9-2021

# Deep Learning for Weather Clustering and Forecasting

Nathaniel R. Beveridge


**Deep Learning for Weather Clustering and Forecasting**

DISSERTATION

Nathanael R. Beveridge, Capt, USAF

AFIT-ENS-DS-21-S-037

**DEPARTMENT OF THE AIR FORCE**
**AIR UNIVERSITY**

***AIR FORCE INSTITUTE OF TECHNOLOGY***

**Wright-Patterson Air Force Base, Ohio**

AFIT-ENS-DS-21-S-037

# DEEP LEARNING FOR WEATHER CLUSTERING AND FORECASTING

DISSERTATION

Presented to the Faculty

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

Nathanael R. Beveridge, BS, MS

Capt, USAF

September, 2021

AFIT-ENS-DS-21-S-037

# DEEP LEARNING FOR WEATHER CLUSTERING AND FORECASTING

DISSERTATION

Nathanael R. Beveridge, BS, MS
Capt, USAF

Committee Membership:

Dr. Andrew J. Geyer, PhD
Chair

Dr. Robert C. Tournay, PhD
Member

Dr. Raymond R. Hill, PhD
Member

Adedeji B. Badiru, PhD
Dean, Graduate School of Engineering and Management

AFIT-ENS-DS-21-S-037

# Abstract

Clustering weather data is a valuable endeavor in multiple respects. The results can be used in various ways within a larger weather prediction framework or could simply serve as an analytical tool for characterizing climatic differences of a particular region of interest. This research proposes a methodology for clustering geographic locations based on the similarity in shape of their temperature time series over a long time horizon of approximately 11 months. To this end an emerging and powerful class of clustering techniques that leverages deep learning, called deep representation clustering (DRC), are utilized. Moreover, a time series specific DRC algorithm is proposed that addresses a current gap in the field. Finally, deep learning based weather prediction is an increasingly common research topic as a means of obtaining more rapid predictions when compared to traditional numerical weather prediction (NWP). Since their are known physical equations that govern atmospheric behavior, namely the Navier-Stokes equations, the concept of reformulating these laws into a physics based loss function is explored with particular interest in whether a model trained with such a loss function can outperform it's baseline counterpart.

*Rols and theo*

# Acknowledgements

I would like to thank my advisor Dr. Andrew Geyer and my co-advisors Dr. Robert Tournay and Dr. Raymond Hill. Your guidance in my research has been invaluable. Thank you for making this a great experience.

Additionally, I would like to thank my wife for her support throughout this endeavor.

Nathanael R. Beveridge

# Table of Contents

Page

Abstract . . . . . iv

Dedication . . . . . v

Acknowledgements . . . . . vi

List of Figures . . . . . ix

List of Tables . . . . . xii

I. Introduction . . . . . 1

1.1 Motivation and Background . . . . . 2
1.1.1 Clustering in Deep Weather Forecasting . . . . . 2
1.1.2 Deep Representation Clustering for Time Series . . . . . 3
1.1.3 Deep Weather Forecasting . . . . . 6
1.2 Problem Statement . . . . . 7
1.3 Organization of the Dissertation . . . . . 7

II. Literature Review . . . . . 9

2.1 Deep Weather Forecasting . . . . . 9
2.1.1 Numerical Weather Prediction . . . . . 9
2.1.2 Neural Network Mechanics . . . . . 13
2.1.3 Recurrent Neural Network Architectures . . . . . 17
2.1.4 Temporal Convolutional Network Architectures . . . . . 20
2.1.5 Deep Learning Based Weather Forecasting . . . . . 25
2.1.6 Physics-based Characterization of the Atmosphere . . . . . 28
2.1.7 Physics Incorporated into Deep Learning . . . . . 31
2.2 Clustering in Deep Weather Forecasting . . . . . 39
2.2.1 Current Applications of Clustering in Climatology . . . . . 41
2.2.2 Dynamical Systems . . . . . 46
2.2.3 Nonlinear Time Series Analysis of Dynamical Systems . . . . . 53
2.3 Deep Representation Clustering for Time Series . . . . . 56
2.3.1 Time Series Clustering . . . . . 56
2.3.2 Dynamic Time Warping . . . . . 58
2.3.3 Soft-DTW . . . . . 60
2.3.4 Autoencoders . . . . . 62
2.3.5 Deep Representation Clustering (DRC) . . . . . 62
2.3.6 Autoencoder-based Deep Clustering . . . . . 64
2.3.7 Training Deep Clustering Models . . . . . 65

Page

2.3.8 DRC in the Literature . . . 65

III. Solution Methodology . . . 70

3.1 Soft-DTW K-Means based DRC (SDTW-KM-DRC) . . . 70
3.1.1 Basic Model Structure . . . 70
3.1.2 SDTW-KM-DRC Clustering Layer . . . 71
3.1.3 Training Pipeline . . . 72
3.2 Soft-DTW KL Divergence based DRC (SDTW-KLD-DRC) . . . 76
3.2.1 SDTW-KLD-DRC Clustering Layer . . . 77
3.2.2 Training Pipeline . . . 77
3.3 Clustering Weather Data . . . 81
3.3.1 Dynamics Aware Embeddings . . . 81
3.3.2 Data . . . 81
3.3.3 Autoencoder Model Specifics (Pretraining) . . . 83
3.3.4 Autoencoder Pretraining Protocol . . . 89
3.3.5 SDTW-KM-DRC Training Protocol . . . 92
3.3.6 SDTW-KLD-DRC Training Protocol . . . 93
3.4 Physics Informed Deep Learning . . . 94
3.4.1 Spatio-Temporal Weather Prediction Problem . . . 95
3.4.2 Physics Loss . . . 96
3.4.3 Data . . . 99
3.4.4 Model Fitting, Evaluation and Analysis Strategy . . . 102

IV. Testing, Results, and Analysis . . . 104

4.1 Clustering Results . . . 104
4.1.1 Inter-algorithm comparison . . . 104
4.1.2 External Map Comparisons . . . 107
4.1.3 Land-based only SDTW-KM-DRC . . . 111
4.2 Physics Informed Deep Weather Forecasting . . . 117

V. Conclusions and Future Work . . . 121

5.1 Clustering . . . 121
5.1.1 SDTW-DRC Algorithm Conclusions . . . 121
5.1.2 Weather Clustering Conclusions . . . 122
5.1.3 Clustering Future Work . . . 123
5.2 Physics Informed Deep Weather Forecasting . . . 124
5.2.1 Physics Informed Deep Weather Forecasting Conclusions . . . 124
5.2.2 Physics Informed Deep Weather Forecasting Future Work . . . 125

Bibliography . . . 126

# List of Figures


| Figure | | Page |
|---|---|---|
| 1 | The two sin waves are exactly the same except for their phase. If compared using the euclidean distance they would be considered far less similar than desired. Additionally, time series centroid calculations are generally better done using dtw barycenter averaging or variants thereof. | 4 |
| 2 | Examples of atmospheric processes that are parameterized [1]. | 12 |
| 3 | Initialization recovers an appropriate starting point for forecasting. | 13 |
| 4 | Generic RNN unit representations. On left is the compressed representation, and on the right it is unfolded [2]. | 18 |
| 5 | LSTM cell [3]. | 19 |
| 6 | TDNN architecture with two hidden layers [4]. | 23 |
| 7 | Dilated and causal temporal convolution [5]. | 25 |
| 8 | End-to-end CDNN for estimation of motion vector, and prediction of $I_{t+1}$ [6]. | 36 |
| 9 | LSTM with explicit conservation of energy modeling [7]. | 39 |
| 10 | Clusters merge backwards in time [8]. | 43 |
| 11 | European and Mediterranean precipitation anomalies ordered by frequency [9]. | 45 |
| 12 | Time series and associated phase space representation of swinging pendulum [10]. | 47 |
| 13 | The Lorenz model is described by three variables whose time series are shown along with it's attractor [11]. | 48 |
| 14 | A contrived example showing a time delay embedding of a time series $y$, using parameters $m = 2$, and $\tau = 2$. | 51 |
| 15 | Time series clustering approaches flowchart [12]. | 57 |

Figure Page

16 Example showing how DTW alignment overcomes temporal shifts compared to Euclidean distance, along with the wapring path corresponding to the DTW alignment. . . . . 60

17 IDEC architecture improves upon DEC by utilizing full autoencoder throughout entire training process, making the embedding space resilient against distortions from $L_c$ [13] . . . . . 67

18 Acausal TCN with dilations = [1,2,4,8], stride = 1, and kernel size = 3 [14]. . . . . 71

19 Basic SDTW-KM-DRC architecture with visualization of processing one example time series through the network. . . . . 75

20 Basic SDTW-KLD-DRC architecture with clustering layer. Note there are no additional external algorithms needed, rather the cluster information update is solely reflected in the periodic target distribution update. . . . . 78

21 SDTW-DRC architecture with additional dynamics prediction layers to facilitate learning of a dynamics aware latent space. . . . . 82

23 Sensitivity analysis of clustering results for $\gamma = 0.1, 0.01,$ and 0.001 to ensure robustness of the proposed value of $\gamma = 0.01$. . . . . 88

24 Cluster maps for $k = 5, 6, 7, 8, 9, 10,$ and 11 showing the results after pretraining the autoencoder but before any cluster specific training. . . . . 90

25 Training and validation $\mathcal{L}_{Total}$ curves along with vertical line indicating that epoch 18 corresponds with minimum validation $\mathcal{L}_{Total}$ . . . . . 91

26 Map showing the region of HRRR data collected (blue), and the region to which a rectilinear grid was interpolated using the HRRR data (yellow) [15]. . . . . 101

Figure Page

27 Cluster maps for k=9. The initial cluster results coming from the autoencoder alone are re-displayed in (27a), and the SDTW-KM-DRC and SDTW-KLD-DRC results after training for two epochs are shown in (27b) and (27c) respectively. . . . . 106

28 Mean annual temperature (Fahrenheit) [16]. . . . . 108

33 SDTW-KM-DRC test after removal of water-based clusters. . . . . 112

29 Elevation comparison. . . . . 114

30 Geographical comparison. . . . . 115

31 Koppen-Geiger climate classification map for North America from 1980-2016 [17] . . . . . 116

32 North American SST's [18]. . . . . 116

34 Physics-based model loss vs traditional model loss instances. . . . . 118

35 Training and validation error for physics-based model over extended analysis 75 epoch training period for Instance 1 (zoomed in). . . . . 120

# List of Tables


| Table | | Page |
|---|---|---|
| 1 | Overview of four clustering algorithm categories according to [19] | 41 |
| 2 | The meanings of different levels of the scaling exponent $\alpha$ [20] | 56 |
| 3 | The data shape dimensions are $(batch\_size, \#filters, sequence\_length)$ for all block types except those with fully connected (FC) layers where the dimensions are $(batch\_size, sequence\_length)$. Padding was left out of the table, but it was structured such that the inputs and outputs of a layer were changed only as a result of stride, upsampling, or downsampling, and never as a result of kernel size. DFA and SampEn each have their own branches off the latent space but are identical. | 85 |
| 4 | TCN architecture for weather prediction. The kernel sizes are along the time and x-y spatial dimensions respectively. | 96 |
| 5 | Comparison of physics-based and traditional neural network test MSE across all three instances. Note that the physics-based model achieves a lower test MSE across all three instances. | 117 |
| 6 | Test set MSE for Instance 1 analysis extended to 75 epochs. | 119 |

# I. Introduction

Weather forecasting has a long history rooted in basic physical principles. Scientists around the turn of the $20^{th}$ century noticed that the atmosphere could be treated as a fluid, and thus modeled using partial differential equations (PDE's) and established physical principles that describe fluid characteristic change over time and space [21]. These methods, which in the weather prediction domain are referred to as Numerical Weather Prediction (NWP), have improved greatly over time and currently represent the standard weather modeling approach. However, advances in deep and machine learning methods have resulted in significant interest in the application of data-driven approaches to the problem of weather modeling and prediction. Further, research is being done to specifically address the use of deep learning methods in the modeling of physical processes where, as with weather, there is often existing information about the process that can be leveraged.

Machine and deep learning techniques like clustering can also be used to analyze weather data. This is often in an effort to group together periods of time that experience similar weather patterns over a broad region, but could also be done to group geographical areas that experience similar weather patterns over long periods of time. Both approaches can aid in a broader weather prediction framework, while the latter can also be used to provide a general characterization of the climatic differences of a region.

This research adds to the deep learning and atmospheric/climate modeling fields in a few ways. First, the United States is clustered into geographic regions of similarity

with respect to time series shape and weather dynamics using a powerful new class of clustering algorithm referred to here as deep representation clustering (DRC). Second, a gap in the current research on DRC will be addressed. Specifically, to the best of our knowledge, there have not been any time series specific DRC algorithms published. Instead, when dealing with time series data most have used Euclidean based distance metrics at the core of their clustering algorithms which can produce undesirable results. Finally, the prospect of leveraging known physical laws within a deep learning model for 24-hour weather prediction is explored.

## 1.1 Motivation and Background

### 1.1.1 Clustering in Deep Weather Forecasting

#### Clustering

The complexity and difficulty of obtaining accurate weather predictions cannot be overstated. The underlying processes are highly nonlinear, chaotic, and non-stationary. While these type of phenomena can be modeled by neural networks, information that in any way hints at the type of weather patterns being experienced can be utilized in a broader prediction framework. This has turned some researchers onto the idea of clustering weather data and using the nature and membership of the clusters to inform a neural network [22]. Further, there is a long history of grouping geographic locations in an effort to categorize regional climates [17]. An approach that has not yet been explored is the direct use of time series data with an appropriate time series distance metric that accounts for time series shape. Doing so yields clusters of locations that are similar with respect to an atmospheric measurement over time. Again, these results could be used for various purposes. In a weather prediction framework this could be used to construct cluster specific models for learning whatever

complex behavior is typical of weather at the locations in each cluster. By fitting unique models to each cluster they are given a better chance of learning the complex relationships observed in their clusters by focusing on the behavior typical of those locations. The cluster results could also be used more generically as a way of broadly categorizing the climate(s) of a region.

#### Dynamics

In certain areas of research there exists a decoupling of the study of time series analysis from that of dynamical systems. When dealing with weather data, remaining cognizant of the fact that each time series is merely a 1-dimensional projection of the larger dynamical system is important, especially since we want cluster formation to be driven in part based on similarity in the underlying dynamics of each location. For this reason, the DRC algorithm proposed takes special care to drive formation of dynamics-aware clusters.

### 1.1.2 Deep Representation Clustering for Time Series

#### Time Series Clustering

There are unique challenges associated with clustering time series data. This is rooted in the fact that clustering necessitates the use of distance measures or metrics to determine the similarity or dissimilarity of points. In the case of static data, a data point or vector $x \in \mathbb{R}^n$ represents $n$ characteristics of an observation, whereas a time series vector $y \in \mathbb{R}^n$ represents $n$ measurements in time of some characteristic of a system. Measuring the similarity between vectors of static data is commonly associated with the Euclidean distance, and appropriately so. However, to understand why this could be problematic in the case of temporal data consider the time series in Figure 1.

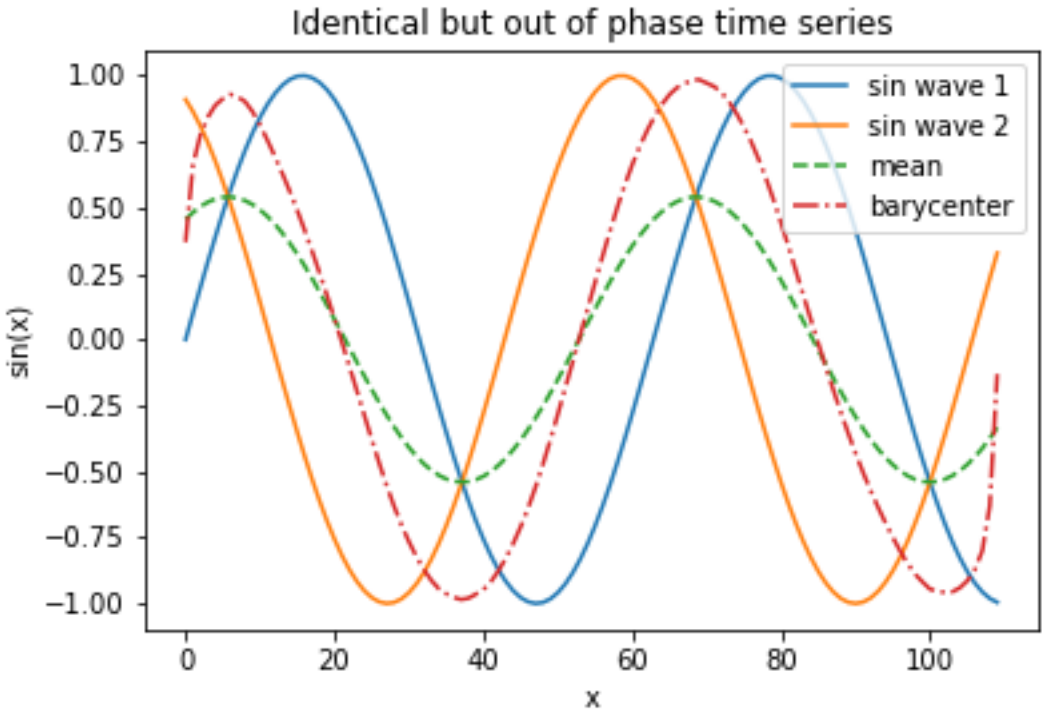



**Figure 1. The two sin waves are exactly the same except for their phase. If compared using the euclidean distance they would be considered far less similar than desired. Additionally, time series centroid calculations are generally better done using dtw barycenter averaging or variants thereof.**

Though it is clear that these are two identical sin waves with slightly offset phases, and should be considered highly similar in the context of clustering, the euclidean distance yields a false sense of dissimilarity. For the purposes of grouping geographic locations by similarity in weather patterns, it is imperative that our distance measure account for temporal shifts and variation.

Centroid calculation is another fundamental component of many clustering algorithms. In the same way that the euclidean distance can misrepresent the true similarity of two time series, the arithmetic mean which is used for centroid calculations can produce an unnatural average of time series. Specifically, the arithmetic mean can often produce an average that is severely distorted in terms of shape, and thus not appropriately representative of its constituents. Referring back to Figure 1 it

can be seen that the arithmetic mean curve (green) is a misrepresentation of the two sin wave shapes' while the red curve, a dynamic time warping (DTW) based average which will be introduced in more detail later, is a much better "average" for these time series.

### Why DRC?

Deep representation clustering describes a class of models that use neural networks to aid in the clustering task. The primary approach combines representation learning, usually through an autoencoder, and a clustering loss to drive formation of cluster friendly representations. Since the process usually involves representation learning anyways, DRC is well suited for high dimensional, complex data, as it can be mapped to a lower dimensional representation. This is useful both computationally, and for avoiding the curse of dimensionality. With the clustering task for this research involving long, complex time series, DRC is a natural approach. Finally, recent research efforts by M. Cuturi and M. Blondel have yielded a differentiable version of the dynamic time warping (DTW) loss called soft-DTW [23], which is important for two main reasons. First, time series cannot be properly compared in terms of similarity with a standard Euclidean distance function. Instead something like DTW, or a relaxation like soft-DTW, is necessary to account for similarity in shape. Second, since DRC is a neural network based method, all loss functions used in training must be differentiable to facilitate parameter optimization via backpropagation which is a characteristic lacking with standard DTW. For these reasons, the development of soft-DTW lends itself nicely to the creation of a DRC algorithm that can appropriately handle time series data.

### 1.1.3 Deep Weather Forecasting

There are many reasons to consider a deep learning approach for modeling physical processes, especially atmospheric dynamics. First, traditional NWP methods are very computationally expensive since the relevant PDE's must be solved with new initial and boundary conditions each forecast cycle. Further, enhancing predictive capability and the resolution of predictions requires increased computational resources. Specifically, researchers have shown that doubling prediction resolution requires an order of magnitude increase in computational power [24]. On the other hand, once a deep learning model is trained, obtaining predictions is relatively quick.

Second, a data driven approach is flexible and can be used to help practitioners quickly begin making localized forecasts at locations for which regional high-resolution NWP is not occurring. This is a noteworthy benefit for the military. For instance, when troops are sent to establish new outposts, one of the first tasks is constructing an airfield so that transportation and operations into and out of the base can occur. The ability to understand and predict weather at and around the airfield is critical for the success of the outpost. However, this is not only applicable for airfield operations. Accurate weather assessment is necessary for all kinds of mission planning, and in many of these type of overseas environments there are no regional/local NWP models offering high resolution predictions. Joint Publication 3-59 [25], titled *Meteorological and Oceanographic Operations*, put out by the Joint Chiefs of Staff specifically lays this out by stating that the cornerstone of it's joint meteorological and oceanographic (METOC) operations are accuracy, consistency, relevancy, and timeliness. To achieve the accuracy objective they state that, "METOC data and information must be measurably correct in representing the current and future state of the environment" [25]. Regarding timeliness they say, "The principle of timeliness depends upon products that are derived from the latest available data, processed and disseminated quickly,

and integrated at the appropriate time into planning and execution processes" [25]. A deep learning, data-driven model specifically addresses the accuracy and timeliness objectives by providing a way to quickly obtain predictions for any given region.

Finally, a data-driven approach does not preclude the incorporation of known physical principles, despite only recently becoming a mainstream research area. The primitive equations, which provide a mathematical description of how atmospheric quantities change in relation to one another across space and time, are valuable information for the problem at hand irrespective of a specific modeling approach. These equations can be used within the neural network training framework to incentivize predictions that adhere to the known physical laws. This approach leverages both the known physical relationships among these variables, while also benefitting from the universal function approximation capability that deep neural networks (DNN's) provide [26].

## 1.2 Problem Statement

This research proposes a novel shape-based, dynamics-aware DRC algorithm for clustering weather data geographically as opposed to temporally. Further we aim to improve the speed, and localization with which weather forecasts are made by making use of deep learning while allowing the deep neural network to leverage known physical principles.

## 1.3 Organization of the Dissertation

Chapter II will first explore literature regarding the history of weather prediction, current research in deep weather forecasting, the Navier-Stokes equations, and physics informed deep learning approaches. Following this, weather clustering, dynamical systems, and DRC will discussed. Chapter III will present two soft-DTW based DRC

algorithms, the application of these algorithms to weather data, and a Navier-Stokes inspired physics-based loss function proof of concept for weather prediction. Finally, Chapters IV and V will provide results, and conclusions and future work respectively.

# II. Literature Review

Chapter II presents a collection of the existing research relevant for addressing the research objectives as stated in Chapter I. We will first explore the history of NWP, basic neural network architectures for temporal modeling, the Navier-Stokes equations, and physics informed neural networks. Then we will examine clustering for weather data, dynamical systems, the soft-DTW distance measure and existing DRC algorithms.

## 2.1 Deep Weather Forecasting

This section will address current research in the field of deep weather forecasting. To provide adequate context and supporting information, subsections on numerical weather prediction (NWP), and neural networks will be presented. Additionally, there are subsections on both existing applications of deep weather forecasting and physics informed deep learning to explore the full spectrum of relevant information.

### 2.1.1 Numerical Weather Prediction

NWP is the current industry standard for weather prediction. So, although this research proposes a deep learning based approach to weather prediction it remains important to address the history of the field as it is still, and will continue to be, an important part of atmospheric modeling. Moreover, since our proposed deep weather forecasting model is informed by physics, our approach draws inspiration from NWP.

#### Origins of NWP

Since the early to mid 1900's, physics has been at the heart of weather modeling. However, this was not always the case. It was not until around 1890 that American

meteorologist Cleveland Abbe realized "[m]eteorology is essentially the application of hydrodynamics and thermodynamics to the atmosphere" [27]. Not long after, Norwegian scientist Vilhelm Bjerknes's 1904 paper titled "The problem of weather prediction, considered from the viewpoints of mechanics and physics" more explicitly formalized the approach of weather prediction from a scientific viewpoint [28]. Bjerknes's proposal came at a time when weather forecasting was primarily an art as opposed to a science. Forecasters relied on experience, memory and sets of empirically derived rules [21]. Bjerknes posited that the use of physical laws for making predictions about future states of the atmosphere based on a current state requires two things:

1. The ability to accurately ascertain the state of the atmosphere at a given time (initialization)

2. Sufficient knowledge of the laws which govern transition of the atmosphere from one state to the next [28][21].

Bjerknes immediately noted that obtaining initial states of the atmosphere would be difficult due to lack of weather measurement data. At the time, observations were not collected over the ocean, or anywhere above the earth's surface. In terms of modeling atmospheric state transitions, he theorized that if at any point in time, density, velocity, pressure, temperature, and humidity of the air could be determined, the state of atmosphere would be completely described [28]. To solve for these parameters, he proposed seven equations: the three hydrodynamic equations of motion, the continuity equation, the equation of state for the atmosphere, and the two fundamental laws of thermodynamics [28]. It was later discovered that instead of the second law of thermodynamics, a continuity equation for water should be used [21].

