AMSIMP: Using Machine Learning to Improve Numerical Weather Prediction

Conor Casey

Teacher: Ms. Abbott

Pobalscoil Inbhear Scéine

This project is submitted into the category:

Physical Sciences

SciFest National 2020

"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

This report hypothesises that it's possible to train a RNN on data from the past 15 years, that such a network captures weather patterns, and that such a model will improve numerical weather prediction.

This project has been enhanced since it was submitted into SciFest@College. To understand the performance increase, a comparison between the performance of the current and previous model was done. Air temperature is the only parameter examined for this comparison. Concerning the two metrics, RMSE and MAE, there has been a dramatic performance improvement. There has been a mean decrease of 50.6% in the aforementioned metrics. The performance has not plateaued, which is promising.

It was expected that the development of a ML model would decrease the computational resources required. Once the model has trained, a performance increase of 3.59× was observed in comparison against a physical model of a similar resolution. It should be noted that the benchmark of the software was run on a consumer-grade laptop; while the benchmark of the IFS T42 model was performed on machine with 36 cores. Hence, a further increase can be expected.

The results are promising, the architecture's spatial awareness is lacking, as evident by the current ACC values. It outperforms persistence and climatology forecasts and performs better than a physical model after five forecast days; more work, potentially incorporating a CNN will improve the performance. The current model is 4x faster than a physical model of an equivalent resolution.

Hence, the proposed hypothesis has partially been proven.

Acknowledgements

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.

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!

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 our 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.

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.

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

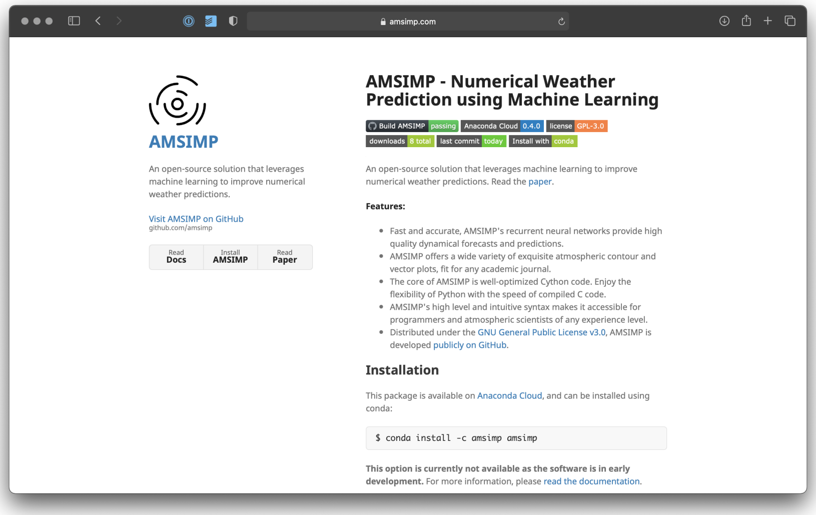
I think it is extremely difficult to pinpoint exactly where the idea originated from; on reflection, however, the spark which really ignited the flame for this project came from a deep interest in the study of the atmosphere and by extension atmospheric science. I initially became interested in this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



An open-source solution that leverages machine learning to improve numerical weather predictions.

Visit AMSIMP on GitHub
github.com/amsimp

AMSIMP - Numerical Weather Prediction using Machine Learning

An open-source solution that leverages machine learning to improve numerical weather predictions. Read the paper.

Features:

- Fast and accurate, AMSIMP's recurrent neural networks provide high quality dynamical forecasts and predictions.
- AMSIMP offers a wide variety of exquisite atmospheric contour and vector plots, fit for any academic journal.
- The core of AMSIMP is well-optimized Cython code. Enjoy the flexibility of Python with the speed of compiled C code.
- AMSIMP's high level and intuitive syntax makes it accessible for programmers and atmospheric scientists of any experience level.
- Distributed under the GNU General Public License v3.0, AMSIMP is developed publicly on GitHub.

Installation

This package is available on Anaconda Cloud, and can be installed using conda:

```
$ conda install -c amsimp amsimp
```

This option is currently not available as the software is in early development. For more information, please read the documentation.

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

NN 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 robust 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]:

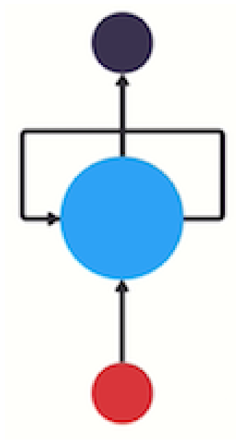


Figure 2.1: Visualisation of a Recurrent Neural Network

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 to the output layer, 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.

