lemma 3 and elementary analytic geometry, we obtain:

$$\begin{aligned}
\max_{\boldsymbol{v} \in \mathbb{R}^n, \|\boldsymbol{v}\|=1} D_{\boldsymbol{v}} f(\boldsymbol{x}) &= \max_{\boldsymbol{v} \in \mathbb{R}^n, \|\boldsymbol{v}\|=1} \boldsymbol{v}^{\mathrm{T}} \nabla f(\boldsymbol{x}) \\
&= \max_{\boldsymbol{v} \in \mathbb{R}^n, \|\boldsymbol{v}\|=1} \|\boldsymbol{v}\| \left\|\nabla f(\boldsymbol{x})\right\| \cos(\measuredangle(\boldsymbol{v}, \nabla f(\boldsymbol{x})) \\
&= \left\|\nabla f(\boldsymbol{x})\right\| \max_{\boldsymbol{v} \in \mathbb{R}^n, \|\boldsymbol{v}\|=1} \cos(\measuredangle(\boldsymbol{v}, \nabla f(\boldsymbol{x})) \\
&= \frac{1}{\left\|\nabla f(\boldsymbol{x})\right\|} \left\|\nabla f(\boldsymbol{x})\right\|^2 \\
&= \frac{\nabla f(\boldsymbol{x})^{\mathrm{T}}}{\left\|\nabla f(\boldsymbol{x})\right\|} \nabla f(\boldsymbol{x}) \\
&= D_{\frac{\nabla f(\boldsymbol{x})}{\|\nabla f(\boldsymbol{x})\|}} \nabla f(\boldsymbol{x})
\end{aligned}$$

Due to the nature of cosine, a direction of steepest *de*scent is given by $-\nabla f(\boldsymbol{x})$. □

Motivated by Theorem 3, Cauchy [Cau47] invented a straightforward algorithm termed *gradient descent* working its way towards a potential minimum: Start with some initial guess of the parameters and then repeatedly perform a step in the direction of the negative gradient. Since derivatives are a local phenomenon, we have no idea of *how far* we should go. Hence, the algorithm depends on a *hyperparameter* which determines the step size. It is commonly dubbed the *learning rate*. While machine learning models depend on many hyperparameters, the learning rate arguably constitutes the most important one [GBC16]. The algorithm repeats making small steps in promising directions until the gradient is zero – or rather almost. Recall that floating-point arithmetic in computers is not exact for very small values, so in practice we content ourselves as soon as the gradient is close enough to zero. For simplicity, though, Algorithm A.1 only gives a conceptual sketch of gradient descent neglecting numerical intricacies.

```
/* approximates a potential minimum of 𝓛_𝒟    */
GD(α ∈ ℝ_{>0} : learning rate, θ ∈ Θ : initial parameters):
1   while ∇𝓛_𝒟(θ) ≠ 0 :
2       θ := θ − α∇𝓛_𝒟(θ)  // apply parameter
            update
```

Algorithm A.1: Gradient Descent

This "vanilla" gradient descent has a significant drawback in practice: Computing the loss function and its gradient over the entire dataset is prohibitively expensive. To mitigate computational complexity and enable practicality, we hence evaluate the loss only on a small random sample of the dataset called a *minibatch*. Clearly, resorting to an estimated gradient instead of the real one sacrifices theoretical elegance as well as determinism, but limited computational resources leave no other choice. The modified procedure known as *stochastic gradient descent* (*SGD*) dates back to Kiefer and Wolfowitz [KW52] and is outlined in Algorithm A.2. In each iteration of SGD known as *epoch*, we first split the dataset into random partitions of a certain maximum size. To ensure an efficient memory layout, one typically chooses a power of two for this *batch size*, our next hyperparameter. [GBC16] Then, we perform gradient descent update steps for each of these partitions.

```
/* approximates a potential minimum of 𝓛_𝒟 using minibatches  */
SGD(α ∈ ℝ_{>0} : learning rate, β ∈ ℕ_{≥1} : batch size, θ ∈ Θ : initial parameters):
1   do
2       𝒟_1, ..., 𝒟_n := batchify(𝒟, β)  // split 𝒟 = 𝒟_1 ⊎ ... ⊎ 𝒟_n
3       for i := 1 to n do
4           θ := θ − α∇𝓛_{𝒟_i}(θ)  // apply parameter update
5   while ¬stopping_criterion(θ)
```

Algorithm A.2: Stochastic Gradient Descent

Note the replacement of the naive exit condition depending on the gradient value with a general one depending on the parameters. Since the gradient updates are now based on constantly changing subsets of the dataset, there is no guarantee for it ever to step sufficiently close to zero. Instead, one often employs a technique called *early stopping*: One tracks the loss value on a separate *evaluation set* disjoint from the *training set* that gradient descent optimizes on. This yields an unbiased estimate of the model's real-world performance, and once it does not continue to improve, we may stop iterating.

Clearly, the descent algorithm depends heavily on the choice of its hyperparameters, and these are not independent: Using a smaller batch size means more parameter updates, and hence the learning rate should be reduced accordingly. However, to get a stable approximation of the true gradient, we usually want the batch size to be as large as the computing hardware allows. Thus, we can consider its value fixed given the execution environment, such that just the learning rate needs further examination.

Overall, two naiveties remain in the update step of Algorithm A.2:

1. *Greedy direction choice:* Blindly following the negative gradient at all costs is not a good idea for functions that are beyond simple. It might lead the descent algorithm to take unnecessary sharp turns despite moving generally in the right direction. Such a behavior significantly increases the time until SGD converges.

2. *Inadaptive learning rate:* Using the same, constant learning rate for all update steps is unjustified. The descent algorithm might traverse regions in parameter space that are very flat at one time, requiring a large learning rate, and regions with steep cliffs at another time, where a lower learning rate is imperative.

These issues are addressed by several extensions to SGD. We describe the three most groundbreaking of them conceptually to highlight their central ideas and refer to the literature for further details:

Momentum In order to alleviate the aggressive turns caused by stubbornly stepping downhill, the *Momentum* extension [Pol64] introduces the homonymous physical concept, or simply velocity[9], into the descent procedure. Starting with a zero value, the algorithm updates velocity in each step as an exponential moving average between the past velocity and the direction according to naive SGD. Then we perform the parameter update in the averaged direction specified by the velocity. In this manner, inertia is added to the process, and thus the traversal through parameter space is smoothened out. Effectively, we can regard the moving average as an estimation of the first statistical moment of the partial derivative distribution.

RMSProp To make the learning rate sensitive to the current gradient value, the *Root Mean Squared Propagation* (*RMSProp*) extension [Hin12] scales it for each parameter individually by a factor depending on that parameter's partial derivative history. In particular, the optimizer tracks an exponential moving average of the square of the partial derivative like Momentum does with the derivatives themselves for velocity. The learning rate is then divided by the square root of this average, effectively being an estimation of the second (uncentered) statistical moment of the partial derivative distribution.

Adam Killing two birds with one stone, the *Adaptive Moments* (*Adam*) [KB15] optimizer essentially combines Momentum with RMSProp. Consequently, it tracks two moving averages for the first and second moment of the partial derivative distribution, respectively. These are then used to adapt the learning rate, hence the name. However, one component makes Adam more than just the union of the other two extensions: Adam adds bias correction terms to the updates to account for the implicit bias resulting from initializing the moving averages with zero.

[9] Since physical momentum is mass times velocity, they only differ by a constant factor.

Other noteworthy optimizers are *Nesterov Accelerated Gradient Descent* (*NAG*) [Nes83], *Averaged SGD* (*ASGD*) [PJ92], *Adaptive Gradient Descent* (*AdaGrad*) [DHS11], *Nesterov Accelerated Adam* (*NAdam*) [Doz16] and *Rectified Adam* (*RAdam*) [Liu+20]. From a theoretical standpoint, an optimizer addressing both consistent direction and adaptive learning rate like Adam would be preferable. However, in practice this conceptual advantage does not seem to play out. In some cases, adaptive optimizers even underperform plain stochastic gradient descent [Wil+17]. Hence, it is still required to experiment with different versions of SGD, the choice of which being yet another hyperparameter.

Any optimizer provides a way to find *potential* local minima of the objective function. Since zero gradient is only a necessary condition also fulfilled by, e.g., saddle points or even maxima, we cannot be sure that the point gradient descent converges to is actually a local minimum. However, unless the algorithm happens to start exactly at a maximum, it will never converge to one since every direction points downwards[10]. With saddle points, this is more complicated, but it has been shown that gradient descent will almost surely overcome saddle points [Lee+16]. Saddle point evasion is further leveraged by the noise in the gradient estimation introduced by using minibatches [Jin+17].

In summary, gradient descent practically finds a local minimum. But that is it, a *local* one. Unless the optimization target is a strictly convex function, there are no guarantees for gradient descent to converge to a *global* minimum. A possible remedy is to execute gradient descent repeatedly with different, usually randomly initialized, starting values for the parameters $\boldsymbol{\theta}$. Then one can at least choose the best from several proposed local minima that hopefully is close enough to the value of a genuine global minimum.

A.2.2 Backpropagation

The only missing piece left in the optimization procedure is a way actually to compute the gradient $\nabla \mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta})$. For evaluation on a real machine, the loss function $\mathcal{L}_{\mathcal{D}}$ has to be a composition of elementary operations that we know the gradients of. Thus, it can be thought of as a directed graph of these operations.

Definition 6 (Computational Graph)**.** A *computational graph* $G := (V, E)$ is a directed acyclic graph (DAG), consisting of

- a set $V = \{(x_1, f_1), \ldots, (x_n, f_n)\}$ of symbolic *variables* x_i with associated elementary *operations* $f_i : \mathbb{R}^{n_i} \to \mathbb{R}$ that are differentiable, and
- a set $E \subseteq \{(i, j) \in \{1, \ldots, n\}^2 \mid i < j\}$ of directed edges such that for each $(x_i, f_i) \in V$ the number of parents matches the input dimension:

$$\big|\{(x_j, f_j) \in V \mid (j, i) \in E\}\big| = n_i.$$

[10] Unless the function is locally constant, in which case we already are at a local minimum.

Illustrative examples for computational graphs are given in Figure A.3. The variables in such a graph represent the objects we can calculate with, in our case scalars of real numbers. However, Definition 6 easily covers vectors, matrices, and higher dimensional tensors, too: Just group individual scalars to form the desired structures and alter the operations accordingly.

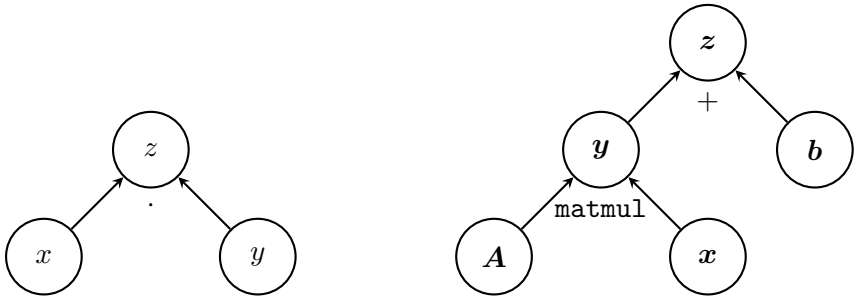


Figure A.3: *Left:* Computational graph for the multiplication $z = xy$. *Right:* Computational graph for the affine transformation $\boldsymbol{z} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$.

The DAG requirement for a computational graph is essential for being able actually to compute the function. This *forward pass* or *forthpropagation* works as follows: Since $\mathcal{G}$ has a topological ordering, we know there is a $k \in \mathbb{N}$ such that the first k vertices of the graph have no incoming edges. Further, suppose we are given real numbers $a_1, \ldots, a_k \in \mathbb{R}$ as value assignments for those vertices. For each vertex further down the graph, we simply collect the values of all direct ancestors, which are guaranteed to already having been calculated as $\mathcal{G}$ is acyclic. Then we perform the vertex operation on the gathered values, resulting in a value assignment for this vertex. We repeat this process until the final vertex of the topological ordering is reached as outlined in Algorithm A.3.

```
/* successively evaluates the function represented by
   the computational graph                              */
fprop(G : computational graph, a_1, ..., a_k ∈ ℝ : input values):
    (x_1, ..., x_k) := (a_1, ..., a_k)  // initialize input variables
    for i = k + 1 to n do
        x_i := f_i(x_j | (j, i) ∈ E)  // apply operation to parents
    return x_n
```

Algorithm A.3: Forward Propagation in Computational Graphs.

However, we do not only want to evaluate the function itself; we also need its gradient! More specifically, we desire the partial derivatives $\frac{\partial f_n}{\partial x_i}(x_n)$ at least for the first k variables. The path from a variable x_i to the output vertex x_n is essentially a function in one dimension. However, following the graph's structure, we can split it in two parts: One function that computes the immediate children of x_i, considering the other input to the child as fixed parameters. And one function that continues from the children through the rest of the graph.