Eleven years later Bjerknes's work was picked up by English scientist Lewis Fry

Richardson. Richardson dove into finding a practical way of using the finite difference method to numerically solve Bjerknes's set of differential equations. A practical method for solving these equations provides the mathematical engine for forecasting some $\Delta t$ time steps from the atmosphere's initial state at $t = 0$. Richardson applied his method to predicting surface pressure and wind at two points in Europe. In hindsight, it has been shown that his method for solving the equations is sound, however his predictions were drastically off due to bad representation of the atmosphere's initial state [21].

Abbe, Bjerknes, and Richardson collectively laid the foundation for modern numerical weather prediction. Richardson's method for solving the differential equations was further refined and initialization techniques were established for providing more accurate initial states for the atmosphere.

### Modern NWP

Computing power was and continues to be one of the biggest limitations faced by NWP. Today these limitations manifest in forecast resolution constraints, but in the years following Richardson, even solving simplified versions of the continually refined system of equations required revolutionary computing power [21]. In fact, it was not until the 1970's that computers powerful enough to solve the equations first set out by Abbe and Bjerknes were developed [1]. Throughout this time period of advances in computational power, analytical work was also done to more effectively utilize the set of equations. Numerical methods were developed to address instabilities in approximate solutions, overall accuracy, and computational speed [1]. Parameterization and initialization have also been key areas of research in NWP. Parameterization is a way of representing the movement of tiny particles whose movements are not accounted for with the governing physical equations [29][21][1]. This is necessary because since

it is mathematically intractable to derive a direct analytical solution to the equations and approximations must be used, a disparity between what can and cannot be resolved by the model emerges. Parameterization allows what the model cannot directly resolve to be accounted for [1]. Figure 2 provides examples of the type of processes which require parameterization.

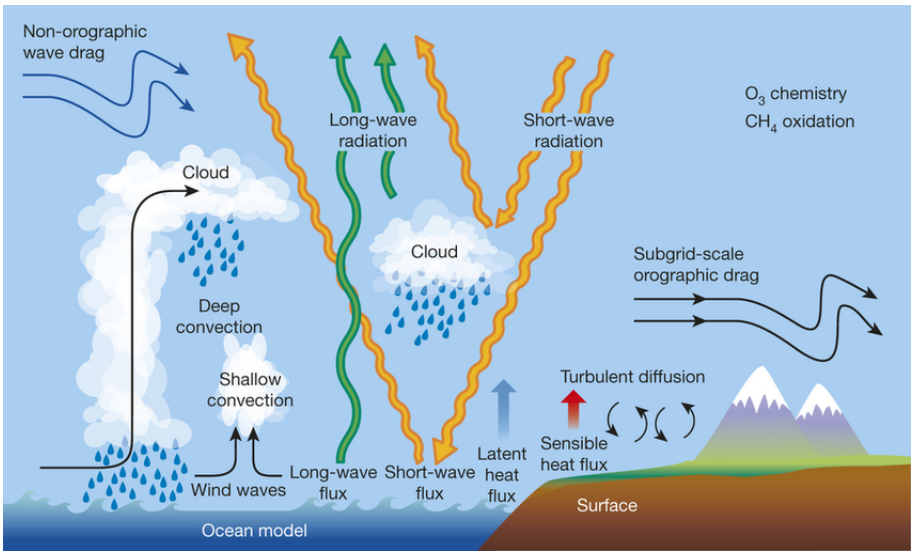



**Figure 2. Examples of atmospheric processes that are parameterized [1].**

Initialization is required to overcome the problems Richardson faced in his first experiment. Namely, there must be a process to acquire accurate starting conditions. Some of the first methods of initialization involved interpolation and the use of graphical and synoptic weather charts. However, these were soon abandoned in pursuit of using optimal control theory to perform data assimilation [1]. The problem lies in that observational data used to provide initial conditions may be noisy and inaccurate. So, exact interpolation of this data to provide starting conditions is problematic [30]. Instead, data assimilation can be accomplished by using observational data and information from short term forecasts, as well as the uncertainty associated with the forecasts, as constraints in a Bayesian inversion problem to recover accurate initial conditions [1]. Data assimilation extracts the meaningful information from incoming

observational data while filtering out noise [30]. Figure 3 displays this graphically.

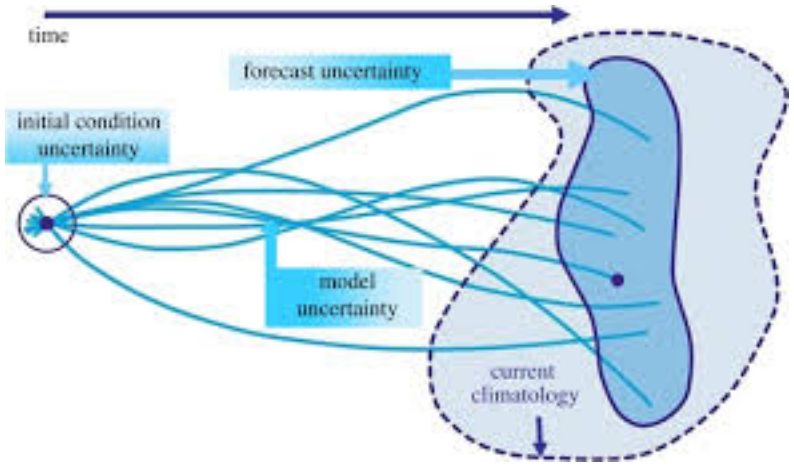
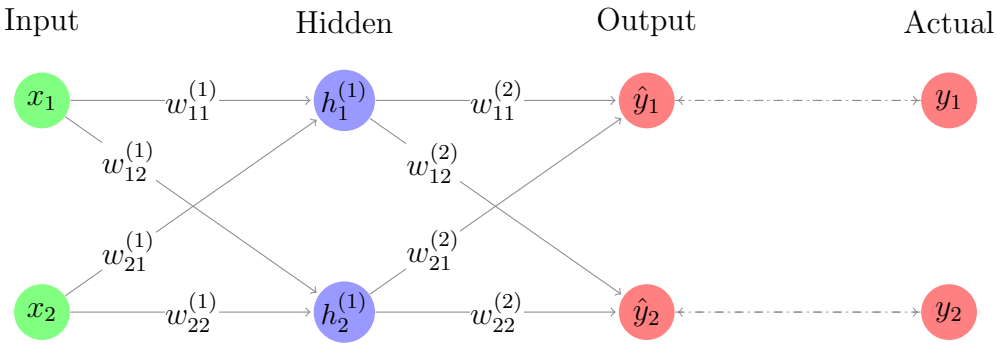



**Figure 3. Initialization recovers an appropriate starting point for forecasting.**

### 2.1.2 Neural Network Mechanics

This section provides a brief overview of forward and backward propagation in a neural network, and serves two purposes. First, to provide context for the next section's discussion which involves various types of neural networks. Second, understanding back-propagation, and more specifically how the loss function is used in learning network parameters, is critical to developing a loss function that appropriately forces the network to learn within the bounds of known physical principles.

To provide ample context for back propagation, forward propagation must first be explained and relevant network components introduced. Consider a very simple feed-forward neural network performing a two-category classification task. This network has two input features, and one hidden layer. Since we are assuming a classification task, the hidden layer has a softmax activation. The input layer activations will be sigmoid functions. Let us feed a single training example through the network, assuming the network is untrained and the weights on each arc are randomly initialized.




Consider node $h_1^{(1)}$. Define $z_1^{(1)} = b_1^{(1)} + w_{11}^{(1)}x_1 + w_{21}^{(1)}x_2$, where $b_1^{(1)}$ is the bias. Now $z_1^{(1)}$ is input to the activation function $a_1^{(1)}$ to determine how much of $z_1^{(1)}$ will be propagated to the next layer. The activation for node $h_1^{(1)}$ is:

$$a_1^{(1)} = \sigma(z_1^{(1)}) = \frac{1}{1+e^{z_1^{(1)}}} = \frac{1}{1+e^{b_1^{(1)}+w_{11}^{(1)}x_1+w_{21}^{(1)}x_2}} \tag{1}$$

The same calculation for $h_2^{(1)}$ yields:

$$a_2^{(1)} = \sigma(z_2^{(1)}) = \frac{1}{1+e^{z_2^{(1)}}} = \frac{1}{1+e^{b_2^{(1)}+w_{12}^{(1)}x_1+w_{22}^{(1)}x_2}} \tag{2}$$

At the hidden layer the process is the same, but the inputs are now $a_1^{(1)}$, and $a_2^{(1)}$ instead of original data values as in the input layer. In fact, for consistency of notation, some would denote the input layer nodes $a_1^{(0)}$, and $a_2^{(0)}$. The output nodes have softmax activation functions which give probabilities that the input belongs to each of the classes in the data-set. The calculations to obtain $\hat{y}_1$, and $\hat{y}_2$ are:

$$\hat{y}_1 = a_1^{(2)} = \frac{e^{z_1^{(2)}}}{e^{z_1^{(2)}}+e^{z_2^{(2)}}} = \frac{e^{b_1^{(2)}+w_{11}^{(2)}a_1^{(1)}+w_{21}^{(2)}a_2^{(1)}}}{e^{b_1^{(2)}+w_{11}^{(2)}a_1^{(1)}+w_{21}^{(2)}a_2^{(1)}}+e^{b_2^{(2)}+w_{12}^{(2)}a_1^{(1)}+w_{22}^{(2)}a_2^{(1)}}} \tag{3}$$

$$\hat{y}_2 = a_2^{(2)} = \frac{e^{z_2^{(3)}}}{e^{z_1^{(3)}}+e^{z_2^{(3)}}} = \frac{e^{b_2^{(2)}+w_{12}^{(2)}a_1^{(1)}+w_{22}^{(2)}a_2^{(1)}}}{e^{b_1^{(2)}+w_{11}^{(2)}a_1^{(1)}+w_{21}^{(2)}a_2^{(1)}}+e^{b_2^{(2)}+w_{12}^{(2)}a_1^{(1)}+w_{22}^{(2)}a_2^{(1)}}} \tag{4}$$

So in our example of an untrained network, the resulting probabilities $\hat{y}_1$, and $\hat{y}_2$ most likely do a bad job of predicting the actual training example outcome. For

instance, imagine $\hat{\boldsymbol{y}} = [0.45,\ 0.55]^T$ while $\boldsymbol{y} = [1,\ 0]^T$. This corresponds to the network predicting the input is from the second class, while it is actually from the first class. To improve network predictions throughout the training process, there must be a mechanism for quantifying the prediction error and appropriately attributing blame for the error to individual network parameters so that they can be adjusted correctly. This is the job of a loss function. For classification problems, the following cross entropy loss function is commonly used:

$$L(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\sum_i y_i \log(\hat{y}_i) \tag{5}$$

where the summation is over the $i$ classes in the dataset [31]. Despite Equation 5 only being written in terms of $\hat{\boldsymbol{y}}$ and $\boldsymbol{y}$, it is actually a function of the network parameters $\boldsymbol{\theta}$ (weights and biases). This is clear by noting that $\hat{\boldsymbol{y}}$ is obtained using Equations 1-4, which are themselves functions of $\boldsymbol{\theta}$. Since the loss function can be expressed in terms of the network parameters, the error can be attributed to these parameters through the gradient of the loss function. Specifically, taking the negative gradient of the loss function indicates how much the loss function goes up or down with changes to the loss function inputs. To see this more clearly, we will continue with our ongoing example. Let's look at just one parameter, $w_{21}^{(1)}$. To figure out how this parameter affects the cost function, $\frac{\partial L}{\partial w_{21}^{(1)}}$ must be calculated using the chain rule. With Equations 1-5 given in their expanded form, this is readily done using rules of partial differentiation.

$$\frac{\partial L}{\partial w_{21}^{(1)}} = \frac{\partial L}{\partial a_1^{(2)}} \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial a_1^{(1)}} \frac{\partial a_1^{(1)}}{\partial z_1^{(1)}} \frac{\partial z_1^{(1)}}{\partial w_{21}^{(1)}} \tag{6}$$

$$= \left(\frac{1}{a_1^{(2)}}\right) \left(-\frac{e^{z_1^{(2)}}}{(1+e^{z_1^{(2)}})^2}\right) (w_{11}^{(2)}) \left(-\frac{e^{z_1^{(1)}}}{(1+e^{z_1^{(1)}})^2}\right) (x_2) \tag{7}$$

$$= \frac{e^{z_1^{(2)}} w_{11}^{(2)} e^{z_1^{(1)}} x_2}{\left(1 + e^{z_1^{(2)}}\right)\left(1 + e^{z_1^{(1)}}\right)^2} \tag{8}$$

The quantity in Equation 8 gives how much $w_{21}^{(1)}$ should be changed to increase the cost function the most. So, since we want to minimize the loss, we adjust $w_{21}^{(1)}$ by the negative of it. Taking a larger view, this can be done on all of the loss function parameters at once using the gradient, $\nabla L$. Recalling that $\boldsymbol{\theta}$ is our parameter vector, the parameter updates take the form: $\boldsymbol{\theta}_{new} = \boldsymbol{\theta}_{current} - \nabla L$.

In practice, training is usually done using batches of training examples. This introduces the cost function $J$, which is simply the average of a batch of $N$ individual training example losses.

$$J = \frac{1}{N} \sum_{j=1}^{N} L(\hat{\boldsymbol{y}}, \boldsymbol{y}) \tag{9}$$

However, this does not change the back-propagation process outlined above. Instead of $\nabla L$, the relevant quantity becomes $\nabla J$; a minor substitution.

To recap, back-propagation involves minimizing a cost function, which encapsulates how far off the network's predictions are at a given setting for network weights and biases. Since $J$ is a function of the network parameters, the gradient of $J$ yields the rate of steepest ascent with respect to those parameters. Since the cost should be minimized, the weights are adjusted every $N$ training examples according to the formula: $\boldsymbol{\theta}_{new} = \boldsymbol{\theta}_{current} - \nabla J$.

### 2.1.3 Recurrent Neural Network Architectures

#### RNN

Recurrent Neural Networks (RNN's) or RNN variants are ubiquitous in deep learning based time series modeling, and a natural place to begin discussion. Consider the sequence $\boldsymbol{x^{(t)}}$. Within an RNN, each element of this sequence is processed by identical RNN units using the same set of parameters within each unit. These units have two inputs at each time step; input from the time series itself at time $t$, and the activation from the previous time step as shown in Figure 4.

$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}, \tag{10}$$

$$\boldsymbol{h}^{(t)} = tanh(\boldsymbol{a}^{(t)}), \tag{11}$$

$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \tag{12}$$

$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}(\boldsymbol{o^{(t)}}) \tag{13}$$

This notion of parameter sharing is shown pictorially in Figure 4. Note that at each time step the same $\boldsymbol{U}, \boldsymbol{W}$, and $\boldsymbol{V}$ weight matrices are used. This has a number of implications. First, parameter sharing makes sequence processing more computationally feasible since only one set of parameters must be learned per RNN unit. Second, it makes the network invariant to temporal shifts. This means the network can learn a generalizable set of rules regarding the behavior of the sequence or time series. Finally, parameter sharing allows RNN's to process observations of variable lengths since the observation is processed sequentially and time steps can simply continually be fed since they will all be processed with the same set of weights. It is also worth noting that while Figure 4 shows the network producing an output at every time step, this does not have to be the case.

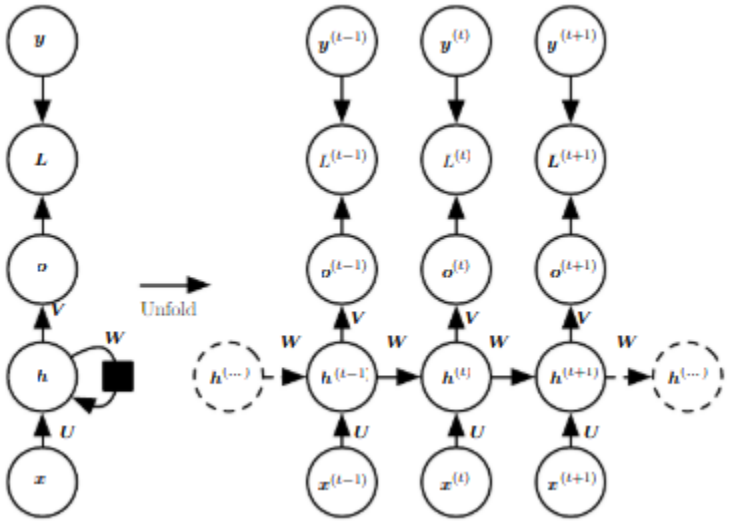



**Figure 4. Generic RNN unit representations. On left is the compressed representation, and on the right it is unfolded [2].**

### LSTM

Recurrent architectures as described above can suffer from vanishing gradients when applied to long sequences which inhibits learning [2]. Long-short term memory networks (LTSM's) and gated recurrent units (GRU's), were developed specifically to address this issue. They do this effectively and as a result have more or less become the default RNN architectures. Recall Equations 10, 11, 12, and 13 which comprise an RNN unit. At each time step $t$ there is input from the time series ($x_t$), and input from the previous hidden activation ($h_{t-1}$), which is the "history" of the time series through time $t-1$. LSTM cells/units have features that allow them to retain relevant "history" over long stretches of the time series. This is accomplished by placing input, output and forget gates within the LSTM cell. This creates paths for information to be propagated backward in time, thereby alleviating the vanishing gradient problem. An LSTM cell is shown in Figure 5. The central component of the LSTM cell is the state component, shown in red in Figure 5 [2]. At each time step, new information is combined with what is deemed relevant information from the previous time step

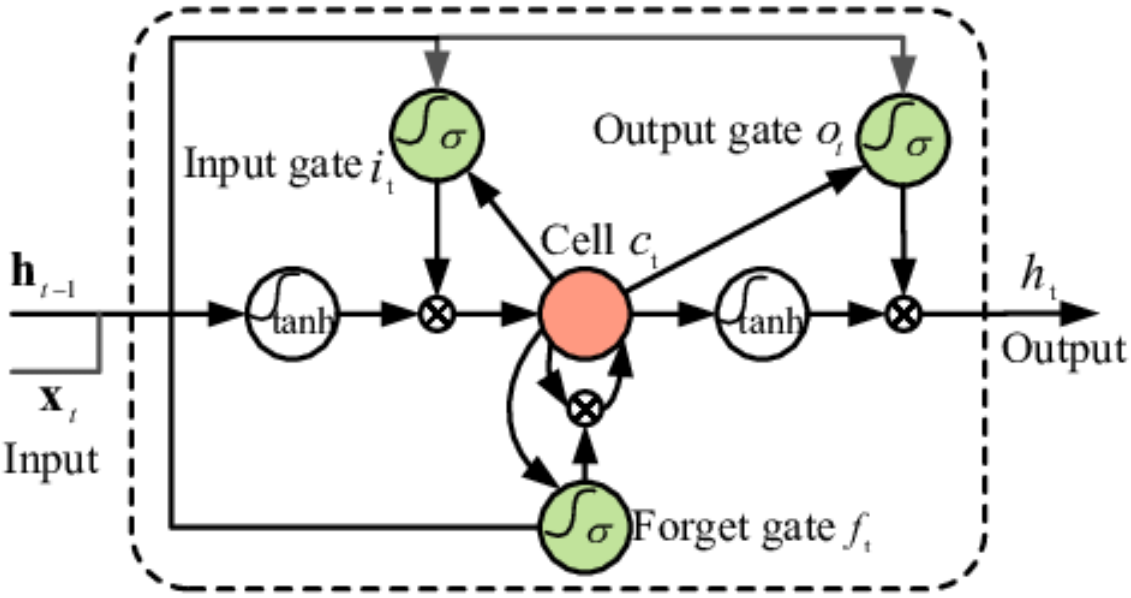



**Figure 5. LSTM cell [3].**

and then the state component ($c_t$) is updated. Note the following equations. First, information contained in $x_t$ and $h_{t-1}$ enters the cell via the linear combination

$$\boldsymbol{e}^{(t)} = tanh(\boldsymbol{b_e} + \boldsymbol{U_e}\boldsymbol{x}^{(t)} + \boldsymbol{W_e}\boldsymbol{h}^{(t-1)}). \tag{14}$$

The state component is then computed, drawing on $\boldsymbol{e}^{(t)}$ and the input and forget gates:

$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} * \boldsymbol{c}^{(t-1)} + \boldsymbol{e}^{t} * \boldsymbol{i}^{t} \tag{15}$$

where $*$ is the element wise product of vectors, $\boldsymbol{c}^{(t-1)}$ is the state component at time $(t-1)$, and $\boldsymbol{f}^{(t)}$ and $\boldsymbol{i}^{(t)}$ are the forget and input gates at time $(t)$ defined as:

$$\boldsymbol{f}^{(t)} = \sigma(\boldsymbol{b_f} + \boldsymbol{U_f}x^{(t)} + \boldsymbol{W_f}h^{(t-1)}), \tag{16}$$

$$\boldsymbol{i}^{(t)} = \sigma(\boldsymbol{b_i} + \boldsymbol{U_i}x^{(t)} + \boldsymbol{W_i}h^{(t-1)}) \tag{17}$$

Finally, the hidden output state ($h^{(t)}$) is computed with the output gate as follows:

$$\boldsymbol{o^{(t)}} = \sigma(\boldsymbol{b_o} + \boldsymbol{U_o}x^{(t)} + \boldsymbol{W_o}h^{(t-1)}) \tag{18}$$

$$\boldsymbol{h^{(t)}} = \boldsymbol{o^{(t)}} * tanh(\boldsymbol{c^{(t)}}). \tag{19}$$

**GRU**

Gated recurrent units are very similar to LSTM's, with one less gate. They have update $\boldsymbol{u}^{(t)}$ and reset gates $\boldsymbol{r}^{(t)}$ defined as [2]:

$$\boldsymbol{u^{(t)}} = \sigma(\boldsymbol{b_u} + \boldsymbol{U_u}x^{(t)} + \boldsymbol{W_u}h^{(t)}), \tag{20}$$

$$\boldsymbol{r^{(t)}} = \sigma(\boldsymbol{b_r} + \boldsymbol{U_r}x^{(t)} + \boldsymbol{W_r}h^{(t)}). \tag{21}$$

Given those gates, the hidden state update is as follows:

$$\boldsymbol{h^{(t)}} = \boldsymbol{u^{(t-1)}}\boldsymbol{h^{(t-1)}} + (1 - \boldsymbol{u^{(t-1)}})\sigma(\boldsymbol{b_h} + \boldsymbol{U_h}x^{(t)} + \boldsymbol{W_h}r^{(t-1)}h^{(t-1)}) \tag{22}$$

### 2.1.4 Temporal Convolutional Network Architectures

Convolutional neural networks (CNN's) perform well on data with a grid-like topology, which includes time series since they can be viewed as 1-dimensional gridded data [2]. Convolution as a general mathematical operation is defined as:

$$s(t) = \int x(a)w(t-a)da \tag{23}$$

for functions $x(t)$ and $w(a)$. In the context of CNN's, $x(t)$ from Equation 23 would be the input and $w(a)$ would be the kernel, or filter [2]. In practice, the kernel function is a tensor, whose entries are trainable parameters. Generally, kernels are significantly smaller than inputs, so convolution ends up reducing the dimensionality

of the original image as it travels through the network. Further, kernels can be designed to pick up on specific features of an input. For example, in the case of image processing, a kernel could be designed to identify vertical edges in a picture. Throughout subsequent convolutional layers the original input can be transformed into multiple lower dimensional representations that highlight various distinguishing features of the original input. To solidify how kernels are convolved with input data, the following example is provided. Take the input matrix $X \in \mathbb{R}^{5x5}$ and filter/kernel $k \in \mathbb{R}^{2x2}$:

$$X = \begin{bmatrix} 3 & 7 & 4 \\ 2 & 3 & 5 \\ 3 & 1 & 5 \end{bmatrix} \quad k = \begin{bmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{bmatrix}$$

The filter and image are convolved by sliding the filter across the image, and at each spot taking the sum product of the filter weights and image components. A separate hyperparameter called the stride, $s$, dictates how the filter slides around the input. A stride of one means it moves horizontally one entry at a time, while larger strides mean the filter skips $s-1$ entries horizontally and vertically. With a stride of one, the convolution of $X$ and $k$ yields:

$$X * k = \begin{bmatrix} 3 \times k_{11} + 7 \times k_{12} + 2 \times k_{21} + 3 \times k_{22} & 7 \times k_{11} + 4 \times k_{12} + 3 \times k_{21} + 5 \times k_{22} \\ 2 \times k_{11} + 3 \times k_{12} + 3 \times k_{21} + 1 \times k_{22} & 3 \times k_{11} + 5 \times k_{12} + 1 \times k_{21} + 5 \times k_{22} \end{bmatrix}$$

One additional hyperparameter that can be specified is padding. The input can be padded with zeros to give the filter more exposure to the edges of an input. Padding of one means there is one layer of zeros added to the outside of an image, padding of two means there are two layers, etc. Finally, in addition to convolutional layers, a CNN can have pooling layers where instead of convolving the image with a filter or kernel, there is some predefined operation applied to the image. For example, a max

pooling layer takes the max of each $n \times n$ sub-matrix of the input and outputs those maximum values. For example, a 2x2 max pool:

$$\text{Input Image} = \begin{bmatrix} 3 & 7 & 4 & 2 \\ 2 & 3 & 5 & 1 \\ 3 & 1 & 5 & 1 \\ 7 & 10 & 3 & 2 \end{bmatrix} \xrightarrow{\text{max pool output}} \begin{bmatrix} 7 & 5 \\ 10 & 5 \end{bmatrix}$$

As with all architectures, there are many reasons to choose certain striding, padding, or pooling layers and many ways to construct filters and convolutional layers. These will not be discussed further and are largely application dependent.

