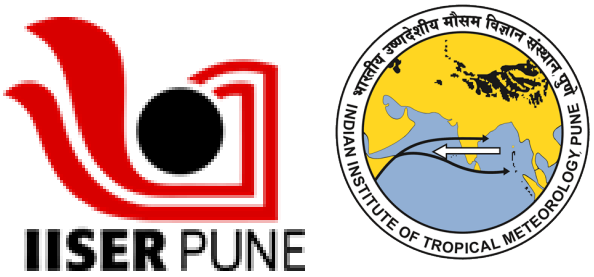
Applications of Convolutional-Recurrent Neural Networks in Weather Forecasting



A thesis
submitted towards the partial fullfilment of
BS-MS dual degree programme
from August 2020 to May 2021

by

Arya Samanta

under the guidance of

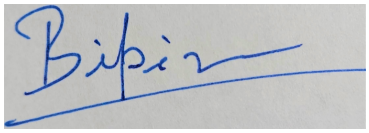
Dr. Bipin Kumar and Dr. Rajib Chattopadhyay

Scientist E, HPCS, IITM, Pune

Indian Institute of Science Education and Research Pune

Certificate

This is to certify that this dissertation entitled Applications of Convolutional-Recurrent Neural Networks in Weather Forecasting submitted towards the partial fulfillment of the BS-MS degree at the Indian Institute of Science Education and Research, Pune represents original research carried out by Arya Samanta at Indian Institute of Tropical Meteorology, Pune, under the supervision of Dr. Bipin Kumar and Dr. Rajib Chattopadhyay during academic year August 2020 to May 2021.

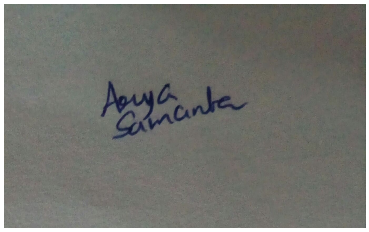


Supervisor:
Dr. Bipin Kumar
Scientist E, HPCS
IITM, Pune

Co-Supervisor:
Dr. Rajib
Chattopadhyay
Scientist E, HPCS
IITM, Pune

Declaration

I, hereby declare that the matter embodied in the report titled Applications of Convolutional-Recurrent Neural Networks in Weather Forecasting is the results of the investigations carried out by me at the Indian Institute of Science Education and Research, Pune from the period 15-08-2020 to 15-05-2021 under the supervision of Dr. Bipin Kumar and Dr. Rajib Chattopadhyay and the same has not been submitted elsewhere for any other degree.



Arya Samanta
20161157
BS-MS
IISER Pune

Acknowledgements

I would like to thank my supervisors, Dr. Bipin Kumar and Dr. Rajib Chattopadhyay for their constant valuable insight and guidance throughout the project. Core parts of the project are formed with discussion with Mr. Manmeet Singh without whom this would not be possible, I am deeply grateful to him for that. I thank Dr. Anindya Goswami for the opportunity to undertake my master's research project in IITM, Pune and for providing me with valuable suggestions and discussions and for embibing me with mathematical clarity throughout my time in IISER. I would like to thank my parents above all for supporting all my endeavors and being there for me all the time and making me their utmost priority.

My friends have been a constant source of motivation and have helped in various capacities throughout my time in IISER, and I would like to thank them all, in no particular order, Adithya S., Vishrut Patel, Yajushi Khurana, Duttatreya, Shubhalaxmi Mukherjee, Durgesh Ajgaonkar, Mansi Budamagunta, Srishti Gupta, Manjul Yadav, Sudheesh S, Adithya Shetty, Mohan Karra, Jitesh Seth, Divyansh Vardhan, Hitesh Wankhede and Koustav Halder.

I would like to thank my previous project mentors under whom I gradually developed the skills and scientific prowess to be able to carry out this final endeavor in IISER. I am grateful to Dr. Sourabh Dube, Dr. Rama Mishra, Dr. Chaitanya Athale and Dr. A. K. Nandakumaran.

Abstract

In this thesis, we use convolutional long short-term memory based neural network architectures to predict certain basic dynamics of weather variables. First, rainfall dynamics are studied across direct and iterative forecasting methodologies from India-restricted data set. Next, we develop models capable of predicting dynamics over a global cube-sphere grid and as a proof of concept study the dynamics of geopotential height while comparing them on a benchmark dataset namely WeatherBench. Experiments indicate that ConvLSTM2D-based models are advantageous for iterative forecasting than direct forecasting than conv2D-based models by slight margins and need further development. When applied to cube-sphere grid, `CubeSphereConvLSTM` did perform better than the benchmark CNN iterative model but did not beat the best operational systems.

Contents

List of Figures

Chapter 1

Introduction

Neural Networks are a subclass of machine learning methods that have recently gained much popularity as they specialize in high dimensional pattern recognition given enough supervised data to learn on which are otherwise hard to algorithmically model in a step-wise interpretive fashion. They are mostly used in image (spatial) and natural language (temporal) pattern recognition by the use of convolution neural networks (CNN) and recurrent neural networks (RNN) respectively (LeCun *et al.* (2015)). Weather prediction being a high-dimensional regression problem with an inherent spatial and temporal structure therefore poses an interesting question of applicability of convolutional-recurrent neural networks (CRNN).

1.1 Problem Statement

Weather forecasting is a prediction problem of the conditions of the atmosphere given past data over location and time. Naturally, this can be stated as a spatiotemporal sequence forecasting problem.

As a dynamical system, this can be represented as vectors of variables over a spatial grid of $M \times N$ locations. On these locations, say, P variables are measured. Therefore any observation at a given time is from the space $R^{M \times N \times P}$, where R is the domain of the observed variables. Given a certain periodicity of the past data, it can be represented as a sequence of elements from this aforementioned space as $X_1, X_2, X_3, \ldots, X_t$. Then the forecasting problem is defined as to predict the least error $K-$length sequence in the future given the previous t observations (including the current one) as input. This can be represented as

$$\hat{X}_{t+1}, \cdots, \hat{X}_{t+K} = f(X_{t+1}, \ldots, X_{t+K} \mid X_1, X_2, X_3, \ldots, X_t)$$

where f is the forecasting function. The function f here is a high-dimensional parameterized function based on artificial neural network architectures. In other words, our problem reduces to finding a suitable architecture which reduces the error between the predicted and the ground truth of observations.

1.2 Previous Work

1.2.1 On Post-processing, Downscaling and Non-Forecasting Applications

Operational weather prediction uses physics-based NWP (Numerical Weather Prediction) models which try to understand the underlying equation of sub-processes such as air flow, cloud formation etc., in the atmosphere on a grid scale of 10 km. As of now, NNs have been used either in post-processing, downscaling, or extreme event detection.

Post processing examples involve extracting information such as prediction uncertainity in NWP weather forecasts as in Scher and Messori (2018). Other examples include Chapman *et al.* (2019) and Davò *et al.* (2016).Rasp and Lerch (2018) used NNs for ensemble probabilistic postprocessing of NWP. Downscaling refers to the problem to generating high resolution images from low resolution images primarily for the purpose of providing more granular data for local impact studies. There two broad forms, dynamic downscaling and statistical downscaling. Deep learning techniques such as Generative Adversarial Networks (GANs) have been utilized for statistical downscaling. Rocha Rodrigues *et al.* (2018) downscaled GCM (General Circulation Model) output.Whereas, Kurth *et al.* (2018), Lagerquist *et al.* (2019) and Liu *et al.* (2016) have used NNs for extreme weather detection.

Apart from Neural Networks, classical machine learning techniques have also been used for geo-scientific classification and anomaly detection (Reichstein *et al.* (2019)). But the classical methods of random forests, feed-forward trees etc. had to be hand designed and could not be scaled to huge data with both spatial and temporal diversity.

Apart from these, a middle ground of ML-based weather prediction work includes the improvement of physics parameterizations in GCM (Brenowitz and Bretherton (2018) and Rasp *et al.* (2018)).

1.2.2 On Data-driven Neural Networks-based Forecasting

As opposed to the previous applications of neural networks, recently there has been some work on addressing whether deep learning techniques can be used to generate purely data-driven statistical forecasts which are not explicitly provided any constraints of atmospheric physics. In other words, this amounts to asking whether neural network models can encode a representational learning of physical processes of the atmosphere.

In this regard, one of the first basic papers which discuss the fundamental design choices for a global forecast model based on neural networks is Dueben and Bauer (2018). They have used spatially localized networks similar to Convolutional Neural Networks (CNNs) and Direct Neural Networks (also knows as fully connected neural networks) to predict a single atmospheric variable, 500-hPa geopotential height trained on 7 years of ERA 5 reanalysis data regridded at 6° spatial resolution. Using local stencils resulted in much better results than DNNs though their predictions were just marginally better than persistence at early forecast lead times. Next we see, Scher (2018) use the input and output states of a highly simplified General Circulation Model (GCM) as supervised

data to train a NN to directly emulate the dynamics of the GCM. It is to be noted that although the results showed no long-term drift and beat baselines of climatology and persistence significantly, they were devoid of any seasonal cycles (eternal Northern Hemispheric winter), diurnal cycle or oceans. Similar model emulation study was done by Vlachas *et al.* (2018) where, like the first section of Dueben and Bauer (2018), a neural network was compared against a theoretical high-dimensional chaotic system using Long short-term memory networks (LSTMs).

The next follow-up study by Scher and Messori (2019) on replicating GCM dynamics with seasonal variation and higher horizontal resolution resulted in the CNN architectures performing slightly worse and revealing a more complicated story though these are not attributed to higher computational costs on a limited model but increasing complexity of high-resolution weather being used.

Improving upon the first work of Dueben, Weyn *et al.* (2019) used a fully data-driven neural network based model (WDC19) to forecast observed weather rather than the forecasts states of idealized GCM models. The data used was restricted to the northern Hemisphere and the model was trained to predict 500-hPa geopotential heights and 700 to 300 hPa geopotential thickness from 24 years of atmospheric reanalysis (ERA 5) on a 2.5° spatial resolution. This model did result in a comparison to forecast accuracy of current operational NWP models, which have been refined by decades of research, operate at much higher resolution, and use far more data to describe the initial condition for each forecast. It significantly outperformed persistence and climatology benchmarks, as well as a basic dynamical model such as barotropic vorticity model, which was the type of model used in the earliest years of NWP. Their best CNN formulation was able to outperform a climatological benchmark for root−mean−squared error (RMSE) in the 500−hPa height field out to about 5 days of forecast lead time. Note that the WDC19 model was restricted to the Northern Hemisphere on a latitude−longitude grid and it did not have appropriate boundary conditions at the North Pole and the equator.

This was followed by an attempt to extend such an architecture to another global end-to-end forecasting model in Weyn *et al.* (2020). The most important change being use of a volume−conservative mapping to project global data from regular latitude−longitude grids onto a cubed sphere and develop CNNs which operate on the cube faces using iterative sequence forecasting technique to feed the outputs of the model into itself and improve long-term climate forecasts. Their best CNN forecast outperforms a climatology benchmark at up to 120 hours of lead time, and appears to correctly asymptote towards persistence forecasts at longer lead times up to 14 days.

The later part of this thesis work uses data from a benchmarking dataset called WeatherBench, (Rasp *et al.* (2020)) which is made to provide a common ground to test various models and compare with the existing models.

1.3 Outline of current work

In first part of this thesis, two variants of a convLSTM-based forecasting model are developed. They are a direct and an iterative forecasting model (see section 3.2 for difference

in direct and iterative forecasting). The direct model is based on the architecture specifications followed by George (2020), a preceding master's thesis work done in IITM, Pune. The model was modified to suit iterative forecasting methodology and compared on the same data set used and over the same specifications to identify any possible advantages over either variant.

