Titre: Title:	Worldwide Weather Forecasting by Deep Learning
Auteur: Author:	Philippe Trempe
Date:	2019
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Trempe, P. (2019). Worldwide Weather Forecasting by Deep Learning [Master's thesis, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/3854/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/3854/
Directeurs de recherche: Advisors:	Michel Gagnon
Programme: Program:	Génie informatique

UNIVERSITÉ DE MONTRÉAL

WORLDWIDE WEATHER FORECASTING BY DEEP LEARNING

PHILIPPE TREMPE
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
MAI 2019

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

WORLDWIDE WEATHER FORECASTING BY DEEP LEARNING

présenté par: TREMPE Philippe
en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de:

EPIGRAPH

"The brain surely does not work by
somebody programming in rules."

— Geoffrey Hinton

ACKNOWLEDGEMENTS

Many thanks go to the great people at Pelmorex (with a special thank to Edwin Vargas (project manager and supervisor)), partners at Google (with a special thank to Shibl Mourad (insightful data science/engineering and machine learning lead)), and the Polytechnique Montreal (University of Montreal) research team (with special thanks to Guillaume-Alexandre Bilodeau (jury president), Christopher J. Pal (jury member) and Michel Gagnon (research director)) for their support, feedback and insightful discussions throughout the undertaking of this work. Special thanks also go to the author's close ones, including family, friends, colleagues and all others who contributed in making the undertaking of this work an enlightening and pleasant experience.

RÉSUMÉ

La prévision météorologique a été et demeure une tâche ardue ayant été approchée sous plusieurs angles au fil des années. Puisque les modèles proéminents récents sont souvent des modèles d'appentissage machine, l'importance de la disponibilité, de la quantité et de la qualité des données météorologiques augmente. De plus, la revue des proéminents modèles d'apprentissage profond appliqués à la prédiction de séries chronologiques météorologiques suggère que leur principale limite est la formulation et la structure des données qui leur sont fournies en entrée, ce qui restreint la portée et la complexité des problèmes qu'ils tentent de résoudre.

À cet effet, cette recherche fournit une solution, l'algorithme d'interpolation géospatiale SkNNI (interpolation des *k* plus proches voisins sphérique), pour transformer et structurer les données géospatiales disparates de manière à les rendre utiles pour entraîner des modèles prédictifs. SkNNI se démarque des algorithmes d'interpolation géospatiale communs, principalement de par sa forte robustesse aux données d'observation bruitées ainsi que sa considération accrue des voisinages d'interpolation.

De surcroît, à travers la conception, l'entraînement et l'évaluation de l'architecture de réseau de neurones profond DeltaNet, cette recherche démontre la faisabilité et le potentiel de la prédiction météorologique multidimensionnelle mondiale par apprentissage profond. Cette approche fait usage de SkNNI pour prétraiter les données météorologiques en les transformant en cartes géospatiales à multiples canaux météorologiques qui sont organisées et utilisées en tant qu'éléments de séries chronologiques. Ce faisant, le recours à de telles cartes géospatiales ouvre de nouveaux horizons quant à la définition et à la résolution de problèmes de prévisions géospatiales (p. ex. météorologiques) plus complexes.

Substantiellement, cette recherche propose une formulation de problèmes de prévisions géospatiales novatrice et riche en information, et montre ses bénéfices quant à la prédiction de multiples paramètres météorologiques à travers le monde. À cet effet, le but principal de ces travaux est de fonder des bases solides dans le domaine, et d'inspirer chercheurs et praticiens à définir des problèmes toujours plus riches en bâtissant sur la présente recherche pour ultimement aider les gens et les organisations à planifier et opérer en sécurité.

ABSTRACT

Weather forecasting has been and still is a challenging task which has been approached from many angles throughout the years. Since recent state-of-the-art models are often machine learning ones, the importance of weather data availability, quantity and quality rises. Also, the review of prominent deep learning models for weather time series forecasting suggests their main limitation is the formulation and structure of their input data, which restrains the scope and complexity of the problems they attempt to solve.

As such, this work provides a solution, the spherical k-nearest neighbors interpolation (SkNNI) algorithm, to transform and structure scattered geospatial data in a way that makes it useful for predictive model training. SkNNI shines when compared to other common geospatial interpolation methods, mainly because of its high robustness to noisy observation data and acute interpolation neighborhood awareness.

Furthermore, through the design, training and evaluation of the DeltaNet deep neural network architecture, this work demonstrates the feasibility and potential of multidimensional worldwide weather forecasting by deep learning. This approach leverages SkNNI to preprocess weather data into multi-channel geospatial weather frames, which are then organized and used as time series elements. Thus, working with such geospatial frames opens new avenues to define and solve more complex geospatial (e.g. weather) forecasting problems.

Essentially, this research proposes a novel and information-rich geospatial forecasting problem formulation and shows its benefits for accurate forecasting of multiple weather parameters worldwide. Hence, the main goal of this work is to lay groundwork in the field, and inspire researchers and practitioners to define ever-richer problems by building upon it to ultimately help people and organizations plan ahead and operate safely.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF LISTINGS

LIST OF SYMBOLS, ACRONYMS AND ABBREVIATIONS

Adam	adaptive moment estimation
AMERPE	absolute maximum error ratio percentage error
CMAE	channel mean absolute error
ConvLSTM	convolutional long short-term memory
ConvNet	convolutional neural network
DRU	delta reduction unit
ELM	extreme learning machine
LSTM	long short-term memory
NDD	neighborhood distribution debiasing
NDDNISD	neighborhood distribution debiased normalized inverse squared distance
NISD	normalized inverse squared distance
ResDRU	residual delta reduction unit
RNN	recurrent neural network
SkNNI	spherical k-nearest neighbors interpolation
SVRM	support vector regression machine

GLOSSARY

nowcast	Forecast at a high time resolution (often hourly or every few minutes)
tensor	Generalization of scalars, vectors, matrices and the like to an arbitrary number of dimensions and represented as *n*-dimensional collections of base data types (integers, floating point numbers, etc.)

CHAPTER 1 INTRODUCTION

Due to its effect on human life and activities, weather remains a very active research field. Throughout the years, researchers have been developing a variety of statistical and algorithmic weather forecasting techniques to better model the environment and predict its evolution [1]. Since weather forecasting remains a modern challenge due to its inherent complexity [2], researchers have started turning to more complex algorithms and models, often machine learning ones [3]. Such models' applications are currently being explored and are already yielding significant results regarding the prediction of weather parameters like temperature [4], air quality (urban air pollution) [5], and solar radiation (horizontal irradiance) [6].

1.1 Motivation

Reviewing the recent scientific literature in the field (see §2) exposes the popularity of machine learning methods' application to weather forecasting. It also exposes a frequently occurring pattern in the problem formulation and proposed solution presented in many of the reviewed articles: the focus is on a single weather parameter for a single geographic location, and the proposed solution is an adapted variant of a machine learning model which is used to forecast the weather parameter of interest [1, 3, 4, 7–10]. These low-dimensional (i.e. one scalar value per time step) formulations of the weather forecasting problem, which are mainly due to data availability limitations, are likely to diminish the richness and contextual information of model inputs and predictions. Thus, questions arise about how restrictive such low-dimensional model inputs might be to the predictive model's learning and prediction ability, notwithstanding having to retrain a new model for each weather parameter and for each location. As such, these interrogations motivate the proposition of a high-dimensional weather forecasting problem formulation with weather states (time steps) which have three extra dimensions: latitude, longitude and weather parameter channel.

1.2 Problem Definition

The broad problem under study is first and foremost a weather forecasting problem, i.e. a time series of past weather states is used to forecast the weather state at the next time step. The main distinction from more typical weather forecasting formulations is the addition of three extra dimensions to the weather states: latitude, longitude and weather parameter channel. Working with these extra weather state dimensions adds richness to them by allowing the model to capture correlations and influences between the variables, but also adds data processing requirements since the data must be made usable for the predictive model.

Thus, there are two main problems under study, the latter depending on at least a partial solution to the former. The first problem is about finding a way to organize and transform scattered potentially sparse multidimensional geospatial data in a way that makes it useful for a predictive model to learn from. The second problem is about automatically forecasting weather parameters worldwide as accurately as possible using a deep neural network working with the transformed input data.

Hence, the goal of this research is to lay some groundwork in the field of multidimensional worldwide weather forecasting by deep learning. There are also some secondary goals like leveraging big data and cloud computing to automate and streamline the process for real-world applications. To reach the main goal, a multidimensional worldwide weather forecasting framework, with its data transformations and predictive models, is developed and evaluated. As such, the project is considered successful if the created framework allows for the automatic training of a deep neural network model which is capable of using multidimensional worldwide weather data to make worldwide forecasts which are, on average, more accurate than the ones made by comparable baseline models.

1.3 Outline

This work is organized as outlined hereby. First, the research context is introduced and leads to the motivation regarding the undertaking of this work. The problems under study are then presented alongside the research framework, goals and success conditions. Then, the relevant background and related work are presented to provide the necessary notions to further understand this work. Subsequently, the body of this work covers both problems of interest, i.e. geospatial data transformation for model usability and automatic multidimensional worldwide weather forecasting by deep learning. For each are thus presented the approach, results and analysis. Ultimately, this work is summed up and concluded with considerations pertaining to its limitations, contributions and potential future work.

1.4 Disclaimer

Before moving on, is hereby presented a brief disclaimer which states that the real world data used throughout this work originates from proprietary datasets which shall remain private. Thus, this work will attempt to provide a general idea as to what such a dataset might look like in similar projects, but will intentionally remain vague about the one used throughout.

CHAPTER 2 BACKGROUND AND RELATED WORK

This chapter explores the state of the art regarding the use of deep learning to tackle meteorology-related problems. In doing so, articles published over the last few years have been reviewed and the ones presenting original and field-leading contributions have been retained for further analysis.

2.1 Meteorological Data

Meteorological data is generated by the numerous automatic and manual weather stations around the world and aggregated by organizations for archiving and further processing. Since weather data is geospatial, it often follows typical schemes like the very common timestamped geospatial value scheme which is of the form shown in table 2.1. This highly dimensional

Table 2.1 Example of Typical Timestamped Geospatial Values

Timestamp	Latitude	Longitude	Value
2000-01-01 01:00:00	-38.123	29.654	12.345
2000-01-01 01:00:00	48.286	-120.259	9.874
2000-01-01 02:20:00	-38.123	29.654	13.736
2000-01-01 02:20:00	48.286	-120.259	11.397

data is then often further organized as time series for a variety of weather parameters like temperature, wind speed, wind direction, pressure, relative humidity, etc. Since gathering clean and complete data for many weather parameters is often challenging for researchers, most reviewed articles focus on one or a few meteorological parameters. For example, [10] focuses on forecasting temperature, while [11] focuses on predicting wind speed, [12] solar radiation, and [1] rainfall.

2.2 Temporal Coverage and Resolution

Temporal coverage and resolution vary from a studied article to another. In most, about a year or two [8, 9, 13] (and sometimes more [10, 14]) worth of weather data is aggregated daily [4,9,10,15], hourly [8,13], or every few minutes [16], and time series of said weather data are used to forecast a meteorological parameter of interest for the next time step. As time resolution increases, fewer articles are found due to how difficult finding high quality datasets with high temporal resolution is, which limits the time-related precision proposed models achieve. As such, [16] exposes the lack of high-temporal-resolution weather forecasting studies while noting their importance in predicting impactful short-duration events like rainfall to allow organizations like airports to plan ahead and ensure safe flights to passengers.

2.3 Geospatial Coverage and Resolution

Geospatial coverage and resolution vary much less from article to article. In fact, most solutions receive meteorological data for one specific location and use it to forecast a weather parameter for that same location [1, 3, 4, 7–10]. Nonetheless, some solutions are based on multiple closely scattered (sometimes autonomous, sometimes manual) weather stations covering a region [13, 16]. A notable article even used weather data from 692 automated weather stations in South Korea [14], covering the whole country at a high resolution.

2.4 Solving Problems With Machine Intelligence

2.4.1 Problems

In almost all reviewed articles, the problem to be solved by researchers is defined as a weather time series forecasting problem (see figure 2.1) for one specific weather parameter like temperature [3, 4, 10], wind speed [8, 11], air quality [13], rainfall [1], etc. One of the notable exceptions is [16] in which the authors define their rainfall nowcasting problem as spatiotemporal since they are working with time series of radar maps. Another considerable article is [14] since the problem its authors aim to solve is a data consistency and error correction one. Thus, their article focuses on finding erroneous values and on determining adequate replacement values for them, which, in the end, helps build more complete and consistent weather datasets for ulterior use.

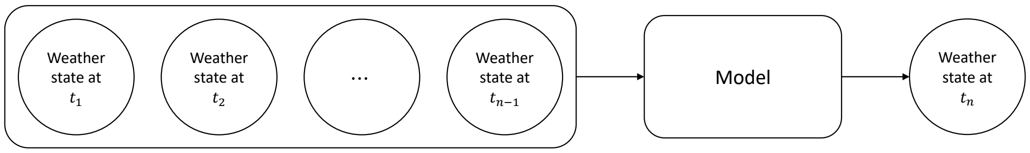

Figure 2.1 Weather Time Series Forecasting Problem Definition. The figure illustrates the problem definition for weather time series forecasting. It shows a sequence of consecutive weather states which are passed to a predictive model which then predicts the weather state at the next time step.

2.4.2 Models

State-of-the-art solutions to weather forecasting problems often use machine learning models. As such, one of the most frequent ones is the support vector regression machine (SVRM) [17] (a support-vector-machine-based regression model which minimizes prediction error by individualizing the hyperplane which maximizes margin while tolerating a part of the error), for which each research team has developed a custom variant better adapted to solving their forecasting problem [4, 12, 14]. Hybrid models were also developed by other researchers to include a feature selection step before model training [9, 10].

Nonetheless, the most popular types of models for weather forecasting using machine intelligence techniques are extreme learning machines (ELMs) [18] (a fast-learning kind of usually shallow feedforward neural network which makes use of random projections and which is trained by computing the Moore-Penrose inverse (pseudoinverse) using singular value decomposition, e.g. [8, 11, 13, 19]) and deep neural networks (a deep computational graph of layers which is trained through gradient-based optimization by using the chain rule of calculus to iteratively compute gradients for topologically ordered layers from prediction backwards, e.g. [1, 3, 7, 16]).

State-of-the-art extreme learning machines sometimes use feature selection [13], neuro-fuzzy models (a kind of predictive model which combines neural networks and fuzzy logic, which is a form of logic which works with probabilities instead of binary values, e.g. [8]), and online machine learning (a machine learning method for which data becomes available in sequential order, as opposed to batch processing on a complete dataset, and which updates its predictive model as new data becomes available, e.g. [19]) to push the boundaries of fast model training.

State-of-the-art deep learning models are mostly recurrent neural networks (RNNs) [20–22] (a class of neural network which excels at processing sequential data due to its internal state), often long short-term memory (LSTM) [23] neural networks (a kind of recurrent neural network which controls its internal state using learned gates and which excels at sequence processing, even when correlated information is far apart in the sequence), and sometimes convolutional neural networks (ConvNets) [24] (a class of neural network which is based on using convolution/correlation operations and which has interesting properties like translation invariance) to make predictions on time series data.

Furthermore, [16], with the researchers' problem definition as spatiotemporal, unites both LSTMs and ConvNets in what is called a convolutional long short-term memory (ConvLSTM) neural network. As the name implies, a ConvLSTM is very similar to a classic LSTM, the main difference being its preactivations are convolutional instead of linear. With such an architecture, the model learns patterns in the temporal evolution of spatial motifs in input sequences, and makes use of its learned hidden parameters to make predictions. Hence, this model architecture has a lot of potential for learning more complex spatiotemporal patterns evolving through time like weather ones.

2.5 Conclusion

As shown throughout this state of the art review, many research teams are working on solving weather forecasting problems by making use of machine intelligence. Even though most of the data is often for a single location and forecasts for a single weather parameter, new breakthroughs keep pushing the domain's frontiers, so much that, nowadays, state-of-the-art models like ELMs and spatiotemporal deep neural networks are able to work with highly dimensional weather datasets to provide ever more accurate forecasts for people to plan their day and live safely.

CHAPTER 3 SPHERICAL K-NEAREST NEIGHBORS INTERPOLATION

This chapter focuses on the problem of sparse spherical interpolation. Such an approach is necessary to transform scattered geospatial data into a dense (not sparse) and always-identical format for predictive model usability. Thus, this chapter presents challenges in organizing and transforming scattered potentially sparse geospatial data, as well as a novel algorithm named spherical k-nearest neighbors interpolation (SkNNI) (pronounced "skinny") which addresses the problem.

3.1 Introduction

A significant proportion of big data is geospatial, and this proportion is estimated to keep growing by at least 20 % every year [25]. Also, big data is highly heterogeneous. Researchers estimate that about 95 % of big data is unstructured, and geospatial big data is, of course, no exception [26]. It is voluminous, heterogeneous, and variable [27]. Nonetheless, geospatial data is useful in a variety of fields like healthcare, security, marketing, environmental modeling, and business intelligence, providing much insight on diverse behavioral and evolutional traits [28–30]. Thus, there is a clear need by researchers, companies and other organizations for processing, organizing and structuring geospatial big data in a way that makes it useful for them.

3.2 Problem Definition

Geospatial big data is highly heterogeneous, often sparse, and scattered around the globe, making it difficult to use. Since many algorithms and predictive models require dense and identically structured inputs, an algorithm shall be designed in order to transform the sparse and scattered geospatial data. Thus, the problem is expressed as follows. See figure 3.1 for an illustration of the problem. Starting with observations of the form $(\text{latitude}, \text{longitude}, \text{value})$ scattered on the surface of a sphere with radius ρ, the objective is to determine sensible data values at given interpolation coordinates of the form $(\text{latitude}, \text{longitude})$. Note that no assumptions regarding the quantity nor distribution of the coordinates are made. Thus, the number and distribution of observation and interpolation coordinates are variable. Also, some extra parameters may be passed to the geospatial interpolation algorithm for configuration purposes. Thus, the main goal of this work is stated as making such an algorithm which is applicable, numerically stable, efficient and simple to use by anyone looking to perform potentially sparse spherical interpolation, e.g. to structure scattered geospatial big data for use in further algorithms and predictive models requiring a dense and fixed input structure.

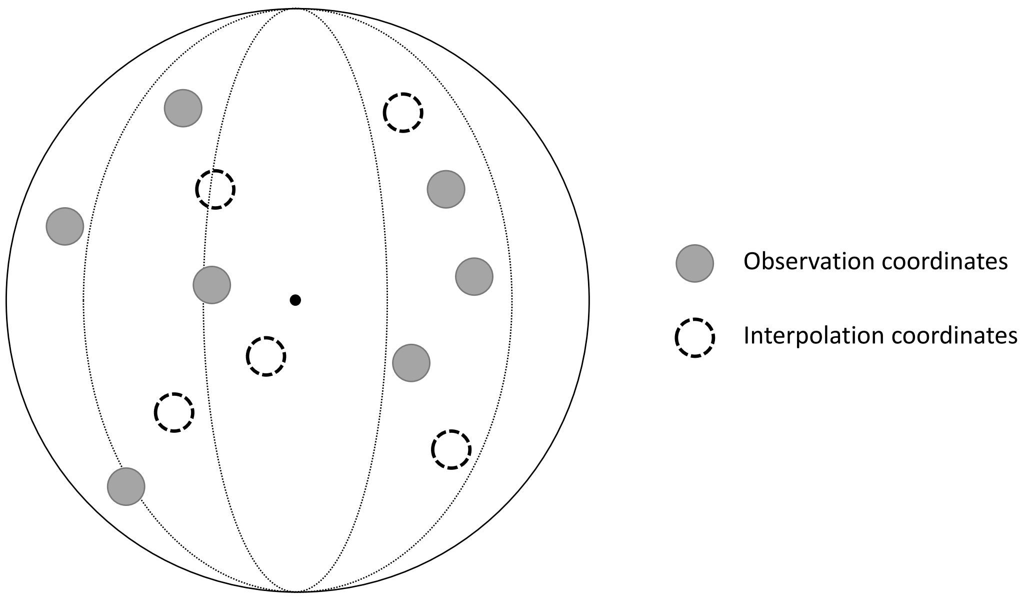

Figure 3.1 Sparse Spherical Interpolation Problem Definition. The figure illustrates the problem definition for sparse spherical interpolation. It shows the observation coordinates as filled and the interpolation coordinates as hollow.