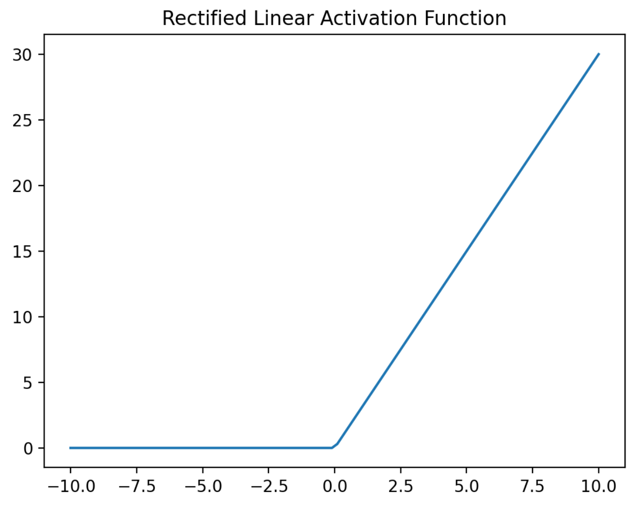


Figure 2.2: Sketch of the Rectified Linear Activation Function

Second, it outputs the results. Third, it compares the prediction to the ground truth 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 stay's the same or is "kept".

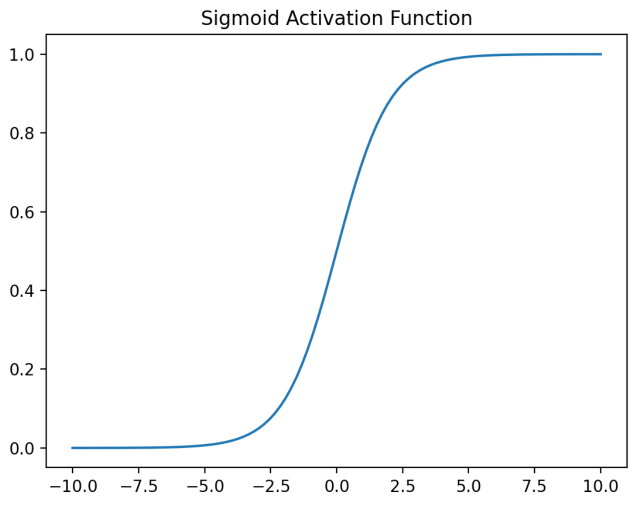


Figure 2.3: Sketch of the Sigmoid Activation Function

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 is then carried over to the next time step[5].

2.1.3 Principal Component Analysis

The number of features in a dataset is referred to as its dimensionality. Dimensionality reduction refers to techniques that reduce the number of input variables in a dataset. More input features often make a predictive modelling task more challenging to model, and can be computational expensive. Considering dimensionality reduction is a data preparation technique performed on data prior to modelling, it might be performed after data cleaning and data scaling and before training a predictive model[7].

For a given training window, there is 22,032,000 points. Attempting such a dataset would be computational burdensome, hence, the attractiveness of dimensionality reduction. Dimensionality reduction can reduce the number of data points without severely impacting accuracy. Fewer input dimensions often mean correspondingly fewer parameters or a simpler structure in the machine learning model, referred to as degrees of freedom. A model with too many degrees of freedom is likely to overfit the training dataset and therefore may not perform well on new data. It is desirable to have simple models that generalise well, and in turn, input data with few input variables. The problem is high-dimensional functions have the potential to be much more complicated than low-dimensional ones, and that those complications are harder to discern. The only way to beat this problem is to incorporate knowledge about the data that is correct[7]. A popular choice for dimensionality reduction is principal component analysis, and was the technique selected for this project.

Definition 7 *Principal Component Analysis, is a dimensionality reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.*

Reducing the number of variables of a data set naturally comes at the expense of accuracy, but the trick in dimensionality reduction is to trade a little accuracy for simplicity. Because smaller data sets are easier to explore and visualise and make analysing data much easier and faster for machine learning algorithms without extraneous variables to process[8].

First, the dataset is standardised to the range of the continuous initial variables so that each one of them contributes equally to the analysis. Second, to see if there is any relationship between the variables of the input dataset, covariance matrix computation is done. The covariance matrix is not more than a table that summaries the correlations between all the possible pairs of variable. The eigenvectors and eigenvalues of the covariance matrix is computed in order to identify the principal components. Principal components are new variables that are constructed as linear combinations or mixtures of the initial variables. PCA tries to put maximum possible information in the first component, then maximum remaining information in the second and so on. Organizing information in principal components this way, will allow for the reduction of dimensionality without losing much information, and by discarding the components with low information and considering the remaining components as your new variables[8]. For the purposes of this project, 99% of the variance was retained in order to balance accuracy with reducing the number of features.

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[9].