In the later part, a new implementation of an end-to-end deep neural network forecasting model was studied which develops upon existing ConvLSTM2D architecture. We develop a CubeSphereConvLSTM2D architecture in order to input a cubed-sphere format of global atmospheric data as inspired by Wyen et al. 2020. As the existing ConvLSTM2D keras class itself depends on ConvRNN, RNN and basic Layer class, we had to spend significant amount of time in unpacking and building CubeSphereConvLSTM2D from scratch. The data used for this model is also significantly complex and large in size and therefore required prepossessing before being deployable for usage in this new architecture.

Chapter 2 provides the necessary background material specific to this work. Chapter 3 contains a detailed review of past global end-to-end deep neural network-based models and descriptions of the two data sets used in this work. Chapter 4 details the methods developed in this work and the respective specifications of our model and workflow. Lastly, Chapter 5 presents the performance and results and Chapter 6 discusses the results and future work possible.

Chapter 2

Neural Network Fundamentals

Artificial Intelligence is the use of machines to perform certain computational tasks. In the age of Symbolic AI, it was restricted to solving problems whose solutions were figured by humans and were then converted to machine readable code to automate its implementation. It was assumed for a long time that for a machine to do complex tasks, one would have to explicitly hand-write a sufficiently large amount of rules that would deterministically let the machine behave in a certain manner.

Soon, a different paradigm evolved where a program would be given data (say, a set of supervised input and output) and encoded to update certain parameters that would help it calculate a new output, given a new input. This was a fundamentally different use of machines as previously, in Symbolic AI, no part of the original instructions updated itself unless explicitly done by the programmer. Today, machine learning comprises of computational algorithms that learn a particular set of representation of rules of the problem when supplied with data and answers i.e. the algorithms learn rules from a data-driven approach. A simple schematic representation of the difference between the paradigms of computation is presented in figure 2.1. This section explains the types of neural networks and the framework of the machine learning algorithms used in this work. A basic categorization of machine learning models based on the ways an algorithm learns a pattern are:

- supervised,
- unsupervised,
- semi-supervised, and
- reinforcement learning.

Our model is a supervised machine learning model and uses previously labeled atmospheric data for training. The input space of supervised learning is of type (X, Y). The model aims to find a function f that best transforms via representational learning, X to Y. The supervised learning takes place with the help of the Y as we keep updating f using gradient-based back-propagation of a particular loss function $L(Y, f(X))$ using suitable optimizers and metrics for evaluation.

More specifically, ours is a supervised deep machine learning model for non-linear time series regression, where "deep" refers to the usage of multiple layers of neural networks. Deep learning models have recently become popular as they have been shown

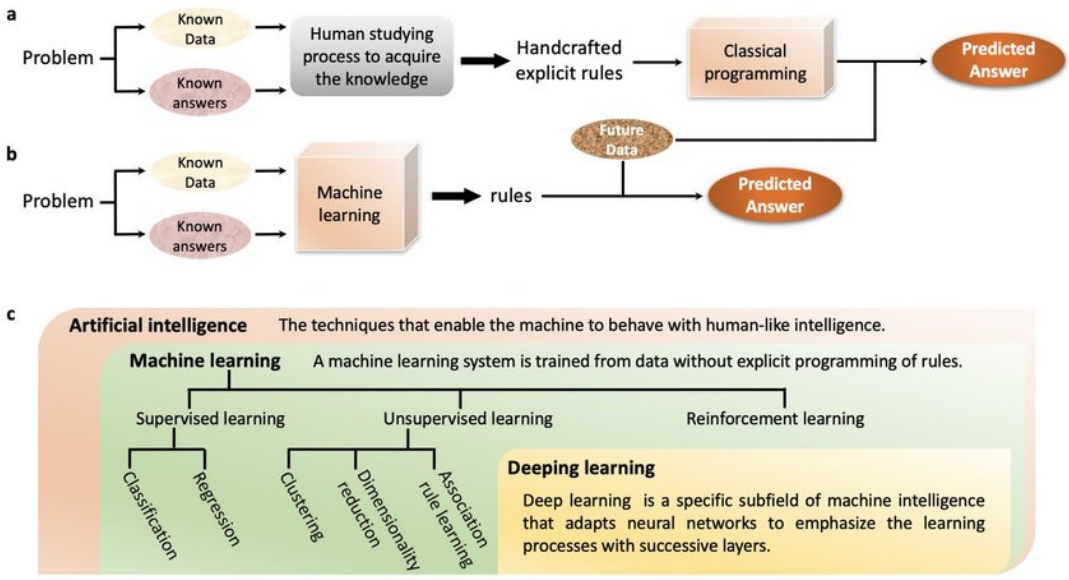


Figure 2.1: From artificial intelligence to deep learning. a. The programming paradigm for symbolic AI. b. The programming paradigm for ML. c. The relationship among artificial intelligence, machine learning, and deep learning. Image from Bian and Xie (2020).

to be universal nonlinear function approximators Nielsen (2015) and also as advances in computational efficiency has made it possible to train such large models. These models are expected to learn meaningful representations of the high-dimensional input through successive layers of transformations.

A general supervised deep learning workflow in shown in figure 2.2. This is the main essence of a supervised deep learning algorithm which loops through the depicted process under some conditional statement. The conditional can be as simple as a certain fixed number of loops (in deep learning parlance, known as epochs) or a while loop until a particular metric that we are interested in has stopped improving after a certain number of epochs, such as the loss value or some scalar function that represents the same.

2.1 NN Layers

The basic building blocks of a neural network are layers which apply transformations to the inputs of the layer into a higher or lower dimensional representation. Mathematically, this can be represented as follows.

Definition 1 (Layer Transformation) *Given an $n-$dimensional input matrix $X \in R^{j_1 \times j_2 \times \cdots \times j_n}$ and a $m-$dimensional layer's weight matrix $w \in R^{l_1 \times l_2 \times \cdots \times l_m}$, a layer performs a well-defined operation $p : R^{j_1 \times j_2 \times \cdots \times j_n} \rightarrow R^{k_1 \times k_2 \times \cdots \times k_o}$, transforming X to an $o-$dimensional output matrix Y.*

The operation, p, can be a multi-step iterative function if the input matrix is a time-series or a single step application of a sigmoid or ReLU activation function. Depending upon the type of operation p, there can be various kinds of layers. Here we present

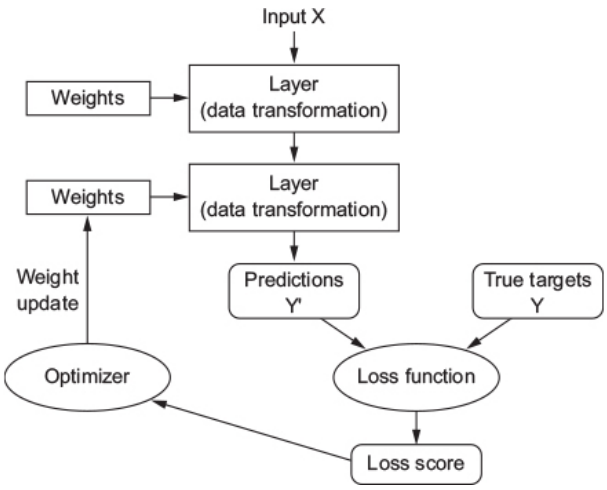


Figure 2.2: Supervised deep-learning workflow. Figure taken from Chollet (2017).

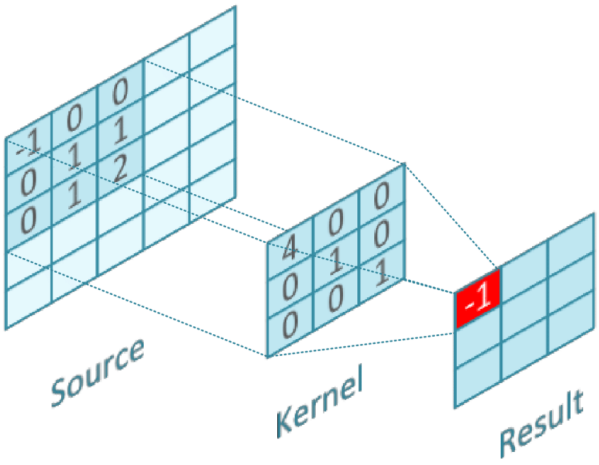


Figure 2.3: Convolution operation of a 5 × 5 image with a 3 × 3 kernel resulting in a 3 × 3 output. Figure taken from Wicht (2018).

the ones that are used in our model. Usually, they are also accompanied by applying a non-linear (called as an activation) function on the output matrix, Y, in an element-wise fashion as a part of an existing layer or as a separate successive layer. Apart from the weight matrices, a layer operation is also characterized by certain hyper-parameters which are set during initialization. The weights are the parameters that the algorithm learns, whereas the hyper-parameters are parameters that control the learning and are not trained from the data. Certain set of consecutive layers are called blocks as grouped by their functionality.

2.1.1 Convolutional Layer

As suggested, this layer performs a convolution operation on its inputs. Depending on the dimensions of the input, a conv2D or a conv3D layer transformation may be used. In its most basic step it takes a $m \times n$ input matrix and given a $k \times k$ kernel (weight) matrix which will stride (move) over the input s times doing element-wise dot product

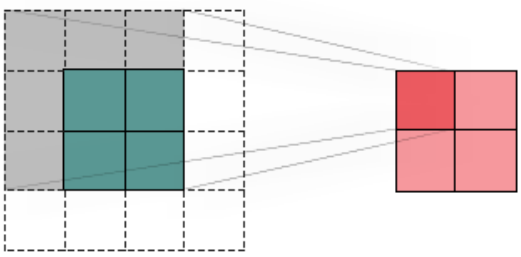


Figure 2.4: Padding input to preserve input matrix dimensionality. By Narges Khatami - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=70625697 / GFDL.

and return a $(m-k)/s+1 \times (n-k)/s+1$ output matrix as shown in figure 2.3.

As it involves a reduction in the dimension of the input matrix, the input matrix is usually padded with suitable amount of zeros (if $s = 1$, then a padding of $\lfloor k/2 \rfloor$ zeros is required to preserve the dimensionality. Given the nature of the inputs, there can be more ways to pad the inputs instead of just zeros, though it is a common practise in computer vision to use zeros. In 4.3.2 we describe a natural padding possible for global gridded meteorological data.

It should be noted here that the values of the kernel are themselves learnable parameters, whereas the number of filters, kernel size, k, stride length, s and the padding width are hyper-parameters to be set while initializing the layer. Stride defaults to one and padding defaults to none. Also, an input to a conv2D layer is a 4-dimensional matrix of the form (samples, channels, height, width). As inherited from computer vision usage each image is not defined as just a 2D matrix but a 3D matrix. This is to incorporate the fact that images are usually a combination of RGB values and has three 2D matrices for each channel that describes an image. Similarily, a conv3D layer is capable of transforming a 5D input of the type (samples, channels, height, width, depth) where it is be understood that a single input itself here is 3D object with as many as channels to represent different characteristics of the input.

2.1.2 Recurrence Layer

Recurrence layers are used for transformation of sequential data. The simplest form of input that a recurrence layer is a 2-dimensional matrix of the type (samples, features). Recurrence layers are characterized by two components, a cell that performs transformations to a single time step of the input data and a function that repeats the process of the transformation for each following time step while carrying forward certain information in the form of weight matrices.

A schematic representation of a simple recurrence layer is shown in figure 2.5. The cell in this representation is the same as single layer of a Dense Neural Network (also knows as Direct NN/Linear NN)

In 1997, Long Short-Term Memory Cell was introduced by Hochreiter and Schmidhuber (1997) . The two main advantages of LSTM over simple RNN were, firstly, the

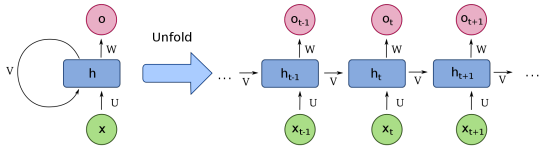


Figure 2.5: Schematic representation of a RNN layer. By fdeloche - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=60109157

addition of a memory matrix which was carried forward into all future time steps and helped in learning local temporal patterns that effect in long-range effects. Secondly, to counter the problem of vanishing or exploding gradient problem.

