Using Machine Learning to Improve Numerical Weather Prediction

Conor Casey

Teacher: Ms. Abbott

SciFest

This project is submitted into the category:

Earth and Environmental Sciences - Atmospheric Science

EAEV076 - Virtual Regeneron ISEF 2021

“Perhaps some day in the dim future it will be possible to advance the computations faster than the weather advances and at a cost less than the saving to mankind due to the information gained. But that is a dream.”

— Lewis Fry Richardson

Abstract

Weather forecasting is traditionally done by physical models, which are unstable to perturbations. Even the best simulators can be inaccurate due to insufficient knowledge, or difficulty approximating, the underlying physics and parameters. An alternative is to use neural networks to train simulators directly from observed data, as this does not suffer from the same limitations.

My project hypothesizes that it is possible to train a neural network on an atmospheric reanalysis dataset from the past 40 years, that the neural network captures weather patterns and can predict the evolution of the atmosphere, and ultimately that a machine learning model will improve weather prediction compared to physics-based models.

A factor which led to the development of a machine learning model was the expected decrease in computational resources required to generate a forecast. Once the model has trained, a performance increase of 100 times is expected in comparison to a physics-based model of a similar resolution.

The benchmarks have demonstrated that the model can be regarded as useful, as it outperforms persistence and climatological forecasts. The root mean squared error was chosen as the primary error metric because it is easy to compute and mirrors the loss used for most machine learning applications. The performance is roughly equivalent to physical models of a similar, or lower spatial resolution, however, it fails to beat well established, operational physics-based models at this time. The model's spatial awareness, as measured by the anomaly correlation coefficient, can be regarded as quite poor. It appears that the spatial aspect of a weather forecast was not captured.

The results from the benchmarks demonstrate that machine learning can be utilised to improve the computational efficiency of numerical weather prediction and could allow for larger ensemble sizes going forward. This would provide greater certainty in the forecast produced, as well as, requiring less computational resources to generate a forecast of an approximately equal accuracy. As such, the hypothesis can be accepted.

Acknowledgements

Ms. Abbott First and foremost, I would like to express my immense gratitude to my science teacher, Ms Abbott. Over the past five years, she has provided me with an unwavering amount of support and faith in any scientific endeavour I have pursued, even when I didn't have much of the latter myself. She's always been there for me, both academically and personally. She has been a source of comfort and refuge, even in the most difficult of times. That is also not to mention the fact that she is probably the most fantastic teacher that walks the face of the planet. She would probably say I am being entirely hyperbolic, even though I wholeheartedly insist I am not. This is not a belief, it's a law of nature! Although, she does have a pathological habit of refusing to take any thanks I give her whatsoever. She is an amazing human being, even if I cannot express how grateful I am in words all the time. So, just to emphasise one last time: THANK YOU SO, SO, SO MUCH!

Sheila Porter I would like to thank Sheila Porter, the CEO of SciFest, for being so generous with her time over the last number of months in preparation for the virtual Regeneron International Science and Engineering Fair and providing all of the assistance I required when it was needed. It has been a honour to represent Ireland and SciFest at this competition, and I would like to thank her for giving me the oppurunity to do so.

Parents To my parents, Timmy and Frances to whom I owe everything, I would like to take the opportunity to express how grateful I am for all the support, help, guidance, feedback and love that they have provided me over these many years. I am and will be eternally grateful for this.

Laura O'Sullivan Laura is a self-taught machine learning developer - she's previously won prizes for exoplanet detection and cervical cancer screening research. During Patch, I collaborated with her to find the most appropriate neural network architecture, and to improve the overall performance of the existing architecture.

Tadhg Fitzgerald I have talked to a machine learning researcher, Tadhg Fitzgerald, who advised me on my machine learning implementation, as well as, providing guidance in relation to troubleshooting extensive memory issues which occurred throughout the duration of the project. His assistance was greatly appreciated.

Tom McCarthy and the extended Patch team I would like to thank the various members of the Patch Summer Accelerator Programme for their assistance in the continued development of the software. I would like to thank Tom McCarthy, founder of Patch, who supported me all through Patch with his world class guidance and counsel, Mario Kostelac, Senior Machine Learning Engineer at Intercom, who gave me outstanding advice on my machine learning implementation and troubleshooting; as well as, Andrea Vaccari, co-founder of Clearspace, who encouraged me to push myself to meet AMSIMP's full potential.

Met Éireann I have talked to researchers from Met Éireann, Colm Clany and Alan Hally, who advised me on the current applications of machine learning at meteorological agencies. I would like to thank them for their time commitment.

Irish Centre for High-End Computing I would like to thank the Irish Centre for High-End Computing and the people who worked therein for their continued assistance with this project, and providing computational resources which made this project possible. I am eternally grateful.

Scot Hosking and Rachael Furner I have talked to researchers from the University of Cambridge, Scot Hosking and Rachael Furner, who advised me on the current applications of machine learning at meteorological agencies, as well as, the application of machine learning in oceanography. I would like to thank them for their time commitment.

Nate Sauder I have had a conversation with Nate Sauder, who is an expert in machine learning and has worked in the meteorological industry in the past. I would like to thank them for their time commitment.

Mr. Healy, and Ms. Foley-Hayes I would like to thank Mr. Healy, and Ms. Foley-Hayes, our Principal and Vice Principal, for their support in relation to funding, incorporating various different ideas into my project, and generally providing all the assistance I needed.

Contents

List of Figures

Chapter 1

Introduction

> "Yes, it is easy to see that nearly six years of magical education has not been wasted on you ... Ghosts are transparent"
>
> Severus Snape

Numerical Weather Prediction focuses on taking current observations of weather and processing this data with computer models to forecast the future state of weather. Knowing the current state of weather is just as important as the numerical computer models processing the data. Current weather observations serve as input to the numerical computer models through a process known as data assimilation to produce outputs of temperature, precipitation, and hundreds of other meteorological elements from the oceans to the top of the atmosphere[1].

However, weather prediction is not all that it could be. For example, you should remember Hurricane Sandy, which hit New York in 2012. The American Global Forecast System predicted it would not even reach the mainland! This, of course, was wrong, and as a result, 285 people died and 68.7 billion dollars worth of damage was done. But, why is weather forecasting so hard? Forecasting today is done by physical modelling. This means we know the equations that govern weather systems, but we can't solve them exactly. Instead, we approximate them. Weather agencies take reams of raw data, feed it into supercomputers, and just number crunch and number crunch. As you can imagine, this requires a significant amount of computational resources. This is not only a cost issue, it also inhibits our ability to produce forecasts of a high temporal resolution; which can be significantly important in predicting extreme weather events, such as tornadoes.

1.1 Development of the Idea

I think it is extremely difficult to pinpoint exactly where the idea originated from; on reflection, however, the spark which really ignited the flame for this project came from a deep interest in the study of the atmosphere and by extension atmospheric

science. I initially became interested in this topic based on two distinct factors.

I became interested in the area of atmospheric science from watching a YouTuber named, Dr. Simon Clark. Dr. Simon Clark recently completed a PhD in theoretical atmospheric physics, researching dynamical stratosphere-troposphere coupling, and made a vlog series documenting his experiences. I was extremely intrigued by this series, with keen interest in his videos explaining his research. These videos captivated me and ultimately led me to reading his thesis (titled Quasi-geostrophic influence of the polar stratosphere on the troposphere)[2].

