Convolutional Neural Networks in space weather predictions

Master's thesis
Iina Leppänen
Y52840513
Research Unit of Mathematical Sciences
University of Oulu
July 23, 2023

Tiivistelmä

Nykyaikainen elämämme maapallolla on haavoittuvainen avaruussään vaikutuksille. Myrskyinen avaruussää voi muun muuassa aiheuttaa häiriötä satelliittiliikenteeseemme, ja siksi Maan lähiavaruuden sähkövirtojen ennustaminen on tärkeää. Vaikka virtojen yleiset piirteet ovat tiedossa, niiden pienemmän skaalan käyttäytymisen ennustaminen voi olla hankalaa. Tässä pro gradu -tutkielmassa toteutetaan kaksi erilaista konvoluutioneuroverkkoa ennustamaan ionosfäärin sähkövirtojen käyttäytymistä aikasarjan pohjalta. Konvoluutioneuroverkot ovat erikoistapaus keinotekoisista neuroverkoista, koneoppimismenetelmistä, jotka kykenevät oppimaan hyvinkin monimutkaisia piirteitä saamastaan syötteestä.

Tässä tutkielmassa tutustutaan lyhyesti Maan lähiavaruuden sähkövirtoihin ja niiden takana vaikuttaviin perusvoimiin. Tämän lisäksi käydään läpi neuroverkkojen teoriaa ja keskitytään erityisesti konvoluutioneuroverkkoihin. Ennustamiseen käytetyistä arkkitehtuureista ensimmäinen on ResNet, joka hyödyntää niin kutsuttua residuaalista oppimista ja päättyy täysin kytkettyyn kerrokseen. Toinen arkkitehtuuri on U-Net, niin sanottu enkoodaus-dekoodaus arkkitehtuuri, joka käsittelee saamaansa syötettä usealla resoluutiolla ja päättyy konvoluutiokerrokseen. Molemmat arkkitehtuurit osoittautuvat kykeniviksi ennustamaan ionosfäärivirtojen käyttätymistä, joskin ennusteissa esiintyy virtojen voimakkuuden aliarviointia. Lisäksi vaikuttaa siltä, että U-Net muodostaa liian vahvan avaruudellisen riippuvuuden syötteen yksittäisten elementtien välille, mikä näkyy ResNetin ennusteita sileämpinä tulosteina ja viittaisi viimeisen täysin kytketyn kerroksen tärkeyteen.

Abstract

Our modern life on Earth is vulnerable to the effects of space weather, as it can for example disturb our satellites. Thus being able to predict the electric currents flowing in the near-Earth space is of importance. However, while the large scale features of these currents are known, their small scale behaviour can be harder to predict. In this thesis two different convolutional neural network architectures are implemented to predict ionospheric currents from time-series input. Convolutional neural networks are a special case of artificial neural networks, a machine learning approach capable of learning complicated features from the input data.

In this thesis a brief introduction to the current systems in the near-Earth space and the fundamental forces behind them is given. In addition to this the theory of artificial neural networks is introduced, with a special interest in convolutional neural networks. The first one of the two implemented neural networks is ResNet, utilising so called residual learning and ending in a linear layer. The other network is U-Net, a fully-convolutional encoder-decoder architecture investigating the input at multiple resolutions. Both architectures are capable of predicting the behaviour of the ionospheric currents, but tend to underestimate the strength of the largest currents. It also appears that U-Net creates too strong spatial dependency between the components of the input resulting in overly smooth predictions. This indicates the importance of the final fully-connected layer.

Foreword

This thesis was done in association with a research project. Thus I would like to thank all the research members for their insights and discussion, including Anita Aikio, Timo Asikainen, Ilkka Virtanen, Marcus Pedersen and Mikko Sillanpää. I would especially like to thank Jemina Manninen for providing me the interpolated data and some of the code used for plotting in Section 5.3, and Heikki Vanhamäki for his feedback on Section 2. I would also like to thank the group members Snizhana Ross and Soléne Four for providing part of the code used to plot the results in Section 5.3. And last but certainly not least I would like to thank final group member and my supervisor Andreas Hauptmann for his guidance and suggestions that have helped to shape this thesis.

Contents

1 Introduction **6**

2 Space weather **8**
2.1 Electric current . . . 8
2.2 Solar wind, magnetosphere and ionosphere . . . 9
2.3 Current systems in ionosphere and magnetosphere . . . 10
2.3.1 Ring currents . . . 12
2.3.2 Field-aligned currents . . . 13
2.3.3 Pedersen, Hall and equivalent currents . . . 14

3 Deep learning **15**
3.1 Learning framework . . . 15
3.2 An Artificial Neural Network . . . 17
3.2.1 Activation . . . 19
3.2.2 Depth and width . . . 22
3.3 Optimisation . . . 29
3.3.1 Gradient Descent . . . 30
3.3.2 Introducing stochasticity . . . 31
3.3.3 Adam . . . 32
3.4 Loss functions and regularisation . . . 33

4 Convolutional Neural Networks **36**
4.1 Convolution . . . 36
4.2 ResNet . . . 39
4.2.1 Skip connection . . . 39
4.3 U-Net . . . 41
4.3.1 Downsampling . . . 41
4.3.2 Upsampling . . . 44

5 Predicting space weather **45**
5.1 Data . . . 45
5.1.1 Preparation . . . 47
5.2 Model architectures and training . . . 48
5.2.1 ResNet . . . 49
5.2.2 U-Net . . . 50
5.3 Results . . . 52

6 Discussion **63**
6.1 Architecture and hyperparameters 63
6.2 Quantitative analysis 64
6.3 Qualitative analysis 65

7 Conclusions **67**

1 Introduction

Even though not always visible to us, the near-Earth space is filled with electric currents. These electric currents are driven greatly by the activity of the Sun, which can subsequently have significant, even dramatic effects on our modern life on Earth. At highly active times, the Sun can for example cause disturbances in satellite navigation services, radio communications or even cause a blackout in a power grid [26]. The activity of the Sun, space weather, can also manifest itself in auroras. The ability to model and predict the space weather, or the electric currents in the near-Earth space or upper parts of the Earth's atmosphere, would help to prepare for these events. While the large-scale structures of these currents are known to an extent, the smaller and more dynamic structures are harder to model. Multiple different modeling techniques have been used, from physics-based models to the empirical and data-driven ones.

In this thesis a data-driven approach to predict the space weather is taken. More precisely a form of machine learning called the artificial neural network (ANN) is used to predict the behaviour of ionospheric currents. ANNs in general require large amounts of data, which is combined into complex functions and finally into a model within the ANN. This model construction is done algorithmically and with minimal human intervention as is typical for machine learning approaches. Today, deep ANNs have been used to various tasks on many fields, their successes ranging from winning games to helping in protein design [38, 19]. However, the drawbacks of ANNs are their typical black-box nature, as it is not usually clear to why the ANN produces the output it does, their need for a large amount of data, and the expensiveness of the training in terms of required computational power and memory capacity.

Convolutional neural networks (CNNs) are a particular family of ANNs utilising a specific mathematical operation called convolution. The benefits of CNNs over the regular fully-connected neural networks comprise of smaller memory and computational capacity requirements and their ability to detect features anywhere within the input by design. While CNNs are particularly suited for image-processing tasks, such as image classification, denoising or segmentation, they have also been used for other signal processing tasks and time-series data. In this thesis two different convolutional neural networks will be trained to predict the ionospheric currents from a set of time-series data.

The first of these architectures is ResNet, a neural network utilising residual learning and the other is U-Net, an encoder-decoder architecture. As implemented for this thesis, the ResNet architecture does not extract features at multiple scales, as is done in the U-Net. But possibly the most important difference between these architectures is their ending layer. ResNet ends with a linear layer, meaning it takes a linear combination of all the nodes in the previous layer, while U-Net is implemented as a fully convolutional network, meaning it maintains the location information encoding stronger spatial dependencies between neighbouring nodes.

To this end, this thesis will first describe shortly some of the basic phenomena behind the electric currents, and give a brief introduction to the near-Earth space and the currents flowing there in Section 2. Then the general features of the feed-forward ANNs and their training are described in Section 3. In Section 4 the convolution operation and the basic features of CNNs will be introduced together with a general description of the two neural network architectures that will be used for the prediction. Finally, in Section 5 the practical implementation will be described together with the obtained results, and in Section 6 these results will be discussed. In Section 7 the final remarks will be made.

2 Space weather

Space weather is a term used to describe the phenomena in the near-Earth space and Earth's upper atmosphere caused by the activity of the Sun. The outflow of charged particles from the Sun, and their interaction with the magnetic field and the atmosphere of the Earth causes electric currents to flow in the near-Earth space and the upper parts of Earth's atmosphere. Ultimately the currents can be explained through electromagnetism, but their behaviour can be more easily explained through plasma physics. These subjects will be discussed briefly in the following subsection.

2.1 Electric current

An *electric current* is, similar to other currents such as ocean currents, a flow of particles. But whereas a neutral current is described as net shift of mass, an electric current is described as net flow of charge (*current density*), $\mathbf{j} = \rho\mathbf{v}$, where ρ is the charge density and v is the speed. On Earth electric currents flow in for example power cords, or they can be seen in lightning strikes. In space electric currents flow in *plasmas*, substance constituting of electrons and ions. To understand better the way electric currents are created, it is useful to understand the interplay between electric and magnetic fields. This description is based on [37].

An *electric charge* denoted with q is an attribute of some particles, such as protons and electrons. Protons and electrons, together with neurons that have no charge, make up the atoms, building blocks of (almost) everything we see around us. A particle with a charge produces an *electric field* $\mathbf{E}$, and another particle with a charge will experience that field as an *electric force* $\mathbf{F_E} = q\mathbf{E}$. The strength of the field (and subsequently the force) depends inversely from the square of distance on the source of the field, meaning the field is stronger closer to the source. A charge can be either positive or negative, and if two particles have both positive or both negative charges they repell one another, whereas two particles with opposite charges attract.

When electric charge carriers (particles for example) are in motion, they exert an another field, namely the *magnetic field* $\mathbf{B}$. Similarly to electric fields, another charged particle in motion with charge q and speed $\mathbf{v}$ will experience this field as a *magnetic force* $\mathbf{F_B} = q\mathbf{v} \times \mathbf{B}$, while introducing a magnetic field of its own. Thus an electric current, being a flow of charge, induces a magnetic field of its own and is affected by one. Magnetic fields are always divergence free, meaning that over any closed surface there are as many entering field lines as there are leaving. The interaction between magnetism and electrostatics does not, however, stop there. A change in magnetic field will induce an electric field and, assuming that the field is strong enough to make charges flow, an electric current, and then this electric current will create a magnetic field. To be precise, already the changing electric field produces a magnetic field.

While it is possible to describe the motions of the single charged particles using the simple interactions between them, this description becomes overly complex quickly when the number of particles in the system grows. Thus it is usually more practical to describe the overall behaviour of the electric currents or, as would be more relevant for this thesis, the plasmas in space. This thesis will not, however, dive into the exact description of the mechanisms of plasmas, but will instead focus on the general features and behaviours of the electric currents flowing in near-Earth space. But to understand what drives these currents, even on a shallow level, it is important to know what is happening in the near-Earth space.

2.2 Solar wind, magnetosphere and ionosphere

The Sun is a star of average luminosity and size. It provides a source for steady radiation, without which life on Earth would not be possible. This radiation is powered by the fusion reaction in the core of the Sun combining light-weight elements into heavier ones. See for example [32] for a description of the reaction chain. *Solar wind* is the flow of charged particles, electrons and ions, from the Sun. It can be divided into two parts, namely the *slow* and the *fast* solar wind based on their origin and characteristics as described for example in [36]. The slow solar wind comes from strong closed magnetic field loops on the surface of the Sun, and the particles forming the fast solar wind escape along radial magnetic field lines called *coronal holes*.

The magnetic field of the Sun, the *solar magnetic field*, can have a very varying shape. While at times it appears dipolar close to Sun, it changes with the activity of the Sun and can be highly structured as shown in [36]. The solar wind drags out this solar magnetic field into the interplanetary space creating the *interplanetary magnetic field*, or IMF for short. The IMF is distorted into a spiral by the rotation of the Sun. The Sun also emits charged particles in bursts such as *flares* and dense plasma clouds called *coronal mass ejections* (CME). These bursts occur above *sunspots*, places of strong localised magnetic fields that appear as darker patches on the Sun. Sunspots speak of the activity of the Sun, with more sunspots at more active times. This activity changes in approximately eleven year cycles during which the Sun's activity goes from minimum to maximum and back. These cycles are known as the *solar cycles* and the polarity of the Sun, and thus the polarity of the interplanetary magnetic field, is reversed during each cycle.

The magnetic field of the Earth, the *geomagnetic field*, is affected greatly by the activity of the Sun [36]. While the geomagnetic field is near an ideal dipole close to the Earth, it is highly distorted by the solar wind at larger distances giving it a comet-like shape. The pressure from the solar wind compresses the sunward side of the geomagnetic field and elongates the side away from the Sun into a tail structure. The supersonic solar wind hitting the geomagnetic field creates a free-standing shock wave called a *bow shock*. Beneath this surface is the *magnetosheath*, where the particles are decelerated, heated and deflected causing magnetic perturbations throughout the area. *Magnetopause* is a boundary layer that separates the magnetised solar wind plasma in the magnetosheath from that confined in the Earth's magnetic field. Beneath the magnetopause is the *magnetosphere*, a vast region where the geomagnetic field dominates. There the plasma originates from both the solar wind and the Earth.

The *ionosphere* is the ionised part of the upper atmosphere of the Earth beneath the magnetosphere and it extends from about 60 to beyond 1000 kilometres according to [36]. The charged particles in ionosphere are mainly produced by photoionisation, where radiation from the Sun makes a neutral molecule loose an electron, or by energetic particles from the magnetosphere and the Sun. These particles can then further interact with other particles, or escape the ionosphere at high latitudes as so called *polar wind* along the geomagnetic field lines. Similarly the solar wind can access the atmosphere of the Earth directly at so called cusps (or *polar cusp* at northern hemisphere) along the field lines. The solar wind-magnetosphere interaction causes plasma to flow away from the Sun over the polar cap in the ionosphere and back equatorward of so called *auroral ovals* that surround the cusp regions. Poleward from the auroral oval is the *polar cap* (on northern hemisphere).

2.3 Current systems in ionosphere and magnetosphere

Before diving into the current systems in the near-Earth space, it is important to understand the directions used to describe events happening in this space. The x-axis is chosen to point directly from the Earth to Sun, with Earth in the origin. The half of the Earth on the positive x axis is on the *day side* as the Sun shines on it, and the *night side* is the half on the negative x axis. The z axis is somewhat parallel with the Earth's rotational axis intersecting the sunlight terminator and pointing approximately towards the north pole. As the Earth spins counter-clockwise around its axis, the left half of the Earth (in the direction of the x axis) is rotating to noon and the right half to midnight. These halves are deemed the *dawn* and *dusk sides*, respectively. *Dawn* is the point between day and night side on the post-midnight sector, and *dusk* is similar point on the pre-midnight sector. These areas are described in Figure 1. The *post-midnight sector* is located on the dawn side near midnight and the *pre-midnight sector* is on the dusk side near midnight. The y axis points then from dawn to dusk and is perpendicular to x and z axes.

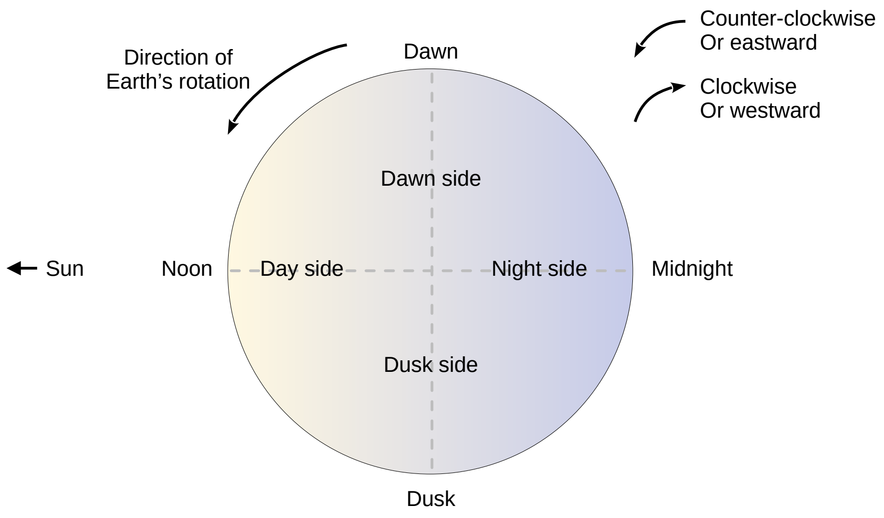

Figure 1: A simplified diagram of different sections of Earth relative to the Sun. The Sun is on the left, and the Earth is viewed on the plane where $z = 0$. See text for details.

The currents flowing in the near-Earth space are higly influenced by the Sun's activity. *Substorms* are quasi-periodic increases in the auroral activity on the nightside lasting a few hours. They can be divided to three phases, namely the *growth phase* in which the auroral oval moves to lower latitudes and polar cap expands, *expansion phase* in which the ionospheric conductance is increased at the ovals and the polar cap contracts, and the *recovery phase* in which the conditions return to a state similar to the one preceding the storm as described in [25, 36]. A *geomagnetic storm* is a significant magnetic perturbation caused by greatly increased energy input from the solar wind. This follows after a *sudden storm commancement* (SSC) which is caused by a sudden change in the solar wind dynamic pressure. This pressure change can occur as a result of a CME or a flare. Geomagnetic storm has an *initial* phase in which the geomagnetic activity starts to increase and electric field grows stonger, the *main* phase, when the energy levels reach their peak, and *recovery* phase during which the storm slowly resides [25, 36].

Some of the large current systems flow mainly at the magnetopause, such as the *Chapman-Ferraro magnetopause current* around the cusps, and the *tail current* at the tail of the magnetopause having closure through the equatorial plane (see [8] for a description of these current systems). This thesis is however more interested of the current systems within the magnetosphere and ionosphere, and next the large scale features of these current systems will be introduced. The description of the current systems will mostly follow [8], except for the ionospheric currents for which [9] is the primary source.

2.3.1 Ring currents

Ring current is an electric current flowing around the Earth at the equatorial plane in magnetosphere in a toroidal shape. The ring current consists of an inner current flowing counter-clockwise (or eastward) and an outer part flowing clockwise (westward). The clockwise flowing ring current tends to be of higher density than the inner one, and it induces a magnetic field opposite to the direction of geomagnetic field. While the ring currents are thought to be symmetric, in practise the currents are more or less asymmetric in terms of current density depending on the space weather. At disturbed times, when the plasma pressure distribution becomes highly asymmetric in the inner magnetosphere, the asymmetric part of the clockwise flowing ring current can be separated as *partial ring current*, having its own closure path through ionosphere. The asymmetric part of the counter-clockwise ring current can be accounted to a *banana current* flowing along both of the symmetric ring currents. The outer clockwise ring current does not flow as one system either, but instead divides to two branches on the dayside. This branching part is known as the *cut-ring current*.

During a geomagnetic storm, the positively charged particles from the tail move through the inner magnetosphere and to the dayside magnetopause. This added pressure increases first the size of the banana current, as most of the clockwise ring current is closed through the inner, counter-clockwise ring current. Then in the main phase the plasma pressure is at its highest further inward, and at that point the partial ring current becomes dominant as the asymmetric part of the outer ring current grows proportionally larger than the inner ring current, and needs a different closure path. Finally at the recovery phase, when the plasma pressure becomes fairly uniform, the symmetric outer ring current becomes stronger.

2.3.2 Field-aligned currents

One of the major contributors to the solar wind-magnetosphere-ionosphere coupling are the *field-aligned currents*, or FACs for short, meaning they transport energy, momentum, charge and mass between magnetosphere and ionosphere. By their name, the field-aligned currents flow somewhat along the geomagnetic field's field lines. Two main regions of the field-aligned current system can be recognised, and some smaller ones.

The *Region 1* FACs flow to magnetopause at the pre-midnight sector, connecting to the magnetopause, and then flow back to the ionosphere on the post-midnigth sector. Part of the currents connect at the dayside and other part connects at the night side of the magnetopause. The field-aligned part of the region 1 currents is at the poleward side of the auroral ovals, with the part connected to dayside flowing along the open field lines, and the other flowing along the closed ones. The *Region 2* field-aligned currents flow equatoward of the region 1 field-aligned current, and closer to Earth. The currents flow away from ionosphere around dawn and back to the ionosphere around dusk. At equatorial plane, the region 2 FACs connect to the partial ring current. The region 2 FACs tend to be smaller than those in region 1.