Following are the equations which represent the transformations to a single time step into a LSTM cell. Given a $d-$length input and a LSTM initialized with h hidden units where

- $x_t \in \mathbb{R}^d$ as an input $d-$lenght vector (1-D matrix) to the LSTM
- $h_t \in \mathbb{R}^h$ is the hidden state vector (also the output vector in LSTM and is initialized with a zero vector of $h-$length if h_0 is not provided)
- $i_t \in \mathbb{R}^h$ is input gate activation vector
- $f_t \in \mathbb{R}^h$ is forget gate activation vector
- $o_t \in \mathbb{R}^h$ is output gate activation vector
- $\tilde{c}_t \in \mathbb{R}^h$ is memory cell input activation vector
- $c_t \in \mathbb{R}^h$ memory cell state vector (zero vector of $h-$length if c_0 is not provided)
- $b \in \mathbb{R}^h$ and $w \in \mathbb{R}^{d\times h}/\mathbb{R}^{h\times h}$ depending on whether it is being multiplied to x_t or h_t
- $\sigma : \mathbb{R} \to \mathbb{R}$ is the sigmoid activation function
- $\sigma_h : \mathbb{R} \to \mathbb{R}$ is by default the hyperbolic activation function
- $\circ : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is the Hadamard product (element-wise product binary operation)

$$
\begin{aligned}
i_t &= \sigma(w_{ix}x_t + w_{ih}h_t + b_i) \\
f_t &= \sigma(w_{fx}x_t + w_{fh}h_t + b_f) \\
o_t &= \sigma(w_{ox}x_t + w_{oh}h_t + b_o) \\
\tilde{c} &= \sigma_h(w_{cx}x_t + w_{ch}h_t + b_c) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
\tag{2.1}
$$

Consequently an RNN can be customized to perform transformation on more complex input matrices by using cells suitable for such inputs. We next present one such layer architecture which takes 5-dimensional inputs of the type (samples, timesteps, channels, height, width).

2.1.3 Convolutional Recurrence Layer

This is the main class of layers that we are going to use in this model. As it suggests, it is a layer which is capable of transforming input data which are a time series of images. It was first introduced by Shi *et al.* (2015) in the implementaion of ConvLSTM2D layer. The basic equations governing the transformation of this layer are exactly like that of a LSTM with matrix multiplication replaced with convolution operations in order to extract spatial features. Following are the equations performed inside a single layer of ConvLSTM2DCell class initialized with hyper-parameters that specify the convolution kernels including k, kernel size, f, number of filters.

$$\begin{aligned} i_t &= \sigma(w_{ix} * x_t + w_{ih} * h_t + b_i) \\ f_t &= \sigma(w_{fx} * x_t + w_{fh} * h_t + b_f) \\ o_t &= \sigma(w_{ox} * x_t + w_{oh} * h_t + b_o) \\ \tilde{c} &= \sigma_h(w_{cx} x_t + w_{ch} h_t + b_c) \\ c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\ h_t &= o_t \circ \sigma_h(c_t) \end{aligned} \tag{2.2}$$

The $*$ denotes the convolution operation here and all other variables mean the same qualitatively but are of different dimensions as follows:

- x is a $5-$dimensional input of the form (samples, timesteps, channels, height, width) where x_t is a single time step.
- h_t and c_t are similarily 4D matrices which are initialized as zero matices by default.
- All activation gates, memory cell candidates and bias terms are therefore elements of $\mathbb{R}^{j_1 \times j_3 \times j_4 \times j_5}$ where j_i are the length of vectors for each dimension of the 5D input vector and $i \in \{\#\text{samples}, \#\text{channels}, \#\text{height}, \#\text{width}\}$.
- w are convolution filters of size $(k,\ k)$ for each input channel and each output number of filters specified.

An important difference between the ConvLSTM2D implementation in current Tensorflow 2.4 library (Abadi *et al.* (2016)) and the equations described in the Shi *et al.* (2015) is that the current ConvLSTM2D layer implementation does not have peephole connections. Peephole connections are additional terms to include explicit c_{t-1} dependence in each gate and memory cell activation equations. With peephole connections the cell transformations as described in Shi *et al.* (2015) are as follows:

$$\begin{aligned} i_t &= \sigma(w_{ix} * x_t + w_{ih} * h_t + v_i \circ c_{t-1} + b_i) \\ f_t &= \sigma(w_{fx} * x_t + w_{fh} * h_t + v_f \circ c_{t-1} + b_f) \\ o_t &= \sigma(w_{ox} * x_t + w_{oh} * h_t + v_o \circ c_{t-1} + b_o) \\ \tilde{c} &= \sigma_h(w_{cx} x_t + w_{ch} h_t + b_c) \\ c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\ h_t &= o_t \circ \sigma_h(c_t) \end{aligned} \tag{2.3}$$

While writing this thesis and to the best of the author's knowledge there has been no implementation of peephole ConvLSTM2D in Tensorflow. Though it would be an impressive addition, but it also increases the number of parameters by a factor of two in the already heavily parametrized class of ConvLSTM2D and there hasn't been enough basic research to prove if that will help with next-frame prediction in a substantial manner. In fact as expanded upon future research direction (6.3), this can possible lead to better results of ConvLSTMs than conv2D but needs to be rigorously compared with certain normalizing parameters.

2.1.4 Activations

Activations are the layers that apply a non-linear transformation to the inputs. It is an element-wise operation on the inputs and are fundamental to the ability of deep neural networks to approximate non-linear functions. The most used activations functions are:

- Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$
- Hyperbolic Tangent: $\sigma_h(x) = \frac{2}{1+e^{-2x}} - 1$
- Rectified Linear Unit: $\sigma_r(x) = \max(0, x)$
- Leaky ReLU: $\sigma_{lr}(x) = \max(x, \alpha x)$ with default $\alpha = 0.3$

Traditionally, sigmoid and hyperbolic tangent were the most prevalent activations used in various neural networks. Recently, Rectified Linear Unit has been found to help convergence of loss much faster than other two.

2.2 Loss Function

Loss function calculates the metric that will be minimized over training epochs. The neural network model may contain high-dimensional predicted values and ground truth matrices but loss is a scalar function over predictions and ground truth and if multiple losses are provided as an array, default action is to average them unless an array of loss weights are specified for each loss.

An important consideration while defining the loss metric is to consider what characteristics are to minimized and whether the neural network may perform an unintended optimization which was not explicitly mentioned to not be carried out. For example, two main metrics are used to measure the performance of time series weather predictions, RMSE and ACC (Anomaly Correlation Coefficient). If we let the model only increase ACC (as the model by default reduces the scalar calculated by the loss function, we can feed it $l' = 1-$ ACC), then it will reduce pattern correlation among the predictions and ground truth but not account for large deviations of the magnitude of difference between the predicted and ground truth. Therefore MSE or RMSE are chosen to be loss functions in our model training which will surely drive the model to learn better transformation than just correlation metrics. The choice of loss metric characterizes what we want the model to minimise; in other words, it drives the model to learn a particular skill.

2.3 Optimizers

Optimizers determine how a specific step of gradient-descent is to be carried out based on the loss value from the loss function. All optimizers are variants of some kind of stochastic gradient descent. It would be important to note here that as opposed to classical supervised machine learning where a certain minima of a loss function is calculated based on all input data, gradient-descent based learning goes through several loops over the input data to learn the trainable parameters. In the quotation below from LeCun *et al.* (2015) is the essence of gradient-based learning that powers NNs towards optimization.

2.4 Metrics

Apart from loss, we can look for certain other evaluation metrics to be calculated to check if the model is improving in an intended way or not improving in an unindented way at the end of each epoch such as accuracy, mean absolute error and RMSE/MSE, Pearson Correlation Coeeficient, ACC etc.

Chapter 3

Review of Models and Data

3.1 Operational Physics-based Models

Weather forecasting methods that are currently used include numerical weather prediction (NWP) models based on General Circulation Models (GCMs). These are a complex set of equations describing the physics of atmospheric state dynamics. These consist of partial differential equations that are solved numerically subject to thermodynamic and the existing physical parametrization of known atmospheric processes.

Over time the predictions using NWP models have become better (Bauer *et al.* (2015)) and all operational forecasts are done using these methods. However, analytical solutions do not exist for fluid dynamical models on spherical domains especially for Navier-Stokes equations and additional complexities arise at the pole as a point of singularity.

Precipitation being another difficult to predict variable has a slew of other methods such as nowcasting extrapolation methods such as COTREC. Although the extrapolation methods are effective for reliably predicting radar echo, they lack a strong nonlinear mapping ability to extract inherent features from large amounts of data Chen *et al.* (2020).

3.2 Neural Network-based Models

The problem statement of weather forecasting which requires some previous knowledge of atmospheric state to predict the next n days/hours in future (in forecasting/nowcasting respectively) allows for being amenable to a supervised machine learning. Although a brute force application of regression using machine learning was considered to be of not much help, the recent development in computer vision techniques to extract spatial patterns using CNNs and the success of Natural Language Processing using RNNs have opened up possibility of considering this approach again.

It is only recently that they has been research in trying to emulate GCMs or produce end-to-end trainable models for purely data-driven predictions using machine learning, specifically neural networks, for such problems. There are three broad classification of models based on the method of prediction: direct, iterative and continuous.

1. Direct: For each successive forecast time step (be it hourly/daily) a separate model is trained. Therefore the in supervised training samples (X, Y), Y is changed accordingly for different time step forecasting model.

2. Iterative: The model is fundamentally different from direct models in the way that it represents *take t previous time steps, and predict n future time steps* and can be used iteratively to predict $2n, 3n$ and even more forecast lead times. Naturally such models are more difficult to train and show lesser skill compared to direct models.

3. Continuous: This type of model takes time as an additional input and a single model is used to generate all forecast lead times. (Sønderby *et al.* (2020) and Rasp and Thuerey (2021))

3.3 Comparison of Previous Work

Following are the 5 major previous works on modeling end-to-end data-driven forecasting of large-scale atmospheric dynamics in medium range (5 lead days extending up-to next 2 weeks) using neural networks. They have been compared extensively along various parameters from the source and type of data used to the minute implementation details. Emphasis has been given on the methodology of sequence prediction, architecture used, and metrics that they considered. The relevant observation are compiled in section 3.4

3.3.1 Dueben and Bauer (2018)

- Aim: To create a toy weather model for global medium range forecasts
- Data Variables: 500-hPa geopotential height (Z500) only and some models with 2-meter temperature alongside Z500. Additionally two time fields were added as an input variable to represent daily cycle (growing linearly from midnight to midnight of the following day) and yearly cycle (with correct representation of leap years)
- Data Frequency: Hourly
- Data Resolution: 6° latitude-longitude grid with 31×60 points.
- Data Normalization: Max-Min normalization, scaled to -1 and $+1$.
- I/O Sequence: Input consisted of one hour of regridded geopotential (Z500) into 60×30 resolution and output consisted of the incremental difference between the next hour's Z500.
- Train/Validation Set: 67,200 total hours in the period between 1 January 2010 and 31 August 2017. First 7 years were used for training, with 20% split for validation.
- Test Set: 10 forecasts distributed between March 2017 and February 2018.
- Models: Two iterative variants - Fully connected neural networks and convolution-filter like local networks.
- Architectures: First one had a Sequential of 4 layers of DNN with 1862 units each (1862 being chosen as it was their input layer width). The local network contained a $N \times N$ stencil layer followed by 4 layers of DNN.
- Loss: Mean Absolute Error

- Optimizer: Stochastic Gradient Descent
- Activation: Hyperbolic Tangent 2.1.4
- Epochs: 200 (fixed)
- Evaluation Metrics: Mean Absolute Error
- Results: Persistence-level performance not better than physical baselines
- Note: Instead of predicting the next hour's Z500, they train the network to predict the difference between the next hour and current hour, which can be seen to be directly motivated as being the right hand-side of differential equations if the time step is one hour. Therefore they use a third-order AdamsâĂŞBashforth explicit time-stepping scheme for produce iterative forecasts.