3.3 Background and Related Work

Before presenting the proposed solution, a review of existing interpolation algorithms related to the problem is undertaken, covering advantages, limitations, and most importantly applicability of each reviewed candidate. Also note that, to be retained, considered algorithms must be both practically implementable as software and executable in a reasonable amount of time.

3.3.1 Bilinear Interpolation

When tackling an interpolation problem, a classic and well-known algorithm to first consider is bilinear interpolation [31]. This algorithm performs interpolation on two-dimensional data residing on a regular grid by first linearly interpolating along one dimension, and then by interpolating along the other. It has the advantage of being rather simple and quite fast to calculate. One of its main limitations is its interpolation accuracy, which gets outperformed by more sophisticated algorithms like bicubic interpolation. Thus, since this work requires an interpolation algorithm which supports irregular and sparse data both in the input and the output of the algorithm, this solution is not retained.

3.3.2 Bicubic Interpolation

When tackling an interpolation problem where bilinear interpolation is considered, bicubic interpolation [31] should also be reviewed. This algorithm performs cubic interpolation on two-dimensional data residing on a regular grid by using third-order polynomials. It first does a cubic interpolation along one dimension, and then does a cubic interpolation along the other. This algorithm's main advantage is its interpolation accuracy while its main downside is being more computationally expensive. Though, as for bilinear interpolation, since this work requires an interpolation algorithm which supports irregular and sparse data both in the input and the output of the algorithm, this solution is not retained.

3.3.3 Smooth Sphere Bivariate Spline

Another explored candidate algorithm is the smooth sphere bivariate spline found in SciPy [32], which is a "Smooth bivariate spline approximation in spherical coordinates" [33]. This algorithm fits a spherical bivariate spline such that it minimizes the error with respect to observations while also applying a smoothing regularization to mitigate the impact of outliers. As such, not smoothing enough has a hard time dealing with outliers and smoothing too much has a hard time fitting the provided observations. Thus, tuning the algorithm's smoothing parameter seems tedious. Also, making such a parameter trainable would lead to some undesirable overhead, on top of the one required to fit the spherical bivariate spline. With such considerations regarding tedious tuning and computational expensiveness, this solution is not retained.

3.3.4 LSQ Sphere Bivariate Spline

The next reviewed candidate algorithm is the LSQ sphere bivariate spline found in SciPy [32], which is a "Weighted least-squares bivariate spline approximation in spherical coordinates" [34]. The algorithm fits a spherical bivariate spline such that it minimizes squared error with respect to observations at given knot coordinates. Some variants of the algorithm allow for the addition of smoothing (through minimization of discontinuity jumps in the spline's derivatives at the provided knots) in the objective to minimize, but not SciPy's, at least not in any way parameterizable by the user. As such, observation outliers are likely to be troublesome. Also, fitting a spline adds undesirable computational overhead. Furthermore, choosing how many knots (which are different from observation and interpolation points, adding to the user's confusion) to use, and with which coordinates, is likely to be application-dependent and require an undesirable amount of tuning. Thus, considering the algorithm's usability complexity, fragility to outliers and computational expensiveness, it is not retained.

3.3.5 Gaussian Process Regression (Kriging)

Gaussian process regression, also known as Kriging, is an interpolation method often used in geostatistics and for which values are modeled using a Gaussian process based on prior covariances [35]. This approach is more flexible since the distribution represents values at a set of interpolation points while being conditioned on observation values at another set of (observation) coordinates. Thus, this approach has the advantage of not requiring a regular grid for inputs and outputs. Unfortunately, Gaussian process regression becomes a problem when having to deal with ever-changing data coordinates, since a new Gaussian process regression model needs to be created and its kernel chosen and parameters fit each time there is a new observation or interpolation point. Considering problematic cases like the addition of new weather stations, the reactivation of weather stations after maintenance, and mobile weather stations with ever-changing coordinates; where the Gaussian process regression model would need to be recreated and refit for every such change, the impracticability (due to computational expensiveness) of such an approach to the studied interpolation problem leads to its rejection.

3.4 Motivation

After reviewing various interpolation methods, from the classic bilinear interpolation to the more sophisticated Gaussian process regression, it seems there is no current approach which is fully suitable to solve the presented sparse spherical interpolation problem while respecting the given constraints. For example, bilinear and bicubic interpolation methods are not applicable since they only operate on regular two-dimensional grids. As for SciPy's bivariate spline interpolations, their tedious tuning and computational expensiveness make them unsuitable. Is then left Gaussian process regression which is, by far, the most applicable currently existing solution since it accepts sparse inputs and outputs, and is already implemented in various frameworks. Unfortunately though, this solution also cannot be retained because of the ever-changing nature of input and output coordinates, and also because of the computational expensiveness required to train the model's kernel each time new coordinates appear, which is expected to happen very often.

With these considerations in mind, the solution sought for is required to be able to deal with both sparse inputs and outputs, in varying numbers and coordinates, without prior assumptions on them except for validity, which may be ensured through software validation. Furthermore, since the algorithm is to be implemented as software and run on very large weather datasets, it needs to be computationally efficient and scalable. Put simply, the algorithm needs to be designed with big data, cloud computing and scale in mind. As such, three main axes to consider throughout the design and implementation of the algorithm are the scientific/research one, the business/engineering one, and the developer/end user one.

3.5 Proposed Algorithm

Since there is currently no algorithm suitable to solve the problem under study while respecting the given constraints, this work presents a novel approach: spherical k-nearest neighbors interpolation (SkNNI) (pronounced “skinny”), along with a novel interpolation function: neighborhood distribution debiased normalized inverse squared distance (NDDNISD).

The main idea behind SkNNI is to be able to find the k-nearest neighbors of each interpolation point fast and efficiently. Thus, an efficient geospatial data structure is needed, which is why a k-dimensional tree is used to index the geospatial data. Since k-dimensional trees operate in Euclidean space for their distance calculations, the spherical/polar (observation and interpolation) coordinates need to be transformed. The k-dimensional tree is then queried to obtain the neighborhood (k-nearest neighbors) of each interpolation point. The information about interpolation points and their neighborhood is then passed to an interpolation function to calculate the interpolated value for each interpolation point. Thus, this work proposes the NDDNISD interpolation function, which shines due to its consideration of locality/proximity and distribution of observation neighbors in interpolation neighborhoods when interpolating.

In the same vein, the inner workings of SkNNI are as follows. For clarity, an overview of the SkNNI algorithm is presented in figure 3.2. The idea is to start with observations, transform them to make them indexable by a k-dimensional tree, build it, and use it to find nearest neighbors of new coordinates. These new coordinates are also transformed to be in the same space, and are then used to query the k-dimensional tree for the k-nearest neighbors of each interpolation point. Once the neighborhood of each interpolation point determined, each neighborhood’s information is then used to calculate the interpolated value at its interpolation coordinates. This interpolation step is realized using an interpolation function, the one put forth in this work being NDDNISD (though arbitrary compatible interpolation functions may be used instead).

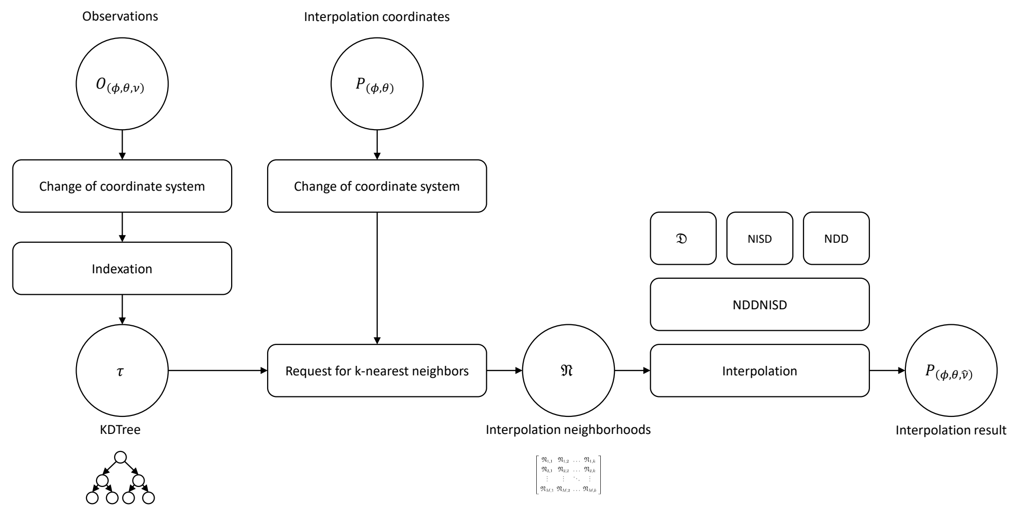
Observations
$O_{(\phi,\theta,\nu)}$
Interpolation coordinates
$P_{(\phi,\theta)}$
Change of coordinate system
Change of coordinate system
Indexation
τ
KDTree
Request for k-nearest neighbors
$\mathfrak{N}$
Interpolation neighborhoods
$\mathfrak{D}$
NISD
NDD
NDDNISD
Interpolation
$P_{(\phi,\theta,\hat{\nu})}$
Interpolation result

Figure 3.2 SkNNI Algorithm Overview. The figure shows an overview of the S*k*NNI algorithm with inputs, outputs, transformations and intermediate results.

With the intuition and algorithm overview covered, §3.6 formalizes and details the inner workings of each of S*k*NNI's steps, as well as the ones of NDDNISD.

3.6 Algorithm Execution Steps

Executing the S*k*NNI algorithm starts with obtaining N observations $O = \langle o_1, o_2, \ldots, o_N \rangle$, where large angle brackets denote sets which preserve insertion order (also known as lists), scattered on a sphere. Each observation o_i is defined with the following three attributes: the observation latitude $o_{i,\phi} \in [-90, 90[$ in degrees, the observation longitude $o_{i,\theta} \in [-180, 180[$ in degrees, and the observation data value $o_{i,\nu} \in \mathbb{R}$. The objective is to determine sensible data values $p_{i,\hat{\nu}} \in \mathbb{R}$ at the M given interpolation coordinates $P = \langle p_1, p_2, \ldots, p_M \rangle$. Each p_i is defined with the following two attributes when passed to the algorithm: the interpolation latitude $p_{i,\phi} \in [-90, 90[$ in degrees and the interpolation longitude $p_{i,\theta} \in [-180, 180[$ in degrees. The third attribute is the interpolated data value $p_{i,\hat{\nu}}$ which is added to each interpolation point throughout the execution of the algorithm. Three arguments may also be configured by the user: the number of nearest neighbors $k \in \mathbb{N}_{>0,\leqslant N}$ to consider when interpolating, the sphere's radius $\rho \in \mathbb{R}_{>0}$, and the interpolation function $\mathfrak{I}$ which is as defined in §3.6.4. Since some configuration parameters will often be taking the same value, for example setting the sphere's radius ρ to the mean radius of the Earth, which is (6371.01 ± 0.02) km [36], the algorithm will have decent default values for each configuration parameter. As a note, the choice of using degrees for input and output latitudes and longitudes is deliberate and comes from the empirical consideration that most geospatial data possessed by potential users of the algorithm is in the format hereinbefore described. Since inputs and outputs are part of the user interface of the algorithm, data formats were chosen such that they will likely match users' data. As a last note before continuing, all trigonometric operators used in the algorithm operate in radians and the input degrees are quickly converted to radians for the algorithm's internal processing. With these considerations covered, the algorithm's main execution steps are presented, each described in its respective section.

3.6.1 Change of Coordinate System

When the input data enters the algorithm, it is first validated and useful error messages are shown to the user if an input gets invalidated. Then, each observation o_i gets transformed by applying the change of coordinate system described by the following expressions.

$$\mathcal{T}_x \colon \mathbb{R}_{\geqslant -90,<90} \times \mathbb{R}_{\geqslant -180,<180} \times \mathbb{R}_{>0} \to \mathbb{R}_{\geqslant -\rho,\leqslant \rho} \tag{3.1}$$

$$\mathcal{T}_y \colon \mathbb{R}_{\geqslant -90,<90} \times \mathbb{R}_{\geqslant -180,<180} \times \mathbb{R}_{>0} \to \mathbb{R}_{\geqslant -\rho,\leqslant \rho} \tag{3.2}$$

$$\mathcal{T}_z \colon \mathbb{R}_{\geqslant -90,<90} \times \mathbb{R}_{\geqslant -180,<180} \times \mathbb{R}_{>0} \to \mathbb{R}_{\geqslant -\rho,\leqslant \rho} \tag{3.3}$$

$$o_{i,x} = \mathcal{T}_x\left(o_{i,\phi}, o_{i,\theta}, \rho\right) = \rho \cdot \cos\left(\frac{\pi \cdot o_{i,\theta}}{180}\right) \cdot \sin\left(\frac{\pi \cdot o_{i,\phi}}{180}\right) \tag{3.4}$$

$$o_{i,y} = \mathcal{T}_y\left(o_{i,\phi}, o_{i,\theta}, \rho\right) = \rho \cdot \sin\left(\frac{\pi \cdot o_{i,\theta}}{180}\right) \cdot \sin\left(\frac{\pi \cdot o_{i,\phi}}{180}\right) \tag{3.5}$$

$$o_{i,z} = \mathcal{T}_z\left(o_{i,\phi}, o_{i,\theta}, \rho\right) = \rho \cdot \cos\left(\frac{\pi \cdot o_{i,\phi}}{180}\right) \tag{3.6}$$

Expressions (3.1) to (3.6) show the transforms $\mathcal{T}_x$, $\mathcal{T}_y$ and $\mathcal{T}_z$ used to convert input coordinates to radians, and then from polar coordinates to Cartesian x, y, and z coordinates.

3.6.2 Indexing Using a k-Dimensional Tree

Once the observations transformed, a k-dimensional tree, in this case a 3-dimensional tree, is built from a set, which preserves insertion order (for later indexing), of Cartesian coordinate tuples $\left(o_{i,x}, o_{i,y}, o_{i,z}\right)$, one for each observation o_i, as shown in equation (3.7).

$$\tau = \beta\left(\left\langle\left(o_{i,x}, o_{i,y}, o_{i,z}\right)\right\rangle_{\forall i \in \left\{1,2,\ldots,N\right\}}\right) \tag{3.7}$$

In the equation, β represents the k-dimensional tree building operator which partitions the space observation points are defined in and uses this tree-structured partitioning to spatially index the observations, and τ represents the built k-dimensional tree. Once constructed, the k-dimensional tree may then be used by the algorithm as a spatial index to quickly find nearest neighbors of points [37]. At this point in the process, the interpolator object (containing the given configuration, observations and built k-dimensional tree) may be returned to user space for eventual querying.

3.6.3 Finding Neighbors by Querying a k-Dimensional Tree

To make use of the interpolator, the user specifies interpolation points P, and optionally the number of nearest neighbors to consult for interpolation k and the interpolation function $\mathfrak{I}$ (which is elaborated further). At first, the input data is validated and a clear error message is provided to the user if any input gets invalidated. The interpolation points P are then transformed using expressions (3.1) to (3.6) and used to issue a query to the k-dimensional tree τ. The query result $\mathfrak{N}$ is a set of sets, which all preserve insertion order, where each inner set contains the k-nearest neighbors for each corresponding queried point $p_i \in P$. This step is also expressible in matrix form, as shown in equation (3.8), which is simpler to visualize and implement.

$$\mathfrak{N} = \tau\left(\begin{bmatrix} p_{1,x} & p_{1,y} & p_{1,z} \\ p_{2,x} & p_{2,y} & p_{2,z} \\ \vdots & \vdots & \vdots \\ p_{M,x} & p_{M,y} & p_{M,z} \end{bmatrix}\right) = \begin{bmatrix} \mathfrak{N}_{1,1} & \mathfrak{N}_{1,2} & \dots & \mathfrak{N}_{1,k} \\ \mathfrak{N}_{2,1} & \mathfrak{N}_{2,2} & \dots & \mathfrak{N}_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ \mathfrak{N}_{M,1} & \mathfrak{N}_{M,2} & \dots & \mathfrak{N}_{M,k} \end{bmatrix} \tag{3.8}$$

In the equation, each $\mathfrak{N}_{i,j}$ corresponds to a neighbor index, and these indices are used to access the corresponding observation points in O, since insertion order was preserved for their containing set. Equation (3.9) shows the neighboring observations, where $o_{\mathfrak{N}_{i,j}}$ denotes the observation at index $\mathfrak{N}_{i,j}$, that will be used to perform interpolation in further steps.

$$O_{\mathfrak{N}} = \begin{bmatrix} o_{\mathfrak{N}_{1,1}} & o_{\mathfrak{N}_{1,2}} & \cdots & o_{\mathfrak{N}_{1,k}} \\ o_{\mathfrak{N}_{2,1}} & o_{\mathfrak{N}_{2,2}} & \cdots & o_{\mathfrak{N}_{2,k}} \\ \vdots & \vdots & \ddots & \vdots \\ o_{\mathfrak{N}_{M,1}} & o_{\mathfrak{N}_{M,2}} & \cdots & o_{\mathfrak{N}_{M,k}} \end{bmatrix} \tag{3.9}$$

At this point, a matrix containing all k-nearest observation neighbors for each interpolation point is obtained. All there is left to do is interpolate the values using an interpolation function.

3.6.4 Interpolating Values

In this last step, interpolated values $P_{\hat{\nu}} \in \mathbb{R}^M$ are calculated for all interpolation points by combining the information of their k-nearest observation neighbors. To execute this part of the algorithm, an interpolation function of the form defined in expressions (3.10) and (3.11) is used.

$$\mathfrak{I} : \mathbb{R}^M_{\geqslant -\frac{\pi}{2}, < \frac{\pi}{2}} \times \mathbb{R}^M_{\geqslant -\pi, < \pi} \times \mathbb{R}^{M \times k}_{\geqslant -\frac{\pi}{2}, < \frac{\pi}{2}} \times \mathbb{R}^{M \times k}_{\geqslant -\pi, < \pi} \times \mathbb{R}^{M \times k} \times \mathbb{R}_{>0} \times \mathbb{N}_{>0, \leqslant N} \to \mathbb{R}^M \tag{3.10}$$

$$P_{\hat{\nu}} = \mathfrak{I}\left(O_{\mathfrak{N},\phi}, O_{\mathfrak{N},\theta}, O_{\mathfrak{N},\nu}, P_\phi, P_\theta, \rho, k\right) \tag{3.11}$$

In these expressions, $O_{\mathfrak{N},\phi}$ is the matrix containing the latitude of each observation neighbor for each interpolation point, $O_{\mathfrak{N},\theta}$ is the matrix containing the longitude of each observation neighbor for each interpolation point, and $O_{\mathfrak{N},\nu}$ is the matrix containing the value of each observation neighbor for each interpolation point. If the user does not specify an interpolation function, the algorithm defaults to the neighborhood distribution debiased normalized inverse squared distance (NDDNISD) function, which is described in §3.6.5. Since this last step works in matrix form, the result is thus a vector $P_{\hat{\nu}} \in \mathbb{R}^M$. At that point, the interpolation points and their associated interpolated values are simply combined to produce the final output P of the algorithm which is ultimately returned to user space.

3.6.5 Neighborhood Distribution Debiased Normalized Inverse Squared Distance Interpolation Function

The neighborhood distribution debiased normalized inverse squared distance (NDDNISD) interpolation function is composed of three primary steps: great circle distance calculation (detailed in §3.6.6) to evaluate geospatial proximity, normalized inverse squared distance (NISD) calculation for proximal importance weighting, and neighborhood distribution debiasing (NDD) to better deal with biased neighborhood geospatial distributions. In the following equations, all operations are element-wise, unless otherwise specified, with broadcasting to extra dimensions à la NumPy if necessary. Also note that large operators like the sum are reductive, unless otherwise specified, meaning the tensors on which the operation is applied lose the dimensions corresponding to the reduction axes.