There are also other field-aligned currents than the two mentioned above. *Region 0* current, located poleward from the auroral oval, is adjacent to region 1, and it can be considered to be a part of the region 1 current. Region 0 current depends largely on the IMF component in the direction of the y axis marked as B_y, and it flows mostly out of ionosphere for positive and into for negative B_y in the northern hemisphere and opposite to this in the southern. The so called *northward Bz* or NBZ current appears during strong northward (or positive) IMF B_z component (component in the direction of z axis), and it flows mostly out of ionosphere at dawn and into at dusk. The *substorm current wedge* or SCW system is associated with magnetospheric substorms flowing into ionosphere at dawnside and out of at dusk. The SCW is a simplified presentation of the current system associated with substorms, and it can form even without a substorm.

The region 1 FAC system increases when the IMF turns southward (B_z component becomes negative). Increase in or change to southward direction in IMF causes the magnetopause shift closer to Earth as the magnetic flux on dayside is eroded away. The transport of the magnetic flux to nightside with the flow increase in high-latitude ionosphere causes the region 1 FACs grow larger. When the IMF turns northward, the region 1 FACs similarly diminish. This is also when the NBZ current appears. The region 2 FACs on the other hand increase during the main phase of geomagnetic storms, as the asymmetric outer ring current closes through them.

2.3.3 Pedersen, Hall and equivalent currents

As said the field-aligned currents flow in and out of the ionosphere at different places. *Pedersen currents* flow in the ionosphere closing the circuit of region 1 and 2 FACs across the auroral zones. They also flow (although less) across the polar cap in the ionosphere, increasing the flow for the region 1 FACs, which come to ionosphere closer to the cap. The electric field in the ionosphere points in the same direction as the Pedersen currents. *Hall currents*, on the other hand, flow perpendicular to the electric field, and constitute the eastward auroral electrojet at dusk and the westward auroral electrojet at dawn, flowing mainly at the auroral oval, where the conductance is typically highest. Weaker Hall currents also flow across the polar cap in sunward direction. See [25] for a schematic of these currents.

In practise it is hard to measure the currents in terms of the charge carriers, and instead the magnetic perturbations they cause are measured. The Pedersen currents are thought to produce only little magnetic perturbations on ground, as the region 1 and region 2 FACs somewhat cancel out their effect. The perturbations caused by the Hall currents on the otherhand can be measured. These measured perturbations can be rotated to produce *equivalent current* maps, as the connection between the direction of the perturbation and that of the associated electric current is known. In practise this is a 90 degrees rotation clockwise, see e.g. [30]. However, these equivalent current maps obviously also include for example the effects of more distant currents. None the less some characteristic patterns have been identified, namely the DP1, DP2 and DPY. DP1 pattern is associated with substorms, showing up mainly at the nightside auroral zone. DP2 pattern is produced by the Hall currents appearing on the dawn and dusk auroral zones and, as the Hall currents like the Pedersen currents are associated to the region 1 and 2 FACs, tells about the strength of the region 1 and 2 FACs. DPY on the other hand is associated with the region 0 FACs.

3 Deep learning

Machine learning is an algorithmic approach to finding a computational model capable of describing empirical data according to [45]. In machine learning especially the construction of the model happens almost without human interaction by optimising the model algorithmically according to the specified criteria. *Artificial neural networks*, or ANNs for short, are a subfield of machine learning where the model construction was inspired by the workings of biological neural network, brain, that is. In these artificial neural networks simple features or functions are combined in a hierarchical way to construct more complicated ones. When a neural network is considered deep enough it is *deep learning*, although sometimes the terms deep learning and neural networks are used interchangeably. It is also not always clear what suffices for a deep network, or how its depth should be computed.

3.1 Learning framework

Like statistical learning, machine, and subsequently deep, learning can be divided into three categories, namely *supervised*, *unsupervised* and *reinforcement learning*, although these categories are not always considered to be completely distinct or all-inclusive scenarios and one can also have combined learning types such as *semi-supervised* and *multi-instance* learning as described in [11]. In supervised learning the data is divided into independent and dependent variables. In this case the difference between the model's prediction and the dependent variable can be computed and optimised. In unsupervised learning (see [11]) there is no dependent variable, but the intention can be for example to cluster the data into similar groups, or find the optimal representation of the data under some constraints. Unsupervised learning can be used to prepare the data for other types of learning. In reinforcement learning (see [20]) the agent (in this case the machine learning system) learns what it should do next based on the current state of its environment and its internal state. The action taken by the agent is followed by a reward, and based on this reward the taken action (or a function) is refined. Supervised learning is the most common learning type also among artificial neural networks, and it is the approach taken in this thesis. Thus next the structure of the data for supervised learning will be defined more closely.

For supervised learning the sample space is of the form $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$. The collected or empirical data, $z \in \mathcal{Z}$, consists of $n \in \mathbb{Z}_+$ *input* variables $x = (x^1, x^2, \dots, x^n) \in \mathcal{X}$ usually referred to also as covariates or explanatory, predictive or independent variables or features depending on the context, and $m \in \mathbb{Z}_+$ *target* variables $y = (y^1, y^2, \dots, y^m) \in \mathcal{Y}$ also known as outcome, response or dependent variables. Next it will be assumed that the data consists of $N \in \mathbb{Z}_+$ observations and a pair $(x, y)_k = (x_k, y_k)$, where $k = 1, 2, \dots, N$, corresponds to each observation. Then all the data is collected into a dataset S_{data} In general and especially in the case of neural networks the input can have very different shapes, and the shape can even be different between observations. This can happen for example when the input consists of images of varying sizes and each observation is one of these images [11]. For simplicity in the future it will be assumed that all of the n input variables obtain a scalar value per observation, i.e., for the kth observation we have a vector $\mathbf{x}_k = (x_k^1, x_k^2, \dots, x_k^n) \in \mathbb{R}^n$. In addition it will be assumed that the target is continuous and also obtains m scalar values for each observation, i.e., $\mathbf{y}_k = (y_k^1, y_k^2, \dots, y_k^m) \in \mathbb{R}^m$, where $k = 1, 2, \dots, N$. It is common to assume that the observations are independent and independently distributed (i.i.d), although in practice this is rarely met [43]. As in [11], the notation p_{data} is used for the data producing distribution.

In machine learning the construction of the model is an iterative process in which the trainable parameters of the chosen machine learning method, such as a neural network, are optimised in step-wise manner according to a chosen metric [45, 11]. This phase is known as the *learning phase* and in the case of ANNs it can also be referred to as *training*. For this phase a subset, denoted with S_{train}, of the complete data is chosen, i.e., $S_{train} \subset S_{data}$. In the next phase the constructed model is *validated*, i.e., the performance of the trained model is measured against new observations, coming from a new subset $S_{validate} \subset S_{data}$. Validation phase is used to improve the *generalisation* of the model, or its success on new observations by tuning so called *hyperparameters*. Hyperparameters are pre-determined before training, and the machine learning algorithm can not optimise them during the training phase. After the hyperparameter tuning, we come back to the learning phase to improve the trainable parameters further. These phases are altered until the model performs well enough, at which point the model is further tested against a third subset $S_{test} \subset S_{data}$. This is called the *testing phase* and it is meant to define the performance of the model.

Eventhough the training and validation phases were described as completely separate above, they are commonly done in partially parallel so that the generalisation ability of the model can be assessed throughout the training and the training can be ended early if necessary. Also, although typically the subset are distinct from each other, i.e., $S_{train} \cap S_{validate} \cap S_{test} = \emptyset$, in some cases the validation set is not completely separate from the training set, but instead a subset of the training set not used for training on that round is used for validation as described for example in [11]. The test set, however, is always separate from the others.

3.2 An Artificial Neural Network

Artificial neural networks are partially based on the idea of modelling the functioning of brains mathematically. As an oversimplified model of brain, ANNs consist of simple computational units called *neurons* stacked together so that the *output* of one neuron is the input of another. Typically these neurons create structures called *layers* in the networks, and the function realised by the neural network is combined from the outputs of these layers. The definition for a neural network to be instroduced is based on the source [5], but also the sources [35] and [13] are utilised. The definition is meant for a *feed-forward* neural network, that is, a neural network with directed connections going only to unseen neurons, as these kinds of networks are of interest here.

In general the *architecture* of a neural network can be presented as a graph in the following way.

Definition 3.1 (Architecture of an Artificial Neural Network)**.** The architecture of an artificial neural network can be written as a graph

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}),$$

where $\mathcal{V} = \{v_1, v_2, \dots, v_k\}$, $k \in \mathbb{Z}_+$, is the set of *nodes* in the neural network and $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ is the set of *edges* or *connections*.

It is required from the set of nodes and set of edges that each element v in $\mathcal{V}$ must be connected to at least one edge $e \in \mathcal{E}$, and each connection is always associated with a scalar $w_e \in \mathbb{R}$ called the *weight*. The set of nodes $\mathcal{V}$ can further be divided into three subsets, namely the *input layer* $\mathcal{I}$ consisting of the inputs x to the network, the *output layer* $\mathcal{O}$ which gives the output $\hat{y}$ of the neural network, and the *hidden layers* $\mathcal{H} = \mathcal{V} \setminus (\mathcal{I} \cap \mathcal{O})$ in between. In addition all the nodes, except for those in the input layer, are commonly associated with an *activation function* $\sigma_v : \mathbb{R} \to \mathbb{R}$ and a *bias* $b_v \in \mathbb{R}$. In the future the notation $\mathcal{E}_v$ will be used for the edges ending in the node $v \in \mathcal{V}$, i.e.,

$$\mathcal{E}_v = \{e \in \mathcal{E} |\; e = (v', v), v' \in \mathcal{V}\}.$$

Definition 3.2 (An Artificial Neuron)**.** Except for the input layer $\mathcal{I}$ each node $v \in \mathcal{V} \setminus \mathcal{I}$ in the neural network defined by the architecture $\mathcal{G} = \mathcal{V}, \mathcal{E})$ is associated with a simple computation unit, a neuron of an ANN, computing

$$x_v = \sigma_v\Big(\sum_{e=(v',v) \in \mathcal{E}_v} w_e x_{v'} + b_v \Big).$$

In a neural network the neurons are thus computing the intermediate results, which can be combined into a more complicated function. Typically the neurons are organised in layers.

Definition 3.3 (Layer, Depth and Width of a Layer of an ANN)**.** The neurons (and the associated nodes) of a neural network defined by the architecture $\mathcal{G} = \mathcal{V}, \mathcal{E})$ are said to be in layer l if the path of the edges from the input layer to the said neuron is of length l. The layer $l \in \{0, 1, \dots, L\}, L \in \mathbb{Z}_+$ of a neural network consists of all the neurons (or nodes) in that layer.

The *depth* of a neural network is the longest path of edges from the input to the output layer, or in a layered network the number of layers $L \in \mathbb{Z}_+$. Note that the input layer I is not counted into the number of layers.

The *width* of a layer l of a neural network is the amount of neurons in that layer, $N_l \in \mathbb{Z}_+$.

Because all the neurons in the same layer typically use the same activation function the activation function used in layer l will be denoted with σ_l when needed. Furthermore the same activation function may be used in all of the layers of a neural network (except for the output layer, which usually has the identity as the activation), in which case the subscript will be omitted completely and simply the notation σ will be used for the activation. In addition the weights and biases in the layer l can be stacked into matrices, in which case it is possible to write the computation of the layer l as

$$\mathbf{x}^l = \sigma_l(\mathbf{W}^{l\intercal}\mathbf{x}^{l-1} + \mathbf{b}^l),$$

where $\mathbf{W}^l \in \mathbb{R}^{N_l \times \mathbb{N}_{l-1}}$, $\mathbf{b}^l \in \mathbb{R}^{N_l}$, and $\mathbf{x}^{l-1} \in \mathbb{R}^{N_{l-1}}$ is the output of the previous layer (or the input of this layer) and $\mathbf{x}^l \in \mathbb{R}^{N_l}$ is the output of this layer. The mapping $\mathbf{x}^{l+1} \mapsto \mathbf{W}^{l\intercal}\mathbf{x}^{l-1} + \mathbf{b}^l$ is the affine-linear transformation of the neural network in layer l, and when needed one may mark it with T_l for example. The architecture of a neural network can also be defined in a layer-wise manner as a organised collection of the activation functions and affine-linear transformations as is done in [13]. See Figure 2 for a description of a simple two layered ANN.

Commonly at least the weights w_e and biases b_v, for all $e \in \mathcal{E}$ and $v \in \mathcal{V}$, of a neural network defined by the architecture $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ constitute the set of *trainable parameters* $\theta \in \Theta \subset \mathbb{R}^M$, where $M \in \mathbb{Z}_+$. In addition, for example the activation functions σ_v can have some trainable parameters as is described in [6]. The architecture of an ANN on the other hand is determined through the hyperparameters. The hyperparameters typically include the depth L of a network, the used activation functions σ_v, and the widths of different layers. In addition to these, for example the optimisation algorithms used for the training have their own set of hyperparameters.

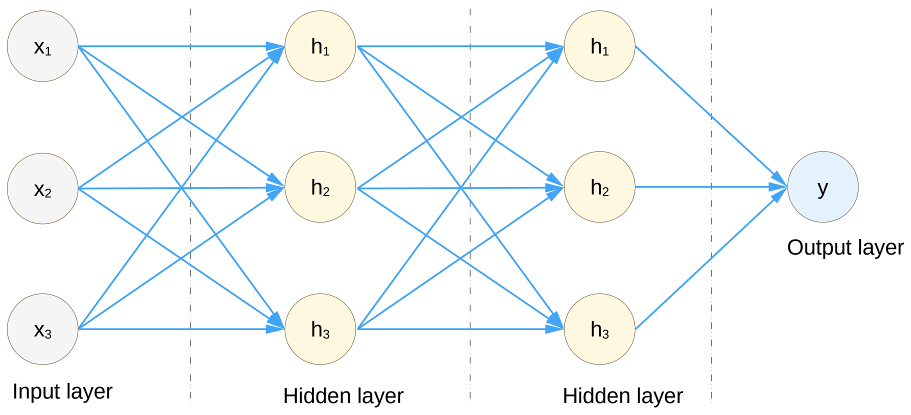

Figure 2: A simple graph of a two layered ANN.

An ANN defined by the architecture $\mathcal{G}$ determines a function class $\mathcal{F}_\mathcal{G}$, consisting of all the functions possible to produce with the network by changing the values of the trainable parameters θ [43]. For example a feed-forward network of depth L, whose layers only consist of activations and affine-linear transformations determines a function class

$$\begin{aligned}\mathcal{F}_\mathcal{G} &= \{f \mid f : \mathbb{R}^n \to \mathbb{R}^m, f(\mathbf{x}) = (\sigma_L \circ T_L \circ \ldots \circ \sigma_2 \circ T_2 \circ \sigma_1 \circ T_1)(\mathbf{x})\} \\ &= \{f \mid f : \mathbb{R}^n \to \mathbb{R}^m, f(\mathbf{x}) = \sigma_L(T_L(\ldots(\sigma_2(T_2(\sigma_1(T_1(\mathbf{x})))))\ldots))\},\end{aligned}$$

using the previous layer-wise notation.

3.2.1 Activation

Activation functions are meant to activate the input coming to it. In the hidden layers of neural networks activation functions are typically nonlinear, which enables the ANNs to learn highly complex functions, but the activation function σ_L in the output layer can also be a linear transform, usually the identity $x \mapsto x$ (see e.g. [20, 6, 5]). Including nonlinearities to the network in the form of activation functions is essential if one wishes the network to produce nonlinear functions. Normally activation functions operate element-wise over the input, so for example given the intermediate result $\mathbf{x}'^l = T_l(\mathbf{x}^l) \in \mathbb{R}^{N_{l+1}}$ in the layer $l+1$ and the activation function σ_{l+1}, we have that

$$\sigma_{l+1}(\mathbf{x}'^l) = \sigma_{l+1}((x_1'^l, x_2'^l, \ldots, x_{N_{l+1}}'^l)) = (\sigma_{l+1}(x_1'^l), \sigma_{l+1}(x_2'^l), \ldots, \sigma_{l+1}(x_{N_{l+1}}'^l)).$$

Because of this an activation function will be marked to map from $\mathbb{R}$ to $\mathbb{R}$, even though sometimes an activation function will be given input in a vector (or higher dimensional) form.

Besides introducing nonlinearity to the function class, activation functions determine whether a neuron is considered active or in-active, that is whether or not the neuron has an impact to the output. According to [45] this stems from the fact that neural networks were used to model the brains, and the first activation function was the binary stepfunction, obtaining either the value 1 (active) or 0 (in-active). So the used activation function was

$$\sigma(x) = \begin{cases} 0, & \text{when } x < 0 \\ 1, & \text{when } x \geq 0 \end{cases}$$

This is in practise a difficult activation function for any deeper networks, because it has a dichotomous response and thus a lot of information of the input is lost in activation, and one can not train a network utilising this activation function with gradient-based methods as the objective function to be optimised becomes piece-wise constant [11, 45]. An activation function that can be optimised with gradient-based methods but that reminds the binary stepfunction is called *logistic sigmoid function* [45].

Definition 3.4 (The logistic sigmoid function)**.** The logistic sigmoid is a function $\sigma : \mathbb{R} \to [0, 1]$,

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

The logistic sigmoid function can estimate the binary stepfunction arbitrarily well by weighting the input with $w \in \mathbb{R}, w > 0$ according to [45]. This can be easily seen by examing the limits of $\sigma(wx)$:

$$\lim_{w\to\infty} \sigma(wx) = \begin{cases} \lim_{w\to\infty} \frac{1}{1+e^{-wx}} = \lim_{w\to\infty} \frac{1}{1+e^{w|x|}} = 0 & \text{when } x < 0 \\ \lim_{w\to\infty} \frac{1}{1+e^{-wx}} = \lim_{w\to\infty} \frac{1}{1+e^{0}} = \frac{1}{2} & \text{when } x = 0 \\ \lim_{w\to\infty} \frac{1}{1+e^{-wx}} = \lim_{w\to\infty} \frac{1}{1+e^{-w|x|}} = 1 & \text{when } x > 0. \end{cases}$$

The logistic sigmoid is differentiable everywhere and its derivative is

$$\sigma'(x) = \frac{\mathrm{d}}{\mathrm{d}x} \frac{1}{1 + e^{-x}} = \frac{\mathrm{d}}{\mathrm{d}x} (1 + e^{-x})^{-1} = \frac{e^{-x}}{(1 + e^{-x})^2},$$

so a neural network using this activation can be optimised with gradient-based methods. However, using the logistic sigmoid as the activation function can lead to a problem known as the *vanishing gradient* as described in [6]. In the vanishing gradient problem the gradient of the loss function goes to zero with respect to some parameters, which slows down or inhibits the update of those parameters completely and thus the learning process itself. This is especially a problem with deep networks as the problem cumulates through layers. The problem with logistic sigmoid is that it maps any absolutely large values close to zero or one, and its derivative similarly maps absolutely large values near (or to) zero leading to the diminishing gradient as the learning proceeds.

Another difficulty with the logistic sigmoid is the fact that it maps to range $[0,1]$, which slows learning down [33]. A usually better choice but still similar activation function is the *hyperbolic tangent* $\sigma(x) = \tanh(x)$, which maps all the values to range $[-1,1]$, and symmetrically around zero meaning that any activation occuring near zero behaves similarly to the identity and eases learning. However, like the derivative of logistic sigmoid, the derivative of hyperbolic tangent maps values far from zero near zero, which can still lead to the problem of vanishing gradient although the problem is slightly milder for hyperbolic tangent as is shown in [33]. Both the hyperbolic tangent and logistic sigmoid have use in the output layer assuming that it is fitting for the task at hand to do so. In addition these activation functions have use in some neural network architectures, such as in recurrent neural networks (RNN) that have connections also to previous neurons, and with an appropriate initialisation of the trainable parameters they can achieve comparable results [33, 11].

Today a common choice of activation function is the *rectified linear unit*, or *ReLU* for short.