Secondly, my interest also came from my concern over human caused climate change. I was extremely fascinated by how a gas, such as carbon dioxide, could capture so much more heat than oxygen, even though there is a higher concentration of oxygen in the atmosphere. This ultimately led to the original idea for this project: creating a computational simulation of both Earth and Venus, and using it to determine if Earth could become just as hellish and uninhabitable as Venus if the rate at which we were pumping carbon dioxide into the atmosphere remained constant. There was a few problems with this idea, primarily being that it would require a ridiculous amount of computational resources, and there was no way of ensuring it was accurate. This led me to narrow the scope of the project, and ultimately, I eventually settled on the idea of improving numerical weather prediction in some fashion. From my initial research, as I previously mentioned, atmospheric simulations required a significant amount of computational power. There have been fantastic developments in machine learning over the past decade, and I wanted to apply these advancements to the problem of weather forecasting. Through various discussions with experts in a wide variety of fields, I came to the conclusion that such an endeavour may be fruitful.

1.2 Aims

The principal aim of this project is to create a neural network architecture, which will reduce the amount of computational resources required to generate weather forecasts compared to traditional physics-based models; and one which will provide a similar level of performance when compared to physical models. Access to the source code of any given physics-based forecast system is also notoriously difficult, with enormous price tags being the norm to gain access to time critical weather information. My vision is to make better weather tools available to everyone. Once the software is created, it will be published on the open source platform, known as, GitHub. This will allow programmers, and atmospheric physicists alike to inspect the source code, and to contribute to the development of the software. The software will also be published on Anaconda Cloud. I did this in order to allow, with just a simple command, the installation of the software. I will also develop a series of documentation that will accompany the software, which will accumulate in the development of a website.

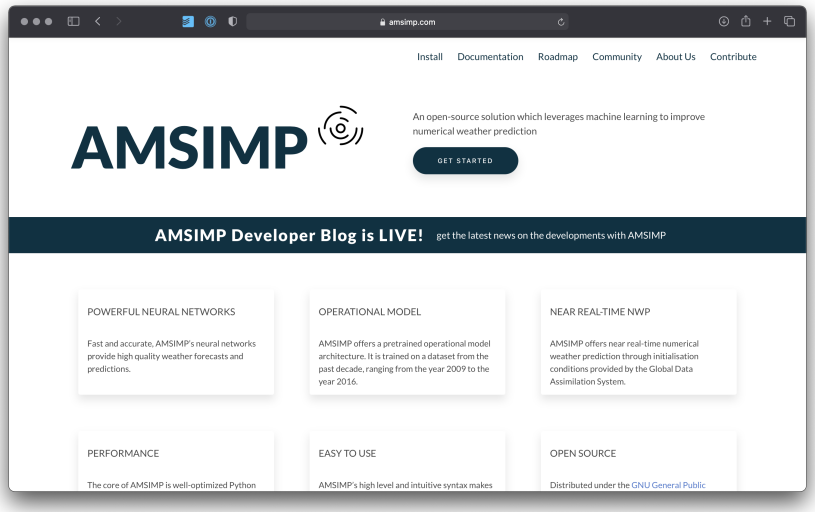

Figure 1.1: A screenshot of the website for the software.

1.3 Organisation of this Work

The work contained in this project book is broken down into a number of different chapters:

- Chapter 2 discusses the various neural network architecture under consideration and the current implementation within the software.
- Chapter 3 discusses some of the decisions made on the software development side of the project, such as, choosing an appropriate programming language for the task.
- Chapter 4 outlines the software benchmarking experiments to be carried out on the software.
- Chapter 5 contains the results of the benchmarking described in the previous chapter.
- Chapter 6 reviews the results and outcomes of the project, discusses the limitations of the software, and lays out a road-map for the continued development and enhancement of the software.

Chapter 2

Model Architecture

"Since Newton, mankind has come to realise that the law of physics are always expressed in the language of differential equations"

Steven Strogatz

2.1 RNN

2.1.1 Introduction

Weather forecasting has traditionally been done by physical models of the atmosphere, which are unstable to perturbations, and thus are inaccurate for large periods of time[3]. Since machine learning techniques are more sensitive to perturbations, it would be logical to combine a neural network with a physical model. Weather forecasting is a sequential data problem, therefore, a recurrent neural network is the most suitable option for this task.

Definition 1 *A recurrent neural network is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence.*

Before, we delve into the specific example of using a recurrent neural network to predict the future state of the atmosphere, it is necessary to review what a recurrent neural network is. Recurrent Neural Networks (RNNs) are neural networks that are used in situations where data is presented in a sequence. For example, let's say you want to predict the future position of a fast-moving ball. Without information on the previous position of the ball, it is only possible to make an inaccurate guess. If you had, however, a large number of snapshots of the previous position, you are then able to predict the future position of the ball with some certainty. RNNs excel at modelling sequential data such as this. This is due to sequential memory.

In order to intuitively understand sequential memory, the prime example would be the alphabet. While it is easy to say the alphabet from A-Z, it is much harder to

go from Z-A. There is a logical reason why this is difficult. As a child, you learn the alphabet in a sequence. Sequential memory is a mechanism that makes it easier for your brain to recognise sequence patterns.

In a traditional neural network, there is an input layer, hidden layer, and an output layer. In a recurrent neural network, a loop is added to pass information forward as seen in the diagram below (provided by Towards Data Science)[4]:

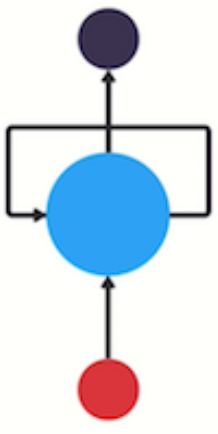

Figure 2.1: Visualisation of a Recurrent Neural Network

The information that is forwarded is the hidden layer, which is a representation of previous inputs. How this works in practise is that you initialise your network layers and the initial hidden state. The shape and dimension of the hidden state will be dependent on the shape and dimension of your recurrent neural network. Then you loop through your inputs, pass the relevant parameter and hidden state into the RNN. The RNN returns the output and a modified hidden state. Last you pass the output of the hidden state to the output layer of the model, and it returns a prediction.

There is, however, a major problem known as short-term memory. Short-term memory is caused by something known as the vanishing gradient problem, which is also prevalent in other neural network architectures. As the RNN processes more steps, it has troubles retaining information from previous steps. Short-Term memory and the vanishing gradient is due to the nature of back-propagation. This can be comprehended through understanding how a neural network is trained[4].

Definition 2 *Back-propagation is an algorithm used to train and optimise neural networks.*

To train a recurrent neural network, you use an application of back-propagation called back-propagation through time. Training a neural network has three major steps. First, the relevant data vector is normalised between 0 and 1, the vector is fed into the RNN, and it goes through an activation function. The activation function utilised in the software is the rectified linear activation function[5].

Definition 3 *The rectified linear activation function is a piece-wise linear function that will output the input directly if is positive, otherwise, it will output zero.*

The function is linear for values greater than zero, meaning it has a lot of the desirable properties of a linear activation function when training a neural network using back-propagation. Yet, it is a nonlinear function as negative values are always output as zero. As a result, the rectified function is linear for half of the input domain and nonlinear for the other half, it is referred to as a piece-wise linear function[6]. This nonlinear element is extremely important if the system has a nonlinear component, for example in predicting the evolution of the future state of the atmosphere.