Growing interest in the application of CNN's to time series data has spurned extensive research into Temporal Convolutional Networks (TCN's). Temporal Convolutional Networks (TCN's) are convolutional networks designed to process time series and sequential data. Due to this generality, the term TCN can be confusing since it applies to many possible architectures and variants. That is to say, there is not one standard TCN architecture. To begin discussion, a pre-cursor to the idea of TCN's, and even to CNN's, will be discussed.

### Time Delay Neural Networks

In the early 1990's Waibel, Hanazawa, Hinton, Shikano and Lang [32] came up with Time Delay Neural Networks (TDNN's) in their effort to build a phoneme classifier, which is a time series classification task. Their TDNN was built to suit a couple of overarching goals: 1) effectively model the time series while making the network invariant to temporal translation in the input, and 2) give the network enough capacity to learn complex relationships without over parameterizing in relation to the amount of training data.

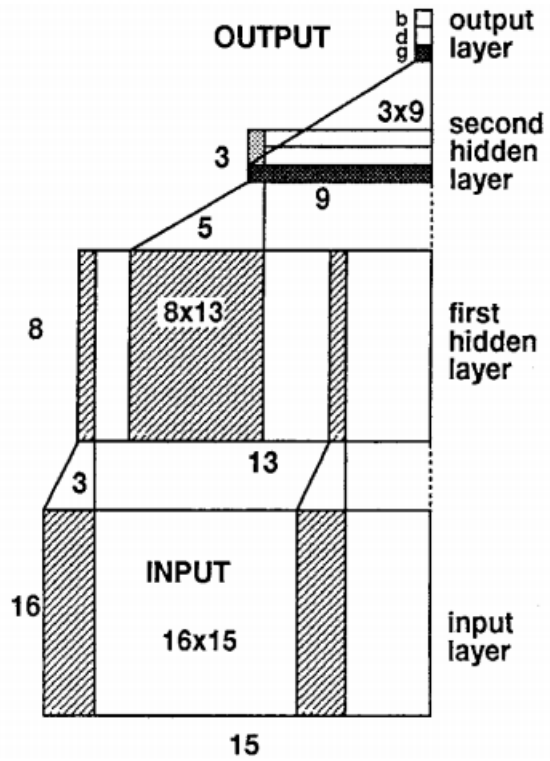



**Figure 6. TDNN architecture with two hidden layers [4].**

The architecture behaves similarly to that of a CNN. At each hidden layer, time delay filters are slid along the input. The size of these filters represent the size of the time delay, which allows past information to be related to the current input [32]. In the example of phoneme prediction provided by the authors, the input is 15 frames of 16 melscale spectral coefficients. This example is illustrated in Figure 6. Notice that the first hidden layer has width 13 since the 15 frame input is compressed by sliding the time delay filter of length 3 over the whole input. To ensure the network is invariant to temporal translation, only one set of parameters is learned for each layer's time delay filter. Specifically, there are a total of $16 \times 3 \times 8$ parameters connecting the input layer to the first hidden layer. So in the first hidden layer there are 8 time delay units each looking at three frame windows, and in the second hidden layer there are 3 time delay units each looking at five frame windows, with parameters being shared within layers [4].

### Formalizing TCN's

In 2018 Bai, Kolter, and Koltun [5] published a paper titled *An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling*. This paper attempts to establish a standard for what constitutes a TCN, and provides an overview of the architecture's history. It also provides a comparison of TCN vs RNN performance on a number of different tasks.

At the time this paper was written, TCN-like architectures had been successfully applied to a range of tasks: audio synthesis, natural language processing, and machine translation [5]. The authors set out to explore whether TCN's thrived only in these domains or whether they ought to be more seriously considered for all sequential and time series modeling tasks.

According to Bai et al. [5], a TCN is a generic convolutional sequence prediction architecture characterized in the following way. Consider an input sequence $x_0, \ldots, x_T$ and a corresponding output sequence $y_0, \ldots, y_T$ that is to be predicted. The goal is to find some mapping $f$ such that the loss between $y_0, \ldots, y_T$ and $f(x_0, \ldots, x_T)$ is minimized. The first defining principle of TCN's is that within the network the only information that can be used to predict an output $y_t$, is the inputs up until time $t$, i.e., $x_0, \ldots, x_T$. This is called causal prediction. The second is that the inputs and outputs must be the same length.

To abide by the first principle of denying leakage of information from future observations into current predictions, causal convolutions are used. Causal convolution is when an output in the network at time $t$ is not convolved with anything in the previous layer occurring after time $t$. To achieve the second principle, a 1D fully convolutional network (FCN) architecture is employed which uses zero-padding, and identical input and hidden layer lengths. The authors note that this is practically the same as the TDNN discussed above, aside from the requirement that the input and

output lengths be the same. Noting the work of Yu and Koltun [33] in their 2016 paper *Multi-Scale Context Aggregation by Dilated Convolutions*, the authors also employ dilated convolution which expands the receptive field of the network. For some sequence $\boldsymbol{x} \in \mathbb{R}^n$ and kernel/filter $f : \{0, \ldots, k-1\} \to \mathbb{R}$ the dilated convolution operator $F$ applied to element $s$, is defined as follows:

$$F(s) = (\boldsymbol{x} *_d f)(s) = \sum_{i=0}^{k-1} f(i) \cdot \boldsymbol{x}_{s-d \cdot i} \tag{24}$$

where $k$ is the filter size, $d$ is the dilation factor, and $s - d \cdot i$ enforces strict causality. A generic dilated and causal convolution example is shown in Figure 7.

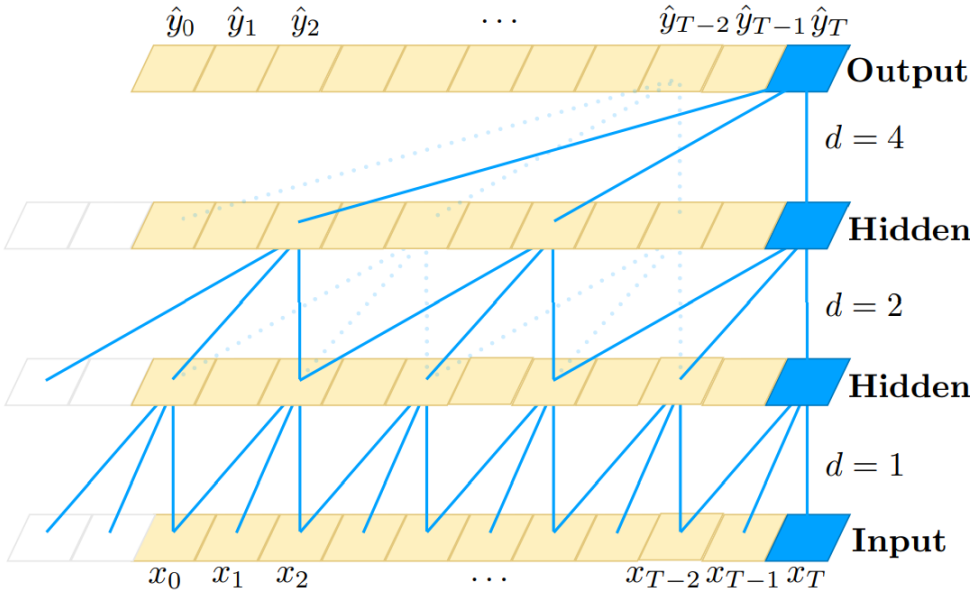



**Figure 7. Dilated and causal temporal convolution [5].**

### 2.1.5 Deep Learning Based Weather Forecasting

Despite the success of modern NWP in forecasting atmospheric dynamics, data driven approaches are used as well. Though some data driven methodologies rely on standard statistical models, most modern successes are largely driven by new advances

in the deep learning field.

In 2015, Shi et al. [34] used a convolutional LSTM to obtain precise precipitation forecasts over time scales of 0-6 hours. They note that the precision and resolution they were seeking could not be provided by existing NWP models. They used radar echo data (images), processed by an encoder-prediction architecture, to output future radar maps for a given local region. This amounts to a spatio-temporal sequence-to-sequence prediction problem. They note the efficacy of using a convolutional LSTM architecture as opposed to a more generic multi-layer perceptron (MLP) since it is a spatio-temporal prediction problem. Their model achieved state of the art precipitation "nowcasting" performance.

Dalto et al. [35] implemented a feedforward MLP for ultra short-term wind forecasting. They examined wind data from three sites in Croatia sampled at 10 minute intervals. In addition, they used output from a model called ALADIN to provide extra data corresponding to the grid point closest to each sampling site. The authors noted the improvement in predictive capability offered by both utilizing a deep, as opposed to shallow, neural network and input variable selection to reduce the size and interdependence of input data. However, note that this was all in the context of an MLP as opposed to an architecture designed for handling sequences and drawing the same conclusions for other architectures may not hold.

Hewage et al.'s [36] 2020 paper titled *Deep learning-based effective fine-grained weather forecasting model* explored the use of LSTM and TCN networks on both multi-input multi-output (MIMO) and multi-input single-output (MISO) models for short and long term forecasting. The dataset contains 10 weather attributes: surface temperature, surface pressure, x-wind (10m), y-wind (10m), specific humidity (2m), accumulated precipitation (convective rain), non-convective rain, snow water equivalent, soil temperature, soil moisture - all at 3 hour temporal and 10km spatial

resolutions. In a MIMO model, only one network is built - it makes predictions for all 10 variables at once, given the 10 inputs. In a MISO framework separate models are built using all 10 inputs to predict each output variable. The short term model takes 7 days of data as input and predicts 3 hours into the future, while the long term predicts 6, 9, 12, 24, or 48 hours out. The authors note their LSTM MIMO model, with 3 layers having 128, 512, and 256 nodes, outperforms the WRF NWP model for up to 12h forecasts while being far more efficient and light-weight.

Jeong et al. [37] studied the benefits of utilizing multiple types of input data over Korea. Specifically, they used observation time series data from AWS (automatic weather stations) along with radar image data from RDAPS (regional data assimilation and prediction system). The model used was tailored to the differences in the two datasets. Specifically, the AWS data was fed through bi-directional LSTM (bi-LSTM) layers, and the radar images through a convolutional bi-LSTM. Subsequently the two data streams were concatenated before the final prediction. The authors found that using both sources of data together, along with an attention layer for the concatenation/data fusion, yielded better results than models using each data source separately.

In a very recent paper (2021) Rasp and Thuerey [38] tried their hand at a medium-range weather forecasting benchmark challenge called "WeatherBench". The challenge is to predict the following attributes up to 5 days into the future: 500 hPa geopotential, 850 hPa temperature, surface temperature (2m), and 6 hourly cummulative precipitation. The input data included geopotential height, temperature, zonal and meridian wind, and relative humidity at 7 pressure levels (50, 250, 500, 600, 700, 850, and 925 hPa), surface temperature (2m), 6 hour cummulative precipitation, and top-of-atmosphere incoming radiation. This data is presented at three time periods (t, t-6h, and t-12h) at three constant fields (land-sea mask, orography, and latitude

at each grid point). To prevent overfitting, especially over longer forecast horizons, the proposed model was pretrained on a large climate model simulation dataset which spans 150 years at 6 hour temporal resolution. Their proposed model, a multilayered convolutional residual network, produced state-of-the-art results for data-driven modeling on the WeatherBench benchmark challenge [38].

Also recently (2021), Shultz et al. [39] published a paper titled *Can deep learning beat numerical weather prediction* in which they discussed the merits and challenges associated with a deep learning data-driven approach to weather forecasting as opposed to NWP. Specifically, they sought to explore whether deep learning could ever completely replace the existing NWP pipeline. They note two current reservations of many researchers with respect to deep learning methods: lack of explainability and lack of physical constraints. Early attempts at deep learning based forecasting began in the 1990's. The field slowly progressed with standard MLP neural networks making localized time series predictions and have improved greatly as more task specific architectures like CNN's, RNN's, etc. have improved. In order to mimic the entire NWP workflow process the deep learning pipeline would need to map observation data to final spatio-temporal forecasts while also addressing things like uncertainty estimation and enforcement of known physical relationships among variables [39].

### 2.1.6 Physics-based Characterization of the Atmosphere

The Navier-Stokes equations describe how a fluid's temperature, pressure, velocity and density change in relation to each other across space and time. This system of equations is at the core of atmospheric dynamics. Within this set of equations is a conservation of mass equation (continuity equation), a conservation of energy equation (First Law of Thermodynamics), and conservation of momentum equations. These equations can be expressed in various ways depending on the specifics of a given

application. One specific example being whether or not the fluid can be considered incompressible. Depending on the application, this may or may not be a reasonable assumption. For clarity, a fluid is incompressible if it's density does not change with changes in temperature and pressure. It is commonly accepted that if moving slowly, a fluid is effectively incompressible [40]. The ubiquitously used threshold for "slowly" is Mach 0.3, or roughly 230 mph depending on local atmospheric conditions [40]. Since it would be very abnormal for atmospheric gases near the surface of earth to move with that speed, we will consider the atmosphere to be an incompressible fluid. Finally, noting that the earth's atmosphere is a Newtonian fluid is also important going forward [41].

Consider an infinitesimally small volume at a fixed point in space with a fluid flowing through it. Conservation of momentum, or the equation of motion as it sometimes referred to, can be derived in a couple of ways. One natural way is using Newtons Second Law, which in this context says that the sum of forces acting on the volume is equal to the mass of fluid in the volume multiplied by the fluid's acceleration. In vector-tensor form, to compactly represent all three dimensions, this can be expressed as follows [42]:

$$\frac{\partial \rho \boldsymbol{v}}{\partial t} = -[\nabla \cdot \boldsymbol{\phi}] + \rho \boldsymbol{g} \tag{25}$$

The left side of Equation 25 represents the fluid's mass times it's acceleration, although it is expressed in terms of density due to division by volume across both sides. The right side contains the sum of forces acting on the volume of fluid. The $-[\nabla \cdot \boldsymbol{\phi}]$ term represents molecular and convective forces, while $\rho \boldsymbol{g}$ is the force due to gravity. By assuming the fluid is incompressible and noting it is Newtonian, with constant viscosity $\mu$, $-[\nabla \cdot \boldsymbol{\phi}]$ can be re-expressed as $(-\nabla p + \mu \nabla^2 \boldsymbol{v})$ [42]. Equations 26-28 provide the full set of momentum equations for each direction written in differential

form.

$$\rho g_x - \frac{\partial p}{\partial x} + \mu(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}) = \rho(\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z}) \tag{26}$$

$$\rho g_y - \frac{\partial p}{\partial y} + \mu(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 v}{\partial z^2}) = \rho(\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial v}{\partial z}) \tag{27}$$

$$\rho g_z - \frac{\partial p}{\partial z} + \mu(\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} + \frac{\partial^2 w}{\partial z^2}) = \rho(\frac{\partial w}{\partial t} + u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y} + w\frac{\partial w}{\partial z}) \tag{28}$$

The continuity equation, which describes conservation of mass, can also conveniently be developed by thinking of an infinitesimally small fixed volume at some point in space with a fluid flowing through it. The rate of change for mass in that volume is the rate of mass flowing in minus the rate flowing out [42]. Not considering incompressibility, this can be expressed as $\frac{\partial \rho}{\partial t} = -(\nabla \cdot \rho \boldsymbol{v})$, where the right side is the net rate of mass flowing in by convection [42]. However, for a volume of fixed size the only way for mass to change would be for density to change. Since we assume the fluid is incompressible, the rate of mass flowing in minus the rate flowing out will be zero, which makes the left side of the expression zero. Finally, the incompressibility assumption makes the divergence of $\rho \boldsymbol{v}$ reduce to $(\nabla \cdot \boldsymbol{v})$. Written out fully, the continuity equation can be expressed as:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0 \tag{29}$$

The equation for conservation of energy is an extension of the First Law of Thermodynamics and is derived using the overarching law of conservation, again applied to a fixed volume in space [42]. In words, the equation states that the rate of increase of total energy within the fluid volume is equal to the net rate of energy addition via heat from kinetic energy, internal energy, and molecular transport, plus the rate of work done on the fluid due to pressure forces, viscous forces, and gravity [42]. Equa-

tion 30 mathematically transcribes the verbal description, where $\hat{U}$ is internal energy, $\boldsymbol{q}$ is heat flux, and $\boldsymbol{\tau}$ is viscous flux. The expression does not contain temperature, and is instead represented using internal energy and heat flux vectors among other things.

$$\frac{\partial(\frac{1}{2}\rho v^2 + \rho\hat{U})}{\partial t} = -\left(\nabla\cdot(\frac{1}{2}\rho v^2 + \rho\hat{U})\boldsymbol{v} - (\nabla\cdot\boldsymbol{q}) - (\nabla\cdot p\boldsymbol{v}) - (\nabla\cdot[\boldsymbol{\tau}\cdot\boldsymbol{v}])\right) + \rho(\boldsymbol{v}\cdot\boldsymbol{g}) \tag{30}$$

Substituting enthalpy, which for a Newtonian fluid is a function of pressure and temperature, for internal energy, among other substitutions and manipulations reduces Equation 30 to the following:

$$\rho\hat{C}_p\frac{DT}{Dt} = -(\nabla\cdot\boldsymbol{q}) - (\boldsymbol{\tau}:\nabla\boldsymbol{v}) - \left(\frac{\partial \ln\rho}{\partial \ln T}\right)_p\frac{Dp}{Dt} \tag{31}$$

Using Fourier's law, and assuming constant thermal conductivity $k$, allows $-(\nabla\cdot\boldsymbol{q})$ to be expressed as $k\nabla^2 T$. Additionally, the viscous dissipation term $-(\boldsymbol{\tau}:\nabla\boldsymbol{v})$ can be omitted since the air at earth's surface has low viscosity, and velocity gradients are relatively small [42]. Finally, since we have been assuming incompressibility, the density is constant which makes the last term in Equation 31 equal to zero. All of this yields Equation 32, our final expression for conservation of energy.

$$\rho\hat{C}_p(\frac{\partial T}{\partial t} + u\frac{\partial T}{\partial x} + v\frac{\partial T}{\partial y} + w\frac{\partial T}{\partial z}) = k(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2}) \tag{32}$$

### 2.1.7 Physics Incorporated into Deep Learning

In 2017 a series of two papers titled *Physics Informed Deep Learning (Parts I): Data Driven Solutions of Nonlinear Partial Differential Equations* and *Physics Informed Deep Learning (Parts II): Data Driven Discovery of Nonlinear Partial Differential Equations* by Raissi, Perdikaris, and Karniadakis [43][44] were released dis-

cussing how and why existing knowledge regarding physical processes could be incorporated into neural network models.

Access to, and the existence of sufficient data can be prohibitive in training a powerful neural network. Lack of data often leads to over-fitting and the inability of the network to converge. Raissi et al. [43] claim that the modeling of many physical systems with machine learning could be buttressed by incorporating prior knowledge about how the system behaves from a theoretical standpoint. Not only does the use of a priori physical process knowledge add information to the model and make it more generalizable, it also encourages parsimony by pruning down the set of available models and excluding those that are unrealistic.

The two papers (Parts 1 & 2) cover two separate problems. They motivate the problem by considering a parameterized, nonlinear partial differential equation (PDE) in general form:

$$u_t + \mathcal{N}[u; \lambda] = 0, \tag{33}$$

where $u(t, x)$ is the hidden solution and $\mathcal{N}[\cdot; \lambda]$ is an operator with parameter $\lambda$. The first paper looks at how to estimate the unknown state of the system, $u(t, x)$, for fixed parameters $\lambda$. The second component looks at how to use observed data to find the parameters, $\lambda$, that best describe the data. So first, assume fixed parameters, continuous time, and partial differential equations that look like:

$$u_t + \mathcal{N}[u] = 0, \quad x \in \omega, t \in [0, T]. \tag{34}$$

Define a function of time and location, $f(t, x)$, equal to Equation 34:

$$f := u_t + \mathcal{N}[u]. \tag{35}$$

Now, the latent solution $u(t, x)$ is approximated by a neural network, which results in

the physics informed neural network (PINN) $f(t, x)$. To solidify this idea they pose the following example. Consider Burgers' equation, along with Dirichlet boundary conditions where $\lambda = [1, \left(\frac{0.01}{\pi}\right)]^T$:

$$u_t + uu_x - \left(\frac{0.01}{\pi}\right) u_{xx} = 0, \quad x \in [-1, 1], \quad t \in [0, 1], \tag{36}$$

$$u(0, x) = -sin(\pi x), \tag{37}$$

$$u(t, -1) = u(t, 1) = 0. \tag{38}$$

So, $f(t, x)$ is defined as,

$$f := u_t + uu_x - \left(\frac{0.01}{\pi}\right) u_{xx}. \tag{39}$$

A deep neural network can now be used to approximate the solution $u(t, x)$. The approximation of $u(t, x)$ and the PINN share the same network parameters, which are learned through minimization of a combined loss function. Specifically, the total loss is computed by adding two individual network losses together,

$$MSE = MSE_u + MSE_f, \tag{40}$$

where

$$MSE_u = \frac{1}{N_u} \sum_{i=1}^{N_u} |u(t_u^i, x_u^i) - u^i|^2 \tag{41}$$

$$MSE_f = \frac{1}{N_f} \sum_{i=1}^{N} |f(t_f^i, x_f^i)|^2. \tag{42}$$

The set $\{t_u^i, x_u^i, u^i\}_{i=1}^{N_u}$ is the training data for $u(t, x)$ and $\{t_f^i, x_f^i\}$ are the collocation points for $f(t, x)$. Equation 42 essentially forces the network to abide by Burgers'

equation by forcing it to be as close to zero as possible. The authors also outline the process in a discrete time context using the Runge-Kutta method. Accordingly, the Burgers' and initial/boundary conditions equations are represented discretely. Finally, the loss function is the sum of squared errors (SSE) in the discrete case, compared to MSE as before. Apart from this, the methodology is the same as the continuous case.

The second paper explores how observational data can be used to obtain PDE's. This time they let $f(t, x)$ now include $\lambda$,

$$f := u_t + \mathcal{N}[u; \lambda]. \tag{43}$$

Again, $u(t, x)$ will be approximated with a neural network and will be used to obtain the PINN $f(t, x)$. The Burgers' equation is used again as the motivating example [44]. So, as before we will let $f(t, x)$ be equal to the left hand side of the Burgers' equation. This will allow $f(t, x)$ within the loss function to act as a penalty for deviating from physically known behavior. In terms of the parameters $\lambda_1$ and $\lambda_2$, $f$ is now defined as:

$$f := u_t + \lambda_1 u u_x - \lambda_2 u_{xx}. \tag{44}$$

The parameters describing neural networks $u(t, x)$ and $f(t, x)$, as well as the parameters $\lambda_1$ and $\lambda_2$ are obtained by minimizing,

$$MSE = MSE_u + MSE_f, \tag{45}$$

where

$$MSE_u = \frac{1}{N} \sum_{i=1}^{N} |u(t_u^i, x_u^i) - u^i|^2 \tag{46}$$

$$MSE_f = \frac{1}{N}\sum_{i=1}^{N} |f(t_u^i, x_u^i)|^2 \tag{47}$$

The only difference between Equations 46 and 47, and Equations 41 and 42 is that now it is assumed there are the same number of collocation points as training data points. Additionally, they are now taken to be at the same location.

In experimentation, the network(s) were trained on 2,000 randomly drawn points across the spatio-temporal domain resulting from the exact solution to Burgers' equation with $\lambda_1 = 1,\ \lambda_2 = \frac{0.01}{\pi}$. The network predicted parameter values were very close to the known values of $\lambda$, along with predicting the solution $u(t, x)$ accurately, as in the first paper. In the discrete case, using Runge-Kutta stages, the authors also demonstrated the ability to predict both the parameters and solution using training data from two time instances which were far away. In concluding their research, the authors note that this was merely the beginning of finding ways to leverage existing physical knowledge about systems into data driven models like neural networks. More research is needed to further develop their ideas.