Thus, as shown in equation (3.12), the great circle distance between each interpolation point and each of its k-nearest observation neighbors is first calculated using the great circle distance function $\mathfrak{D}$ which is detailed in §3.6.6.

$$\delta = \mathfrak{D}\left(P_\phi, P_\theta, O_{\mathfrak{N},\phi}, O_{\mathfrak{N},\theta}, \rho\right) \tag{3.12}$$

The distances in each neighborhood (see figure 3.3) are then converted into weights through the NISD function (equation (3.13)) which computes the element-wise square of each distance, adds a very small positive constant $\varepsilon_{\to 0^+, >0}$ to it to avoid division by zero, inverts the values, and normalizes each row of the distance weight matrix w_δ such that its elements sum up to one.

$$w_\delta = \frac{\frac{1}{\delta^2+\varepsilon}}{\sum_{i=1}^{k} \frac{1}{\delta_i^2+\varepsilon}} \tag{3.13}$$

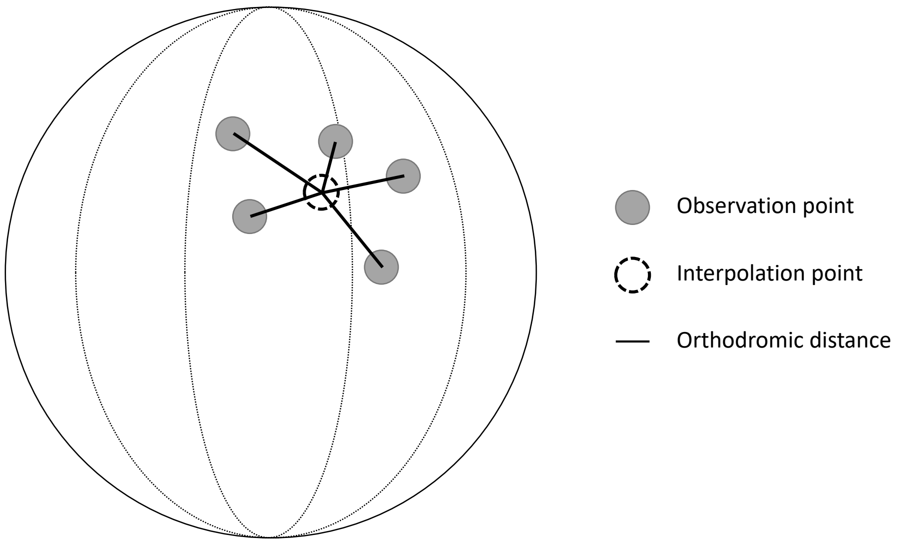

Figure 3.3 Interpolation Neighborhood. The figure shows an interpolation neighborhood with the orthodromic distances between its interpolation coordinates and the ones of each observation neighbor. These distances are then used in the subsequent calculations performed by the NISD step to determine the importance of each neighbor.

Then, neighborhood distribution debiasing (NDD) is applied to deal with biased neighborhood distributions like the one shown in figure 3.4. Thus, NDD is calculated by first finding each observation neighborhood centroid's latitude $\bar{\phi}$ (equation (3.14)) and longitude $\bar{\theta}$ (equation (3.15)).

$$\bar{\phi} = \frac{1}{k}\sum_{i=1}^{k} O_{\mathfrak{N},\phi,i} \tag{3.14}$$

$$\bar{\theta} = \frac{1}{k}\sum_{i=1}^{k} O_{\mathfrak{N},\theta,i} \tag{3.15}$$

Great circle distances between each neighborhood's centroid and each of the neighborhood's observations is then calculated (equation (3.16)), and these distances are used to debias the earlier distance weights w_δ, ending up with neighborhood distribution debiased distance weights w_η (equation (3.17)).

$$\eta = \mathfrak{D}\left(\bar{\phi}, \bar{\theta}, O_{\mathfrak{N},\phi}, O_{\mathfrak{N},\theta}, \rho\right) \tag{3.16}$$

$$w_\eta = \frac{w_\delta \eta}{\sum_{i=1}^{k} w_{\delta,i}\eta_i} \tag{3.17}$$

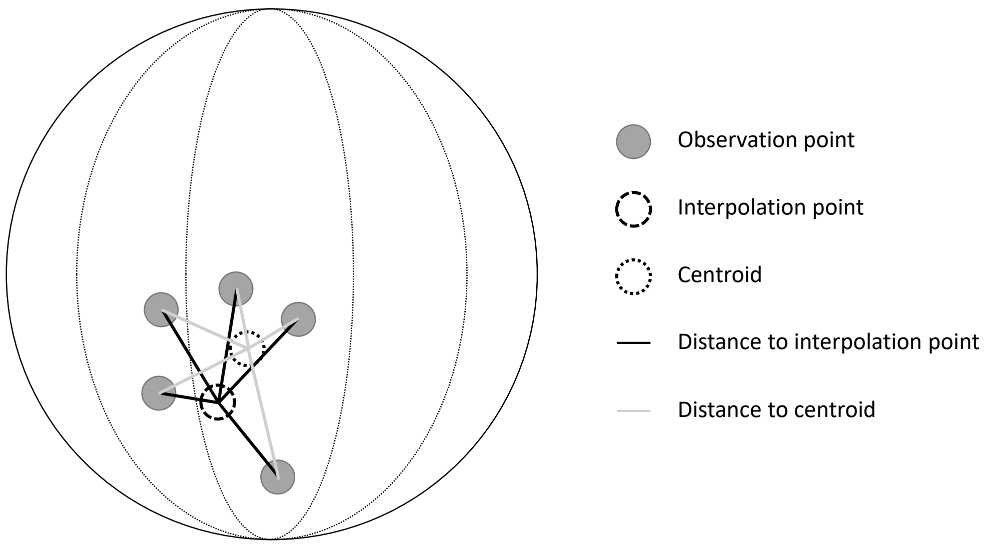

Figure 3.4 Neighborhood Distribution Debiasing. The figure shows a biased interpolation neighborhood. The distances from observations to the neighborhood's interpolation point are also illustrated, as well as distances from observations to the neighborhood's centroid. Intuitively, the more an observation neighbor is close to the neighborhood's centroid, the more it is biased. NDD uses this information to renormalize the distance weights w_δ.

This NDD step is useful since it allows the algorithm to deal with interpolation neighborhoods which have a non-uniform spatial distribution of observations, thus over- and under-representing some values due to the neighborhood distribution's bias. To counteract this bias, NDD calculates each neighborhood's centroid to find where the neighborhood's center of bias is, and then weights down neighbors proportionally to how close they are to the aforementioned center of bias. Once the NDD part complete, the debiased weights w_η are ultimately used to perform a weighted sum for each neighborhood (equation (3.18)).

$$P_{\hat{\nu}} = \mathfrak{I}_{\text{NDDNISD}}\left(O_{\mathfrak{N},\phi}, O_{\mathfrak{N},\theta}, O_{\mathfrak{N},\nu}, P_\phi, P_\theta, \rho, k\right) = \sum_{i=1}^{k} w_{\eta,i} O_{\mathfrak{N}_i,\nu} \tag{3.18}$$

3.6.6 Calculating Great Circe Distances

To calculate great circle distances (also known as orthodromic distances), the orthodrome (shortest path between two points on the surface of a sphere) from each interpolation point to each of its k-nearest observation neighbors is determined and used to compute the geodesic/orthodromic distance (which corresponds to the length of the ohthodrome) [38]. In other words, the shortest distance on the surface of a sphere between two points is calculated, for each pair of points of interest, as illustrated in figure 3.5.

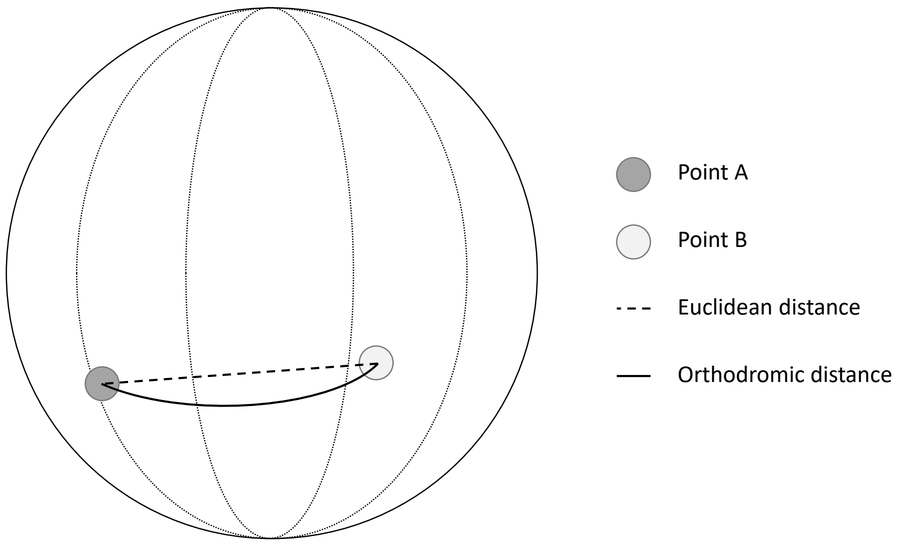

Figure 3.5 Orthodromic Distance. The figure shows the difference between orthodromic distance and Euclidian distance between two points, A and B, on the surface of a sphere.

This great circle distance is calculated as defined in expressions (3.19) to (3.21) using the famous haversine formula [39].

$$a \leftarrow \sin^2\left(\frac{B_\phi - A_\phi}{2}\right) + \cos\left(A_\phi\right)\cos\left(B_\phi\right)\sin^2\left(\frac{B_\theta - A_\theta}{2}\right) \tag{3.19}$$

$$a \leftarrow \min\left(1, \max\left(0, a\right)\right) \tag{3.20}$$

$$\mathfrak{D}\left(A_\phi, A_\theta, B_\phi, B_\theta, \rho\right) = 2\rho \cdot \operatorname{arctan2}\left(\sqrt{a}, \sqrt{1-a}\right) \tag{3.21}$$

The equation receives two points, A and B, with their respective latitude ϕ and longitude θ, and also the sphere's radius ρ, and computes the great circle distance. Note that arctan2 (y, x) is a version of the arctangent of y over x which chooses the correct quadrant correctly, and which operates as described in the NumPy documentation [40]. Also note that expression (3.20) is an extra step that is added to the calculation to ensure numerical stability. With the equations defined, they are now used to compute the great circle distances for the pairs of points of interest. An advantage of this calculation is that it also works in vector and matrix forms simply by using vectors or matrices of same cardinality as inputs instead of scalars, and by applying scalar operators element-wise. In that case, instead of producing a single scalar output, the result is now a vector or matrix with the same cardinality as the input vectors or matrices. Lastly, note that operation and array broadcasting à la NumPy may also be used to achieve the same results simply and efficiently.

3.7 Algorithm Evaluation Methodology

To validate and evaluate the algorithm, the validation methodology presented in figure 3.6 is undertaken. Thus, the idea is to evaluate the accuracy of some of SkNNI's interpolation functions $\mathfrak{I}$ in a way which remains agnostic to the evaluated quantity's nature Υ (e.g. temperature, wind speed, pressure, and even artificially generated weather parameters). As such, the experiments compare various combinations of interpolation functions $\mathfrak{I}$ and numbers of nearest neighbors k by evaluating their interpolation error on observation sets of various natures Υ.

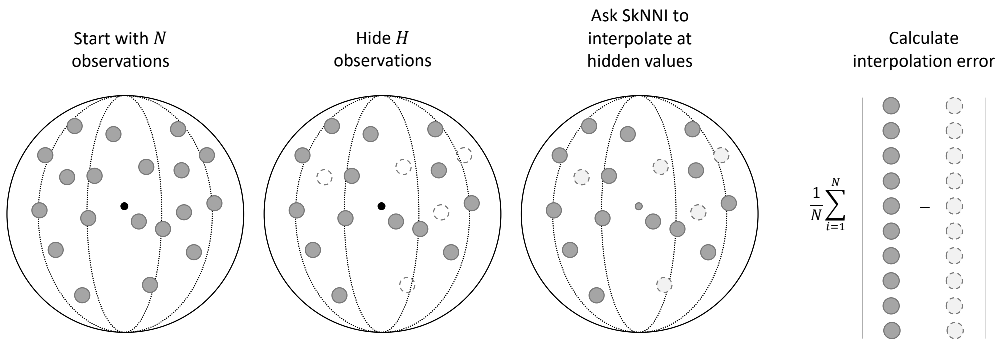
Start with N observations
Hide H observations
Ask SkNNI to interpolate at hidden values
Calculate interpolation error
$\frac{1}{N}\sum_{i=1}^{N}$
–

Figure 3.6 Evaluation Process for SkNNI. The figure shows the evaluation process for SkNNI, which is performed in four main steps. The first step consists in acquiring a set of observation values. The second step then hides a portion of the observations in a holdout set used for evaluation. The third step consists in having SkNNI perform interpolation at the coordinates where values were hidden. The fourth and last step then simply measures the algorithm's accuracy using a given metric.

For each experiment, an observation set of about 4000 elements is collected, 1000 of which are sampled without replacement and used to build a SkNNI interpolator. Note that the sphere radius ρ used for all experiments is the algorithm's default value, which is the Earth's mean radius in kilometers. Then, the remaining observation values are held out for validation and their coordinates are passed as a query to the SkNNI interpolator. Once the SkNNI interpolator built and the holdout set prepared, one interpolation query is made for each configuration pair $(\mathfrak{I}, k) \in \{\mathfrak{I}_{\text{Mean}}, \mathfrak{I}_{\text{Median}}, \mathfrak{I}_{\text{NDDNISD}}, \mathfrak{I}_{\text{Nearest}}\} \times \mathbb{N}_{>0,\leqslant 25}$ and the results and settings are collected for further analysis. As such, equations (3.22), (3.23) and (3.24) show the respective definitions of $\mathfrak{I}_{\text{Mean}}$, $\mathfrak{I}_{\text{Median}}$ and $\mathfrak{I}_{\text{Nearest}}$, where $\mathbb{1}$ is an indicator function which takes the value 1 if its condition is met and the value 0 otherwise. Ultimately, for each quantity nature $\Upsilon \in \left\{\Upsilon_{\text{Synthetic}}, \Upsilon_{\text{Temperature}}, \Upsilon_{\text{DewPoint}}, \Upsilon_{\text{Pressure}}, \Upsilon_{\text{WindSpeed}}\right\}$, 100 different observation sets are gathered and used to produce the results to be further analyzed.

$$\mathfrak{I}_{\text{Mean}}\left(O_{\mathfrak{N},\phi}, O_{\mathfrak{N},\theta}, O_{\mathfrak{N},\nu}, P_\phi, P_\theta, \rho, k\right) = \frac{1}{k}\sum_{i=1}^{k} O_{\mathfrak{N}_i,\nu} \tag{3.22}$$

$$\mathfrak{I}_{\text{Median}}\left(O_{\mathfrak{N},\phi}, O_{\mathfrak{N},\theta}, O_{\mathfrak{N},\nu}, P_\phi, P_\theta, \rho, k\right) = \text{median}_{i=1}^{k}\left(O_{\mathfrak{N}_i,\nu}\right) \tag{3.23}$$

$$\mathfrak{I}_{\text{Nearest}}\left(O_{\mathfrak{N},\phi}, O_{\mathfrak{N},\theta}, O_{\mathfrak{N},\nu}, P_\phi, P_\theta, \rho, k\right) = \sum_{i=1}^{k} \mathbb{1}_{i=1}\, O_{\mathfrak{N}_i,\nu} \tag{3.24}$$

3.7.1 Absolute Maximum Error Ratio Percentage Error

Once all the experiments for a quantity nature completed, the data is aggregated by interpolation function $\mathfrak{I}$ and number of nearest neighbors k, and interpolated values are evaluated using the absolute maximum error ratio percentage error (AMERPE) metric which is as described in equation (3.25).

$$\text{AMERPE}\left(\widetilde{\mathcal{V}}, \widehat{\mathcal{V}}; \mathcal{V}_{\min}, \mathcal{V}_{\max}\right) = \frac{100}{\mathcal{V}_{\max} - \mathcal{V}_{\min}}\left|\widetilde{\mathcal{V}} - \widehat{\mathcal{V}}\right| \tag{3.25}$$

The AMERPE is a variant of absolute error (which is the absolute difference between the ground truth value $\widetilde{\mathcal{V}}$ and the interpolated value $\widehat{\mathcal{V}}$) which applies a normalization coefficient based on the domain extrema, $\mathcal{V}_{\min}$ and $\mathcal{V}_{\max}$, of the input variable $\mathcal{V}$. Note that the condition $\mathcal{V}_{\max} > \mathcal{V}_{\min}$ must hold true. The main idea is to scale the absolute interpolation error as a percentage of the maximal error that could ever be made and use that relative error $\text{AMERPE}\left(\widetilde{\mathcal{V}}, \widehat{\mathcal{V}}; \mathcal{V}_{\min}, \mathcal{V}_{\max}\right) \in [0, 100]$ (a quantity expressed without units of measurement) instead. To expose why such a metric might be useful, consider the following examples. If the input variable varies from 0 to 5 and a prediction of 3 is made when the ground truth is 2, the obtained AMERPE is 20, as shown in equation (3.26), whereas if the input variable varies from 0 to 50 and a prediction of 3 is made when the ground truth is 2, the obtained AMERPE is 2, as shown in equation (3.27).

$$\mathrm{AMERPE}\left(2,3;0,5\right) = \frac{100}{5-0}\left|2-3\right| = 20 \tag{3.26}$$

$$\mathrm{AMERPE}\left(2,3;0,50\right) = \frac{100}{50-0}\left|2-3\right| = 2 \tag{3.27}$$

These toy examples show that, for a same absolute interpolation error of 1 in these cases, the former ends up with a much larger AMERPE since an absolute error of 1 over a range of 5 is much worse than an error of 1 over a range of 50. Thus, use of AMERPE allows for fairer comparison of absolute interpolation errors made on various quantity natures.

3.7.2 Evaluation Data

To evaluate the algorithm and various of its configurations on a data nature Υ for which the underlying generation function is fully known, noisy synthetic observation sets are generated for the data of synthetic nature $\Upsilon_{\text{Synthetic}}$ using expressions (3.28) to (3.33).

$$\Phi \sim \mathcal{C}\left(0,30,-90,90\right) \tag{3.28}$$

$$\Theta_1 \sim \mathcal{C}\left(0,60,-180,180\right) \tag{3.29}$$

$$\Theta_2 \sim \mathcal{S}\left(\langle -125,-75,0,75,100,135\rangle, \langle 0.15,0.15,0.15,0.2,0.2,0.15\rangle\right) \tag{3.30}$$

$$\Theta = \left(\Theta_1 + \Theta_2 + 180\right) \bmod^{+} 360 - 180 \tag{3.31}$$

$$\mathfrak{Z} \sim \mathcal{U}\left(0,8\right) \tag{3.32}$$

$$\mathcal{V} = \mathcal{G}\left(\Phi,\Theta,\mathfrak{T},\mathfrak{Z}\right) = 42\sin\left(\frac{\pi(\Phi+90)}{180}\right) + 7\cos\left(\frac{3}{2}\frac{\pi(\Theta+180)}{180} + \frac{\pi}{12}\mathfrak{T}\right) + \mathfrak{Z} - 25 \tag{3.33}$$

In these expressions, $\mathcal{C}\left(\mu,\sigma,a,b\right)$ represents a truncated normal distribution, $\mathcal{U}\left(a,b\right)$ represents a uniform distribution, $\mathcal{S}\left(A,P\right)$ represents random sampling with replacement from elements of A with probabilities P where the probability of selecting $a_i \in A$ is $p_i \in P \mid \sum_{p_i \in P} p_i = 1$, and $\bmod^{+}$ represents the modulo operator which returns the first positive remainder. The synthetic data generation function $\mathcal{G}$ depends on input latitude Φ, longitude Θ, time $\mathfrak{T}$, and noise $\mathfrak{Z}$. The idea is to use a different time $\mathfrak{T}$ value to create each experiment's observation set, thus simulating a noisy geospatial function evolving through time. The uniform noise $\mathfrak{Z}$ included in the synthetic geospatial function $\mathcal{G}$ enforces a limit on the expected minimal absolute interpolation error which is as shown in lemma 3.1. This limit turns out to be important to analyze and compare interpolation errors on noisy data.