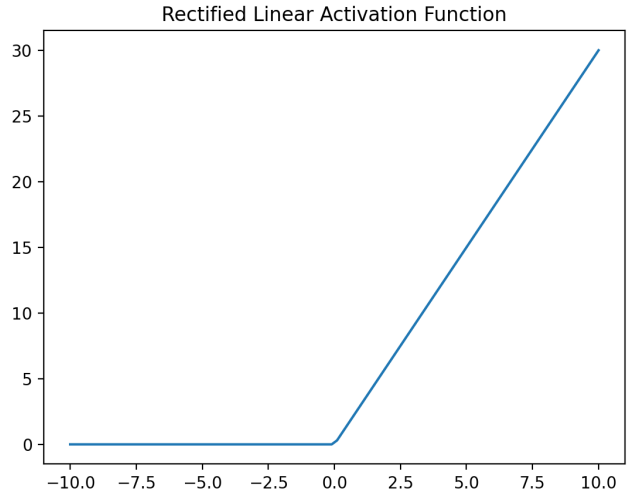

Figure 2.2: Sketch of the Rectified Linear Activation Function

Second, it outputs the results. Third, it compares the prediction to the observed state using a loss function.

Definition 4 *A loss function outputs an error value which is an estimate of how poorly the network is performing.*

The lost function that will be utilised in the software will be the function for mean squared error. The reason for choosing this particular function is that it heavily penalises large errors, as it squares the difference between the predicted and actual value. A large error in a weather forecast is highly undesirable, hence, the use of this function. The function is represented below:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{2.1}$$

If a vector of n predictions is generated from a sample of n data points on all variables, and Y is the vector of observed values of the variable being predicted, with $\hat{Y}_i$ being the predicted values.

Definition 5 *Mean squared error is the average squared difference between the estimated values and the actual value.*

Returning to the training of the RNN, it uses that error value from the loss function to do back propagation which calculates the gradients for each time step in the network. The gradient is the value used to adjust the networks internal weights, allowing the network to learn. The bigger the gradient, the bigger the adjustments and vice versa. Here is where the problem lies. When doing back propagation, the gradient of the current time step is calculated with respect to the effects of the gradients, in the time step before it. So if the adjustments to the time step before it is small, then adjustments to the current time step will be even smaller. The gradient values will exponentially shrink as it propagates through each time step. That causes gradients to exponentially shrink as it back propagates down. The earlier layers fail to do any learning as the internal weights are barely being adjusted due to extremely small gradients.

Because of vanishing gradients, the RNN doesn't learn the long-range dependencies across time steps. So not being able to learn on earlier time steps causes the network to have a short-term memory. In order to combat this, a long short-term memory is used[4].

2.1.2 LSTM

LSTM's were created as a solution to the short-term memory problem. They have internal mechanisms called gates that can regulate the flow of information. These gates can learn which data in a sequence is important to keep or throw away. By doing that, it can pass relevant information down the long chain of sequences to make predictions. For example, if you were interested in buying a particular product, you might read a review in order to determine if the purchase of the product is a good decision. When you read a review, your brain subconsciously only remembers important keywords. You pick up words like "amazing", "superb", or "awful", you don't remember words such as "the", "as", or "because". This is what an LSTM does, it learns to keep only the relevant information to make predictions.

An LSTM has a similar control flow as a recurrent neural network. It processes data passing on information as it propagates forward. The differences are the operations within the LSTM's cells. The core concept of LSTM's are the cell state, and it's various gates. The cell state is the method by which information is transferred down the sequence chain. The cell state, in theory, can carry relevant information throughout the processing of the sequence. So even information from the earlier time steps can make its way to later time steps, reducing the effects of short-term memory. As the cell state goes on its journey, information gets added or removed to the cell state via gates[5].

Definition 6 *A gate is an electric circuit with an output which depends on the combination of several inputs.*

Gates contain the sigmoid activation function. The sigmoid activation function squishes values between 0 and 1. That is helpful to update or forget data because any number getting multiplied by 0 is 0, causing values to disappears or be "forgotten".

Any number multiplied by 1 is the same value therefore that value stays the same or is "kept".

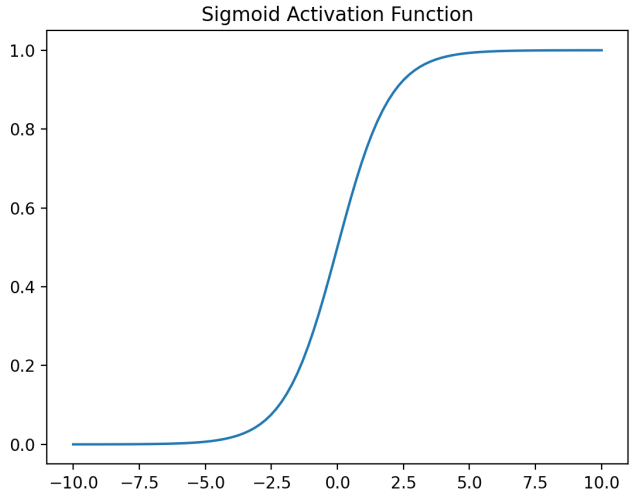

Figure 2.3: Sketch of the Sigmoid Activation Function

There are three types of gates utilised within a neural network: a forget gate, an input gate, and an output gate. A forget gate decides what information should be thrown away or kept. Information from the previous hidden state and information from the current input is passed through the sigmoid function. An input gate is where the previous hidden state and current input are passed into a sigmoid function. The output gate decides what the next hidden state should be. The hidden state is also used for predictions. First, we pass the previous hidden state and the current input into a sigmoid function. Then we pass the newly modified cell state to the rectified linear activation function. We multiply the rectified linear activation function output with the sigmoid output to decide what information the hidden state should carry. The output is the hidden state. The new cell state and the new hidden state is then carried over to the next time step[5].

2.1.3 Convolutional LSTM Network

The formulation of a numerical weather prediction model is a spatiotemporal sequence forecasting problem that can be solved under a general sequence-to-sequence learning framework.

Definition 7 *A spatiotemporal sequence is a sequence that contains both spatial and temporal information.*

In order to better model the spatiotemporal relationships, this project will utilise ConvLSTM layers; which was proposed by Xingjian Shi et. al[7]. This spatiotemporal sequence forecasting problem is different from the one-step time series forecasting

problem because the prediction target of the problem is a sequence which contains both spatial and temporal structures. Although a LSTM layer has proven powerful for handling temporal correlation, it contains too much redundancy for spatial data. To address this problem, this project will use a ConvLSTM layer which has convolutional structures in both the input-to-state and state-to-state transitions[7].

The major drawback of LSTMs is in the handling spatiotemporal data due to its usage of full connections in input-to-state and state-to-state transitions in which no spatial information is encoded. To overcome this problem, a distinguishing feature of a ConvLSTM cell is that all the inputs and gates of the ConvLSTM layer are 3D tensors whose last two dimensions are spatial dimensions. To get a better picture of the inputs and states, we may imagine them as vectors standing on a spatial grid. The ConvLSTM determines the future state of a certain cell in the grid by the inputs and past states of its local neighbour. This can easily be achieved by using a convolution operator in the state-to-state and input-to-state transitions[7]. This is what the ConvLSTM layer does.

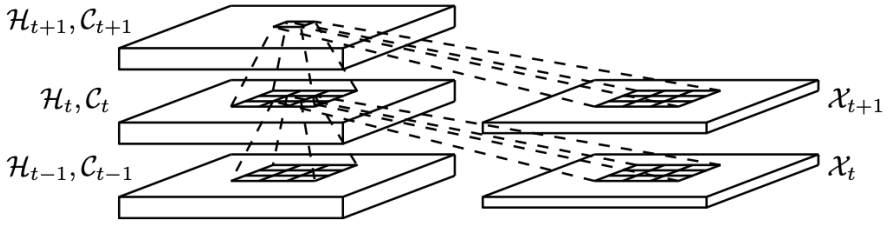

Figure 2.4: The architecture of a ConvLSTM cell.