Bezenec et al. [6] offer a different type of deep learning solution for modeling the advection-diffusion class of physical processes in the context of sea surface temperature (SST) prediction. Classically, the physics based approach to this problem seeks solutions to the advection-diffusion equation [6]. Let $I(x, t)$, be some fluid quantity at location $x$ and time $t$. In this case, the ocean is the fluid and $I$ is the sea surface temperature (SST). Equation 48 describes the change of $I$ in terms of the motion vector $w = \frac{\Delta x}{\Delta t}$, and the diffusion coefficient $D$.

$$\frac{\partial I}{\partial t} + (w \cdot \nabla)I = D\nabla^2 I \tag{48}$$

Given an initial condition $I_0$, there exists a unique general solution to Equation 48,

$I(x,t)$:

$$I(x,t) = \int_{\mathbb{R}^2} = k(x - tw, y)I_0(y)\mathrm{d}y \tag{49}$$

where $k(u,v) = \frac{1}{4\pi Dt}e^{-\frac{1}{4Dt}\|u-v\|^2}$ is a radial basis function kernel (rbf). The difficulty in using this equation to obtain solutions is that $I_0$, $w$, and $D$ are not known. Bezenec et al. [6] use a deep learning architecture to estimate the motion vector $w$. They create a convolutional-deconvolutional neural network (CDNN) that takes $k$ previous satellite images containing temperature, and produce a corresponding motion, or velocity field $\hat{w}_t$ that estimates $w$. The temperature image at time $t$ is then used as $I_0$ in Equation 49, along with $\hat{w}_t$, to estimate $I_{t+1}$. They refer to this part of their process as a "warping scheme". A graphic of this end-to-end model is shown in Figure 8. This approach yielded promising results, achieving better performance than other machine/deep learning approaches and classical numerical models.

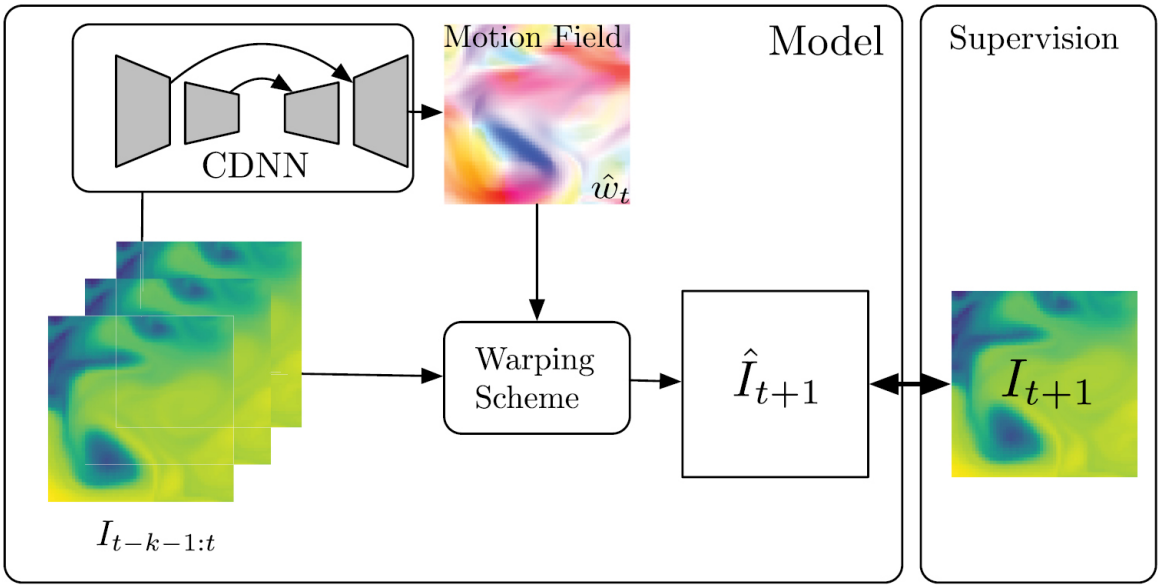



**Figure 8. End-to-end CDNN for estimation of motion vector, and prediction of $I_{t+1}$ [6].**

*Physics-Informed Echo State Networks for Chaotic Systems Forecasting* by authors Doan, Polifke and Magri [45] puts forth another approach to forecasting the

evolution of a choatic dynamical system with physics informed machine learning. The model follows a typical data-driven approach for time series prediction using an Echo State Network (ESN) however it is enhanced through the use of additional physical constraints. They propose the following two-part loss function:

$$E_{total} = E_{MSE} + E_{physical}, \tag{50}$$

where,

$$E_{physical} = \frac{1}{N_y} \sum_{i=1}^{N_y} \frac{1}{N_p} \sum_{p=1}^{N_p} |\mathcal{F}(\hat{y}_i(n_p))|^2, \tag{51}$$

$\mathcal{F}$ is a general non-linear operator that governs the dynamical system:

$$\mathcal{F} \equiv \partial_t y + \mathcal{N}(y) = 0, \tag{52}$$

and $\{\hat{y}(n_p)\}_{p=1}^{N_p}$ are $\mathcal{F}$'s collocation points. Equations 50 and 51 encapsulate their primary contribution. Specifically, in addition to their ESN learning to make predictions via $E_{MSE}$, those predictions will become increasingly physically consistent as a result of $E_{physical}$. Their approach was tested by predicting the evolution of the Lorenz system in which they noted remarkable improvement of their physics-informed ESN compared to the baseline ESN.

Karpatne et al. [7] published a collection of papers looking at physics-guided deep learning in the context of lake temperature modeling. One paper in particular, titled *Physics Guided RNNs for Modeling Dynamical Systems: A Case Study in Simulating Lake Temperature Profiles* gives particular focus to their physics guided deep learning methodology. Their specific problem area is temperature change throughout a lake over time $t$, and depth $d$. The model input, defined as $X = \{x_{d,t}\}$ and termed input drivers, are the physical variables that govern lake thermodynamics, such as lake

surface level readings for solar radiation, wind speed, air temperature, etc. The input drivers are then used to predict $Y = \{y_{d,t}\}$, which represents lake temperature at depth $d$, and time $t$. The data driven portion of their model is an LSTM - a separate one built for each depth of the lake. Two different physical laws were worked into the model. The first is energy conservation. This is integrated into their unrolled LSTM in Figure 9.

The authors define the change in thermal energy of the lake as:

$$\Delta U_t = R_{SW}(1 - \alpha_{SW}) + R_{LWin}(1 - \alpha_{LW}) - R_{LWout} - E - H, \tag{53}$$

where $\Delta U_t = U_{t+1} - U_t$, $R_{SW}$ and $R_{LW}$ are short and long wave radiation, $\alpha_{SW}$ and $\alpha_{LW}$ are short and long wave albedo, which indicate the proportion of short and long wave radiation reflected by lake, H are sensible heat fluxes and E are evaporative heat fluxes. Their proposed constraint associated with conservation of energy is:

$$\mathcal{L}_{EC} = \frac{1}{T_{ice-free}} \sum_{t \in ice-free} ReLU(|\Delta U_t - \mathcal{F}| - \tau_{EC}) \tag{54}$$

There are a couple of things to note about $\mathcal{L}_{EC}$. First, $T_{ice-free}$ refers to the fact that the authors only consider times when the lake is ice-free. Second, the authors use the threshold $\tau_{EC}$ to eliminate error associated with less important, unmeasured factors that affect the physical process, observation measurement error, etc.

The second physical law is a density depth constraint which is relevant because of the explicit relationship that can be derived between density and temperature. Specifically, the density-depth constraint leverages the fact that density monotonically increases with depth [7]. The authors use the following equation to relate temperature

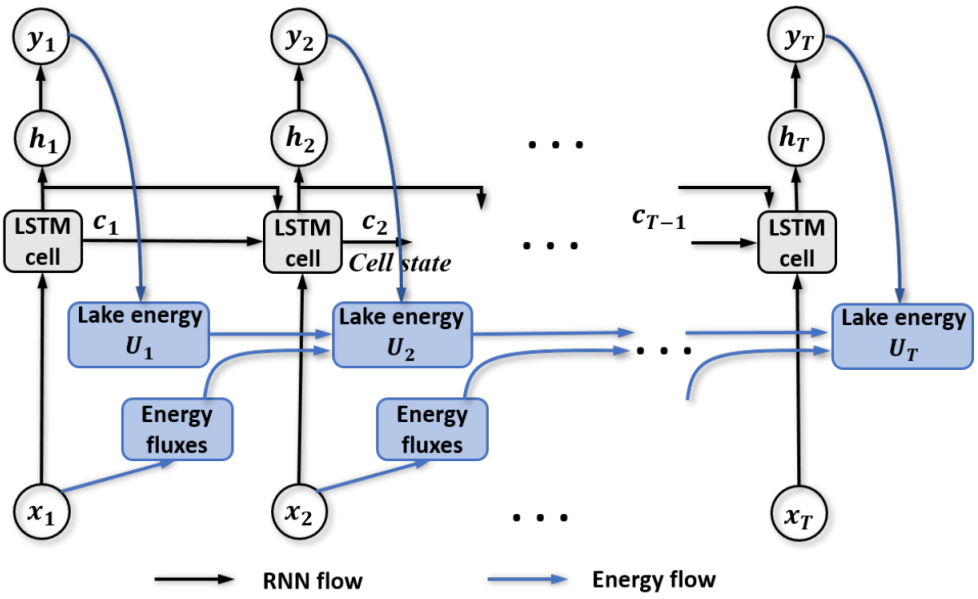



**Figure 9. LSTM with explicit conservation of energy modeling [7].**

and density:

$$\rho = 1000 \times \left(1 - \frac{(Y + 288.9414) \times (Y - 3.9863)^2}{508929.2 \times (Y + 68.12963)}\right) \tag{55}$$

So, if $\Delta\rho_{d,t} = \rho_{d,t} - \rho_{d+1,t} > 0$, then the model is penalized. The density-depth constraint formulation is:

$$\mathcal{L}_{DC} = \frac{1}{T(N_{d-1})} \sum_t \sum_d ReLU(\Delta\rho_{d,t}) \tag{56}$$

Combining the two physical constraints with the standard RNN loss, the total loss function is:

$$\mathcal{L} = \mathcal{L}_{RNN} + \lambda_{EC}\mathcal{L}_{EC} + \lambda_{DC}\mathcal{L}_{DC} \tag{57}$$

## 2.2 Clustering in Deep Weather Forecasting

Before presenting current research on clustering weather data, some clustering fundamentals will be discussed. Clustering is the act of grouping data together into

clusters such that the similarity of points within a cluster is maximized while the similarity between clusters is minimized, where similarity is measured using a clearly defined metric, measure, or discrepancy [46] [47]. In clustering static data points (traditional clustering), point similarity can be quantified with widely used distance metrics such as Euclidean distance, cosine similarity, Mahalonobois distance, etc. A vast suite of algorithms that use these various distance metrics then perform the actual clustering of data points. These algorithms can be broken down into four overarching categories: partitioning, density-based, hierarchical, and grid-based methods. Table 1, taken from *Data Mining* by Jiawei Han, Micheline Kamber, and Jian Pei [19], briefly describes each of these categories. If the reader would like a more thorough discussion of clustering static data points and specific algorithms corresponding to the methods from Table 1, Chapter 10 in [19] is a helpful resource. Specifics on the time series case will be discussed in the next section, but clustering algorithms can be largely the same as in the static case if the similarity measure is made to be time series friendly.

| Method | General Characteristics |
|---|---|
| Partitioning methods | • Find mutually exclusive clusters of spherical shape<br>• Distance-based<br>• May use mean or medoid (etc.) to represent cluster center<br>• Effective for small- to medium-size data sets |
| Hierarchical methods | • Clustering is a hierarchical decomposition (i.e., multiple levels)<br>• Cannot correct erroneous merges or splits<br>• May incorporate other techniques like microclustering or consider object linkages |
| Density-based methods | • Can find arbitrarily shaped clusters<br>• Clusters are dense regions of objects in space that are separated by low density regions<br>• Cluster density: each point must have a minimum number of points within it's "neighborhood"<br>• May filter out outliers |
| Grid-based methods | • Use a multiresolution grid data structure<br>• Fast processing time (typically independent of the number of data objects, yet dependent on grid size) |

**Table 1. Overview of four clustering algorithm categories according to [19]**

### 2.2.1 Current Applications of Clustering in Climatology

Clustering has been used in many different ways to aid in atmospheric modeling efforts. From the literature, it is evident that problems are most commonly addressed by treating data points as spatial snapshots in time instead of time series at a single location. For instance, imagine average daily surface pressure data is collected at 1 degree spatial resolution over the United states for a year. Clustering this data spatially would mean grouping locations together based on temporal similarity of the time series, where each data element being fed to the clustering algorithm is the time series for a single location. On the other hand, clustering temporally means that

similar time periods are grouped together over the entire region of interest.

Chattopadhyay et al. [48] propose a cluster based analog method for forecasting extreme temperature behavior over North America. The goal was to uncover the relationship between leading geopotential height at 500 mb (Z500) patterns and extreme temperatures. To do so, they first used standard Euclidean distance based clustering of daily averaged surface air temperature fields (2 meters above ground T2m) for both winter and summer separately by applying k-means to a principal components analysis (PCA) reduced version of the data, which yielded four clusters. An auto-labeling strategy was then used to assign Z500 fields with appropriate labels for their corresponding future temperature fields, i.e., assigned one of the four extreme temperature field indices or an index indicating no extreme temperatures are to come. They then fit neural network models $f : \boldsymbol{X} \rightarrow \boldsymbol{y}$, where $\boldsymbol{X}$ are Z500 patterns from 1-5 days leading up to a given temperature field, and $\boldsymbol{y}$ is an index specifying the type of extreme temperature field.

In another paper titled *Predicting clustered weather patterns: A test case for applications of convolutional neural networks to spatio-temporal climate data* by some of the same authors [49], similar research was done. Here the authors again propose a deep learning approach to categorizing weather data, specifically Z500 fields, that relies on cluster indices generated from K-means applied to PCA projections of daily pressure fields. The goals of the research were to see if a CNN could then learn to both re-identify the proper cluster labels, and/or predict the cluster labels of future Z500 fields.

Visualizing weather forecast ensembles, as done by Ferstl et al. [8] in *Time-hierarchical Clustering and Visualization of Weather Forecast Ensembles* is an entirely different application which called for a unique clustering approach. Due to the chaotic nature of the atmosphere, forecast quality degrades rapidly and the uncertainty re-

garding predictions grows. To help alleviate this issue the authors proposed time hierarchical clustering of forecast ensemble iso-contours to segment the most probable outcomes into groups. Before clustering, the $m$ iso-contours are transformed using a signed distance function (SDF) so that each iso-contour is represented as a point in high dimensional Euclidean space, and the full set of iso-contours is a point cloud in $\mathbb{R}^m$. This representation then allows for the use of standard clustering methods for the iso-contours. The authors choose an idea similar to agglomerative hierarchical clustering, with a slight variation to account for the fact that the iso-contours vary over time. The process is initialized by obtaining clusters for averaged iso-contours within a user-defined time-frame. Given the initial cluster assignments, the algorithm steps backward in time, merging iso-contour clusters (in SDF transformed space) that have similarity above a chosen threshold. An example of the resulting iso-contour clusters is shown in Figure 10.

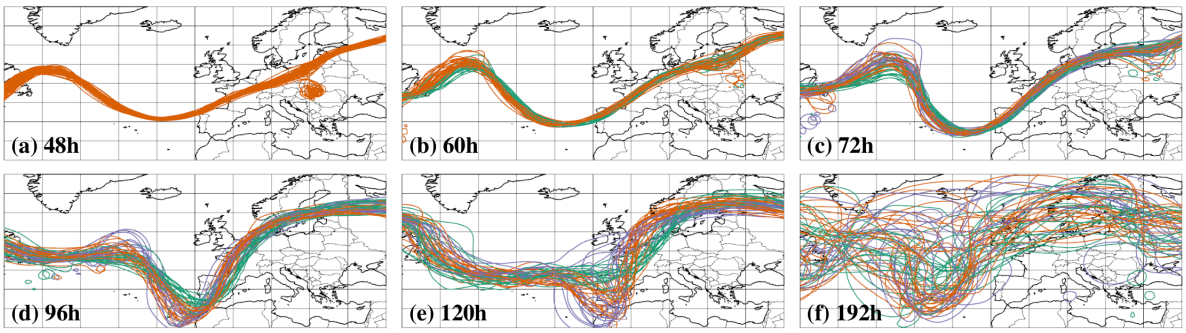



**Figure 10. Clusters merge backwards in time [8].**

*Long lead predictions of eastern United States hot days from pacific sea surface temperatures* by McKinnon et al. [50] uses a clustering based methodology to examine the relationship between precipitation deficits and the Pacific Extreme Pattern (PEP), and hot days. However, they hypothesize that it would be beneficial to divide the country up into regions which tend to experience hot weather at the same time, thereby establishing the relationship of these precursors to hot weather at a granular level. To do so they use station observation data of temperature near the earth's

surface for the 60 warmest days of summer. Hierarchical agglomerative clustering is applied to this information where the clustering variable is an indicator saying whether the station has a temperature above the $95^{th}$ percentile of its peak summer climatology on the same day as another station. To demonstrate their ability to predict hot days within their clusters by learning the relationship between precipitation and PEP, and hot days they chose to examine just one cluster which represented the Eastern part of the US. Based on their results they suggest that their clustering based approach, which focuses on regions with similar temperature anomalies, helped in identifying the relationships between hot days and the precipitation and PEP precursors.

Totz et al. [9] pose a solution to the problem of predicting winter precipitation in Europe and the Mediterranean, a task for which current methods are rather ineffective. Their approach was to cluster the region based on precipitation anomaly and then build a predictive model which learns the relationships between precursors (sea ice concentration, snow cover extent, sea surface temperature, geopotential height, and sea level pressure) and precipitation. For clustering, the precipitation anomalies are arranged as vector data points, and clustered using hierarchical agglomerative clustering presumably with a standard Euclidean based distance metric. The resulting precipitation anomaly clusters are shown in Figure 11. The authors note that their model, which relies at its core on the clustering of precipitation anomalies, outperformed both climate models and canonical correlation analysis in predicting precipitation anomalies in the region.

As alluded to in the review done on Ferstl et al.'s [8] work on ensemble forecasting, there can be considerable uncertainty in forecast estimates. In their work titled *Clustering numerical weather forecasts to obtain statistical predictions*, Zarnani et al. [51] argue that quantifying this uncertainty can be as important as the actual forecast itself. The field of weather modeling already has methods for describing

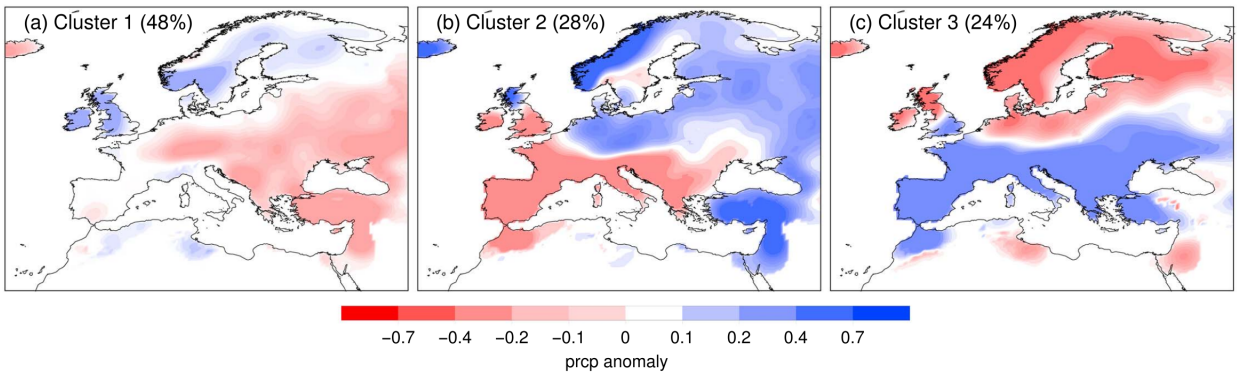



**Figure 11. European and Mediterranean precipitation anomalies ordered by frequency [9].**

this uncertainty however doing so is very computationally expensive so the authors proposed addressing the issue from the error uncertainty side of statistical forecasting. They posit that places with similar weather forecasts would also share similarity in forecast error patterns. So, they propose clustering predictions and then performing their statistical error analysis on the forecast errors within each cluster. The clustering was done by treating the predictions made for various attributes at a given time as the data points and then running standard clustering algorithms like k-means, CLARA, and hierarchical clustering on those data points.

The final piece of work that will be discussed looks at using clustering to discover climate indices, which are time series that describe atmospheric behavior at local or regional scales and it's relation to other events [52]. Specifically, given vast amounts of data being collected as of late, the authors set out to use unsupervised machine learning methods to extract ocean climate indices from historical data, whereas in the past discovery of such indices had largely been observational. Additionally, the authors test their clustering results as predictors by using them to predict air temperature and precipitation at 9 locations around the world. They tested a couple of different clustering approaches, but their primary approach was a network based clustering method, in which they actually did make use of time series data directly.

The used monthly averages of 7 weather attributes over a 60 year period: sea surface temperature, sea level pressure, geopotential height at 500 mb, precipitable water, relative humidity, and horizontal and vertical wind speeds. In their network clustering approach each grid point location for which data was collected served as a vertex in the graph, and the edges were the strength of connection between weather attribute time series at different locations. To measure similarity across locations they used Pearson's correlation coefficient. Furthermore, to avoid connecting every grid point to every other grid point in the graph, thresholding was used to only allow grid points to be connected if their similarity with respect to a given attribute was over the threshold. Separate clustering networks were built for each of the seven variables, and the specific network based clustering algorithm was a community detection algorithm which will not be discussed in detail [53][52]. They did indeed find that their network based clustering approach aided in predictive power for air temperature and precipitation across the selected locations by 35% over the current baseline.

### 2.2.2 Dynamical Systems

Considering the difficulty meteorologists can face in predicting the weather, one may be tempted to claim such problems arise because of randomness in the atmosphere. However, the atmosphere is a deterministic system. The irregularities that arise in this case are actually attributes of a nonlinear chaotic dynamical system. We observe dynamical systems by measuring one or more of their characteristics, which form (multiple) time series. There is myriad research devoted to gleaning information from time series data, much of which is very useful in atmospheric forecasting. However, there is also value in studying the trajectory, or evolution of the system in phase space as this can be an optimal way of examining the system's dynamics [54].

First consider a simple deterministic dynamical system - a swinging pendulum.

The state of this system at a given time $t$ can be fully described by its velocity and angle. Over time the system reaches equilibrium, or a fixed point in phase space as shown in Figure 12. A fixed point is one of three different types of attractors. An attractor is a sub-set of phase space that the system is attracted to following a transient period, given a particular set of initial conditions. The second type of attractor,

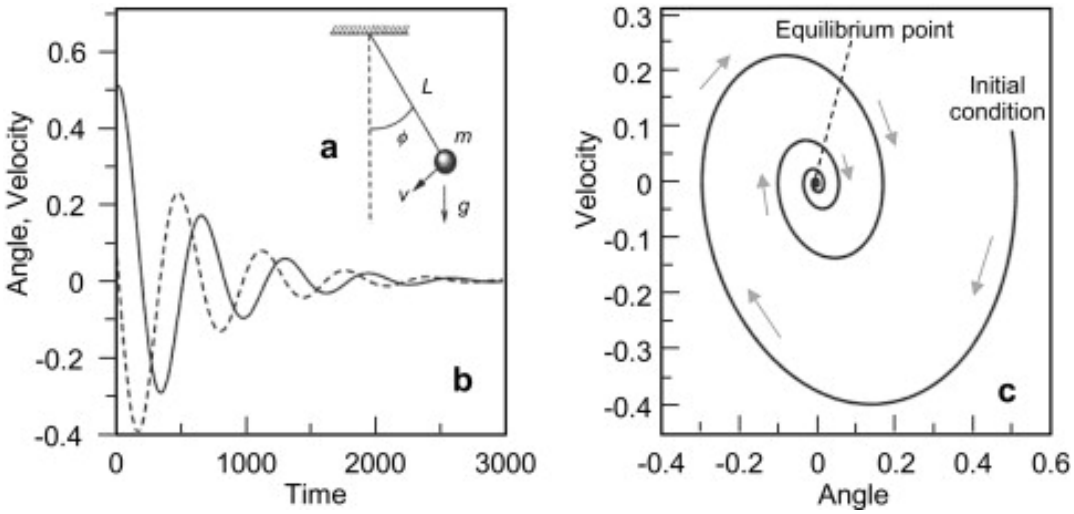



**Figure 12. Time series and associated phase space representation of swinging pendulum [10].**

also non-chaotic, is a limit cycle which results from a system with periodicity. As can be imagined, this trajectory manifests as a circle or ellipse. Finally, there are strange, or fractal attractors which result from chaotic systems such as the atmosphere. The Lorenz attractor, as depicted in Figure 13, is a prime example. In contrast to more "tame" attractors like fixed points and limit cycles which arise from non-chaotic systems, strange attractors often exhibit complex shapes [54]. Studying the shape, or topology, of these attractors can be valuable in describing their dynamics.