Lemma 3.1. *Let $\mathcal{V}$ be a random variable such that $\mathcal{V} \sim \mathcal{U}(a,b)$, a uniform distribution with lower bound a and upper bound b such that $a < b$. Then, the expected minimal absolute error $\mathbb{E}\left[\left|\widetilde{\mathcal{V}} - \mathbb{E}[\mathcal{V}]\right|\right]$ is equal to $\frac{b-a}{4}$.*

Proof. To prove it, start with the expectation of the uniformly distributed random variable $\mathbb{E}[\mathcal{V}]$, which is used as the best prediction value $\widehat{\mathcal{V}}$ to try.

$$\mathcal{V} \sim \mathcal{U}(a,b) \Longrightarrow \mathbb{E}[\mathcal{V}] = \frac{a+b}{2} \tag{3.34}$$

Define the expectation of the absolute error given the best prediction value to try as follows.

$$\mathbb{E}\left[\left|\widetilde{\mathcal{V}} - \mathbb{E}[\mathcal{V}]\right|\right] \tag{3.35}$$

Then, calculate the expected minimal absolute error by averaging (where the sum is done through integration) over all possible truth values $\widetilde{\mathcal{V}}$.

$$\begin{aligned}
\mathbb{E}\left[\left|\widetilde{\mathcal{V}} - \mathbb{E}[\mathcal{V}]\right|\right] &= \frac{1}{b-a}\int_a^b \left|\widetilde{\mathcal{V}} - \mathbb{E}[\mathcal{V}]\right| d\widetilde{\mathcal{V}} && (3.36)\\
&= \frac{1}{b-a}\left(\int_a^{\mathbb{E}[\mathcal{V}]} \left|\mathbb{E}[\mathcal{V}] - \widetilde{\mathcal{V}}\right| d\widetilde{\mathcal{V}} + \int_{\mathbb{E}[\mathcal{V}]}^b \left|\widetilde{\mathcal{V}} - \mathbb{E}[\mathcal{V}]\right| d\widetilde{\mathcal{V}}\right) && (3.37)\\
&= \frac{1}{b-a}\left(\left(\mathbb{E}[\mathcal{V}]\widetilde{\mathcal{V}} - \frac{\widetilde{\mathcal{V}}^2}{2}\right)\Bigg|_a^{\mathbb{E}[\mathcal{V}]} + \left(\frac{\widetilde{\mathcal{V}}^2}{2} - \mathbb{E}[\mathcal{V}]\widetilde{\mathcal{V}}\right)\Bigg|_{\mathbb{E}[\mathcal{V}]}^b\right) + C && (3.38)\\
&= \frac{1}{b-a}\left(\mathbb{E}^2[\mathcal{V}] - a\mathbb{E}[\mathcal{V}] - b\mathbb{E}[\mathcal{V}] + \frac{a^2}{2} + \frac{b^2}{2}\right) + C && (3.39)\\
&= \frac{1}{b-a}\left(\left(\frac{a+b}{2}\right)^2 - a\frac{a+b}{2} - b\frac{a+b}{2} + \frac{a^2}{2} + \frac{b^2}{2}\right) + C && (3.40)\\
&= \frac{a^2 - 2ab + b^2}{4(b-a)} + C && (3.41)\\
&= \frac{(b-a)^2}{4(b-a)} + C && (3.42)\\
&= \frac{b-a}{4} + C && (3.43)
\end{aligned}$$

□

Analyzing the synthetic function $\mathcal{G}$, its extrema are as calculated in equations (3.44) and (3.45).

$$\min(\mathcal{G}) = 42 \cdot 0 + 7 \cdot -1 + 0 - 25 = -32 \tag{3.44}$$

$$\max(\mathcal{G}) = 42 \cdot 1 + 7 \cdot 1 + 8 - 25 = 32 \tag{3.45}$$

Considering the AMERPE's definition as a scaled absolute error, the extrema of the synthetic function $\mathcal{G}$, and lemma 3.1; the theoretically expected best (minimal) AMERPE is as calculated in equation (3.46).

$$\mathrm{AMERPE}\left(\widetilde{\mathcal{V}}, \mathbb{E}\left[\mathcal{V}\right]; \min(\mathcal{G}), \max(\mathcal{G})\right) = \frac{100}{32 - -32}\frac{8-0}{4} = \frac{25}{8} = 3.125 \tag{3.46}$$

Thus, the lowest interpolation errors on data of synthetic nature $\Upsilon_{\text{Synthetic}}$ generated through $\mathcal{G}$ will tend towards 3.125 because of the 12.5 % noise included in $\mathcal{G}$, which is slightly higher than would be expected in real world datasets (even though some are even noisier than this synthetic function).

3.8 Empirical Results & Analysis

Once the experiments (as defined in §3.7) run and results collected, the data is first aggregated by nature Υ (for each figure), and then by interpolation function $\mathfrak{I}$ and number of nearest neighbors k (for every bar in the figures). The experimental conditions defined in §3.7 ensure high statistical significance by calculating statistics on 300 000 truth-interpolation pairs per bar of every figure, totaling 30 000 000 truth-interpolation pairs per figure.

Figure 3.7 shows experimental results for data of synthetic nature $\Upsilon_{\text{Synthetic}}$. In it, it is possible see that the nearest neighbor interpolation function $\mathfrak{I}_{\text{Nearest}}$ does not benefit from having additional neighbors available to it since it always uses the value of the nearest one, thus not having an optimal k value. Note that the reason why this interpolation function is included in the experiments is because it is a simple and commonly used interpolation function which is well known and meaningful as a comparison baseline. The figure also shows that, for $k = 1$, all evaluated interpolation functions degenerate into nearest neighbor interpolation, which is why they all end up with the same error, statistically. Furthermore, table 3.1 shows that, as k increases from $k = 1$, $\mathfrak{I}_{\text{Mean}}$ and $\mathfrak{I}_{\text{Median}}$ seem to benefit from the information provided by the extra neighbors, up to an optimal value of $k = 9$ for $\mathfrak{I}_{\text{Mean}}$ and of $k = 12$ for $\mathfrak{I}_{\text{Median}}$, after which their error starts to slowly but steadily increase. Interestingly, $\mathfrak{I}_{\text{NDDNISD}}$'s error drops significantly more per neighbor, showing a better use of the information provided by each neighbor, and does not reach an obvious optimal value of $k \in \mathbb{N}_{>0,\leqslant 25}$, suggesting that using even more neighbors would allow the error to keep improving, even though less and less, as it approaches the theoretical limit of 3.125 (see equation (3.46)). Table 3.1 also shows the overall minimal error is achieved by $\mathfrak{I}_{\text{NDDNISD}}$ at $k = 25$ with an AMERPE of 3.814 303 and which is about four times closer to the theoretical optimum than the second-best interpolation function's ($\mathfrak{I}_{\text{Mean}}$ at $k = 9$ with an AMERPE of 5.697 394), putting forth $\mathfrak{I}_{\text{NDDNISD}}$'s robustness to noise in observation values. Along the same line of thought, $\mathfrak{I}_{\text{NDDNISD}}$ achieves errors comparable to other interpolation functions' minimal errors using only two nearest neighbors ($k = 2$), hinting at the importance of considering the locality, proximity and distribution of observation neighbors.

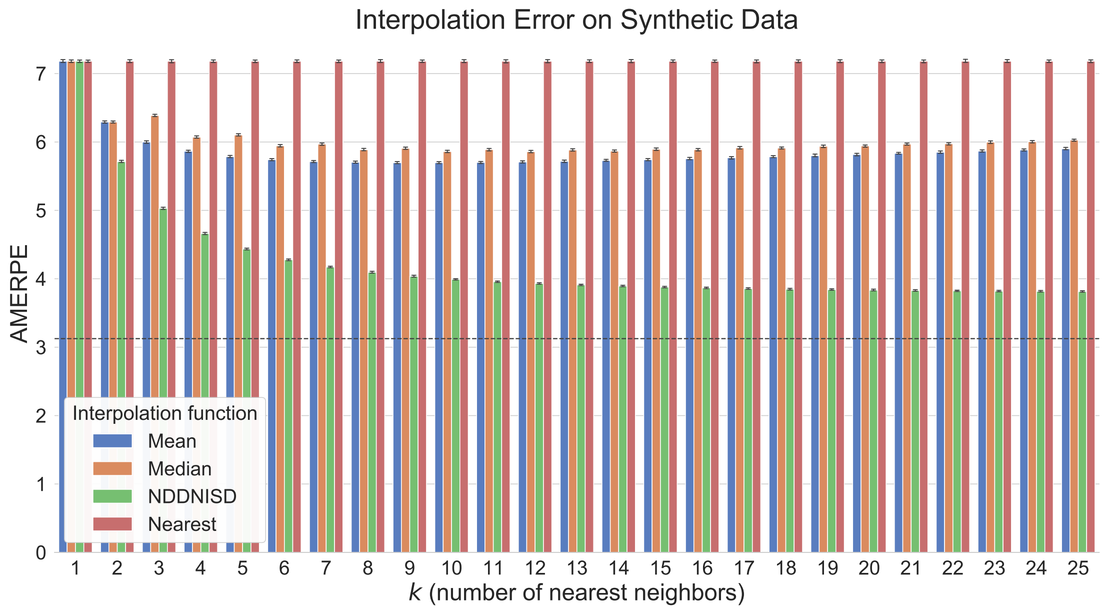

Figure 3.7 Interpolation Error on Synthetic Data. The figure shows the interpolation error of various interpolation functions on noisy synthetic data. 100 experiments were run, each for a different noisy geospatial function, and AMERPE was calculated for various interpolation functions for 25 values of k (number of nearest neighbors) using 1000 observations and 3000 validation observations that were held out for evaluation. The error bars represent the 95 % confidence interval for the bootstrap mean (over 100 bootstrap samples) of the evaluated quantities. The dashed horizontal line represents the best (minimal) error expectation considering the noise included in the geospatial functions.

In figure 3.8, experimental results on real world hourly temperature data are shown. One of the main differences is that $\mathfrak{I}_{\text{Mean}}$ and $\mathfrak{I}_{\text{Median}}$ now end up with a higher error than $\mathfrak{I}_{\text{Nearest}}$ and their error seems to be steadily increasing as k grows. This seemingly odd result may be explained by the consideration of more and more locally impertinent nearest neighbors as k grows, which increases the error. $\mathfrak{I}_{\text{NDDNISD}}$, though, still behaves similarly to the way it does on synthetic data, having its error lower as k grows from $k = 1$, but this time reaches a minimal AMERPE of 2.116 623 at $k = 13$ (as shown in table 3.2), after which its error starts increasing, even though very slightly. The interpolation difficulty as k gets larger could be explained as originating from observation sparsity which would render most further neighbors not only useless, but actually detrimental to the interpolation as their larger distance makes them less locally pertinent up to a point where their values simply become additional noise. Nonetheless, $\mathfrak{I}_{\text{NDDNISD}}$'s error is still significantly lower than the one of the other interpolation functions, suggesting its robustness to observation location sparsity.

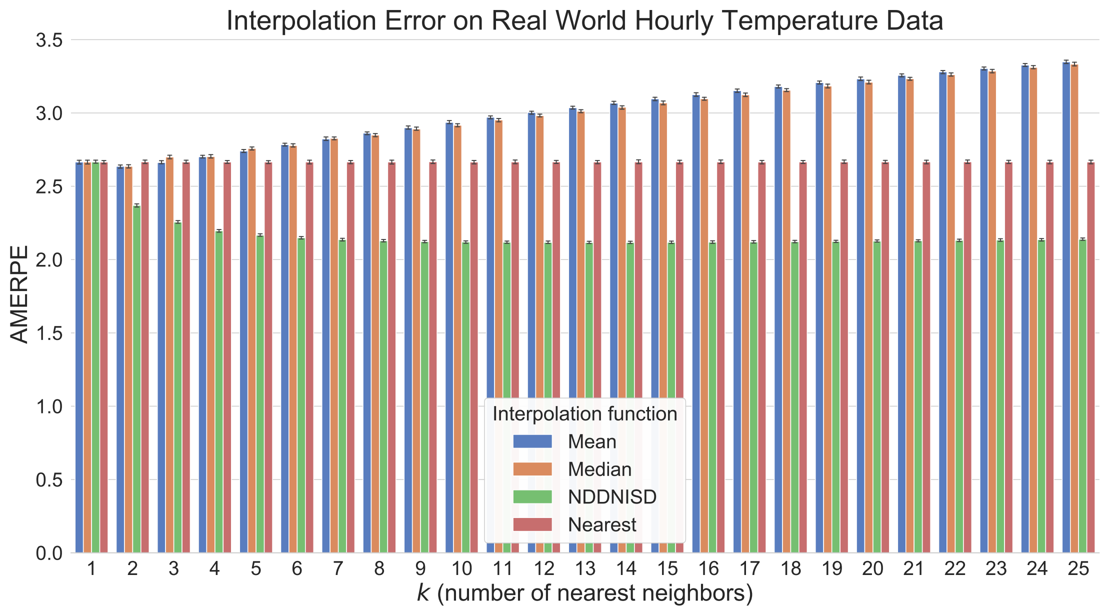

Figure 3.8 Interpolation Error on Real World Hourly Temperature Data. The figure shows the interpolation error of various interpolation functions on real world hourly temperature data. 100 experiments were run, each for a different hourly temperature observation set (sampled from a set of observation sets), and AMERPE was calculated for various interpolation functions for 25 values of k (number of nearest neighbors) using 1000 sampled observations and about 3000 validation observations (the rest) that were held out for evaluation. The error bars represent the 95 % confidence interval for the bootstrap mean (over 100 bootstrap samples) of the evaluated quantities.

Figure 3.9 shows experimental results on real world hourly dew point data. The first point to note is the similarity of these results to the ones on real world hourly temperature data, since temperature and dew point are likely following similar distributions. One thing to note is that, as k grows to larger values, $\mathfrak{I}_{\text{Median}}$ ends up with a significantly lower error than $\mathfrak{I}_{\text{Mean}}$, hinting at observation location sparsity and large observation value variance per interpolation neighborhood, giving the median an advantage over the mean since it is more robust to outliers. $\mathfrak{I}_{\text{NDDNISD}}$ still shows similar behavior, its error diminishing as k grows from $k = 1$, and reaching a minimal AMERPE of 2.043 507 at $k = 10$ (as shown in table 3.3), after which it starts increasing, even though still very slightly. Furthermore, $\mathfrak{I}_{\text{NDDNISD}}$'s error is significantly lower than the one of the other interpolation functions, suggesting its robustness to higher neighborhood observation value variance.

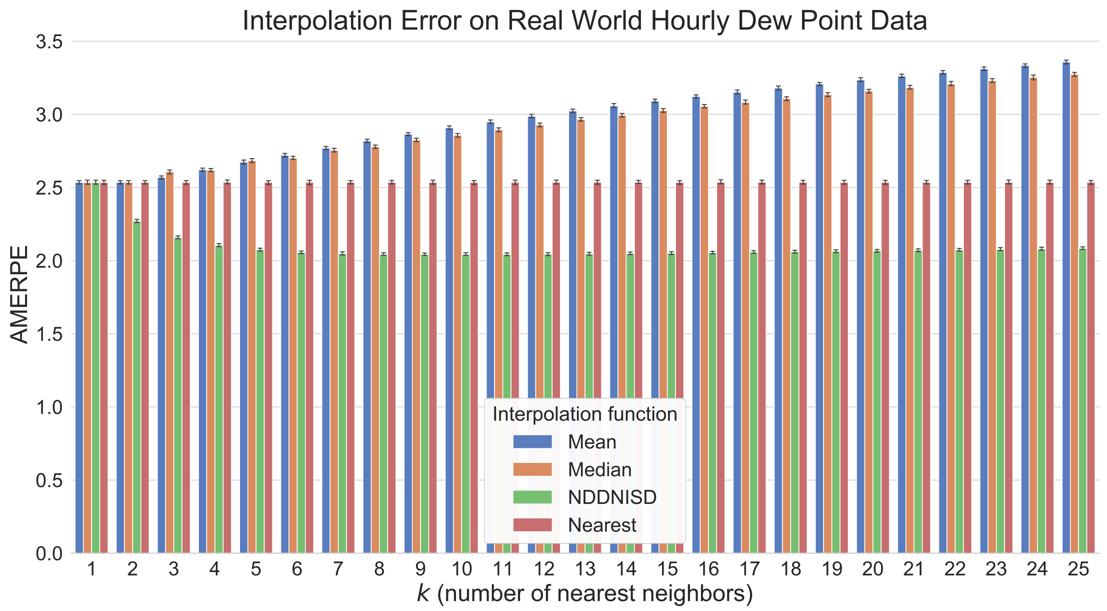

Figure 3.9 Interpolation Error on Real World Hourly Dew Point Data. The figure shows the interpolation error of various interpolation functions on real world hourly dew point data. 100 experiments were run, each for a different hourly dew point observation set (sampled from a set of observation sets), and AMERPE was calculated for various interpolation functions for 25 values of k (number of nearest neighbors) using 1000 sampled observations and about 3000 validation observations (the rest) that were held out for evaluation. The error bars represent the 95 % confidence interval for the bootstrap mean (over 100 bootstrap samples) of the evaluated quantities.

In figure 3.10, experimental results on real world hourly pressure data are shown. The first element to note is the steep and steady rise in error for $\mathfrak{I}_{\text{Mean}}$ and $\mathfrak{I}_{\text{Median}}$ as k grows, which might be explained by the very localized nature of pressure, meaning that pressure values are likely to have relatively high local variance which accentuates the importance of the very near observation neighbors and rapidly decreases the importance of further neighbors. This effect may also be seen in $\mathfrak{I}_{\text{NDDNISD}}$'s error which reaches it optimal k values at $k = 8$ and $k = 9$ (as shown in table 3.4), its lowest optimal k values thus far.