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[10]. 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 3° (60×120 grid points) and a temporal resolution of 2 hours. In regards to pressure surfaces, 17 vertical pressure levels were ultimately chosen: 1, 2, 5, 10, 20, 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 950, 1000 hPa. Note that it is common to use pressure in hectopascals as a vertical coordinate instead of physical height. The pressure at sea level is approximately 1000 hPa and decreases roughly exponentially with height. 850 hPa is at around 1.5 km height. 500 hPa is at around 5.5 km height. The data is split into yearly NetCDF files for each variable. The entire dataset at 3° resolution has a size of 75GB. The available variables were chosen based on meteorological consideration. Geopotential, temperature, humidity and wind are prognostic state variables in most physical NWP and climate models[10].

2.2.2 Integrated Forecasting System

The Integrated Forecast System is a global numerical weather prediction system jointly developed and maintained by the European Centre for Medium-Range Weather Forecasts. 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 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.*

It is one of the predominant global medium-range models in general use worldwide; its most prominent rival is 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. It 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[10]. The Integrated Forecasting System will be used as a comparison against the neural network.

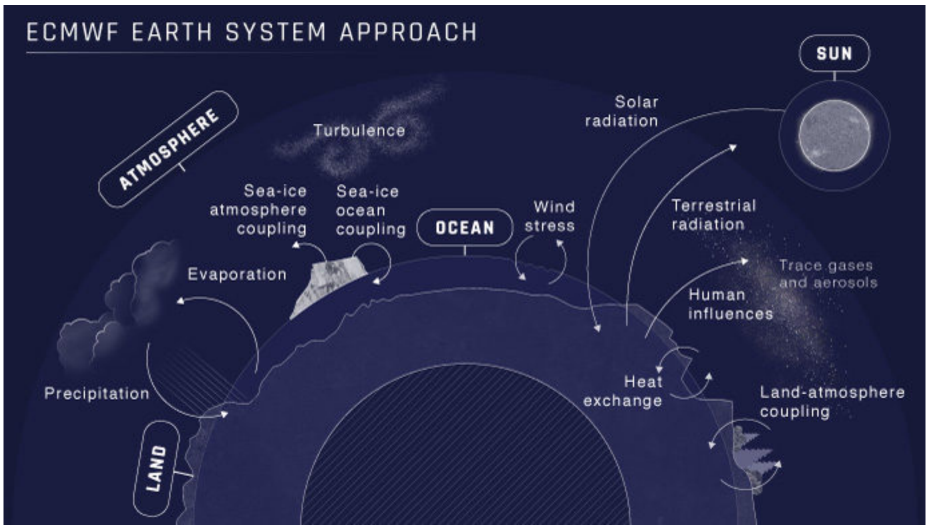


Figure 2.4: ECMWF Integrated Forecast System

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

The dataset consists of five features: geopotential, zonal wind, meridional wind, air temperature, and relative humidity. For a single day, there will be twelve observations. The goal for this project will be to predict the relevant atmospheric parameter in 6 hours time given the last fifteen days of data. In order to make such predictions, it is necessary to create a window of the last 180 (15×12) observations to train the model[11]. The neural network was trained on observational data from 2005 to 2016. The remainder of the dataset, 2017 to 2019, was preserved for validation, testing and benchmarking the neural network against physics-based models.

At the start, a seed is set in order to ensure reproducibility. As mentioned previously, it is important to scale features before training a neural network. Normalisation is a common way of doing this scaling by subtracting the mean and dividing by the standard deviation of each feature. In order for the most optimal performance, the method "MinMaxScaler" from the library, scikit-learn, is utilised within the software[12]. An LSTM requires a 1-dimensional sequence, however, the atmosphere is a 3-dimensional system. Hence, it is necessary to flatten the 3-dimensional vector that represents the state of the atmosphere. This is done in order to avoid the need of repeatably running the RNN. Batches are then created to split the data into manageable sequences. A large batch size is used in order to greater emphasise values at the extremes, as such observations are particularly important in numerical weather predictions in tracking hurricanes and tornadoes. The diagram on the following page shows how the data is represented after flattening the data and batching it (provided by Tensorflow)[11].

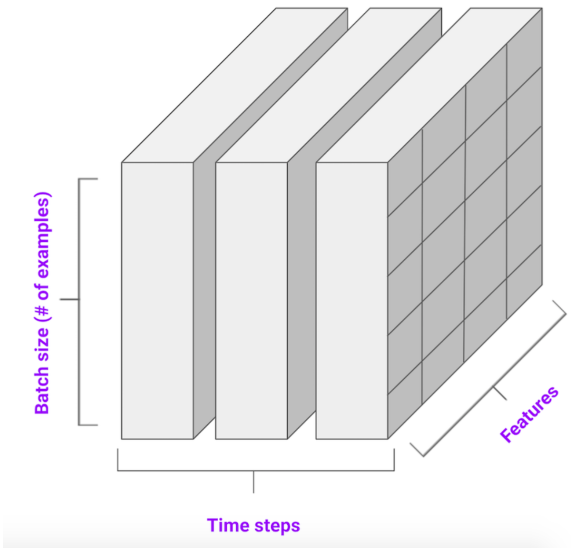


Figure 2.5: Visualisation of how the data is represented after flattening and batching.