In addition to topology, dynamical systems' Lyapunov exponents are another useful characteristic to analyze. Lyapunov exponents are a way of measuring how far apart two trajectories will become starting from nearby initial conditions [55]. Both the topology of attractors, Lyapunov exponents, and other dynamical system mea-

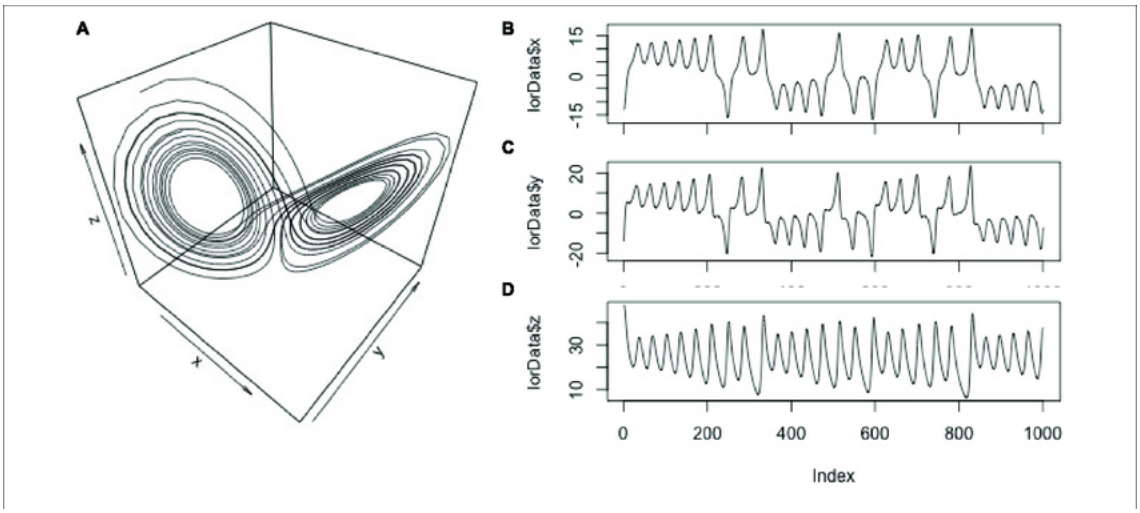



**Figure 13. The Lorenz model is described by three variables whose time series are shown along with it's attractor [11].**

sures will now be described in greater detail, as well as some roadblocks in performing this type of analysis.

### Time Delay Embedding: Takens Embedding Theorem

In the examples provided above, Figures 12 and 13, the attractor can easily be obtained by simply taking each time series to be a dimension and plotting parametrically. This requires that all quantities which comprise the system have been measured and are available. Unfortunately, this is often times not the case. In fact, it is not uncommon to be entirely unaware of a system's true dimensionality and variable composition. In such a case, Taken's embedding theorem provides a way to reconstruct an attractor that is diffeomorphic (topologically equivalent) to the true attractor under certain conditions and assumptions. Before presenting Taken's Theorem, two important definitions are provided:

[Diffeomorphism] A function $f : U \longrightarrow V$, where $U$ and $V$ are open subsets of $\mathbb{R}^m$, is a diffeomorphism if [56]:

- $f$ and $f^{-1}$ are smooth to degree $r$ ($C^r$ for $r \in \mathbb{W}$)

- $f$ is bijective

[Homeomorphism] A function $f : U \longrightarrow V$, where $U$ and $V$ are open subsets of $\mathbb{R}^m$, is a homeomorphism if [57]:

- $f$ is bijective
- $f$ and $f^{-1}$ are continuous

[Embedding] An embedding is a homeomorphic map $f : \mathbb{X} \longrightarrow \mathbb{Y}$, where $\mathbb{X}$ and $\mathbb{Y}$ are topological spaces [57][58].

**Theorem 1** *Let $M$ be a compact manifold with dimension $m$. Then for pairs $(\phi, y)$ where $\phi : M \longrightarrow M$ is a diffeomorphism and $y : M \longrightarrow \mathbb{R}$ is a smooth (at least twice differentiable, $C^2$) function, it is a generic property that the map $\Phi_{(\phi,y)} : M \longrightarrow \mathbb{R}^{2m+1}$, which is defined by*

$$\Phi_{(\phi,y)}(x) = (y(x), y(\phi(x)), \ldots, y(\phi^{2m}(x)) \tag{58}$$

*is an embedding [59].*

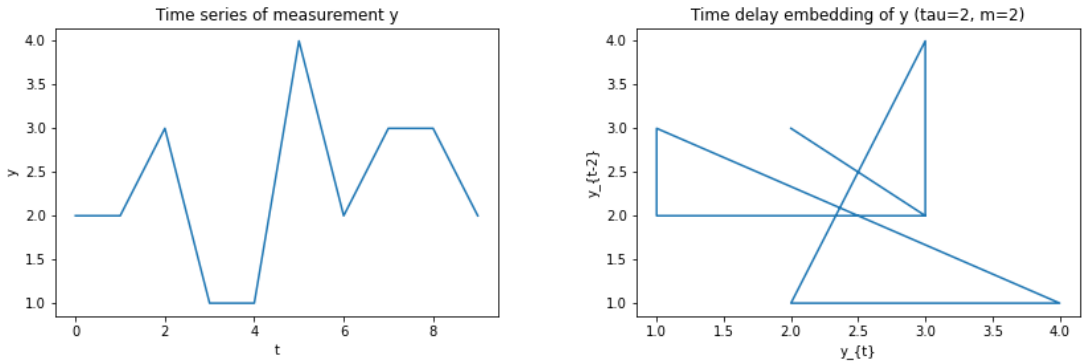
From a dynamical systems perspective, $\phi$ is thought of as the function that moves the system from one state to the next along $M$, and $y$ is a measurement or observable of the system. Finally, the function $\Phi$ is simply a time delay coordinate embedding. Takens' theorem assumes a number of things that may not hold in practice. These issues, along with some historically proposed solutions will now be explored.

Takens' theorem assumes arbitrarily precise measurements and an infinite stream of output from $y$ [60][61]. However, for many (if not all) real world problems this is not a realistic assumption. In *State space reconstruction in the presence of noise* Casdagli et al. [61] highlight three possible problems with real world data, the first two relating to noise and the last a result of finite data:

- Observational noise: Noise stemming from the measuring instrument.
- Dynamic noise: Influences external to the system perturbing the system, making $\phi$ appear stochastic.
- Estimation error: Stems from lack of knowledge of $\phi$ and $y$. Estimating dynamics in a reconstructed phase space with finite measurement data always results in an imperfect approximation.

Noise is problematic in that a noisy measurement could have been generated by many different possible states. While noise reduction techniques on the time series could prove helpful, the authors stress the importance of having a good state space reconstruction as mitigation against issues arising from noise [61]. A good reconstruction largely depends on two quantities, which have not yet been discussed but are of significant importance: the embedding dimension $m$, and the time delay $\tau$. The choice of $m$ determines the number of coordinates for the reconstruction space, and $\tau$ is the time delay between each coordinate, or element of the time series. For example, consider the measurement vector $y = [2, 2, 3, 1, 1, 4, 2, 3, 3, 2]$, and parameters $\tau = 2$ and $m = 2$. The reconstruction will be non-sensical since the data is contrived, however a simple example illustrates the mechanics nicely. In this case, since $m = 2$, we know this will result in a 2-dimensional embedding. Then with time delay $\tau = 2$, each pair of successive points will be composed of entries $(y_t, y_{t-2})$. The measurement vector and time delay embedding for this example are shown in Figure 14.

There has been substantial research done on picking parameters $m$ and $\tau$ as it is central to a good reconstruction. Too small of a $\tau$ and the reconstruction suffers from "redundance". This is result of each coordinate being very similar, since they are so close in time, which causes the attractor to flatten along a diagonal [61]. Alternatively, if $\tau$ is too large and the dynamical system is chaotic and noisy then the elements of




**(a)** **(b)**

**Figure 14. A contrived example showing a time delay embedding of a time series $y$, using parameters $m = 2$, and $\tau = 2$.**

each coordinate become effectively unrelated dynamically. This is called "irrelevance", and results in overly complex attractor shapes. In choosing $m$ it is important to remember that Takens' theorem simply provides a sufficient condition for obtaining an embedding. This can turn out to be an overly conservative estimate of the number of dimensions needed though, as in the Lorenz system which has three variables and can be embedded in three dimensions. In some circumstances, embedding in the smallest reconstruction space required can be critical. This is because each dimension that is added to the embedding space requires additional data for the reconstruction. In their 1993 paper titled *Estimating the Dimension of Weather and Climate Attractors: Important Issues about the Procedure and Interpretation*, Tsonis et al. [62] discuss this issue. For embedding dimension $m$, they put forth the following rule of thumb:

$$N_{min} \propto 10^{(2+0.4m)}. \tag{59}$$

They also cite some work by Lorenz [62] in 1991 which suggests that for some chaotic systems performing the embedding on variables that are known to be highly related to other variables in the system can improve results. Now, after motivating the need to take care in choosing the time delay $\tau$ and embedding dimension $m$, we

will briefly outline a couple of common approaches to doing so.

Takens' theorem guarantees that if the embedding dimension $m$ is greater than $2d+1$, where $d$ is the true dimensionality of the system, then the system dynamics can be embedded using a single time series. Though we have discussed this often times being a conservative estimate, it is beneficial to have an upper bound. In numerous applications though, this calculation is meaningless as a way for choosing $m$ since the true dimensionality of many systems is not known. For this reason, as well as the desire to embed in the smallest dimension possible due to data constraints, we will describe the false nearest neighbor approach to choosing $m$. The false nearest neighbor approach relies on the idea that two neighboring states in a dynamical system will remain close neighbors a short distance into the future since they have almost the same trajectories [54]. It provides a way to verify whether a given embedding dimension $m$ is sufficient. Let $m_{true}$ be the true embedding dimension for a system. Now consider $m_{insufficient} < m_{true}$, which is a projection that eliminates certain axes from $m_{true}$. The elimination of axes can make points in an $m_{insufficient}$ embedding appear to be neighbors, when they are actually not neighbors in the correct embedding dimension [54]. Thus the crux of the false nearest neighbors method: iteratively search pairs of dimensions until the ratio of false nearest neighbors between pairs of dimensions is lower than some threshold [54]. The statistic commonly used is

$$X_{fnn}(r) = \frac{\sum_{n=1}^{N-m-1} \Theta\left(\frac{\left|\boldsymbol{s}_n^{(m+1)} - \boldsymbol{s}_{k(n)}^{(m+1)}\right|}{\left|\boldsymbol{s}_n^{(m)} - \boldsymbol{s}_{k(n)}^{(m)}\right|} - r\right) \Theta\left(\frac{\sigma}{r} - \left|\boldsymbol{s}_n^{(m)} - \boldsymbol{s}_{k(n)}^{(m)}\right|\right)}{\sum_{n=1}^{N-m-1} \Theta\left(\frac{\sigma}{r} - \left|\boldsymbol{s}_n^{(m)} - \boldsymbol{s}_{k(n)}^{(m)}\right|\right)}, \tag{60}$$

where $\sigma$ is the standard deviation of the data, and $\boldsymbol{s}_{k(n)}^{(m)}$ is the nearest neighbor of $\boldsymbol{s}_n$ in $m$ dimensional space.

There is no optimal method for estimating the time delay $\tau$ since in the theorems the data is assumed to be arbitrarily precise (noiseless). In practice though, this choice

can be important in avoiding redundance and irrelevance as discussed previously. One common heuristics involves looking at the mutual information of a signal and delayed versions of itself at various choices of $\tau$ and then choosing the first minimum. This indicates the first point at which the signal contains maximum additional information compared to the lagged signal [54]. Technically though, this argument is only valid in two dimensional embeddings [63]. Despite this, it is often still used. In their paper titled "*Criterion for determining the optimal delay of attractor reconstruction using persistent homology*", Tsuji and Aihara [64] suggest choosing the first $\tau$ that maximizes the persistence of holes in the reconstructed attractor. Said another way, they choose the first $\tau$ that maximally widens and separates holes in the attractor. Noisiness in the data is also implicitly taken into consideration.

### 2.2.3 Nonlinear Time Series Analysis of Dynamical Systems

While phase space reconstruction and topological analysis of the resulting attractors is a direct way of quantifying the properties of dynamical systems, there are other methods that can be used to get at some of the same information. Various quantities like, the Lyapunov exponent, de-trended fluctuation analysis, sample entropy, and correlation dimension, indirectly provide some of the same information that would be gleaned from attractor analysis. Some of these quantities will now be discussed.

#### Lyapunov exponent

Lyapunov exponents are a way of measuring the chaos inherent in a dynamical system. In the realm of dynamical systems chaos is often taken to mean "aperiodic long-term behavior in a deterministic system that exhibits sensitive dependence on initial conditions" [65][66]. Lyapunov exponents reflect this sensitive dependence on initial conditions (SDIC) by calculating how much two trajectories in phase space

diverge over time after having originated close to one another [66]. Dynamical systems have as many Lyapunov exponents as there are dimensions of the system, however often times the largest one is given most consideration since it reports the highest average exponential rate of divergence in trajectories [66]. Finally, a deterministic system is considered chaotic if the largest Lyapunov exponent is positive.

### Sample Entropy

Sample entropy describes the predictability or complexity of a given time series, [67][68]. Consider a time series $y \in \mathbb{R}^N$. The sample entropy (SampEn), with parameters $m =$ number of state variables in the system, $r =$ chosen threshold, and $\tau =$ interval between time series elements is [67]:

$$SampEn(m, r, N) = -ln\frac{I^{m+1}(r)}{I^m(r)}, \tag{61}$$

where,

$$I^{m+1}(r) = \frac{1}{N - m\tau} \sum_{j=1}^{N-m\tau} C_j^m(r) \tag{62}$$

and

$$C_j^m(r) = \frac{S_j^r}{N - (m+1)\tau}. \tag{63}$$

Still not yet defined is the parameter $L[Y(j), Y(k)]$, which is the distance, given any number of distance metrics, between vectors $Y(j)$ and $Y(k)$. These vectors are defined as [67]:

$$Y(j) = \{y_j, y_{j+\tau}, \ldots, y_{j+(m-1)\tau}\} \tag{64}$$

$$Y(k) = \{y_k, y_{k+\tau}, \ldots, y_{k+(m-1)\tau}\} \tag{65}$$

with both $j$ and $k$ not equal to one another and ranging from 1 to $N - m\tau$. Finally,

with these definitions, the quantity $S_j^r$ from Equation 63 is the number of times $L[Y(j), Y(k)] \leq r$ [67]. From the definition, the number of times $L[Y(j), Y(k)] < r$ in the $(m+1)$ case will always be less than or equal to the number of times in the $(m)$ case. Therefore, SampEn is always greater than or equal to zero. Finally, smaller values of SampEn correspond to greater predictability and less complexity, while the opposite is true for higher values of SampEn.

### Correlation Dimension

The fractal dimension of a system is another quantity often of interest in characterizing dynamical systems. One popular way of calculating the fractal dimension is the correlation dimension algorithm [69]. The process involves deriving a correlation integral $C(r)$, which corresponds to the probability that any two random points in state space orbit are closer together in distance than $r$. Given $C(r)$, which is computed by calculating the pairwise distance between all $N$ data points, and assigning to bins according to width $\Delta r$ proportional to $r$ [70], the correlation dimension is [69]:

$$\nu = \lim_{r \to 0} \lim_{N \to \infty} \frac{d\log C(N, r)}{d\log r}. \tag{66}$$

### Detrended Fluctuation Analysis

The detrended fluctuation analysis (DFA) method was derived to study long-range correlations in DNA nucleotides without any assumptions about stationarity of input data [71][72][20]. As described by some of the original authors in early work on DFA, the process is as follows. Obtain a cumulative sum of the original time series which will be denoted $y(k)$. Then for varying window sizes $n$ of the cumulative sum, ranging from four elements to the entire series, detrend the data in each box by subtracting each box's least squared linear fit $y_n(k)$. Equation 67 is then used to calculate the

root-mean-squared fluctuation of $y(k)$ and $y(k) - y_n(k)$.

$$F(n) = \sqrt{\frac{1}{N}\sum_{k=1}^{N}[y(k) - y_n(k)]^2} \tag{67}$$

Repeating this computation over various size time windows yields a collection of $n$'s and $F(n)$'s, which are then plotted on a double log scale. The resulting slope of this line is denoted as $\alpha$, and is called the scaling exponent. Table 2 explains the meanings of different values of $\alpha$.

| **$\alpha$ value(s)** | **Corresponding behavior** |
|---|---|
| $\alpha = 0.5$ | Uncorrelated data |
| $0.5 < \alpha \leq 1$ | persistent long-range power law correlations |
| $0 < \alpha < 0.5$ | negative (power law) correlations |
| $\alpha = 1$ | $1/f$ noise |
| $\alpha \geq 1$ | existent correlations but of power law form |
| $\alpha = 1.5$ | Brownian noise |

**Table 2. The meanings of different levels of the scaling exponent $\alpha$ [20].**

## 2.3 Deep Representation Clustering for Time Series

### 2.3.1 Time Series Clustering

The task of clustering time series is not as straightforward as that of clustering static data. Most problematic is that it is not obvious what it means for two time series to be "similar". In some cases, the time series overall movement patterns could be more important than strict point-wise similarity. Alternatively, it could be more meaningful to compare statistics of the time series instead of it's raw values, or fit models to the time series and compare their parameters. Differences in time series lengths can also cause complications.

Time series clustering can be broken into raw-data or shape, feature, and model based approaches [12][73]. Raw-data based approaches compare actual time series

values themselves or associated frequency information to determine similarity. Feature based approaches extract features from the time series and then employ static data clustering algorithms. Model based approaches assume that each time series was generated from a model and so they compare models as proxies for the original time series [73]. Additionally, researchers sometimes combine aspects from multiple approaches as shown in Figure 15.

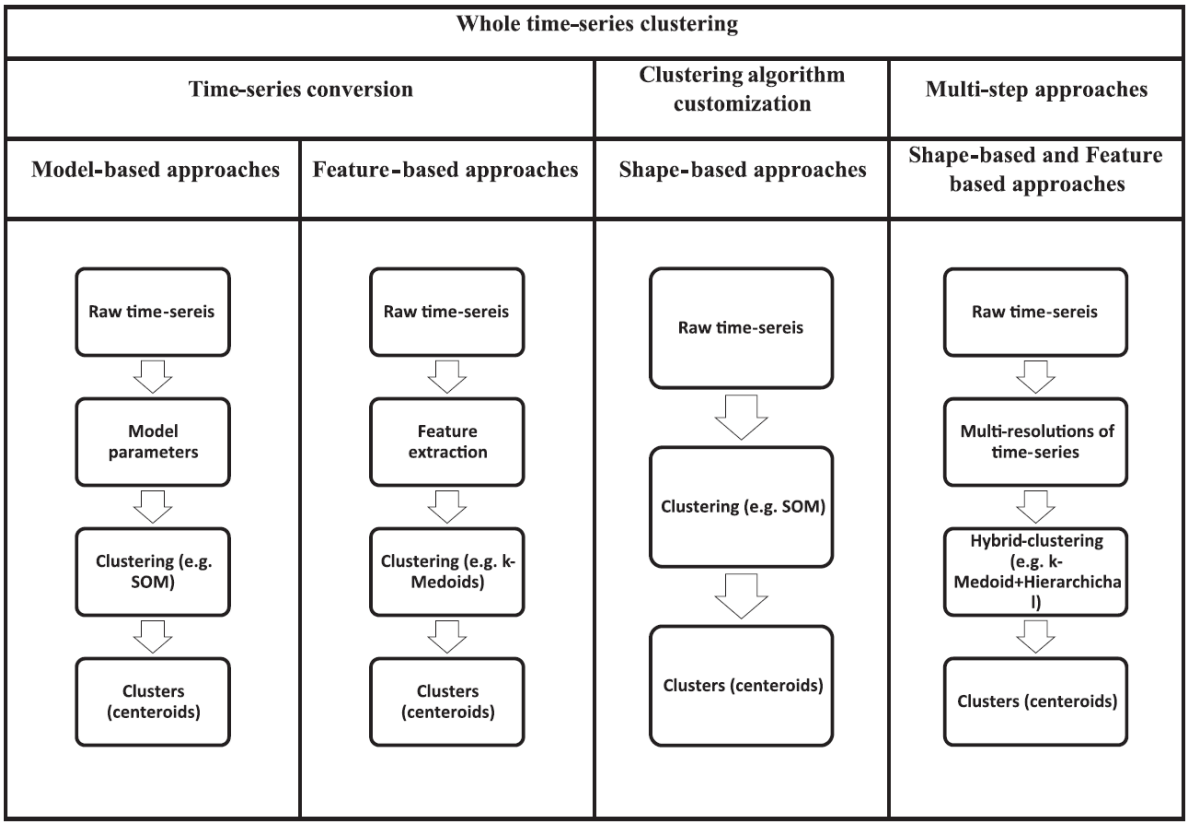



**Figure 15. Time series clustering approaches flowchart [12].**

### Algorithmic Adaptation for Time Series

Lloyd's algorithm, a partitioning algorithm that acts as an approximation to K-means, provides an excellent opportunity to explore how a centroid-based partitioning algorithm works, and where it must be modified to handle time series data [74]. The algorithm (Algorithm 1) begins with an initial set of $k$ cluster centers, then alternates

between assigning data points to the nearest (or most similar) cluster center and updating cluster centers based on the data points assigned to it. It goes on like this for a predertmined number of iterations or until some convergence criteria is met.

**Algorithm 1** Lloyd's Algorithm

**Inputs:**
Initial cluster centers and data points
**Hyperparameters:**
MaxIter = maximum number of iterations
k = number of clusters
**Outputs:**
Cluster centers and assignments
1: **for** 1 : MaxIter **do**
2: Assign data points cluster of nearest centroid
3: Update cluster centroids given new data point assignments
4: **end for**
**return** Cluster centers and assignments

Both of these main steps are straightforward in the static data case. The assignment of data points to new clusters could be done using Euclidean distance, and then the cluster centers updated to be the centroid (arithmetic mean) of each cluster. However, as alluded to in Chapter I, Euclidean distance and arithmetic mean centroid calculations are not appropriate for time series in most scenarios. Figure 1 from Chapter I shows how this is problematic. In that example the Euclidean distance between the two sin waves does not account for phase shifts since it is a simple pointwise calculation, and is therefore a poor proxy for similarity. Instead researchers often look to dynamic time warping for a more robust time series distance measure.

### 2.3.2 Dynamic Time Warping

Dynamic time warping (DTW) is a dynamic programming based approach for finding an optimal alignment between two sequences such that a given distance measure is minimized [75]. For sequences $S \in \mathbb{R}^n$ and $T \in \mathbb{R}^m$, the set of all possible alignments of the two sequences is represented by an $n \times m$ grid whose elements are

the distances, with respect to a given distance metric $\delta$, between all points in the two sequences. The optimal alignment $W$ then is given by the path with minimum cumulative distance over the set of all possible alignments:

$$DTW(S,T) = min_W[\sum_{k=1}^{p} \delta(w_k)] \tag{68}$$

A number of constraints can be placed on the problem, both to reduce the search space and enforce sensible alignments. A few common ones are [75]:

1. Monotonicity with respect to time $\rightarrow$ $\mathrm{i}_{k-1} \leq i_k$ and $j_{k-1} \leq j_k$
2. Path must be continuous $\rightarrow$ $\mathrm{i}_k - i_{k-1} \leq 1$ and $j_k - j_{k-1} \leq 1$
3. Warping path cannot veer outside of a given bandwidth ($\omega$) from the diagonal $\rightarrow$ —$\mathrm{i}_k - j_k| \leq \omega$

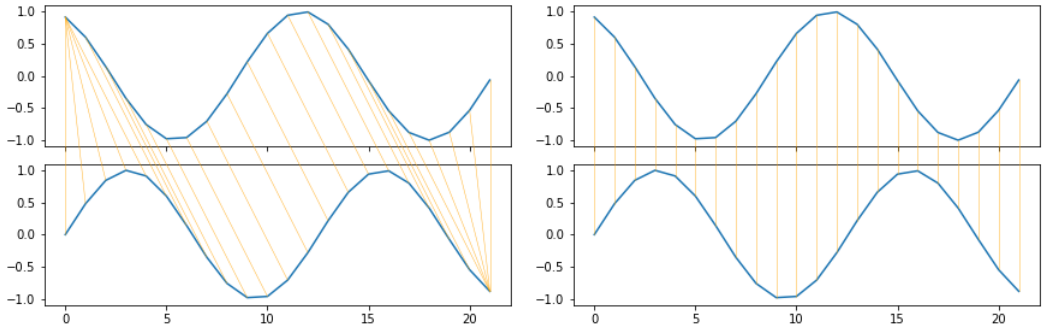
The following recurrence relation forms the core of the dynamic programming formulation:

$$r(i,j) = \delta(i,j) + min\left[r(i-1,j), r(i-1,j-1), r(i,j-1)\right], \tag{69}$$

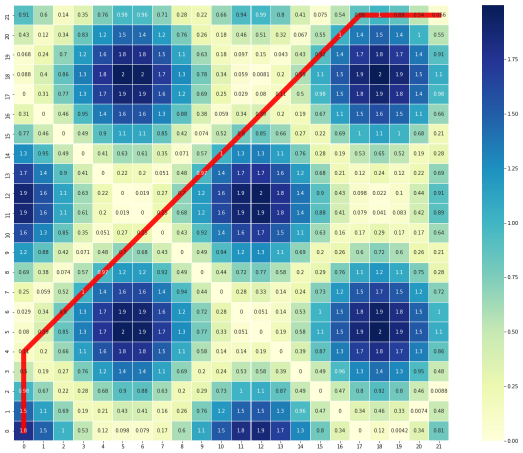
where $r(i,j)$ is the running cumulative distance of point $(i,j)$ [75]. The algorithm fills out a matrix of cumulative distances according to this recurrence relation. The optimal path is then obtained by traversing backwards through the grid moving from point to point according to lowest cumulative distances [75].