Lemma 4 (Inflated Composition Derivative)**.** Let $f : \mathbb{R} \to \mathbb{R}^d$ and $g : \mathbb{R}^d \to \mathbb{R}$ be differentiable. Then the derivative of $h := g \circ f$ is given by

$$h' = \sum_{i=1}^{d} \frac{\partial g}{\partial x_i} \cdot \frac{\mathrm{d}f_i}{\mathrm{d}x}. \tag{A.4}$$

Proof. This is a special case of the chain rule: For $x \in \mathbb{R}$, we have

$$h'(x) = \nabla g(f(x)) f'(x) = \begin{bmatrix} \frac{\partial g}{\partial x_1}(f(x)) \cdots \frac{\partial g}{\partial x_d}(f(x)) \end{bmatrix} \begin{bmatrix} f_1'(x) \\ \vdots \\ f_d'(x) \end{bmatrix} = \sum_{i=1}^{d} \frac{\partial g}{\partial x_i}(f(x)) \frac{\mathrm{d}f_i}{\mathrm{d}x}(x)$$

where f_i shall denote the i-th component function of f. □

The simple differentiation rule from Lemma 4 forms the heart of the *backward pass* or *backpropagation* algorithm [RHW86] that follows the forward pass. Starting from the output, as opposed to the forward pass, we calculate partial derivatives with respect to the immediate parents in the graph, who combine the incoming gradients according to (A.4). It is fairly common to say that the gradients *flow* backward through the computational graph, which also explains the name of Algorithm A.4.

```
/* calculates partial derivatives of the last vertex
   with respect to all vertices                        */
bprop(G : computational graph):
    grads[x^(n)] := 1  // derivative w.r.t. oneself is 1
    for i = n − 1 to 1 do
        grads[x_i] := Σ_{(i,j)∈E} grads[x_j] · ∂f_j/∂x_i (x_j)
    return grads[x_1, ..., x_k]
```

Algorithm A.4: Backward Propagation in Computational Graphs

It is worth noting that backpropagation follows the principle of *dynamic programming*: As soon as the gradient of the output with respect to a variable has been computed, it saves the result in a data structure so it can be reused. Furthermore, by performing one forward pass through the graph once in the beginning, the values for intermediate variables do not have to be recalculated. Trading memory for time, this avoids redundantly computing the same expressions repeatedly.

To conclude, with gradient descent, we now have a means to optimize the loss $\mathcal{L}_\mathcal{D}$ with respect to the model parameters $\boldsymbol{\theta}$. Throughout this process, we repeatedly evaluate the loss function for the current parameters by forthpropagation, determine the gradient by backpropagation and ultimately update the parameters according to the specific optimizer version. In this vein, the model can be fit to the data distribution.

A.3 Neural Networks

With a loss function and an optimization strategy for our machine learning model $\hat{f}_{\boldsymbol{\theta}}$ at hand, it remains to specify how the model itself looks like. More precisely, we need to define how the parameters $\boldsymbol{\theta}$ are interweaved with the input $\boldsymbol{x}$ to yield a result. Over the last decade, a particular kind of model called *artificial neural network* has entered the center of machine learning research. As the name suggests, it aims to imitate the mechanics going on in a human brain.

A.3.1 Multi-Layer Perceptrons

The elementary building block for artificial neural networks is, of course, an artificial neuron. Human neurons, as depicted in Figure A.4, receive inputs from multiple so-called dendrites, process these in the cell body resulting in a binary *activation* that in the positive case makes the neuron fire an output along its axon to subsequent neurons. Inspired from their biological counterparts, artificial neurons mimic this behavior by modeling the cell body processing as a parameterized transformation of the inputs.

Definition 7 (Artificial Neuron)**.** An *artificial neuron* with *weights* $\boldsymbol{w} \in \mathbb{R}^n$, *bias* $b \in \mathbb{R}$ and *activation function* $\sigma : \mathbb{R} \to \mathbb{R}$ is a function $\nu_{\boldsymbol{w},b,\sigma} : \mathbb{R}^n \to \mathbb{R}$ given by

$$\boldsymbol{x} \mapsto \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b) = \sigma\Big(\sum_{i=1}^{n} w_i x_i + b\Big).$$

Importantly, the learnable parameters of an artificial neuron are its weights and bias. They are combined with the neuron's input through a rather simple affine transformation mainly for numerical reasons. Linear algebra operations are implemented very efficiently on current computer hardware, especially graphics processing units. To compensate for their lack of expressiveness, the activation function allows for an arbitrarily complex transformation of the result. We will shortly consider good choices for activation functions in detail.

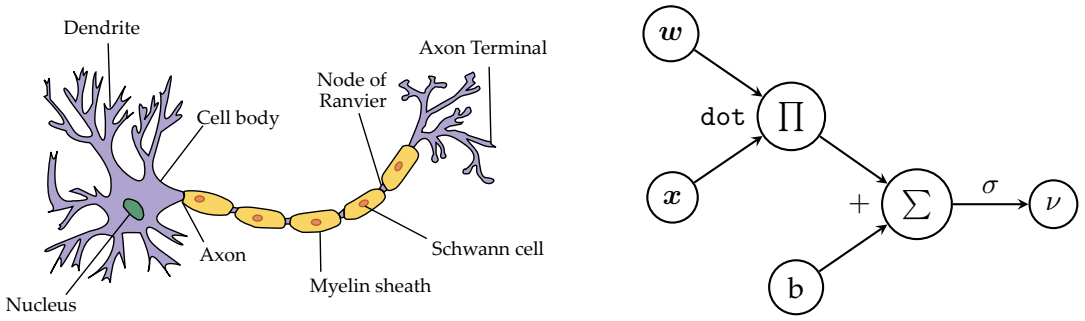


Figure A.4: *Left:* Biological model of a real neuron, adapted from Wikimedia [Wik19]. *Right:* Computational graph of an artificial neuron.

A single neuron cannot be expected to learn complicated patterns from the dataset. Like in the human brain, intelligence arises from interconnecting *many* neurons to form a *network*. In principle, any connected topology of artificial neurons can be regarded as an artificial neural network. However, for practical reasons, certain standard structures have evolved that can be easily implemented. The major concept in building neural network architectures is that of a *layer* of neurons that process the same inputs in parallel to concentrate on different aspects of the task at hand. The simplest possible layer is obtained by connecting all the neurons of that layer to all incoming inputs.

Definition 8 (Dense Layer)**.** A *dense layer* with *input size* $n \in \mathbb{N}$ and *output size* $d \in \mathbb{N}$, *weights* $\boldsymbol{W} = (\boldsymbol{w}_1, \dots, \boldsymbol{w}_d)^{\mathrm{T}} \in \mathbb{R}^{d \times n}$, biases $\boldsymbol{b} = (b_1, \dots, b_d)^{\mathrm{T}} \in \mathbb{R}^d$ and *activation function* $\sigma : \mathbb{R} \to \mathbb{R}$ is a collection of d single neurons $\nu_{\boldsymbol{w}_i, b_i, \sigma} : \mathbb{R}^n \to \mathbb{R}$:

$$\mathrm{dense}_{\boldsymbol{W},\boldsymbol{b},\sigma}(\boldsymbol{x}) \coloneqq \sigma(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) = \sigma\left(\begin{bmatrix} \boldsymbol{w}_1^{\mathrm{T}} \\ \vdots \\ \boldsymbol{w}_d^{\mathrm{T}} \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} b_1 \\ \vdots \\ b_d \end{bmatrix}\right) = \begin{bmatrix} \nu_{\boldsymbol{w}_1, b_1, \sigma}(\boldsymbol{x}) \\ \vdots \\ \nu_{\boldsymbol{w}_d, b_d, \sigma}(\boldsymbol{x}) \end{bmatrix}$$

Definition 8 demonstrates the big advantage of using ordinary affine transformations: The parameters can be straighforwardly stuffed into higher-dimensional tensors, here a matrix of weights and a vector of biases. In batched optimization, this scalability allows to arrange the inputs of a batch in a matrix $\boldsymbol{X} \in \mathbb{R}^{n \times b}$ and calculate the output of a dense layer as simply as $\sigma(\boldsymbol{W}\boldsymbol{X} + \boldsymbol{b})$, with $\boldsymbol{b}$ added column-wise and σ applied element-wise.

Stacking multiple dense layers on top of each other allows neurons to depend on the work of other neurons and yields the most basic neural network architecture. It is named a *Multi-Layer Perceptron* (*MLP*) after the historical Perceptron model [Ros58], a special kind of artificial neuron. An MLP consists of the *input layer* and *output layer* with an arbitrary number of *hidden layers* in between, as exemplified in Figure A.5.

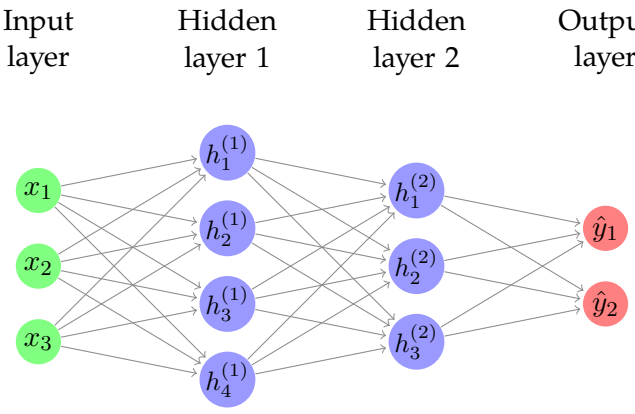


Figure A.5: Example of a multi-layer perceptron. Note that this is no computational graph, since weights and biases are missing and the operations are not specified.

A.3.2 Activation Functions

The expressive power of neural networks depends solely on the choice of activation function: Suppose we took an affine transformation for activation just like we did for the input processing. Then Linear Algebra tells us that the resulting neuron as a composition of affine transformations would itself again be such a one.

Lemma 5 (Affine closure)**.** If $f : \mathbb{R}^n \to \mathbb{R}^m$ and $g : \mathbb{R}^m \to \mathbb{R}^d$ are affine, so is $g \circ f$.

Proof. Write $f(\boldsymbol{x}) = \boldsymbol{A}_1\boldsymbol{x} + \boldsymbol{b}_1$ with $\boldsymbol{A}_1 \in \mathbb{R}^{m\times n}, \boldsymbol{b}_1 \in \mathbb{R}^m$ and $g(\boldsymbol{y}) = \boldsymbol{A}_2\boldsymbol{y} + \boldsymbol{b}_2$ with $\boldsymbol{A}_2 \in \mathbb{R}^{d\times m}, \boldsymbol{b}_2 \in \mathbb{R}^d$. Then

$$g(f(\boldsymbol{x})) = \boldsymbol{A}_2(\boldsymbol{A}_1\boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2 = \underbrace{(\boldsymbol{A}_2\boldsymbol{A}_1)}_{=:\boldsymbol{A}}\boldsymbol{x} + \underbrace{(\boldsymbol{A}_2\boldsymbol{b}_1 + \boldsymbol{b}_2)}_{=:\boldsymbol{b}}$$

shows that $(g \circ f)(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$ is affine. □

In fact, affine transformations even form a monoid with respect to function composition. Stacking multiple of those neurons thus would not expand expressivity either since the entire model collapses into a single affine transformation. This would render the neural network learning nothing else than overly complicated linear regression, which is not particularly expressive. For this reason, it is imperative to choose an activation function that is *non-affine*.

If we wanted to model biological neurons as close as possible, we would intuitively choose a function that behaves as follows: Starting from negative infinity, it has the constant value zero, meaning no activation of the neuron. At a certain threshold then, it suddenly jumps to a positive value, as shown in Figure A.6(a), and stays constant from then onwards. Such a function was used historically with Perceptrons [Ros58].

Definition 9 (Heaviside Step Function)**.** The *Heaviside step function* is given for $x \in \mathbb{R}$ by

$$\text{Heaviside}(x) := \mathbb{1}_{(0,\infty)} = \begin{cases} 1 & \text{for } x > 0 \\ 0 & \text{for } x \leq 0 \end{cases}$$

Although the Heaviside function would be the most plausible choice, it has two fatal deficiencies. Recall that we intend to optimize the model's parameters with gradient descent. Then firstly, Heaviside is not differentiable in zero[11]. For gradient descent to work, we need the entire model and hence all its components to be differentiable everywhere. Secondly, even if we somehow fixed the first problem with an artificial derivative value, the derivative would be zero almost everywhere and thus the whole gradient for this neuron's parameters, annihilating all optimization endeavors.

[11] Or the derivative is infinity if we consider the extended real number line.

The observations so far show that a good activation function has to be at least *non-affine*, *differentiable* and optimally *non-constant* everywhere. To achieve this, we could "smoothen out" the Heaviside function so that it fulfills these properties. Indeed, there is a function arising from the study of dynamical systems that does just this.

Definition 10 (Logistic Function)**.** The *logistic* function is given for $x \in \mathbb{R}$ by

$$\operatorname{expit}(x) \coloneqq \frac{1}{1+e^{-x}} \tag{A.5}$$

Remark 2 (Properties of the logistic function)**.** Since the denominator in (A.5) is always greater than 1, the logistic function takes only values in $(0,1)$ and the limits are

$$\lim_{x\to-\infty} \operatorname{expit}(x) = \lim_{x\to-\infty} \frac{1}{1+\underbrace{e^{-x}}_{\to\infty}} = 0, \qquad \lim_{x\to\infty} \operatorname{expit}(x) = \lim_{x\to-\infty} \frac{1}{1+\underbrace{e^{-x}}_{\to 0}} = 1.$$

Furthermore, as a composition of differentiable functions, expit itself is differentiable with its derivative given by

$$\operatorname{expit}'(x) = \frac{d}{dx}(1+e^{-x})^{-1} = \frac{d}{dx}(1+e^{-x})(1+e^{-x})^{-2} = \underbrace{\operatorname{expit}(x)}_{>0}\underbrace{(1-\operatorname{expit}(x))}_{>0} > 0$$

and therefore, the logistic function increases strictly monotonically from 0 to 1.