2.2 Dataset

2.2.1 ERA5 Atmospheric Reanalysis Dataset

ERA5 provides hourly estimates of a large number of atmospheric, land and oceanic climate variables. The data covers the Earth on a 30km grid and resolves the atmosphere using 137 levels from the surface up to a height of 80km. ERA5 includes information about uncertainties for all variables at reduced spatial and temporal resolutions. Quality-assured monthly updates of ERA5 are published within 3 months of real time. Preliminary daily updates of the dataset are available to users within 5 days of real time. ERA5 combines vast amounts of historical observations into global estimates using advanced modelling and data assimilation systems[8].

The ERA5 reanalysis dataset was used for training, validating and testing the performance of the neural network architecture. Reanalysis datasets provide the best guess of the atmospheric state at any point in time by combining a forecast model with the available observations. The raw data is available hourly for 40 years from 1979 to 2019 on a 0.25° latitude-longitude grid (721×1440 grid points) with 37 vertical levels. Since this raw dataset is quite significant, it is necessary to regrid the

dataset to a lower resolution and use a smaller fraction of the available dataset[9]. The poles were excluded from the dataset in order to avoid a singularity, and the potential negative impact that could have on predictive ability of the neural network.

It was ultimately decided to use a spatial resolution of 1° (179 × 360 grid points) and a temporal resolution of 2 hours. The data is split into yearly NetCDF files for each variable. The entire dataset at 0.25° resolution has a size of 400GB, before the dataset was interpolated to a grid of a lower resolution. The prognostic variables of interest are temperature and geopotential, and were chosen based on meteorological considerations. Geopotential, and temperature are prognostic state variables in most physical numerical weather prediction and climate models[9].

2.2.2 Integrated Forecasting System

The Integrated Forecast System is a global numerical weather prediction system developed and maintained by the European Centre for Medium-Range Weather Forecasts organisation. The version of the IFS run at ECMWF is often referred to as the "ECMWF" or the "European model" in North America, to distinguish it from the American GFS. It comprises of a spectral atmospheric model with a terrain-following vertical coordinate system coupled to a 4D variational data assimilation system. In 1997 the IFS became the first operational forecasting system to use a 4D variational, data assimilation system

Definition 8 *4D dimensional variational data assimilation system adjusts a short-range forecast, called the background, in space and time to bring it into closer agreement with meteorological observations.*

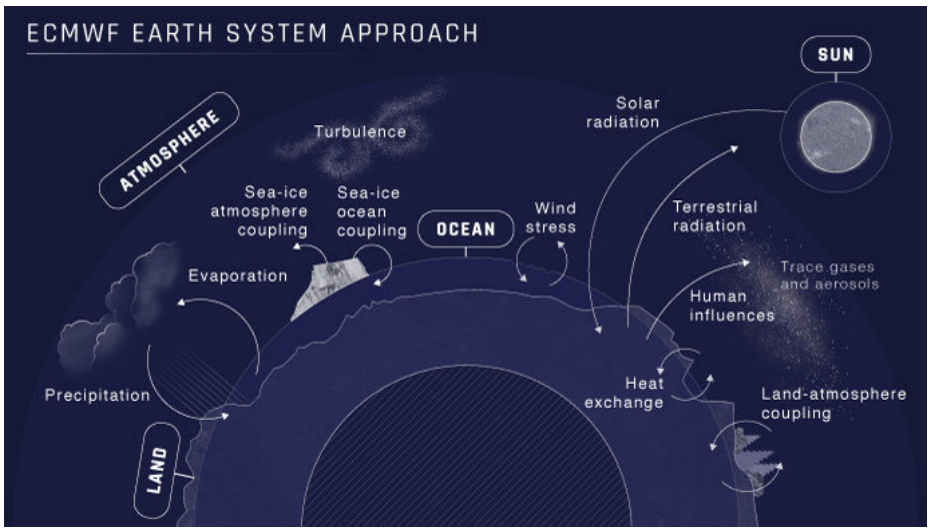

Figure 2.5: ECMWF Integrated Forecast System

It is one of the predominant global medium-range models in general use worldwide; its most prominent rival in the 6–10 day medium range include the American Global

Forecast System, the Canadian Global Environmental Multiscale Model and the UK Met Office Unified Model. For context, Met Éireann utilises the IFS for medium-range forecasts; however for short-range forecasts, Met Éireann uses its own HARMONIE-AROME NWP model. The operational configuration consists of 1000×900 grid points in the horizontal at 2.5km resolution, with 65 levels in the vertical. A 54-hour forecast is produced four times a day, at 00Z, 06Z, 12Z and 18Z[10].

The IFS is the gold standard of medium-range numerical weather prediction. The current IFS deterministic forecast is computed on a cluster with 11,664 cores. One 10 day forecast at 10 km resolution takes around 1 hour of real time to compute[9]. The Integrated Forecasting System will be used as a comparison against the neural network.

To provide physical baselines more in line with the current resolution of the neural network, it was compared against the IFS model at two coarser horizontal resolutions, T42 (approximately 2.8°) with 62 vertical levels and T63 (approximately 1.9°) with 137 vertical levels. It must be noted that I personally did not generate such forecasts, or perform the analysis on said forecasts. It acquired the results from Weather Bench, a benchmark dataset for data-driven weather forecasting. According to said source; computationally, a single forecast takes 270 seconds for the T42 model and 503 seconds for the T64 model on a single XC40 node with 36 cores.

2.3 Implementation

Drawing from knowledge of current numerical weather prediction model frameworks, it may seem intuitive to train a machine learning model to produce the best possible single-step forecast from a given atmospheric state. In practice, this can yield a model that performs well for short-range forecasts but diverges from reality for longer range predictions. This is because there are no constraints on the CNN, physical or mathematical, that would prevent it from diverging from reality when its prediction fed back in as inputs no longer resemble an atmospheric state in the training data. In order to nudge the numerical weather prediction model toward learning to predict longer-term weather and improve its long-term stability, the model will be trained on multiple iterated predictive steps.

Initially, the model inputs included two time steps and it is tasked with predicting two output time steps. This resulted in the first couple of time step predictions producing relatively decent result, however, it quickly diverged from reality after this point. Hence, it was necessary to increase both the time steps and the output time steps to six. This increased the overall numerical stability of the model.

The dataset consists of three features: air temperature at a pressure surface of 850 hPa, geopotential at a pressure surface of 500 hPa, and air temperature at 2 metres above the surface. For a single day, there is twelve observations. The goal for this project will be to predict the relevant atmospheric parameter in 12 hours time given the last twelve hours of data. In order to make such predictions, it is necessary to create a window of the last 6 ($\frac{12}{2}$) observations to train the model[11]. The neural network was trained on observational data from 1980 to 2008. The remainder of

the dataset, 2009 to 2020, was preserved for validation, testing and benchmarking the neural network against physics-based models. The model was built using the open-source Keras library for Python with Google's TensorFlow backend[12].

```
# Optimiser.
opt = Adam(lr=1e-3, decay=1e-5)
```

Adam optimisation was chosen as the most appropriate optimiser for this particular model. This is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments. This method is computationally efficient, has little memory requirements, invariant to diagonal rescaling of gradients, and is well suited for problems that are large in terms of parameters.

```
# First layer of model.
model.add(
    ConvLSTM2D(
        filters=64,
        kernel_size=(7, 7),
        input_shape=(6, 179, 360, 3),
        padding='same',
        return_sequences=True,
        activation='tanh',
        recurrent_activation='hard_sigmoid',
        kernel_initializer='glorot_uniform',
        unit_forget_bias=True,
        dropout=0.3,
        recurrent_dropout=0.3,
        go_backwards=True
    )
)
```

The above code is the first layer of the model. The activation functions utilised in this layer are predefined within Tensorflow, and have been described at great length previously. The model consists of four ConvLSTM layers in total, with similar parameters as the aforementioned layer with a few variations.

```
# Batch normalisation.
model.add(BatchNormalization())
```

A consistent challenge in machine learning is that the model is updated layer-by-layer backward from the output to the input using an estimate of error that assumes the weights in the layers prior to the current layer are fixed. Because all layers are changed during an update, the update procedure is forever chasing a moving target. A batch normalisation layer occurs after each ConvLSTM layer to resolve this problem.