Following this process, the data is fed into the RNN. The LSTM model is built using Keras in TensorFlow, which is an free and open-source software library for machine learning. It was developed by the Google Brain Team[13]. It is apparent that a multi-step model is needed as the model needs to learn to predict a range of future values. The source code for the LSTM model developed for the software is shown below:

```
# Prepossessed historical data, which has been flatten and batched.
x_data, y_data, x_val, y_val = prepossessing_function(input_data)
# Prepossessed initial conditions, which has been flatten and batched.
initial_conditions = prepossessing_function(input_initialconditions)

# The network is shown data from the last 15 days.
past_history = 15 * 12

# The network predicts the next 6 hours worth of steps.
future_target = 0.5 * 12

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

```python
# Create model.
model = Sequential()
model.add(
    LSTM(
        480, activation='relu', input_shape=(past_history, features)
    )
)
model.add(RepeatVector(future_target))
model.add(LSTM(480, activation='relu', return_sequences=True))
model.add(TimeDistributed(Dense(features)))
model.compile(
    optimizer=opt, loss='mse', metrics=['mean_absolute_error']
)

# Train.
model.fit(
    x_data, y_data, validation=(x_val, y_val), epochs=epochs, batch_size=120
)

# Predict (this function is iterated to the forecast length required).
future_state = model.predict(initial_conditions)
# Invert normalisation, and flattening.
future_state = inverse_prepossessing(future_state)
```

The model consists of four LSTM layers, which in combination are able to produce a more accurate and reliable prediction than a single LSTM layer. As is evident, the activation function for each LSTM is the rectified linear activation function, which is built into Keras. The number of epochs can be specified by the end user depending on the computational resources they have and what they need. More epochs will evidently lead to a more accurate neural network.

Definition 9 *An epoch is one forward pass and one backward pass of all the training examples.*

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.*

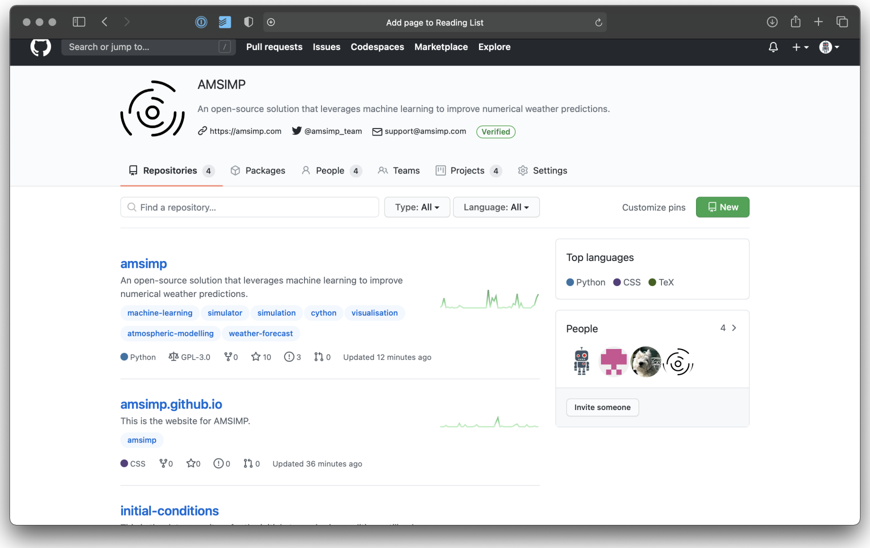
Source code is the code that computer programmers use to modify, and change how a piece of software functions[14]. Programmers with access to the source code can improve that program by adding features to it, fixing bugs, or by improving the documentation of the surrounding source code.

According to Tom Macaulay[15], 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`.



AMSIMP

An open-source solution that leverages machine learning to improve numerical weather predictions.

https://amsimp.com @amsimp_team support@amsimp.com Verified

amsimp

An open-source solution that leverages machine learning to improve numerical weather predictions.

machine-learning simulator simulation cython visualisation atmospheric-modelling weather-forecast

Python GPL-3.0 0 10 3 0 Updated 12 minutes ago

amsimp.github.io