The derived properties can be seen clearly in Figure A.6(b). Towards infinity, the logistic function becomes flatter and flatter and, therefore, more insensitive to small changes. This is commonly referred to as *saturation*. Functions behaving like this are commonly called *sigmoids*. Another famous sigmoid is closely related to the logistic function and mainly differs from the latter in that its activation can become negative. It is well-known from hyperbolic geometry.

Definition 11 (Tangens Hyperbolicus)**.** The *hyperbolic tangent* is given for $x \in \mathbb{R}$ in analogy to the ordinary tangent via

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

Remark 3 (expit and $\tanh$)**.** The tangens hyperbolicus is nothing but a stretched and shifted logistic function:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{1-e^{-2x}}{1+e^{-2x}} = \frac{2-(1+e^{-2x})}{1+e^{-2x}} = \frac{2}{1+e^{-2x}} - 1 = 2\operatorname{expit}(2x) - 1$$

Specifically, this proves that its image is $\tanh(\mathbb{R}) = 2\cdot(0,1) - 1 = (-1,1)$, as Figure A.6(c) visually suggests.

Although expit and $\tanh$ satisfy our necessary conditions to a proper activation function, one of their beautiful properties turns out as a major drawback: According to Remark 2, the logistic function is contained in its own derivative, and due to Remark 3 in the derivative of the hyperbolic tangent as well. In a multi-layer network, the gradient will, during backpropagation, flow through successive exponential terms, which can cause the gradient to become very small or large. As a result, the gradient descent update step size for earlier layers is either insignificant or overshoots. This is known as the *exploding gradient problem* and *vanishing gradient problem*, respectively, and applies to all sigmoidal functions [LeC+98].

To stabilize the gradient descent optimization for arbitrarily deep neural networks, we must discard sigmoids altogether. Their use as activation functions for hidden neurons is nowadays discouraged [GBC16]. Since exponential terms are problematic, we resort to expressions that do not blow up upon composition. In practice, a deceptively simple activation function using only linear expressions turned out highly successful [NH10; KSH12].

Definition 12 (Rectified Linear Unit)**.** The *rectified linear unit* (*ReLU*) activation function is given for $x \in \mathbb{R}$ by

$$\mathrm{ReLU}(x) \coloneqq \max\{0, x\} = \int_0^x \mathrm{Heaviside}(t)\,\mathrm{d}t$$

Remark 4 (Differentiability of ReLU)**.** ReLU is not differentiable in the origin since

$$\lim_{x \nearrow 0} \mathrm{ReLU}'(x) = 0 \neq 1 = \lim_{x \searrow 0} \mathrm{ReLU}'(x).$$

Usually one defines $\mathrm{ReLU}'(0) \coloneqq 0$, so we consistently have $\mathrm{ReLU}' = \mathrm{Heaviside}$.

A plot of ReLU is given in Figure A.6(d). Its advantages are twofold: Firstly, the computational effort for a forward pass is minimal, being a simple value comparison. Secondly, due to the linear nature gradients will keep their magnitude during a backward pass. Variants and generalizations of ReLU include *absolute value rectification* [Jar+09], *Leaky ReLU* (*LReLU*) [Maa13] *parametric ReLU* (*PReLU*) [He+15b], *Gaussian Error Linear Unit* (*GELU*) [HG16] and *maxout units* [Goo+13].

Having discussed several activation functions, we wonder if they indeed make a neural network more expressive than linear regression. Fortunately, according to the *universal approximation theorem*, feedforward networks with certain types of bounded activation functions like the logistic function or the hyperbolic tangent are capable of approximating *any Borel-measurable function* between finite-dimensional spaces [Cyb89; HSW89]. However, this result does not apply to the rectified linear unit, which is unbounded in the positive direction. It was later generalized to locally bounded piecewise continuous activation functions [Les+93], a category which also ReLU and its variants fall into.

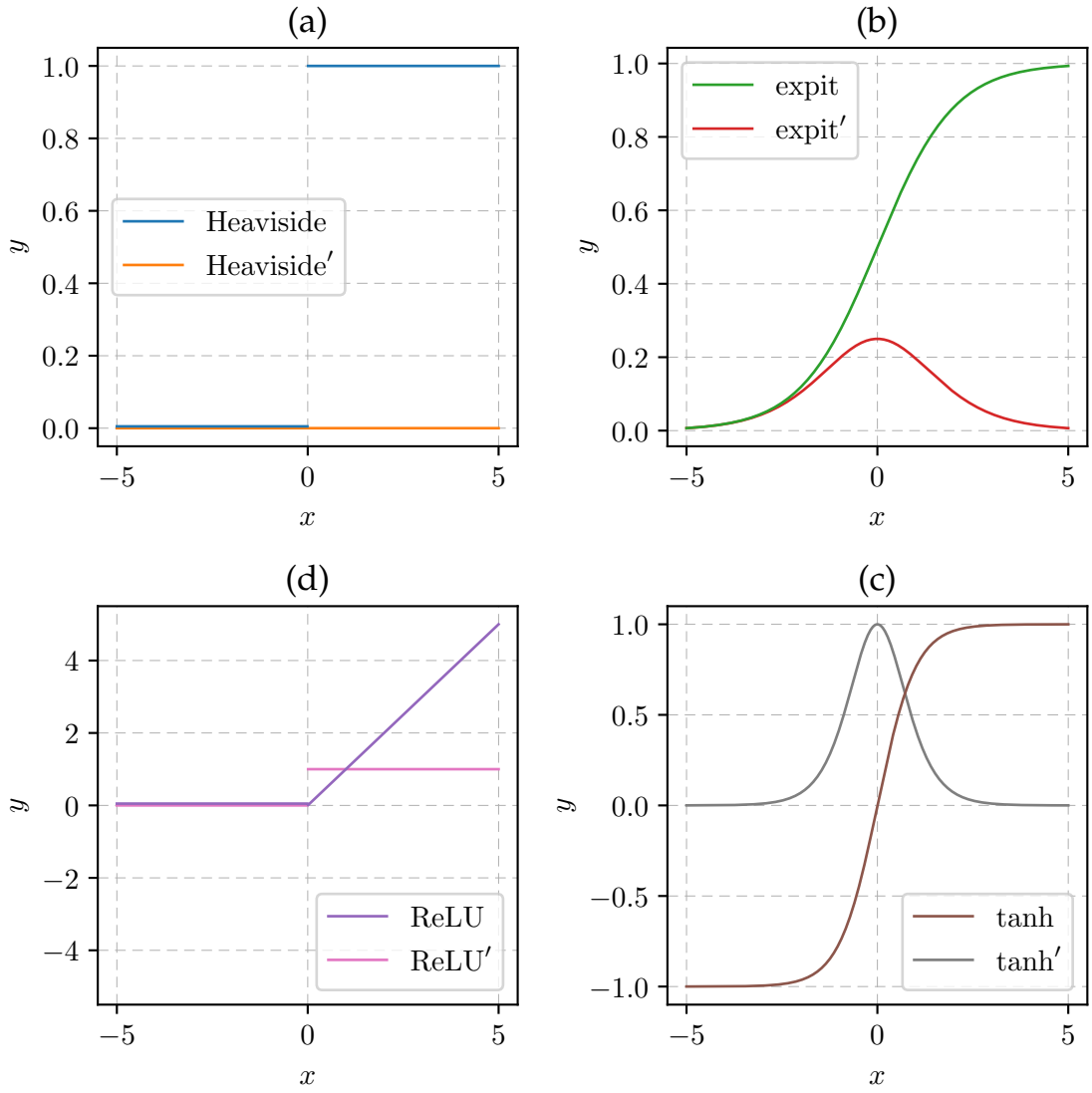


Figure A.6: Common activation functions. (a) The Heaviside step function and its zero-everywhere derivative. (b) The logistic function compressing all of $\mathbb{R}$ into $(0, 1)$, and its hill-shaped derivative. (c) The hyperbolic tangent being symmetric about the origin along with its derivative. (d) The rectified linear unit and its Heaviside derivative.

Activation functions like those in Figure A.6 should only be used for neurons in hidden layers of a neural network since they all have a restricted image. Instead, for *regression tasks* where the desired output is real-valued and numerical, one simply omits the activation, i.e., uses the identity to allow the affine transformation result to pass through. For *classification tasks*, where the output is discrete and categorical, we need an additional layer that allows the network to communicate its categorical decision.

Definition 13 (Softmax)**.** The *softmax* activation function is given for $\boldsymbol{x} \in \mathbb{R}^n$ as

$$\operatorname{softmax}(\boldsymbol{x}) := \left[\frac{\exp(x_1)}{\sum_{k=1}^{n} \exp(x_k)}, \ldots, \frac{\exp(x_n)}{\sum_{k=1}^{n} \exp(x_k)} \right] \in \mathbb{R}^n$$

Please note that $\mathrm{softmax}$ is *not* an activation function of a single neuron, but rather an entire layer that takes into account all outputs of the previous layer. It has a crucial property that enables neural networks to "decide" between different options.

Remark 5 (Probability interpretation)**.** After applying softmax to a vector, its components take values in $(0, 1)$ and add up to 1, since for each $i = 1, \dots, n$ we have:

$$0 < \frac{\overbrace{\exp(x_i)}^{>0}}{\sum_{k=1}^{n} \exp(x_k)} < \sum_{j=1}^{n} \frac{\exp(x_j)}{\sum_{k=1}^{n} \exp(x_k)} = 1$$

Thus, the components of the result can be interpreted as probabilities associated with the respective entry. If the dimensionality is chosen equal to the number of categorical possibilities of a classification task, we can interpret the $\mathrm{softmax}$ output as *directly* realizing the discrete case of the probability distribution considered in Section A.1. Beyond that, this special output layer has further desirable properties.

Remark 6 (Monotonicity of $\mathrm{softmax}$)**.** Let $\boldsymbol{x} \in \mathbb{R}^n$. If $x_i \leq x_j$, then also

$$\mathrm{softmax}(\boldsymbol{x})_i = \frac{\exp(x_i)}{\sum_{k=1}^{n} \exp(x_k)} \leq \frac{\exp(x_j)}{\sum_{k=1}^{n} \exp(x_k)} = \mathrm{softmax}(\boldsymbol{x})_j.$$

Hence, the order of components is preserved and $\arg\max \mathrm{softmax}(\boldsymbol{x}) = \arg\max \boldsymbol{x}$. This explains the function's name as portmanteau of *soft* and $\arg\max$.

Lemma 6 (Automonotonicity of $\mathrm{softmax}$)**.** The activation function $\mathrm{softmax}$ is *automonotonic*, i.e., for each $i \in \{1, \dots, n\}$, the component $\mathrm{softmax}_i$ is monotonic with respect to x_i, but reversly monotonic with respect to all other x_j with $j \neq i$.

Proof. Since $\mathrm{softmax}$ is composed of differentiable functions on $\mathbb{R}$, it is itself differentiable. Using the quotient rule, we calculate the diagonal derivative

$$\frac{\partial \,\mathrm{softmax}_j}{\partial x_j}(\boldsymbol{x}) = \frac{\exp(x_j) \sum_{k=1}^{n} \exp(x_k) - \exp(x_j) \exp(x_j)}{(\sum_{k=1}^{n} \exp(x_k))^2} = \frac{\sum_{k \neq j} \exp(x_k)^2}{(\sum_{k=1}^{n} \exp(x_k))^2} > 0$$

and find that f_j is monotonically increasing with respect to x_j. For $i \neq j$, we get

$$\frac{\partial \,\mathrm{softmax}_i}{\partial x_j}(\boldsymbol{x}) = -\frac{\exp(x_i) \exp(x_j)}{(\sum_{k=1}^{n} \exp(x_k))^2} < 0$$

and hence f_j is monotonically decreasing with respect to x_i. □

Informally speaking, automonotonicity means increasing the value of a component before applying the function also increases that component's output value but decreases all other output component values. This is exactly the behavior of one of many mutually exclusive possibilities becoming more likely. We could hence interpret the neurons in the last layer before $\mathrm{softmax}$ to compete for delivering the maximum output.

A.3.3 Convolutional Neural Networks

As we have seen, MLPs with appropriate activation functions are theoretically sufficient to learn any task. However, in practice the domain of this task function is often signal data like time series or images. They exhibit a structured topology where proximity is meaningful, like in time or space. In this case, the connection globality of MLPs has some drawbacks [GBC16]:

- *Inefficiency:* Full connection between two layers is the most expensive topology possible. According to Definition 8, a dense layer with n inputs and d outputs requires $(n+1)d$ parameters. These need to be processed during forthpropagation and adapted during backpropagation, which leads to significant time and memory consumption.
- *Redundancy:* Since a dense layer connects to every neuron in the previous layer, it treats them all the same and hence can only find global patterns in their activations. To represent a local pattern, the weight matrix of the dense layer would need to repeat this pattern for every possible input location, effectively constituting a waste of memory.
- *Translation Variance:* Each weight of a dense layer is used for a single neuron-neuron pair only. Thus, when the input is slightly shifted, the output can dramatically differ. For data that preserve semantic under translation like signals, this is problematic.

The described issues are addressed by *convolutional neural networks* (*CNN*) [LeC89], a special type of neural network that introduces input topology awareness. They have been popularized in 2012 when the *AlexNet* [KSH12] architecture won the ImageNet object recognition challenge. The concept of a *convolution* which CNNs rely on stems from the field of signal processing that we will briefly pay a visit to.