Figure 3.11 shows experimental results on real world hourly wind speed data. A first point to notice is that $\mathfrak{I}_{\text{Mean}}$ and $\mathfrak{I}_{\text{Median}}$ end up with lower errors than $\mathfrak{I}_{\text{Nearest}}$, suggesting that wind speed values vary in a smoother way locally. This result may also be caused by the wind speed value distribution being zero/low-inflated. As shown in table 3.5, the error of

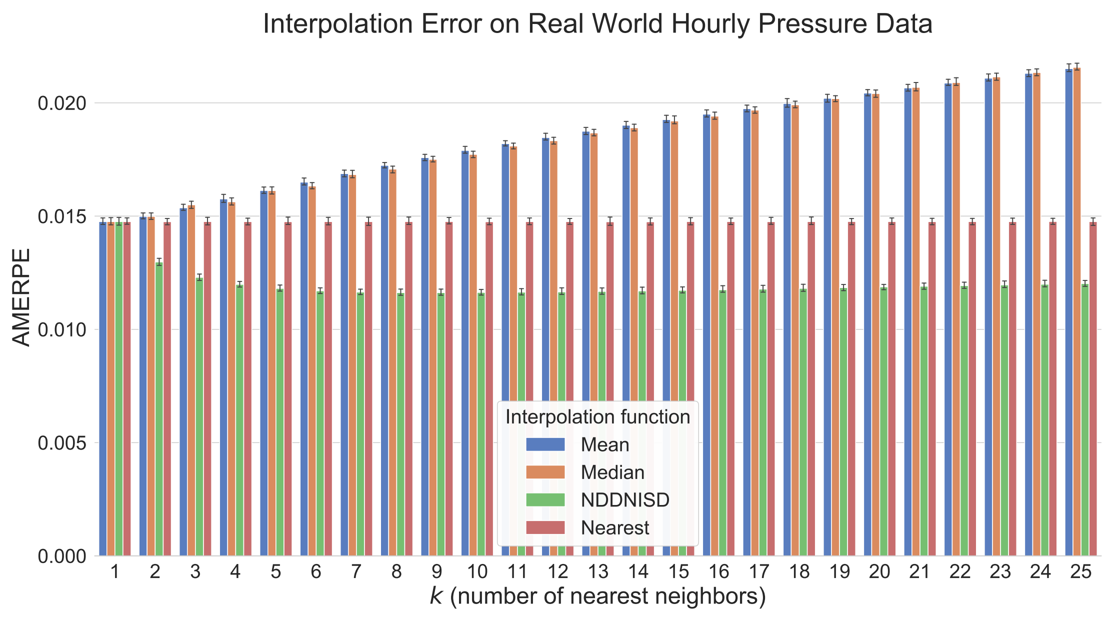

Figure 3.10 Interpolation Error on Real World Hourly Pressure Data. The figure shows the interpolation error of various interpolation functions on real world hourly pressure data. 100 experiments were run, each for a different hourly pressure observation set (sampled from a set of observation sets), and AMERPE was calculated for various interpolation functions for 25 values of k (number of nearest neighbors) using 1000 sampled observations and about 3000 validation observations (the rest) that were held out for evaluation. The error bars represent the 95 % confidence interval for the bootstrap mean (over 100 bootstrap samples) of the evaluated quantities.

$\mathfrak{I}_{\text{Mean}}$ and $\mathfrak{I}_{\text{Median}}$ decreases rapidly for low values of k and quickly reaches an optimum at respectively low k values of $k = 5$ and $k = 6$, after which the error starts slowly but steadily increasing as more and more impertinent neighbors get considered. On its side, $\mathfrak{I}_{\text{NDDNISD}}$ behaves similarly to the way it behaves on synthetic data, its error steadily decreasing as k grows without reaching an obvious optimum such that $k \in \mathbb{N}_{>0,\leqslant 25}$, suggesting a smaller local neighborhood distribution variance, even though potentially accompanied by higher distribution skewness/kurtosis.

Ultimately, SkNNI's $\mathfrak{I}_{\text{NDDNISD}}$ significantly outperforms all other evaluated interpolation functions, both on synthetic and real world data, regardless of distribution variance, skewness, zero inflation and noise, which clearly exposes its interpolation robustness for a variety of data natures and distributions.

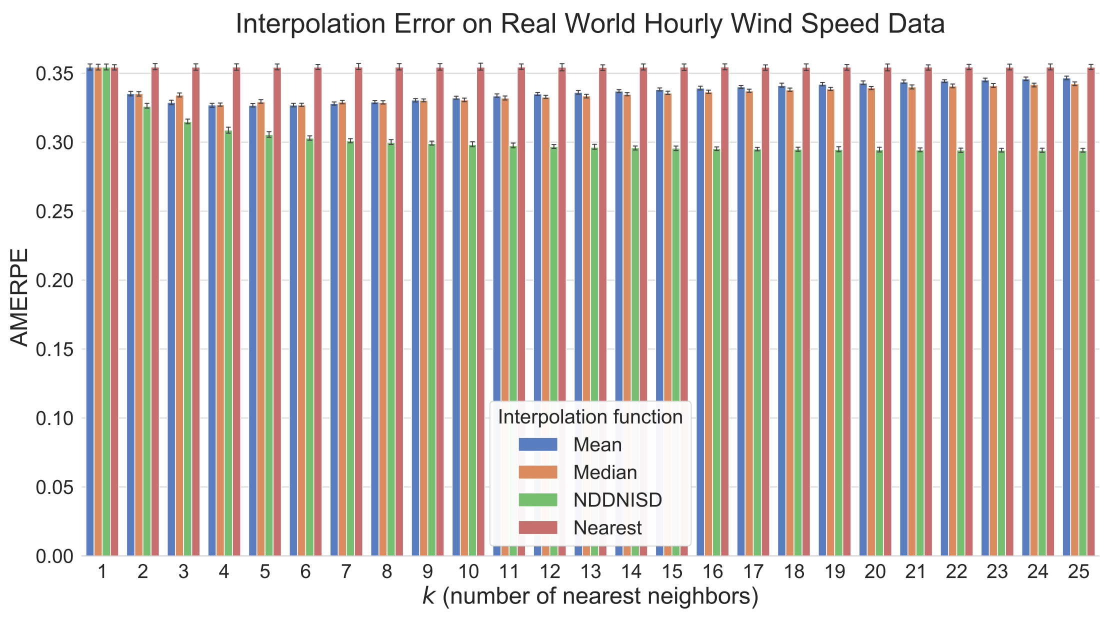

Figure 3.11 Interpolation Error on Real World Hourly Wind Speed Data. The figure shows the interpolation error of various interpolation functions on real world hourly wind speed data. 100 experiments were run, each for a different hourly wind speed observation set (sampled from a set of observation sets), and AMERPE was calculated for various interpolation functions for 25 values of k (number of nearest neighbors) using 1000 sampled observations and about 3000 validation observations (the rest) that were held out for evaluation. The error bars represent the 95 % confidence interval for the bootstrap mean (over 100 bootstrap samples) of the evaluated quantities.

3.9 Implementation

Since its inception, SkNNI has been designed and developed with practical and empirical considerations in mind. Thus, the solution which is used to run the experiments and produce the results is implemented in Python using NumPy and SciPy as its sole external dependencies. To promote transparency, exemplification and extensibility of the software, the SkNNI project is open source and the source code for the Python implementation is available at `https://ptrempe.page.link/sknni`. Thus, users are encouraged to fork the repository to extend the code and build upon it. Of course, the code is also designed and developed with usability in mind. As such, people who only want to use SkNNI as is may easily add the dependency to their Python projects simply by installing SkNNI through PyPI with the following one-liner: `pip install sknni`. Usability and abstraction of the algorithm's complexity for end users being main goals, listing 3.1 shows how simple SkNNI is to use through a brief example.

Listing 3.1 SkNNI Usage Example

```python
import numpy as np

from sknni import SkNNI

if __name__ == '__main__':
    observations = np.array([[30, 120, 20],
                             [30, -120, 10],
                             [-30, -120, 20],
                             [-30, 120, 0]])
    interpolator = SkNNI(observations)
    interp_coords = np.array([[30, 0],
                              [0, -120],
                              [0, 0],
                              [0, 120],
                              [-30, 0]])
    interpolation = interpolator(interp_coords)
    print(interpolation)
    # Output:
    # [[  30.           0.           9.312546]
    #  [   0.        -120.          14.684806]
    #  [   0.           0.          12.5     ]
    #  [   0.         120.          10.315192]
    #  [ -30.           0.          16.464548]]
```

3.10 Contribution

This chapter proposes a novel solution to the problem of efficient sparse spherical interpolation at scale: SkNNI which shines when using the proximity-and-neighborhood-distribution-aware NDDNISD interpolation function. Thus, the main contributions of the solution presented in this chapter are:

- SkNNI, an efficient algorithm for sparse spherical interpolation;
- NDDNISD, a proximity-and-neighborhood-distribution-aware interpolation function;
- a maximal error guarantee for interpolated values;
- `sknni`, an open source Python package also available through PyPI;
- an easy-to-use and comprehensible programmatic user interface.

3.11 Conclusion

Motivated by the need for an algorithm which structures scattered geospatial data, a solution is proposed: the spherical k-nearest neighbors interpolation (SkNNI). Throughout this chapter, each step of the algorithm is explained along with the associated rationale and equations, and various interpolation functions are then evaluated to measure the algorithm's performance on noisy synthetic and real world data. After running the experiments, a key takeaway is that the consideration of many nearest interpolation neighbors is important since they provide additional information, but only up to a point after which they become detrimental. Another key takeaway is that SkNNI's $\mathfrak{I}_{\text{NDDNISD}}$ significantly outperforms all other evaluated interpolation functions thanks to its robustness to interpolation neighborhood distribution variance, skewness, zero inflation and noise. Thus, SkNNI's numerical stability and robustness to noise in the observation data make it suitable for real world applications like data mining, predictive modeling, and data visualization. Hence, this work shows that it is possible to structure scattered geospatial data efficiently through spherical k-nearest neighbors interpolation (SkNNI) while providing an easy-to-use and comprehensible user interface for researchers, engineers, and other practitioners wanting to incorporate it into their data processing pipelines. Ultimately, SkNNI is released in the optic of improving the data science field, allowing for simple interpolation, easy visualization, rich predictive model input provisioning, and much more.

Table 3.1 Interpolation Error on Synthetic Data

k	$\mathfrak{I}_{\text{Mean}}$	$\mathfrak{I}_{\text{Median}}$	$\mathfrak{I}_{\text{NDDNISD}}$	$\mathfrak{I}_{\text{Nearest}}$
1	7.180887	7.180887	7.180887	7.180887
2	6.292216	6.292216	**5.713150**	7.180887
3	6.000302	6.386995	**5.030378**	7.180887
4	5.864746	6.070917	**4.661821**	7.180887
5	5.785400	6.102587	**4.435114**	7.180887
6	5.741767	5.943362	**4.279173**	7.180887
7	5.713855	5.969839	**4.172184**	7.180887
8	5.704050	5.886915	**4.095598**	7.180887
9	5.697394	5.907827	**4.036693**	7.180887
10	5.698507	5.860449	**3.992791**	7.180887
11	5.701431	5.889342	**3.958226**	7.180887
12	5.708400	5.858672	**3.931962**	7.180887
13	5.717604	5.883218	**3.911465**	7.180887
14	5.728999	5.867701	**3.894426**	7.180887
15	5.741268	5.894593	**3.879615**	7.180887
16	5.755385	5.888917	**3.867770**	7.180887
17	5.769680	5.914400	**3.856452**	7.180887
18	5.784062	5.912072	**3.848904**	7.180887
19	5.799974	5.938317	**3.841940**	7.180887
20	5.817045	5.942745	**3.835128**	7.180887
21	5.834190	5.970097	**3.829496**	7.180887
22	5.850272	5.973504	**3.824670**	7.180887
23	5.867263	5.998613	**3.820316**	7.180887
24	5.883805	6.004929	**3.817285**	7.180887
25	5.900620	6.028701	**3.814303**	7.180887
min	5.697394	5.858672	**3.814303**	7.180887
max	7.180887	7.180887	7.180887	7.180887

Table 3.2 Interpolation Error on Real World Hourly Temperature Data

k	$\mathfrak{I}_{\text{Mean}}$	$\mathfrak{I}_{\text{Median}}$	$\mathfrak{I}_{\text{NDDNISD}}$	$\mathfrak{I}_{\text{Nearest}}$
1	2.665625	2.665625	2.665625	2.665625
2	2.636061	2.636061	**2.369657**	2.667078
3	2.662057	2.699771	**2.256119**	2.665861
4	2.701233	2.702824	**2.196502**	2.665602
5	2.740470	2.758294	**2.166395**	2.665686
6	2.784114	2.778144	**2.148715**	2.665751
7	2.823021	2.826885	**2.136740**	2.665983
8	2.862797	2.849402	**2.128386**	2.665781
9	2.899099	2.890800	**2.122914**	2.667346
10	2.935949	2.916202	**2.120297**	2.665583
11	2.970875	2.952571	**2.118820**	2.666155
12	3.003557	2.981707	**2.117649**	2.666342
13	3.035741	3.012049	**2.116623**	2.666025
14	3.067085	3.039471	**2.116854**	2.666124
15	3.096562	3.068821	**2.118038**	2.666754
16	3.125052	3.097134	**2.119352**	2.666136
17	3.152291	3.124024	**2.120373**	2.665670
18	3.180208	3.155309	**2.121920**	2.666228
19	3.207222	3.183880	**2.124083**	2.666197
20	3.232494	3.209806	**2.126432**	2.666571
21	3.256624	3.233558	**2.128680**	2.666834
22	3.280750	3.261104	**2.131506**	2.666941
23	3.303521	3.286242	**2.133911**	2.666331
24	3.326694	3.310927	**2.136222**	2.666163
25	3.348467	3.333161	**2.138759**	2.665903
min	2.636061	2.636061	**2.116623**	2.665583
max	3.348467	3.333161	**2.665625**	2.667346

Table 3.3 Interpolation Error on Real World Hourly Dew Point Data

k	$\mathfrak{I}_{\text{Mean}}$	$\mathfrak{I}_{\text{Median}}$	$\mathfrak{I}_{\text{NDDNISD}}$	$\mathfrak{I}_{\text{Nearest}}$
1	2.535426	2.535426	2.535426	2.535426
2	2.536053	2.536053	**2.271442**	2.536933
3	2.570109	2.608266	**2.158391**	2.535842
4	2.621533	2.619108	**2.105821**	2.536399
5	2.674192	2.684520	**2.075117**	2.536114
6	2.720712	2.701085	**2.057358**	2.536434
7	2.769336	2.755300	**2.048393**	2.536098
8	2.818598	2.779703	**2.044607**	2.536168
9	2.865431	2.825556	**2.043763**	2.536392
10	2.908990	2.857349	**2.043507**	2.535781
11	2.948776	2.895035	**2.044385**	2.536299
12	2.987787	2.928608	**2.044863**	2.536306
13	3.024064	2.966058	**2.046658**	2.536498
14	3.058141	2.995154	**2.050000**	2.536402
15	3.091800	3.027724	**2.052573**	2.535999
16	3.123234	3.055308	**2.055930**	2.536802
17	3.151773	3.082732	**2.058755**	2.536968
18	3.180382	3.108762	**2.061609**	2.536130
19	3.208803	3.135793	**2.064967**	2.535474
20	3.236057	3.160292	**2.068183**	2.535938
21	3.262319	3.184537	**2.071683**	2.536565
22	3.286715	3.208171	**2.074704**	2.536197
23	3.311579	3.230688	**2.077754**	2.536191
24	3.335427	3.253906	**2.080968**	2.535762
25	3.358386	3.275029	**2.084425**	2.536456
min	2.535426	2.535426	**2.043507**	2.535426
max	3.358386	3.275029	**2.535426**	2.536968

Table 3.4 Interpolation Error on Real World Hourly Pressure Data

k	$\mathfrak{I}_{\text{Mean}}$	$\mathfrak{I}_{\text{Median}}$	$\mathfrak{I}_{\text{NDDNISD}}$	$\mathfrak{I}_{\text{Nearest}}$
1	0.014757	0.014757	0.014757	0.014757
2	0.014987	0.014987	**0.012975**	0.014752
3	0.015366	0.015498	**0.012296**	0.014758
4	0.015758	0.015631	**0.011985**	0.014755
5	0.016130	0.016127	**0.011808**	0.014752
6	0.016500	0.016330	**0.011699**	0.014756
7	0.016876	0.016834	**0.011647**	0.014756
8	0.017238	0.017075	**0.011623**	0.014758
9	0.017583	0.017505	**0.011623**	0.014760
10	0.017898	0.017725	**0.011631**	0.014750
11	0.018199	0.018094	**0.011643**	0.014756
12	0.018468	0.018327	**0.011654**	0.014756
13	0.018747	0.018672	**0.011676**	0.014749
14	0.019011	0.018899	**0.011697**	0.014749
15	0.019260	0.019200	**0.011724**	0.014759
16	0.019501	0.019413	**0.011749**	0.014758
17	0.019743	0.019681	**0.011779**	0.014755
18	0.019971	0.019913	**0.011810**	0.014758
19	0.020201	0.020186	**0.011843**	0.014761
20	0.020432	0.020411	**0.011875**	0.014759
21	0.020658	0.020684	**0.011905**	0.014754
22	0.020874	0.020897	**0.011933**	0.014758
23	0.021090	0.021146	**0.011961**	0.014756
24	0.021302	0.021339	**0.011993**	0.014762
25	0.021511	0.021567	**0.012023**	0.014757
min	0.014757	0.014757	**0.011623**	0.014749
max	0.021511	0.021567	**0.014757**	0.014762

Table 3.5 Interpolation Error on Real World Hourly Wind Speed Data

k	$\mathfrak{I}_{\text{Mean}}$	$\mathfrak{I}_{\text{Median}}$	$\mathfrak{I}_{\text{NDDNISD}}$	$\mathfrak{I}_{\text{Nearest}}$
1	0.354403	0.354403	0.354403	0.354403
2	0.335207	0.335207	**0.325974**	0.354427
3	0.328778	0.334170	**0.314835**	0.354389
4	0.326969	0.327124	**0.308738**	0.354506
5	0.326677	0.329275	**0.305537**	0.354306
6	0.326898	0.327010	**0.302942**	0.354396
7	0.328020	0.329073	**0.301002**	0.354475
8	0.329175	0.328735	**0.299813**	0.354419
9	0.330543	0.330327	**0.298995**	0.354358
10	0.332047	0.330648	**0.298228**	0.354362
11	0.333515	0.331980	**0.297561**	0.354497
12	0.334981	0.332697	**0.296825**	0.354347
13	0.336048	0.333577	**0.296298**	0.354203
14	0.336997	0.334670	**0.295942**	0.354409
15	0.338150	0.335601	**0.295647**	0.354487
16	0.339171	0.336448	**0.295299**	0.354537
17	0.340138	0.337216	**0.295078**	0.354316
18	0.341059	0.337959	**0.294868**	0.354291
19	0.341956	0.338635	**0.294705**	0.354416
20	0.342809	0.339416	**0.294517**	0.354416
21	0.343650	0.340104	**0.294406**	0.354468
22	0.344366	0.340731	**0.294263**	0.354452
23	0.345121	0.341029	**0.294155**	0.354366
24	0.345912	0.341870	**0.294046**	0.354432
25	0.346618	0.342357	**0.293959**	0.354370
min	0.326677	0.327010	**0.293959**	0.354203
max	0.354403	0.354403	0.354403	0.354537

CHAPTER 4 MULTIDIMENSIONAL WORLDWIDE WEATHER FORECASTING BY DEEP LEARNING

This chapter focuses on the problem of automatic multi-weather-parameter worldwide weather forecasting by deep learning. In it, challenges and ideas related to the topic are discussed, and a novel deep neural network architecture is presented to show how preprocessed and structured initially scattered geospatial big data may be used to train a model to make accurate and insightful worldwide forecasts automatically and without prior weather-related knowledge.

4.1 Introduction

Weather forecasting has remained an interesting problem for years due to its inherent complexity and high impact on humanity and its infrastructures. Weather systems being this complex, there is no simple way of expressing them and of making very accurate forecasts, especially far into the future [41, 42]. Even though long-term forecasting proves to be a challenge, the higher collection and availability of meteorological data allow for the creation and training of more accurate models. Earlier weather forecasting models were mostly theoretical and equations were implemented and calculated as is. Later models started leveraging collected weather data to compute statistics and other predefined metrics, and used these to compute their forecast. Over the last few years, because of the much higher data availability and advances in the field of machine intelligence, researchers have started leveraging weather data to train models using a wide variety of optimization algorithms [13, 14, 16]. This trend culminated in many recent breakthroughs like the ones discussed in chapter 2. The main limitation nowadays often not being data anymore but the problem formulation and model architecture design for efficient and accurate model training instead, chapter 3 of this work presents such a way of organizing and structuring globally scattered geospatial data in that way which makes it useful for a trainable model to learn from. Thus, this chapter aims to provide an example of multi-weather-parameter worldwide weather forecasting deep learning model architecture and show its relevance with the ultimate goal of laying groundwork in the field and inspiring others to build upon it to provide ever more accurate weather forecasts to help people plan ahead and live safely.

4.1.1 Problem Definition

The problem is defined similarly to a classic weather forecasting problem, meaning the model receives a sequence of weather system states as input and its task is to predict the next one. The main differences with the proposed problem formulation are spatiotemporality and working with multi-channel weather data, since the inputs and predictions are worldwide for all weather parameters.