Definition 3.5 (ReLU)**.** Rectified Linear Unit is the function $\sigma : \mathbb{R} \to [0,\infty[$,

$$\sigma(x) = \max(0,x) = \begin{cases} 0, & \text{when } x \leq 0 \\ x, & \text{when } x > 0. \end{cases}$$

The benefit of ReLU over the sigmoid and tanh activation functions is computational simplicity and its capability to set a part of the input to absolute zero. ReLU functions as identity to all positive input meaning that it preserves the output of active neurons as is [11, 33]. This property can, however, result in *exploding gradient* problem in which the estimated gradient becomes too large making the learning unstable or in halting it all together, if the gradient estimate goes beyond what the computer can represent with a set number of bits. This can happen because ReLU, unlike tanh and sigmoid, is unrestricted and allows for arbitrarily large numbers. Exploding gradient is, however, more of a problem with recurrent neural networks [11], and ReLU, functioning as the identity, does not at least accumulate the problem.

The derivative of ReLU is

$$\sigma(x) = \begin{cases} 0, & \text{when } x < 0 \\ 1, & \text{when } x > 0, \end{cases}$$

but when $x = 0$, it is not defined. This is however not a problem in practise. The subdifferential of ReLU at the origin is $\partial\sigma(0) = [0, 1]$, and in practise typically the left derivative of ReLU is used at the origin if needed (see [11] and [2]).

Another problem that may arise with ReLU is the *dying ReLU* problem [11]. Because ReLU sets all negative inputs to absolute zero, it may inactivate too many neurons already at the early stages of learning. In principle inactivating some of the neurons can be seen as a good thing, as it reduces the computational load and removes the unecessary neurons from use, but if too many neurons are set to inactive too early, the neural network may be unable to learn the appropriate function. The dying ReLU may be avoided by initialising the values of trainable parameters appropriately at the start.

There are also variants of ReLU that allow some values for negative inputs as well, as to not kill the learning completely at them. These activation functions are for example the Leaky ReLU, which allows a negative input x to obtain value αx, and the everywhere differentiable Exponential Linear Unit, or ELU, giving value $\alpha(e^x + 1)$ to negative input x. Here $\alpha \in \mathbb{R}$ is some positive constant. These are described for example in [33] and [6]. Both Leaky ReLU and ELU act as the identity for any positive input. There are also variants of activation functions, whose parameters can be altered during training as described in e.g. [6]. ReLU, however, is considered as a good starting point for an activation function at least for FNNs.

3.2.2 Depth and width

Deep or wide enough networks can be *universal approximators*, i.e., a neural network with an appropriate architecture $\mathcal{G}$ can approximate any continuous and bounded function to an arbitrary accuracy. In the following this property is proven for a *shallow* neural network, i.e., a network having only one hidden layer. For simplicity it will also be assumed that the neural network has a scalar output. The proofs and concepts in this section are mainly based on [43]. Let us first define a universal approximator.

Definition 3.6 (Universal approximator (real case))**.** Let $\mathcal{F}$ be a function class. $\mathcal{F}$ is a universal approximator over some compact, i.e., closed and restricted, area S if for every continuous function $g : S \to \mathbb{R}$ and approximation parameter $\varepsilon \in \mathbb{R}, \varepsilon > 0$ there exists $f \in \mathcal{F}$ so that

$$\sup_{\mathbf{x} \in S} |f(\mathbf{x}) - g(\mathbf{x})| \leq \varepsilon.$$

While the above definition is given only for real-valued functions, as that is more fitting for the purposes of this thesis, universal approximator can also be defined for function classes with vector-valued functions by changing the absolute value to an appropriate norm.

The Stone-Weierstrass theorem shows that function classes fulfilling certain properties are universal approximators, and it can be highly useful when proving the universal approximator property. The current standard for the proof of the following theorem is by Bernstein, but let us take it here as given.

Theorem 3.7 (Stone-Weierstrass). *Let $\mathcal{F}$ be a function class fulfilling the following properties.*

1. *Each $f \in \mathcal{F}$ is continuous.*
2. *For each $\mathbf{x} \in \mathbb{R}^n$ there is $f \in \mathcal{F}$ so that $f(\mathbf{x}) \neq 0$.*
3. *For each $\mathbf{x}' \neq \mathbf{x}$ there is $f \in \mathcal{F}$ so that $f(\mathbf{x}) \neq f(\mathbf{x}')$.*
4. *$\mathcal{F}$ is closed under multiplication and addition.*

Then $\mathcal{F}$ is a universal approximator.

For a shallow network producing a scalar output and without any additional operations besides the activations and affine-linear transformations, the function class induced by that neural network is

$$\mathcal{F}_{\mathcal{G}} = \{f : \mathbb{R}^n \to \mathbb{R} \,|\, f(\mathbf{x}) = \mathbf{a}^\intercal \sigma(\mathbf{W}^\intercal \mathbf{x} + \mathbf{b})\}$$

where σ is the activation function used by all the neurons in the hidden layer, $\mathbf{a} \in \mathbb{R}^s$ is the weight matrix of the output layer, $\mathbf{W} \in \mathbb{R}^{n \times s}$ is the weight matrix of the hidden layer, $\mathbf{b} \in \mathbb{R}^s$ is the bias vector of the hidden layer, and $s \in \mathbb{Z}_+$ is the number of neurons in the hidden layer. In the following the universal approximator property for a shallow network having an unusual activation function, namely the cosine, will be proven. This will then be used to prove the universal approximator property for a shallow, ReLU activated neural network.

Lemma 3.8. *Let the function class induced by the neural network with architecture $\mathcal{G}$ be*

$$\mathcal{F}_{\mathcal{G}} = \{f : \mathbb{R}^n \to \mathbb{R} \,|\, f(\mathbf{x}) = \mathbf{a}^\intercal \cos(\mathbf{W}^\intercal \mathbf{x} + \mathbf{b}), \mathbf{a} \in \mathbb{R}^s, \mathbf{W} \in \mathbb{R}^{n \times s}, \mathbf{b} \in \mathbb{R}^s\}.$$

Then $\mathcal{F}_{\mathcal{G}}$ is a universal approximator.

Proof. Let us prove this in four sections using the Theorem 3.7:

1. Clearly each $f \in \mathcal{F}_{\mathcal{G}}$ is continuous since for each component f_i in f it holds that

$$f_i(\mathbf{x}) = a_i \cos(\mathbf{w}_i^\intercal \mathbf{x} + b_i) = a_i \cos(\sum_{k=1}^{n} w_{ki} x_k + b_i),$$

   where $i = 1, 2, \ldots, s$, $\mathbf{w}_i$ is the ith column in matrix $\mathbf{W}$ and w_{ki} is the kth element in the ith column, consists of a combination of continuous function and is thus continuous. As each component in f is continuous, f is continuous.

2. Let us notice that $\mathbf{a}^\intercal \cos(\mathbf{0}^\intercal \mathbf{x} + \mathbf{0})$, where $\mathbf{a}^\intercal = (1, 0, \ldots, 0) \in \mathbb{R}^s$, is in $\mathcal{F}_{\mathcal{G}}$. Since for each $\mathbf{x} \in \mathbb{R}^n$ it holds that $\mathbf{a}^\intercal \cos(\mathbf{0}^\intercal \mathbf{x} + \mathbf{0}) = \cos(0) = 1 \neq 0$, then for each $\mathbf{x} \in \mathbb{R}^n$ there exists $f \in \mathcal{F}_{\mathcal{G}}$ so that $f(\mathbf{x}) \neq 0$.

3. Let $\mathbf{x}' \neq \mathbf{x}$. Then $f(\mathbf{z}) = \mathbf{a}^\intercal \cos(\frac{(\mathbf{x}-\mathbf{x}')}{||\mathbf{x}-\mathbf{x}'||_2}^\intercal)(\mathbf{z} - \mathbf{x}'))$, where $\mathbf{a} \in \mathbb{R}^s$ is defined as before, is in $\mathcal{F}_{\mathcal{G}}$. Thus

$$\begin{aligned} f(\mathbf{x}) &= \mathbf{a}^\intercal \cos(\frac{(\mathbf{x}-\mathbf{x}')}{||\mathbf{x}-\mathbf{x}'||_2}^\intercal)(\mathbf{x} - \mathbf{x}')) = \cos(1) \\ &\neq \cos(0) = \mathbf{a}^\intercal \cos(\frac{(\mathbf{x}-\mathbf{x}')}{||\mathbf{x}-\mathbf{x}'||_2}^\intercal)(\mathbf{x}' - \mathbf{x}')) = f(\mathbf{x}'), \end{aligned}$$

   i.e., for each $\mathbf{x}' \neq \mathbf{x}$ there is $f \in \mathcal{F}_{\mathcal{G}}$ so that $f(\mathbf{x}) \neq f(\mathbf{x}')$.

4. Let us finally show that $\mathcal{F}_{\mathcal{G}}$ is closed under multiplication and addition.

Let $f, g \in \mathcal{F}_{\mathcal{G}}$. Now

$$\begin{aligned}
(fg)(\mathbf{x}) =& f(\mathbf{x})g(\mathbf{x}) = \mathbf{a}^\intercal \cos(\mathbf{W}^\intercal\mathbf{x} + \mathbf{b})\mathbf{c}^\intercal \cos(\mathbf{V}^\intercal\mathbf{x} + \mathbf{d}) \\
=& \left(\sum_{i=1}^{s} a_i \cos(\mathbf{w}_i^\intercal\mathbf{x} + b_i)\right)\left(\sum_{k=1}^{t} \mathbf{c}_k \cos(\mathbf{v}_k^\intercal\mathbf{x} + d_k)\right) \\
=& \sum_{i=1}^{s}\sum_{k=1}^{t} a_i c_k \cos(w_i^\intercal\mathbf{x} + b_i)\cos(v_k^\intercal\mathbf{x} + d_i) \\
=& \sum_{i=1}^{s}\sum_{k=1}^{t} a_i c_k \frac{1}{2}(\cos(\mathbf{w}_i^\intercal\mathbf{x} + b_i + \mathbf{v}_k^\intercal\mathbf{x} + d_k) \\
&+ \cos(\mathbf{w}_i^\intercal\mathbf{x} + b_i - (\mathbf{v}_k^\intercal\mathbf{x} + d_k))) \\
=& \sum_{i=1}^{s}\sum_{k=1}^{t} \left(a_i c_k \frac{1}{2}\right)(\cos((\mathbf{w}_i^\intercal + \mathbf{v}_k^\intercal)\mathbf{x} + (b_i + d_k)) \\
&+ \cos((\mathbf{w}_i^\intercal - \mathbf{v}_k^\intercal)\mathbf{x} + (b_i - d_k))) \in \mathcal{F}_{\mathcal{G}}
\end{aligned}$$

and

$$\begin{aligned}
(f+g)(\mathbf{x}) =& f(\mathbf{x}) + g(\mathbf{x}) = \mathbf{a}^\intercal \cos(\mathbf{W}^\intercal\mathbf{x} + \mathbf{b}) + \mathbf{c}^\intercal \cos(\mathbf{V}^\intercal\mathbf{x} + \mathbf{d}) \\
=& \sum_{i=1}^{s} a_i \cos(\mathbf{w}_i^\intercal\mathbf{x} + b_i) + \sum_{k=1}^{t} c_k \cos(\mathbf{v}_k^\intercal\mathbf{x} + d_k) \\
=& \sum_{i=1}^{s+t} o_i \cos(\mathbf{u}_i^\intercal\mathbf{x} + p_i) \in \mathcal{F}_{\mathcal{G}},
\end{aligned}$$

where $o_i = a_i$, $\mathbf{u}_i = \mathbf{w}_i$ and $p_i = b_i$, when $i = 1, 2, \ldots, s$, and $o_i = c_i$, $\mathbf{u}_i = \mathbf{v}_i$ and $p_i = d_i$, when $i = s+1, s+2, \ldots, t$.

Then according to Theorem 3.7 $\mathcal{F}_\Phi$ is a universal approximator.

□

Theorem 3.9 (Universal approximator theorem (shallow, ReLU activated neural network))**.** *Let the activation function $\sigma : \mathbb{R} \to \mathbb{R}$ of a shallow neural network be ReLU, i.e., $\sigma(x) = max(0, x)$. Then the function class induced by the network with architecture Φ*

$$\mathcal{F}_\Phi = \{f : \mathbb{R}^n \to \mathbb{R} \,|\, f(\mathbf{x}) = \mathbf{a}^\intercal\sigma(\mathbf{W}^\intercal\mathbf{x} + \mathbf{b}), \mathbf{a} \in \mathbb{R}^s, \mathbf{W} \in \mathbb{R}^{n\times s}, \mathbf{b} \in \mathbb{R}^s\}$$

is a universal approximator.

Proof. Let $S \subset \mathbb{R}^n$ be a compact area, $g : S \to \mathbb{R}$ be a continuous function and $\varepsilon \in \mathbb{R}, \varepsilon > 0$. Then according to Lemma 3.8 the function class induced by a cosine activated shallow neural network with unlimited width $\mathcal{F}_\Psi$ is a universal approximator, i.e., there is $h \in \mathcal{F}_\Psi$, $h(\mathbf{x}) = \mathbf{c}^\intercal \cos(\mathbf{V}^\intercal \mathbf{x} + \mathbf{d})$, so that

$$\sup_{\mathbf{x} \in S} |h(\mathbf{x}) - g(\mathbf{x})| \leq \frac{\varepsilon}{2}.$$

Now, if we can approximate function h with function $f \in \mathcal{F}_\Phi$, we can prove that $\mathcal{F}_\Phi$ is a universal approximator.

Let us first show, that we can approximate parts of function h with function f, when $n = 1$. Let us first notice that cosine is Lipschitz continuous, i.e., there exists $L > 0$ so that

$$|\cos(x) - \cos(y)| \leq L|x - y|.$$

Set S is now the closed interval $[s, t]$ (as $n = 1$). Let us choose points k_i, $i = 0, 1, \ldots, K$ from the interval $[s, t]$ so that $k_0 = s$, $k_K = t$ and $k_i - k_{i-1} = l$, where $l = \min(\frac{\varepsilon}{4|c|L}, \frac{\varepsilon}{4|c|})$ and $c \in \mathbb{R}$, for all $i = 1, \ldots, K$, and form function f in the following way:

$$\begin{aligned} f(x) &= \sum_{i=0}^{K-1} a_i \sigma(x - k_i) + \sum_{i=0}^{K-1} (-1) a_i \sigma(x - k_i - l) \\ &= \sum_{i=0}^{K-1} a_i \left(\sigma(x - k_i) - \sigma(x - k_i - l) \right), \end{aligned}$$

where

$$a = (a_0, a_1, \ldots, a_{K-1}) = c(\cos(k_1), \cos(k_2) - \cos(k_1), \ldots, \cos(k_K) - \cos(k_{K-1}).$$

Now

$$\begin{aligned} (\sigma(x - k_i) - \sigma(x - k_i - l)) &= \begin{cases} 0, \text{ when } x - k_i - l < x - k_i < 0 \\ x - k_i, \text{ when } 0 \leq x - k_i < l \\ 1, \text{ when } x - k_i \geq l. \end{cases} \\ &= \begin{cases} 0, \text{ when } x < k_i \\ x - k_i, \text{ when } k_i \leq x < l + k_i = k_{i+1} \\ 1, \text{ when } x \geq k_i + l = k_{i+1}. \end{cases} \end{aligned}$$

Let $x \in S$, and k_o be the largest point $k_i, i = 0, 1, \dots, K$, so that $k_o \leq x$. We can aproximate

$$\begin{aligned}
|c\cos(x) - f(x)| &= |c\cos(x) - c\cos(k_o) + c\cos(k_o) - f(k_o) + f(k_o) - f(x)| \\
&= |c||\cos(x) - \cos(k_o)| + |c\cos(k_o) - f(k_o)| + |f(k_o) - f(x)| \\
&\leq |c|L|x - k_o| \\
&\quad + |c\cos(k_o) - \sum_{i=0}^{K-1} a_i(\sigma(k_o - k_i) - \sigma(k_o - k_i - l))| \\
&\quad + |\sum_{i=0}^{K-1} a_i(\sigma(k_o - k_i) - \sigma(k_o - k_i - l)) \\
&\quad - \sum_{i=0}^{K-1} a_i(\sigma(x - k_i) - \sigma(x - k_i - l))| \\
&=|c|Ll + |c\cos(k_o) - \sum_{i=0}^{l-1} a_i| + |\sum_{i=0}^{l-1} a_i - \sum_{i=0}^{l-1} a_i - a_l(x - k_o)| \\
&\leq \frac{\varepsilon}{4} + |c||\cos(k_o) - \cos(k_1) - \cos(k_2) + \cos(k_1) - \dots \\
&\quad - \cos(k_o) + \cos(k_{o-1})| + |c||\cos(k_{o+1}) - \cos(k_o)||x - k_o| \\
&\leq \frac{\varepsilon}{4} + 0 + 2|c|l \leq \frac{\varepsilon}{4} + \frac{\varepsilon}{2} < \varepsilon.
\end{aligned}$$

Let us prove next that this can be generalised to a case where $n > 1$, i.e., that we can approximate function $h = \mathbf{c}^\intercal \cos(\mathbf{V}^\intercal \mathbf{x} + \mathbf{d})$ with an appropriate $f \in \mathcal{F}_\Phi$. Notice now that h can be written as

$$h = \sum_{i=1}^{s} c_i \cos(\sum_{j=1}^{n} v_{ij} x_j + d_i) = \sum_{i=1}^{s} c_i \cos(z_i).$$

We showed previously that we can approximate $c\cos(x)$, where $c \in \mathbb{R}$ to arbitrary accuracy with $f\mathcal{F}_\Phi$ when $n = 1$. Let us use the notation f_i for these functions and choose them so that

$$\sup_{z_i \in [s_i, t_i]} |f_i(z_i) - c_i \cos(z_i)| < \frac{\varepsilon}{2s},$$

where $[s_i, t_i]$ is the real interval, to which x_i maps to. By choosing $f(\mathbf{x}) = \sum_{i=1}^{s} f_i(z_i)$ we obtain that

$$|h(\mathbf{x}) - f(\mathbf{x})| = |\sum_{i=1}^{s} c_i \cos(z_i) - \sum_{i=1}^{s} f_i(z_i)| = \sum_{i=1}^{s} |c_i \cos(z_i) - f_i(z_i)| < s\frac{\varepsilon}{2s} = \frac{\varepsilon}{2}.$$

Furthermore

$$\begin{aligned}|g(\mathbf{x}) - f(\mathbf{x})| = |g(\mathbf{x}) - h(\mathbf{x}) + h(\mathbf{x}) - f(\mathbf{x})| &= |g(\mathbf{x}) - h(\mathbf{x})| + |h(\mathbf{x}) - f(\mathbf{x})| \\ &< \frac{\varepsilon}{2} + \frac{\varepsilon}{2} = \varepsilon,\end{aligned}$$

and so $\sup_{\mathbf{x}\in S} |g(\mathbf{x}) - f(\mathbf{x})| < \varepsilon$. And thus $\mathcal{F}_\Phi$ is a universal approximator.

□

The above universal approximator theorem has only been proven for the ReLU activation, but the original proof given by Hornik, Stinchcombe and White (cf. [15]) was written for sigmoidal activation functions [43]. In fact, the universal approximator theorem for shallow neural networks can be generalised for neural networks with non-polynomic activation functions, see [24]. Theorem 3.9 is not particularly useful in the sense that it is an existence proof, and does not for example state how many neurons the shallow architecture might need to able to learn the desired function. It is however possible to compute a grude estimate for the needed amount and there are also different proves giving approximates for the required amount. For a shallow network, in the worst case the width of a network will depend exponentially on the number of elements in the input, an expression of the curse of dimensionality (see [11, 43]).

The depth of a neural network is commonly more important for the effectiveness of the network than the width, as typically a deeper neural network is capable of approximating the desired function with fewer neurons. There are multiple theoretical results for the approximator abilities of deeper networks, encompassing for example different architectures, and many of them show that a deeper network requires notably fewer neurons than a similar shallow one (see for example [16] and [43]). In [43] (referering to the proofs given in [41] and [42]) it is shown that a ReLU activated neural network with certain depth can not be approximated by a shallower network, if it has less than exponential amount of nodes compared to the depth.