Definition 14 (Discrete Signal Space)**.** The set $\ell_1(\mathbb{Z})$ of all functions $x : \mathbb{Z} \to \mathbb{R}$ that are *absolutely summable*, i.e.,

$$\|x\|_1 = \sum_{k\in\mathbb{Z}} \big|x(k)\big| < \infty$$

and have a *finite support*, i.e.,

$$\big|\operatorname{supp}(x)\big| = \big|\{k \in \mathbb{Z} \mid x(k) \neq 0\}\big| < \infty$$

is the *discrete signal space*. We call elements of $\ell_1(\mathbb{Z})$ *signals*.

Signals on their own are not mathematically interesting. One typically wants to perform operations on them. In principle, any transformation could be applied to a signal, but in practice, we expect a certain behavior that we formalize next.

Definition 15 (LTI Filter)**.** A *filter* $F : \ell_1(\mathbb{Z}) \to \ell_1(\mathbb{Z})$ is an operator that maps an input signal x to an output signal Fx. It is called

- **linear** if any linear superposition of signals is preserved by the filter, i.e., for all signals $x, y \in \ell_1(\mathbb{Z})$ and coefficients $a, b \in R$, it holds that:

$$F[\lambda x + \mu y] = \lambda Fx + \mu Fy \tag{A.6}$$

- **time-independent** if shifting the input signal before applying the filter is the same as shifting the result of the filter, i.e., if for any signal $x \in \ell_1(\mathbb{Z})$ and offset $k \in \mathbb{Z}$, it holds that:

$$F[x(\cdot + k)] = (Fx)(\cdot + k) \tag{A.7}$$

If F fulfills both properties, it is called a *linear and time-independent* (*LTI*) filter.

Of course, we can not only perform operations on a single signal but also combine them via component-wise or scalar addition and multiplication. However, the central notion of signal processing is that of *convolving* two signals into a new one, being the eponym of convolutional neural networks.

Definition 16 (Convolution)**.** The *convolution* of two signals $x, y \in \ell_1(\mathbb{Z})$ is given by

$$x * y \coloneqq \sum_{k \in \mathbb{Z}} x(k) y(\cdot - k) \in \ell_1(\mathbb{Z}).$$

Lemma 7 (Commutativity)**.** Convolution is *commutative*: For $x, y \in \ell_1(\mathbb{Z})$,

$$x * y = y * x \tag{A.8}$$

Proof. Let $n \in \mathbb{Z}$. Through the affine index substitution $u \coloneqq n - k$, we obtain:

$$(x * y)(n) = \sum_{k \in \mathbb{Z}} x(k) y(n-k) = \sum_{u \in \mathbb{Z}} x(n-u) y(u) = \sum_{u \in \mathbb{Z}} y(u) x(n-u) = (y * x)(n) \quad \square$$

Example 2. Consider the signals $x, y \in \ell_1(\mathbb{Z})$ defined by:

$$x(k) = \begin{cases} k & \text{if } 0 \leq k \leq 42 \\ 0 & \text{otherwise} \end{cases} \qquad \text{and} \qquad y(k) = \begin{cases} 1 & \text{if } k = 0 \\ 0 & \text{otherwise} \end{cases}$$

Their convolution comes about as:

$$(x * y)(n) = \sum_{k \in \mathbb{Z}} x(k) \underbrace{y(n-k)}_{\neq 0 \text{ iff } k = n} = x(n) \underbrace{y(0)}_{1} = x(n) = \sum_{k \in \mathbb{Z}} y(k) x(n-k) = (y * x)(n)$$

Example 2 shows that any signal can be represented as convolution with the so-called *Dirac impulse* $\delta := \mathbb{1}_{\{0\}}$ that has a peak at the origin and is zero everywhere else:

$$x = x * \delta \tag{A.9}$$

This allows us to exploit the properties of LTI filters for attaining a central result of discrete signal processing.

Theorem 4 (Kernel Theorem)**.** An LTI filter F is fully defined by its *kernel* or *impulse response* $f := F\delta$ via a convolution: For any signal $x \in \ell_1(\mathbb{Z})$, it holds that

$$Fx = f * x.$$

Proof. This follows by direct calculation and leveraging Lemma 7:

$$\begin{aligned} Fx &\overset{(A.9)}{=} F\left[\sum_{k\in\mathbb{Z}} x(k)\delta(\cdot - k)\right] \\ &\overset{(A.6)}{=} \sum_{k\in\mathbb{Z}} x(k) F[\delta(\cdot - k)] \\ &\overset{(A.7)}{=} \sum_{k\in\mathbb{Z}} x(k) \underbrace{(F\delta)}_{f}(\cdot - k) \\ &= x * f \overset{(A.8)}{=} f * x \end{aligned}$$

□

Definition 17 (Cross-Correlation)**.** The *cross-correlation* of $x, y \in \ell_1(\mathbb{Z})$ is given by

$$x \star y := \sum_{k\in\mathbb{Z}} x(k) y(\cdot + k) \in \ell_1(\mathbb{Z}).$$

Remark 7. As the operation symbol suggests, cross-correlation is just a convolution in disguise: With $\overleftarrow{y} := y(-\cdot)$ being the time-reversed version of y, we get:

$$x \star y = \sum_{k\in\mathbb{Z}} x(k) y(\cdot + k) = \sum_{k\in\mathbb{Z}} x(k - \cdot) y(k) = \sum_{k\in\mathbb{Z}} x(\cdot - k) y(-k) = \overleftarrow{y} * x = x * \overleftarrow{y}$$

However, unlike convolution, the cross-correlation is not commutative: In the setting of Example 2, we must realize:

$$(x \star y)(1) = \sum_{k\in Z} x(k) \underbrace{y(1+k)}_{\neq 0 \text{ iff } k=-1} = x(-1) = 0 \neq 1 = x(1) = \sum_{k\in\mathbb{Z}} y(k) x(1+k) = (y \star x)(1)$$

Despite the mathematical beauty of the convolution operation, numerical libraries typically favor cross-correlation since it is conceptually simpler (no kernel flipping), and for many purposes, the choice between the two does not matter.

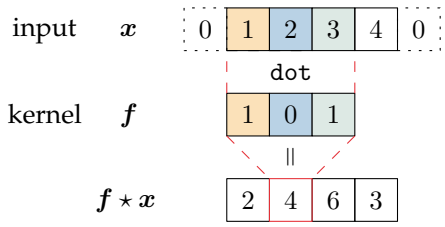


Figure A.7: Cross-correlation of vectors.

To implement signal processing, we need a discrete representation that the computer can handle. Fortunately, this is possible with a simple index shift due to the finite support requirement to signals in the $\ell_1(\mathbb{Z})$ space.

Remark 8 (Vectors $\leftrightarrow$ Signals)**.** We regard any vector $\boldsymbol{x} \in \mathbb{R}^n$ as signal $x \in \ell_1(\mathbb{Z})$ and any signal $y \in \ell_1(\mathbb{Z})$ as vector $\boldsymbol{y} \in \mathbb{R}^{M-m}$ by defining

$$x(k) := \begin{cases} x_k & 1 \leq k \leq n \\ 0 & \text{otherwise} \end{cases} \qquad \text{and} \qquad y_i := y(m+i)$$

where $m := \min \operatorname{supp}(y) - 1$ and $M := \max \operatorname{supp}(y)$.

Thanks to Remark 8, we can apply all signal processing results to vectors. In particular, the operations $\boldsymbol{x} * \boldsymbol{y}$ and $\boldsymbol{x} \star \boldsymbol{y}$ are straightforwardly defined, as illustrated in Figure A.7. In this way, we are ready to define a new layer. The key idea is using a cross-correlation whose kernel is defined by learnable parameters.

Definition 18 (1D Convolutional Layer)**.** A *one-dimensional convolutional layer* with *filter* $\boldsymbol{f} \in \mathbb{R}^w$ of *window size* $w \in \mathbb{N}$, *bias* $\boldsymbol{b} \in \mathbb{R}^{2\lfloor \frac{n}{2} \rfloor + 1}$ and *activation function* $\sigma : \mathbb{R} \to \mathbb{R}$ is for input $\boldsymbol{x} \in \mathbb{R}^n$ given by

$$\operatorname{conv1d}_{\boldsymbol{f},\boldsymbol{b},\sigma}(\boldsymbol{x}) := \sigma(\boldsymbol{f} \star \boldsymbol{x} + \boldsymbol{b}) = \boldsymbol{W}\widetilde{\boldsymbol{x}} + \boldsymbol{b} \in \mathbb{R}^{2\lfloor \frac{n}{2} \rfloor + 1}$$

with $\widetilde{\boldsymbol{x}} := [0_1, \ldots, 0_{\lfloor \frac{w}{2} \rfloor}, x_1, \ldots, x_n, 0_1, \ldots, 0_{\lfloor \frac{w}{2} \rfloor}]^{\mathrm{T}}$ and the Toeplitz matrix

$$\boldsymbol{W} := \begin{bmatrix} \boxed{\boldsymbol{f}^{\mathrm{T}}} & 0 & \ldots & 0 \\ 0 & \boxed{\boldsymbol{f}^{\mathrm{T}}} & \ddots & 0 \\ \vdots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & \boxed{\boldsymbol{f}^{\mathrm{T}}} \end{bmatrix} \in \mathbb{R}^{(2\lfloor \frac{n}{2} \rfloor + 1) \times (n + 2\lfloor \frac{w}{2} \rfloor)}.$$

The convolutional layer's name might be misleading. After all, it uses cross-correlation. However, and this is the crucial point, the parameters of the kernel are not manually set but *learned* during the optimization process! Hence, the difference between convolution and cross-correlation is rendered void. If we replaced cross-correlation with real convolution in Definition 18, the network could simply learn the flipped kernel.

Convolutional layers are very similar to fully-connected layers in that they may be represented as an affine transformation. However, the key difference is that the weight matrix of a convolutional layer is *sparse* due to the window size usually being much smaller than the input size. Furthermore, the *same* parameters are used along the diagonal, meaning that they are *shared* across all input locations. This leads convolutional layers to be *translation-equivariant*[12], so that when a pattern is shifted in the input, the corresponding representation resulting from cross-correlation is shifted in the same way.

Our definition of a convolutional layer is simplified. Firstly, convolutional layers are normally stacked to allow for *multiple filters* to be used. This, of course, increases the parameter number and computational cost but is usually still cheaper than a fully-connected layer and further has all the properties that the latter lacks. Secondly, plain cross-correlation slides the filter window across the input one at a time. In practice, one allows for this *stride* to be a higher value than one, which is mathematically equivalent to a cross-correlation with unit stride and subsequent downsampling. Thirdly, our formulation implicitly adds zero padding to the input borders of half the kernel size so that each component receives an output value. This is called *same* convolution, but there are also other strategies such as *valid* convolution without zero-padding and *full* convolution that pads with the entire kernel size [GBC16].

Regardless of the concrete arrangement of a convolutional layer, it is highly efficient. Since the results for different output positions are entirely independent from one another, convolutional layers can be parallelized straightaway. Specialized routines for convolution operations in modern GPUs speed things up even more. To also lessen memory requirements, it is common to follow a convolutional layer with another one that shrinks output size by producing local summaries.

Definition 19 (Pooling Layer)**.** A *one-dimensional pooling layer* with *reduction function* $\varrho : \mathbb{R}^w \to \mathbb{R}$ of *window size* $w \in \mathbb{N}$ is a function $\mathrm{pool}_\varrho : \mathbb{R}^{nw} \to \mathbb{R}^n$ given by:

$$\mathrm{pool}_\varrho(\boldsymbol{x}) \coloneqq [\varrho(x_1, \ldots, x_w), \ldots, \varrho(x_{(n-1)w+1}, \ldots, x_{nw})] \in \mathbb{R}^n$$

Typical reduction functions for pooling layers are *average-pooling* or *max-pooling*. Using the latter adds an additional property to a CNN. Since the summary statistic within a window will not change for most windows when shifting the input by a tiny amount, we can regard CNNs with pooling as being *approximately locally translation-invariant* [GBC16]. This is especially the case when using multiple filters.

Our considerations so far were restricted to one-dimensional inputs. However, everything discussed here can be lifted to two or more dimensions in a straightforward way: The windows of filters and pooling layers simply become multi-dimensional as well.

[12] From Latin *equi* = equal and *variare* = change, meaning "equally changing".

A.3.4 Recurrent Neural Networks

The success of CNNs shows that exploiting structure present in the underlying data through specialized components is crucial in deep learning. While convolutional networks are able to filter out certain translation-equivariant local features in the data, another important variant of neural networks are *recurrent* ones. They model time semantics, which shows up in all kinds of *sequential* data like time series or text. We motivate their construction by considering the parameterized dynamical system

$$\boldsymbol{h}^{(t)} = f_{\boldsymbol{\theta}}(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}), \tag{A.10}$$

where each *state* $\boldsymbol{h}^{(t)}$ is a function of the *previous* state $\boldsymbol{h}^{(t-1)}$ and the current input $\boldsymbol{x}^{(t)}$. This is a recurrent definition, so it requires some *initial* state $\boldsymbol{h}^{(0)}$. We can then unfold the expression to obtain, as visualized in Figure A.8, a sequential formulation:

$$\boldsymbol{h}^{(t)} = f_{\boldsymbol{\theta}}(f_{\boldsymbol{\theta}}(\dots f_{\boldsymbol{\theta}}(\boldsymbol{h}^{(0)}, \boldsymbol{x}^{(1)}) \dots, \boldsymbol{x}^{(t-1)}), \boldsymbol{x}^{(t)}).$$

The dynamical system model (A.10) thus assumes that each state can be determined from the previous states and inputs by using the *same function* for all time steps. If you will, you could interpret $f_{\boldsymbol{\theta}}$ as some "law of nature" inherent in the data. Consequently, the parameters $\boldsymbol{\theta}$ are *shared* across all time steps. Recall that parameter sharing was already a winner concept for CNNs in Section A.3.3! To turn our dynamical system into a neural network layer as well, Elman [Elm90] chose $f_{\boldsymbol{\theta}}$ in the standard way as an affine transformation followed by a non-affinity.