3.3.2 Scher (2018)

This is

- Aim: To train a network to emulate a simplified GCM model (PUMA) by feeding it supervised form of its inputs and outputs.
- Data Variables: 4 variables consisting of horizontal and meridional wind, temperature, and 500−hPa geopotential height interpolated for 10 pressure level from 10 model levels and converted to daily frequency from 45 min time steps. Results tested on 500−hPa geopotential height.
- Data Frequency: Daily
- Data Resolution: 32×64
- Data Normalization: All data was normalized to mean 0 and standard deviation of 1 right from the input layer and though all subsequent layers.
- I/O Sequence: Input consisted of daily atmospheric state input as fed into the PUMA model.
- Train/Validation Set: 100 years of training data, 20 years for validation set
- Test Set: 30 years
- Models: Convolutional encoder-decoder with separate networks for each day up-to 14 days (direct models).
- Architectures: Sequential layers consisting of an encoder block, reshaping layer and a decoder block as shown in 3.1. Both encoding and decoding block are similar and have 2 Conv2D layers with hyperparameters of 32 filters and (6×6) kernel size. Encoder has a MaxPooling layer with pool size of 2 and decoder therefore has a UpSampling layer of size 2 to ensure that the output dimensions match the input dimensions. The reshaping layer is a Dense layer with 500 units.

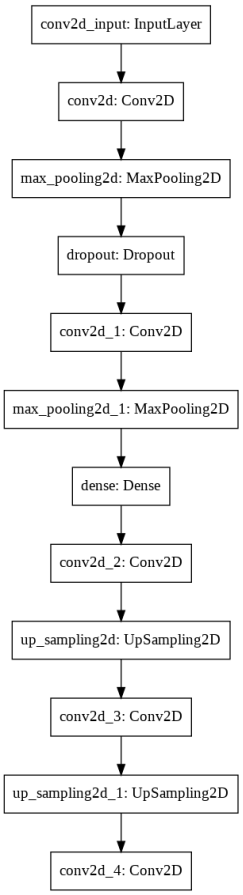


Figure 3.1: Regenerated model used in Scher (2018).

- Loss: Mean Squared Error
- Optimizer: Adam with 0.0001 learning rate (Kingma and Ba (2014))
- Activation: Rectified Linear Unit 2.1.4
- Epochs: 10 (minimum) with EarlyStopping with the following parameters:
  - Metric Monitored: Validation Loss
  - MinDelta: 0 (meaning)
  - Patience: 5 epochs
- Evaluation Metrics: RMSE, normal correlation and ACC
- Results: Long-term stable results with high skill
- Note: This model nicely emulated the a highly idealized GCM Model devoid of seasonal variability (eternal Northern Hemispheric winter) and therefore points to possible expansion to more complex external forcing such as diurnal and seasonal cycle.

3.3.3 Scher and Messori (2019)

This paper was a much more complicated study building upon the previous work. A total of 7 models were trained to compare against each other with varying degrees of complexity in both spatial resolution and physical

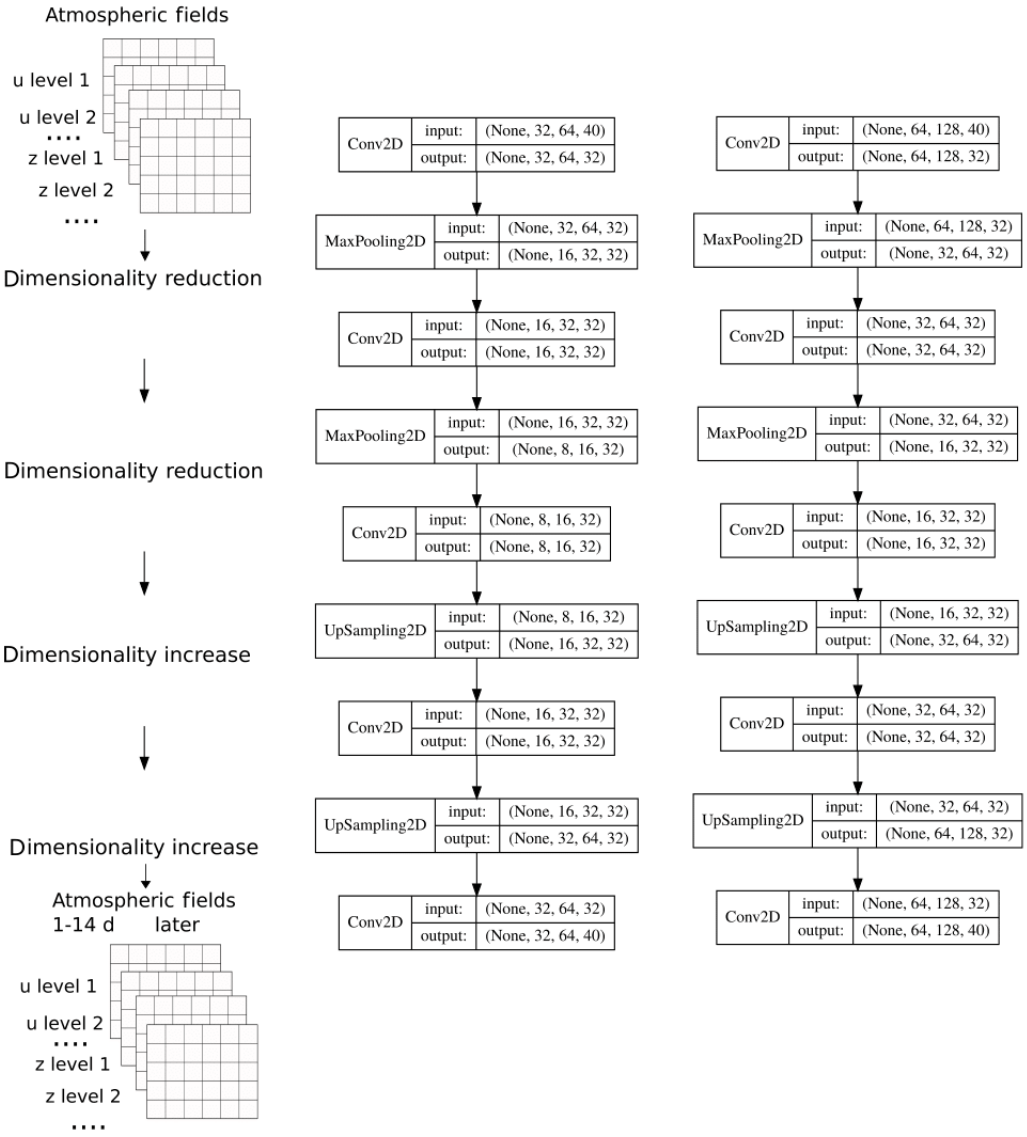
- Aim: To emulate PLASIM (intermediate-complexity GCM) and PUMA (simplified GCM) Model
- Data Variables:
  - PUMA Model: 4 variables (u, v, t, z) as used in the full state used in Scher (2018)
  - PLASIM Model: 7 variables consisting of 4 variables of PUMA model and 3 additional fields related to the hydrological cycle (relative humidity, cloud liquid water content and cloud cover). Though for comparison of these two models, the additional fields have not been used, a test result was presented in the paper including the additional fields.
- Data Frequency: Hourly
- Data Resolution: Two different models for each PLASIM and PUMA were trained with resolution of T21 (32×64) and T42 (64×128) grids.
- Data Normalization: Standardization to zero mean and standard deviation of 1.
- I/O Sequence: Input consisted of daily atmospheric state input as fed into the PLASIM and PUMA models.
- Train/Validation Set: Training consisted of upto 800 years. The last 10% samples of the training data are used for validation.

- Test Set: First 30 years were used as test set which were not used in training or validation.
- Models: Iterative and direct models using purely convolutional layers.
- Architectures: Same as Scher (2018)
- Loss: Mean Squared Error
- Optimizer: Adam with 0.0001 learning rate.
- Activation: Rectified Linear Unit 2.1.4
- Epochs: Limited by maximum number of epochs at 100 and otherwise determined by EarlyStopping with the following parameters:
  - Metric Monitored: Validation Loss
  - MinDelta: 0
  - Patience: 5 epochs
- Evaluation Metrics: RMSE and ACC
- Results: Good skill for short lead times but unstable predictions on long time scales (climate runs)
- Note: Even though the complexity of the GCM was increased the same architecture as in Scher (2018) proved to able to handle these dynamics in the short range.

3.3.4 Weyn *et al.* (2019)

- Aim: To build a end-to-end NN-based forecasting model to predict two atmospheric variables on restricted Northern Hemisphere grid from reanalysis data as opposed to GCm emulations done before.
- Data Variables: Z500 and geopotential 700-300 hPa thickness
- Data Frequency: 6-hourly
- Data Resolution: 2.5° horizontal resolution cropped to Northern Hemisphere.
- Data Normalization: Standardization as before by subtracting Northern Hemisphere climatological mean, followed by division of mean climatological standard deviation.
- I/O Sequence: 2 input time steps separated by 6-hours, and 2 output time steps separated by 6-hours.
- Train/Validation Split: 1979âĂŞ2002 for model training (24 years) and 2003âĂŞ2006 for model validation (4 years).
- Test Set: 2007âĂŞ2010 were set aside for the test set and used in final model performance evaluation.
- Models: Iterative convolutional encoder-decoder type with 2 additional variants using LSTM-based layers.

Figure 3.2: Model architecture used in Scher and Messori (2019)



- Architectures: Similar to Scher (2018) with additional 2 convolutional layers with zero padding to conserve dimensions of input and output. Some variants had one first layer of ConvLSTM2D and some variants last layer of custom versions of LocallyConnected2D layer class for latitudinally dependent output layer convolutional filters.
- Loss: Mean Squared Error
- Optimizer: Adam with 0.001 learning rate.
- Activation: Hyperbolic Tangent Function 2.1.4
- Epochs: Minimum of 200 epochs conditioned on early stopping with parameters as follows:
  - Metric Monitored: Validation Loss
  - MinDelta: 0
  - Patience: 50 epochs

- Evaluation Metrics: RMSE and ACC
- Results: One of their variants outperform a climatological benchmark for root-mean-squared error (RMSE) for geopotential height at 500 hPa out for 5 days.
- Note: It is assumed that the climatological shifts in geopotential heights are insignificant enough to consider the data to be from the same distribution.