This is proven through the use of a piecewise-affine function $\Delta : \mathbb{R} \to \mathbb{R}$,

$$\Delta(x) = 2\sigma(x) - 4\sigma\left(x - \frac{1}{2}\right) + 2\sigma(x-1) = \begin{cases} 2x, & \text{when } x \in [0, \frac{1}{2}[\\ 2x - 2, & \text{when } x \in [\frac{1}{2}, 1[\\ 0, & \text{otherwise,} \end{cases}$$

where σ is the ReLU activation. When this function is combined with itself, it produces an exponential amount of copies of itself that are equally spaced on the interval $[0, 1]$. More precisely, combining this function $l \in \mathbb{Z}_+$ times with itself ($\Delta^l = \Delta \circ \Delta \circ \ldots \circ \Delta$) produces 2^{l-1} copies. Now, the number of oscillations a ReLU activated neural network can represent is limited polynomially in the width but exponentially in depth of the network, and the number of oscillations in Δ^l grows exponentially with l. What is proven, is that a ReLU activated neural network with $3l^2 + 2$ nodes and $2l^2 + 4$ layers can represent the function Δ^{l^2+2}, where $l \geq 2$, but a ReLU activated network with fewer than $2^l + 1$ nodes and $l + 1$ layers can not. Showing that a shallow network requires exponential width to represent this highly oscillating function whereas an adequately deep network requires a constant width and thus notably fewer parameters.

Using a deep neural network architecture is also assertion of the assumption that the required function is a combination of simpler ones, and empirically it appears that deeper networks learn better than the shallower ones [11].

3.3 Optimisation

Optimisation is finding the minimum or maximum of some function j. The function j is called the objective function, although when minimising it can also be called *loss* or *cost function.* To be precise, these names are used in slightly different situations, but in the future the function to be minimised will be mainly referred to as loss function. Also, as maximation of j is equivalent to minimising $-j$, only the minimisation problem is considered. According to [11] optimisation in terms of training a neural network is somewhat different from the pure optimisation, as the aim is not typically exactly the minimisation of the chosen loss function. Instead the loss function works as a proxy to the actual goal, which can for example be a well generalising model. In addition the training is not always continued to the global or even local minimum, if doing so would result in overfitting. The following section is largely based on [11], although for section 3.3.3 also the source [21] is used.

Usually the performance is optimised relative to the training set S_{train}. In this case the difference between the output of the network $\hat{y}_i = f(x_i)$, where $f \in \mathcal{F}_G$ for the observation i, and the target y_i is optimised relative to the trainable parameters θ. As the produced output or rather the function f depends on the choice of trainable parameters θ, the notation $\hat{y}_i = f(x_i, \theta)$ will be used in the future to make the connetion to the parameters clearer. For observation i the loss is thus

$$j(\hat{\mathbf{y}}_i, \mathbf{y}_i) = j(f(\mathbf{x}_i, \theta), \mathbf{y}_i),$$

and the intention is to minimise the expectation of the losses over the whole training set

$$\hat{J}(\theta) = \mathbb{E}_{(\mathbf{x},\mathbf{y})_i \sim \hat{p}_{data}}(j(f(\mathbf{x}_i, \theta), \mathbf{y}_i)) = \frac{1}{N_{train}} \sum_{i=1}^{N_{train}} j(f(\mathbf{x}_i, \theta), \mathbf{y}_i),$$

where N_{train} is the number of observations in the training set and $\hat{p}_{data}$ is the empirical distribution defined by the training set.

However, in the ideal case the loss could be minimised relative to the true data generating distribution p_{data}, i.e.,

$$J(\theta) = \mathbb{E}_{(\mathbf{x},\mathbf{y})_i \sim p_{data}}(j(f(\mathbf{x}_i, \theta), \mathbf{y}_i))$$

would be minimised. This is known as *risk* and the measure constrained to the empirical distribution is the *empirical risk*, in which case we speak of empirical risk minimisation.

Empirical risk is sometimes prone to overfitting, where the neural network learns to model the training set too well in the sense that it fails on new observations. Sometimes a regularisation term is added to the empirical risk $\hat{J}$.Regularisation term can for example emphasize the sparsity of the trainable parameters θ or their small size. In this case the task is to minimise

$$\hat{J}(\theta) = \frac{1}{N_{train}} \sum_{i=1}^{N_{train}} j(f(\mathbf{x}_i, \theta), \mathbf{y}_i) + \lambda P(\theta),$$

where $\lambda \in \mathbb{R}$ dictates the strength of the regularisation and $P : \Theta \to \mathbb{R}$ is the regularisation function.

The optimisation task is thus to find values that minimise the empirical risk for the trainable parameters $\theta \in \Theta$. In other words, the task is to find $\hat{\theta}$ so that

$$\hat{\theta} = \underset{\theta \in \Theta}{\operatorname{argmin}} \, \hat{J}(\theta).$$

There are of course many ways to optimise a function, but especially in the training of neural networks common methods are stochastic version of the *gradient descent* and its variations. The estimation of the gradient is typically implemented with backpropagation algorithm that uses the chain-rule recursively as described for example in [11].

3.3.1 Gradient Descent

In general the gradient of a function f at point $\theta = (\theta_1, \theta_2, \ldots, \theta_M) \in \Theta$ gives the direction of the steepest ascent for the function in space Θ at the given point assuming that the function f is differentiable around the point θ.

Definition 3.10 (The gradient of a function at a given point)**.** Let f be a function $f : \mathbb{R}^M \to \mathbb{R}$ and $\mathbf{x} = (x_1, x_2, \dots, x_M) \in \mathbb{R}^M$. Then the gradient of a function f at point $\mathbf{x}$ is

$$\nabla f(\mathbf{x}) = \left(\frac{\partial}{\partial x_1} f(\mathbf{x}), \frac{\partial}{\partial x_2} f(\mathbf{x}), \dots, \frac{\partial}{\partial x_M} f(\mathbf{x}) \right),$$

assuming that the partial derivatives $\frac{\partial}{\partial x_i} f(\mathbf{x})$ exist for all $i = 1, 2, \dots, M$.

Because the gradient of a function gives the direction of steepest ascent, the negative of the gradient $-\nabla f$ gives the direction of the steepesct descent. By moving short distances or steps in the direction of the negative gradient and by updating the gradient at every new point, the function f can be updated iteratively. If we now assume the empirical risk $\hat{J}$ is differentiable, the algorithm can be defined as shown in 1. The *learning rate* in the Algorithm 1 dictates the length of the step at each round.

Algorithm 1 Gradient descent

```
Require: Learning rate $\lambda \geq 0, \lambda \in \mathbb{R}$
    Initialise the trainable parameters
    $\theta \leftarrow \theta_0$
    while stopping criterion has not been met do
        Initialise gradient at point $\theta$
        $g \leftarrow \nabla_\theta \hat{J}(\theta)$
        Update the trainable parameters
        $\theta \leftarrow \theta - \lambda g$
    end while
```

3.3.2 Introducing stochasticity

In practise gradient descent tends to be too heavy when training neural networks, as it requires the computation of the gradient over the whole training set. This is a particularly costly operation with neural networks because not only do the deep networks have multiple trainable parameters making the computation of the partial derivatives demanding, but there is also typically heeps of data. Thus the training set (and the other sets as well to speed-up the computations) are divided into subsets called *mini-batches*, and the gradient is estimated over one mini-batch at a time. To ensure that the estimated gradient would describe the training set as well as possible, the division is usually done randomly and without replacement. When done this way, the mini-batches constitute of i.i.d. observation on the first run-through of the training set and give an un-biased estimate of the gradient computed over the training set at once. Dividing the training set into mini-batches randomly also sometimes helps to maintain the generalisation ability of the network, and ease the learning in varying loss landscapes, as SGD takes more steps during optimisation and can follow the true landscape more easily with the same learning rate as GD (cf. [46]).

In *stochastic gradient descent* (SGD) the gradient is always estimated over one randomly selected mini-batch instead of the whole training set. As the estimation of the gradient over a smaller subset induces some bias to the estimate especially when running through the set again, it is important that the learning rate diminishes with increasing rounds. Otherwise the bias may cause the learning to never converge, as the estimated gradient might not be zero even in global optimum thanks to the bias. That is, assuming the learning can reach a minimum. The SGD algorithm is described in 2 for the empirical risk $\hat{J}$.

Algorithm 2 Stochastic gradient descent

```
Require: Learning rates $\lambda_t$ for each iteration round $t = 1, 2, \ldots, T$
Require: Size of the mini-batch $b$
    Initialise the trainable parameters
    $\theta \leftarrow \theta_0$
    Intialise the counter
    $t \leftarrow 0$
    while stopping criterion has not been met do
        Update the counter
        $t \leftarrow t + 1$
        Randomly select a mini-batch $B \subset S_{train}$ of $b$ observations
        Compute the gradient estimate at the point $\theta$ over the batch
        $\hat{g} \leftarrow \frac{1}{b} \nabla_\theta \sum_{i=1}^{b} j(f(\mathbf{x}_i, \theta), \mathbf{y}_i)$
        Update the trainable parameters
        $\theta \leftarrow \theta - \lambda_t \hat{g}$
    end while
```

3.3.3 Adam

GD and SGD algorithms utilise the gradient to find the optimum, but they do not for example consider the previous behaviour of the gradient and they can oscillate highly if the loss function changes rapidly in regard to some variables but not with others. These kind of areas are known as valleys and they can slow down the learning process. In addition to this the learning rate needs to be determined separately for each round in SGD, or at least its dependancy of the round number need to be pre-determined. This increases the amount of hyperparameters that need to be set separately to the training process. Tuning the hyperparameters can be a computationally expensive process as the neural network needs to be re-trained always after updating the hyperparameters. Finding an adequate learning rate can also be one of the most crucial hyperparameters, as with a too large learning rate the training might not converge at all and the results might change vastly between iterations but a learning rate too small will slow down the training or make it halt altogether.

As it is important to find a good learning rate, there are attempts to automate the search for it during training. This is what optimisation methods with adaptive learning rate intend to do, and one of these algorithms is *Adam*, which is described in [21]. The name Adam comes from adaptive moment estimation, and this optimisation method computes the estimates of the first and second moments of the gradient over each mini-batch, and computes individual learning rates for each of the parameters. The estimates of the moments are updated based on the previous iterations, and thus Adam utilises knowledge of the previous behaviour of the gradient to the current step. The algorithm for Adam is described in 3. The step length α creates an approximate upperbound to the invidual learning rates used in the update, but the actual learning rates are adapted based on the estimated moments. The bias correction in the Algorithm 3 is done to avoid biasing towards zero in the estimates, created by the zero initialisation.

3.4 Loss functions and regularisation

Loss functions measure the similarity between the output of the neural network $\hat{y}_i = f(x_i, \theta)$ and the target y_i for some observation i, where $i \in \{1, 2, \ldots, N\}$. In tasks with real valued targets, loss functions typically measure the distance between the output and the observation. Regularisation terms can be used to induce additional requirements to the trainable parameters for example, and they are used to improve the generalisation ability of the network. The loss functions introduced in this section are based on sources [4] and [44].

Definition 3.11 (Absolute error)**.** The *absolute error* between the output of a neural network $\hat{\mathbf{y}}$ and measured target $\mathbf{y}$ is

$$||\mathbf{y} - \hat{\mathbf{y}}||_1 = \sum_{i=1}^{m} |\mathbf{y}_i - \hat{\mathbf{y}}_i|.$$

Algorithm 3 Adam

```
Require: Step length $\alpha \in \mathbb{R}$
Require: Exponential decay rates for the moments $\beta_1, \beta_2 \in [0, 1[$
Require: Initial values for the trainable parameters $\theta_0$
Require: Small constant $\epsilon$ to ensure numerical stability
    Initialise the first and second moment
    $\mathbf{m}_0 \leftarrow \mathbf{0}$
    $\mathbf{v}_0 \leftarrow \mathbf{0}$
    Initialise the counter
    $t \leftarrow 0$
    while stopping criterion has not been met do
        Update the counter
        $t \leftarrow t + 1$
        Estimate the gradient at round $t$ over the batch $B$
        $g_t \leftarrow \frac{1}{b}\nabla_\theta \sum_{i=1}^{b} j(f(x_i, \theta_{t-1}), y_i)$
        Update the biased estimates of the first and second moments
        $\mathbf{m}_t \leftarrow \beta_1 \cdot \mathbf{m}_{t-1} + (1 - \beta_1) \cdot g_t$
        $\mathbf{v}_t \leftarrow \beta_2 \cdot \mathbf{v}_{t-1} + (1 - \beta_2) \cdot g_t^2$
        Compute the unbiased estimates
        $\hat{\mathbf{m}}_t \leftarrow \mathbf{m}_t / (1 - \beta_1^t)$
        $\hat{\mathbf{v}}_t \leftarrow \mathbf{v}_t / (1 - \beta_2^t)$
        Update the trainable parameters
        $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{\mathbf{m}}_t / (\sqrt{\hat{\mathbf{v}}_t} + \epsilon)$
    end while
```

In other words absolute error is the L_1 norm of the *residual* $\mathbf{r} = \mathbf{y} - \hat{\mathbf{y}}$, $||\mathbf{r}||_1$.

Absolute error measures thus scale of the difference between the output and target, and the losses for each of the observations impact the total error in direct relation to their scale. This makes the absolute error robust to outliers (when compared to for example squared error introduced below), but the gradient is not dependent on the scale of the error, which may make the training harder for small errors. The absolute error also suffers from the same problem as ReLU, i.e., it is not differentiable at origin. A simple variant of absolute error is to take its average, $\frac{1}{m}||\mathbf{y} - \hat{\mathbf{y}}||_1$.

Definition 3.12 (Squared error)**.** The *squared error* between the ouput of a neural network $\hat{\mathbf{y}}$ and the target $\mathbf{y}$ is

$$||\mathbf{y} - \hat{\mathbf{y}}||_2^2 = \sum_{i=1}^{m} (\mathbf{y}_i - \hat{\mathbf{y}}_i)^2,$$

which is the L_2 norm or Euclidean norm of the residual, $||\mathbf{r}||_2^2$.

Squared error reacts more to large errors than the absolute error making it emphasize outliers more. Squared error is differentiable everywhere and its gradient changes according to the changes in the component of the residual, which improves learning for gradient based optimisation methods. A simple transform of the squared error is the mean squared error (MSE), which takes the average of the squared error, and the squareroot of the MSE (RMSE), which gives the loss on the same scale as the target.

Losses combining the robustness of the absolute error and the differentiability of the squared error are the Hubert loss and the log-cosh loss. Hubert loss achieves this by straigthfowardly using the absolute error for differences above a certain threshold and squared error otherwise, while log-cosh takes a different approach and computes the logarithm of the hyperbolic cosine transformation of the residual. The threshold parameter for the Hubert loss needs to be chosen, while log-cosh induces no extra hyperparameters, but is computationally more expensive. Root mean squared logarithmic error takes the root of the average difference between logarithmic transformations of the output and target. This error is again fairly robust to outliers and emphasizes under-estimation more than over-estimation of the target, but it can not be computed to negative values.

Regularisation is used to hinder the model from overfitting [11]. A common way is to for example prefer small or zero weights in the model when the corresponding input does not have a significant effect. Adding L_2 norm of the weights to the risk makes the model to search for smaller weights, and the L_1 norm of the weights drives the weights of the least significant inputs to zero ensuring *sparsity* of the matrix representation. Regularisation can also be achieved through architectural choices. For example using *convolutional layers* instead of fully-connected ones ensures sparsity of the weight matrix.

4 Convolutional Neural Networks

Convolutional neural networks, or just convolutional networks, CNNs for short, are a special kind of artificial neural networks utilising the convolution operation as their affine-linear transform, at least in one of their layers according to [11]. Part of the design of convolutional networks was inspired by the workings of the mammalian visual system, and they are commonly used for image processing tasks, but they also have use for other types of data, such as signals and time-series. Depending on the choice of convolution and whether or not an operation known as pooling is applied in the network, convolutional neural networks can be shown to be universal approximators [1], but even without that property CNNs are good at detecting and substracting features in the data, and they are less expensive to teach then the fully-connected ANNs.

4.1 Convolution

Convolution is an operation over two functions, and it can be defined as below (see [39]).

Definition 4.1 (Convolution)**.** Given two functions $f : \mathbb{R}^n \to \mathbb{R}$ and $k : \mathbb{R}^n \to \mathbb{R}$, for which their absolute value is Lebesque integrable over domain $\mathbb{R}^n$, i.e.,

$$\int\limits_{\mathbb{R}^n} |f(\mathbf{x})| \, \mathrm{d}\mathbf{x} < \infty,$$

convolution is defined as

$$(f \star k)(\mathbf{x}) = \int\limits_{\mathbb{R}^n} k(\mathbf{x} - \mathbf{y}) f(\mathbf{y}) \, \mathrm{d}\mathbf{y}.$$

The function k is commonly referred to as the *kernel.*

Convolution is commutative, so one can write it equally as

$$(f \star k)(\mathbf{y}) = \int\limits_{\mathbb{R}^n} k(\mathbf{x}) f(\mathbf{y} - \mathbf{x}) \, \mathrm{d}\mathbf{x}.$$

Actually convolution can be extended to for example complex numbers and it is also possible to have a two-dimensional convolution, in which case the convolution is taken over both dimensions. However, for the purposes of this thesis, we are mainly interested in $n = 1$, i.e.,

$$(f \star k)(\mathbf{y}) = \int\limits_{-\infty}^{\infty} k(\mathbf{x}) f(\mathbf{y} - \mathbf{x}) \, \mathrm{d}\mathbf{x}$$

and its discrete version.

In a discrete setting, as is the case when implementing the machine learning algorithms, the functions f and k can be viewed to obtain values only at integers, and the convolution can be written as below (see e.g. [11]).

Definition 4.2 (Discrete convolution)**.** Given two functions $f : \mathbb{Z} \to \mathbb{R}$ and $k : \mathbb{Z} \to \mathbb{R}$, *discrete convolution* is defined as

$$(f \star k)(y) = \sum_{i=-\infty}^{\infty} k(y - i) f(i),$$

where $y \in \mathbb{Z}$.

Of course in the case of machines the sum is not taken over infinitely many points as the values for both functions need to be stored. Instead the functions or, as they can also be viewed, vectors are thought to obtain zero values everywhere but in the finite set of values we are interested in and so the sum can be reduced to a sum over that set, say S [11].

The discrete convolution operation takes thus a weighted sum over all the values of the function (or a vector) f, and the weights are determined by the kernel k. Let us now consider the notation for a neuron from section 3.2 for a node v, which was

$$x_v = \sigma_v \left(\sum_{e=(v',v) \in \mathcal{E}_v} w_e x_{v'} + b_v \right).$$

For simplicity of notation let us as assume that the neurons in the network construct layers as is commonly the case, and index all the nodes in the lth layer with x_i^l, where $i = 1, 2, \ldots, N_l$. Now when we are dealing with neurons that implement convolution the weights are actually the values taken by the kernel k (or k_l for the kernel in layer l), and the outputs of the previous layer $l - 1$ constitute the function or vector f (denoted here f_l for outputs of layer l). So the output of the ith neuron in layer l is

$$x_i^l = f_l(i) = \sigma_i^l \left(\sum_{j=1}^{N_{l-1}} k_l(i - j) f_{l-1}(j) + b_i^l \right) = \sigma_i^l \left((f_{l-1} \star k_l)(i) + b_i^l \right).$$

And for the whole layer the computation could be written as

$$\mathbf{x}^l = \sigma^l(f_{l-1} \star k_l + \mathbf{b}_l) = \sigma^l(\mathbf{x}_{l-1} \star k_l + \mathbf{b}^l).$$

The difference between a regular fully-connected neuron and one that utilises the convolution is that the kernel is typically chosen to obtain non-zero values on a substantially smaller range of indices than the function f_{l-1}. For example, the kernel can be chosen to have a *size* of three, in which case the kernel can obtain non-zero values only at indices -1, 0 and 1. This results in *sparse interactions* (or *sparse connections* or *weights*), i.e., a node in $(l-1)$th layer is connected only to a few nodes in the next layer. The benefits of sparse weights is the reduced memory and computational losses, since there are less connections for which the weights need to be stored and optimised [11].

Another benefit that naturally occurs with convolution is the *parameter sharing.* The kernel k stays the same for all the neurons in the layer, meaning that the trainable weights determined by the kernel are the same for that layer. This way one only needs to store the weights of one kernel for that layer further reducing the memory cost. Sharing the parameters across a layer also gives convolution its property of *equivariance to translation*, meaning that if the input to the layer is shifted in some direction, than the output is shifted equally [11]. According to [20] this is particularly beneficial for detecting features in the input data, as it no longer matters where in the input some feature appears, it will still be detected.