Definition 9 *Batch normalization is a technique to help coordinate the update of multiple layers in the model.*

It does this by scaling the output of the layer, specifically by standardising the activation of each input variable per mini-batch, such as the activation of a node from the previous layer. Standardising the activation of the prior layer means that assumptions the subsequent layer makes about the spread and distribution of inputs during the weight update will not change. This has the effect of stabilising and speeding-up the training process of neural networks[13].

```
# Dropout.
model.add(Dropout(0.1))
```

Each batch normalisation layer is then followed by a dropout. Dropout is a regularisation method that approximates training a large number of neural networks with different architectures in parallel. During training, some number of layer outputs are randomly ignored or "dropped out." This has the effect of making the layer look-like and be treated-like a layer with a different number of nodes and connectivity to the prior layer. In effect, each update to a layer during training is performed with a different "view" of the configured layer. Dropout has the effect of making the training process noisy, forcing nodes within a layer to probabilistically take on more or less responsibility for the inputs. This conceptualisation suggests that perhaps dropout breaks-up situations where network layers co-adapt to correct mistakes from prior layers, in turn making the model more robust[14].

```
# Add dense layer.
model.add(Dense(3))
```

Following the final ConvLSTM layer, a dense layer is implemented. The dense layer is a neural network layer that is connected deeply, which means each neuron in the dense layer receives input from all neurons of its previous layer[15]. In our case, it results in the model outputting the expected shape by reducing the filters down to the three previously mentioned features. The code for the model in its entirety can be found in appendix A.1.

Chapter 3

Implementation Details

> "Curious that we spend more time congratulating people who have succeeded than encouraging people who have not."
>
> Neil deGrasse Tyson

3.1 Open Source Software

Definition 10 *Open Source Software is software with source code that anyone can inspect, modify, and enhance.*

Source code is the code that computer programmers use to modify, and change how a piece of software functions[16]. Programmers with access to the source code can improve that program by adding features to it, fixing bugs, or by improving the documentation of the surrounding source code.

According to Tom Macaulay[17], the open source development of software has a number of advantages over the traditional development of proprietary software, including but not limited to:

- Lower costs.
- Extensive customisation.
- Higher quality software.
- Greater security.
- Regular updates.
- Quick fixes.

If the source code of the software was available to the open source community, it could lead to a low cost, and high quality numerical weather prediction scheme being produced. If such an event occurs, it could, theoretically, vastly enhance existing weather predicting software, and could lead to a further dramatic decline in deaths from weather related phenomena. This project is an attempt to kick start such a future.

Everything related to this project, from the source code of the simulator to this very paper, can be accessed at the organisation known as 'AMSIMP' on the open source platform known as GitHub. This can be accessed at `https://github.com/amsimp`.

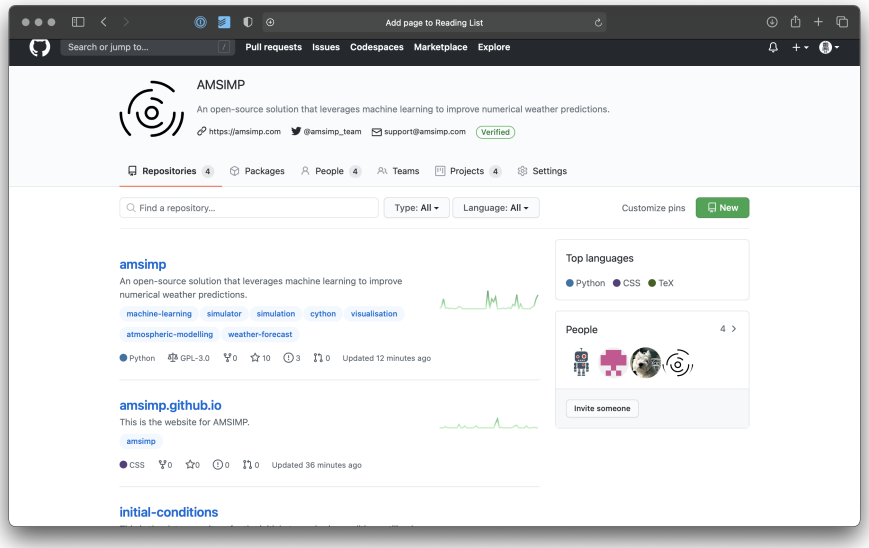

Figure 3.1: A screenshot of the AMSIMP organisation, hosted by the open source platform known as GitHub.

3.2 Language Selection

3.2.1 Initial Language Selection

The most crucial element of this project was choosing an appropriate programming language for the task. During the initial consideration period, I analysed two different programming languages: JavaScript, and Python[18].

Initially, I considered the programming language, JavaScript with a run-time environment known as Node.js (Node.js executes code outside of a browser), for two reasons primarily:

- JavaScript excels at data visualisation.
- Node.js is extremely well suited for memory intensive activities.

In line with the expectation of using this programming language, Miss Abbott, and I attended a Dublin Node.js Meetup. I did this in order to gain an understanding of what Node.js is used for, and what would be the best way one would go about using it.

In the end, however, I ultimately chose Python for the task due to a number of shortcomings on the part of JavaScript (Node.js), and advantages on the part of Python. Firstly, JavaScript just doesn't have the same enormous suite of scientific packages and inbuilt functionality that Python does. Using JavaScript, therefore, would waste valuable development time, and ultimately would be entirely inefficient.

Secondly, Python already has an extensive ecosystem with how-to's available for almost any scientific task you would ever want to do. For JavaScript, this is simply not the case.[19]

Chapter 4

Benchmarking

> "It's known in theory that log(log(n)) approaches infinity, but no one has ever observed it in practice."
>
> Grant Sanderson

To prove the hypothesis that 'it is possible to train a neural network on an atmospheric reanalysis dataset from the past 40 years, that the neural network captures weather patterns and can predict the evolution of the atmosphere, and ultimately that a machine learning model will improve weather prediction compared to physics-based models', it is necessary to carry out a series of appropriate benchmarks.

It was determined that it was necessary to carry out a series of appropriate benchmarking within the area of performance. These benchmarks would highlight whether or not the forecasts produced by the neural network capture crucial weather patterns, and can predict the future evolution of the atmosphere. As mentioned in section 2.2.2, the ECMWF's Integreated Forecasting System weather model will act as a comparison in order to understand the performance of the software. The time required to generate a forecast will also be measured. This should give an idea as to the potential advantages of using neural networks with respect to reducing the amount of computational resources required to generate a forecast. This will be compared against the IFS T42 model as it has a similar resolution to the neural network developed for this project. As a reminder, the IFS T42 model is a lower resolution physical model (approximately 2.8°). Alongside this, the software will also be compared against a persistence forecast in which the fields at initialisation time are used as forecasts and a climatological forecast. This means that to be useful, a forecast system needs to beat the climatology and the persistence forecast.

This benchmarking process was inspired by the open-source project, known as WeatherBench. WeatherBench is a benchmark dataset for data-driven medium-range weather forecasting. It provides data derived from the ERA5 archive that has been processed to facilitate the use in machine learning models. It also proposes simple and clear evaluation metrics which will enable a direct comparison between different

forecasting models[9].