The cumulative distance measure of the optimal warping path $W$ provides a much better measure of similarity between sequences than Euclidean distance since it geometrically accounts for similarity by adjusting for temporal shifts. Figure 16 gives a visual representation of DTW in action. Figures 16a and 16b show the differences in alignment between DTW and Euclidean distances. The flexibility offered by dynamic

time warping yields significant benefits when trying to asses the similarity of the two sequences as evidenced by the fact that the dynamic time warped distance between the two curves is 1.35 compared to 5.31 with the Euclidean distance.



**(a) DTW alignment of sequences.** **(b) Euclidean alignment of sequences.**



**(c) DTW warping path.**

**Figure 16. Example showing how DTW alignment overcomes temporal shifts compared to Euclidean distance, along with the wapring path corresponding to the DTW alignment.**

### 2.3.3 Soft-DTW

In their paper *Soft-DTW: a Differentiable Loss Function for Time-Series* authors Marco Cuturi and Mathieu Blondel [23] note that the traditional DTW function is not differentiable, which in many machine or deep learning contexts makes it's use as

a loss function problematic. To remedy this the authors propose a new formulation that combines traditional DTW and the Global Alignment Kernel (GAK)[23][76]. Their formulation of the DTW function is slightly different than that of Equation 68. Let $\mathcal{A}_{n,m} \in 0, 1^{n \times m}$ be the set of possible alignment matrices where the matrix is zero everywhere except along the given warping path, and the pairwise distance (cost) matrix be $\Delta(\boldsymbol{x}, \boldsymbol{y})$. The optimal DTW alignment is then given by,

$$DTW(\boldsymbol{x}, \boldsymbol{y}) \coloneqq \min_{A \in \mathcal{A}_{n,m}} \langle A, \Delta(\boldsymbol{x}, \boldsymbol{y}) \rangle \tag{70}$$

and the GAK is defined as,

$$k_{GA}^{\gamma} \coloneqq \sum_{A \in \mathcal{A}_{n,m}} e^{-\langle A, \Delta(\boldsymbol{x}, \boldsymbol{y}) \rangle / \gamma}. \tag{71}$$

The authors synthesize Equations 70 and 71 into their proposed soft-DTW operator:

$$DTW_{\gamma}(\boldsymbol{x}, \boldsymbol{y}) \coloneqq min^{\gamma}\{\langle A, \Delta(\boldsymbol{x}, \boldsymbol{y}) \rangle, A \in \mathcal{A}_{n,m}\}, \tag{72}$$

where

$$\min^{\gamma}\{a_1, \ldots, a_n\} \coloneqq \begin{cases} \min_{i \leq n} a_i & \gamma = 0, \\ -\gamma \log \sum_{i=1}^{n} e^{-a_i/\gamma}, & \gamma > 0. \end{cases} \tag{73}$$

Their reformulation introduces a soft minimum (Equation 73) which makes Equation 72 explicitly differentiable everywhere when $\gamma > 0$. For the sake of computational efficiency, differentiation is done via backpropagation - application of the chain rule backwards through the recursion from Equation 69.

### 2.3.4 Autoencoders

Autoencoders are a deep learning tool that can be used for many things, one of which is representation learning. Representation learning is the process of learning a feature rich embedding space from a given data set. In general, representation learning does not have to be done with neural networks, as principal components analysis (PCA) itself is a form of representation learning. However, the ability of deep neural networks to learn complex non-linear relationships makes them a popular choice [2][77].

The autoencoder (AE) is comprised of an encoder portion and decoder portion. The encoder, or feature extractor, is a function that maps an input to a latent space, $e_\theta : x \rightarrow h$. Subsequently, the decoder function attempts to map the latent embedding $h$, back to the input space, $d_\nu : h \rightarrow r$. These encoder and decoder weights, arbitrarily termed $\theta$ and $\nu$ respectively, are learned simultaneously by minimizing the reconstruction error, or the difference between $x$, and $r$ [77]. There are no strict requirements on the form of the latent space embedding $h$, however in some cases it has smaller dimensionality than the input to yield a lower dimensionsal but feature rich representation of the input space. This type of AE is referred to as "undercomplete" and the latent space referred to as the bottleneck [2]. Finally, the type of neural network architecture used to learn the encoder and decoder functions is driven by the problem application. In certain domains a simple network of feed-forward layers is sufficient, while in others cases like image or time series data, convolutional or recurrent layers may be more appropriate for the AE function.

### 2.3.5 Deep Representation Clustering (DRC)

With the success of deep learning in so many research areas it is no surprise that is has been applied to the problem of clustering. Researchers have varied slightly

in how they go about this, but most follow a generally consistent approach. Survey papers *Deep learning-based clustering approaches for bioinformatics* by Karim et al. [78] and *Survey of Clustering With Deep Learning: From the Perspective of Network Architectures* from Min et al. [79] explore the existing literature in depth.

Deep clustering is guided by a loss function that can generally be expressed as:

$$\mathcal{L} = \lambda \mathcal{L}_n + (1 - \lambda)\mathcal{L}_c, \tag{74}$$

where $\mathcal{L}_n$, the network reconstruction loss, is the portion of loss associated with representation learning and $\mathcal{L}_c$, the clustering loss, encourages cluster formation in the latent space. Though the underlying mechanics are manipulated by researchers for specific applications, Equation 74 describes the essence of deep clustering approaches. First, consider the case where $\lambda = 1$, which places all of the weight on the network loss. In this case an AE learns a lower dimensional, feature rich representation for the data. Once learned, the decoder is discarded and a standard clustering algorithm is applied to the data in its embedded form. This alone can be beneficial in that the clustering is performed on lower dimensional data that is comprised of it's most important features. In the case where $\lambda = 0$, the loss becomes entirely cluster driven - Min et al. [79] refer to this as clustering deep neural networks (CDNN) [79]. Since they do not attempt representation learning, CDNN's are simple CNN's, LSTM's or FCN's without an autoencoder structure. Instead the input data is passed through network layers and fed to a clustering loss. The fact that they are optimized only to minimize clustering loss without the counteracting reconstruction loss can result in a corrupted feature space where the cluster formation is dense but the feature space is nonsensical [79]. The case where $\lambda \in (0, 1)$ is of primary interest in this research - an in depth analysis of such models will now be explored.

In their survey paper which looks at deep clustering methods from the perspec-

tive of network architectures, Min et al. [79] propose four categories of underlying architectures: autoencoders, standard CNN, RNN or FCN's, generative adversarial networks (GAN's), and variational autoencoders (VAE's). Both GAN's and VAE's are deep generative networks, capable not only of being adapted for tasks such as clustering but also of generating data samples once trained [2][79]. Unfortunately training these type of generative models can be difficult. Specifically, VAE's have high computational complexity and GAN's can face convergence issues in practice [2]. The use of a standard CNN, RNN, or FCN architecture is associated with a solely cluster driven loss (CDNN), the case where $\lambda = 0$. Due to the convergence and computational complexity issues associated with GAN's and VAE's and the lack of need for their generative abilities, and the potential pit falls that come with CDNN's, the focus will now be placed on an autoencoder driven approach.

### 2.3.6 Autoencoder-based Deep Clustering

#### Deep Clustering Loss Function Types

The manner in which $\mathcal{L}_c$ is made to affect the latent space formation depends on they type of loss function used and how it is implemented. Min et al. [79] argue these clustering losses can be broken into two groups: principal and auxiliary losses. The difference between these two could be thought of as loss functions that directly vs indirectly influence the latent space to take on a cluster friendly formation. Principal loss functions, like k-means, make use of actual cluster centroids and assignments. On the other hand, auxiliary loss functions use losses that encourage cluster friendly formations in a more indirect way. One such example is enforcing a locality preserving constraint, which forces points that are near each other in the original space to also be near each other in the embedded space [80].

### 2.3.7 Training Deep Clustering Models

There are a number of ways to go about the training process, with differences largely being driven by the type of clustering loss function being used and how the clustering and representation learning losses are combined to perform the joint optimization. To illustrate some of the different approaches and motivate the need for these differences, we will examine some deep clustering models from the literature.

### 2.3.8 DRC in the Literature

#### Deep Embedded Clustering (DEC)

The Deep Embedded Clustering (DEC) algorithm is trained in two separate phases. First an autoencoder is used to learn an initial latent representation for the data. In the second phase, the decoder portion of the network is abandoned, and the reconstruction loss used by the autoencoder is replaced with a soft-assignment clustering loss. The clustering step is initialized by running k-means on the embedded dataset $f_\theta(\boldsymbol{X})$, learned through phase one, to generate initial cluster centroids. The algorithm then iterates through the following procedure until convergence:

1. Calculate soft cluster assignments ($q_{ij}$) between data points and centroids using Equation 75.

2. Update encoder weights ($f_\theta$) and centroids using the Kullback-Leibler divergence loss (Equation 77) between the soft assignments and an auxiliary distribution ($p_{ij}$) constructed by the authors as Equation 76.

$$q_{ij} = \frac{(1 + ||f_\theta(x_i) - \mu_j||^2/\alpha)^{-\frac{\alpha+1}{2}}}{\sum_{j'}(1 + ||f_\theta(x_i) - \mu_{j'}||^2/\alpha)^{-\frac{\alpha+1}{2}}} \tag{75}$$

$$p_{ij} = \frac{q_{ij}^2 / \sum_i q_{ij}}{\sum_{j'} \frac{q_{ij'}^2}{\sum_i q_{ij}}} \tag{76}$$

$$L = KL(P||Q) = \sum_i \sum_j p_{ij} log \frac{p_{ij}}{q_{ij}} \tag{77}$$

**Improved Deep Embedded Clustering (IDEC)**

IDEC improves upon the DEC algorithm by taking specific action to ensure the embedded space structure remains intact throughout the clustering phase. Specifically, authors Guo, Gao, Liu and Yin [13] noticed in their first model (DEC) that by getting rid of the reconstruction error term during the clustering phase and fine tuning using only the clustering loss, the embedded space can become distorted. To remedy this the authors propose the architecture in Figure 17. The difference in relation to DEC is that the decoder portion of the network remains intact through the entire training procedure. This allows both reconstruction error and clustering error to be jointly optimized throughout the whole training process, thus preserving local structure in the embedding space. The authors maintain use of the same KL Divergence as in DEC for the clustering portion of the loss ($L_c$).

**Deep Clustering Network (DCN)**

Yang et al. [81] present their DCN algorithm in *Towards K-means-friendly Spaces: Simultaneous Deep Learning and Clustering*. These authors were also concerned with possibility of a cluster only loss distorting the embedding space and leading to trivial solutions. With this in mind, they designed a loss function that uses reconstruction error as a bulwark against trivial solutions, and employs k-means as a loss to optimize for a k-means friendly latent space. Their architecture is very similar to the IDEC architecture in Figure 17. The encoder-decoder pair learn the embedded space with help from the clustering loss which is computed at the bottleneck of the autoencoder.

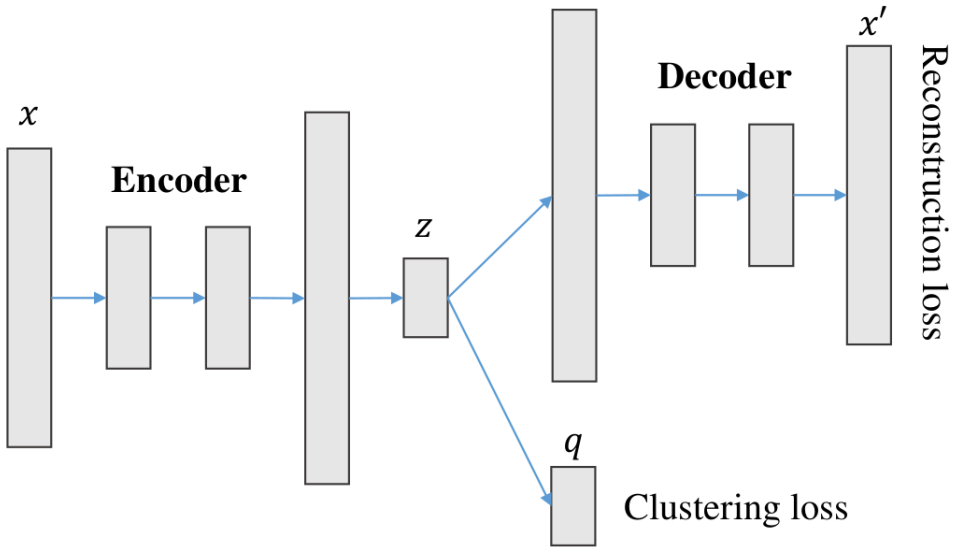



**Figure 17. IDEC architecture improves upon DEC by utilizing full autoencoder throughout entire training process, making the embedding space resilient against distortions from $L_c$ [13]**

Their model must be trained in an alternating fashion to accommodate the discrete nature of the k-means loss. Specifically, the training procedure alternates between 1) fixing the cluster parameters (centroids and cluster assignments) and optimizing the network, and 2) fixing the network parameters and updating the clustering parameters. The network optimization portion is straightforward and can be done using backpropagation on the following loss function:

$$\min_{\mathcal{W},\mathcal{Z}} L^i = l(\boldsymbol{g}(\boldsymbol{f}(\boldsymbol{x_i})), \boldsymbol{x_i}) + \frac{\lambda}{2}||\boldsymbol{f}(\boldsymbol{x}_i) - \boldsymbol{M}\boldsymbol{s}_i||_2^2, \tag{78}$$

where the first portion of the loss function is the reconstruction error and the second part captures the difference between the $i^{th}$ data point and the centroid of the cluster to which the $i^{th}$ data point belongs. Within the cluster parameter update step the cluster indicator vectors ($s_i$'s) are updated according to the following rule:

$$s_{j,i} \leftarrow \begin{cases} 1, & \text{if } j = \underset{k=\{1,\ldots,K\}}{\operatorname{argmin}} ||\boldsymbol{f}(\boldsymbol{x}_i) - \boldsymbol{m}_k||_2, \\ 0, & \text{otherwise.} \end{cases} \tag{79}$$

The centroid update step is done using the arithmetic mean of the samples assigned to a given cluster, with a slight adjustment that allows the algorithm to be implemented in an online fashion for added flexibility [82]. Specifically, for the $k^{th}$ centroid $m_k$, and $c_k^i$ which represents the number of samples assigned to the $k^{th}$ cluster at the processing time of the $i^{th}$ sample, the centroid update step is:

$$\boldsymbol{m}_k \leftarrow \boldsymbol{m}_k - (1/c_k^i)(\boldsymbol{m}_k - \boldsymbol{f}(\boldsymbol{x}_i))s_{(k,i)}. \tag{80}$$

**Deep Temporal Clustering Representation (DTCR)**

Ma et al. [83] present their DTCR algorithm in *Learning Representations for Time Series Clustering*. DTCR follows the general framework of the other algorithms with an autoencoder, which yields $\mathcal{L}_r$, and a clustering layer off of the autoencoder bottleneck, yielding $\mathcal{L}_c$. However, there are two particularly unique components to DTCR. First, an additional layer is added off of the autoencoder bottleneck that performs the task of identifying real vs fake time series. This additional classification task was aimed at enhancing the ability of the encoder and thus producing a better latent space. To accomplish this, each time series in the dataset was given a corresponding "fake" time series by randomly shuffling 20% of the time steps. Second, the clustering loss function is a reformulation of K-means as a trace maximization problem involving $H$, which is the learned matrix of data points in embedded space, and $F \in \mathbb{R}^{N \times k}$, which is a cluster indicator matrix where $N$ is the number of data points and $k$ is the

number of clusters. Specifically, the reformulated K-means clustering loss is:

$$\mathcal{L}_{K-means} = Tr(H^T H) - Tr(F^T H^T H F) \tag{81}$$

Additionally, the cluster indicator matrix $F$ has a closed form solution via the Ky Fan theorem [84]:

$$F_{update} = eigs(H^T H) \leftarrow \text{ take first } k \text{ eigenvectors} \tag{82}$$

Optimizing DTCR is done in an iterative fashion since $F$ needs to be updated – every $T$ epochs – as the embedding space is learned. The total loss function is:

$$\mathcal{L} = \mathcal{L}_{reconstruction} + \mathcal{L}_{classification} + \lambda \mathcal{L}_{K-means}, \tag{83}$$

where $\lambda$ is the weight placed on the clustering loss. The proposed range for weights is $[1, 0.1, 0.01, 0.001]$. However through inspection of the authors published GitHub code the clustering weight used for their example dataset was 0.1.

# III. Solution Methodology

There are two overarching tasks in this research, which together yield three separate objectives. First, the construction of two DTW-based DRC algorithms. Second, spatially clustering geographical locations with respect to a weather attribute measured over time. Third, a physics informed deep neural network for time series prediction of $u$ and $v$ wind components, temperature, and, pressure - the variables that are used to describe 2-dimensional, dry atmosphere fluid dynamics according to the Navier-Stokes equations.

## 3.1 Soft-DTW K-Means based DRC (SDTW-KM-DRC)

This section outlines a K-means based soft-DTW deep representation clustering (SDTW-KM-DRC) algorithm for integrating a differentiable version of DTW into a DRC framework.

### 3.1.1 Basic Model Structure

As described in Chapter II, DRC algorithms are typically built with an autoencoder at their core. The choice of such an autoencoder should be dictated by the nature of the data being encoded. In this case, the data is time series so an architecture capable of processing temporal data is required. Until recently, an RNN variant (LSTM, GRU, bi-LSTM, etc.) was the only contender but TCN's are particularly compelling in this case because of their ability to quickly, in relation to recurrent architectures, process arbitrarily long sequences. Though the proposed algorithm can be used for any time series clustering problem, the particular application of long weather time series necessitates that whatever architecture is used be able to efficiently handle and embed long time series.

Typically in prediction contexts, a causal TCN is used, meaning that only information from $t-1$ and before is convolved with information at $t$, i.e., no information leakage. However, since this is not a prediction task and it is desired that the latent representation be as expressive as possible, acausal TCN layers, as shown in Figure 18, are used.

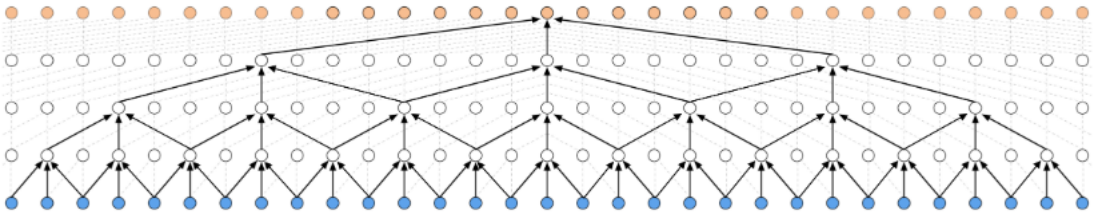


**Figure 18. Acausal TCN with dilations = [1,2,4,8], stride = 1, and kernel size = 3 [14].**

To reduce the size of the latent space SDTW-KM-DRC downsamples the data after each TCN layer, and conversely upsamples after each corresponding TCN layer in the decoder.

### 3.1.2 SDTW-KM-DRC Clustering Layer

The crux of any DRC algorithm is the clustering layer, as this is what drives formation of a cluster-friendly latent space. This specific application of SDTW-DRC uses a K-means clustering layer. Adapted to use soft-DTW instead of Euclidean distance, the following cluster loss function is used:

$$\mathcal{L} = \sum_{i}^{N} \sum_{j}^{K} \delta(x_i \in C_j) \text{sdtw}(\mu, x_i), \tag{84}$$

where $\delta$ is an indicator function that is 1 if data point $x_i$ is in cluster j ($C_j$) and 0 otherwise, $\mu_j$ is the centroid of cluster j, and N and C are the number of data points and clusters respectively. So, the clustering layer is rather simple - it computes the loss between a set of data points and their nearest cluster centroids with respect to the soft-DTW discrepancy. Since the soft-DTW loss is differentiable this error

can be backpropagated through the encoder such that future embeddings will be more cluster friendly. Specifically, there will be some amount of pressure on the network, counterbalanced by the pressure to minimize reconstruction error, for the shape of each data point to become incrementally more similar to their respective cluster centroids. This was alluded to previously, but the key to ensuring the latent space does not get distorted by the clustering loss is placing sufficient weight on the reconstruction loss.

### 3.1.3 Training Pipeline

The training phase is broken up into two main parts. The first phase consists of pretraining the autoencoder so that the introduction of the cluster layer is associated with a properly trained latent space. The next phase is characterized by network training with both the clustering and reconstruction loss functions punctuated by periodic updates of cluster centroid and membership information.

The training pipeline is described rigorously in Algorithms 2 & 3. Algorithm 2 describes the full training process while Algorithm 3 describes the process for obtaining updated cluster centroids and membership with respect to soft-DTW. Steps 7 and 8 in Algorithm 2 are particularly noteworthy as they rely on the differentiability of soft-DTW. Specifically, since the soft-DTW loss function is differentiable, there is a gradient attached to $\mathcal{L}_c$ from Step 7. This allows backpropagation of the neural network weight updates in Step 8 to occur.

**Algorithm 2** SDTW-KM-DRC

**Inputs:**
$X \in \mathbb{R}^{N \times S}$
**Definitions:**
$N :=$ number of observations
$S :=$ length of original time series
$T :=$ length of embedded time series
$M \in \mathbb{R}^{N \times T} :=$ matrix of $\mu_j$'s associated with each $x \in X$
$c \in \mathbb{Z}_{\geq 0}^{N \times 1} :=$ cluster membership for each $x \in X$
$f_{AE} :=$ full autoencoder, $f_{enc} :=$ encoder, $f_{dec} :=$ decoder
$\text{sdtw}(x_1, x_2) :=$ soft-DTW discrepancy between $x_1$ and $x_2$
$\text{mse}(x_1, x_2) :=$ mean squared error between $x_1$ and $x_2$
**Parameters:**
$\theta =$ parameters/weights for $f_{AE}$
**Hyperparameters:**
E = total number of epochs (after pretraining)
I = cluster information update interval
$\alpha =$ learning rate
**Outputs:**
$M_{final}$, $c_{final}$

1: **Pretrain:**
$f_{AE} : X \to X$
2: **for** epoch = 0 to E **do**
3: **if** epoch % I == 0 **then**
4: $\boldsymbol{M}$, $\boldsymbol{c} \leftarrow$ **Algorithm 3**($f_{enc}$(X))
5: **end if**
6: $\mathcal{L}_r \leftarrow \text{mse}(f_{AE}(X), X)$
7: $\mathcal{L}_c \leftarrow \text{sdtw}(f_{enc}(X), M)$
8: $\theta \leftarrow \theta - \alpha \frac{\partial(\mathcal{L}_r + \mathcal{L}_c)}{\partial \theta}$
9: **end for**
10: $M_{final}$, $c_{final} \leftarrow$ **Algorithm 3**($f_{enc}$(X))
**return** $M_{final}$, $c_{final}$

Algorithm 3 is an altered version of Lloyd's algorithm that uses soft-DTW as the driver of similarity and centroid calculations, as opposed to Euclidean distance and arithmetic mean. To speed up convergence and improve the baseline solution quality a soft-DTW version of K-means++, which is a popular centroid initialization strategy, is used [85] in Step 1. Subsequently, steps 3-5 are a fairly direct substitution of soft-DTW for Euclidean distance in the assignment of data points to clusters $S_1, \ldots, S_k$. The primary variation from the standard Euclidean based Lloyd's algorithm is the

centroid calculation in Steps 6-9. Specifically, instead of centroids being the arithmetic mean of the data points in a cluster, the centroid must be calculated via optimization. Given an initial set of weights, in this case the arithmetic mean of the data but it could be a medoid or random initialization, the centroid is taken to be the set of weights that yields the minimum soft-DTW distance between all members of the cluster. Again, since soft-DTW is differentiable, gradient based methods can be employed in the minimization. Specifically, the quasi-Newton Limited memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS) algorithm from SciPy was chosen [86].

The main result of Algorithm 3 is matrix $M \in \mathbb{R}^{N \times T}$, where $N$ is the number of time series and $T$ is the length of the time series in embedded space. This matrix contains the centroids associated with each data point. Algorithm 3 also returns the vector $c \in \mathbb{Z}_{\geq 0}^{N \times 1}$ which is a cluster indicator vector for each time series, though this is not directly required as input to Algorithm 2 as is $M$. As is alluded to in Step 12, careful bookkeeping across iterations is required for constructing final $W$ and $c$ matrices. Finally, the architecture can be visualized in Figure 19.