It should be noted, that while it was previously stated that only one kernel needs to be saved for each layer, this is not what is done in practise. In a convolutional neural network one kernel learns one feature from the data, such as horisontal lines. To learn more features from the data, more kernels per layer are needed. This is why a convolutional layer typically computes multiple convolutions parallel for the input and its output constitutes of multiple *feature maps*, each separating different features from the data [11]. The later layers take in these feature maps and combine them together to form more complicated features. While the need to save more kernels per layer is more costly than saving just one, the overall cost can still be much smaller than the cost for having a fully-connected layer assuming the kernel is chosen to be small. So for a convolutional layer l obtaining input $\mathbf{x}^{l-1}$ with $t \in \mathbb{Z}_+$ feature maps, the computation could be written as

$$\mathbf{x}^{l,j} = \sigma^l(\sum_{i=1}^{t}(\mathbf{x}^{l-1,i} \star k_l^j) + \mathbf{b}^{l,j}),$$

where $j = 1, 2, \ldots, u \in \mathbb{Z}_+$ encompasses all the feature maps extracted at layer l.

4.2 ResNet

When training a deep neural network one can encounter difficulties, that do not occur with the shallower versions. One of these problems is the vanishing or exploding gradients already introduced in the context of activation functions in section 3.2.1. Another problem discovered empirically for deep networks was that while the training could converge, the deeper networks had higher errors than their shallower (but suitably deep) counterparts. This problem is known as the *degradation* problem and it is not due to overfitting, as even the training error is higher for the deeper network [1]. In theory a deeper network should perform at least as well as its shallower counterpart if the deeper layers can learn the identity exactly or approximate it arbitrarily well. Thus the degradation problem is thought to be a problem with learning the identity mapping when needed in practice. A *Residual Neural Network* or simply *ResNet* is a neural network design that addresses this problem through *residual learning*. The following description of ResNet and skip-connections is based on [14].

In residual learning, instead of trying to find the optimal mapping $h : \Omega \to \mathbb{R}^c$, where $\Omega \subset \mathbb{R}^d$ and $c, d \in \mathbb{Z}_+$, one tries to learn the residual version $f : \Omega \to \mathbb{R}^c$, $f(\mathbf{x}) = h(\mathbf{x}) - \mathbf{x}$. (More precisely one tries to find the trainable parameters $\hat{\theta} \in \Theta$ so that $f(\mathbf{x}, \hat{\theta}) = h(\mathbf{x}) - \mathbf{x}$, where $f : \Omega \times \Theta \to \mathbb{R}^c$.) Then the optimal function can be written as $h(\mathbf{x}) = f(\mathbf{x}) + \mathbf{x}$ and in the extreme case, where the optimal mapping is the identity, the learner (the neural network) can simply set the weights coverning the shape of f to zero to obtain $h(\mathbf{x}) = \mathbf{x}$. A ResNet applies residual learning by using shortcut connections in its architecture called *skip* or *residual connections*. See Figure 3 for a depiction of simple convolutional ResNet having kernel of size 3.

4.2.1 Skip connection

A skip connection is a shortcut connection that connects a neuron in a layer $l - a$ to another neuron in layer l, where $a \in \mathbb{Z}, i > 1$, and so the connection skips at least one layer in between. In a ResNet the connection sums the output of layer $l - a$ and the output of layer l (before activation) together element-wise, and so either the outputs of both layers need to be of the same size, or they are changed to match each other through some linear projection applied to the output of layer $l - a$. For the ith neuron in the lth layer which is connected to ith neuron in layer $l - a$, assuming convolutional layers and using the same notation as before, the computation can be written as

$$x_i^l = \sigma_i^l \left((f_{l-1} \star k_l)(i) + b_i^l + x_j^{l-a} \right).$$

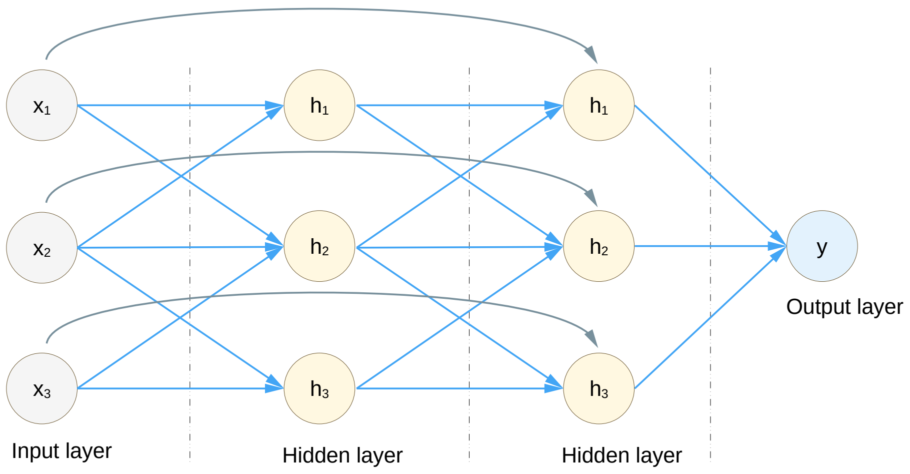

Figure 3: A simple convolutional ResNet. Notice the sparse connections.

To simplify the notation further for the layer-wise case, the convolution and the addition of bias vector for layer l will be marked with T_l, i.e.,

$$\begin{aligned} T_l\left(\mathbf{x}^{l-1}\right) &= T_l\left(\left(x_1^{l-1}, x_2^{l-1}, \ldots, x_{N_{l-1}}^{l-1}\right)\right) \\ &= \left((f_{l-1} \star k_l)(1), (f_{l-1} \star k_l)(2), \ldots, (f_{l-1} \star k_l)(N_{l-1})\right) + \mathbf{b}_l. \end{aligned}$$

Then for the layer l the skip connection from layer $l-a$ can be written as

$$\mathbf{x}^l = \sigma^l\left(T_l(\mathbf{x}^{l-1}) + \mathbf{x}^{l-a}\right),$$

where σ_l is the activation used in all the nodes of the layer (as is typically the case) and $\mathbf{x}^{l-a}, \mathbf{x}^{l-1}$ and $\mathbf{b}_l$ are as before.

If we next assume that $a = 2$, then the corresponding computation in the layer l will be

$$\begin{aligned} \mathbf{x}^l &= \left(\sigma^l \circ \left(T_l \circ \sigma^{l-1} \circ T_{l-1} + I\right)\right)\left(\mathbf{x}^{l-2}\right) \\ &= \sigma^l\left(T_l\left(\sigma^{l-1}\left(T_{l-1}\left(\mathbf{x}^{l-2}\right)\right)\right) + I\left(\mathbf{x}^{l-2}\right)\right), \end{aligned}$$

where I is the identity.

Having the residual connections allows the network to skip some of the layers with identity when needed. Having the shortcut connections also allows for a better gradient flow in backpropagation. Skip connections can also be implemented in a different way. For example in U-Nets (see [34]) the skip connection concatenates the feature maps from layer $l-a$ to layer l rather than using the summation as in ResNet. This allows the features extracted in the earlier layer to be directly used in the current one.

4.3 U-Net

U-Net is a so called encoder-decoder architecture as it consists of a contracting path, that encodes the input, and an expansive path decoding the encoded part to output [1]. U-Net was created to do cell segmentation tasks for medical images (see [34]), and it has use especially in inverse problems for imaging science. The contracting path of U-Net consists of blocks of convolutional layers using ReLU activations and these blocks are separated by a *pooling* operation used to make the resolution of the input smaller. When the resolution is diminished the number of feature maps is doubled. On the expanding path, the input is *upsampled* to increase the resolution back, and the number of feature maps is halved in the process. Each upsampling is followed by a similar block of convolutional layers as on the contracting path, but the input to each block is concatenated with the output from a block on the contracting path on the same scale. This results in the paths being fairly symmetric to each other and giving the network a u-shape.

The contracting path is thus decreasing the resolution of the input image one scale at a time, while extracting features from it. Extracting features on smaller resolutions allows the network to learn features on a larger scale. On the expanding path the resolution is increased again. By concatenating the higher resolution features extracted on the contracting path to the input on the expanding path, the intention is to give context information that has been lost on the contracting path back to the network. See Figure 4 for an illustration of the U-Net architecture, the figure is adapted from [34]. Notice that this is a layer-wise depiction and differs from the earlier presentations (see e.g. Figure 2 in Section 3.2).

4.3.1 Downsampling

Downsampling is used to decrease the size of the input to or output from a layer. One of the benefits of downsampling is the decreased computational complexicity as the input to the following layer becomes smaller. In convolutional neural networks the convolution operation can be used to achieve downsampling by choosing *stride* to be larger than one. Stride can be viewed as skipping part of the input nodes (see [7] for illustration in two dimensions). Another possibility is to use pooling. Pooling operation in general combines values of the input over some region and produces an summary statistic as the output [11]. Pooling creates invariance to small transformations of the input and makes the the model more robust to noise (by combining multiple points into a single statistic) and it can be used to make the network accept varying sizes of input [11, 3].

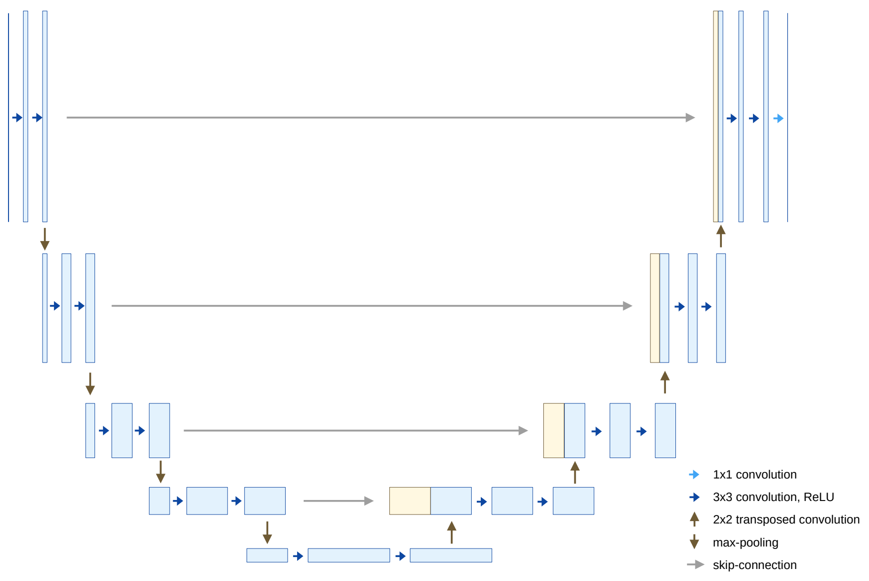

Figure 4: Illustration of the U-Net, adapted from [34].

In the following a few simple pooling methods will be introduced. As said, pooling is always performed over regions of the input to a layer, say $\mathbf{x}^l$. These regions can be chosen to overlap or be distinct from each other, and they can be chosen to be of different sizes or of the same size. The sizes can even be chosen randomly as is done for example in fractional max pooling (see [12]). So for example, choosing 2×2 pooling regions, which is a typical choice, computes a statistic over four neighbouring elements. A pooling region of size 2×1 (or 2 depending on the shape of the input) would simply compute the statistic over two neighbouring elements. These pooling regions will be marked with R_i, where $i = 1, 2, \ldots, t$ and $t \in \mathbb{Z}_+$, and their exact shapes will be omitted.

Definition 4.3 (Max pooling)**.** Given the pooling regions $R_1, R_2, \ldots, R_t$ as above of the input $\mathbf{x}_{l-1}$, the *max pooling* operation computes

$$x_i^l = \max_{r \in R_i}(r),$$

for all $i = 1, 2, \ldots, t$.

So the max pooling operation takes always the largest value at the region. Another simple pooling strategy is to use *average pooling*, which takes the mean over the pooling region.

Definition 4.4 (Average pooling)**.** Given the pooling regions $R_1, R_2, \dots, R_t$ of the input $\mathbf{x}_{l-1}$ as before, the average pooling operation computes

$$x_i^l = \sum_{r \in R_i} \frac{r}{|R_i|},$$

for all $i = 1, 2, \dots, t$, where $|R_i|$ denotes the number of elements in R_i.

Whether max or average pooling is the ideal choice for a network is a data dependent question. However, some empirical results show that max pooling performs slightly better than average pooling in general and some combination of the two tends to outperform both (see for example [3]). Both of the afore mentioned strategies are also deterministic, which can sometimes cause overfitting. *Stochastic pooling* (see [47]) can help with overfitting by introducing stochasticity to the choise of the elements of the output.

Definition 4.5 (Stochastic pooling)**.** In stochastic pooling one counts the probabilities p_j for each element r_j in every pooling region R_i (as defined above) by normalising the elements with

$$p_j = \frac{r_j}{\sum\limits_{r_j \in \mathbb{R}_i} r_j}.$$

These probabilites p_j then define the multinomial distribution from which to sample a location s within the pooling region R_i and the element at that location is used, i.e., for each $i = 1, 2, \dots, t$ we have

$$x_i^l = r_s, \quad \text{where} \quad s \sim P(p_1, p_2, \dots, p_{|R_i|}).$$

According to [47] stochastic pooling can also be viewed as giving copies of the same input to the model, with each copy having different small local distortions.

Stochastic pooling should not be used as is for testing or validation phases as it introduces noise to the network's predictions. Instead in testing or validation phase a weighted sum of the elements in region R_i should be used. That is, compute

$$x_i^l = \sum_{r_j \in R_i} p_j r_j,$$

for all $i = 1, 2, \dots, t$, where the probabilities p_j are computed as before. According to [47] this probabilistic weighting can be seen as model averaging, as in training only some of the nodes get connected to the following layer when stochastic pooling is used, and this connectivity pattern changes at each run. Then in the testing phase using the probabilities as weights to the sum, is similar to averaging over all the afore mentioned models, without the need of training all these models separately.

4.3.2 Upsampling

Upsampling is used to increase the size of input to (or output from) a layer by adding the number of elements in the input. The values in these new nodes can be chosen for example through different interpolation methods, or operations such as *unpooling* or *deconvolution*. Unpooling and deconvolution can be thought of as the inverse operations of pooling and convolution operations respectively, although they are not inverse operations in the mathematical sense as they do not necessarily produce the original input to the corresponding pooling or convolution operation.

A simple interpolation method is *linear interpolation*. In linear interpolation a linear curve is fitted between two consecutive points, and the points to be added are taken from the line. To add a new point to index p in the output of layer l one would compute

$$x_p^l = x_u^{l-1}\frac{v-p}{v-u} + x_v^{l-1}\frac{p-u}{v-u},$$

where $u, v \in \mathbb{Z}_+$ are the indices of the points based on which were are interpolating. *Bilinear interpolation* is a generalisation of linear interpolation to two dimensional space, where the linear interpolation is taken over both dimentions.

Unpooling (see [49]) is used together with pooling. During the pooling phase, the location of each pooled statistic x_i^l within its pooling region R_i is saved. Then in the unpooling phase the value (at the same location as the pooled statistic) x_i^k $(k > l)$ is placed to the same place within the pooling region R_i as the statistic. The other elements within the pooling region obtain some prechosen value, typically zero. Unpooling results in a sparse output and the amount of sparsity depends on the choice of the pooling regions.

Deconvolution (introduced in [48]) does the opposite of convolution in the sense that it expands a given input element according to the kernel k. Deconvolution can also be called transposed convolution or fractionally strided convolution [7]. Deconvolution can be viewed as the backward pass of convolution, when computing the gradient in backpropagation algorithm. Deconvolution can be obtained by multiplying the input with the transpose of the convolution matrix, a specific shape of matrix filled with zeros and repetitions of the kernel on the diagonal, or it can be implemented through convolution by first enlargening the input appropriately with zeros and then applying convolution (see [7]). Deconvolution is more versatile than the other upsampling methods introduced above, as the kernel can be optimised in the training phase as is the case with convolution.

5 Predicting space weather

Space weather, especially at the most active times, can have significant effects on the modern life on Earth, as it can cause different disturbances for example in satellite navigation systems, radio communications or even problems with the power grid [26]. While the large scale structures of the electric currents flowing in the near-Earth space are somewhat known, the more dynamic and small scale behaviour of the currents is can still be difficult to predict accurately. To this end, next the data used to predict the ionospheric currents is described and the implemented neural network architectures will be introduced.

5.1 Data

Previously [22] used a ResNet to predict the field-aligned current measurements provided by the satellites in AMPERE project. Following that idea this thesis aims to predict the equivalent current vectors measured on Earth as given by the *SuperMAG* collaboration on the high polar latitudes (see [10]). The data for SuperMAG is gathered from different organisations and agencies operating and maintaining nearly 600 magnetometers all around the world [28]. As described in [10, 29] the raw data coming in various forms is resampled to 1-minute temporal resolution, units are converted to nanotesla, the data is rotated into local magnetic coordinate system, the NEZ-system, and the baseline caused by the Earth's magnetic field is removed. In the NEZ-system the measured magnetic perturbations are defined by three components, the local magnetic north component B_N pointing towards the magnetic pole, the local magnetic east component B_E, and the vertically down component B_Z. The rotation to this system is done so that the magnetic east component is minimised and the north component maximised.

As stated previously in section 2.3 the magnetic perturbation measurements can be rotated to produce the equivalent current maps. For this thesis the east and north components of the equivalent current vectors, deemed as J_e and J_n respectively, were chosen as the target. These current vectors (or their corresponding magnetic perturbations) can be plotted spatially over a grid of *magnetic latitude* (MLAT) and *magnetic local time* (MLT) coordinates. Magnetic latitude is similar to geographic latitude but instead of being centered to geographich north or south pole magnetic latitude is centered to the magnetic north or south pole. Magnetic local time is similar to longitude but instead of being fixed with respect to Earth it is fixed with respect to the Sun. It is defined against magnetic longitude, which is orthogonal to magnetic latitude, and follows the time of day in the sense that at the points where MLT is equal to 12 the Sun is directly overhead (noon) and where MLT is 24 it is midnight. For a closer description of the magnetic coordinate systems see [23].

The chosen target had a grid size 25×24 encompassing the magnetic latitudes from 40 degrees north to all the way up to the North pole every 2 degrees, and all the magnetic local times every 1 hour. Thus the target for one observation consisted of two matrices in $\mathbb{R}^{25\times 24}$ (or vectors in $\mathbb{R}^{600}$ depending on the presentation) determining the east and north components of the equivalent current vectors at all spatial locations at a chosen minute. See Figure 5 for an example of a polar plot of the equivalent current vectors and the grid. This target was predicted based on a timeseries of thirteen input variables, and for each observation the target was the equivalent current vectors at the time point at the end of that time series. The training data consisted of the solar cycle 23 consisting of the years 1997 to 2008. (As the whole years were used, this time interval actually extended slightly over the solar cycle.) For validation the year 2015 was used. Only one year was chosen for validation in order to make the training faster. For testing, the year 2016 was used.

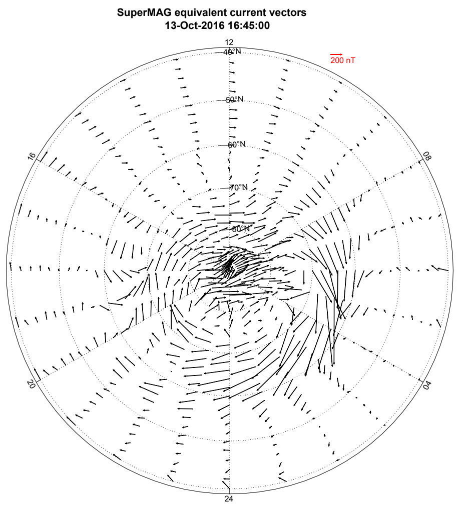

Figure 5: An example of the target for one of the observations depicted as a polar plot.

The thirteen variables used for the input were the x, y, and z components of the interplanetary magnetic field (Bx, By, and Bz, respectively), the flow speed (v) and proton density (Np) of the solar wind, the solar radio flux at 10.7 cm also known as f10.7 index, five provisional activity indexes (SME, SMU, SML, ASY-H and SYM-H), and sine and cosine transformation of the date number ($\mathrm{dn}_{\mathrm{sin}}$ and $\mathrm{dn}_{\mathrm{cos}}$, respectively). These were mainly acquired from NASA/GSFC's OMNI data set through OMNIWeb (see [31]), except for the SMU, SML and SME indices, which were part of the SuperMAG data [27]. All of the input variables obtain a scalar value for every minute of the data.

The IMF coordinates, solar wind speed and density and the f10.7 index, an indicator of solar activity (see e.g. [40]), describe the activity of the Sun and thus the interplanetary conditions. The SMU and SML indices (see [27]) describe the auroral electrojet activity by giving the largest and smallest value for the north component of the magnetic perturbations at every time point. The SME index is the difference between the largest and the smallest north component, i.e., SML substracted from SMU. SYM-H index describes the symmetric (average) north component of the magnetic disturbances at mid-latitudes, and is used to describe the symmetric part of the storm time ring current. ASY-H index describes the asymmetric portion of the north component, and thus the asymmetric portion of the storm time ring current (see [18]). The sine and cosine transformations of the date number combined are used to indicate the time of the year and help to encode the seasonal pattern to the model.