This is the website for AMSIMP.

amsimp

CSS 0 0 0 0 Updated 36 minutes ago

initial-conditions

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[16].

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.[17]

3.2.2 Switching to Cython

During the continued development of the software, I discovered a rather severe bottleneck as a result of my choice of programming language: the execution time performance. The culprit behind this bottleneck was a result of Python's dynamically typed nature. Generally, you can classify programming languages into two categories: dynamically typed languages and statically typed languages.

Definition 11 *A dynamically typed language is one in which the type of the variable is not known at compile time, and generally, you can define a variable as an integer type at the start of the program for example and later redefine it as a string type.*

On the other hand, a statically typed language is a language where the type of variable is known at compile time, and during the execution of the program, the variable type must remain constant.

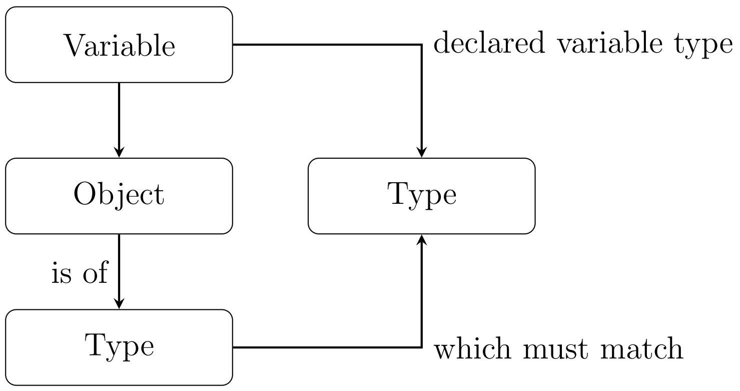


Figure 3.2: How statically typed languages handle variables.

A statically typed language is particularly useful for software with a large number of loops, or repetitive calculations contained within them. During a loop, a dynamically typed language will continuously check the type of variable after each loop, even if it has done so in previous loops. In a small number of loops, this has a negligible impact on the execution time; however, in a large enough quantity, this has a cumulative effect, significantly stifling the execution time performance. Generally, a piece of software written in a statically typed language will execute a lot faster. The reason behind picking Cython over other statically typed languages, such as C, is that Cython retains all the benefits of Python, with its extensive scientific package library and its simple syntax, while gaining the performance benefits of a statically typed language.

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 recurrent neural network on an atmospheric reanalysis dataset based on data from the past 15 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.', 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 benchmark 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. These are the two simplest possible forecasts. For the climatological forecast, climatology was computed from the ERA5 Atmospheric Reanalysis dataset from 1979 to 2016. 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[10].

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. 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 2017 00UTC, while the last training target is 31 December 2018 23UTC.

For benchmarking purposes, the 500 hPa geopotential and the 800 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 800 hPa temperature field was chosen as the secondary verification field because temperature is a more impact-related variable. The 800 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.