3.3.5 Weyn *et al.* (2020)

- Aim: To extend the previous model upto several variables and global-scale forecasts by converting regular latitude-longitude data into cubed-sphere format and identify characteristics for long-term stable climate-like simulations.
- Data Variables: 4 variables consisting of Z500, Z1000, geopotential thickness as before of $700 - 300$ and 2-meter temperature.
- Data Frequency: 3-hourly
- Data Resolution: Global 2° regular lat-lon data converted to cubed sphere with each face of resolution 48×48.
- Data Normalization: Standardization as before.
- I/O Sequence: 2 input time steps separated by 6 hours, and total of 4 output time steps separated by 6 hours with the last two being the iterated output of the first two outputs.
- Train/Validation Set: Data from 1979 to 2012 was used for training, and 2013-2016 was set for validation.
- Test Set: 2017-2018
- Model: Iterative forecasting as well as iteratively trained purely-CNN based model.
- Architecture: 11 layers of custom neural network layer class called CubeSphere-Conv2D with filter sizes in increasing and decreasing and fixed kernel size of (3×3)
- Loss: Mean Squared Error
- Optimizer: Adam with 0.001 learning rate.
- Activation: Custom Leaky Rectified Linear Unit which is said to help in asymptotic predictions over long time scales
- Epochs: Minimum of 100 epochs conditioned on early stopping with parameters as follows:
  - Metric Monitored: Validation Loss
  - MinDelta: 0
  - Patience: 50 epochs
- Evaluation Metrics: RMSE and ACC
- Results: The best CNN forecast outperforms a climatology benchmark at up to 120 hours of lead time, and appears to correctly asymptote towards persistence forecasts at longer lead times up to 14 days.
- Note: As in Weyn *et al.* (2019) climatological shifts are considered insignificant to current predictions.

3.3.6 Rasp and Thuerey (2021)

- Aim: To develop a DataâĂŘDriven MediumâĂŘRange Weather Prediction With a ResNet architecture pre-trained on climate simulations of CMIP Data.
- Data Variables:
  1. Inputs: Geopotential, temperature, zonal and meridional wind and specific humidity at seven vertical levels (50, 250, 500, 600, 700, 850, and 925 hPa), 2âĂŘm temperature, 6âĂŘh accumulated precipitation, the topâĂŘofâĂŘatmosphere incoming solar radiation and three constant fields: the landâĂŘsea mask, orography and the latitude at each grid point
  2. Outputs: 500 hPa geopotential (Z500), 850 hPa temperature (T850), 2âĂŘm temperature (T2M) and 6âĂŘhourly accumulated precipitation (PR)
- Data Frequency: Hourly
- Data Resolution: 5.625° of regular lat-lon grid with 32×64.
- Data Normalization: All fields were normalized by subtracting the mean and dividing by the standard deviation, with the exception of precipitation for which the mean was not subtracted to keep the lower bound at zero. Additionally, we logâĂŘtransform of the precipitation to make the distribution less skewed.
- I/O Sequence: 3 time steps of input variables consisting of 114 channels separated by 6 hours. Output consisted of one time step each for each direct model trained.
- Train/Validation Set: 1979 to 2015 was used for training, 2016 for validation.
- Test Set: 2017-2018
- Models: Fully convolutional ResNet
- Architecture: 19 Residual blocks, each containing conv2D, LeakyReLU, Batch normalization and Dropout (0.1). Each conv2D layer has 128 filters with kernel size of 3, except the first one which has 7. Weight decay of 1×10^{-5} is used in layers.
- Loss: Latitude−weighted mean squared error
- Activations: Leaky ReLU with default alpha of 0.3
- Optimizer: Adam with initial learning rate of 10^{-5} and it was decreased twice by a factor of five when the validation loss has not decreased for two epochs. (using ReduceLROnPlateau)
- Epochs: Conditioned on early stopping with parameters as follows:
  - Metric Monitored: Validation Loss
  - MinDelta: 0
  - Patience: 5 epochs
- Evaluation Metrics: RMSE and ACC

- Results: The resulting forecasts outperform previous submissions to WeatherBench and are comparable in skill to a physical baseline at similar resolution. Their study also considers saliency maps to check for physically reasonable correlations and difference of model performance on scaling resolution.
- Note: As in Weyn *et al.* (2019) climatological shifts are considered insignificant to current predictions.

3.4 Relevant Observations

Given these are studies, let us observe the development of techniques and questions explored in them. We notice a few key things from the previous studies:

- Most studies have used Z500 as a standard forecasting variable as it does not depend on near-surface condition as opposed to 2-meter temperature or surface pressure and it also does not feature strong local gradients as present in humidity or precipitation fields.
- Most models have used purely convolution architectures such as multiple layers of Conv2D as their basic neural network layer where various time steps are concatenated in the channel axis. If more than one variable or pressure level is used they are also concatenated in channel axis.
- Padding was first used in preserving dimension in Weyn *et al.* (2019) in the form of zero padding. Later we see more natural extension of padding to cubed-sphere padding in Weyn *et al.* (2020).
- Choice of loss functions have changed from mean absolute error to latitude weighted mean squared error. MAE was easily applicable for a positive only field such as geopotential height in a single variable model in Dueben and Bauer (2018), whereas lat-weighted mean squared error makes most sense in evaluating output of models which are trained on data projected on regular lat-lon grid to account for more distorted polar points.
- Activation functions preferences have moved from sigmoid to hyperbolic tangent to ReLU to LeakyReLU.
- Early Stopping using validation loss as a metric for model performance is used in all later models.
- ERA 5 Reanalysis Dataset has been used for all end-to-end NN-based weather forecasting.

3.5 Conv2D architectures

The Conv2D layer is the basic architectural layer used in all the deep learning based forecasting methods we have seen till now. This is primarily motivated by its ability to capture spatial features. Here, we explain how the inputs are transformed in a given conv2D layer. As visible in the figure 3.3, for each filter convolution operation is performed with each image presented in the channel axis with a different kernel. Therefore

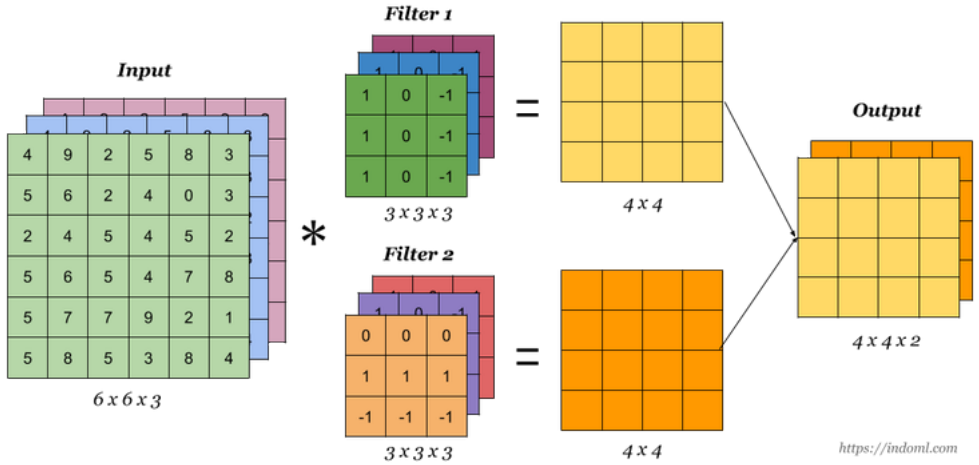
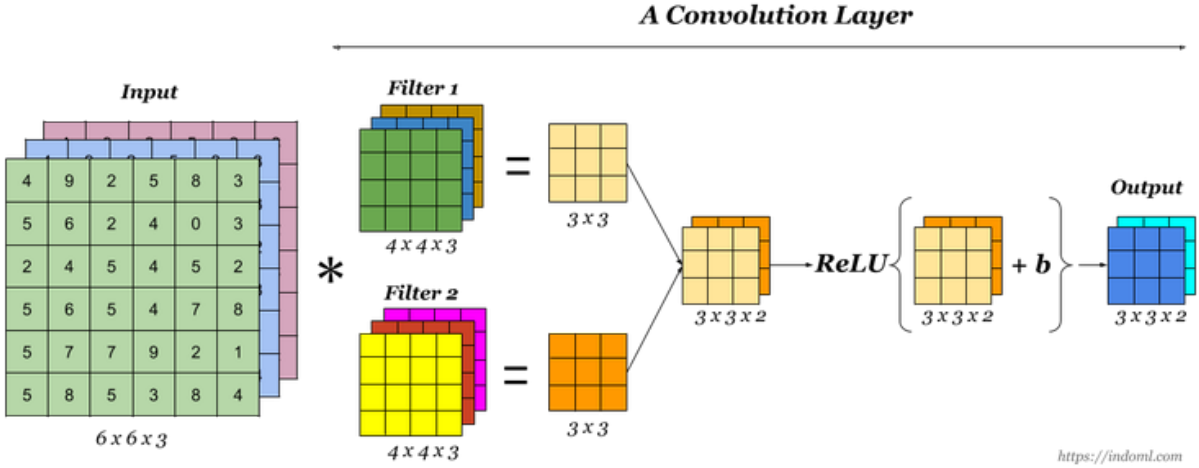


Figure 3.3: Multiple filters. Image used with permission from `Benny Prijono; indoml.com`

each image concatenated in the channel axis is treated independently and transformed according to the convolution kernel learnt using back-propagation. Most convolution layers are followed by an activation either just after it or through in-layer integration by supplying an activation attribute and the schematic representation for the same is as follows:

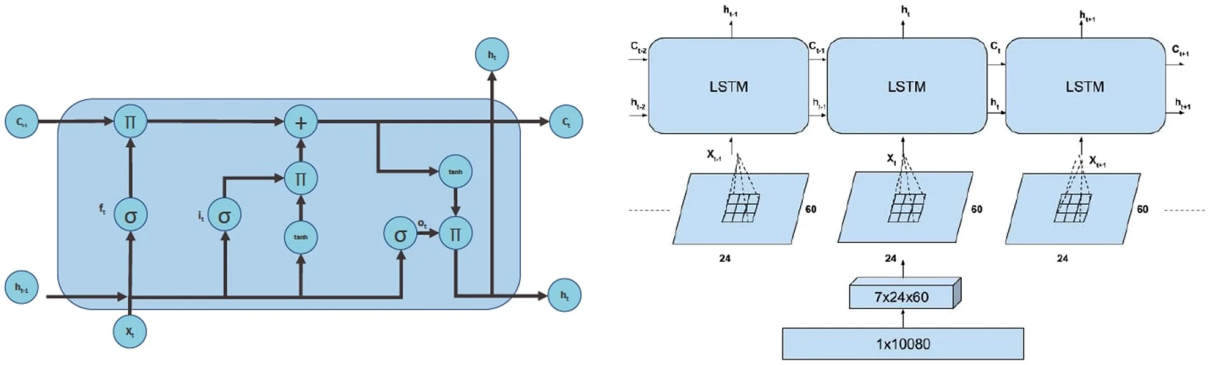


3.6 ConvLSTM2D architectures

This layer is a convolutional-recurrent layer which performs the convolution operation not just on multiple channels but successively on different time steps. This is the main layer type used in our models. As explained in section 2.1.3 one of the main operations of the layer carries out the exact operation as inside a conv2D layer but instead of treating each time step as a separate channel it repeats the same convolution filter for all time steps of a given channel. A schematic representation of this layer is shown in fig 3.4.

3.7 Data

Two datasets have been used for this work, namely the IMD Dataset and the WeatherBench Dataset. The first one being a regridded format of data from ground stations



(a) LSTM **(b)** Structure ConvLSTM*

Figure 3.4: ConvLSTM2D Schematic (dimension values just for representation); Image from Rahman and Adjeroh (2019)

across India while the second one is satellite observation processed through current reanalysis atmospheric models to capture the best possible state of atmosphere. In this chapter, we first explain the general restructuring required to convert the daily/hourly data available for a local/global region into supervised training form.

Apart from satellite observations, weather forecasting models also use data from a slew of other sources such as radar echo, ground-based interpolation methods, etc.

3.7.1 Supervised Restructuring

Depending on the architecture of the model separate restructuring is required for different models. For example, the models can be of the type:

1. Given the last 5 days of precipitation fields, predict the output for the next 5 days (in meteorological parlance, lead time of 5 consecutive days)

2. Given 500−hPa geopotential height over the last 2 hours, predict it's variation in the next hour.

In order to prepare a supervised input data of the required type, the available data has to be converted into a different shape. This involves restructuring the data which usually leads to an increase in the size of the actual input data as multiple copies of the data exist in the restructured supervised format. This increases the amount of data to be loaded in the RAM by the number input and output time steps required by the model. For the IMD Dataset this is not a major limitation of the RAM usage but, for the WeatherBench Dataset it quickly becomes a bottleneck.