Definition 20 (RNN Layer)**.** A *Recurrent Neural Network* (*RNN*) *layer* with *input weight* $\boldsymbol{W} \in \mathbb{R}^{m \times d}$*, hidden weight* $\boldsymbol{U} \in \mathbb{R}^{d \times d}$*, bias* $\boldsymbol{b} \in \mathbb{R}^d$*, activation function* $\sigma : \mathbb{R} \to \mathbb{R}$ and *initial state* $\boldsymbol{h}^{(0)} \in \mathbb{R}^d$ is given for *input sequence* $\boldsymbol{X} = [\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(n)}] \in \mathbb{R}^{m \times n}$ by

$$\mathrm{RNN}_{\boldsymbol{W},\boldsymbol{U},\boldsymbol{b},\sigma,\boldsymbol{h}^{(0)}}(\boldsymbol{X}) \coloneqq [\boldsymbol{h}^{(1)}, \dots, \boldsymbol{h}^{(n)}] \in \mathbb{R}^{d \times n}$$

where each *hidden state* $\boldsymbol{h}^{(t)}$ for $t = 1, \dots, n$ is

$$\boldsymbol{h}^{(t)} \coloneqq \sigma(\boldsymbol{W}\boldsymbol{x}^{(t)} + \boldsymbol{U}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}).$$

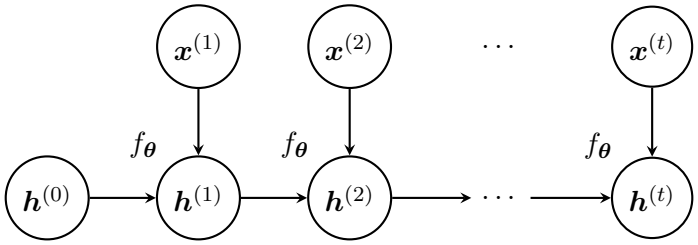


Figure A.8: Unfolded computational graph of the dynamical system (A.10)

Recurrent layers have some advantages: For one, their parameter number only depends on the dimensionality of the input and the hidden states, but not on the sequence length. Thus, they work with arbitrarily long inputs. Furthermore, they can use their hidden state to transport information through time, i.e., use it as memory. Thanks to this, RNNs are universal: They have been shown to be capable of representing any function that Turing machines can compute [SS91; SS95; Hyo96].

However, there is a major drawback to vanilla RNN layers as per Definition 20: Since they repeatedly apply the same function over and over again for potentially a large number of steps, they tremendously suffer from the vanishing gradient problem introduced in Section A.3.2. This leads to RNNs not being able to capture *long-term dependencies*, as they require gradients to follow a long path in the computational graph without changing their magnitude significantly [Hoc91; BFS93; BSF94].

The key idea to mitigating the vanishing gradient problem in recurrent networks is adding paths to the computational graph along which the gradient can flow unimpeded. This has been famously done by Hochreiter and Schmidhuber [HS97] with an architecture evocative of combinatorial circuits. We first give its formal layer definition and then elucidate the mechanics based on Figure A.9.

Definition 21. (LSTM layer) A *long short-term memory* (*LSTM*) *layer* with *input weights* $\boldsymbol{W}_i, \boldsymbol{W}_f, \boldsymbol{W}_o, \boldsymbol{W}_a \in \mathbb{R}^{m\times d}$, *hidden weights* $\boldsymbol{U}_i, \boldsymbol{U}_f, \boldsymbol{U}_o, \boldsymbol{U}_a \in \mathbb{R}^{d\times d}$, *biases* $\boldsymbol{b}_i, \boldsymbol{b}_f, \boldsymbol{b}_o, \boldsymbol{b}_a \in \mathbb{R}^d$, *activation function* $\sigma : \mathbb{R} \to \mathbb{R}$ and *initial states* $\boldsymbol{h}^{(0)}, \boldsymbol{s}^{(0)} \in \mathbb{R}^d$ is given for *input sequence* $\boldsymbol{X} = [\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(n)}] \in \mathbb{R}^{m\times n}$ by

$$\mathrm{LSTM}_{\boldsymbol{W}_i,\boldsymbol{W}_f,\boldsymbol{W}_o,\boldsymbol{W}_a,\boldsymbol{U}_i,\boldsymbol{U}_f,\boldsymbol{U}_o,\boldsymbol{U}_a,\boldsymbol{b}_i,\boldsymbol{b}_f,\boldsymbol{b}_o,\boldsymbol{b}_a,\sigma,\boldsymbol{h}^{(0)},\boldsymbol{s}^{(0)}}(\boldsymbol{X}) \coloneqq [\boldsymbol{h}^{(1)}, \dots, \boldsymbol{h}^{(n)}] \in \mathbb{R}^{d\times n}$$

where each *hidden state* $\boldsymbol{h}^{(t)}$ for $t = 1, \dots, n$ results from the components

$$\begin{aligned}
(\textit{Input gate}) \quad & \boldsymbol{i}^{(t)} \coloneqq \mathrm{expit}(\boldsymbol{W}_i\boldsymbol{x}^{(t)} + \boldsymbol{U}_i\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_i) \\
(\textit{Forget gate}) \quad & \boldsymbol{f}^{(t)} \coloneqq \mathrm{expit}(\boldsymbol{W}_f\boldsymbol{x}^{(t)} + \boldsymbol{U}_f\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_f) \\
(\textit{Output gate}) \quad & \boldsymbol{o}^{(t)} \coloneqq \mathrm{expit}(\boldsymbol{W}_o\boldsymbol{x}^{(t)} + \boldsymbol{U}_o\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_o) \\
\\
(\textit{Activation}) \quad & \boldsymbol{a}^{(t)} \coloneqq \sigma(\boldsymbol{W}_a\boldsymbol{x}^{(t)} + \boldsymbol{U}_a\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_a) \\
(\textit{Cell state}) \quad & \boldsymbol{s}^{(t)} \coloneqq \boldsymbol{f}^{(t)} \odot \boldsymbol{s}^{(t-1)} + \boldsymbol{i}^{(t)} \odot \boldsymbol{a}^{(t)} \\
(\textit{Hidden state}) \quad & \boldsymbol{h}^{(t)} \coloneqq \boldsymbol{o}^{(t)} \odot \sigma(\boldsymbol{s}^{(t)})
\end{aligned}$$

with $\odot$ denoting the componentwise *Hadamard product*.

The central component of an LSTM layer, besides the hidden state also present in a vanilla RNN layer, is its additional *cell state* that only participates in "lightweight" operations so as to keep gradients stable. We can think of it as a highway with occasional on-ramps and exits. Like in a vanilla RNN, the LSTM produces in each time step an activation as an affine transformation of the current context.

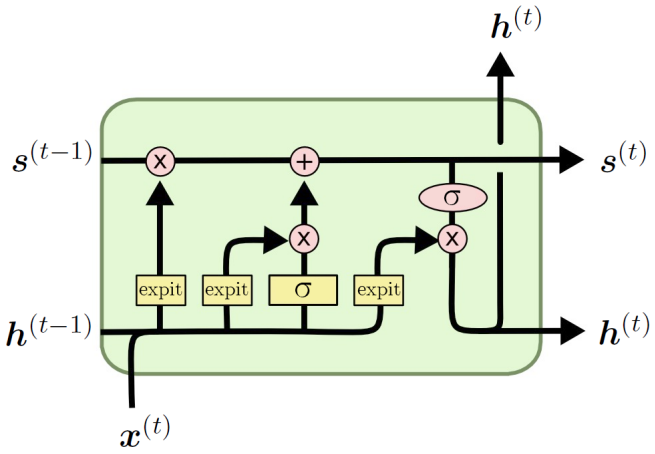


Figure A.9: Schematic overview of an LSTM cell, modified from Wikimedia [Wik18].

In contrast to the RNN layer, this context activation is further modified via three *gates*. Each gate has its own parameters to be conditioned on the current context and yields a factor between zero and one, indicating how much information to let through: The *forget gate* and *input gate* control the contribution of the old cell state and the context activation to the new internal cell state, respectively. The *output gate* controls the contribution of the updated internal cell state to the hidden state that is exposed as output.

The LSTM performs remarkably in many applications including unconstrained speech recognition [GMH13], machine translation [SVL14] and image captioning [Mao+14]. For completeness, we mention another RNN extension addressing the vanishing gradient problem called *gated recurrent unit* (*GRU*) [Cho+14]. As the name suggests, it just like the LSTM utilizes the concept of gates but arranges these in a slightly different way. Most significantly, it combines the forget and input gate to a single update gate that produces a weighted average of the past state and the current activation.

Whatever recurrent layer one chooses, it provides representations that take into account *past* information. Depending on the application, this one-sidedness is strictly required. If it is not, a *bidirectional* RNN [SP97] enables output to depend on the whole input sequence. It consists of two RNNs that work in parallel: A forward RNN processes the input in order, while a backward RNN proceeds in reverse order. Their respective hidden state outputs are simply concatenated to yield the overall output. Bidirectional RNNs have been successful, for instance, in handwriting recognition [Gra+09], speech recognition [GS05] and protein structure prediction [Bal+99].

Finally, we note that despite the success of modern recurrent architectures, they all have a performance problem: Any RNN is inherently sequential since each time step may only be processed once the previous is done. Consequently, RNNs cannot be efficiently parallelized, which is a point in which they dramatically differ from CNNs. Hence, they should not be used with overly long input sequences [GBC16].

A.3.5 Attention Mechanisms

The non-parallelizability of recurrent architectures as described in Section A.3.4 poses a serious problem in practice. Furthermore, due to the sequential nature, even LSTMs suffer from vanishing gradients over long paths [Li+18]. Recently, RNNs are increasingly replaced by so-called *attention mechanisms* that lack the temporal semantics of RNNs. Their power arises from imitating the eponymous cognitive process of humans.[13]

At first, attention mechanisms were used only as a supplement to RNNs in the context of machine translation. Suppose a two-layer RNN where the first layer has produced hidden states $\boldsymbol{h}^{(1)}, \dots, \boldsymbol{h}^{(n)} \in \mathbb{R}^n$ and the second layer is about to determine its hidden state $\boldsymbol{s}^{(t)} \in \mathbb{R}^d$. Bahdanau et al. [BCB14] modify the second layer as follows: Instead of directly using $\boldsymbol{h}^{(t)}$ as current input, they interpose an *alignment model* $a : \mathbb{R}^d \times \mathbb{R}^n \to \mathbb{R}$, itself a small neural network. It scores how well the input at position i aids the output calculation at time t:

$$e_i^{(t)} := a(\boldsymbol{s}^{(t-1)}, \boldsymbol{h}^{(i)})$$

These *energy* values are then via $\mathrm{softmax}$ turned into a probability distribution over the input hidden states:

$$\alpha_i^{(t)} := \frac{\exp(e_i^{(t)})}{\sum_{j=1}^{n} \exp(e_j^{(t)})}$$

The resulting *attention weights* can differ for any combination of previous state and input. They finally determine what parts of the input to attend to in a weighted sum defining the *context vector*:

$$\boldsymbol{c}^{(t)} := \sum_{i=1}^{n} \alpha_i^{(t)} \boldsymbol{h}^{(i)}$$

It is this context vector that the second RNN layer receives as input at time t. Thanks to the alignment model a, the context vector $\boldsymbol{c}^{(t)}$ contains a summary of the information most relevant for processing this time step. Furthermore, the attention weights $\alpha_i^{(t)}$ explicitly model an information selection process. Hence, we can gain insight into this process for a fully trained model by inspecting the attention weight structure as in Figure A.10. This increases interpretability for sequence models.

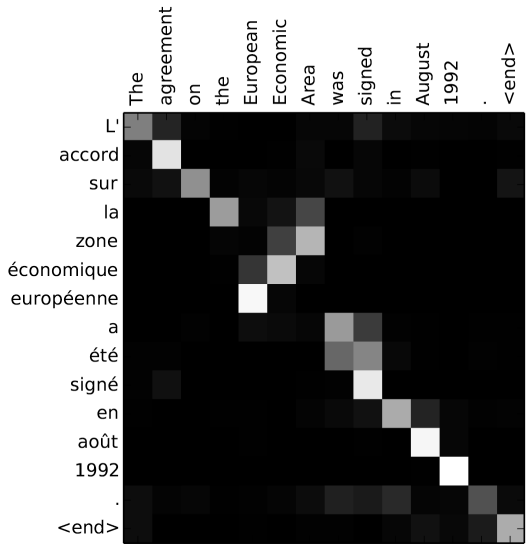


Figure A.10: Visualization of the attention weights between words in an English-French machine translation task, with brightness indicating weight size. Taken from Bahdanau et al. [BCB14].

[13] As great as it may sound, we should be cautious with neuroscientific analogies in deep learning.

The described attention mechanism is a so-called *soft attention* mechanism since it uses a weighted sum to smoothly attend to input vectors. On the other hand, *hard attention* mechanisms aim to model human attention more closely by only attending to one or several inputs fully and completely ignoring the rest. However, since the involved $\arg\max$ operation is not differentiable, hard attention cannot be trained with gradient descent. Instead, specialized optimization routines are required. Beyond that, attention mechanisms are classified as being *additive* or *multiplicative*, and *global* or *local* [LPM15].