Please note that all of the benchmarking code is available on the AMSIMP repository, which can be accessed using the following url: `https://github.com/amsimp/amsimp/tree/dev/benchmarking`

4.1 Evaluation

The benchmarking was carried out for the year of 2019 and 2020. To make sure no overlap exists between the training, validation and test dataset, the first test date is 1 January 2019 00UTC plus forecast time, for a seven day forecast the first test date would be 7 January 2019 00UTC, while the last training target is 31 December 2020 23UTC.

For benchmarking purposes, the 500 hPa geopotential and the 850 hPa temperature were chosen as the primary verification fields. Geopotential at 500 hPa pressure, often abbreviated as Z500, is a commonly used variable that encodes the synoptic-scale pressure distribution. It is the standard verification variable for most medium-range numerical weather prediction models. The 850 hPa temperature field was chosen as the secondary verification field because temperature is a more impact-related variable. The 850 hPa pressure surface is usually above the planetary boundary layer and therefore not affected by diurnal variations but provides information about broader temperature trends, including cold spells and heat waves.

The root mean squared error was chosen as the primary metric because it is easy to compute and mirrors the loss used for most machine learning applications. The root mean squared error was defined as the mean latitude-weighted root mean squared error:

$$RMSE = \sqrt{\frac{1}{N_{lat}N_{lon}} \sum_{j}^{N_{lat}} \sum_{k}^{N_{lon}} L(j)(f_{i,j,k} - t_{i,j,k})^2} \tag{4.1}$$

where f is the forecasted value and t is the observational value from the ERA5 Atmospheric Reanalysis dataset. In addition, I also evaluated the baselines using the latitude weighted anomaly correlation coefficient and the latitude weighted mean absolute error. The Anomaly Correlation Coefficient (ACC) is one of the most widely used measures in the verification of spatial fields. It is the spatial correlation between the forecast anomaly and the verifying analysis anomaly where the anomalies are computed with respect to a model climate. This is defined by the following equation, where the prime ($\prime$) is defined as climatology:

$$ACC = \frac{\sum_{i,j,k} L(j) f'_{i,j,k} t'_{i,j,k}}{\sqrt{\sum_{i,j,k} L(j) f'^2_{i,j,k} \sum_{i,j,k} L(j) t'^2_{i,j,k}}} \tag{4.2}$$

Chapter 5

Results

"Of course it is happening inside your head, Harry, but why on earth should that mean it is not real?"