This is an usual problem in training deep neural networks with huge datasets and given in our case where we have a single data file to restructure from, we can load the input file once as use a DataGenerator class to feed restructured data on-the-fly to the model for training and evaluating. For the specific case of our project a custom DataGenerator class was implemented, the details of which are in section 4.2.1.

3.7.2 IMD Dataset

The Indian Meteorological Department (IMD) Dataset used here is from Pai *et al.* (2014), which is a regridded re-assimilated format of observations from IMD ground stations across the country. The IMD dataset is from 1 Jan 1975 to 31st Dec 2004. The frequency of the data is daily with each observation from various scattered stations over India into a regular latitude-longitude of resolution 128×135. Given the period, there are 10,950 samples and therefore the raw nature of the data is available as a 3D array of type (days, lat, lon). The size of the dataset is not exceeding one gigabyte.

3.7.3 WeatherBench Dataset

Reanalysis data are the state of an atmospheric model with a continuous assimilation of observations to generate the best possible picture of the global atmosphere at a given time. One of the biggest advantage of reanalysis dataset over ground-based interpolation for training is that data is available for each grid point at each time step and that the data is consistent over the entire data window. Whereas due to difference in equipment and other human factors, IMD Dataset can have much more variance and noise which may not be structurally distinguishable.

Regular Latitude-Longitude Format

The WeatherBench data as provided is a regridded format of the 0.25° spatial resolution ERA5 data as available from CDS into 1.4625, 2.8 and 5.625 degrees. It is in a regular latitude-longitude format.

Cubed Sphere Remap Format

As used in Weyn *et al.* (2020), a volumeâĂŘconservative cubedâĂŘsphere mapping is used to convert the regular lat-lon data. The transformation is parametrized by the choice of the resolution of each cube face which is the number of points in each length of each face of the resulting cubed sphere. One way to decide what resolution to map to is to conserve the total number of grid points in both formats and pick the closest integer value of the parameter that accomplishes this. For the case of converting 5.625° spatial resolution (i.e. a lat-lon grid of 32×64 points), we used a resolution value of (16×16) for each of the six faces whereas the closest integer to equal number of grid points is 18. This was because GPU computation are also optimized for inputs and batch sizes of computations which are powers of 2.

One could also try to maintain the equatorial spacing to be equal to the regular lat-lon spacing. Calculating this way, the total number of points in the longitudinal direction (64, in our case) would be divided by 4 equatorial faces and hence lead to easier integer choice of cubed sphere resolution. Simple mathematical calculation will show that both these methods will give numbers close to each other. Maintaining equal spacing will lead to slightly fewer total grid points but not low enough to loose spatial complexity.

Chapter 4

Methods

One of the main aim of this project was also the development of custom neural network classes by building on existing architectures. While moving from the IMD Dataset to working on the WeatherBench Dataset, certain classes and functions needed to be constructed for preprocessing and optimization. Even in the WeatherBench dataset, separate focus had to be given in developing classes and functions for both regular lat-lon format and cubed sphere remapped format of data. The WeatherBench repository does comes with certain helper functions but they are constrained for usage on regular lat-lon data format with single time step input-output models, therefore we develop more flexible tools for varied dataset formats and model topologies. All implementation are carried out using open source deep learning libraries such as Keras (Chollet *et al.* (2015)) with Tensorflow (Abadi *et al.* (2015)) backend and open source xarray (Hoyer and Hamman (2017)) for managing netCDF data pipelines. A detailed discussion of their working can be found in the github documentation which will be uploaded shortly, links in 7. Here we list the most important ones and a summary of their implementation.

4.1 Model Specifications

We have used two basic architectural variants. ConvLSTM2D and CubeSphereConvLSTM2D. With the first architecture, ConvLSTM2D we compare across two different forecasting variants, direct and iterative, on IMD Rainfall Data, while with the more-complicated second variant we do a proof of concept supervised geopotential prediction. Following are the specifications of the models used in our work.

4.1.1 ConvLSTM-based Model

- Data Variables: IMD Precipitation value in mm. (single variable input-output)
- Data Frequency: Daily
- Data Resolution: 0.25° spatial resolution restricted the the political boundary of India with `nan` values outside filling a 128×135 grid.
- Data Normalization: As carried in George (2020). Divided by max observed rainfall to transform data from $(0, \max) \to (0, 1)$. Followed by exponentiation to transform data from $(0, 1) \to (1, e^{\alpha \max})$. Prescribing zero value to `nan` value points outside India in the lat-lon grid. α was chosen to make sure $e^{\alpha \max}$ was close to max. Therefore finally the range is $\{0\} \cup (1, e^{\alpha \max})$.

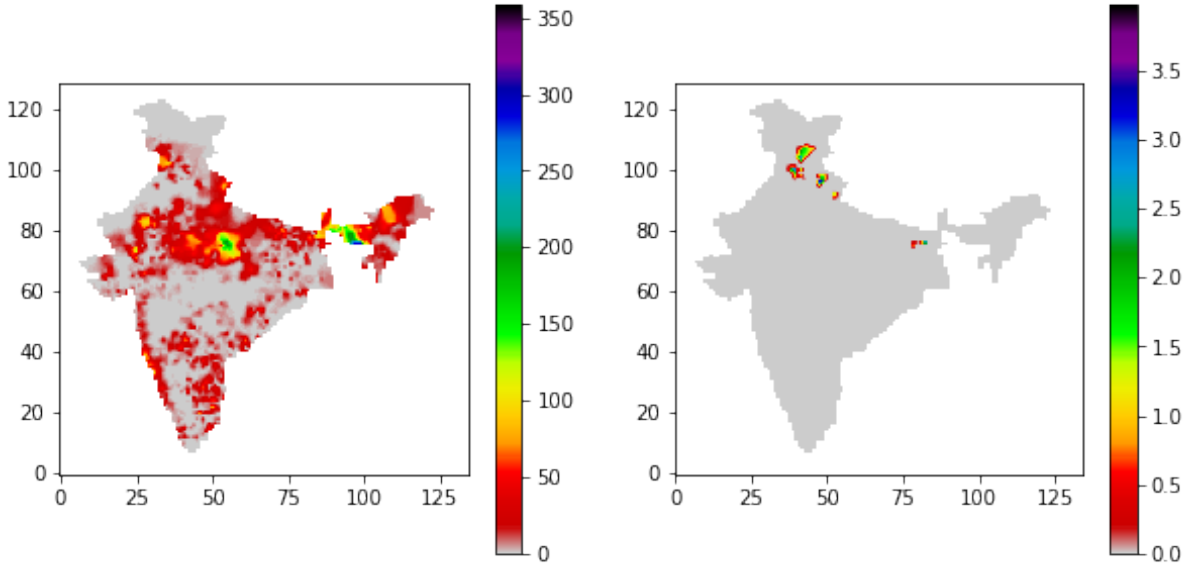


Figure 4.1: IMD Input Data ranging from 3.5 mm daily max rainfall to 350 mm daily max rainfall.

- I/O Sequence: 5 input time steps corresponding to 5 input days, and 1 successive day as lead day for output.
- Train/Validation Split: 1 Jan 1975 to 31st Dec 2004 for model training and validation and testing (30 years). Out of which first 8000 samples were split 90/10 for training and validation.
- Test Set: Leaving the first 8000 samples, the rest were used as the test set and used in final model performance evaluation.
- Models: Direct forecasting using convolutional LSTM-based layers, followed by Conv2D layer for 2D output.
- Architectures: As shown in figure 4.2
- Loss: Mean Squared Error
- Optimizer: Adam with 0.0001 learning rate.
- Activation: Hyperbolic Tangent Function 2.1.4
- Epochs: Fixed 500 epochs
- Evaluation Metrics: Pearson Correlation Coefficient
- Results: Section 5
- Note: It is assumed that the climatological shifts in rainfall are insignificant enough to consider the data to be from the same distribution.

Foe the direct case, five separate models were trained to take the input defined in the description above to predict the rainfall over the next 5 days, where each model was trained on the same predictors but different targets depending on which day is to be predicted. We will refer to the first predicted day as lead day 1 and so on and the model used to predict lead day 1 as lead day 1 model.

This methodology was replaced in the iterative forecasting by using the lead day 1 model to predict lead day 2 results iteratively. This was done by concatenating the input

Model: “convLSTM-based network architecture”
Note: None corresponds to variable number of samples and atmospheric measurables

Layer (type)	Output Shape	Parameters #
conv_lstm_2d_1 (ConvLSTM2D)	(None, None, 4, 129, 135)	736
conv_lstm_2d_2 (ConvLSTM2D)	(None, None, 8, 129, 135)	3488
conv_lstm_2d_3 (ConvLSTM2D)	(None, None, 8, 129, 135)	4640
conv_lstm_2d_4 (ConvLSTM2D)	(None, None, 16, 129, 135	13888
conv_lstm_2d_5 (ConvLSTM2D)	(None, 16, 129, 135)	18496
conv2d_1 (Conv2D)	(None, 15, 129, 135)	2175
conv2d_2 (Conv2D)	(None, 1, 129, 135)	136

Trainable params: 43,559

Figure 4.2: ConvLSTM2D model architecture

of the lead day 1 model with the prediction of lead day 1 (and removing the first day of the input to maintain a 5-day input size). This was fed to the lead day 1 model to output the lead day 2 prediction. Therefore we now have a single model that needs to be improved over hyper-parameter tuning and input data structure changes.

4.1.2 CubeSphereConvLSTM2D-based Model

- Data Variables: WeatherBench 500-hPa geopotential height (single variable).
- Data Frequency: Hourly
- Data Resolution: 5.625° spatial resolution (32×64 global grid) converted to Cube Sphere Grid with 16×16 points in each face.
- Data Normalization: Standardization as before by subtracting climatological mean, followed by division of mean climatological standard deviation.
- I/O Sequence: 2 input time steps and 2 output time steps for easy iteration.
- Train/Validation Split: 1979-2015 for training, 2016 for validation.
- Test Set: 2017 and 2018 for testing.
- Models: Iterative forecasting using Cube Sphere Convolutional LSTM-based layers, followed by Cube Sphere Conv2D layer for 2D output.
- Architectures: Filter configuration as used in Weyn *et al.* (2020).
- Loss: Mean Squared Error
- Optimizer: Adam with 0.001 learning rate.
- Activation: Leaky ReLU 2.1.4
- Epochs: Minimum of 50 epochs conditioned on early stopping with parameters as follows:

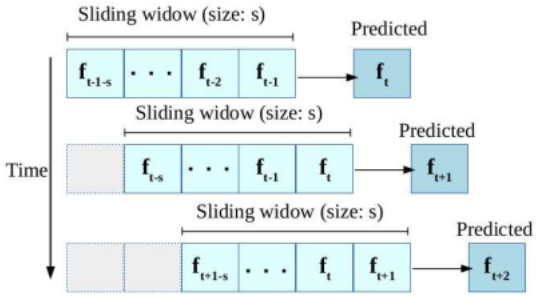


Figure 4.3: Data Generator using tunable parameters of 4 input frames, 1 lead time separation and 1 output frames with 0 spacing. (For example)

  - Metric Monitored: Validation Loss
  - MinDelta: 0
  - Patience: 20 epochs

- Evaluation Metrics: RMSE
- Results: Beats the WeatherBench toy model but falls short of multi-input model of Weyn *et al.* (2020).