5.1.1 Preparation

As said the data for the target and the SME, SMU, and SML indices was accessed from the SuperMag site, and the rest of the input data was accessed from the OMNIWeb. Under 10 minute gaps of missing values in the input data were interpolated using linear interpolation, and gaps beyond this were imputed with zeros, effectively setting these input values to the average of that input variable. The sine and cosine transformations of the date number were obtained by first shifting the date number to start from zero and then scaling with the last date number of the first year of data. In other words

$$\mathrm{dn}_{\mathrm{sin}} = \sin\left(2\pi\left(\frac{\mathrm{dn} - \mathrm{dn}_0}{\mathrm{dn}_{525599} - \mathrm{dn}_0}\right)\right)$$

for the sine transformation, and

$$\mathrm{dn}_{\mathrm{cos}} = \cos\left(2\pi\left(\frac{\mathrm{dn} - \mathrm{dn}_0}{\mathrm{dn}_{525599} - \mathrm{dn}_0}\right)\right)$$

for the cosine transformation. Here dn_0 stands for the first date number and dn_{525599} stands for the last date number in the first year of data. Transforming the date number this way allows to distinguish between different seasons as the transforms have a full cycle in one year, but they are at a different phase.

After this a z-score was taken from each input variable x^i, where $i = 1, 2, \ldots, 13$, by substracting the empirical mean $\hat{x}^i$ computed over the training set and dividing by the empirical standard deviation computed over the training set s^i. In other words

$$x^i_{\text{z-scored}} = \frac{x^i - \hat{x}^i}{s^i}$$

Z-scoring gives each variable zero mean and unit variance, which helps the training process by simplifying (making it more symmetric?) the loss lanscape [45]. The means and standard deviations used for z-scoring are in Table 1. The validation and test data were z-scored using the means and standard deviations from the training set.

Variable	mean	SD
Bx	-0.06	3.94
By	0.11	4.45
Bz	-0.04	3.75
v	446.43	106.87
Np	6.27	5.21
f10.7	120.18	47.69
SME	250.74	248.04
SMU	96.90	88.48
SML	-153.84	181.63
ASY-H	22.06	17.61
SYM-H	-13.86	22.06
dn_{sin}	$17.76 \cdot 10^{-6}$	0.71
dn_{cos}	$0.69 \cdot 10^{-3}$	0.71

Table 1: Means and standard deviations (SD) of the input variables.

5.2 Model architectures and training

Below the general features of the architectures chosen for this thesis and some of the initial architectural choices are described. Also a visual depiction of the networks are given. It should be noted that instead of describing the networks through the node presentation (as is done in section 3.2), these figures represent the network architectures in layer wise manner, where the dimensions of the input to a layer is described with a box and the arrows describe the computations done in (or between) layers. As for the training of the neural networks Adam was used for optimisation with the default choices of 0.001 learning rate and 0.9 and 0.999 for the decay parameters of the first and second moments respectively. These are also suggested in [21]. For the loss function the squared error was used. The training was ran for two to three *epochs*, which are complete run throughs of the entire training data set, and 128 observations constituted a mini-batch. The number of minutes in a time-series was chosen to be 60 following the convention in [22].

5.2.1 ResNet

The implemented ResNet was chosen to have two to three blocks of layers that are connected through the identity mapping, and a final linear, also known as fully connected, layer that flattens the output into two vectors of 600 elements and takes a linear combination of them. Each block consists of two convolution layers with batch normalisation and ReLU activations. Batch normalisation (see [17]) computes the means and variances of the mini-batch it receives as input during training, and reduces the means from the input and divides it with the standard deviations (this is done for each element in the input). Then it scales and shifts the elements as determined by trainable parameters γ and β. This is intended to ease the learning by fixing the distribution of the inputs to the following layer while still maintaining the representation abilities of a layer through the linear transform. For validation and testing phases, batch normalisation uses estimates of means and variances gathered from the training set for normalisation. The skip-connection was done by adding the input of a block to the output through identity when the dimensions matched. The final activation with ReLU was done after the skip-connection. See Figure 6 for a visual depiction of one of the implemented architectures.

Having two blocks in the network appeared to be the most appropriate choise as the model did not start over-fitting, but could still learn as well as a three blocked version. However, as the squared error does not provide complete information, also the three blocked version was assesed. Similarly, while having 32 feature maps appeared an adequate amount, also 64 maps were tested and assessed to further see which choise appeared better. The convolution was done with zero-padding so that the size of the output did not change through convolution. Zero-padding means that the size of the input to a convolution layer is padded with zeros at the edges, which effectively allows the convolution to be used even for the edges. (This is also how the convolutional layer is described in the Section 4) For the skip-connection, the identity was used, except for the case of the first skip-connection, which required the input to have more dimensions to be added to the output of that block thanks to the larger number of feature maps. To achieve matching dimensions, a convolution with 1×1 kernel was used as suggested in [14]. Following the idea in [22], no maxpooling was used.

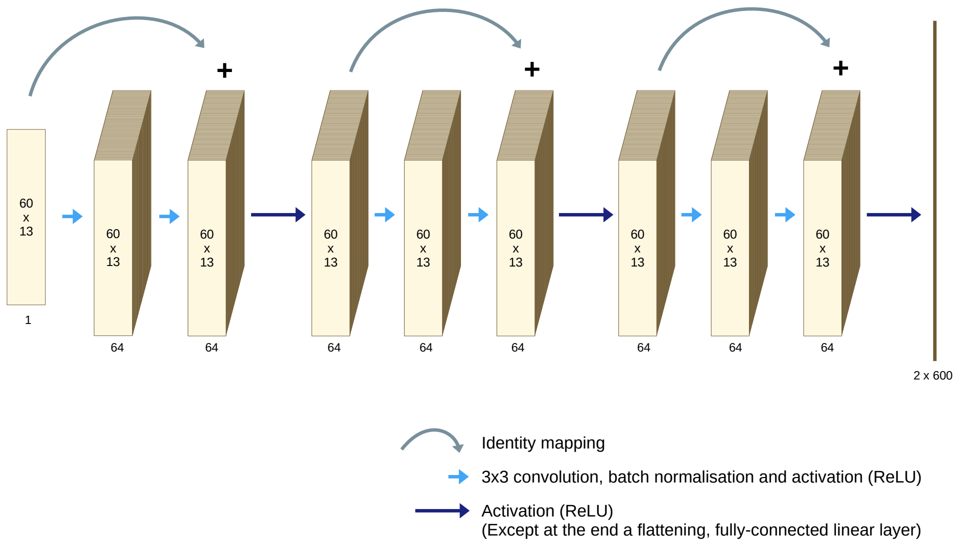
60 x 13
1
60 x 13
64
2 x 600
Identity mapping
3x3 convolution, batch normalisation and activation (ReLU)
Activation (ReLU)
(Except at the end a flattening, fully-connected linear layer)

Figure 6: The implemented ResNet consisting of three blocks and having 64 feature maps in convolutions, see text for details.

5.2.2 U-Net

The implemented U-Net had two downsampling operations meaning the contracting path had three scales altogether. Similar to the implemented ResNet, each scale (on the contracting path) had a block consisting of two convolutional layers with batch normalisation and ReLU activation. On the expanding path the blocks were similar to those on the contracting path, except that the input to a block was concatenated with the output of a block at the same scale on the contracting path. See Figure 7 for a visual depiction. As the input to the neural network (a 60×13 matrix) had a fairly different shape to the output (two 25×24 matrices), the down- and up-samplings needed to be done more or less asymmetrically to obtain the correct shape. Similarly to this, the skip-connections could not simply crop and concatenate the outputs to inputs as is done in [34], but had to also do some resizing. After the last block on the expanding path a convolution with kernel of size 1×1 was done to finally reduce the number of feature maps to two to obtain the correct output shape.

At an initial state it was validated that a shallower U-Net would not be able to produce good results, while the implemented version did produce results comparable to those of the implemented ResNets. As the deeper U-Net is heavier to train, a deeper version still was not even tested after concluding that the results were adequate. For the lowest scale 16, 32 and 64 feature maps were tested. Also following the choices in the ResNet, a kernel of size 3×3 was used for convolution and zero-padding. For downsampling max-pooling with pooling region of size 2×1 was used, effectively halving the height of the second dimension at each downsampling. After downsampling the number of feature maps was doubled following the common convention used for example in [34]. For upsampling transposed convolution was used with first kernel of size 6×7 and then with kernel of size 6×6 without padding to obtain the correct output shape. To be able to concatenate the outputs of the contracting path to the inputs of the expanding path, the image was resized bilinearly.

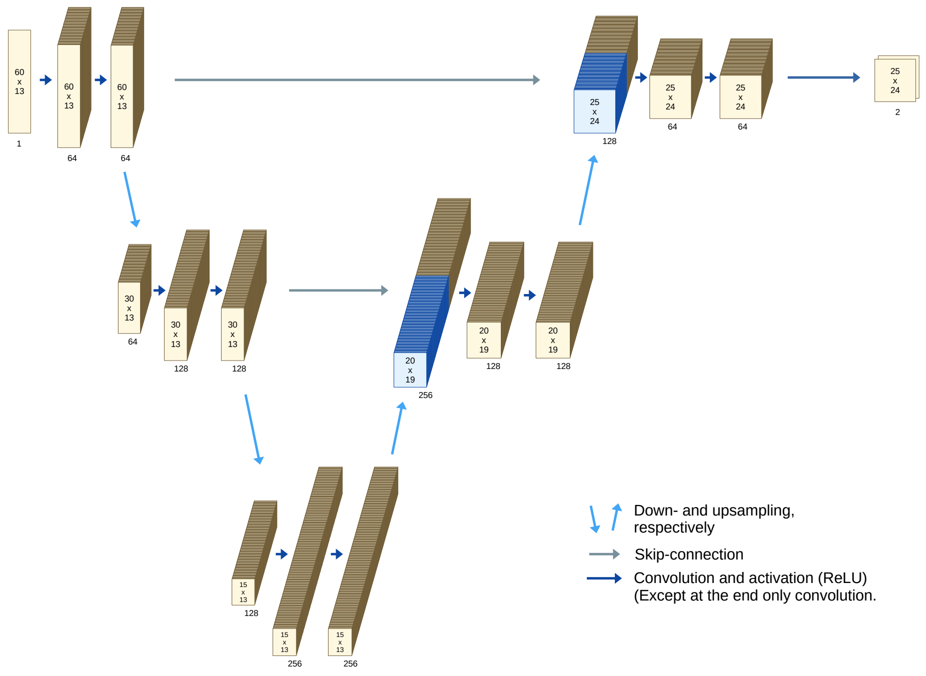

Figure 7: The implemented U-Net, see text for details.

5.3 Results

To assess the model performance during training and to decide at which state each of the architectures were producing the best fitting model, the training and validation losses were monitored. In Figure 8 the losses are depicted as root mean squared errors, as that gives a more intuitive scale for them, although the original loss metric was squared error. To make the plots more readable, only the most relevant architectures are included, and for the training loss a sliding window mean of the 100 nearest losses is depicted to have a better sense of the general behaviour of the losses. Based on the validation loss in the top panel of Figure 8, it appears that the versions with at least 64 feature maps begin to overfit, although even for the ResNet the increase is around 1 nT and thus fairly small. All the versions with 32 feature maps appear to be performing as well as the larger ones, and not overfit similarly, although in the case of the ResNets, they too begin to overfit slightly as the training proceeds. Based on the validation loss, the U-Net architectures appear to be performing worse than the tested ResNets in general, although again the difference is around 1 nT. For further testing the best performing model was chosen, although a run of at least one epoch was required, so that the neural network would have seen the complete training dataset.

For further investigation, the average differences between the norms and the angles of the targets y and the predictions of the neural network $\hat{y}$ over the test data were computed for both the temporal and the spatial dimensions. In the following the east component of the target will be marked with y^1 and north component with y^2 (or $\hat{y}^1$ and $\hat{y}^2$ for the predictions of the neural network). The observations will be considered to be vectors of 600 elements, and the number of observations in the target will be marked with t. Subscripts i and j will be used to denote the element in temporal and spatial dimensions respectively, i.e., $y^1_{i,j}$ is the east component of the equivalent current vector at time i and location j, or the jth element of the ith observation.

To compute the average difference in norms over all locations, first the euclidian norm of both the target and the output is computed as $l_{i,j} = \sqrt{(y^1_{i,j})^2 + (y^2_{i,j})^2}$ and $\hat{l}_{i,j} = \sqrt{(\hat{y}^1_{i,j})^2 + (\hat{y}^2_{i,j})^2}$. Then the average difference is taken over all the locations $j = 1, 2, \ldots, 600$ at time point i,

$$r^i_{\text{norm}} = \frac{1}{s}\sum_{j=1}^{s} l^i_j - \hat{l}^i_j.$$

To obtain the average over time at location j

$$r^j_{\text{norm}} = \frac{1}{t}\sum_{i=1}^{t} l^i_j - \hat{l}^i_j$$

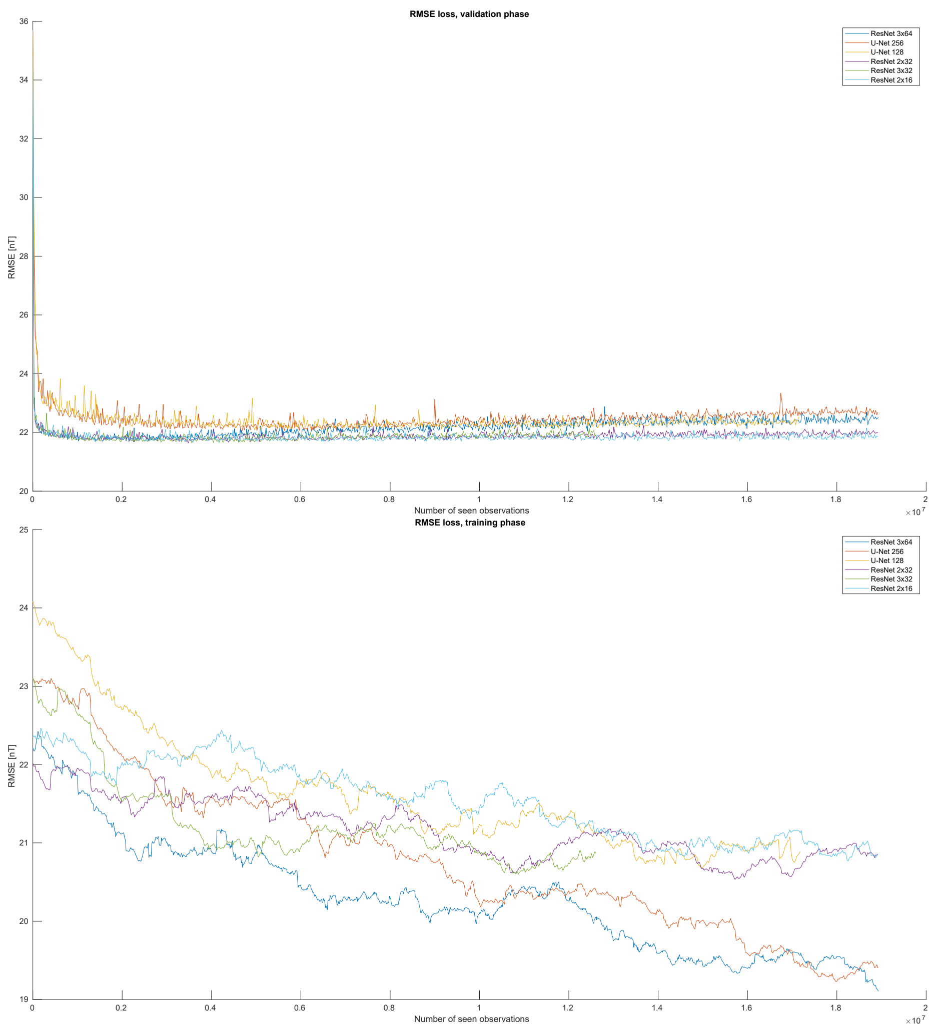

Figure 8: The validation (top) and training (bottom) losses as RMSE for all the trained models.

is computed. In the top panel of Figure 9 means of the averages over locations for norms are depicted against the SME index. The SME index is binned to intervals $[0, 100[, [100, 200[, \cdots, [2600, 2700[$, and the means are computed over these bins. It should be noted that the number of observations within the bin decreases with the growing SME index, and this is presented by the changing circle size.

In the middle panel of Figure 9 the average difference in norms over locations $j = 1, 2, \ldots, 600$ is related to the maximum norm length at that time, i.e.,

$$r^i_{\text{norm}} = \frac{1}{s}\left(\max_{j\in\{1,2,\ldots,600\}} l^i_j\right)^{-1} \sum_{j=1}^{s} l^i_j - \hat{l}^i_j$$

is computed. This plotted against the SME index as before. Similarly the average of relative difference in norms over time is computed as

$$r^j_{\text{norm}} = \frac{1}{t}\left(\max_{i\in\{1,2,\ldots,t\}} l^i_j\right)^{-1} \sum_{i=1}^{t} l^i_j - \hat{l}^i_j.$$

In Figure 10 the average relative differences are plotted for four different models.

The average difference in angles over locations is obtained by first computing the angles for both the target and the output as four-quadrant inverse tangent, which gives the angle as radians in range $]-\pi,\pi]$. The computed angles for the target $\gamma_{i,j}$ and output $\hat{\gamma}_{i,j}$ are changed to degrees through multiplication by $\frac{180}{\pi}$. Then the absolute value of the difference between the angles is taken as $a_{i,j} = |\gamma_{i,j} - \hat{\gamma}_{i,j}|$ for all locations j at time point i. If the difference is over 180°, it is substracted from 360 to obtain the actual difference ($\hat{a}_{i,j} = 360 - a_{i,j}$, when $a_{i,j} > 180$). The average is taken over all the locations $j = 1, 2, \ldots, 600$ at time point i as

$$r^i_{\text{angle}} = \frac{1}{s}\sum_{j=1}^{s} \hat{a}_{i,j}.$$

To obtain the average difference in angle over time at location j,

$$r^j_{\text{angle}} = \frac{1}{t}\sum_{i=1}^{t} \hat{a}_{i,j}$$

is computed. As previously, the averages over locations for the angles are depicted against the SME index in the bottom panel of Figure 9. The average differences in angles are depicted in Figure 11 for the four most interesting models, as the to lessen the number of plots. The four chosen models were ResNet with two blocks and 32 feature maps (deemed as ResNet 2x32), ResNets with three blocks and 32 and 64 feature maps (ResNet 3x32 and 3x64, respectively) and the largest U-Net with 256 feature maps at the lowest resolution (U-Net 256). In Figure 15 there are predictions for the ResNet 2x32 model and the corresponding targets for a time which was predicted as falsely active, and one which was predicted quiet, although the target is more active. While similar times exist for the other models as well, these plots were similar to others, and so these are shown only for one model.

To further observe the different behaviour at different times and have a clearer understanding of the error, different polar plots were produced. In Figure 12 the predictions of three chosen models are plotted with the corresponding target when the equivalent current vectors are at their largest on average. The models chosen for this were ResNet 2x3, ResNet 3x64 and U-Net 256. In Figure 13 the predictions and the corresponding target are depicted at the time point, when the SME index was at its largest, for the same three models. Also the time point when the difference in norms was at its largest on average is shown in Figure 14 for the three architectures. For U-Net this was not exactly the time point with the largest difference, as that was minute earlier, but the results were very similar, and thus for simplicity the one shown was chosen.