Regardless of the concrete attention mechanism, it was originally an extension to RNNs and thus, by design, relies on them. On the contrary, the highly successful *Transformer* architecture [Vas+17; Dev+19; Bro+20] follows the principle "attention is all you need" by relying solely on attention as a first-class citizen. To this end, it generalizes the concept to a broader scheme drawing from concepts of information retrieval: An attention mechanism can be regarded as matching queries to a set of key-value pairs, in analogy to matching search strings to characteristic metadata of documents. In the RNN setting, the queries are the respective previous states, and the values are the hidden states from the first layer. The keys are equal to the values, which is the most common case.

Breaking away from RNNs, Transformers allow all queries to attend to all keys at the same time. As this removes the temporal dependence from the architecture, the input itself has to carry this information through an appropriate temporal encoding. Packing all queries, keys, and values as columns into matrices gives rise to a standalone attention layer depicted in Figure A.11 that employs the dot product as an alignment model.

Definition 22 (Attention Layer)**.** An *attention layer* is given for *queries* $\boldsymbol{Q} \in \mathbb{R}^{d_k \times n}$ and *keys* $\boldsymbol{K} \in \mathbb{R}^{d_k \times n}$ with associated *values* $\boldsymbol{V} \in \mathbb{R}^{d_v \times n}$ by

$$\mathrm{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) := \boldsymbol{V}\,\mathrm{softmax}\left(\frac{K^T Q}{\sqrt{d_k}}\right) \in \mathbb{R}^{d_v \times n}.$$

In the case $\boldsymbol{Q} = \boldsymbol{K} = \boldsymbol{V}$ it is called a *self-attention* layer.

What catches the eye in Definition 22 is the division of the dot product score by the square root of the value dimension. This amounts to a normalization factor for numerical stability, since otherwise $\mathrm{softmax}$ increasingly tends towards regions of small gradient, impeding backpropagation [Vas+17].

Note that this attention layer introduces no learnable parameters. While the earlier attention mechanisms employed a dedicated sub-network to determine alignment scores, the Transformer attention layer expects the previous layers to learn appropriate representations such that the dot product is a meaningful measure of mutual relevance. Only stacking multiple attention layers does not increase the learning capabilities. Transformers thus always employ dense layers before attention layers, which one should keep in mind when using the attention layer outside of a Transformer architecture.

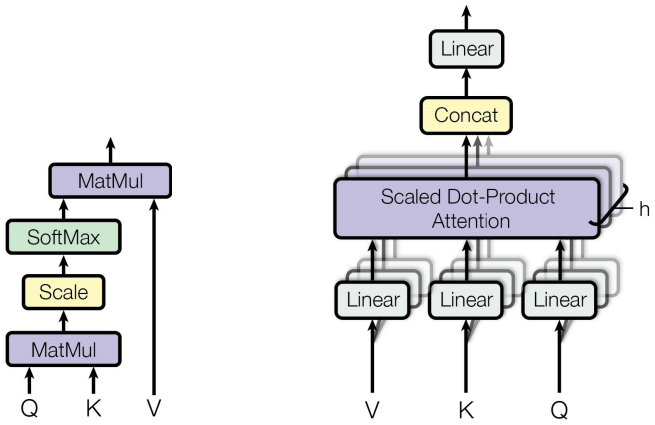


Figure A.11: Schematic sketch of attention layers, adapted from Vaswani et al. [Vas+17]. *Left:* Plain attention layer. *Right:* Multi-head attention layer.

Different aspects of a learning task might require different attention alignments. For this reason, Transformers use an improved version of the plain attention layer from Definition 22 employing multiple so-called *attention heads*: The queries, keys, and values are first projected into different subspaces capturing different aspects. On each of these subspaces, the standard attention layer is applied. Finally, the resulting "heads" are reintegrated to yield an output, as shown in Figure A.11.

Definition 23 (Multi-Head Attention Layer)**.** A *multi-head attention layer* with $h \in \mathbb{N}$ heads and *weights* $\boldsymbol{W}_i^{\boldsymbol{Q}} \in \mathbb{R}^{d_k \times hd_k}, \boldsymbol{W}_i^{\boldsymbol{K}} \in \mathbb{R}^{d_k \times hd_k}, \boldsymbol{W}_i^{\boldsymbol{V}} \in \mathbb{R}^{d_v \times hd_v}, \boldsymbol{U} \in \mathbb{R}^{hd_v \times hd_v}$ is given for *queries* $\boldsymbol{Q} \in \mathbb{R}^{hd_k \times n}$ and *keys* $\boldsymbol{K} \in \mathbb{R}^{hd_k \times n}$ with *values* $\boldsymbol{V} \in \mathbb{R}^{hd_v \times n}$ by

$$\mathrm{MultiheadAttention}_{\boldsymbol{W}_1^{\boldsymbol{Q}},\dots,\boldsymbol{W}_h^{\boldsymbol{Q}},\boldsymbol{W}_1^{\boldsymbol{K}},\dots,\boldsymbol{W}_h^{\boldsymbol{K}},\boldsymbol{W}_1^{\boldsymbol{V}},\dots,\boldsymbol{W}_h^{\boldsymbol{V}},\boldsymbol{U}}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}) \coloneqq \boldsymbol{U} \begin{bmatrix} \mathrm{head}_1 \\ \vdots \\ \mathrm{head}_h \end{bmatrix}$$

where the *attention heads* are defined as:

$$\mathrm{head}_i \coloneqq \mathrm{Attention}(\boldsymbol{W}_i^{\boldsymbol{Q}}\boldsymbol{Q}, \boldsymbol{W}_i^{\boldsymbol{K}}\boldsymbol{K}, \boldsymbol{W}_i^{\boldsymbol{V}}\boldsymbol{V})$$

To prevent misunderstandings, we highlight that although the multi-head attention layer does have a bunch of weights, these are not sufficient for learning appropriate alignments, alone for their comparatively low dimensionality. They serve only as a means to an end to performing the subspace transformation. As discussed before, preceding (usually dense) layers are responsible for the actual learning process. Attention layers merely impose a strong inductive bias on *how* the parameters should be learned.

In summary, we have now covered the elementary concepts in neural network design. The major architectures comprise MLPs, CNNs, RNNs, and Transformer networks. They do not need to be used in isolation but can, of course, be employed jointly, depending on the application.

A.4 Multi-class evaluation

Suppose that, applying everything discussed so far, we have trained a neural network model $\hat{f}_{\boldsymbol{\theta}}$ with gradient descent based on the cross-entropy loss function $\mathcal{L}_{\mathcal{D}}$. Now we intend to evaluate the performance of the model. To this end, we need an adequate metric. In principle, we could simply call on the loss since it indicates the prediction error used for optimization. However, while the cross-entropy loss is the theoretically most grounded measure, it is not very interpretable. In this section, we derive the crucial evaluation metrics used for classification tasks, where the codomain $\mathcal{Y}$ is finite and categorical. Without loss of generality, we may thus assume $\mathcal{Y} = \{1, \dots, C\}$ for the number $C \in \mathbb{N}$ of classes.

For discrete tasks, as classification is one, we correspondingly desire a discrete performance measure. The most intuitive such choice is to count the percentage of correct classifications, i.e., where model prediction and label in the dataset agree.

Definition 24 (Accuracy)**.** The *accuracy* of $\hat{f}_{\boldsymbol{\theta}}$ is defined as:

$$\alpha := \frac{|\{(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D} \mid \hat{f}_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{y}\}|}{|\mathcal{D}|}$$

Example 3 (Imbalanced Classes)**.** On $\mathcal{D} := \{(1,1), \dots, (99,1), (100,2)\}$, the majority class predictor $\hat{f}_{\boldsymbol{\theta}} : \mathbb{R} \to \{1,2\}, \boldsymbol{x} \mapsto 1$ has accuracy $\alpha = 99\,\%$.

As the example reveals, accuracy is highly sensitive to class imbalance. This implies that, depending on the underlying dataset, the same accuracy score may be considered either really good or disastrously bad. To obtain more robust metrics, we must break down the model's predictions on the dataset into all possible combinations of true and predicted value.

Definition 25 (Confusion Matrix)**.** The *confusion matrix* $\mathfrak{C} \in \mathbb{N}^{C \times C}$ of $\hat{f}_{\boldsymbol{\theta}}$ is given by:

$$\mathfrak{C}_{ij} := |\{(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D} \mid \boldsymbol{y} = i, \hat{f}_{\boldsymbol{\theta}}(\boldsymbol{x}) = j\}|$$

Remark 9 (Accuracy Reloaded)**.** We can obtain accuracy from the confusion matrix:

$$\alpha = \frac{\text{“diagonal sum”}}{\text{“total sum”}} = \frac{\sum_{c=1}^{C} \mathfrak{C}_{cc}}{\sum_{i,j=1}^{C} \mathfrak{C}_{ij}}$$

The confusion matrix not only subsumes accuracy but, by definition, contains all quantitative information about the model's predictions. Certain groupings in the matrix are particularly meaningful and well-known from medical test design.

Definition 26 (Positives and Negatives)**.** For each class $c \in \{1, \dots C\}$, we define

(a) the number of *true positives* as $\mathrm{TP}_c := \mathfrak{C}_{cc}$,

(b) the number of *false positives* as $\mathrm{FP}_c := \sum_{i \neq c} \mathfrak{C}_{ic}$,

(c) the number of *false negatives* as $\mathrm{FN}_c := \sum_{i \neq c} \mathfrak{C}_{ci}$,

(d) the number of *true negatives* as $\mathrm{TN}_c := \sum_{i,j \neq c} \mathfrak{C}_{ij}$.

These statistics are disjoint and partition the prediction space into groups as illustrated in Figure 26. Based on them, further quantities of interest are the fraction of correct positive predictions and the hit ratio of actually positive objects.

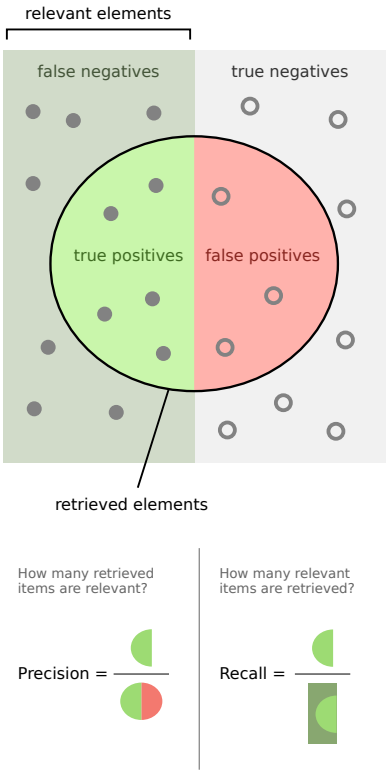


Figure A.12: Conceptualization of positives, negatives, precision and recall. Taken from Wikimedia [Wik14].

Definition 27 (Precision)**.** The *precision* of $\hat{f}_{\boldsymbol{\theta}}$ on class $c \in \{1, \dots, C\}$ is:

$$\pi_c := \frac{\mathrm{TP}_c}{\mathrm{TP}_c + \mathrm{FP}_c}$$

Definition 28 (Recall)**.** The *recall* of $\hat{f}_{\boldsymbol{\theta}}$ on class $c \in \{1, \dots, C\}$ is:

$$\rho_c := \frac{\mathrm{TP}_c}{\mathrm{TP}_c + \mathrm{FN}_c}$$

Remark 10 (Accuracy Revolutions)**.** Recall[14] that accuracy is the fraction of correct predictions out of all. Definition 26 lets us thus reformulate it for the third time:

$$\alpha = \frac{\sum_{c=1}^{C} \mathrm{TP}_c}{\sum_{c=1}^{C} (\mathrm{TP}_c + \mathrm{FP}_c)}$$

Metrics based on true/false positives/negatives have the advantage of yielding a class-wise score. This allows for fine-grained insight into the model's prediction behavior. They achieve class-specificity by evaluating the predictions of a class in a one-vs-rest fashion. In the same vein, we can define a per-class version of accuracy as well.

Definition 29 (Class Accuracy)**.** The *accuracy* of $\hat{f}_{\boldsymbol{\theta}}$ on class $c \in \{1, \dots, C\}$ is:

$$\alpha_c := \frac{\mathrm{TP}_c + \mathrm{TN}_c}{\mathrm{TP}_c + \mathrm{TN}_c + \mathrm{FP}_c + \mathrm{FN}_c}$$

[14] Yes, pun intended.

We know from Example 3 that accuracy is highly sensitive to imbalanced label distributions and thus easily misleading when the imbalance is not taken into account. For its class-wise version, this holds just as well. Furthermore, precision and recall are not immune from this behavior either.

Example 4 (Precision and Recall Pathology)**.** On $\mathcal{D} \coloneqq \{(1,1),(2,2),(3,2)\}$, the constant classifier $\hat{f}_{\boldsymbol{\theta}} : \mathbb{R} \to \{1,2\}, \boldsymbol{x} \to 1$ has recalls $\rho_1 = 100\,\%$, $\rho_2 = 0\,\%$ and precisions $\pi_1 \approx 33\,\%$, $\pi_2 = 100\,\%$.

Intuitively, recall rewards overconfidence in a class decision while precision rewards careful decisions. Example 4 hints at two remedies for making the metrics more robust: Combining precision and recall into a single metric, and integrating class-wise metrics to construct a summary metric. The first option can be implemented with a simple harmonic mean.