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

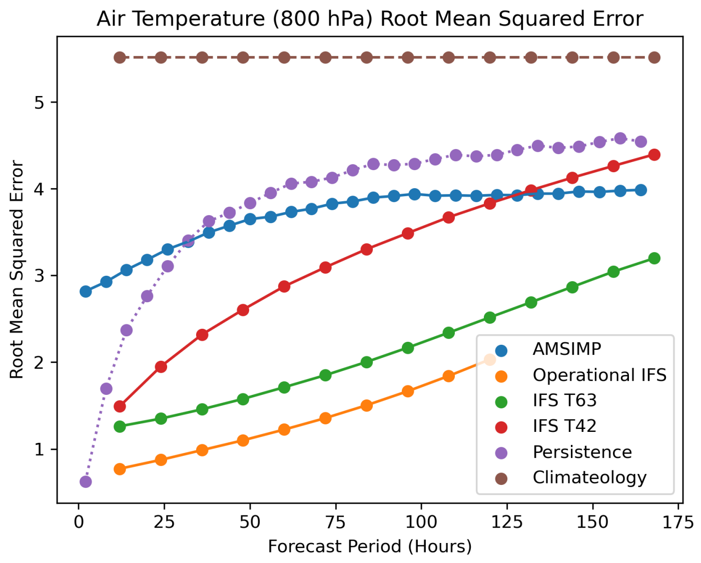


Figure 5.1: Air Temperature at 800 hPa

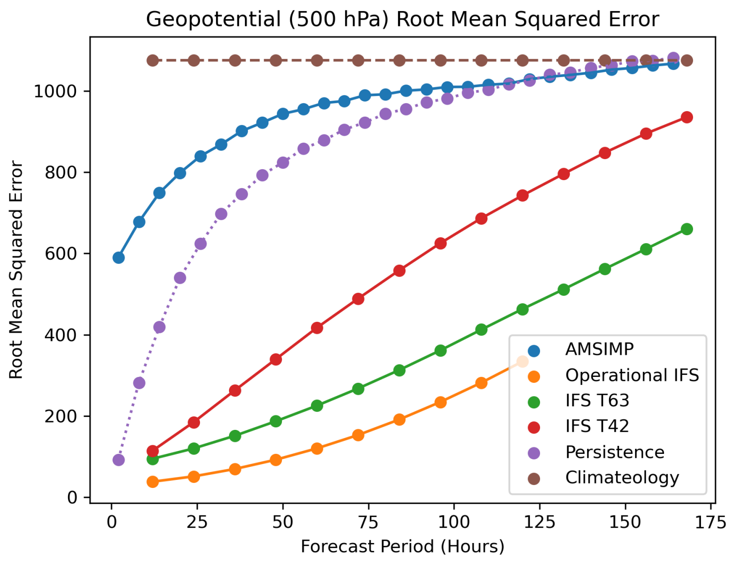


Figure 5.2: Geopotential at 500 hPa

5.2 Mean Absolute Scaled Error

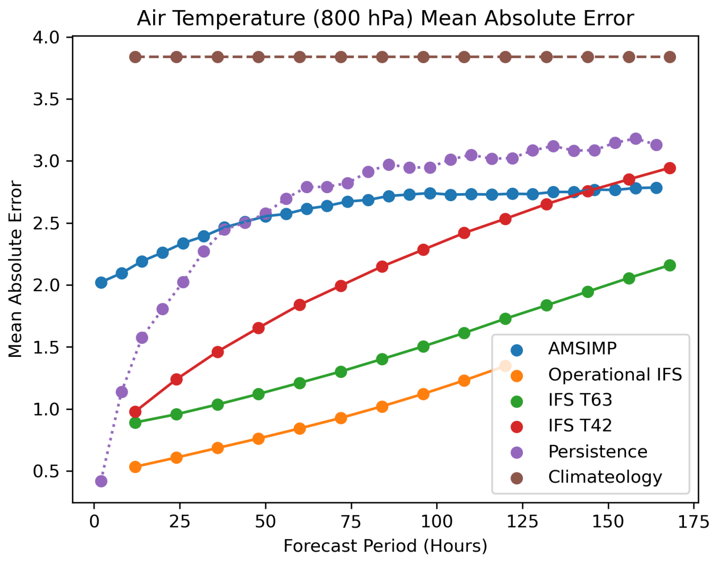


Figure 5.3: Air Temperature at 800 hPa

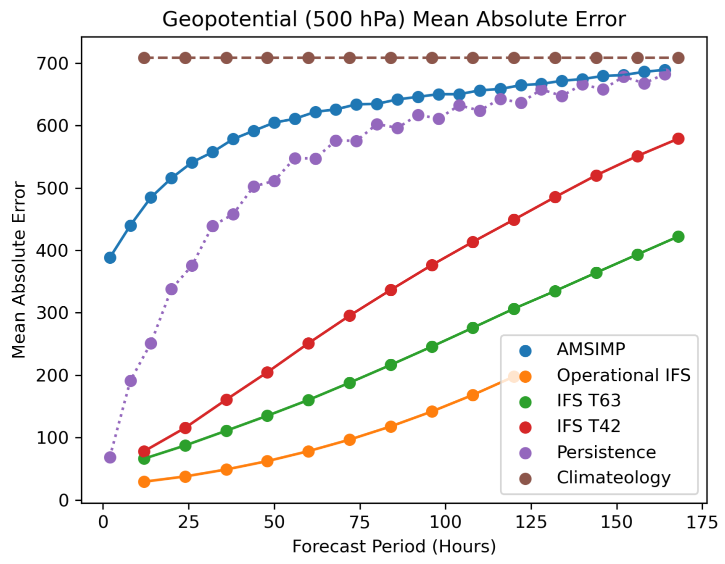


Figure 5.4: Geopotential at 500 hPa

5.3 Anomaly Correlation Coefficient

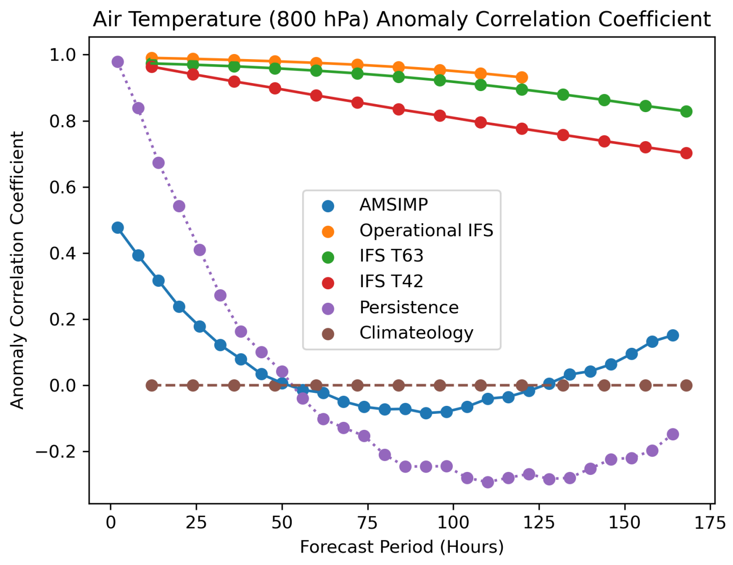


Figure 5.5: Air Temperature at 800 hPa

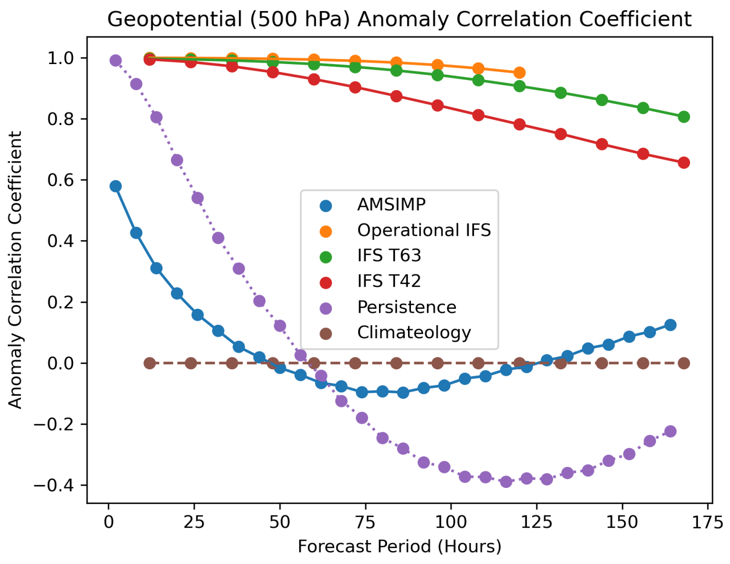


Figure 5.6: Geopotential at 500 hPa

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 at 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 recurrent neural network on an atmospheric reanalysis dataset based on data from the 15 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.', 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 recurrent neural network on an atmospheric reanalysis dataset based on data from the 15 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.

This project has been significantly enhanced and improved since it was submitted into SciFest@College Online last May. To gain an understanding of the performance increase since that time, a comparison between the performance of the current model against the performance of the previous model is shown in section A.1. Air temperature is the only parameter examined for this comparison, as the previous model did not incorporate geopotential. Concerning the two metrics, root mean squared error and mean absolute error, there has been a dramatic performance improvement. There has been a mean decrease of 52.5 % in the root mean squared error values, and a mean decrease of 48.6 % in the mean absolute error values; on average a 50.6 % decrease in error metrics across the board. This demonstrates the continued improvement and enhancement of the models over the last few months; but, it also demonstrates that the performance of the software can still be improved drastically. The performance increase has not reached a plateau, which is extremely promising.

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 3.59 times can be expected in comparison against a physics-based model of a similar resolution, the ECMWF IFS T42. It is also important to note that the benchmark of the software was run on a consumer-grade, MacBook Pro; while the benchmark of the ECMWF IFS T42 model was performed on a single XC40 node with 36 cores. Hence, a further increase in performance can be expected with a similar configuration.

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 can be generally regarded as useful, particularly on longer periods and in relation to air temperature, in particular, however, the models generally fail to beat well established physics-based models at this time. The model is significantly better at creating air temperature predictions and appears to suffer with geopotential predictions. The root mean squared error and mean squared error demonstrate that the model's air temperature becomes useful after