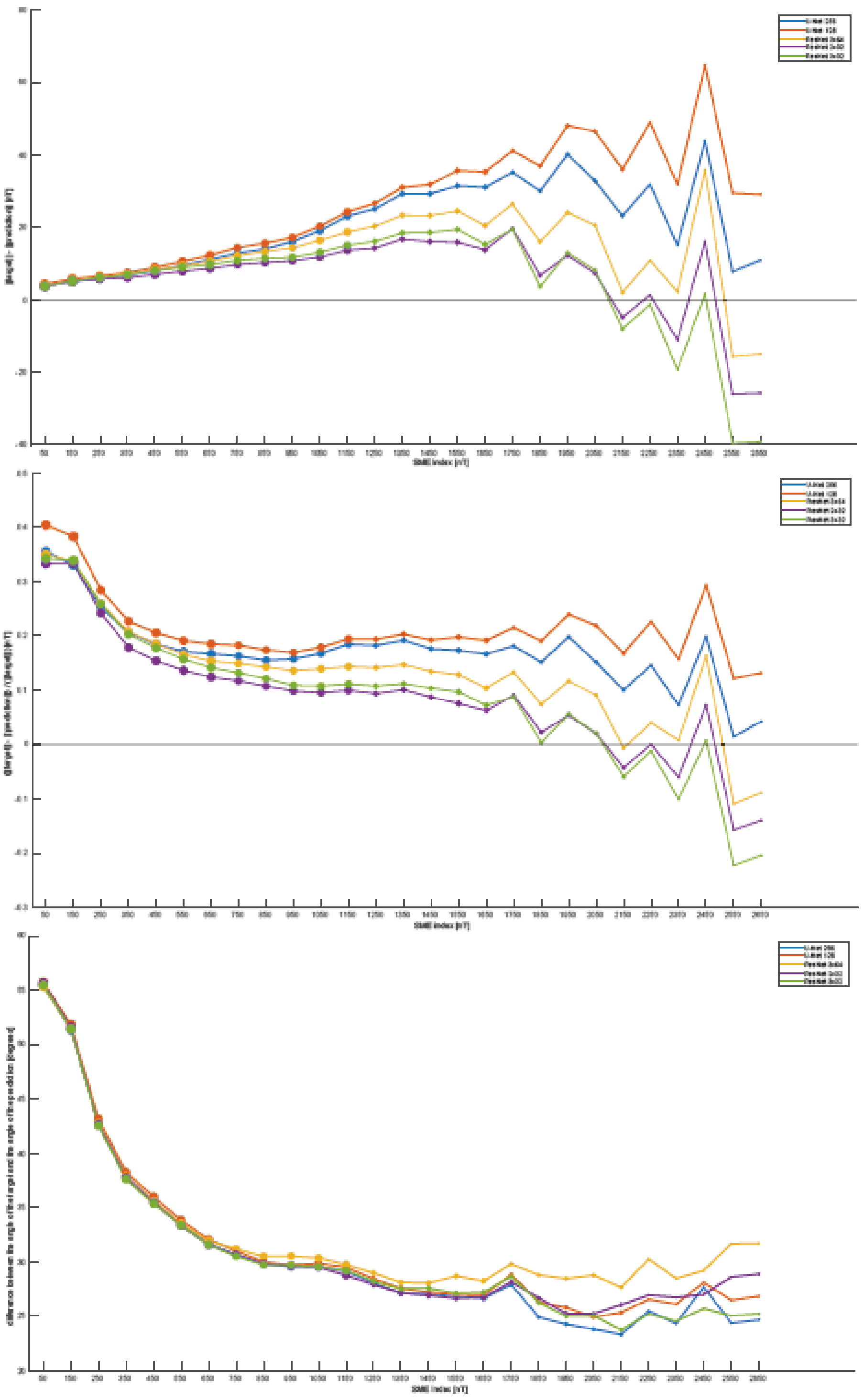

Figure 9: The mean difference over locations in norms (top) and angles (bottom) against the SME index for chosen models.

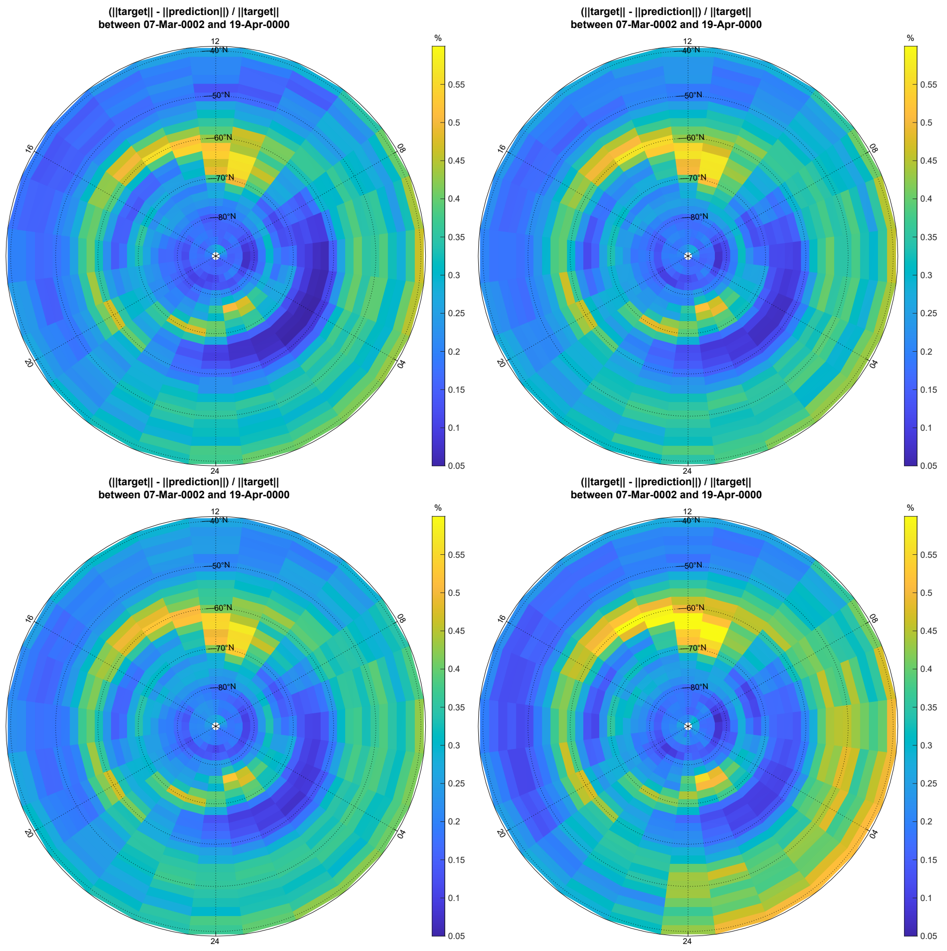

Figure 10: The average relative difference in norms over time for architectures ResNet 2x32 (top left), ResNet 3x32 (top right), ResNet 3x64 (bottom left) and U-Net 256 (bottom right).

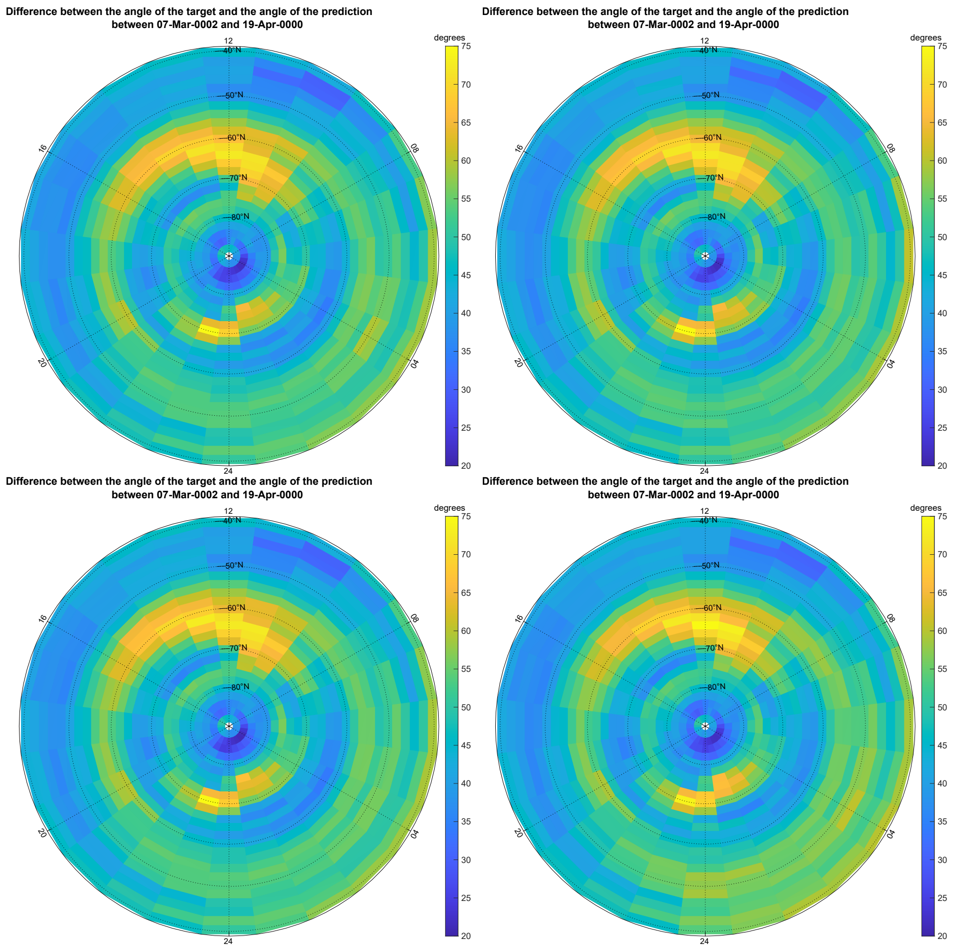

Figure 11: The average difference in angles over time for architectures ResNet 2x32 (top left), ResNet 3x32 (top right), ResNet 3x64 (bottom left) and U-Net 256 (bottom right).

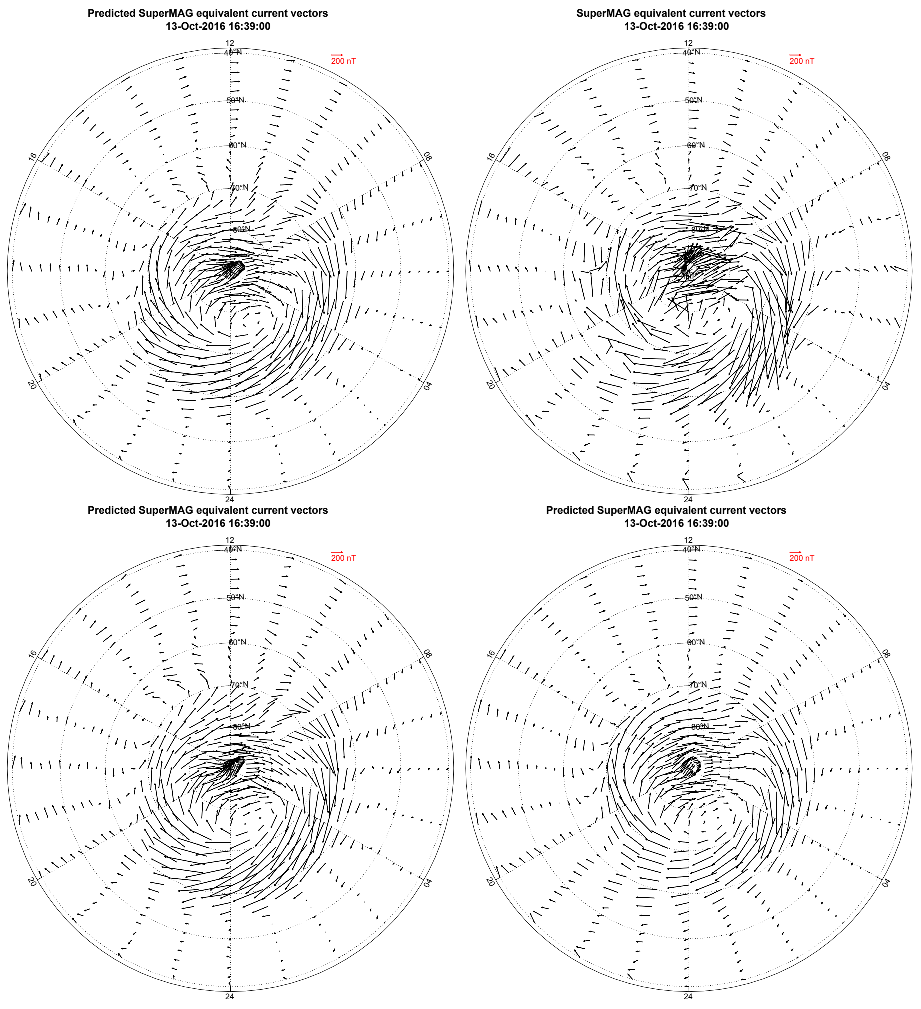

Figure 12: Polar plots of the target (top right panel) and the predictions of the different models when the equivalent currents vectors were at their largest on average. Predictions are from models ResNet 2x32 (top left), ResNet 3x64 (bottom left) and U-Net 256 (bottom right).

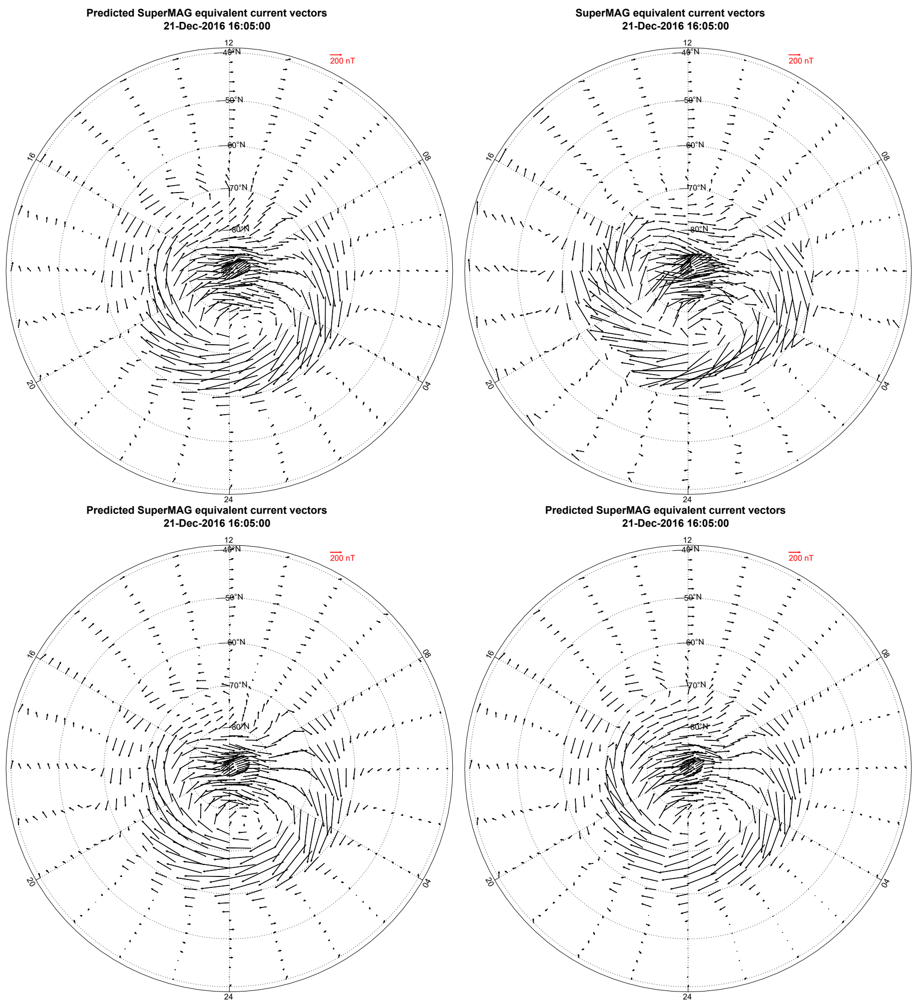

Figure 13: Polar plots of the target (top right panel) and the predictions of the different models when the SME index was at its largest in 2016. Predictions are from models ResNet 2x32 (top left), ResNet 3x64 (bottom left) and U-Net 256 (bottom right).

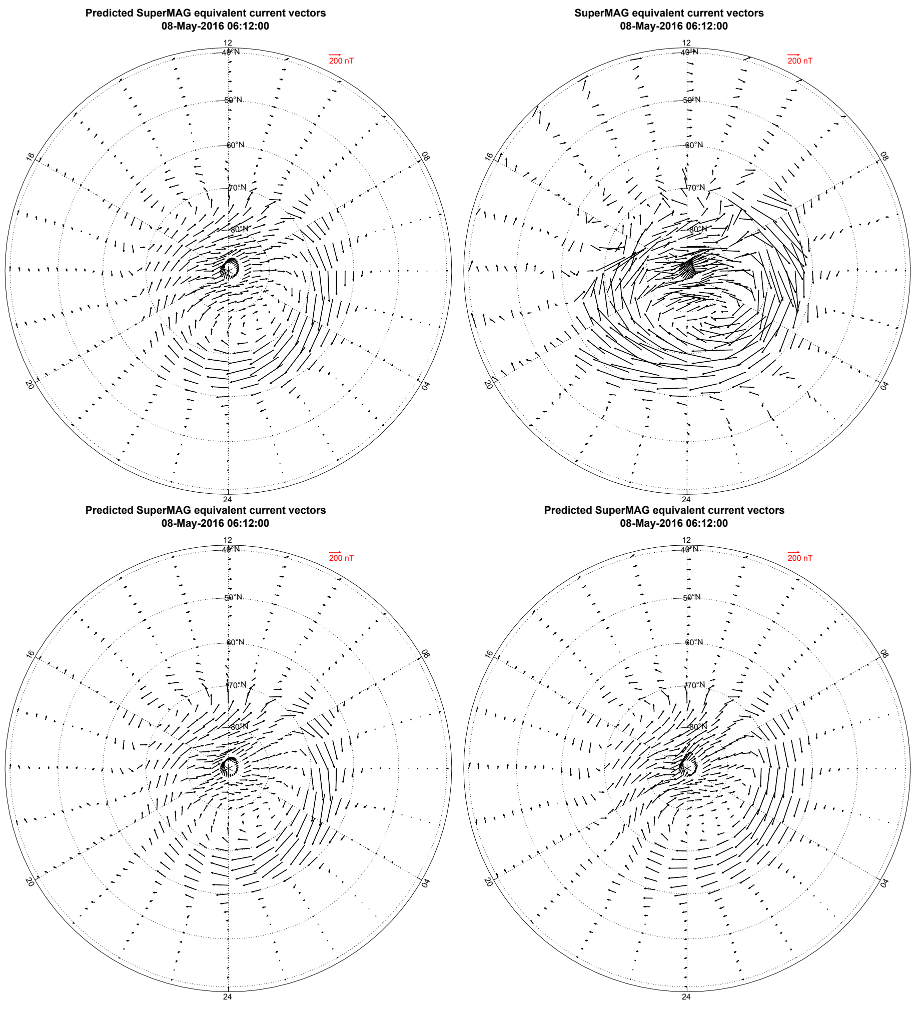

Figure 14: Polar plots of the target (top right panel) and the predictions of the different models for the time point when the average difference between the norms was (nearly) absolutely largest. Predictions are from models ResNet 2x32 (top left), ResNet 3x64 (bottom left) and U-Net 256 (bottom right).

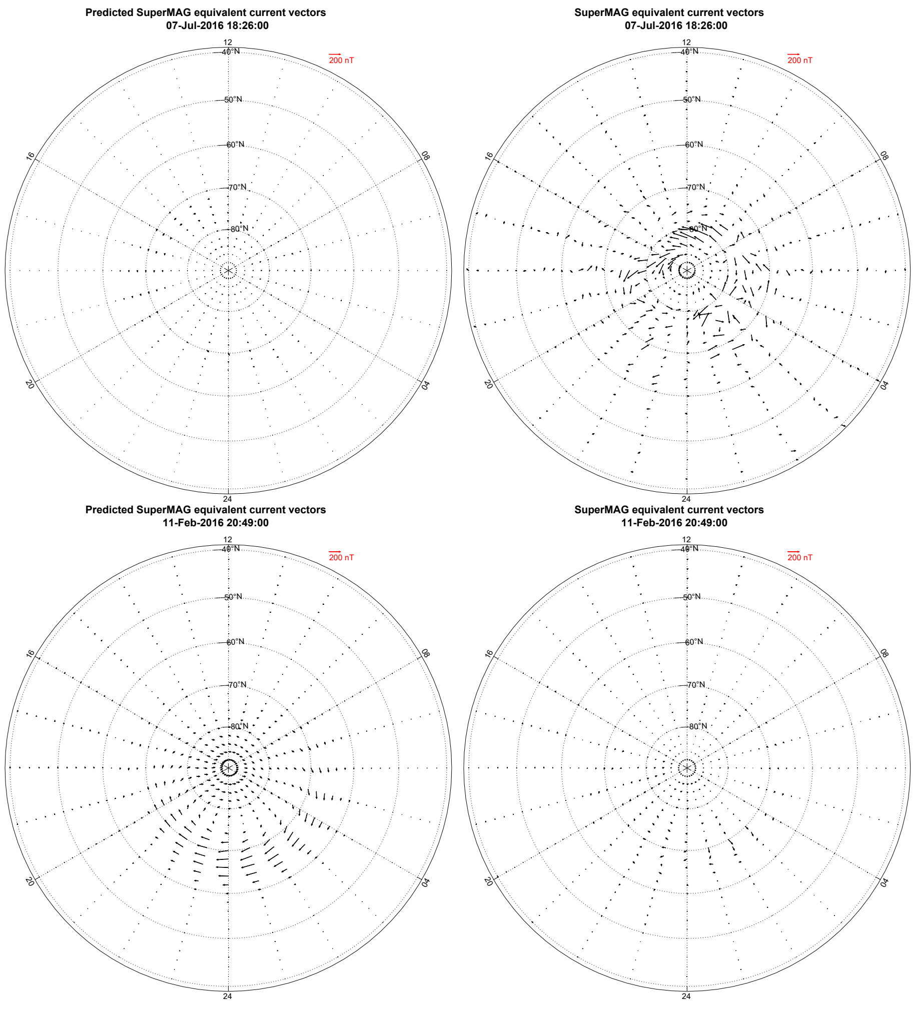

Figure 15: Polar plots of the targets (right panels) and the predictions of the ResNet 2x32 (left panels) to showcase a few uncommon times.