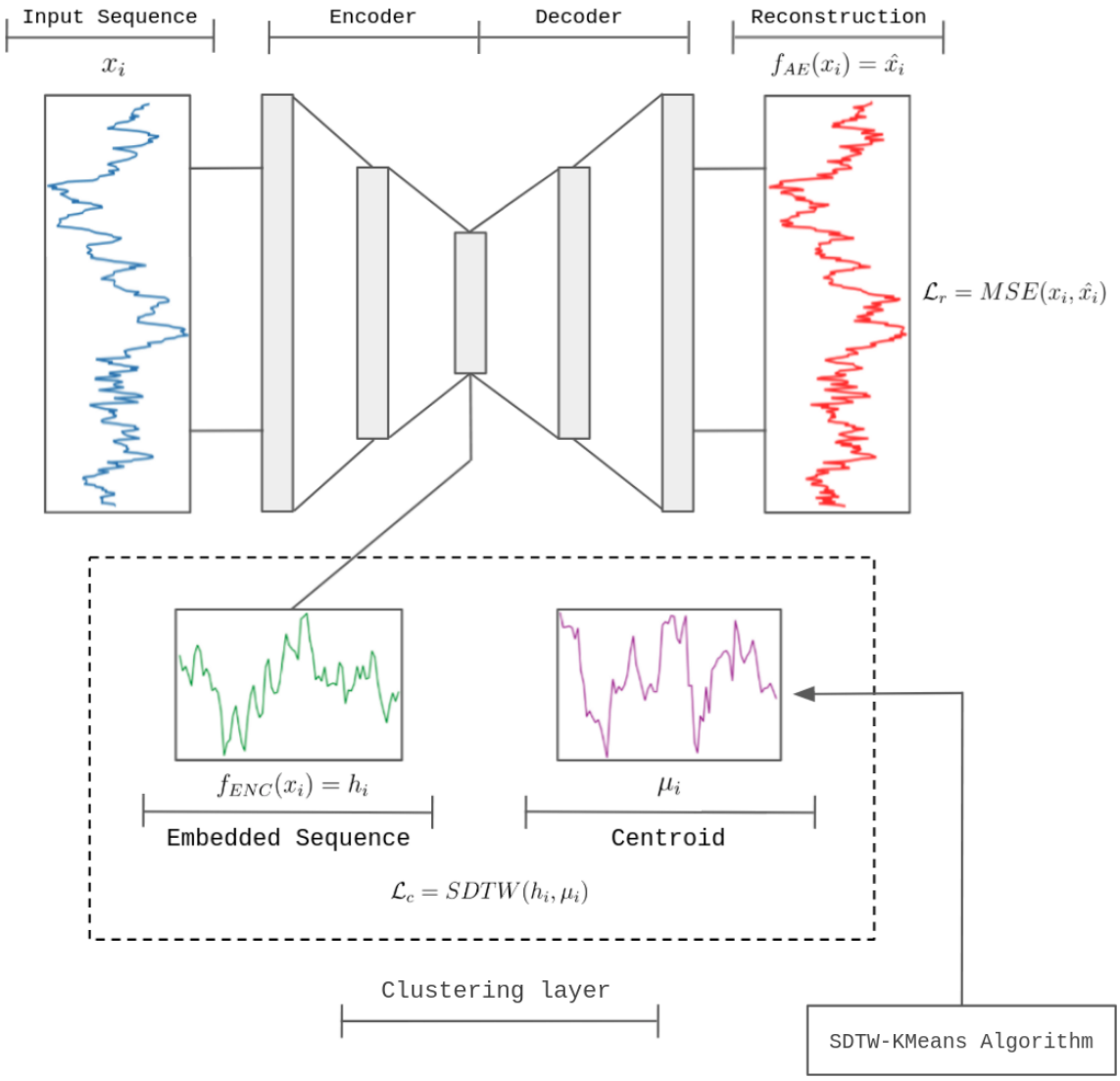



**Figure 19. Basic SDTW-KM-DRC architecture with visualization of processing one example time series through the network.**

**Algorithm 3** SDTW-KMeans

**Inputs:**
$H \in \mathbb{R}^{N \times T} = f_{enc}(X)$
**Definitions:**
**inherit all definitions from Algorithm 2**
k-means++ := "smartly" initializes centroids
$W \in \mathbb{R}^{k \times T}$ := matrix of centroids
$S_1, \ldots, S_k$ := sets that contain the points assigned to each of the $k$ clusters (can change with each iteration)
**Hyperparameters:**
MaxIter = maximum number of iterations
k = number of clusters
**Outputs:**
$M$, $c$
**Initialize:**
$S_1 = \{\emptyset\}, \ldots, S_k = \{\emptyset\}$
$W$ = k-means++$(H)$
**for** iterations = 1 : MaxIter **do**
  **for** $i = 1 : N$ **do**
    $S_j \cup X_{i,:}$, where $j = \underset{j=\{1,\ldots,k\}}{\operatorname{argmin}} \operatorname{sdtw}(X_{i,:}, W_{j,:})$
  **end for**
  **for** i = 1 : k **do**
    $\mu_i = \text{ArithmeticMean}(S_i)$
    $W_{i,:} = \underset{\mu_i}{\text{minimize}} \sum_{s \in S_i} \operatorname{sdtw}(s, \mu_i)$
  **end for**
  $S_1 \leftarrow \{\emptyset\}, \ldots, S_k \leftarrow \{\emptyset\}$
**end for**
$M, c \leftarrow$ Construct $M$ and $c$ using $W$ and $S_1, \ldots, S_k$ throughout iterations
**return** M, c

### 3.2 Soft-DTW KL Divergence based DRC (SDTW-KLD-DRC)

The autoencoder portion of the SDTW-KLD-DRC model is the same as that of the SDTW-KM-DRC model. Specifically, it is again built around a TCN.

### 3.2.1 SDTW-KLD-DRC Clustering Layer

The clustering layer of SDTW-KLD-DRC is actually a layer in the traditional neural network sense in that the cluster centroids that comprise the layer are treated as network weights and are thus learned. At each network iteration the clustering layer weights, $C \in \mathbb{R}^{k \times T}$, are fed to the Kullback-Leibler divergence loss function (Equation 87):

$$q_{ij} = \frac{(1 + sdtw(f_\theta(x_i), \mu_j))}{\sum_{j'}(1 + sdtw(f_\theta(x_i), \mu_{j'}))} \tag{85}$$

$$p_{ij} = \frac{q_{ij}^2 / \sum_i q_{ij}}{\sum_{j'} \frac{q_{ij'}^2}{\sum_i q_{ij}}} \tag{86}$$

$$L = KL(P||Q) = \sum_i \sum_j p_{ij} log \frac{p_{ij}}{q_{ij}}, \tag{87}$$

There are a couple of things to note in the re-introduction of these equations from their original presentation in Chapter II [13]. Most importantly, the soft-DTW discrepancy as the measure of distance instead of the Euclidean distance as was found in the original formulation in Equation 75. Additionally, the terms involving $\alpha$, the degrees of freedom for the Student's t distribution, from the original equation were eliminated by setting $\alpha = 1$ per the recommendation of [87]. Finally, the DEC and IDEC author's target distribution of $p_{ij}$ is used as originally proposed.

### 3.2.2 Training Pipeline

Similar to SDTW-KM-DRC, SDTW-KLD-DRC begins by pretraining the autoencoder so that the cluster specific portion of training is initiated with a properly encoded latent space. However, unlike in SDTW-KM-DRC, instead of fully recalculating both cluster assignments and the cluster centroids every $I$ iterations, the only update required is that of the target distribution $p_{ij}$ which is done using Equation

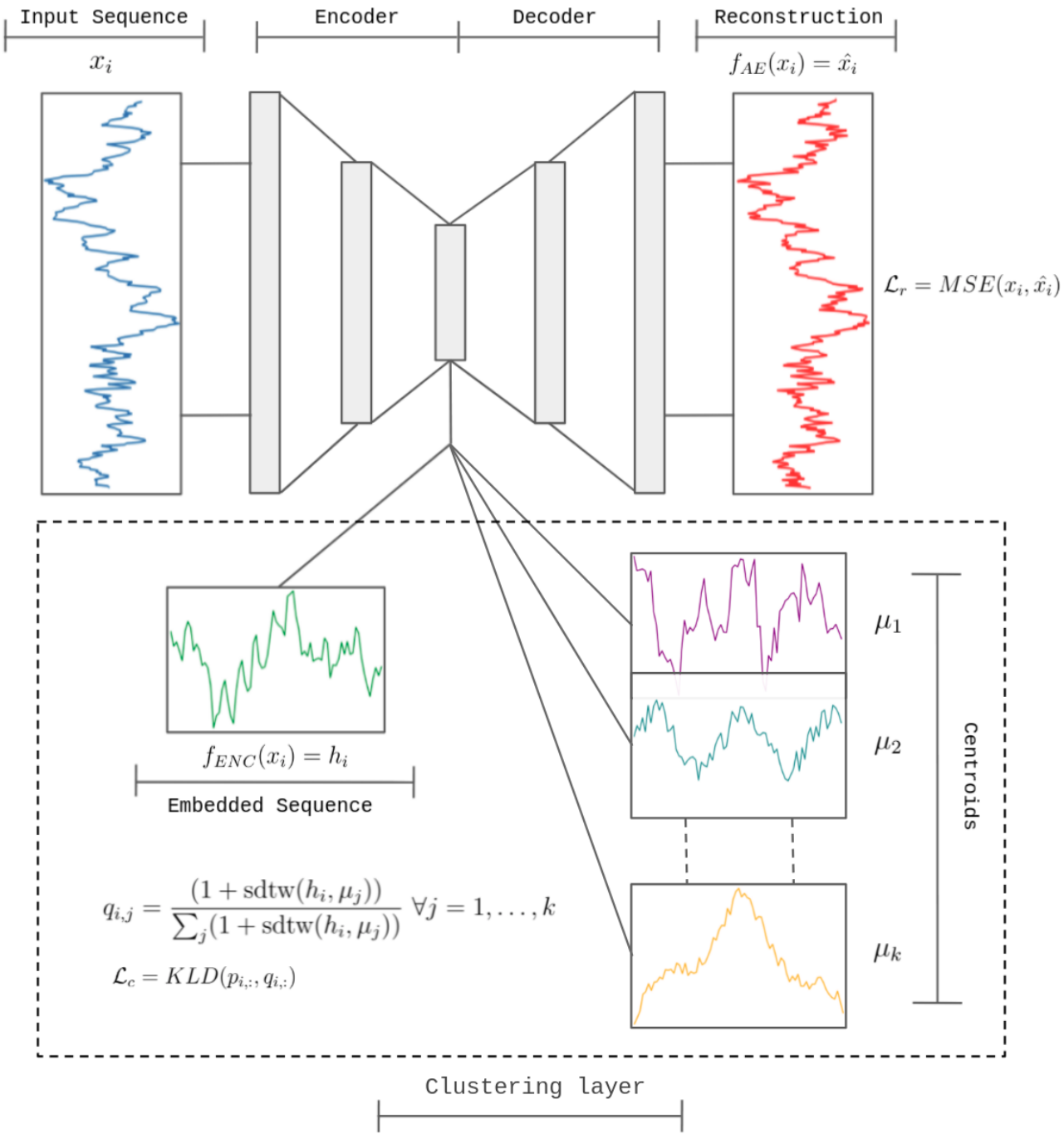



**Figure 20. Basic SDTW-KLD-DRC architecture with clustering layer. Note there are no additional external algorithms needed, rather the cluster information update is solely reflected in the periodic target distribution update.**

86. This update is fast and as a result provides a computational advantage over SDTW-KM-DRC. The full training process is outlined in Algorithm 4. Finally, it is important for the SDTW-KLD-DRC to have a well initialized set of centroid weights. This is accomplished by running Algorithm 3 on the latent space after pretraining the autoencoder and then using the resulting centroids as the starting cluster layer centroid weights as dictated by Step 2.

**Algorithm 4** SDTW-KLD-DRC

**Inputs:**

$X \in \mathbb{R}^{N \times S}$

**Definitions:**

$N :=$ number of observations

$S :=$ length of original time series

$T :=$ length of embedded time series

$C \in \mathbb{R}^{k \times T} :=$ matrix of $\mu_j$'s

$a \in \mathbb{R}^{N \times 1} :=$ cluster assignment $\forall x \in X$

$f_{AE} :=$ full autoencoder, $f_{enc} :=$ encoder, $f_{dec} :=$ decoder

$\text{kld}(P, Q) :=$ KL divergence between distributions $P$ and $Q$

$\text{mse}(x_1, x_2) :=$ mean squared error between $x_1$ and $x_2$

**Parameters:**

$\theta =$ parameters/weights for $f_{AE}$

**Hyperparameters:**

E = total number of epochs (after pretraining)

I = cluster information update interval

$\alpha =$ learning rate

**Outputs:**

$M_{final}$, $c_{final}$

**Pretrain:**

$f_{AE} : X \rightarrow X$

**Centroid Initialization:**

Use Algorithm 3 to initialize centroid weights $C$

**for** epoch = 0 to E **do**

  **if** epoch % I == 0 **then**

    Get soft assignments $Q$ based on current network state

    Update distribution $P$ according to Equation 86

  **end if**

  $\mathcal{L}_r \leftarrow \text{mse}(f_{AE}(X), X)$

  $\mathcal{L}_c \leftarrow \text{kld(P,Q)}$

  $\theta \leftarrow \theta - \alpha \frac{\partial(\mathcal{L}_r + \mathcal{L}_c)}{\partial \theta}$

**end for**

Compute final Q matrix of soft cluster assignment probabilities

$a \leftarrow \text{argmax}_j(Q) \; \forall \; i = 1, \ldots, N$

$C \leftarrow$ final network weights in clustering layer

**return** a,C

## 3.3 Clustering Weather Data

This section describes the application of SDTW-KM-DRC and SDTW-KLD-DRC to weather data for the purposes of breaking up a geographical region into clusters based on similarity in the shape and dynamics of a time series over a long time frame.

### 3.3.1 Dynamics Aware Embeddings

Though the two SDTW-DRC algorithms already cluster time series based on shape, which should in part be reflective of dynamics, an additional component was added to the architecture to directly facilitate dynamics aware clustering. This is illustrated in Figure 21. To avoid a cluttered visual representation the architecture omits the details of any clustering layer, though it is clear from Figures 19 and 20 where they fit in for the respective architectures. For $N$ dynamics measures there are $N$ separate branches that lead out of the latent space and make predictions for the (scalar) dynamics measures. The MSE loss functions associated with each of the $N$ dynamics branches encourages the latent space embeddings to encode, at least in part, the dynamics of the underlying time series as related to the specific dynamics measures included.

### 3.3.2 Data

Since clustering is being done to yield groupings irrespective of seasons or time of year, it is important to make sure that enough data is used. To this end, 8,000 consecutive hours of High Resolution Rapid Refresh (HRRR) model temperature data were used from November 2018 through October 2019. HRRR model data has a spatial resolution of 3km, and covers the entire Contiguous United States, some of Canada and Mexico, and some of the surrounding bodies of water (Pacific, Atlantic, and Gulf of Mexico). This is a very high spatial resolution and results in

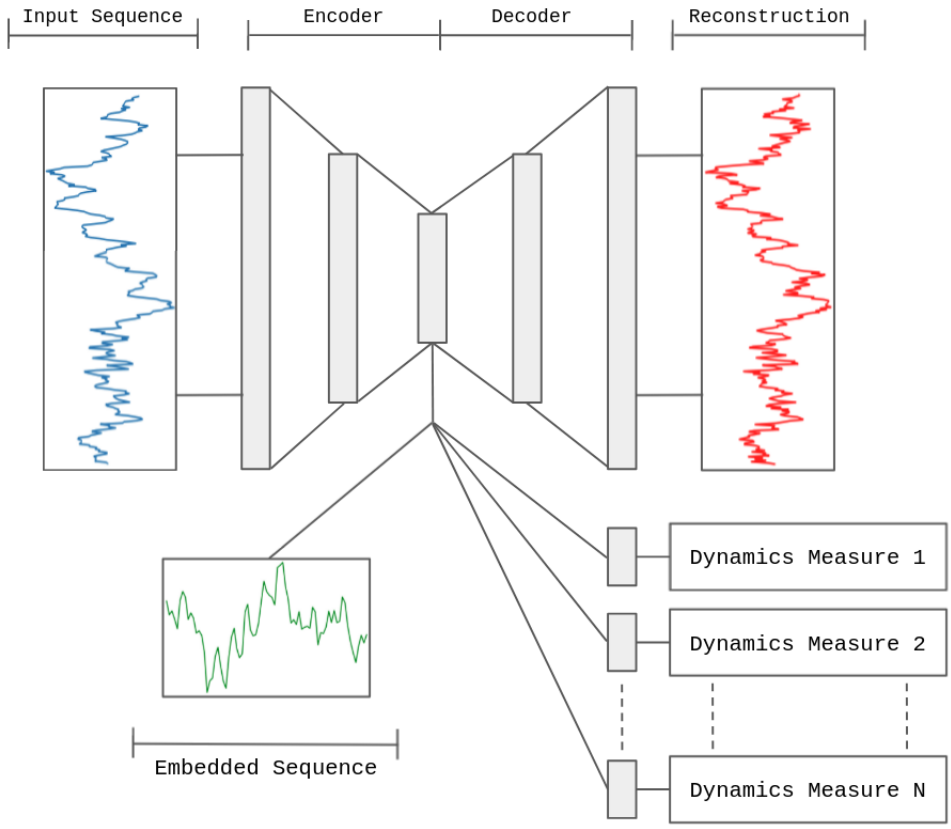



**Figure 21. SDTW-DRC architecture with additional dynamics prediction layers to facilitate learning of a dynamics aware latent space.**

1,905,141 locations. The dynamics measures chosen were detrended fluctuation analysis (DFA) and sample entropy (SampEn). Correlation dimension (CorrDim) was also considered, however the computational burden of this calculation for 1,905,141 observations was prohibitive. Though clustering is an unsupervised training method and there is no direct need for holding out a test set, a small amount of data was held out in case it was needed for future analysis. Specifically, the data was randomly shuffled into $train\_data \in \mathbb{R}^{1{,}680{,}140 \times 8{,}000}$, $validation\_data \in \mathbb{R}^{75{,}000 \times 8{,}000}$, and $test\_data \in \mathbb{R}^{150{,}000 \times 8{,}000}$. One data point was held out as it was found to be corrupted, so the final total count of data points is 1,905,140.

#### Pre-processing

Time series data pre-processing was minimal. In this case min-max normalization was chosen since it does not rely on the mean or standard deviation of the data which may change throughout the time series. This transformed all time series values to be in the interval $[0, 1]$. Calculating the SampEn was done using parallel cpu processing to improve the computation time. The actual SampEn calculation was done using a nonlinear time series analysis tool called "nolds" [88]. Recalling the SampEn calculation as described in Chapter II, there are a couple of parameters to be specified. The embedding dimension, or alternatively, the number of state variables in the system was chosen to be 10. This is most likely an overestimation in the case of the atmosphere, however overestimating the embedding dimension is relatively harmless compared to underestimating since underestimating fails to unfold the dynamics sufficiently. The other hyperparameter to be specified is the tolerance, or threshold, for which the default value from "nolds" was used. For DFA calculations, a python package called "fathon" was used [89]. The "fathon" package was chosen because it was written primarily in Cython and C and therefore is extremely fast.

### 3.3.3 Autoencoder Model Specifics (Pretraining)

The SDTW-DRC model proposed earlier describes a recommended baseline architecture and algorithmic training approach. However the overall description is purposefully generic and lacks application specific details that will now be discussed for this case of weather clustering.

#### Architecture

The main autoencoder architecture details are given in Table 3. The encoder and decoder are comprised of four blocks each. Each block of the encoder has a TCN layer

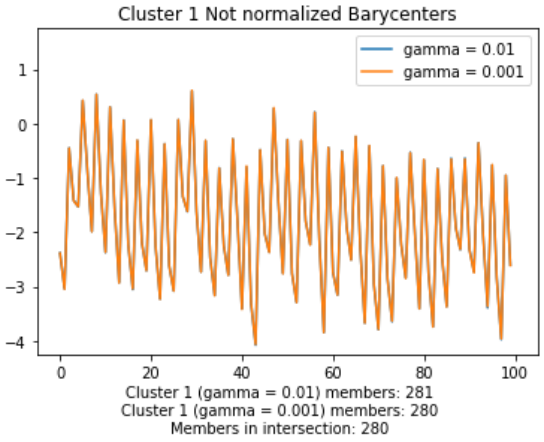
for time series processing, either a MaxPooling layer or Conv1D layer with $stride = 2$ that cuts the sequence in half, and both a batch normalization layer and activation function which are not displayed in the table. The decoder is identical but with upsampling layers instead of downsampling layers. Additionally, to recap what was discussed in Chapter II about TCN's, the dilations mentioned in Table 3 refer to the stacking of TCNs in each layer. For instance, the TCN layer in Block 1 has dilations $= [1, 2, 4, 8, 16, 32, 64]$, which means there are 7 TCNs in the layer with the kernel in the first one having dilation = 1 (i.e., normal convolution), the kernel of the second having dilation = 2, etc. Finally, A Pytorch module for the TCN layers released by Bai et al. [5], the authors cited in Chapter II, was used.

| Autoencoder Architecture Description | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | **layer** | **data shape (in):** | **data shape (out):** | **dilations** | **kernel size** | **stride** |
| **Encoder** | **Block 1** | TCN | ( –, 1, 8000) | (–, 16, 8000) | [1,2,4,8,16,32,64] | 5 | 1 |
| | | Conv1D | (–, 16, 8000) | (–, 16, 4000) | - | 5 | 2 |
| | **Block 2** | TCN | (– 16, 4000) | (–, 32, 4000) | [1,2,4,8,16,32] | 5 | 1 |
| | | MaxPool1D | (–, 32, 4000) | (–, 32, 2000) | - | 2 | 2 |
| | **Block 3** | TCN | (–, 32, 2000) | (–, 64, 2000) | [1,2,4,8,16] | 5 | 1 |
| | | MaxPool1D | (–, 64, 2000) | (–, 64, 1000) | - | 2 | 2 |
| | **Block 4** | TCN | (–, 64, 1000) | (–, 128, 1000) | [1,2,4,8] | 5 | 1 |
| | | MaxPool1D | (–, 128, 1000) | (–, 128, 500) | - | 2 | 2 |
| | | Conv1D | (–, 128, 500) | (–, 1, 500) | - | 3 | 1 |
| **Decoder** | **Block 1** | UpSample | (–, 1, 500) | (–, 1, 1000) | - | - | - |
| | | TCN | (–, 1, 1000) | (–, 128, 1000) | [1,2,4,8] | 5 | 1 |
| | **Block 2** | UpSample | (–, 128, 1000) | (–, 128, 2000) | - | - | - |
| | | TCN | (–, 128, 2000) | (–, 64, 2000) | [1,2,4,8,16] | 5 | 1 |
| | **Block 3** | UpSample | (–, 64, 2000) | (–, 64, 4000) | - | - | - |
| | | TCN | (–, 64, 4000) | (–, 32, 4000) | [1,2,4,8,16,32] | 5 | 1 |
| | **Block 4** | UpSample | (–, 32, 4000) | (–, 32, 8000) | - | - | - |
| | | TCN | (–, 32, 8000) | (–, 16, 8000) | [1,2,4,8,16,32,64] | 5 | 1 |
| | | Conv1D | (–, 16, 8000) | (–, 1, 8000) | - | 5 | 1 |
| **SampEn & DFA** | **Block 1** | Conv1D | (–, 1, 500) | (–, 4, 250) | - | 5 | 2 |
| | **Block 2** | Conv1D | (–, 4, 250) | (–, 8, 125) | - | 5 | 2 |
| | **Block 3** | Conv1D | (–, 8, 125) | (–, 16, 63) | - | 5 | 2 |
| | **Block 4** | Conv1D | (–, 16, 63) | (–, 32, 32) | - | 5 | 2 |
| | **Block 5** | Conv1D | (–, 32, 32) | (–, 1, 32) | - | 5 | 1 |
| | **Block 6** | FC | (–, 32) | (–, 10) | - | - | - |
| | **Block 7** | FC | (–, 10) | (–, 1) | - | - | - |

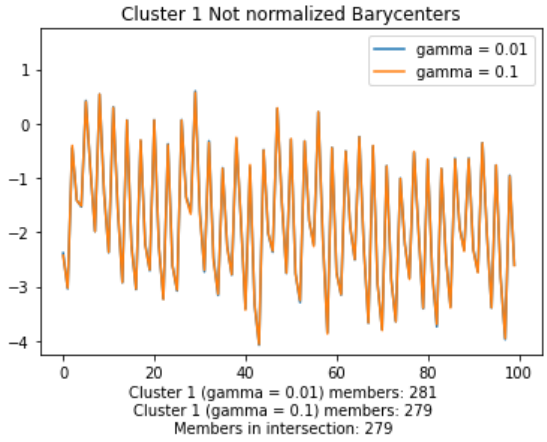
**Table 3. The data shape dimensions are** $(batch\_size, \#filters, sequence\_length)$ **for all block types except those with fully connected (FC) layers where the dimensions are** $(batch\_size, sequence\_length)$**. Padding was left out of the table, but it was structured such that the inputs and outputs of a layer were changed only as a result of stride, upsampling, or downsampling, and never as a result of kernel size. DFA and SampEn each have their own branches off the latent space but are identical.**

### Hyperparameters

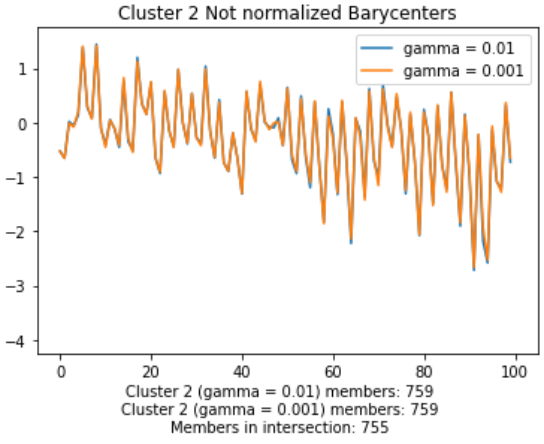
Since batch normalization was used, careful choice of batch size is imperative. Specifically, the batch size must be big enough for accurate estimation of the batch normalization parameters. To this end, the batch size was chosen to be 32. The ADAM optimizer at it's default learning rate of 0.001 was used. Recall from discussion of the soft-DTW function in Chapter II that the parameter $\gamma$ exerts significant control in the amount of smoothing taken into account. The closer that gamma is to zero the more it resembles strict DTW. Specific consideration was made to avoid choosing too high of gamma since an overly smoothed signal could mask some of the shape based dynamics it was hoped that the algorithm would pick up on. So, gamma was chosen to be 0.01. This choice was intuitively made, given the reason just outlined, and confirmed with exploratory data analysis. Specifically, Algorithm 3 was run on a subset of the data for $k = 6$ clusters at various settings of $\gamma$ to determine the sensitivity of the clustering results, both with respect to cluster membership and resulting centroids. Figure 23 shows the analysis done, with the plots showing just a slice out of the middle of the actual embedded time series for better visualization. The analysis was done for $\gamma = 0.1, 0.01,$ and $0.001$, with $\gamma = 0.01$ being the candidate value. Clustering was done for each of the $\gamma$ values and their results compared to ensure that similar centroids and cluster membership were observed at values above ($\gamma = 0.1$) and below ($\gamma = 0.001$) the candidate value. Figure 23 shows that across each clustering, the centroids remain largely the same, though with more smoothing at $\gamma = 0.1$, and more importantly, extremely high overlap in cluster membership across all of the gamma values. This was taken as sufficient justification for choosing $\gamma = 0.01$ as the smoothing hyperparameter.