Thus, a geospatial frame is defined as a multi-channel geospatial map (of latitudinal resolution $|\Phi|$ and longitudinal resolution $|\Theta|$) containing C channels. The model's input $\tilde{x} \in \mathbb{R}^{B \times T \times |\Phi| \times |\Theta| \times C}$ is defined as a batch (of length B) of time-contiguous sequences (of length T) of geospatial frames (of shape $\big(|\Phi|, |\Theta|, C\big)$), and the model's output $\hat{y} \in \mathbb{R}^{B \times |\Phi| \times |\Theta| \times C}$ as a batch (of length B) of geospatial frames (of shape $\big(|\Phi|, |\Theta|, C\big)$) which correspond to the temporally next frames to be predicted.

To perform inference (prediction), the model $\mathcal{M}$ with trainable parameters θ is given an input batch $\tilde{x}$ and predicts an output batch $\hat{y}$ using forward propagation as shown in equation (4.1).

$$\hat{y} = \mathcal{M}\left(\tilde{x}; \theta\right) \tag{4.1}$$

Thus, the model training process's goal is to find optimal trainable parameter values θ^* that minimize a loss function $\mathcal{L}\left(\tilde{y}, \hat{y}\right)$ which measures the model's prediction $\hat{y}$ error with respect to given ground truth $\tilde{y}$.

4.2 Background and Related Work

This section discusses important external contributions and topics related to this work. In it, key notions to better understand this chapter and the context in which it unfolds are presented.

4.2.1 Physics Simulations

This first section briefly addresses physics simulations, since they are used extensively for weather forecasting. Such simulations perform very well when the equations used to model the problem are accurate and encompass a very large proportion of the underlying reality.

Unfortunately, this is not always the case and many of these models end up with a multitude of parameters to manually tune. As such, meteorologists have quickly realized that some models perform better for some parts of the world and worse for others, and better during a specific season than during others. The problem is they have to keep track of a lot of information, and of various models, on top of having to tune every single one of them. Since such a process rapidly becomes tedious, most weather forecasts are no more than national.

About aforementioned fastidious manual or semi-automatic tuning, some researchers have started investigating ways of automating the tuning of differential equations using deep learning [43] and this could prove to be a very interesting avenue for future research. Though, since this work aims to forecast weather automatically and only through data, no prior information about weather or weather systems should be incorporated into the model, i.e. the model shall learn meteorology and weather forecasting only through data.

4.2.2 Spherical Convolutional Neural Networks

The spherical ConvNet is a kind of ConvNet which operates by performing spherical convolutions which are practically implemented as spherical cross-correlations using the fast Fourier transform [44]. As such, it is useful for spherical-rotation-invariant classification and regression problems.

Thus, such models are expected to perform well on inputs like spherical images, e.g. having a model determine the distance to the closest object based on a spherical image provided by a drone. This example case is spherical-rotation-invariant, because rotating the spherical image does not influence the closest object distance metric to be predicted by the model.

Since the worldwide weather forecasting problem this work aims to solve is spherical-rotation-variant, spherical ConvNets cannot be used, though they would be very interesting to consider if some kind of spherical deconvolution is ever developed.

4.2.3 Residual Neural Networks and Functions

Residual neural networks are based on the consideration that, as neural networks get deeper and use many non-linearities, it becomes difficult for them to learn the identity function, and these very deep models tend to suffer from exploding and vanishing gradient problems [45]. The proposed solution makes clever use of residual functions which are as shown in equation (4.2).

$$\mathcal{H}(x) = \mathcal{F}(x) + x \tag{4.2}$$

In the equation, x represents an input, $\mathcal{H}(x)$ is the hidden mapping to optimize and $\mathcal{F}(x)$ is the residual mapping which is optimized instead. The addition of x is incorporated into models as a shortcut connection (also called a skip connection) and allows for model optimization as usual. The authors of [45] hypothesize that it might be easier for models to optimize the residual mapping than the full hidden mapping. They then show that using residual functions allows for the training of very deep neural networks without encountering the aforementioned problems. Since the publication of their findings, residual neural networks have been used to achieve state-of-the-art results for a variety of tasks and challenges.

4.2.4 Adaptive Moment Estimation Optimizer

Adaptive moment estimation (Adam) is a gradient-based optimizer which uses adaptive moment estimation to compute trainable parameter updates [46]. Adam is a very efficient optimizer based on the ideas used in AdaGrad [47] and RMSProp [48]. It has appealing properties like not requiring a stationary objective and yielding parameter updates which are robust to sparse gradients and gradient rescaling. A typical trainable parameter update is as shown in expression (4.3).

$$\theta \leftarrow \text{Adam}\Big(\nabla_\theta \mathcal{L}\left(\tilde{y}, \mathcal{M}\left(\tilde{x};\theta\right)\right), \theta; \alpha, \beta_1, \beta_2, \epsilon\Big) \tag{4.3}$$

In this expression, $\nabla_\theta \mathcal{L}\left(\tilde{y}, \mathcal{M}\left(\tilde{x};\theta\right)\right)$ is the gradient of the loss $\mathcal{L}$ with respect to the model's trainable parameters θ; and α, β_1, β_2 and ϵ are hyperparameters used to tune the optimizer. Though, in practice; β_1, β_2 and ϵ are often left to their default values which are 0.9, 0.999 and 10^{-8}, respectively. Since its publication, the Adam optimizer has been implemented in many, if not all, major deep learning frameworks (e.g. TensorFlow [49], Keras [50], PyTorch [51], Caffe [52], Lasagne [53], Theano [54], CNTK [55], MxNet [56], etc.) and has seen a lot of use due to its simplicity and efficiency, often helping researchers achieve state-of-the-art results.

4.3 Approach and Methodology

The following sections detail the undertaken approach and methodology. Are hereafter presented the data organization, model architecture, evaluation metrics, and training process.

4.3.1 Data and Datasets

The original data source for this work's experiments is the proprietary dataset mentioned in §1.4, which is constituted of weather data collected at various locations scattered around the world.

This data is first processed using S*k*NNI to transform the time-aggregated scattered values into geospatial weather maps (where all values are assumed to be at sea level). The geospatial weather maps corresponding to the same time step are then stacked to form geospatial frames.

Time-contiguous geospatial frames are then stacked to form time series of multi-channel worldwide weather states. These data transformations are illustrated in figure 4.1. Each time series is then converted into an example composed of an input part $\tilde{x}$ which contains all of the sequence except the last time step, and an output part $\tilde{y}$ which contains the last time step (which is the prediction target).

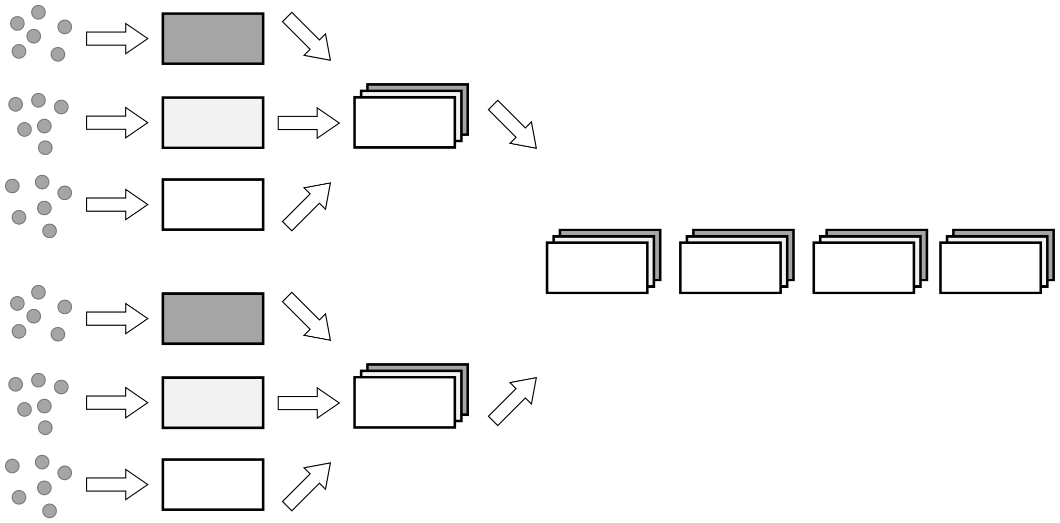

Figure 4.1 Data Preprocessing for Multidimensional Worldwide Weather Forecasting. The figure shows various transformations used to prepare the data for multidimensional worldwide weather forecasting. From left to right, the first step uses S*k*NNI to transform scattered geospatial observations into geospatial frame channels. The next transformation stacks geospatial frame channels into complete geospatial frames. The last step stacks geospatial frames into time-contiguous sequences.

All of these examples ultimately form the dataset $\mathcal{D}$ which is split (using the training split $\mathcal{S}_{\text{Training}}$, validation split $\mathcal{S}_{\text{Validation}}$ and test split $\mathcal{S}_{\text{Test}}$ defined in table 4.1) into three mutually exclusive datasets: the training set $\mathcal{D}_{\text{Training}}$, the validation set $\mathcal{D}_{\text{Validation}}$ and the test set $\mathcal{D}_{\text{Test}}$, as shown in equations (4.4) and (4.5) (where $\mathcal{S}$ denotes the dataset splitting function).

$$\mathcal{S}_{\text{Training}} + \mathcal{S}_{\text{Validation}} + \mathcal{S}_{\text{Test}} = 1 \tag{4.4}$$

$$\left(\mathcal{D}_{\text{Training}}, \mathcal{D}_{\text{Validation}}, \mathcal{D}_{\text{Test}}\right) = \mathcal{S}\left(\mathcal{D}, \mathcal{S}_{\text{Training}}, \mathcal{S}_{\text{Validation}}, \mathcal{S}_{\text{Test}}\right) \tag{4.5}$$

Table 4.1 Hyperparameters

Hyperparameter	Value
Input sequence length	12
Sequence time step duration	$\leqslant 1\,\mathrm{h}$
Latitudinal resolution	90
Longitudinal resolution	180
Training split	70 %
Validation split	15 %
Test split	15 %
Optimizer	Adam
Learning rate	10^{-3}
Number of trainable parameters	300 000

4.3.2 Model Architecture

Before presenting the model architecture used in this work, it is important to mention that a variety of model architectures are capable of operating with the problem formulation defined in this work. For example, when considering time series forecasting, recurrent neural network architectures like LSTMs are prominent candidates. In this case, a variant of LSTMs, namely the ConvLSTM, could be used to work with the aforementioned problem formulation. Other model architectures, like residual or non-residual ConvNets could also be used in conjunction with reshaping operations to work with the same problem formulation. Of course, when considering custom model architectures which are sometimes adapted for low latency, low memory usage, low power usage and the like, the possibilities become almost limitless.

As such, this work does not attempt to create a large model with hundreds of millions (or even billions) of parameters, which would require enormous computational resources, though that would make for very interesting future work. Instead, the proposed model architecture is mainly meant as a proof of concept for the aforementioned formulation with the goal of showing its practicality and potential.

Thus, the proposed model architecture, named DeltaNet, is a deep neural network based on the use of residual delta reduction units (ResDRUs) (see figure 4.2 for architectural details) which are a kind of residual block. Each block is a hyperparameterizable group of potentially trainable layers. Conceptually, each ResDRU computes a variation (the delta) and hidden representations (using the $\mathcal{H}$ transforms) based on the time steps it receives, and then merges aforementioned hidden representations through stacking before compacting them using the reduction transform $\mathcal{R}$. As such, the ResDRU is designed to receive two inputs: the last time step of a sequence t_{n-1} and an anterior time step t_{n-L} of the same sequence. It then carries on the computation presented in equations (4.6) to (4.11).

$$\Delta_t = t_{n-1} - t_{n-L} \tag{4.6}$$

$$h_{\Delta_t} = \mathcal{H}_{\Delta_\mathrm{t}}\left(\Delta_t\right) \tag{4.7}$$

$$h_{t_{n-1}} = \mathcal{H}_{\mathrm{t_{n\text{-}1}}}\left(t_{n-1}\right) \tag{4.8}$$

$$h_{t_{n-L}} = \mathcal{H}_{\mathrm{t_{n\text{-}L}}}\left(t_{n-L}\right) \tag{4.9}$$

$$h_{\Delta} = \mathcal{R}\left(\left[h_{\Delta_t}, h_{t_{n-1}}, h_{t_{n-L}}\right]\right) \tag{4.10}$$

$$\hat{t_n} = t_{n-1} + h_{\Delta} \tag{4.11}$$

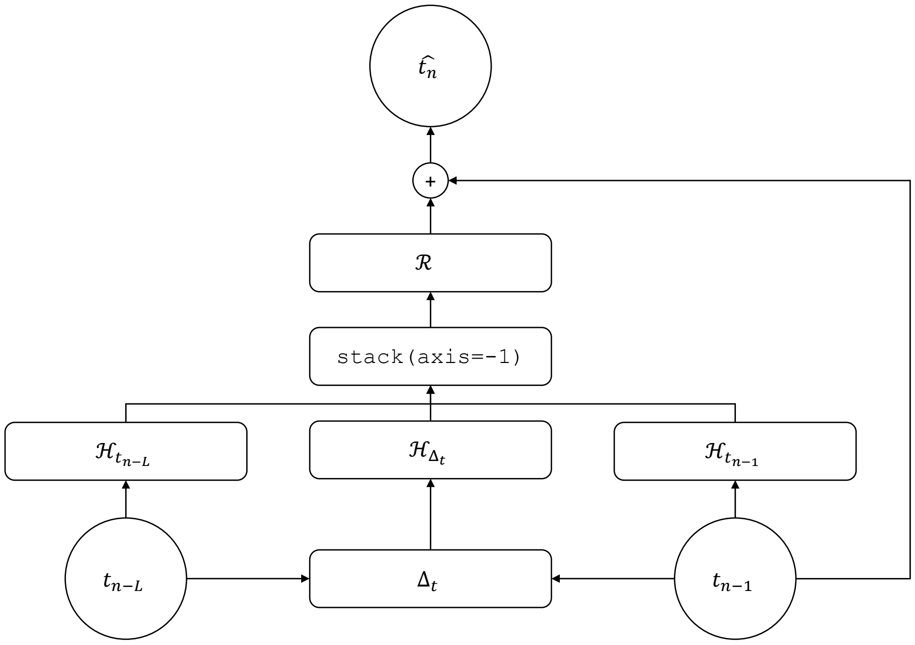

Figure 4.2 ResDRU Architecture. The figure illustrates the architecture of a ResDRU block with its inputs, inner operations, residual connection and output.

Thus, the ResDRU first computes the difference Δ_t between the last time step t_{n-1} and the anterior time step t_{n-L}. It then computes hidden representations for Δ_t, t_{n-1} and t_{n-L} using the $\mathcal{H}_{\Delta_t}$, $\mathcal{H}_{t_{n-1}}$ and $\mathcal{H}_{t_{n-L}}$ transforms, respectively. The hidden representations are then stacked along a new last axis and reduced using the $\mathcal{R}$ transform. Note that the transforms are conceptually generic and may be realized in many ways as long as compatible tensor shapes and types are used. In this work, the $\mathcal{H}$ transforms are realized using two-dimensional 3×3 convolutions with "same" padding, and the $\mathcal{R}$ transform is realized as a single-channel three-dimensional pointwise (1×1) convolution with "same" padding followed by last dimension squeezing. Once the delta reduction unit (DRU) part computed, the residual skip connection is incorporated by adding the last time step t_{n-1} to the reduced representation. Along the same line of thought, equations (4.12) to (4.14) show various representations of the ResDRU which are all equivalent by definition.

$$\hat{t_n} = t_{n-1} + h_\Delta \tag{4.12}$$

$$\overset{\Delta}{\iff} \hat{t_n} = t_{n-1} + \text{DRU}\left(t_{n-1}, t_{n-L}\right) \tag{4.13}$$

$$\overset{\Delta}{\iff} \hat{t_n} = \text{ResDRU}\left(t_{n-1}, t_{n-L}\right) \tag{4.14}$$

Regarding the DeltaNet model (see figure 4.3 for architectural details), it starts with an input $\tilde{x} \in \mathbb{R}^{B\times T\times|\Phi|\times|\Theta|\times C}$ and first applies a channel normalization step as described in equation (4.15).

$$\mathcal{C}\left(c; c_{\min}, c_{\max}\right) = 2\frac{c - c_{\min}}{c_{\max} - c_{\min}} - 1 \tag{4.15}$$

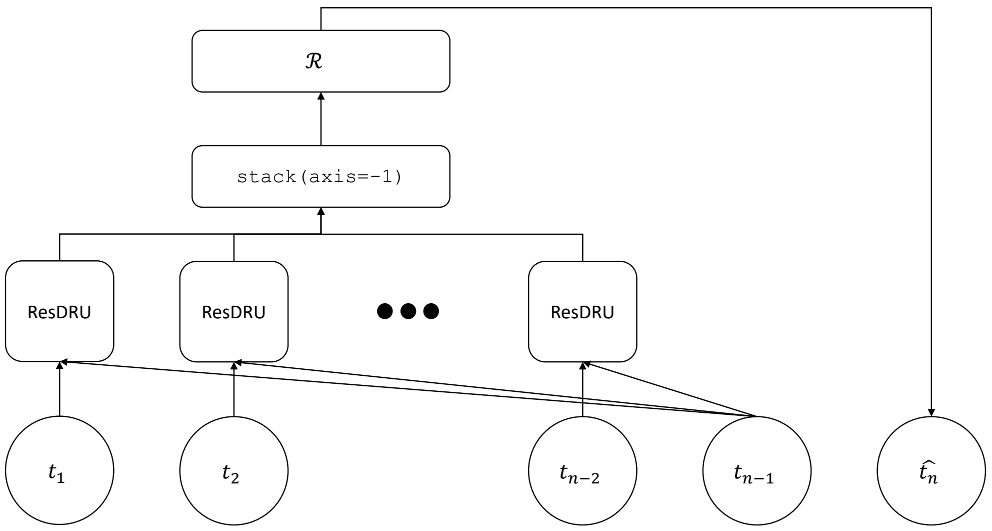

Figure 4.3 DeltaNet Architecture. The figure illustrates the architecture of the DeltaNet model with its input sequence of multidimensional weather states, inner ResDRUs, consensual reduction step, and output.

This normalization effectively transforms each channel's values so that they lie in $[-1, 1]$. The model then applies a ResDRU to each of the T time steps of the input sequence, except the last one. The intuition is that each ResDRU is computing an estimate of the target time step $\hat{t_n}$ based on the variation between the last time step t_{n-1} and its given earlier time step t_{n-L}. Once every ResDRU has made its estimation, they are all processed (stacked along a new last axis and then reduced through pointwise-convolutional compaction of the aforementioned last axis and squeezing) into a form of consensual representation. A final channel restoration step is then performed as described in equation (4.16).

$$\mathcal{C}^{-1}\left(c; c_{\min}, c_{\max}\right) = \frac{c+1}{2}\left(c_{\max} - c_{\min}\right) + c_{\min} \tag{4.16}$$

This restoration step brings each channel's values back to their original range such that they lie in $[c_{\min}, c_{\max}]$. Thus, the output of the channel restoration step becomes the model's output prediction $\hat{y}$.