Albus Dumbledore

5.1 Root Mean Squared Error

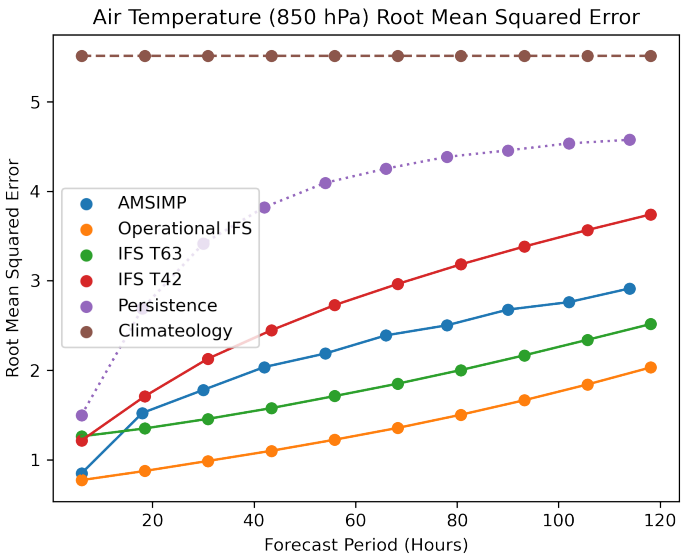

Figure 5.1: Air Temperature at 850 hPa

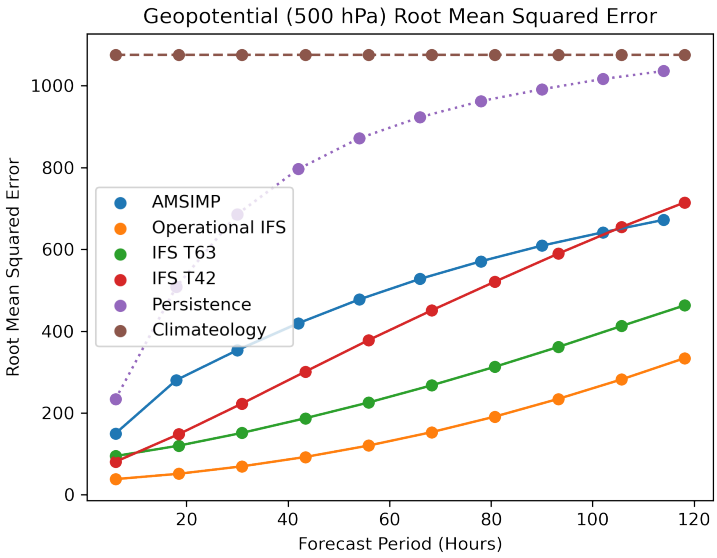

Figure 5.2: Geopotential at 500 hPa

5.2 Mean Absolute Scaled Error

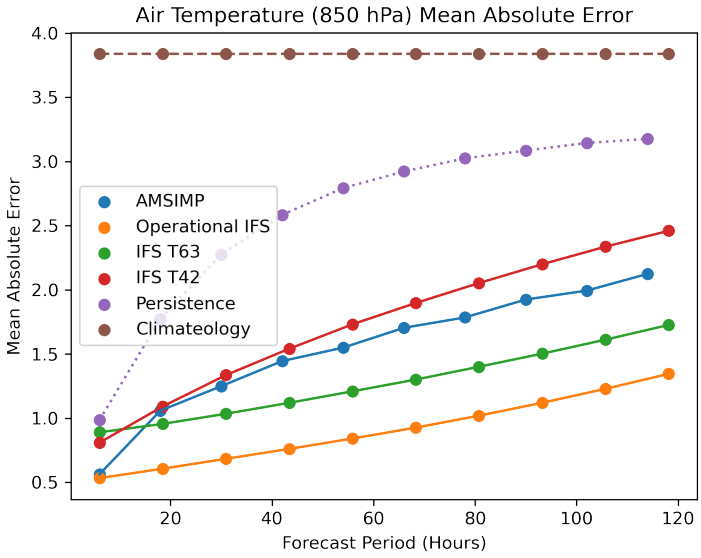

Figure 5.3: Air Temperature at 850 hPa

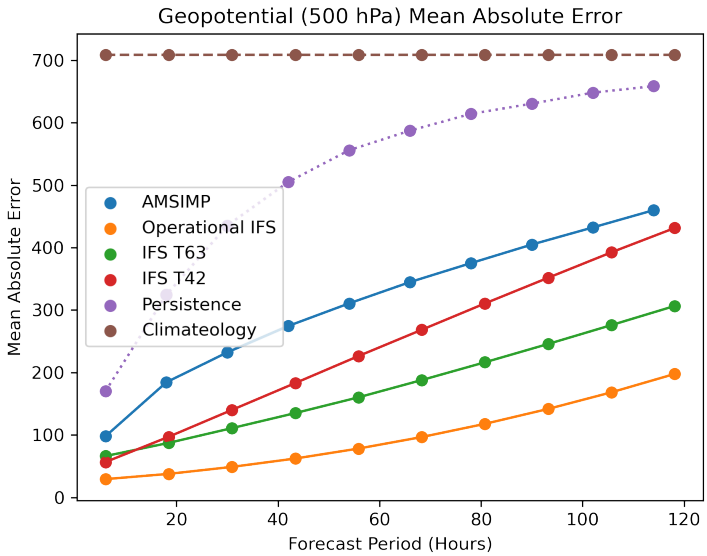

Figure 5.4: Geopotential at 500 hPa

5.3 Anomaly Correlation Coefficient

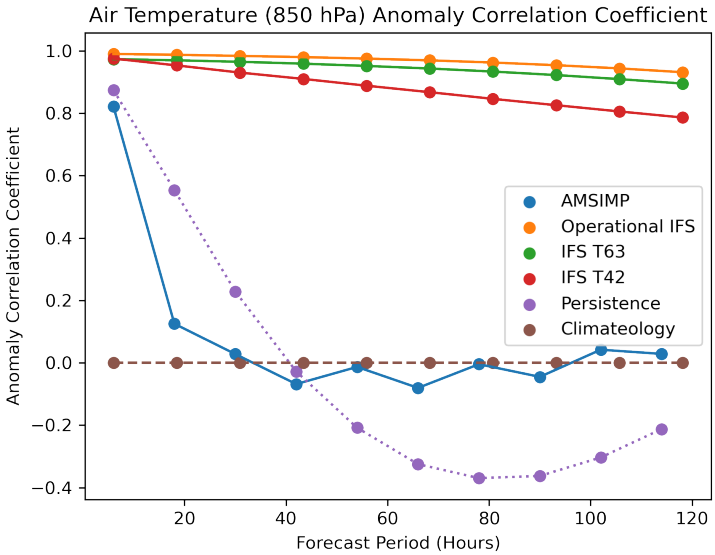

Figure 5.5: Air Temperature at 850 hPa

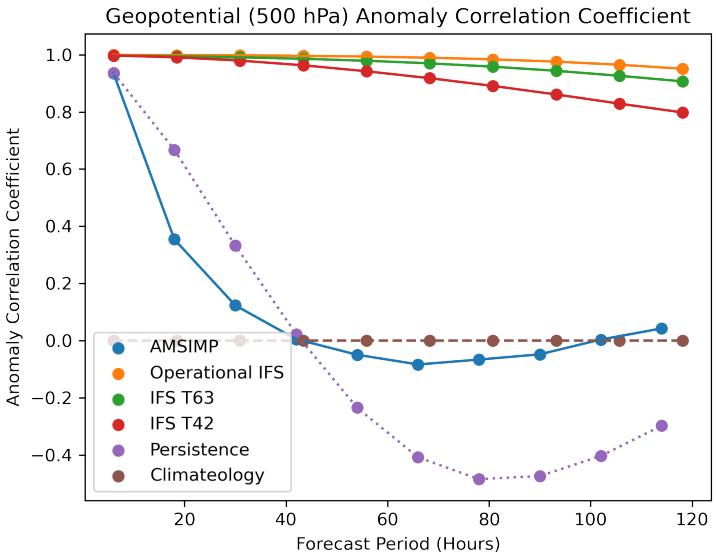

Figure 5.6: Geopotential at 500 hPa

Chapter 6

Conclusions

> "It is our choices, Harry, that show what we truly are, far more than our abilities."
>
> Albus Dumbledore

The previous chapters have discussed the development of an open source implementation to improve numerical weather prediction through the utilisation of a neural network architecture. Although extremely time consuming, the importance of an open source implementation cannot be understated. It could potentially open up a field to a wider group, and often doesn't receive the media attention it rightfully deserves.

Future applications of this work are extremely wide ranging, from usage in numerical weather prediction schemes that take in observational data collected from satellites, and ground stations in order to provide a picture of future weather events; to forming a starting point in future research of atmospheric phenomena. While, at the moment, it definitely would be inaccurate to say the software is ready for such usage, with future enhancements which I will touch on in a minute, it most certainly will.

6.1 Looking Back

To prove the hypothesis that 'it is possible to train a neural network on an atmospheric reanalysis dataset from the past 40 years, that the neural network captures weather patterns and can predict the evolution of the atmosphere, and ultimately that a machine learning model will improve weather prediction compared to physics-based models', it was necessary to carry out a series of appropriate benchmarks.

6.1.1 Analysis of Results

This report hypothesises that it is possible to train a neural network on an atmospheric reanalysis dataset based on data from the 10 years, that such a neural network

captures crucial weather patterns, and can predict the future evolution of the atmosphere, and that such a machine learning model will ultimately improve numerical weather prediction in comparison to established physics-based models.

One of the key factors which led to the development of a machine learning model was the expected decrease in computational resources required to generate a forecast. While the initial training of the model was computationally burdensome, particularly with respect to memory, the assumption made at the start of this project holds once the model is trained. Once the model has trained, a performance increase of 100 times can be expected in comparison against a physics-based model of a similar resolution, the ECMWF IFS T63.

With respect to the benchmarking outlined in chapter 4, the forecast system needs to beat the climatology forecast and the persistence forecast to be classified as useful. The benchmarks have demonstrated that the model performs roughly equivalent to physical models of a similar, or lower spatial resolution, however, it fails to beat well established, operational physics-based models at this time. The model is significantly better at creating air temperature predictions and appears to suffer with geopotential predictions at times. The root mean squared error and mean squared error demonstrate that the model's air temperature is useful throughout the 120-hour forecast period, with the model consistently beating the ECMWF IFS T42 model and it outperforming the ECMWF IFS T63 model in the short term. The picture for geopotential is less rosy, with the root mean squared error and mean squared error demonstrating that the model's geopotential predictions are useful throughout the 120-hour forecast period, with the model consistently beating the ECMWF IFS T42 model, however, it does not beat the ECMWF IFS T63 model at any point in these predictions. Concerning spatial awareness as measured by the anomaly correlation coefficient, it can be generally regarded as terrible, it appears that the spatial aspect of a weather forecast was not captured by the model. This may be due to the loss function chosen for the learning phase.

Hence, the hypothesis that was proposed has been proven and can be accepted, as such.

6.1.2 Sources of Error

Hence, the hypothesis that was proposed has partially been proven, however, there are a few areas which could have hindered the performance of the software or led to a possible source of error:

- The anomaly correlation coefficient values are particularly poor, therefore, the loss function of RMSE may not have been the correct choice. NNs are incentive to make the loss function as low as possible, potentially to the detriment of not learning the actually dynamics of the atmosphere.

- As mentioned in section 2.2.1, it was decided to use a spatial resolution of 1° (179×360 grid points) and a temporal resolution of 2 hours. A lower resolution was chosen in order to reduce the amount of computational resources required

to train the model. Through high spatial resolution, however, a forecast can show the effects of local air currents, topography and soil cover. The forecasts produced thereby show local weather differences in more precise way[20]. High resolution produces high precision, hence, while choosing a lower resolution may have lowered the computational burden during training, it may have had a significant on the performance of the model.

6.2 Looking Ahead

The software is currently in an alpha release state. An alpha version of any software is a very early version of the software that may not contain all of the features that are planned for the final version[21]. In this section, I will briefly outline the enhancements and features that will be released in the beta version of the software, which is planned for release in Spring 2021:

- One of the most natural coordinate systems to use on Earth is a latitude-longitude grid. This was the coordinate system of choice for this project due to its simplicity, but, this system has singularities at the North Pole and South Pole that makes it difficult to use translationally-invariant convolution operations on this grid. To combat this particular problem in this project, the poles were excluded from the dataset, however, a more elegant solution to preserve spatial locality is to approximate data on the globe using the cubed sphere. This projection has been shown to give more uniformly sized grid cells than the alternative projections and to also produce better solutions to finite-difference and discontinuous Galerkin approximations to partial differential equations on the sphere. The cubed sphere is used for state-of-the-art NWP such as in the FV3 dynamical core of the National Oceanic and Atmospheric Administration's Global Forecast System model[22]. This is an avenue that will be explored in the coming months.