6 Discussion

In this section the results shown in Section 5.3 are discussed further. First the validation results are evaluated and the architecture and hyperparameter choices are discussed. Then the few chosen models are compared further, first looking at their general performance over the complete test dataset, and then at few individual time points to have a better understandment as to where the differences may be occuring.

6.1 Architecture and hyperparameters

To choose the best performing models for further evaluation, one needs to look at the validation losses. Based on these losses (see Figure 8 top panel), it would appear that having 64 feature maps does not significantly improve the model and only results in overfitting, and for both the ResNets and U-Nets 32 appears an adequate amount (in the case of U-Nets this is the number of feature maps at the scale with the highest resolution. Having 32 feature maps at the highest scale results in 128 feature maps for the lowest resolution, which is the number used in the legends to separate between U-Nets). However, in the case of ResNet even 16 feature maps appears to produce comparable results. Also it seems that having only two blocks for ResNet appears to be a suitable depth, as increasing the depth does not significantly improve the results. Based on the training losses in Figure 8 bottom panel (and the gradients, which were monitored but are not plotted here) none of the models have reached a minima, but as there were no improvements in the validation losses, the training was still halted to save computation time. For the smallest models that were not overfitting (mainly ResNet with 16 feature maps), a further training may still improve their results.

It appears that all of the models plotted in Figure 8 achieve similar levels of performance on the training and validation sets, after which they are more inclined to start overfitting the training data, with the largest architectures showing this behaviour clearly sooner than others. This indicates that the larger models are incapable of extracting more general information from the training data then their shallower counterparts, even though they would have the capacity. This may be a problem with some of the untested hyperparameters, such as the learning rate or the error metric. These were left out to lessen the number of tunable hyperparameters, but they may still change the results. Poor initialisation may cause issues, such as the dying ReLU problem, but that does not seem to be the case here, as the gradients were still fairly big and the training error was becoming smaller. It may also be, that it is not possible to extract more general features from the training data. There is also a separation in performance on the validation data between ResNets and U-Nets with U-Nets performing worse in general. This is, however, not visible in the training loss, which may indicate that U-Nets are either learning features more inherent to the training data, and thus generalise worse on new data, or then the one year validation data may be missing some features, that are present in the training data, and for which the U-Nets are actually performing better.

6.2 Quantitative analysis

As the validation loss encompasses only one year and truncates the difference between the target and prediction into one statistic, more evaluation was required to have a clearer picture of the differences between the models. The average difference in norms over locations (see Figure 9 top panel) appears to be very similar for all of the architectures when the SME index is below 250 nT. After this, the ResNets with 32 feature maps begin to perform seemingly better to the rest of the models. When the SME index reaches values beyond 1600 nT, the errors begin to oscillate for all of the observed networks. This is because for higher values of SME index there are less observations, and especially for index values beyond 2200 nT the average is taken over 5 to 2 time points meaning that an unusual observation may throw the average off. While all of the models have a tendency to underestimate the length of the vectors, both U-Nets tend to do this most on average, as the difference in norm is generally highest for them, and slightly suprisingly especially for the U-Net with 128 feature maps on the lowest resolution, despite the validation loss showing similar performance to the larger U-Net. All of these features are still visible when comparing the relative differences (see the middle panel of Figure 9).

For the average difference in angles over locations (see Figure 9 bottom panel, the largest U-Net actually appears to be estimating the directions best in general, although for SME index values below 1700 nT there are not many differences between the models. When the SME index begins to reach more uncommon values, reaching beyond 2100 nT, the ResNet with three blocks and 32 feature maps appears to be performing best, although the largest U-Net is close or better on average. However, at these points the averages are taken over only few observations, and as before an odd observation may be throwing the error off. Similarly to the relative difference in norms, the most quiet times (having the lowest SME index values) have highest errors. One explanation as to why all the models struggle in predicting the quiet times correctly may be that the inevitable noise in the measurements is more dominant. Alternatively the chosen error metric, squared error, may be ignoring the more quiet times as it emphasizes larger errors. This is not necessarily unwanted behaviour, as the prediction of the more active times is more of interest.

For the average relative differences in norms and average differences in angles over time (see Figures 10 and 11) all of the models appear to be producing similar patterns with the dawn side at the lowest magnetic latitudes and noon or afternoon at magnetic latitudes $60 - 70$ degrees north being the hardest to predict. From the relative error in norms, it is clear that the U-Net has the hardest time getting the lengths of the vectors right, and slightly suprisingly (when comparing to the behaviour seen in Figure 9 bottom panel) is also getting the angles slightly more wrong than the other architectures. As for the ResNets there are no significant differences, although it seems that the largest ResNet is predicting the lengths of the vectors at the lowest magnetic latitudes slightly better than the others while the two blocked version appears to be most on point at inner areas, and both three blocked ResNets appear to predict the angles at the edges slightly better than the two blocked version. It may be that the 64 feature maps are capable at cathcing the effect of ring currents better, although it is difficult to say without further analysis. This will, however, be left out as it is outside the scope of this thesis.

6.3 Qualitative analysis

At invidual time points only three models were assessed to lessen the number of figures. These models were ResNet 2x32, ResNet 3x64 and U-Net 256, mainly as these were the best performing models with most differences between them. Even so, there are no substantial differences between the predictions, especially when comparing the two ResNets (see later figures in Section 5.3). All of the models predict cleaner equivalent current vectors, and typically underestimate the length especially for the largest vectors. However, they all seem to also catch the general patterns fairly well. The differences between the two ResNets are focusing mainly to the center, although it appears the ResNet with three blocks has more changes at lower latitudes than the two blocked version. Comparing U-Net to the ResNets, U-Net tends to produce smoother plots, with a greater underestimation of the vector lengths as was seen from the previous comparisons. This could be explained by the difference between the fully convolutional U-Net and the ResNets that end with a linear layer. As U-Net is fully convolutional, it encodes the location information into the output. This may cause it to encode too high spatial dependency between the vectors. In Figure 15 a falsely active time is shown in the bottom panels, and a falsely quiet time in the top panels. However, for the falsely quiet time, the target is fairly chaotic, possibly indicating erroneuos measurements.

7 Conclusions

In general the tested ResNets and U-Nets were capable of producing the equivalent current vectors fairly well, and especially for more active times. While all of the models were able to predict the general patterns well, they all tended to underestimate length of the largest vectors. This was especially true for U-Net, which may be explained by the stronger spatial dependency in the model, resulting in overly smooth polar plots and underestimated vector lengths. Thismay indicate the necessity of the final linear layer. Having a ResNet with two blocks and 32 feature maps appeared to be adequate, although a version with three blocks appeared to catch the variability at the lowest latitudes better, and there did not seem to be any significant differences between the predictions.

Overall it would seem that convolutional neural networks are capable of predicting the ionospheric currents from time-series data, at least at the end of the time-series, and even a fairly light-weight neural network suffices. Future research may look into testing different error metrics to see if it were possible to overcome the underestimation problem. It should also be tested how far into the future the predictions can be made based on the given time-series. And of course, there are multiple different artificial neural network architectures that may have the potential to perform better in this particular task, and finding the optimal one is a question of its own.

References

[1] Julius Berner, Philipp Grohs, Gitta Kutyniok, and Philipp Petersen, *The modern mathematics of deep learning*, Mathematical Aspects of Deep Learning, Cambridge University Press, dec 2022, `https://doi.org/10.1017/9781009025096.002`, pp. 1–111.

[2] David Bertoin, Jérôme Bolte, Sébastien Gerchinovitz, and Edouard Pauwels, *Numerical influence of relu'(0) on backpropagation*, Advances in Neural Information Processing Systems (Paris, France), Advances in Neural Information Processing Systems 34 (NeurIPS 2021), December 2021, `https://hal.science/hal-03265059`.

[3] Y-Lan Boureau, Jean Ponce, and Yann LeCun, *A theoretical analysis of feature pooling in visual recognition*, Proceedings of the 27th International Conference on International Conference on Machine Learning (Madison, WI, USA), ICML'10, Omnipress, 2010, p. 111–118.

[4] Lorenzo Ciampiconi, Adam Elwood, Marco Leonardi, Ashraf Mohamed, and Alessandro Rozza, *A survey and taxonomy of loss functions in machine learning*, 2023, `https://doi.org/10.48550/arXiv.2301.05579`.

[5] Ronald DeVore, Boris Hanin, and Guergana Petrova, *Neural network approximation*, 2020, `https://doi.org/10.48550/arXiv.2012.14501`.

[6] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri, *Activation functions in deep learning: A comprehensive survey and benchmark*, Neurocomputing **503** (2022), 92–108, `https://doi.org/10.1016/j.neucom.2022.06.111`.

[7] Vincent Dumoulin and Francesco Visin, *A guide to convolution arithmetic for deep learning*, 2016, `https://doi.org/10.48550/ARXIV.1603.07285`.

[8] Natalia Ganushkina, Michael Liemohn, and Stepan Dubyagin, *Current systems in the earth's magnetosphere*, Reviews of Geophysics **56** (2018), no. 2, 309–332, `https://doi.org/https://doi.org/10.1002/2017RG000590`.

[9] Natalia Ganushkina, Michael Liemohn, Stepan Dubyagin, Ioannis Daglis, Iannis Dandouras, Darren De Zeeuw, Yusuke Ebihara, Raluca Ilie, Roxanne Katus, Marina Kubyshkina, Steve Milan, Shin Ohtani, Nikolai Ostgaard, Jone Reistad, Paul Tenfjord, Frank Toffoletto, Sorin

Zaharia, and Olga Amariutei, *Defining and resolving current systems in geospace*, Annales Geophysicae **33** (2013), 12721–, https://doi.org/10.5194/angeo-33-1369-2015.

[10] Jesper W. Gjerloev, *The supermag data processing technique*, Journal of Geophysical Research: Space Physics **117** (2012), no. A9, https://doi.org/https://doi.org/10.1029/2012JA017683.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep learning*, MIT Press, 2016, https://www.deeplearningbook.org.

[12] Benjamin Graham, *Fractional max-pooling*, 2014, https://doi.org/10.48550/ARXIV.1412.6071.

[13] Rémi Gribonval, Gitta Kutyniok, Morten Nielsen, and Felix Voigtlaender, *Approximation spaces of deep neural networks*, 2020, https://doi.org/10.48550/arXiv.1905.01208.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, *Deep residual learning for image recognition*, 2015, https://doi.org/10.48550/arXiv.1512.03385.

[15] Kurt Hornik, Maxwell Stinchcombe, and Halbert White, *Multilayer feedforward networks are universal approximators*, Neural Networks **2** (1989), no. 5, 359–366, https://doi.org/10.1016/0893-6080(89)90020-8.

[16] Xia Hu, Lingyang Chu, Jian Pei, Weiqing Liu, and Jiang Bian, *Model complexity of deep learning: A survey*, 2021, https://doi.org/10.48550/arXiv.2103.05127.

[17] Sergey Ioffe and Christian Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, 2015, https://doi.org/10.48550/ARXIV.1502.03167.

[18] T Iyemori, T Araki, T Kamei, and M Takeda, *Mid-latitude geomagnetic indices asy and sym (provisional) no. 1, 1989*, Data Analysis Center for Geomagnetism and Space Magnetism, Kyoto University, Kyoto (1992), Retrieved from https://wdc.kugi.kyoto-u.ac.jp/aeasy/asy.pdf.

[19] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Zídek, Anna Potapenko, Alex Bridgland, Clemens Meyer,

Simon Kohl, Andrew Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, and Demis Hassabis, *Highly accurate protein structure prediction with alphafold*, Nature **596** (2021), 1–11, `https://doi.org/10.1038/s41586-021-03819-2`.

[20] John Kelleher, *Deep learning*, MIT Press essential knowledge series, MIT Press, 2019.

[21] Diederik P. Kingma and Jimmy Ba, *Adam: A method for stochastic optimization*, 2014, `https://doi.org/10.48550/ARXIV.1412.6980`.

[22] Bharat S. R. Kunduri, Maidina Maimaiti, Joseph B. H. Baker, J. Michael Ruohoniemi, Blake J. Anderson, and Sarah K. Vines, *A deep learning-based approach for modeling the dynamics of ampere birkeland currents*, Journal of Geophysical Research: Space Physics **125** (2020), no. 8, e2020JA027908, `https://doi.org/10.1029/2020JA027908`.

[23] Karl Laundal and Arthur Richmond, *Magnetic coordinate systems.*, Space Science Reviews **206** (2017), no. 1-4, 27 – 59, `https://doi.org/10.1007/s11214-016-0275-y`.

[24] Moshe Leshno, Vladimir Ya. Lin, Allan Pinkus, and Shimon Schocken, *Multilayer feedforward networks with a nonpolynomial activation function can approximate any function*, Neural Networks **6** (1993), no. 6, 861–867, `https://doi.org/10.1016/S0893-6080(05)80131-5`.

[25] Steve Milan, Lasse B. N. Clausen, John C. Coxon, Jared A. Carter, Maria-Theresia Walach, Karl Laundal, Nikolai Østgaard, Paul Tenfjord, Jone Peter Reistad, Kristian Snekvik, Hank Korth, and Blake J. Anderson, *Overview of solar wind-magnetosphere-ionosphere-atmosphere coupling and the generation of magnetospheric currents*, Space Science Reviews **206** (2017), no. 1-4, 547–573 (English), `https://doi.org/10.1007/s11214-017-0333-0`.

[26] Steven K. Morley, *Challenges and opportunities in magnetospheric space weather prediction*, Space Weather **18** (2020), no. 3, e2018SW002108, `https://doi.org/10.1029/2018SW002108`.

[27] Patrick T. Newell and Jesper W. Gjerloev, *Evaluation of supermag auroral electrojet indices as indicators of substorms and auroral power*, Journal of Geophysical Research: Space Physics **116** (2011), no. A12, `https://doi.org/10.1029/2011JA016779`.

[28] SuperMAG Organization, *About supermag*, `https://supermag.jhuapl.edu/info/`, Retrieved: July 19, 2023.

[29] ———, *Download data (about)*, `https://supermag.jhuapl.edu/mag/?fidelity=low&tab=description&start=2001-01-29T02%3A30%3A00.000Z&interval=1%3A00%3A00`, Retrieved: July 19, 2023.

[30] ———, *Polar plots (about)*, `https://supermag.jhuapl.edu/rBrowse/?fidelity=low&start=2001-01-29T02%3A30%3A00.000Z&step=300&tab=about&layers=bg-bump-fg-imagefuv-polarvis-magfit-magvec-rbsp.foot-swarm.foot-tgrd-imf.clock-refvec.supermag-refvec.supermag-fit-refvec.ulfband-colorkey.imagefuv-colorkey.polarvis-colorkey.ulf-time-`, Retrieved: July 19, 2023.

[31] Natalia E. Papitashvili and Joseph H. King, *Omni 1-min data [data set]*, `https://doi.org/10.48322/45bb-8792`, 2020, Retrieved: November 20, 2022.

[32] Alexander Piel, *Plasma physics: An introduction to laboratory, space, and fusion plasmas*, Springer Berlin Heidelberg, 2010, `https://doi.org/10.1007/978-3-642-10491-6`.

[33] Andrinandrasana Rasamoelina, Fouzia Adjailia, and Peter Sincak, *A review of activation function for artificial neural network*, 01 2020, `https://doi.org/10.1109/SAMI48414.2020.9108717`, pp. 281–286.

[34] Olaf Ronneberger, Philipp Fischer, and Thomas Brox, *U-net: Convolutional networks for biomedical image segmentation*, 2015, `https://doi.org/10.48550/arXiv.1505.0459`.

[35] Jürgen Schmidhuber, *Deep learning in neural networks: An overview*, Neural Networks **61** (2015), 85–117, `https://doi.org/10.1016/j.neunet.2014.09.003`.

[36] Robert Schunk and Andrew Nagy, *Ionospheres : Physics, plasma physics, and chemistry.*, Cambridge Atmospheric and Space Science Series, vol. 2nd ed, Cambridge University Press, 2009, `https://doi.org/10.1017/CBO9780511635342`.

[37] R. Shankar, *Fundamentals of physics ii : Electromagnetism, optics, and quantum mechanics.*, Fundamentals of Physics, no. II, Electromagnetism, optics, and quantum mechanics, Yale University Press, 2016, `https://doi.org/10.2307/j.ctv10sm90h`.

[38] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George Driessche, Thore Graepel, and Demis Hassabis, *Mastering the game of go without human knowledge*, Nature **550** (2017), 354–359, `https://doi.org/10.1038/nature24270`.

[39] Elias M. Stein and Guido Weiss, *Introduction to fourier analysis on euclidean spaces (pms-32)*, Princeton University Press, 1971, `http://www.jstor.org/stable/j.ctt1bpm9w6`.

[40] Ken F. Tapping, *The 10.7 cm solar radio flux (f10.7)*, Space Weather **11** (2013), no. 7, 394–406, `https://doi.org/10.1002/swe.20064`.

[41] Matus Telgarsky, *Representation benefits of deep feedforward networks*, 2015, `https://doi.org/10.48550/arXiv.1509.08101`.

[42] Matus Telgarsky, *Benefits of depth in neural networks*, 29th Annual Conference on Learning Theory (Columbia University, New York, New York, USA) (Vitaly Feldman, Alexander Rakhlin, and Ohad Shamir, eds.), Proceedings of Machine Learning Research, vol. 49, PMLR, 23–26 Jun 2016, `https://proceedings.mlr.press/v49/telgarsky16.html`, pp. 1517–1539.

[43] Matus Telgarsky, *Deep learning theory lecture notes*, 2021, `https://mjt.cs.illinois.edu/dlt/` Version: 2021-10-27 v0.0-e7150f2d (alpha).

[44] Qi Wang, Yue Ma, Kun Zhao, and Yingjie Tian, *A comprehensive survey of loss functions in machine learning*, Annals of Data Science **9** (2022), `https://doi.org/10.1007/s40745-020-00253-5`.

[45] Jeremy Watt, Reza Borhani, and Aggelos K. Katsaggelos, *Machine learning refined: Foundations, algorithms, and applications*, 2 ed., Cambridge University Press, 2020, `https://doi.org/10.1017/9781108690935`.

[46] D. Randall Wilson and Tony R. Martinez, *The general inefficiency of batch training for gradient descent learning*, Neural Networks **16** (2003), no. 10, 1429–1451, `https://doi.org/10.1016/S0893-6080(03)00138-2`.

[47] Matthew D. Zeiler and Rob Fergus, *Stochastic pooling for regularization of deep convolutional neural networks*, 2013, `https://doi.org/10.48550/ARXIV.1301.3557`.

[48] Matthew D. Zeiler, Dilip Krishnan, Graham W. Taylor, and Rob Fergus, *Deconvolutional networks*, 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2010, `https://doi.org/10.1109/CVPR.2010.5539957`, pp. 2528–2535.

[49] Matthew D. Zeiler, Graham W. Taylor, and Rob Fergus, *Adaptive deconvolutional networks for mid and high level feature learning*, 2011 International Conference on Computer Vision, 2011, `https://doi.org/10.1109/ICCV.2011.6126474`, pp. 2018–2025.