4.3.3 Model Evaluation and Metrics

To evaluate the model's performance on the datasets, a loss function $\mathcal{L}(\tilde{y}, \hat{y})$ and metrics are defined. The idea is to first compute channel-wise metrics like mean absolute error, and then use a function of these as the loss. Furthermore, since some parts of the world are very well covered with weather stations and others are not, a binary weather station presence mask is used to segment the metrics into two subsets which allow for more insightful optimization and analysis. The binary weather station presence mask $\Omega \in \{0,1\}^{|\Phi| \times |\Theta|}$ is a gridded world map, of which each cell is set to one if it contains at least one weather station, and to zero otherwise. Such a mask is useful to allow fine-tuning of the loss, for example to penalize the model more for errors where there are weather stations, since these places are more likely to be important to humanity. Such a mask may also be useful to analyze how interpolated values where there are no weather stations affect the prediction abilities of the model compared to where there are. Thus, for each channel, the channel mean absolute error (CMAE) for locations with weather stations (WS; see equation (4.17)) and locations with no weather stations (NWS; see equation (4.18)) are computed.

$$\mathrm{CMAE}_{\mathrm{WS}}\left(\tilde{y}_c, \hat{y}_c\right) = \mathrm{MAE}\left(\tilde{y}_c \odot \Omega, \hat{y}_c \odot \Omega\right) \tag{4.17}$$

$$\mathrm{CMAE}_{\mathrm{NWS}}\left(\tilde{y}_c, \hat{y}_c\right) = \mathrm{MAE}\left(\tilde{y}_c \odot (1-\Omega), \hat{y}_c \odot (1-\Omega)\right) \tag{4.18}$$

In these equations, $\odot$ denotes the element-wise product of the binary mask with the prediction or target along the latitudinal and longitudinal axes. Building on top of these metrics, the loss $\mathcal{L}$ (also known as the objective function) is as defined in equation (4.19).

$$\mathcal{L}\left(\tilde{y}, \hat{y}\right) = \sum_{c=1}^{C}\left(\frac{\beta_{c,\mathrm{WS}}\,\mathrm{CMAE}_{\mathrm{WS}}\left(\tilde{y}_c, \hat{y}_c\right) + \beta_{c,\mathrm{NWS}}\,\mathrm{CMAE}_{\mathrm{NWS}}\left(\tilde{y}_c, \hat{y}_c\right)}{c_{i,\max} - c_{i,\min}}\right) \tag{4.19}$$

This equation shows the loss is computed by summing over sublosses, one per channel, where each subloss is the sum of weighted normalized CMAEs. Regarding weighting, the βs are hyperparameters used to adjust the weight of each CMAE. If no particular weighting is desired, all βs may simply be left to their default value of one. The loss $\mathcal{L}$ now defined, the optimization process's goal is thus to find optimal model parameters θ^* that minimize the loss $\mathcal{L}$.

Furthermore, to evaluate DeltaNet's performance, since there exists no model which uses the novel problem formulation presented in this work, DeltaNet is compared to two baseline models. The first one is the persistence model which computes its output as described in equation (4.20).

$$\hat{y} = \hat{t_n} = t_{n-1} \tag{4.20}$$

This baseline model is expected to be the hardest one to outperform due to the relatively short time step duration used (see table 4.1). The second one is the naive linear model which computes its output as described in equation (4.21).

$$\hat{y} = \hat{t_n} = t_{n-1} + (t_{n-1} - t_{n-2}) \tag{4.21}$$

To prove its potential, DeltaNet must outperform these two baseline models for at least one weather parameter, and ideally many.

4.3.4 Model Training

To train the model, the training set $\mathcal{D}_{\text{Training}}$ is used to get batches of inputs $\tilde{x}$ and output targets $\tilde{y}$. Note that this work uses the term "batch" (and not "mini-batch") to refer to groups of examples. During training, input batches are fed to the model $\mathcal{M}$ with trainable parameters θ which runs its forward pass and makes a prediction $\hat{y}$. The ground truth $\tilde{y}$ and model prediction $\hat{y}$ are then used to compute the loss $\mathcal{L}\left(\tilde{y}, \hat{y}\right)$. Then, an optimizer $\mathcal{O}$ (with hyperparameters $\theta_{\mathcal{O}}$), in this case the Adam optimizer (with its default hyperparameter values), computes the gradient of the loss with respect to the model's trainable parameters and uses it to update the model's trainable parameters θ as shown in expression (4.22).

$$\theta \leftarrow \mathcal{O}\left(\nabla_\theta \mathcal{L}\left(\tilde{y}, \mathcal{M}\left(\tilde{x}; \theta\right)\right), \theta; \theta_{\mathcal{O}}\right) \tag{4.22}$$

The training process ends when a termination condition (like reaching a given number of epochs/iterations or not seeing any improvement on the validation set for a given number of epochs/iterations) is met.

4.4 Empirical Results and Analysis

This section presents empirical results and discusses them to determine how DeltaNet fairs against the persistence and naive linear baseline models in terms of prediction accuracy.

Figure 4.4 shows model prediction experimental results for temperature. In it, DeltaNet proves to significantly outperform both persistence (by a small yet clear margin) and naive linear (by a large margin) models. Comparing the results where there are weather stations (darker portion of the bars) and where there are not (lighter portion of the bars), one may observe the noticeably larger confidence intervals hinting at larger error variances where there are no weather stations than where there are. This phenomenon may be explained by the interpolation inducing a larger variance (both on interpolated observations and model predictions) due to its inherent approximating nature.

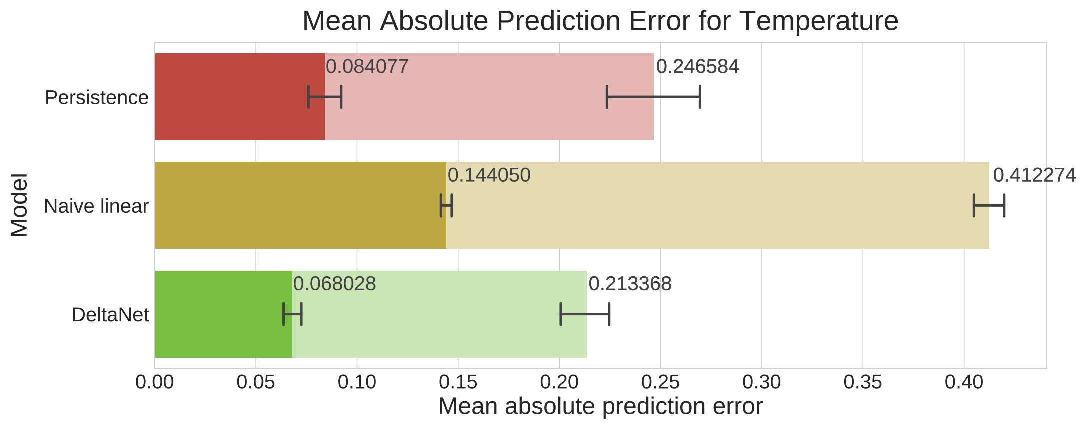

Figure 4.4 Mean Absolute Prediction Error for Temperature. The figure shows the prediction error (on a held-out test set) of various predictive models for real world temperature data. The darker portion of the bars represents the mean absolute prediction error at locations where there are weather stations and the lighter portion of the bar represents the mean absolute prediction error at locations where there are no weather stations. The error bars represent the 95 % confidence interval for the bootstrap mean (over 1000 bootstrap samples) of the evaluated quantities.

In figure 4.5, model prediction experimental results for dew point are presented. The first element to notice is that DeltaNet outperforms the naive linear model, but does not outperform the persistence model, even though not by far. This shows the limits of the admittedly small (with its 300 000 trainable parameters) DeltaNet model which may likely be able to perform better given a larger number of trainable parameters. Nonetheless, the point of making DeltaNet is to lay groundwork in the field and show such a model's potential for at least some weather parameter and expose its limitations to stimulate new research in the field. Thus, dew point nowcasting seems to be more challenging than the one of other weather parameters. This may be caused by the more independent/decoupled nature of dew point with respect to other weather parameters.

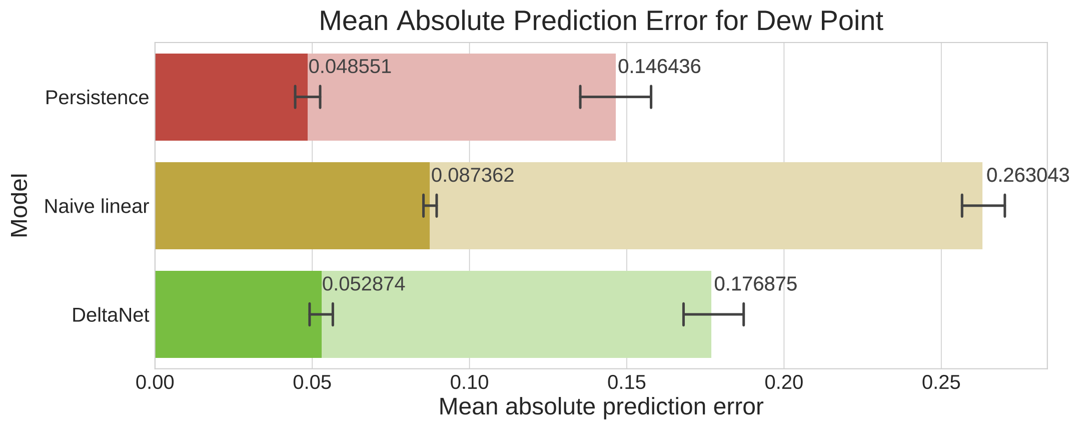

Figure 4.5 Mean Absolute Prediction Error for Dew Point. The figure shows the prediction error (on a held-out test set) of various predictive models for real world dew point data. The darker portion of the bars represents the mean absolute prediction error at locations where there are weather stations and the lighter portion of the bar represents the mean absolute prediction error at locations where there are no weather stations. The error bars represent the 95 % confidence interval for the bootstrap mean (over 1000 bootstrap samples) of the evaluated quantities.

Regarding relative humidity, figure 4.6 shows model prediction experimental results for it. The chart shows DeltaNet ends up with a slightly yet clearly lower prediction error than persistence and a significantly lower one than the one of the naive linear model. Akin to the aforementioned hypothesis, relative humidity's higher dependence on other weather parameters might make it easier for models to predict.

Figure 4.7 exhibits the experimental results for ultraviolet index. The visualization shows DeltaNet outperforms both persistence and naive linear baseline models by a significant margin. This result suggests ultraviolet index is simpler to predict, maybe because of its more predictable nature which could be due to its simpler dependence on other weather parameters and seasonality.

Lastly, figures 4.8, 4.9 and 4.10 respectively show the ground truth, DeltaNet model prediction and absolute prediction error for sample temperature data. These visualizations put forth the

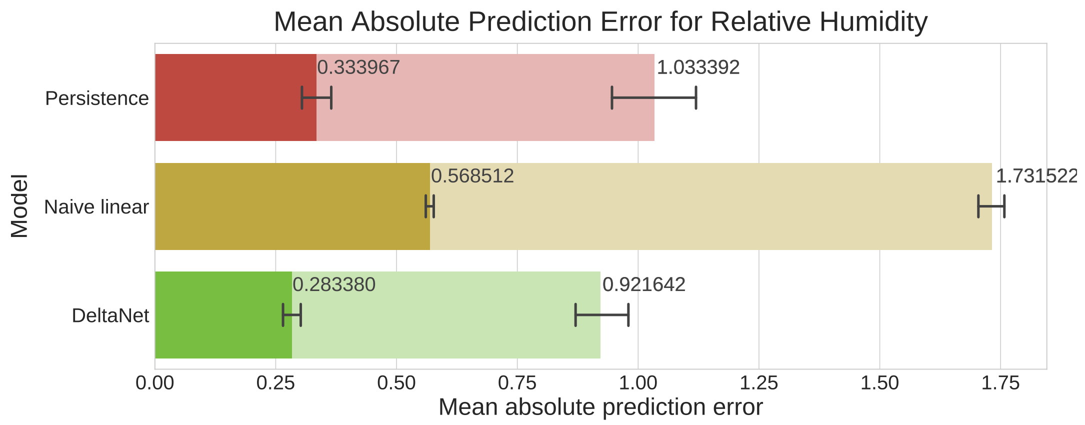

Figure 4.6 Mean Absolute Prediction Error for Relative Humidity. The figure shows the prediction error (on a held-out test set) of various predictive models for real world relative humidity data. The darker portion of the bars represents the mean absolute prediction error at locations where there are weather stations and the lighter portion of the bar represents the mean absolute prediction error at locations where there are no weather stations. The error bars represent the 95 % confidence interval for the bootstrap mean (over 1000 bootstrap samples) of the evaluated quantities.

model's high worldwide prediction accuracy and the preservation of natural-looking weather patterns (notwithstanding the distortions caused by the projection from the data's *plate carrée* projection to the visualization's Miller projection). Also note that some values closer to the planetary poles might seem odd. This is due to the weather station sparsity at those locations. Considering such sparsity, the model's predicted values are still very sensible with respect to the given ground truth, which may be improved by collecting more observations at these locations.

4.5 Contribution

This brief section simply puts forth the contribution of this work to the field of machine intelligence. Thus, the main contributions presented in this chapter are:

- a multi-channel worldwide weather forecasting problem formulation;
- DeltaNet, a deep neural network model architecture for automatic multidimensional worldwide weather forecasting;
- metrics to evaluate such a model's performance;
- a use case of SkNNI for geospatial forecasting;
- a demonstration through empirical results of the feasibility and potential of automatically learning to forecast weather by deep learning only through data.

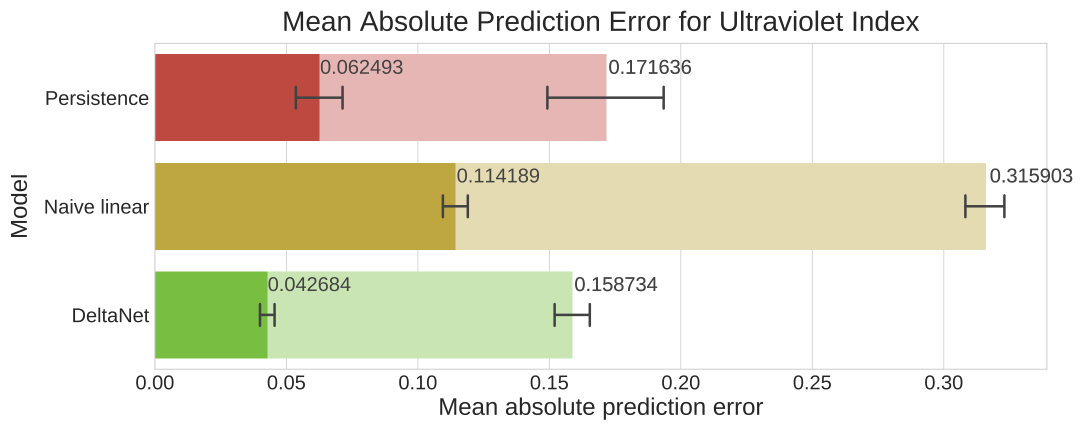

Figure 4.7 Mean Absolute Prediction Error for Ultraviolet Index. The figure shows the prediction error (on a held-out test set) of various predictive models for real world ultraviolet index data. The darker portion of the bars represents the mean absolute prediction error at locations where there are weather stations and the lighter portion of the bar represents the mean absolute prediction error at locations where there are no weather stations. The error bars represent the 95 % confidence interval for the bootstrap mean (over 1000 bootstrap samples) of the evaluated quantities.

4.6 Conclusion

Guided by the motivation for demonstrating the feasibility and potential of multi-channel worldwide weather forecasting by deep learning, the DeltaNet model is designed, trained, and evaluated against baseline models. Upon achieving better prediction accuracy than the baseline models most of the time, the initial hypothesis regarding the feasibility and potential of such an approach to weather forecasting holds true. Hence, the use of SkNNI to structure scattered geospatial data in a way that makes it useful for a predictive model and the design of the DeltaNet deep neural network architecture allow for the realization of automatic multi-channel worldwide weather forecasting with high accuracy in the optic of allowing people to plan ahead and live safely.

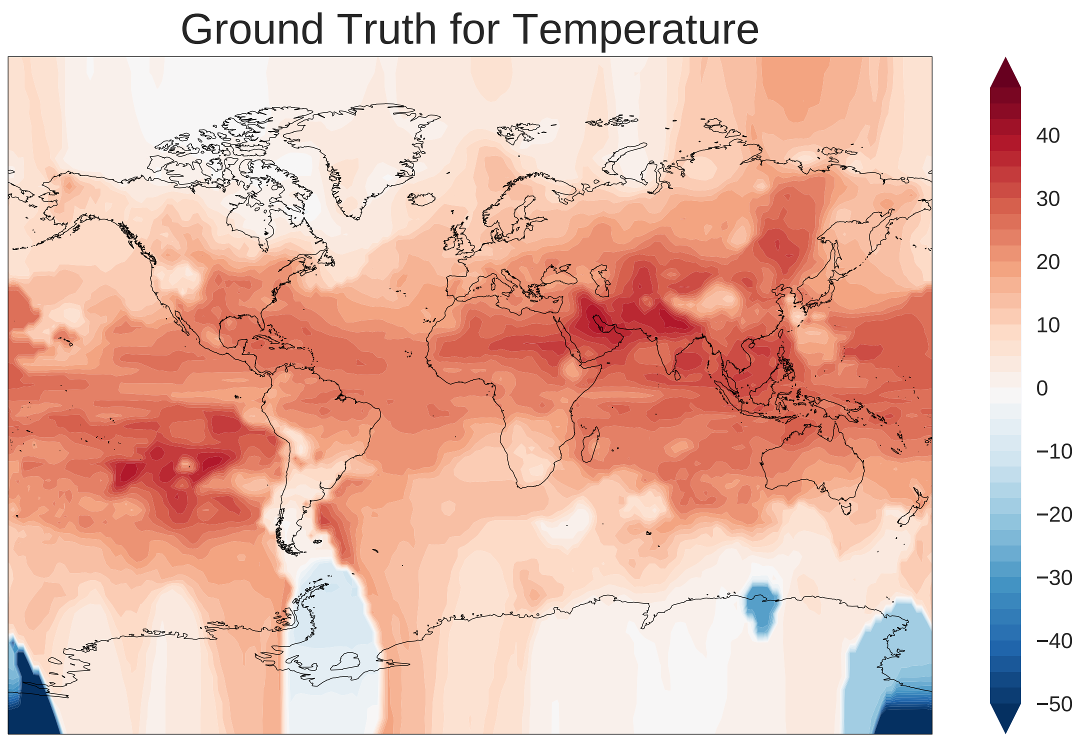

Figure 4.8 Ground Truth for Temperature. The figure shows the ground truth for the temperature channel of a sample nowcast. To make this visualization, the geospatial data is projected from its original *plate carrée* projection to the visualization's Miller projection, which causes some distortion.

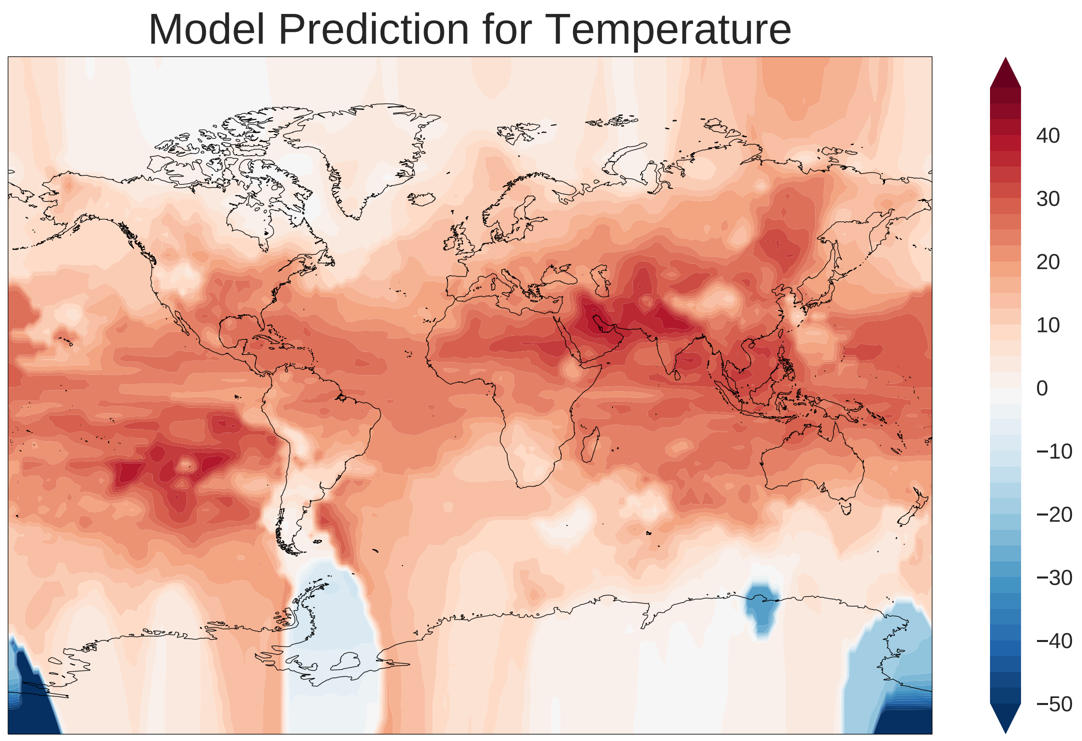

Figure 4.9 Model Prediction for Temperature. The figure shows the model prediction for the temperature channel of a sample nowcast. To make this visualization, the geospatial data is projected from its original *plate carrée* projection to the visualization's Miller projection, which causes some distortion.