4.2 Technical Helper Classes

4.2.1 Data Generator

As mentioned earlier, owing to the size of the WeatherBench dataset it was needed to construct certain custom pipelines that can supply data on-the-fly to the model for optimizing the usage of GPU RAM which would otherwise be used to load the entire dataset. This custom class was built on top of `keras.utils.Sequence` class. The important modifications of the class include tunable attributes of an data array file, input time steps, output frames, spacing and lead time, the Data Generator outputs the restructured format of supervised training samples. A schematic example of the same is shown in 4.3

This is different from the previous approaches used in George (2020) where a function was used to create the supervised data from the original dataset. Given the original daily frequency data was converted to pairs of (X, Y) (where X consisted of time step of 5 days and Y was the successive day), all of this was done on the RAM thus leading to input data size equivalent to 6 copies of the original dataset that would be stored in the GPU RAM. Using Data Generator provided for much more free RAM to train bigger network sizes.

4.2.2 Iterative Forecasting

Previous work by George (2020) consisted of separate direct models trained to predict lead time of one through five days ahead. All the models had the same underlying architecture and only differed in the output lead time predicted.

The first part of this work consists of training an alternate iterative model and using

to it to forecast the next 5 lead days iteratively using the first predicted output and the last 4 ground truth days and so on. Once again, the iterative forecasting function in WeatherBench feeds the output of the single input-output model into itself. But for model which take 5, predict 1 or the similar kind required a separate iterator function.

4.2.3 Cube Sphere Remapping

The inspiration for using Cube Sphere Remapped format of data comes from the first work of this type in Weyn *et al.* (2020). As the primary operation performed on the data is convolution operation it demands careful consideration as to what format of data is best suited for such operations. The regular latitude-longitude data is a 2D array suited for convolution. But the convolution operation by default gives equal preference to each grid point whereas the lat-lon grip is inherently distorted as 3D $\rightarrow$ 2D projections are concerned. Therefore conversion of global lat-lon data into a volume conservative mapping into planar cube faces might improve feature extraction over lat-lon data.

4.3 Model Methods

We first now describe the general outline of the model training and testing workflow.

1. The regular lat-lon data is bunched together using `cdo` utilities into training, validation and testing splits.

2. The regular lat-lon data is converted to cubed-sphere data using the `CubeSphereRemap` class from the `DLWP-CS.remap` module of Weyn *et al.* (2020).

3. The Cube Sphere Data is then assembled into a data pipeline via loading through xarray and restructuring using custom DataGenerator.

4. A Functional API model is created with different architectures. Loss and optimizers are initialized.

5. If iterative training is used, separate functions is implemented to feed-back outputs to the model and concatenate the multiple outputs.

6. The model is trained on the fly using Data Generator with callbacks initialized for stopping when a particular metric stops improving and saving the model state after each epoch.

7. The trained model is them used to iteratively forecast on the testing set and calculate evaluation metrics like RMSE and PCC.

4.3.1 CubeSphereConvLSTM2D

In order to use existing ConvLSTM2D layer class for the cube sphere remapped data format, it was required to construct a new layer class from scratch. As ConvLSTM itself is a complex class, it has multiple inheritances. Given it is primarily a RNN class, it consists of a `cell` and a `iterator` class which preforms the operations of the cell on multiple time steps of the input provided. For example, consider the basic LSTM layer class. The

generic RNN iterator layer is called `keras.layers.RNN` and the specific cell for LSTM is called `LSTMCell`. Hence an LSTM implementation looks like `RNN(LSTMCell(n_units))`.

Therefore the first step in creating custom LSTM layers capable of convolutional cellular operations is to create a custom cell and a custom iterator. The generic iterator will not work as the tensors operations outside the cell are no longer of the same dimensionality as generic RNN iterations. This is exactly what the default `ConvLSTM2D` class is built on. The `ConvLSTM2D` layer class inherits from the `ConvRNN` class the iterator functionality and initializes within itself the `ConvLSTM2DCell` instance for cell construction.

Similarly, in order to make CubeSphereConvLSTM2D, we needed to develop the cell, namely `CubeSphereConvLSTM2DCell` and the iterator `CubeSphereConvRNN` first. Then these are wrapped under `CubeSphereConvLSTM2D` with the incorporation of `CubeSpherePadding` (described in next section) to integrate all the required functionality.

4.3.2 Natural Padding for 5D Input

Apart from being suited for planar convolutions, cubed sphere format of data helps to provide natural padding required tp preserve input-output dimension in convolution operations as described in section 2.1.1. The only ambiguity remains for the corner points which are filled with equatorial faces. While the regular lat-lon data can be naturally padded in the longitudinal direction, the latitude has to be zero padded. Developing on the natural padding extension of `CubeSpherePadding` layer of Weyn *et al.* (2020), we developed similar layer classes for ConvLSTM inputs and integrated them inside the layer construction so that they do not have to be called separately.

4.3.3 Iterative Training

Iterative training was first used in Weyn *et al.* (2020). Usually, as described in section 3.7.1, a certain fixed number of input-output steps are fed to the model. In iterative training, during one training cycle, the outputs are fed back to the model in order to predict twice the number of output steps so that the loss is calculated on multiple output time steps rather than a single one. This way the model is trained on to predict future days with not only the past ground truth but as well as past predictions of itself. Iterative prediction has been found to be one of the main reasons for improvement from Weyn *et al.* (2019) to Weyn *et al.* (2020).

4.4 Evaluation Metrics

4.4.1 Baselines

In order to make sense of our second model's performance, we need to have baselines to compare with. Some of the typical baselines used in weather forecasting are climatology and persistence. Climatology refers predicting the mean value over a given duration. For example, weekly climatology assigns the weekly mean value to each day of the 52 calendar week. Another baseline is persistence, which is based on "tomorrowâĂŹs weather is todayâĂŹs weather". For single time step output models, the persistence can be defined

as the last input time step. But for multi input and output, persistence can be defined in more than one way. The most general is to assign one of the input time steps as the initialization time step and consider it for persistence comparison. Here, the baselines are used to compare the results on the WeatherBench data only.

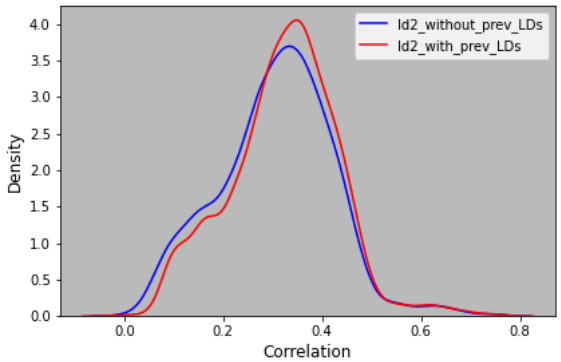
Chapter 5

Results

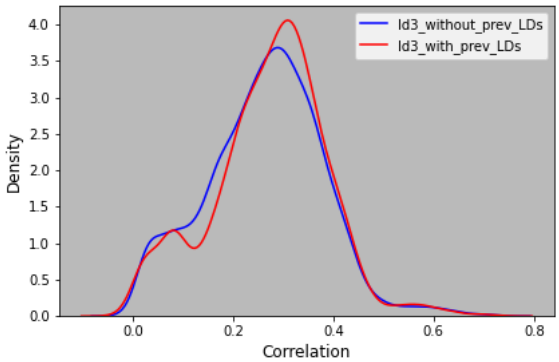
5.1 IMD Iterative Model

We have produced the following results using IMD data after developing iterative forecasting technique as opposed to the direct forecasting in George (2020). The evaluation metric of this model is the Pearson correlation coefficient. The correlation is computed for the time series predictions of the input and output samples at all locations. As the predictions are themselves 2-D spatial images, the correlation coefficient is computed for the corresponding time series for each spatial point across the samples and thus is represented as a correlation matrix of the same shape as the input grid shape. In order to understand improvements of the correlation over various model testing, we compare the respective correlation density distributions of the matrix produced.

Note that while comparing two time-series data, the correlation coefficient captures

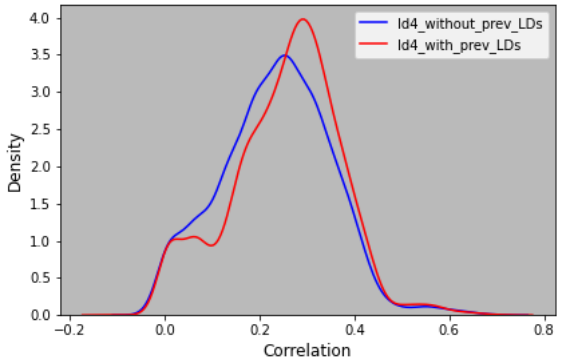


(a) Lead Day 2

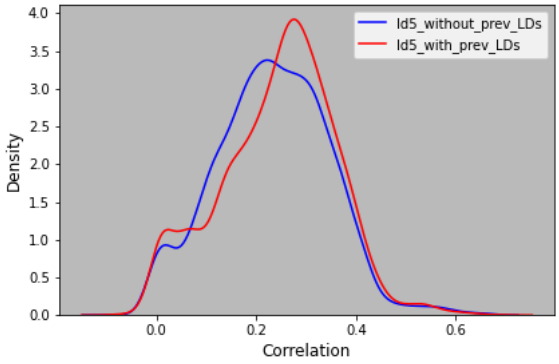


(b) Lead Day 3

the extent of the phase shift of the prediction. Therefore it is used as our primary evaluation metric rather than root mean square error, even though the models were trained to reduce MSE as their loss function 4.1.1. Even if amplitude errors exist, they can be scaled to match the ground truth through statistical corrections accounting for biases. We compared the correlations generated by both methods. The differences in the distribution of pattern correlation over the input grid space after changing the methodology is shown in the figures below. Finally, we also plot the iterative PCC distribution results for corresponding lead days.



(a) Lead Day 4



(b) Lead Day 5

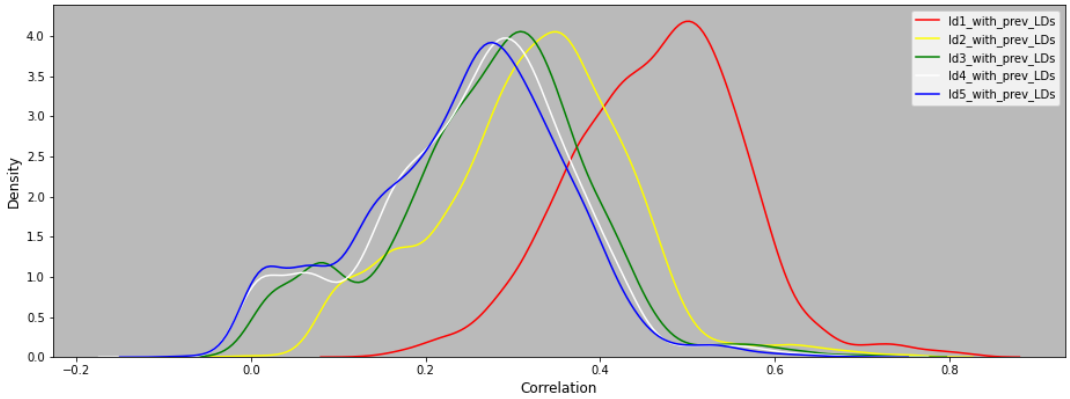
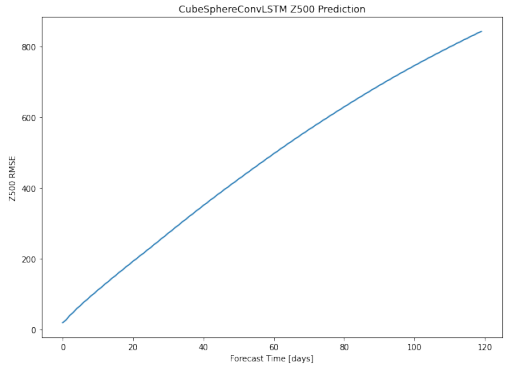


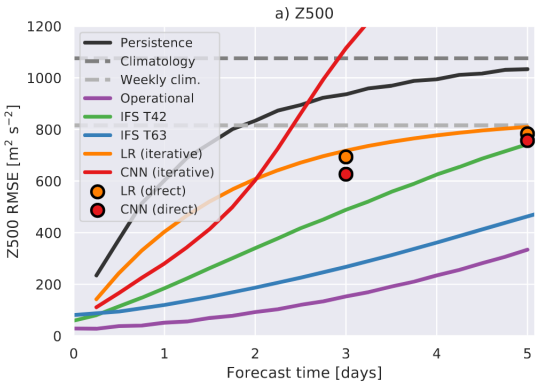
Figure 5.3: Forecast skill differences with increasing lead day

5.2 CubeSphereConvLSTM Iterative Model

Here we present the results of single variable Z500 prediction using CubeSphereConvLSTM Model as compared against the baselines computed in WeatherBench, Rasp *et al.* (2020). The following figure in the left show shows the global mean RMSE of network forecasts at lead times of up to 120 hours (5 days) at each hourly computation. As expected, the skill of the network forecasts decreases monotonically with lead time.