- As mentioned previously, a high resolution weather forecast produces high precision. As a result, in order to improve the performance of the model, the model will be trained on a higher resolution dataset. At this point, a resolution has not been decided, however, the decision will be made based on the computational resources available to initially train the model and the expected increase in performance that could be made by switching to a higher resolution.

- The three parameters on which the machine learning models were trained upon were: air temperature at 850 hPa, geopotential at 500 hPa, and air temperature at 2 metres above the surface. While these parameters are extremely important to predict from a meteorological point of view, the general public require predictions for the amount of precipitation to be made several days in advance; in order to make personal, and business decisions. This may be supplying shops with more food during periods of snowfall, or county councils setting up flood defences in town. In the coming months, the model will be trained on such parameters in order to provide the most useful weather forecast possible.

Appendices

A.1 Code for the AMSIMP Global Forecast Model

```
def model(epochs, bs):
    # Number of elements.
    n = len(os.listdir("processed_dataset/"))
    lst_n = np.linspace(1, n, n)

    # Define training dataset and validation dataset.
    # Training.
    train_ns = lst_n[:int(0.7 * n)]
    train_generator = DataGenerator(
        train_ns,
        batch_size=bs,
    )
    print("Training dataset created.")

    # Validation.
    val_ns = lst_n[int(0.7 * n):int(0.9 * n)]
    val_generator = DataGenerator(
        val_ns,
        batch_size=bs,
        shuffle=False
    )
    print("Validation dataset created.")

    with mirrored_strategy.scope():
        # Create, and train models.
        # Optimiser.
        opt = Adam(lr=1e-3, decay=1e-5)
        # Create model.
        model = Sequential()

        # First layer.
        model.add(
            ConvLSTM2D(
                filters=64,
                kernel_size=(7, 7),
                input_shape=(6, 179, 360, 3),
                padding='same',
                return_sequences=True,
                activation='tanh',
                recurrent_activation='hard_sigmoid',
                kernel_initializer='glorot_uniform',
                unit_forget_bias=True,
```

```
            dropout=0.3,
            recurrent_dropout=0.3,
            go_backwards=True
        )
    )
    # Batch normalisation.
    model.add(BatchNormalization())
    # Dropout.
    model.add(Dropout(0.1))

    # Second layer.
    model.add(
        ConvLSTM2D(
            filters=32,
            kernel_size=(7, 7),
            padding='same',
            return_sequences=True,
            activation='tanh',
            recurrent_activation='hard_sigmoid',
            kernel_initializer='glorot_uniform',
            unit_forget_bias=True,
            dropout=0.4,
            recurrent_dropout=0.3,
            go_backwards=True
        )
    )
    # Batch normalisation.
    model.add(BatchNormalization())

    # Third layer.
    model.add(
        ConvLSTM2D(
            filters=32,
            kernel_size=(7, 7),
            padding='same',
            return_sequences=True,
            activation='tanh',
            recurrent_activation='hard_sigmoid',
            kernel_initializer='glorot_uniform',
            unit_forget_bias=True,
            dropout=0.4,
            recurrent_dropout=0.3,
            go_backwards=True
        )
    )
```

```
88          # Batch normalisation.
89          model.add(BatchNormalization())
90          # Dropout.
91          model.add(Dropout(0.1))
92
93          # Final layer.
94          model.add(
95              ConvLSTM2D(
96                  filters=32,
97                  kernel_size=(7, 7),
98                  padding='same',
99                  return_sequences=True,
100                 activation='tanh',
101                 recurrent_activation='hard_sigmoid',
102                 kernel_initializer='glorot_uniform',
103                 unit_forget_bias=True,
104                 dropout=0.5,
105                 recurrent_dropout=0.3,
106                 go_backwards=True
107             )
108         )
109         # Batch normalisation.
110         model.add(BatchNormalization())
111
112         # Add dense layer.
113         model.add(Dense(3))
114
115     # Compile model.
116     model.compile(
117         optimizer=opt,
118         loss='mse'
119     )
120     # Summary of model.
121     model.summary()
122
123     # Train.
124     model.fit(
125         train_generator,
126         validation_data=val_generator,
127         epochs=epochs,
128         callbacks=[
129             tf.keras.callbacks.EarlyStopping(
130                 monitor="val_loss", min_delta=0, patience=2, mode="auto"
131             )
132         ],
```

```
133     )
134
135     return model
```

Bibliography

[1] *Numerical weather prediction*, accessed on 2020-10-12, https://www.ncdc.noaa.gov/data-access/model-data/model-datasets/numerical-weather-prediction.

[2] S. J. Clark, "Quasi-geostrophic influence of the polar stratosphere on the troposphere", Doctor of Philosophy in Mathematics, PhD thesis (University of Exeter, 2017).

[3] C. V. Mark Holmstrom Dylan Liu, "Machine learning applied to weather forecasting", accessed on 2020-05-12 (2016).

[4] M. Phi, "Illustrated guide to recurrent neural networks", accessed on 2020-05-12 (2018).

[5] M. Phi, "Illustrated guide to lstm's and gru's: a step by step explanation", accessed on 2020-05-12 (2018).

[6] J. Brownlee, "A gentle introduction to the rectified linear unit (relu)", accessed on 2020-05-12 (2019).

[7] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. Woo, *Convolutional lstm network: a machine learning approach for precipitation nowcasting*, 2015.

[8] *Era5*, accessed on 2020-10-12, https://www.ecmwf.int/en/forecasts/datasets/reanalysis-datasets/era5.

[9] S. Rasp, P. D. Dueben, S. Scher, J. A. Weyn, S. Mouatadid, and N. Thuerey, *Weatherbench: a benchmark dataset for data-driven weather forecasting*, accessed on 2020-10-12, 2020.

[10] M. Éireann, *Operational nwp in met éireann*, accessed on 2020-12-28, https://www.met.ie/science/numerical-weather-prediction/operational-nwp-in-met-eireann.

[11] TensorFlow, "Time series forecasting", accessed on 2020-05-12 (2020).

[12] J. A. Weyn, D. R. Durran, and R. Caruana, "Can machines learn to predict weather? using deep learning to predict gridded 500-hpa geopotential height from historical weather data", Journal of Advances in Modeling Earth Systems **11**, 2680–2693.

[13] J. Brownlee, "A gentle introduction to batch normalization for deep neural networks", accessed on 2020-12-28 (2019).

[14] J. Brownlee, "A gentle introduction to dropout for regularizing deep neural networks", accessed on 2020-12-28 (2018).

[15]P. Sharma, "Keras dense layer explained for beginners", accessed on 2020-12-28 (2020).

[16]opensource.com, *What is open source?*, accessed 2019-04-11, https://opensource.com/resources/what-open-source.

[17]T. Macaulay, *What are the advantages of open source software?*, (2017) https://www.computerworlduk.com/galleries/open-source/advantages-of-open-source-software-3664823/.

[18]P. S. Foundation, "Python", Version 3.7.1.

[19]P. Gleeson, *Should data scientists learn javascript?*, (2018) https://medium.freecodecamp.org/should-data-scientists-learn-javascript-e611d45804b8.

[20]*Resolution*, accessed on 2020-10-16, https://content.meteoblue.com/ru/specifications/weather-model-theory/resolution.

[21]V. Beal, *Alpha version*, accessed 2019-04-29, https://www.webopedia.com/TERM/A/alpha_version.html.

[22]J. A. Weyn, D. R. Durran, and R. Caruana, "Improving data-driven global weather prediction using deep convolutional neural networks on a cubed sphere", Journal of Advances in Modeling Earth Systems **12**, e2020MS002109 10.1029/2020MS002109, e2020MS002109 (2020).