Definition 30 (F1 Score)**.** The *F1 score* of $\hat{f}_{\boldsymbol{\theta}}$ on class $c \in \{1,\dots,C\}$ is defined as:

$$\mathrm{F1}_c \coloneqq \frac{2}{\frac{1}{\pi_c} + \frac{1}{\rho_c}} = \left(\frac{\pi_c^{-1} + \rho_c^{-1}}{2} \right)^{-1}$$

Note that an arithmetic average would not be appropriate since precision and recall do not share the same denominator but the same numerator! Through averaging, the $\mathrm{F1}$ score provides a more equilibrated performance summary. To address the second remedy from above, we need to combine the metrics across classes. To this end, two major strategies exist.

Definition 31 (Macro Averaging)**.** The *macro precision*, *macro recall* and *macro F1 score* of $\hat{f}_{\boldsymbol{\theta}}$ are respectively given by:

$$\pi_M \coloneqq \frac{1}{C} \sum_{c=1}^{C} \pi_c, \qquad \rho_M \coloneqq \frac{1}{C} \sum_{c=1}^{C} \rho_c, \qquad \mathrm{F1}_M \coloneqq \frac{2}{\frac{1}{\pi_M} + \frac{1}{\rho_M}}$$

Clearly, macro averaging admits of equal importance for all classes, regardless of their relative appearance in the dataset. In contrast, the second approach scales the individual classes' contribution according to their prevalence.

Definition 32 (Micro Averaging)**.** The *micro precision*, *micro recall* and *micro F1 score* of $\hat{f}_{\boldsymbol{\theta}}$ are respectively given by:

$$\pi_\mu \coloneqq \frac{\sum_{c=1}^{C} \mathrm{TP}_c}{\sum_{c=1}^{C} (\mathrm{TP}_c + \mathrm{FP}_c)}, \qquad \rho_\mu \coloneqq \frac{\sum_{c=1}^{C} \mathrm{TP}_c}{\sum_{c=1}^{C} (\mathrm{TP}_c + \mathrm{FN}_c)}, \qquad \mathrm{F1}_\mu \coloneqq \frac{2}{\frac{1}{\pi_\mu} + \frac{1}{\rho_\mu}}$$

Despite having the more complicated definition, micro averaged metrics surprisingly do not constitute new performance measures. Quite the opposite: They are equivalent to the simplest measure that we encountered.

Theorem 5 (Micro Averaging = Accuracy)**.** Micro F1, micro precision, micro recall and accuracy of $\hat{f}_{\boldsymbol{\theta}}$ are all equal:

$$\mathrm{F1}_\mu = \pi_\mu = \rho_\mu = \alpha$$

Proof. Comparing Definition 32 with Remark 10, we immediately get $\pi_\mu = \alpha$. Motivated by the observation that any false positive for one class is exactly one false negative for another class and vice versa, we further obtain:

$$\sum_{c=1}^{C} \mathrm{FP}_c = \sum_{c=1}^{C} \sum_{i \neq c} \mathfrak{C}_{ic} = \sum_{i \neq j} \mathfrak{C}_{ij} = \sum_{c=1}^{C} \sum_{i \neq c} \mathfrak{C}_{ci} = \sum_{c=1}^{C} \mathrm{FN}_c$$

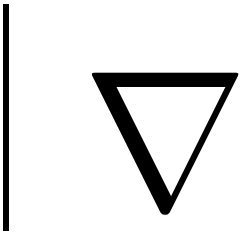
Substituting this in Definition 32 shows $\rho_\mu = \pi_\mu = \alpha$. Hence also

$$\mathrm{F1}_\mu = \frac{2}{\frac{1}{\alpha} + \frac{1}{\alpha}} = \alpha\,. \qquad \square$$

Thereby macro F1 remains as the most robust evaluation metric of those discussed. However, we remark that Theorem 5 only holds in the case of single-label classification, which we considered in this whole Section. When we allow multiple labels for a sample, micro averages get to play their role as well. Furthermore, we note the existence of many further, yet less common, metrics. Worth mentioning are general *F-scores*, which our F1 is a special case of, *specificity* and *sensitivity* arising from the confusion matrix, as well as the more involved *Cohen's Kappa*.

This concludes our journey through the mathematical foundations of deep learning. Understanding the main part of this thesis should now be obstacle-free.

Bibliography



[And+07] Brian J. Anderson et al. "The Magnetometer Instrument on MESSENGER". In: *Space Science Reviews* 131.1 (Aug. 2007), pp. 417–450. ISSN: 0038-6308, 1572-9672. DOI: 10.1007/s11214-007-9246-7.

[And+10] Brian J. Anderson et al. "The Magnetic Field of Mercury". In: *Space Science Reviews* 152.1 (May 2010), pp. 307–339. ISSN: 0038-6308, 1572-9672. DOI: 10.1007/s11214-009-9544-3.

[BCB14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *arXiv: 1409.0473* [*cs, stat*] (Sept. 1, 2014). arXiv: 1409.0473.

[BBL09] Maria-Florina Balcan, Alina Beygelzimer, and John Langford. "Agnostic active learning". In: *Journal of Computer and System Sciences* 75.1 (Jan. 2009), pp. 78–89. ISSN: 00220000. DOI: 10.1016/j.jcss.2008.07.003.

[Bal+99] Pierre Baldi et al. "Exploiting the past and the future in protein secondary structure prediction". In: *Bioinformatics* 15.11 (Nov. 1, 1999), pp. 937–946. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/15.11.937.

[BFS93] Y. Bengio, P. Frasconi, and P. Simard. "The problem of learning long-term dependencies in recurrent networks". In: *IEEE International Conference on Neural Networks*. IEEE International Conference on Neural Networks. San Francisco, CA, USA: IEEE, 1993, pp. 1183–1188. ISBN: 978-0-7803-0999-9. DOI: 10.1109/ICNN.1993.298725.

[BSF94] Y. Bengio, P. Simard, and P. Frasconi. "Learning long-term dependencies with gradient descent is difficult". In: *IEEE Transactions on Neural Networks* 5.2 (Mar. 1994), pp. 157–166. ISSN: 1045-9227, 1941-0093. DOI: 10.1109/72.279181.

[Ben+10] Johannes Benkhoff et al. "BepiColombo—Comprehensive exploration of Mercury: Mission overview and science goals". In: *Planetary and Space Science* 58.1 (Jan. 2010), pp. 2–20. ISSN: 00320633. DOI: 10.1016/j.pss.2009.09.020.

[Bro+20] Tom B. Brown et al. "Language Models are Few-Shot Learners". In: *arXiv: 2005.14165* [*cs*] (July 22, 2020). arXiv: 2005.14165.

[Cau47] Augustin-Louis Cauchy. "Méthode générale pour la résolution des systémes d'équations simultanées". In: *Comptes Rendus de l'Academie des Science* 25 (1847), pp. 536–538.

[Chi70] Aleksandr Ja Chinčin. *Mathematical foundations of information theory*. Dover ed., new transl. Dover books on intermediate and advanced mathematics. New York: Dover Publ, 1970. 120 pp. ISBN: 978-0-486-60434-3.

[Cho+14] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. EMNLP 2014. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734. DOI: 10.3115/v1/D14-1179.

[Cur44] Haskell B. Curry. "The method of steepest descent for non-linear minimization problems". In: *Quarterly of Applied Mathematics* 2.3 (1944), pp. 258–261. ISSN: 0033-569X, 1552-4485. DOI: 10.1090/qam/10667.

[Cyb89] G. Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals, and Systems* 2.4 (Dec. 1989), pp. 303–314. ISSN: 0932-4194, 1435-568X. DOI: 10.1007/BF02551274.

[Dev+19] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. NAACL-HLT 2019. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423.

[Die02] Thomas G. Dietterich. "Machine Learning for Sequential Data: A Review". In: *Structural, Syntactic, and Statistical Pattern Recognition*. Ed. by Terry Caelli et al. Red. by G. Goos, J. Hartmanis, and J. van Leeuwen. Vol. 2396. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 15–30. ISBN: 978-3-540-44011-6 978-3-540-70659-5. DOI: 10.1007/3-540-70659-3_2.

[Doz16] Timothy Dozat. "Incorporating Nesterov Momentum into Adam". In: 2016.

[DHS11] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *The Journal of Machine Learning Research* 12 (null July 1, 2011), pp. 2121–2159. ISSN: 1532-4435.

[Elm90] Jeffrey L. Elman. "Finding Structure in Time". In: *Cognitive Science* 14.2 (Mar. 1990), pp. 179–211. ISSN: 03640213. DOI: 10.1207/s15516709cog1402_1.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. Adaptive computation and machine learning. Cambridge, Massachusetts: The MIT Press, 2016. 775 pp. ISBN: 978-0-262-03561-3.

[Goo+13] Ian Goodfellow et al. "Maxout Networks". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research. Issue: 3. Atlanta, Georgia, USA: PMLR, June 17, 2013, pp. 1319–1327.

[GMH13] Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. "Speech recognition with deep recurrent neural networks". In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing* (2013), pp. 6645–6649.

[GS05] Alex Graves and Jürgen Schmidhuber. "Framewise phoneme classification with bidirectional LSTM and other neural network architectures". In: *Neural Networks* 18.5 (July 2005), pp. 602–610. ISSN: 08936080. DOI: 10.1016/j.neunet.2005.06.042.

[Gra+09] Alex Graves et al. "A Novel Connectionist System for Unconstrained Handwriting Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31.5 (May 2009). Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 855–868. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2008.137.

[He+15a] Guoliang He et al. "Active Learning for Multivariate Time Series Classification with Positive Unlabeled Data". In: *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)* (2015). DOI: 10.1109/ICTAI.2015.38.

[He+15b] Kaiming He et al. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *2015 IEEE International Conference on Computer Vision (ICCV)* (2015), pp. 1026–1034.

[He+16] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). ISSN: 1063-6919. June 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[HG16] Dan Hendrycks and Kevin Gimpel. *Gaussian Error Linear Units (GELUs)*. Publication Title: arXiv e-prints ADS Bibcode: 2016arXiv160608415H Type: article. June 1, 2016.

[Hin12] Geoffrey Hinton. *Neural networks for machine learning*. 2012.

[Hoc91] Sepp Hochreiter. "Untersuchungen zu dynamischen neuronalen Netzen". In: *Master's thesis, Institut für Informatik, Technische Universität, München* 1 (1991).

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1, 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5 (Jan. 1989), pp. 359–366. ISSN: 08936080. DOI: 10.1016/0893-6080(89)90020-8.

[Hua+20] Lei Huang et al. "Normalization Techniques in Training DNNs: Methodology, Analysis and Application". In: *arXiv:2009.12836 [cs, stat]* (Sept. 27, 2020). arXiv: 2009.12836.

[Hyo96] Heikki Hyotyniemi. *Turing Machines are Recurrent Neural Networks*. 1996.

[IS15] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 7, 2015, pp. 448–456.

[Ism+19] Hassan Ismail Fawaz et al. "Deep learning for time series classification: a review". In: *Data Mining and Knowledge Discovery* 33.4 (July 2019), pp. 917–963. ISSN: 1384-5810, 1573-756X. DOI: 10.1007/s10618-019-00619-1.

[Jar+09] Kevin Jarrett et al. "What is the best multi-stage architecture for object recognition?" In: *2009 IEEE 12th International Conference on Computer Vision*. 2009 IEEE 12th International Conference on Computer Vision. ISSN: 2380-7504. Sept. 2009, pp. 2146–2153. DOI: 10.1109/ICCV.2009.5459469.

[Jin+17] Chi Jin et al. "How to Escape Saddle Points Efficiently". In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, Aug. 6, 2017, pp. 1724–1732.

[Joh+12] Catherine L. Johnson et al. "MESSENGER observations of Mercury's magnetic field structure: MERCURY'S MAGNETIC FIELD STRUCTURE". In: *Journal of Geophysical Research: Planets* 117 (E12 Dec. 2012), n/a–n/a. ISSN: 01480227. DOI: 10.1029/2012JE004217.

[Kao+18] Chieh-Chi Kao et al. "R-CRNN: Region-based Convolutional Recurrent Neural Network for Audio Event Detection". In: *Interspeech 2018*. Interspeech 2018. ISCA, Sept. 2, 2018, pp. 1358–1362. DOI: 10.21437/Interspeech.2018-2323.

[KW52] J. Kiefer and J. Wolfowitz. "Stochastic Estimation of the Maximum of a Regression Function". In: *The Annals of Mathematical Statistics* 23.3 (Sept. 1952), pp. 462–466. ISSN: 0003-4851. DOI: 10.1214/aoms/1177729392.

[KB15] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015.

[KR95] M. G Kivelson and C. T Russell. *Introduction to space physics*. OCLC: 1124679918. 1995. ISBN: 978-1-139-87829-6.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012.

[Lai+18] Guokun Lai et al. "Modeling Long- and Short-Term Temporal Patterns with Deep Neural Networks". In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. SIGIR '18: The 41st International ACM SIGIR conference on research and development in Information Retrieval. Ann Arbor MI USA: ACM, June 27, 2018, pp. 95–104. ISBN: 978-1-4503-5657-2. DOI: 10.1145/3209978.3210006.

[Lav+20] Alexander Lavrukhin et al. *Automatic detection of magnetopause and bow shock crossing signatures in MESSENGER magnetometer data*. other. oral, Oct. 8, 2020. DOI: 10.5194/epsc2020-826.

[Lav+21a] Alexander Lavrukhin et al. *Automatic detection of magnetopause and bow shock crossing signatures in MESSENGER magnetometer data using Convolutional Neural Networks.* other. pico, Mar. 4, 2021. DOI: 10.5194/egusphere-egu21-12703.