approximately 24 hours of forecast time, with the model ultimately beating the ECMWF IFS T42 model after approximately 96 hours. The picture for geopotential is less rosy, with the root mean squared error and mean squared error demonstrating that the model's geopotential predictions become useful after approximately 96 hours of forecast time. An interesting point to note is that the error values initially are quite high, the error values appear to plateau. This may suggest that the model may be quite useful at generating climate forecasts. Concerning spatial awareness as measured by the anomaly correlation coefficient, both the mode's geopotential and air temperature predictions become useful after approximately 120 hours of forecast time. The spatial awareness of the model can be generally regarded as quite poor, it appears that the spatial aspect of a weather forecast was not captured by the long short-term memory cell. An interesting point to note again is that the anomaly correlation coefficient begins to increase after approximately 72 hours of forecast time. It appears the model has captured the long term climate trends while neglecting local weather trends. This is an unexpected result and needs to be investigated further.

Hence, the hypothesis that was proposed has partially 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 lead to a possible source of error:

- As mentioned in section 2.1.3, principal component analysis was used as a form of dimensionality reduction to reduce the number of features in the dataset, ultimately reducing the amount of memory required for the training phase of the project. For a given training window, there is 22,032,000 points. Attempting such a dataset would be computational burdensome, hence, the attractiveness of dimensionality reduction. During this process, 99% of the variance was retained. While this maintains the majority of the important features, considering the atmosphere is a dynamical system, it is effected by chaos theory. Considering, chaos theory states that a small change in initial conditions can lead to dramatically different outcomes; this process of dimensionality reduction may be fundamentally hindering the possible performance of the software.

- As mentioned previously, the benchmarking process was inspired by the open-source project, known as WeatherBench. WeatherBench proposes simple and clear evaluation metrics which will enable a direct comparison between different forecasting models[10]. This was done in order to conform with the meteorological and wider academic community. It be must be noted, however, that our current models predict values for five distinctive fields. As such, three fields were not rigorously analysed and studied. This has the potential of the models, in reality, either performing better than the results may suggest, or the limited results may be hiding a demon.