(a) RMSE of Z500 prediction from CubeSphereConvLSTM2D Model



(b) RMSE of other baseline comparisons

Chapter 6

Discussions

6.1 IMD Iterative

We see an improvement in the correlation density distribution after using a particular model iteratively over separate direct models. The nature of improvement is visible from the right shift in the distribution, which signifies an overall increase in the correlation between the prediction and ground truth time series spatially. Visibly, the mean increases. The increase in peak corresponds to a reduction in the variance of correlation values spatially and signifies more robust results. As usual the correlation decreases with lead days as seen in 5.3. Note that the improvement in better for higher lead days signifying a better understanding. This motivates us to consider further improvements such as incorporation of iterative training and more rigorous difference analysis of distributions which will be followed up with this study.

From the previous result, we can also infer certain advantages and limitations of different neural architectures. Usually, direct forecasting supersedes the skills of iterative forecasting. This is clearly seen in Scher and Messori (2019) and Rasp and Thuerey (2021) which are based on convolution-only architectures. We therefore see another advantage of ConvLSTM-based model. This feature can be ascribed to the fact that ConvLSTM layers have an additional representation of the temporal features and therefore are inherently more well suited for spatio-temporal forecasting than Conv2D-based models.

6.2 Cube Sphere Specific

As a proof of concept we see that we are able to integrate all new tools and classes whose components are built from scratch. We see that our model performs better than the CNN iterative model used in Rasp *et al.* (2020) but is still behind operational forecasting skill level. In fact in comparison to Weyn *et al.* (2020) we are close behind as they reach a 5-day RMSE of around 600, though they train a multi-variable model to benefit from predicting multiple correlated variables. The expectation here would be incorporate more correlated inputs like in Weyn *et al.* (2020) and compare them on a more equal footing than single-variable dynamics in our future studies.

6.3 Future research directions

6.3.1 Probabilistic Forecasts

As weather forecasts are uncertain is nature, it is usually accompanied by a probability scoring of the predictions. In NWP, this is done by ensemble forecasting using different initialization times or perturbations. Given NN-based forecasting is deterministic in nature, the problem of finding a probabilistic scoring is a new research question in its own right and can be explored further.

6.3.2 Hyper-parameter tuning

Given a proof-of-concept model has been made, another avenue of research can be a rigorous hyper-parameter tuning of the model based on uniform or bayesian based search on possible tuning-space. The model can be tuned to perform better by exploring the space of hyper-parameters like the number of hidden units and layers in each unit, network weight initialization, choice of non-linear activations, learning rate, number of epochs, time steps, and batch size and optimization techniques.

6.3.3 Addition of Input-space Variables

Owing to the constraints of GPU RAM optmization and computation time, we had to trade-off train single variable predictions with bigger architecture. As mentioned earlier, more variables need to be added from the WeatherBench datasets in order to expect that the neural network will understand the dynamics of rainfall as required with the help of relevant variables like solar insolation, humidity, wind speed vectors, etc. For global forecasts, solar insolation, topography and land-sea mask and latitudinal information are important static conditions and therefore are planned to provided with the model in future runs.

Chapter 7

Code Availability and Open Data

The custom layer classes and helper modules along with training scripts for the models trained for this work will be uploaded at github.com/aryax265/ and offline at IITM Pratyush HPC at `/lus/dal/hpcs_rnd/IISER_Pune/Arya/codes`.

The IMD Dataset is available at www.imdpune.gov.in. The WeatherBench Dataset is available as https://mediatum.ub.tum.de/1524895
The relevant source code for reproducing some of the models from previous works can be found from the source code repository found in the original papers and are listed below:

- Dueben and Bauer (2018) - https://gmd.copernicus.org/articles/11/3999/2018/gmd-11-3999-2018-supplement.zip
- Scher (2018) - Sebastian (2018), Zenodo Link
- Scher and Messori (2019) - Scher (2019), Zenodo Link
- Weyn *et al.* (2019) - github.com/jweyn/DLWP
- Weyn *et al.* (2020) - github.com/jweyn/DLWP-CS
- Rasp and Thuerey (2021) - github.com/raspstephan/WeatherBench
- Rasp *et al.* (2020) - github.com/pangeo-data/WeatherBench

References

Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, Ghemawat S, Goodfellow I, Harp A, Irving G, Isard M, Jia Y, Jozefowicz R, Kaiser L, Kudlur M, Levenberg J, Mané D, Monga R, Moore S, Murray D, Olah C, Schuster M, Shlens J, Steiner B, Sutskever I, Talwar K, Tucker P, Vanhoucke V, Vasudevan V, Viégas F, Vinyals O, Warden P, Wattenberg M, Wicke M, Yu Y, Zheng X (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, *et al.* (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:160304467 .

Bauer P, Thorpe A, Brunet G (2015). The quiet revolution of numerical weather prediction. Nature 525(7567), 47–55.

Bian Y, Xie X (2020). Generative chemistry: drug discovery with deep learning generative models.

Brenowitz ND, Bretherton CS (2018). Prognostic validation of a neural network unified physics parameterization. Geophysical Research Letters 45(12), 6289–6298.

Chapman WE, Subramanian AC, Delle Monache L, Xie SP, Ralph FM (2019). Improving atmospheric river forecasts with machine learning. Geophysical Research Letters 46(17-18), 10627–10635.

Chen L, Cao Y, Ma L, Zhang J (2020). A deep learning-based methodology for precipitation nowcasting with radar. Earth and Space Science 7(2), e2019EA000812. E2019EA000812 10.1029/2019EA000812.

Chollet F (2017). Deep Learning with Python. Manning Publications Company.

Chollet F, *et al.* (2015). Keras.

Davò F, Alessandrini S, Sperati S, Delle Monache L, Airoldi D, Vespucci MT (2016). Post-processing techniques and principal component analysis for regional wind power and solar irradiance forecasting. Solar Energy 134, 327–338.

Dueben PD, Bauer P (2018). Challenges and design choices for global weather and climate models based on machine learning. Geoscientific Model Development 11(10), 3999–4009.

George S (2020). Application of Deep learning algorithm for Monsoon Forecast. Master's thesis, Indian Institute of Technology, Madras.

Hochreiter S, Schmidhuber J (1997). Long short-term memory. Neural computation 9, 1735–80.

Hoyer S, Hamman J (2017). xarray: Nd labeled arrays and datasets in python. Journal of Open Research Software 5(1).

Kingma DP, Ba J (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:14126980 .

Kurth T, Treichler S, Romero J, Mudigonda M, Luehr N, Phillips E, Mahesh A, Matheson M, Deslippe J, Fatica M, Prabhat P, Houston M (2018). Exascale deep learning for climate analytics. In SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, 649–660.

Lagerquist R, McGovern A, Gagne I David John (2019). Deep Learning for Spatially Explicit Prediction of Synoptic-Scale Fronts. Weather and Forecasting 34(4), 1137–1160.

LeCun Y, Bengio Y, Hinton G (2015). Deep learning. Nature 436–444.

Liu Y, Racah E, Prabhat, Correa J, Khosrowshahi A, Lavers D, Kunkel K, Wehner M, Collins W (2016). Application of deep convolutional neural networks for detecting extreme weather in climate datasets.

Nielsen M (2015). Neural Networks and Deep Learning. Determination Press.

Pai D, Sridhar L, Rajeevan M, Sreejith OP, Satbhai N, Mukhopadhyay B (2014). Development of a new high spatial resolution (0.25Âř ÃŮ 0.25Âř) long period (1901-2010) daily gridded rainfall data set over india and its comparison with existing data sets over the region. Mausam 65, 1–18.

Rahman SA, Adjeroh DA (2019). Deep learning using convolutional lstm estimates biological age from physical activity. Scientific reports 9(1), 1–15.

Rasp S, Dueben PD, Scher S, Weyn JA, Mouatadid S, Thuerey N (2020). Weatherbench: A benchmark data set for dataâĂŘdriven weather forecasting. Journal of Advances in Modeling Earth Systems 12(11).

Rasp S, Lerch S (2018). Neural Networks for Postprocessing Ensemble Weather Forecasts. Monthly Weather Review 146(11), 3885–3900.

Rasp S, Pritchard MS, Gentine P (2018). Deep learning to represent subgrid processes in climate models. Proceedings of the National Academy of Sciences 115(39), 9684–9689.

Rasp S, Thuerey N (2021). Data-driven medium-range weather prediction with a resnet pretrained on climate simulations: A new model for weatherbench. Journal of Advances in Modeling Earth Systems 13(2), e2020MS002405. E2020MS002405 2020MS002405.

Reichstein M, Camps-Valls G, Stevens B, Jung M, Denzler J, Carvalhais N, Prabhat (2019). Deep learning and process understanding for data-driven earth system science. Nature 566, 195–204.

Rocha Rodrigues E, Oliveira I, Cunha R, Netto M (2018). Deepdownscale: A deep learning strategy for high-resolution weather forecast. In 2018 IEEE 14th International Conference on e-Science (e-Science), 415–422.

Scher S (2018). Toward data-driven weather and climate forecasting: Approximating a simple general circulation model with deep learning. Geophysical Research Letters 45(22), 12,616–12,622.

Scher S (2019). Code and data for "Weather and climate forecasting with neural networks: using GCMs with different complexity as study-ground".

Scher S, Messori G (2018). Predicting weather forecast uncertainty with machine learning. Quarterly Journal of the Royal Meteorological Society 144(717), 2830–2841.

Scher S, Messori G (2019). Weather and climate forecasting with neural networks: using general circulation models (gcms) with different complexity as a study ground. Geoscientific Model Development 12(7), 2797–2809.

Sebastian S (2018). Code and data for "Towards data-driven weather and climate forecasting: Approximating a simple general circulation model with deep learning".

Shi X, Chen Z, Wang H, Yeung DY, kin Wong W, chun Woo W (2015). Convolutional lstm network: A machine learning approach for precipitation nowcasting.

Sønderby CK, Espeholt L, Heek J, Dehghani M, Oliver A, Salimans T, Agrawal S, Hickey J, Kalchbrenner N (2020). Metnet: A neural weather model for precipitation forecasting. arXiv preprint arXiv:200312140 .

Vlachas PR, Byeon W, Wan ZY, Sapsis TP, Koumoutsakos P (2018). Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks. Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences 474(2213), 20170844.

Weyn JA, Durran DR, Caruana R (2019). 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(8), 2680–2693.

Weyn JA, Durran DR, Caruana R (2020). Improving data-driven global weather prediction using deep convolutional neural networks on a cubed sphere. Journal of Advances in Modeling Earth Systems 12(9), e2020MS002109. E2020MS002109 10.1029/2020MS002109.

Wicht B (2018). Deep Learning feature Extraction for Image Processing. Ph.D. thesis.