[Lav+21b] Alexander Lavrukhin et al. *Determination of magnetopause and bow shock shape based on convolutional neural network modelling of MESSENGER data*. other. oral, July 22, 2021. DOI: 10.5194/epsc2021-651.

[LeC89] Yann LeCun. *Generalization and Network Design Strategies*. Technical Report CRG-TR-89-4. University of Toronto Connectionist Research Group, June 1989.

[LeC+98] Yann LeCun et al. "Efficient BackProp". In: *Neural Networks: Tricks of the Trade*. Berlin, Heidelberg: Springer-Verlag, Jan. 1, 1998, pp. 9–50. ISBN: 978-3-540-65311-0.

[Lee+16] Jason D. Lee et al. "Gradient Descent Only Converges to Minimizers". In: *29th Annual Conference on Learning Theory*. Ed. by Vitaly Feldman, Alexander Rakhlin, and Ohad Shamir. Vol. 49. Proceedings of Machine Learning Research. Columbia University, New York, New York, USA: PMLR, June 23, 2016, pp. 1246–1257.

[Les+93] Moshe Leshno et al. "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function". In: *Neural Networks* 6.6 (Jan. 1993), pp. 861–867. ISSN: 08936080. DOI: 10.1016/S0893-6080(05)80131-5.

[Li+18] Shuai Li et al. "Independently Recurrent Neural Network (IndRNN): Building A Longer and Deeper RNN". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. ISSN: 2575-7075. June 2018, pp. 5457–5466. DOI: 10.1109/CVPR.2018.00572.

[LZ21] Bryan Lim and Stefan Zohren. "Time-series forecasting with deep learning: a survey". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 379.2194 (Apr. 5, 2021), p. 20200209. ISSN: 1364-503X, 1471-2962. DOI: 10.1098/rsta.2020.0209.

[Lin+17] Tsung-Yi Lin et al. "Focal Loss for Dense Object Detection". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017 IEEE International Conference on Computer Vision (ICCV). ISSN: 2380-7504. Oct. 2017, pp. 2999–3007. DOI: 10.1109/ICCV.2017.324.

[Liu+20] Liyuan Liu et al. "On the Variance of the Adaptive Learning Rate and Beyond". In: *ICLR* (2020).

[LPM15] Thang Luong, Hieu Pham, and Christopher D. Manning. "Effective Approaches to Attention-based Neural Machine Translation". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. EMNLP 2015. Lisbon, Portugal: Association for Computational Linguistics, Sept. 2015, pp. 1412–1421. DOI: 10.18653/v1/D15-1166.

[Maa13] Andrew L. Maas. "Rectifier Nonlinearities Improve Neural Network Acoustic Models". In: ICML Workshop on Deep Learning for Audio, Speech, and Language Processing. 2013.

[Mao+14] Junhua Mao et al. "Explain Images with Multimodal Recurrent Neural Networks". In: *arXiv:1410.1090 [cs]* (Oct. 4, 2014). arXiv: 1410.1090.

[MKH19] Rafael Müller, Simon Kornblith, and Geoffrey Hinton. "When does label smoothing help?" In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. 422. Red Hook, NY, USA: Curran Associates Inc., Dec. 8, 2019, pp. 4694–4703.

[NH10] Vinod Nair and Geoffrey E. Hinton. "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Madison, WI, USA: Omnipress, June 21, 2010, pp. 807–814. ISBN: 978-1-60558-907-7.

[NAS19] NASA. *In Depth: MESSENGER*. NASA Solar System Exploration. 2019. URL: https://solarsystem.nasa.gov/missions/messenger/in-depth (visited on 10/17/2021).

[Nes+74] N. F. Ness et al. "Magnetic Field Observations near Mercury: Preliminary Results from Mariner 10". In: *Science* 185.4146 (July 12, 1974), pp. 151–160. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.185.4146.151.

[Nes83] Yurii Nesterov. "A method for unconstrained convex minimization problem with the rate of convergence o(1/k^2)". In: 1983.

[Ngu+18] Van Quan Nguyen et al. "Applications of Anomaly Detection Using Deep Learning on Time Series Data". In: *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*. 2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech). Athens: IEEE, Aug. 2018, pp. 393–396. ISBN: 978-1-5386-7518-2. DOI: 10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00078.

[Par+21] David Parunakian et al. *MESSENGER magnetometer and coordinates prepared dataset*. Type: dataset. Nov. 29, 2021. DOI: 10.5281/zenodo.5733418.

[Pas+19] Adam Paszke et al. "PyTorch: an imperative style, high-performance deep learning library". In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. 721. Red Hook, NY, USA: Curran Associates Inc., Dec. 8, 2019, pp. 8026–8037.

[Pea00] Stanton Peale. *The MESSENGER Orbiter Mission to Mercury*. Nov. 1, 2000.

[PLN17] Fengchao Peng, Qiong Luo, and Lionel M. Ni. "ACTS: An Active Learning Method for Time Series Classification". In: *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. 2017 IEEE 33rd International Conference on Data Engineering (ICDE). ISSN: 2375-026X. Apr. 2017, pp. 175–178. DOI: 10.1109/ICDE.2017.68.

[Phi+20a] L. C. Philpott et al. "The Shape of Mercury's Magnetopause: The Picture From MESSENGER Magnetometer Observations and Future Prospects for BepiColombo". In: *Journal of Geophysical Research: Space Physics* 125.5 (May 2020). ISSN: 2169-9380, 2169-9402. DOI: 10.1029/2019JA027544.

[Phi+20b] Lydia Philpott et al. *MESSENGER bowshock and magnetopause crossings*. In collab. with Lydia Philpott. Type: dataset. 2020. DOI: 10.5683/SP2/1U6FE0.

[Pol64] B. T. Polyak. "Some methods of speeding up the convergence of iteration methods". In: *USSR Computational Mathematics and Mathematical Physics* 4.5 (Jan. 1, 1964), pp. 1–17. ISSN: 0041-5553. DOI: 10.1016/0041-5553(64)90137-5.

[PJ92] B. T. Polyak and A. B. Juditsky. "Acceleration of Stochastic Approximation by Averaging". In: *SIAM Journal on Control and Optimization* 30.4 (July 1992), pp. 838–855. ISSN: 0363-0129, 1095-7138. DOI: 10.1137/0330046.

[Ren+20] Pengzhen Ren et al. "A Survey of Deep Active Learning". In: *arXiv: 2009.00236 [cs, stat]* (Aug. 30, 2020). arXiv: 2009.00236.

[Ros58] F. Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain." In: *Psychological Review* 65.6 (1958), pp. 386–408. ISSN: 1939-1471, 0033-295X. DOI: 10.1037/h0042519.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (Oct. 1986), pp. 533–536. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/323533a0.

[SP97] M. Schuster and K.K. Paliwal. "Bidirectional recurrent neural networks". In: *IEEE Transactions on Signal Processing* 45.11 (Nov. 1997). Conference Name: IEEE Transactions on Signal Processing, pp. 2673–2681. ISSN: 1941-0476. DOI: 10.1109/78.650093.

[Sel+17] Sreelekshmy Selvin et al. "Stock price prediction using LSTM, RNN and CNN-sliding window model". In: *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI). Sept. 2017, pp. 1643–1647. DOI: 10.1109/ICACCI.2017.8126078.

[Set09] Burr Settles. *Active Learning Literature Survey*. Technical Report 1648. Madison, Wisconsin: University of Wisconsin–Madison, Jan. 9, 2009.

[Set12] Burr Settles. "Active Learning". In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 6.1 (June 30, 2012), pp. 1–114. ISSN: 1939-4608, 1939-4616. DOI: 10.2200/S00429ED1V01Y201207AIM018.

[SCR07] Burr Settles, Mark Craven, and Soumya Ray. "Multiple-instance active learning". In: *Proceedings of the 20th International Conference on Neural Information Processing Systems*. NIPS'07. Red Hook, NY, USA: Curran Associates Inc., Dec. 3, 2007, pp. 1289–1296. ISBN: 978-1-60560-352-0.

[Sha48] C. E. Shannon. "A mathematical theory of communication". In: *The Bell System Technical Journal* 27.3 (July 1948). Conference Name: The Bell System Technical Journal, pp. 379–423. ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1948.tb01338.x.

[SS95] H.T. Siegelmann and E.D. Sontag. "On the Computational Power of Neural Nets". In: *Journal of Computer and System Sciences* 50.1 (Feb. 1995), pp. 132–150. ISSN: 00220000. DOI: 10.1006/jcss.1995.1013.

[SS91] Hava T. Siegelmann and Eduardo D. Sontag. "Turing computability with neural nets". In: *Applied Mathematics Letters* 4.6 (1991), pp. 77–80. ISSN: 08939659. DOI: 10.1016/0893-9659(91)90080-F.

[Sla04] J.A. Slavin. "Mercury's magnetosphere". In: *Advances in Space Research* 33.11 (Jan. 2004), pp. 1859–1874. ISSN: 02731177. DOI: 10.1016/j.asr.2003.02.019.

[Smi18] Leslie N. Smith. "A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay". In: *arXiv:1803.09820 [cs, stat]* (Apr. 24, 2018). arXiv: 1803.09820.

[Sri+14] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958.

[SL11] Patrick Sudowe and Bastian Leibe. "Efficient Use of Geometric Constraints for Sliding-Window Object Detection in Video". In: *Computer Vision Systems*. Ed. by James L. Crowley, Bruce A. Draper, and Monique Thonnat. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2011, pp. 11–20. ISBN: 978-3-642-23968-7. DOI: 10.1007/978-3-642-23968-7_2.

[SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks". In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'14. Cambridge, MA, USA: MIT Press, Dec. 8, 2014, pp. 3104–3112.

[Vas+17] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc., 2017.

[Wik14] Wikimedia Commons. *Precision and Recall*. Nov. 22, 2014. URL: https://commons.wikimedia.org/wiki/File:Precisionrecall.svg (visited on 01/08/2022).

[Wik18] Wikimedia Commons. *LSTM*. Dec. 26, 2018. URL: https://commons.wikimedia.org/wiki/File:LSTM.jpg (visited on 01/06/2022).

[Wik19] Wikimedia Commons. *Neuron description*. Mar. 17, 2019. URL: https://commons.wikimedia.org/wiki/File:Neuron.svg (visited on 01/05/2022).

[Wil+17] Ashia C. Wilson et al. "The marginal value of adaptive gradient methods in machine learning". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., Dec. 4, 2017, pp. 4151–4161. ISBN: 978-1-5108-6096-4.

[Win+13] Reka M. Winslow et al. "Mercury's magnetopause and bow shock from MESSENGER Magnetometer observations". In: *Journal of Geophysical Research: Space Physics* 118.5 (May 2013), pp. 2213–2227. ISSN: 21699380. DOI: 10.1002/jgra.50237.

[YK19] Donggeun Yoo and In So Kweon. "Learning Loss for Active Learning". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Long Beach, CA, USA: IEEE, June 2019, pp. 93–102. ISBN: 978-1-72813-293-8. DOI: 10.1109/CVPR.2019.00018.

[YK16] Fisher Yu and Vladlen Koltun. "Multi-Scale Context Aggregation by Dilated Convolutions". In: *arXiv:1511.07122 [cs]* (Apr. 30, 2016). arXiv: 1511.07122.

[Zer+21] George Zerveas et al. "A Transformer-based Framework for Multivariate Time Series Representation Learning". In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. Virtual Event Singapore: ACM, Aug. 14, 2021, pp. 2114–2124. ISBN: 978-1-4503-8332-5. DOI: 10.1145/3447548.3467401.

[Zha+21] Chiyuan Zhang et al. "Understanding deep learning (still) requires rethinking generalization". In: *Communications of the ACM* 64.3 (Mar. 2021), pp. 107–115. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/3446776.

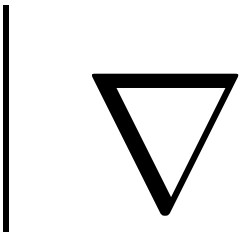
[ZD20] Zao Zhang and Yuan Dong. "Temperature Forecasting via Convolutional Recurrent Neural Networks Based on Time-Series Data". In: *Complexity* 2020 (Mar. 20, 2020). Publisher: Hindawi, e3536572. ISSN: 1076-2787. DOI: 10.1155/2020/3536572.

[Zho+15] J. Zhong et al. "Mercury's three-dimensional asymmetric magnetopause". In: *Journal of Geophysical Research: Space Physics* 120.9 (Sept. 2015), pp. 7658–7671. ISSN: 2169-9380, 2169-9402. DOI: 10.1002/2015JA021425.

[ZPT17] Martin Zihlmann, Dmytro Perekrestenko, and Michael Tschannen. "Convolutional recurrent neural networks for electrocardiogram classification". In: *2017 Computing in Cardiology (CinC)*. 2017 Computing in Cardiology (CinC). ISSN: 2325-887X. Sept. 2017, pp. 1–4. DOI: 10.22489/CinC.2017.070-060.

[Zur+11] T. H. Zurbuchen et al. "MESSENGER Observations of the Spatial Distribution of Planetary Ions Near Mercury". In: *Science* 333.6051 (Sept. 30, 2011), pp. 1862–1865. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.1211302.

Affidavit



Hiermit versichere ich, dass ich diese Bachelorarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe und alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, als solche gekennzeichnet sind, sowie, dass ich diese Bachelorarbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt habe.

Passau, den 28.01.2022

Ort, Datum

Nikolas Kirschstein