- As mentioned in section 2.2.1, it was decided to use a spatial resolution of 3° (60×120 grid points) and a temporal resolution of 2 hours. In regards to pressure surfaces, 17 vertical pressure levels were ultimately chosen: 1, 2, 5, 10, 20, 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 950, 1000 hPa. 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[18]. 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[19]. 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:

- As mentioned in section 2.3, it is necessary to flatten the 3-dimensional vector that represents the state of the atmosphere in order for it to match the shape of the LSTM cell. While this is an effective approach, a new approach has been shown to be more effective. By combining a convolutional neural network with a neural network, it will result in a more accurate output for spatial-temporal type data. A convolutional neural network is a class of deep neural networks. They are also known as shift invariant or space invariant artificial neural networks, based on their shared-weights architecture and translation invariance characteristics. Considering the current model particularly suffers in the metric of the anomaly correlation coefficient, which is a spatial correlation between the forecast anomaly and the verifying analysis anomaly, a model which is built with spatial awareness may further improve performance. This will be investigated in the coming months. It may also reduce the need to use dimensionality reduction, as such a model may dramatically decrease the memory requirements to train the software.

- 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 five parameters on which the machine learning models were trained upon were: air temperature, relative humidity, geopotential, zonal wind, and meridional wind. 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, our model will be trained on such parameters in order to provide the most useful weather forecast possible.

Appendices

A.1 Comparison against Previous Model

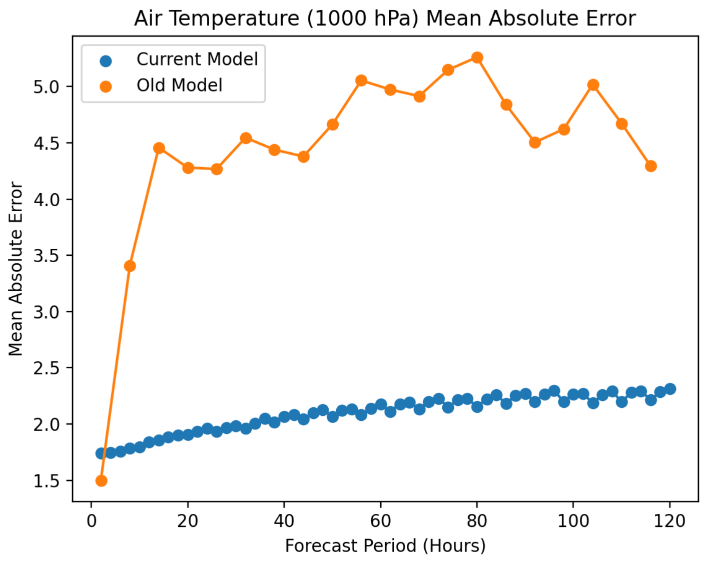


Figure 1: MAE for Air Temperature at 1000 hPa

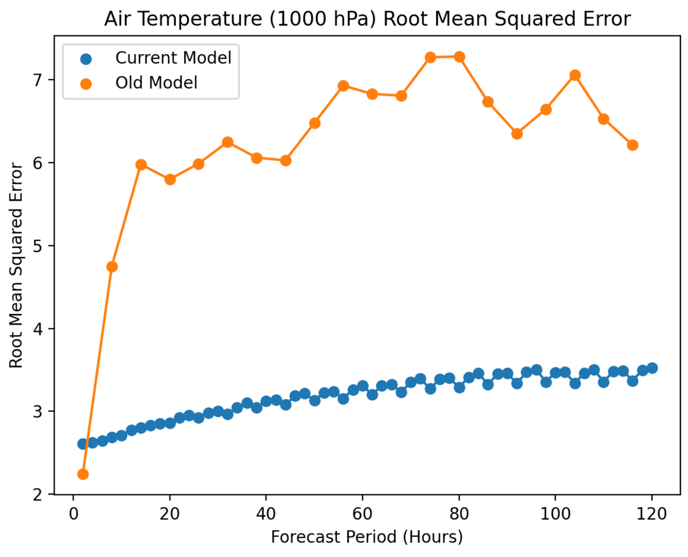


Figure 2: RMSE for Air Temperature at 1000 hPa

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]J. Brownlee, "Introduction to dimensionality reduction for machine learning", accessed on 2020-10-12 (2020).

[8]Z. Jaadi, "A step by step explanation of principal component analysis", accessed on 2020-10-12 (2019).

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

[10]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.

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

[12]F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: machine learning in Python", Journal of Machine Learning Research **12**, 2825–2830 (2011).

[13]Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, *TensorFlow: large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015.

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

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

[16]P. S. Foundation, “Python”, Version 3.7.1.

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

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

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