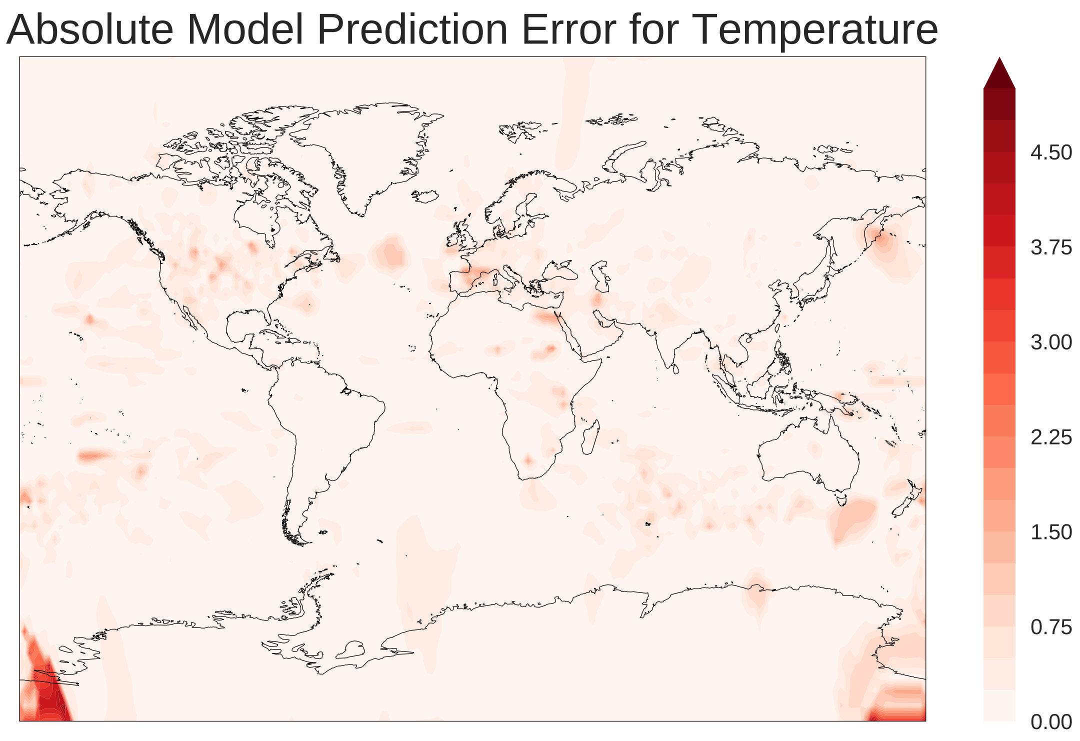

Figure 4.10 Absolute Model Prediction Error for Temperature. The figure shows the absolute model prediction error for the temperature channel of a sample nowcast. To make this visualization, the geospatial data is projected from its original *plate carrée* projection to the visualization's Miller projection, which causes some distortion.

CHAPTER 5 CONCLUSION

The present chapter concludes this work by first summarizing it and then presenting its contribution to the machine intelligence field. Are then presented the limitations of the approach put forth and future work pertaining to the topics coved throughout.

5.1 Summary

To briefly sum up this work, after introducing the problem under study and presenting related work in the field, throughout are presented SkNNI, a geospatial interpolation algorithm, and the feasibility and potential of multi-channel worldwide weather forecasting by deep learning through the design, implementation and evaluation of the DeltaNet deep neural network architecture. After running experiments which compare SkNNI and DeltaNet against respective baselines, both solutions significantly outperform their respective baselines and thus put forth their relevance at finding better solutions to the problems they are designed to solve.

5.2 Contribution

This work's fruition yields two main contributions.

The first one is SkNNI, the geospatial interpolation algorithm. Some of its subcontributions are the thought process behind the algorithm's design (like the importance of neighboring observations' distribution), and an efficient and user-friendly Python implementation of the algorithm with strong guarantees regarding maximal interpolation errors.

The second one is DeltaNet, a multi-channel worldwide weather forecasting model. Some of its subcontributions include the demonstration of a good use case for SkNNI, the presentation of a deep neural network architecture which outperforms baseline models and shows its relevance at finding more insightful solutions to the worldwide weather forecasting problem it aims to solve, and importantly a demonstration of the feasibility and potential of such an approach to weather forecasting.

5.3 Limitations

Since no real world solution is perfect, this section takes a look at both SkNNI and DeltaNet's limitations.

Starting with SkNNI, one of its prominent limitations when using the NDDNISD interpolation function is being an interpolation-only algorithm (with respect to provided observation value ranges) by design, meaning it cannot perform any form of extrapolation beyond the provided observation values. Thus, users must make sure to provide representative and range-covering observation values to the algorithm when building the interpolator to expect good results. Another limitation of the algorithm (which may also be seen as an advantage depending on context) is its data nature agnosticism, which might limit the algorithm's performance, i.e. if the algorithm were desired to be data-nature aware, it could potentially be updated to include trainable parameters and learn to interpolate better if using a decent observation dataset.

Moving on to DeltaNet, one of its important limitations is its small number of trainable parameters, which is close to 300 000. And yes; 300 000 trainable parameters is a noticeably small figure when compared to recent state-of-the-art deep neural network architectures which easily reach the hundreds of millions of trainable parameters [57] and even over a billion [58]. Though, DeltaNet is intentionally made to be relatively small, since the goal is to show the potential of the proposed problem formulation, which DeltaNet does, even with such a low number of trainable parameters. Ideas to remedy such a limitation would be to update DeltaNet's ResDRU transformations such that they use more trainable parameters, or simply to propose a new deep neural network architecture with its own set of trainable parameters. Another of DeltaNet's limitations (which might also be considered an advantage) is the use of channel-wise restoration operations which constrain the model's outputs to lie in each channel's predefined bounds. If no such constraint is desired, the channel-wise restoration step may simply be omitted or replaced by another kind of output activation function.

5.4 Future Work

To spark innovation and motivate researchers to always come up with ever-improving approaches and solutions, this section suggests ideas pertaining to potential future work related to or ensuing from this work.

On SkNNI's side, there are two prominent avenues awaiting future research. The first one is defining new interpolation functions to use within SkNNI. The second one is about the creation of a trainable version of SkNNI, potentially through deep learning or any other similar automatic learning method.

On DeltaNet's side, there are also two main avenues awaiting further exploration. The first one is about experimenting with different configurations of the problem formulation, e.g. by varying the input sequence length, the time step duration, the number of channels, and importantly which channels to use. The second one is about finding model architectures which yield high forecast accuracy while scaling well to hundreds of millions or even billions of trainable parameters. In that vein, this work offers ideas and suggestions like ConvNets, spherical ConvNets, ConvLSTMs and custom architectures like DeltaNet in the optic of stimulating the machine intelligence community to keep proposing novel solutions to challenging problems which impact millions on a daily basis.

5.5 Closing Thoughts

Looking forward, this work aims at laying groundwork regarding multi-channel worldwide weather forecasting by deep learning with the goal of showing its feasibility and potential, and inspiring researchers to build with and upon the core ideas behind the success of SkNNI and DeltaNet to provide increasingly more accurate weather forecasts all around the globe for people to plan their day and live safely.

REFERENCES

[1] A. G. Salman, B. Kanigoro, and Y. Heryadi, “Weather forecasting using deep learning techniques,” in *2015 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, 2015, Conference Proceedings, pp. 281–285.

[2] S. Lu, H. Youngdeok, I. Khabibrakhmanov, F. J. Marianno, S. Xiaoyan, J. Zhang, B. M. Hodge, and H. F. Hamann, “Machine learning based multi-physical-model blending for enhancing renewable energy forecast - improvement via situation dependent error correction,” in *2015 European Control Conference (ECC)*, 2015, Conference Proceedings, pp. 283–290.

[3] S. Suksri and W. Kimpan, “Neural network training model for weather forecasting using fireworks algorithm,” in *2016 International Computer Science and Engineering Conference (ICSEC)*, 2016, Conference Proceedings, pp. 1–7.

[4] L. Houthuys, Z. Karevan, and J. A. K. Suykens, “Multi-view ls-svm regression for black-box temperature prediction in weather forecasting,” in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, Conference Proceedings, pp. 1102–1108.

[5] Y. Rybarczyk and R. Zalakeviciute, “Machine learning approach to forecasting urban pollution,” in *2016 IEEE Ecuador Technical Chapters Meeting (ETCM)*, 2016, Conference Proceedings, pp. 1–6.

[6] B. Manning, “A machine learning based application for predicting global horizontal irradiance,” in *SoutheastCon 2017*, 2017, Conference Proceedings, pp. 1–6.

[7] K. L. M. D. Sobrevilla, A. G. Quiñones, K. V. S. Lopez, and V. T. Azaña, “Daily weather forecast in tiwi, albay, philippines using artificial neural network with missing values imputation,” in *2016 IEEE Region 10 Conference (TENCON)*, 2016, Conference Proceedings, pp. 2981–2985.

[8] E. Lazarevska, “Comparison of different models for wind speed prediction,” in *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, 2016, Conference Proceedings, pp. 5544–5549.

[9] A. Ahmadi, Z. Zargaran, A. Mohebi, and F. Taghavi, “Hybrid model for weather forecasting using ensemble of neural networks and mutual information,” in *2014 IEEE*

Geoscience and Remote Sensing Symposium, 2014, Conference Proceedings, pp. 3774–3777.

[10] Z. Karevan and J. A. K. Suykens, "Clustering-based feature selection for black-box weather temperature prediction," in *2016 International Joint Conference on Neural Networks (IJCNN)*, 2016, Conference Proceedings, pp. 2722–2729.

[11] E. Lazarevska, "Wind speed prediction with extreme learning machine," in *2016 IEEE 8th International Conference on Intelligent Systems (IS)*, 2016, Conference Proceedings, pp. 154–159.

[12] S. Sreekumar, K. C. Sharma, and R. Bhakar, "Optimized support vector regression models for short term solar radiation forecasting in smart environment," in *2016 IEEE Region 10 Conference (TENCON)*, 2016, Conference Proceedings, pp. 1929–1932.

[13] C. Zhao, M. v. Heeswijk, and J. Karhunen, "Air quality forecasting using neural networks," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2016, Conference Proceedings, pp. 1–7.

[14] M. K. Lee, S. H. Moon, Y. H. Kim, and B. R. Moon, "Correcting abnormalities in meteorological data by machine learning," in *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2014, Conference Proceedings, pp. 888–893.

[15] T. R. V. Anandharajan, G. A. Hariharan, K. K. Vignajeth, R. Jijendiran, and Kushmita, "Weather monitoring using artificial intelligence," in *2016 2nd International Conference on Computational Intelligence and Networks (CINE)*, 2016, Conference Proceedings, pp. 106–111.

[16] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," in *Advances in Neural Information Processing Systems 28*, 2015, Conference Proceedings, pp. 802–810.

[17] H. Drucker, C. J. Burges, L. Kaufman, A. J. Smola, and V. Vapnik, "Support vector regression machines," in *Advances in neural information processing systems*, 1997, Conference Proceedings, pp. 155–161.

[18] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: a new learning scheme of feedforward neural networks," *Neural networks*, vol. 2, pp. 985–990, 2004.

[19] J. M. Park and J. H. Kim, "Online recurrent extreme learning machine and its application to time-series prediction," in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, Conference Proceedings, pp. 1983–1990.

[20] M. Jordan, "Attractor dynamics and parallelism in a connectionist sequential machine," in *Proc. of the Eighth Annual Conference of the Cognitive Science Society (Erlbaum, Hillsdale, NJ), 1986*, 1986, Conference Proceedings.

[21] B. A. Pearlmutter, "Learning state space trajectories in recurrent neural networks," *Neural Computation*, vol. 1, no. 2, pp. 263–269, 1989.

[22] A. Cleeremans, D. Servan-Schreiber, and J. L. McClelland, "Finite state automata and simple recurrent networks," *Neural computation*, vol. 1, no. 3, pp. 372–381, 1989.

[23] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[24] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[25] J.-G. Lee and M. Kang, "Geospatial big data: challenges and opportunities," *Big Data Research*, vol. 2, no. 2, pp. 74–81, 2015.

[26] A. Gandomi and M. Haider, "Beyond the hype: Big data concepts, methods, and analytics," *International Journal of Information Management*, vol. 35, no. 2, pp. 137–144, 2015.

[27] S. Li, S. Dragicevic, F. A. Castro, M. Sester, S. Winter, A. Coltekin, C. Pettit, B. Jiang, J. Haworth, and A. Stein, "Geospatial big data handling theory and methods: A review and research challenges," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 115, pp. 119–133, 2016.

[28] H. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, and C. Shahabi, "Big data and its technical challenges," *Communications of the ACM*, vol. 57, no. 7, pp. 86–94, 2014.

[29] S. Erevelles, N. Fukawa, and L. Swayne, "Big data consumer analytics and the transformation of marketing," *Journal of Business Research*, vol. 69, no. 2, pp. 897–904, 2016.

[30] X.-W. Chen and X. Lin, "Big data deep learning: challenges and perspectives," *IEEE access*, vol. 2, pp. 514–525, 2014.

[31] S. Gao and V. Gruev, "Bilinear and bicubic interpolation methods for division of focal plane polarimeters," *Optics express*, vol. 19, no. 27, pp. 26 161–26 173, 2011.

[32] E. Jones, T. Oliphant, P. Peterson *et al.*, “SciPy: Open source scientific tools for Python,” 2001–. [Online]. Available: http://www.scipy.org/

[33] The Scipy community, “scipy.interpolate.smoothspherebivariatespline,” 2016. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.SmoothSphereBivariateSpline.html

[34] ——, “scipy.interpolate.lsqspherebivariatespline,” 2016. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.LSQSphereBivariateSpline.html

[35] The Tensorflow Community, “tfp.distributions.gaussianprocessregressionmodel,” 2019. [Online]. Available: https://www.tensorflow.org/probability/api_docs/python/tfp/distributions/GaussianProcessRegressionModel

[36] W. McDonough, “Compositional model for the earth’s core,” *Treatise on geochemistry*, vol. 2, p. 568, 2003.

[37] The Scipy community, “scipy.spatial.ckdtree,” 2016. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.cKDTree.html

[38] Wolfram Research, Inc., “Great circle,” 2018. [Online]. Available: http://mathworld.wolfram.com/GreatCircle.html

[39] N. R. Chopde and M. K. Nichat, “Landmark based shortest path detection by using a* and haversine formula,” *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 1, no. 2, pp. 298–302, 2013.

[40] The Scipy community, “numpy.arctan2,” 2009. [Online]. Available: https://docs.scipy.org/doc/numpy-1.12.0/reference/generated/numpy.arctan2.html

[41] M. A. Zaytar and C. El Amrani, “Sequence to sequence weather forecasting with long short-term memory recurrent neural networks,” *International Journal of Computer Applications*, vol. 143, no. 11, pp. 7–11, 2016.

[42] W. Wu, K. Chen, Y. Qiao, and Z. Lu, “Probabilistic short-term wind power forecasting based on deep neural networks,” in *2016 International Conference on Probabilistic Methods Applied to Power Systems (PMAPS)*. IEEE, 2016, Conference Proceedings, pp. 1–8.

[43] M. Raissi, A. Yazdani, and G. E. Karniadakis, "Hidden fluid mechanics: A navier-stokes informed deep learning framework for assimilating flow visualization data," *arXiv preprint arXiv:1808.04327*, 2018.

[44] T. S. Cohen, M. Geiger, J. Köhler, and M. Welling, "Spherical cnns," *arXiv preprint arXiv:1801.10130*, 2018.

[45] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, Conference Proceedings, pp. 770–778.

[46] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[47] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[48] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[49] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, and M. Isard, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, Conference Proceedings, pp. 265–283.

[50] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[51] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *31st Conference on Neural Information Processing Systems (NIPS 2017)*, 2017, Journal Article.

[52] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[53] S. Dieleman, J. Schlüter, C. Raffel, E. Olson, S. K. Sønderby, D. Nouri, D. Maturana, M. Thoma, E. Battenberg, J. Kelly, J. D. Fauw, M. Heilman, D. M. de Almeida, B. McFee, H. Weideman, G. Takács, P. de Rivaz, J. Crall, G. Sanders, K. Rasul, C. Liu,

G. French, and J. Degrave, "Lasagne: First release." Aug. 2015. [Online]. Available: http://dx.doi.org/10.5281/zenodo.27878

[54] R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, A. Belopolsky, Y. Bengio, A. Bergeron, J. Bergstra, V. Bisson, J. Bleecher Snyder, N. Bouchard, N. Boulanger-Lewandowski, X. Bouthillier, A. de Brébisson, O. Breuleux, P.-L. Carrier, K. Cho, J. Chorowski, P. Christiano, T. Cooijmans, M.-A. Côté, M. Côté, A. Courville, Y. N. Dauphin, O. Delalleau, J. Demouth, G. Desjardins, S. Dieleman, L. Dinh, M. Ducoffe, V. Dumoulin, S. Ebrahimi Kahou, D. Erhan, Z. Fan, O. Firat, M. Germain, X. Glorot, I. Goodfellow, M. Graham, C. Gulcehre, P. Hamel, I. Harlouchet, J.-P. Heng, B. Hidasi, S. Honari, A. Jain, S. Jean, K. Jia, M. Korobov, V. Kulkarni, A. Lamb, P. Lamblin, E. Larsen, C. Laurent, S. Lee, S. Lefrancois, S. Lemieux, N. Léonard, Z. Lin, J. A. Livezey, C. Lorenz, J. Lowin, Q. Ma, P.-A. Manzagol, O. Mastropietro, R. T. McGibbon, R. Memisevic, B. van Merriënboer, V. Michalski, M. Mirza, A. Orlandi, C. Pal, R. Pascanu, M. Pezeshki, C. Raffel, D. Renshaw, M. Rocklin, A. Romero, M. Roth, P. Sadowski, J. Salvatier, F. Savard, J. Schlüter, J. Schulman, G. Schwartz, I. V. Serban, D. Serdyuk, S. Shabanian, E. Simon, S. Spieckermann, S. R. Subramanyam, J. Sygnowski, J. Tanguay, G. van Tulder, J. Turian, S. Urban, P. Vincent, F. Visin, H. de Vries, D. Warde-Farley, D. J. Webb, M. Willson, K. Xu, L. Xue, L. Yao, S. Zhang, and Y. Zhang, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016. [Online]. Available: http://arxiv.org/abs/1605.02688

[55] F. Seide and A. Agarwal, "Cntk: Microsoft's open-source deep-learning toolkit," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM, 2016, Conference Proceedings, pp. 2135–2135.

[56] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *arXiv preprint arXiv:1512.01274*, 2015.

[57] D. Mahajan, R. Girshick, V. Ramanathan, K. He, M. Paluri, Y. Li, A. Bharambe, and L. van der Maaten, "Exploring the limits of weakly supervised pretraining," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, Conference Proceedings, pp. 181–196.

[58] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," in *Proceedings of the 25 th International Conference on Machine Learning*, 2019, Journal Article.