Cluster 1 Not normalized Barycenters
gamma = 0.01
gamma = 0.001
Cluster 1 (gamma = 0.01) members: 281
Cluster 1 (gamma = 0.001) members: 280
Members in intersection: 280

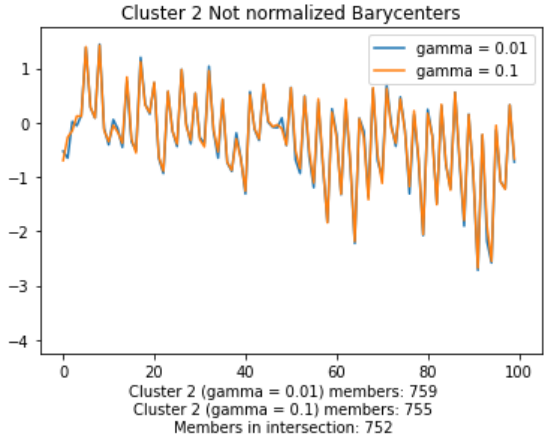
(a)


Cluster 1 Not normalized Barycenters
gamma = 0.01
gamma = 0.1
Cluster 1 (gamma = 0.01) members: 281
Cluster 1 (gamma = 0.1) members: 279
Members in intersection: 279

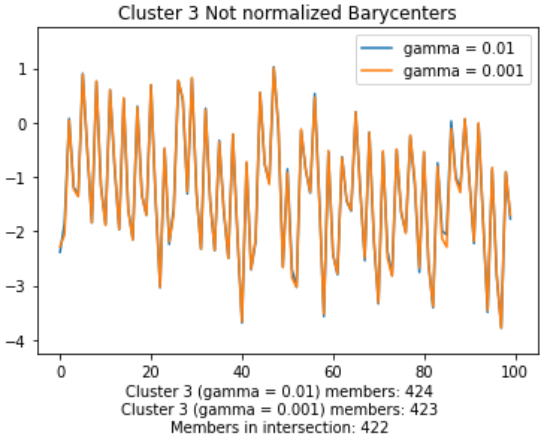
(b)


Cluster 2 Not normalized Barycenters
gamma = 0.01
gamma = 0.001
Cluster 2 (gamma = 0.01) members: 759
Cluster 2 (gamma = 0.001) members: 759
Members in intersection: 755

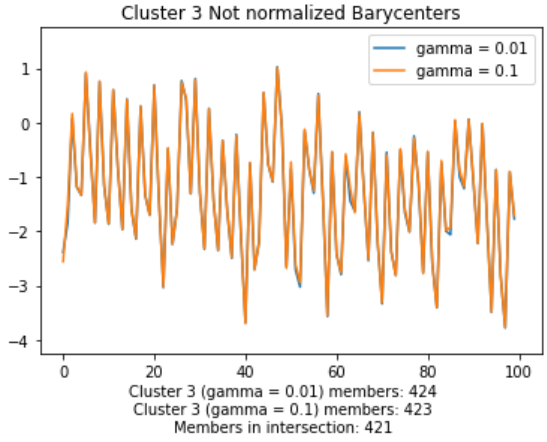
(c)


Cluster 2 Not normalized Barycenters
gamma = 0.01
gamma = 0.1
Cluster 2 (gamma = 0.01) members: 759
Cluster 2 (gamma = 0.1) members: 755
Members in intersection: 752

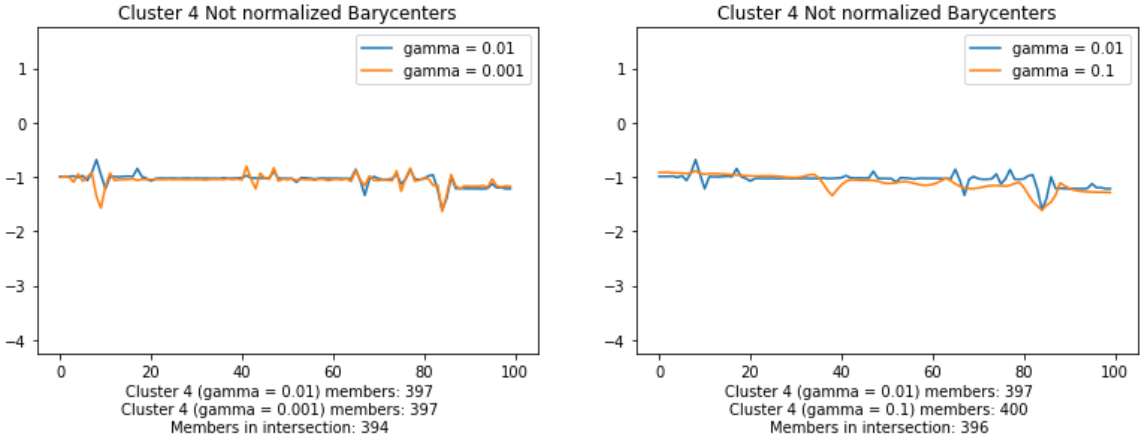
(d)


Cluster 3 Not normalized Barycenters
gamma = 0.01
gamma = 0.001
Cluster 3 (gamma = 0.01) members: 424
Cluster 3 (gamma = 0.001) members: 423
Members in intersection: 422

(e)


Cluster 3 Not normalized Barycenters
gamma = 0.01
gamma = 0.1
Cluster 3 (gamma = 0.01) members: 424
Cluster 3 (gamma = 0.1) members: 423
Members in intersection: 421

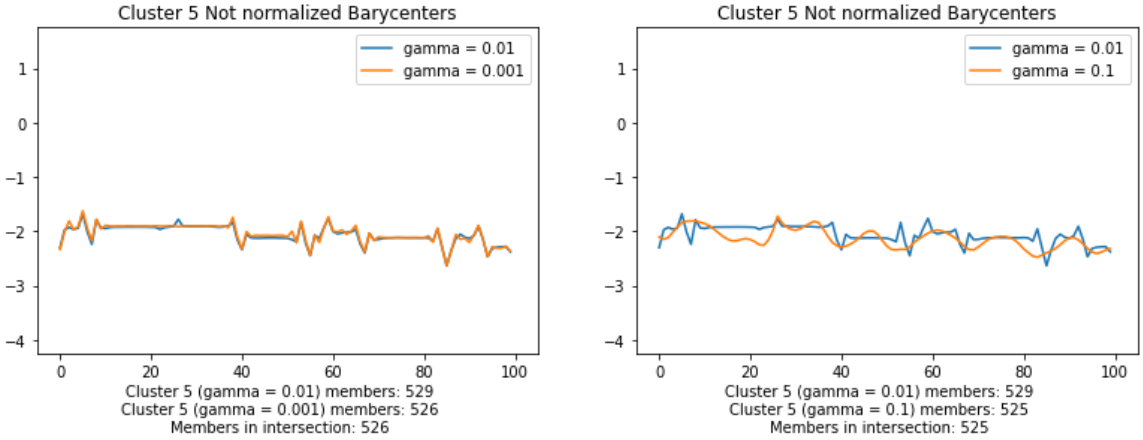
(f)




(g)



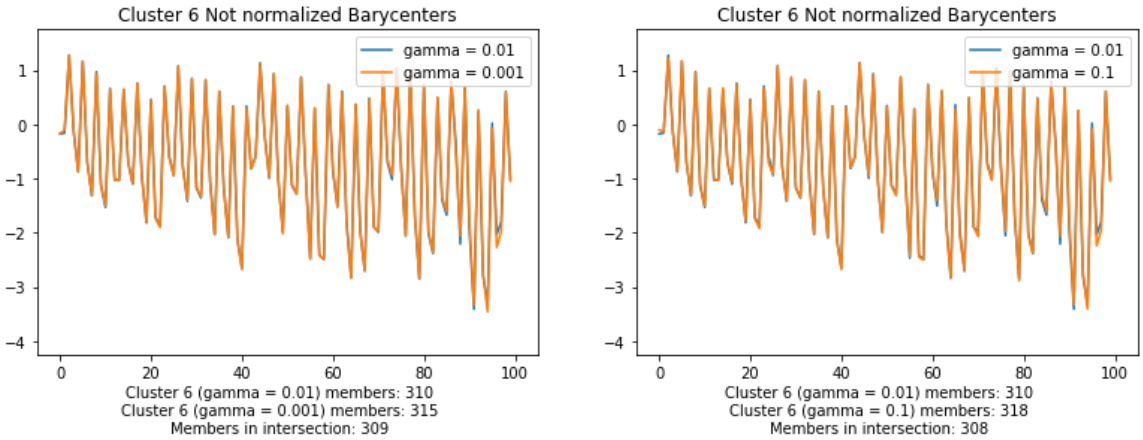

(h)




(i)




(j)




(k)




(l)

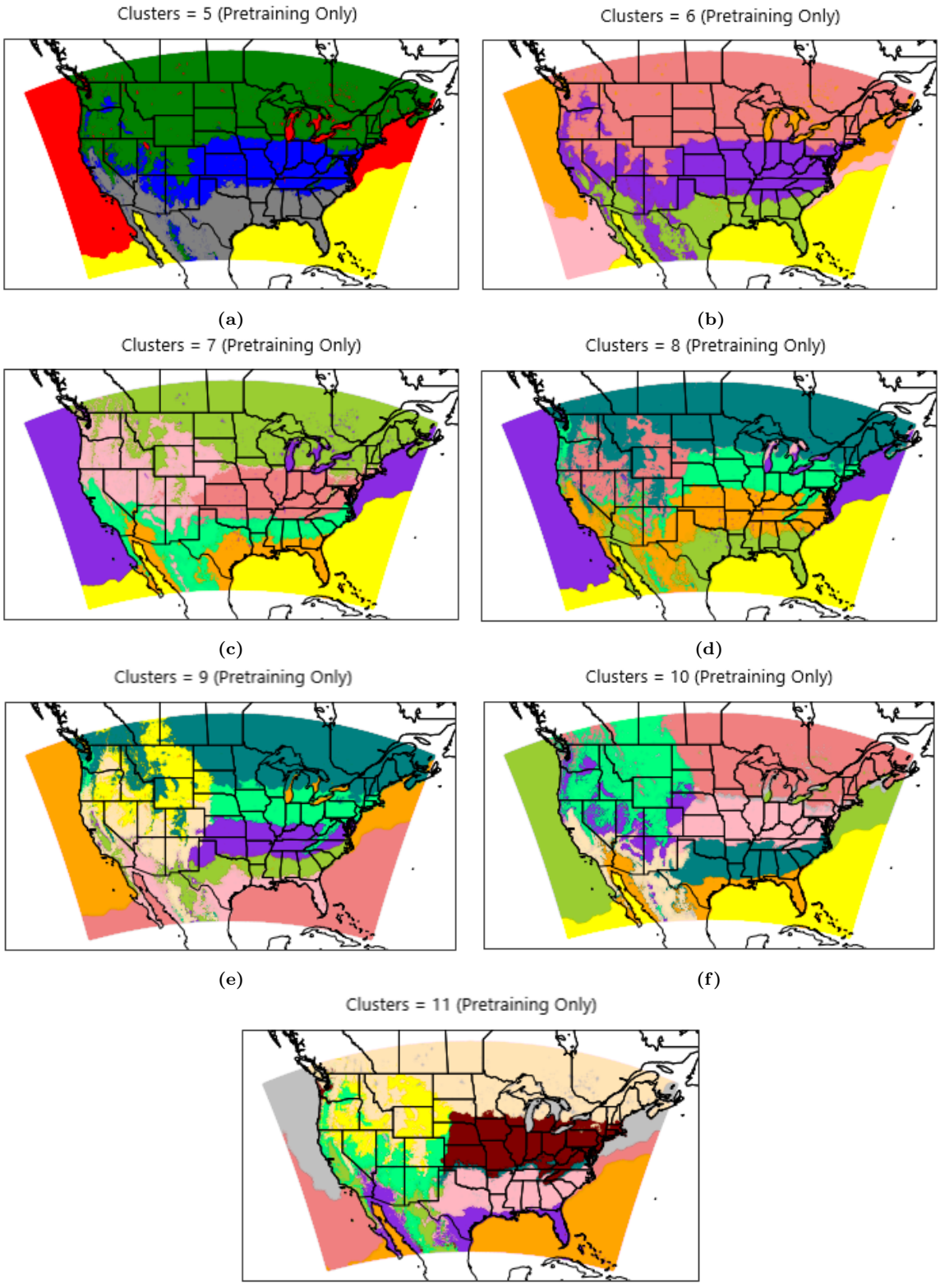
**Figure 23. Sensitivity analysis of clustering results for $\gamma = 0.1, 0.01,$ and $0.001$ to ensure robustness of the proposed value of $\gamma = 0.01$.**

#### Choosing $k$

Since the clustering phases of SDTW-KM-DRC and SDTW-KLD-DRC are so computationally expensive, Algorithm 3 was implemented for $k = 5, 6, 7, 8, 9, 10,$ and 11 on the latent space data after autoencoder pretraining to select a single $k$ for further analysis with the two SDTW-DRC algorithms. These cluster maps are shown in Figure 24. It was clear ahead of time that since locations over the water were included in the analysis, some number of clusters would be assigned to represent those locations. Upon examination of the cluster maps it appears that there are consistently between 2 and 3 water based clusters. Specifically, for $k = 5, 6,$ and 7, when accounting for water based clusters there are only 3, 3 and 5 clusters with which to resolve land based climatic differences. Likewise, looking at Figure 24(g) where $k = 11$, it appears that $k$ is too large and thus arbitrarily split clusters. So, the ideal candidate $k$ value ought to be large enough to facilitate sufficient segmentation of locations without so many as to begin arbitrarily splitting groups further. Upon inspection of the cluster maps, it was determined that $k = 9$ best serves these purposes. Finally, again due to the large amount of data and computational burden associated with repeatedly utilizing the soft-DTW discrepancy in all proposed algorithms, a special CUDA inspired soft-DTW implementation was used in Algorithms 1, 2, and 3 because of it's ability to leverage GPU speed-ups [90][91].

### 3.3.4 Autoencoder Pretraining Protocol

As is described in Algorithm 2, the autoencoder is pre-trained to learn a proper embedding space before fine-tuning with the clustering layer. The loss function used to learn the embedding space during pre-training was a linear combination of recon-




**(a)** **(b)** **(c)** **(d)** **(e)** **(f)** **(g)**

**Figure 24. Cluster maps for $k = 5, 6, 7, 8, 9, 10,$ and $11$ showing the results after pretraining the autoencoder but before any cluster specific training.**

struction error (MSE), along with both DFA and SampEn prediction errors (MSE).

$$\mathcal{L}_{Total} = \mathcal{L}_{reconstruction} + 0.1\mathcal{L}_{DFA} + 0.1\mathcal{L}_{SampEn} \tag{88}$$

Since the primary mode of building a latent embedding space is the reconstruction error, this part of the total loss ($\mathcal{L}_{Total}$) was given the most weight.

The model was pre-trained for 20 epochs. For certain data-sets this would not be enough, but due to the large quantity of training data it was more than sufficient.

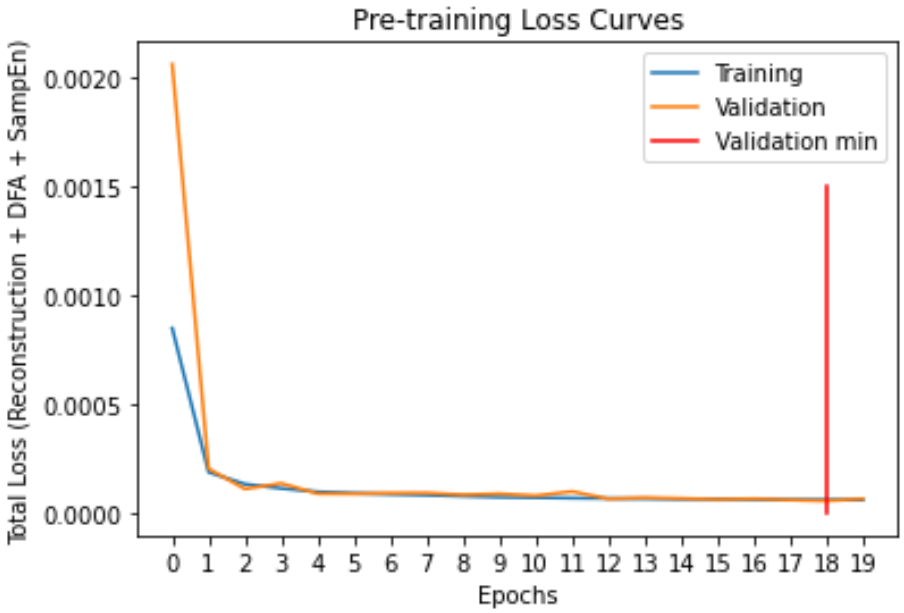



**Figure 25. Training and validation $\mathcal{L}_{Total}$ curves along with vertical line indicating that epoch 18 corresponds with minimum validation $\mathcal{L}_{Total}$.**

Figure 25 shows a couple of things. First, it provides visual confirmation that sufficient pre-training was done since it appears that marginal improvement in $\mathcal{L}_{Total}$ with respect to additional training epochs largely falls off at around epoch 3 or 4. Second, the pre-trained network generalizes well as evidenced by the closely tracking validation and training errors across the entirety of pre-training. It is impossible to attribute generalizability to any specific modeling choice but it is speculated that both the

use of batch-normalization, and the large dataset were helpful. Finally, all model parameters were saved after each of the 20 training epochs so that the best set of weights could be carried forward. Specifically, the best validation $\mathcal{L}_{Total}$ occurs at epoch 18, so the corresponding set of weights were selected for continued network training with addition of the clustering layer.

### 3.3.5 SDTW-KM-DRC Training Protocol

Given the large amount of training data points in this specific application, direct implementation of Algorithm 3 is computationally prohibitive. To cut down on the computation time, a subsampling approach was used. Specifically, instead of applying Algorithm 3 to the entire training dataset every time it is called in step 4 of Algorithm 2, Algorithm 3 is applied to 30 random samples of 1000 data points. For a given $k$, this procedure yields $30 * k$ centroids ($k$ centroids for each of the 30 random samples), which themselves are then fed to Algorithm 3 to yield the final centroids. A separate function then assigns the remaining data points to clusters based on soft-DTW distance to each of the final centroids. Another slight difference from the general methodology provided in Algorithm 2 is the cluster information update step. Again, since the amount of training data is so large it was determined that an "iteration" would constitute a certain number of batches in the training set as opposed to an epoch. One reason for this strategy is that it is unwise to push the network heavily in the direction of one given set of centroids by comparing all $1,680,140$ points to the same centroids. We suspect that without intervening sooner and collecting intra-epoch centroid updates given whatever changes have already been learned in the latent space via training on some intermediary set of points, the latent space would become too specifically learned to a specific set of centroids. Further, since the data is so spatially dense, data points in close proximity to one another will be very similar

to the other points in the immediate geographic area. This means that one could take many random samples of the data spatially and expect to get many "mini" datasets that both resemble one another and the overall dataset. So, given this and that there is plenty of data it was decided that the cluster interval update be $I = 500$ batches, i.e., $16,000$ points given a batch size of 32. For ease of training, the number of training points was dropped from $1,680,140 \rightarrow 1,680,000$ to make it divisible by $16,000$ resulting in 105 cluster updates per epoch. The cluster training phase was run for two epochs and a total of 210 cluster updates. Regarding the rest of the hyperparameters the maximum number of iterations of Algorithm 3, denoted $MaxIter$, was 50. However, in the interest of saving computational time, another clustering termination criteria was added to allow the loop in step 2 of Algorithm 3 to stop early. Specifically, the centroids from iteration to iteration were tracked and if the largest $L_2$ norm of the difference between a given centroid from $iteration_i$ and $iteration_{i+1}$ was less than a predetermined tolerance, the loop ended. In this case the tolerance was set to 0.001. Finally, with the addition of the clustering loss, the new loss function is:

$$\mathcal{L}_{Total} = 0.8\mathcal{L}_{reconstruction} + 0.05\mathcal{L}_{DFA} + 0.05\mathcal{L}_{SampEn} + 0.1\mathcal{L}_{clustering} \tag{89}$$

The reconstruction component of the loss was again given more weight than any other part, emphasizing the necessity of maintaining the integrity of the latent space. Additionally, the ADAM optimizer learning rate was lowered from $0.001 \rightarrow 0.0001$ at the onset of clustering training to encourage a light touch.

### 3.3.6 SDTW-KLD-DRC Training Protocol

The cluster information update strategy and reasoning is the same as for that of SDTW-KM-DRC. The update interval $I$ is 500 batches, and again 140 training points

were dropped to yield a training data size that is divisible by $16,000$. The differences in training protocol from SDTW-KM-DRC arise entirely from the differences in the clustering layers themselves. This was done purposefully in an attempt to standardize as much about the two algorithms as possible aside from their specific clustering layers so that the cause of any difference in outcomes could be better isolated and understood. One unique difference in the SDTW-KLD-DRC training protocol aside from the inherently different clustering layers is the normalization of the soft-DTW discrepancy. Technically since soft-DTW is a soft minimizer and averages over all possible alignments, instead of just the optimal one, the discrepancy can be negative. Looking at Equations 85-87 it is clear that negative values would be unacceptable in the calculations, primarily because it could result in negative probabilities. To overcome this the normalization in Equation 90 was performed [92]:

$$sdtw(x,y)_{normalized} = sdtw(x,y) - \frac{1}{2}(sdtw(x,x) + sdtw(y,y) \tag{90}$$

While this slows the algorithm down it is a necessity when using KL divergence in this context. Finally, as with SDTW-KM-DRC the ADAM optimizer learning rate was lowered from $0.001 \rightarrow 0.0001$ at the onset of cluster training.

### 3.4 Physics Informed Deep Learning

To leverage the use of known physical laws in weather prediction, the rules inherent to Navier Stokes equations are reformulated and added to a deep neural network as a second loss function in addition to the traditional MSE penalty. This section outlines the methodology for implementing a physics based approach to spatio-temporal weather forecasting with deep neural networks.

### 3.4.1 Spatio-Temporal Weather Prediction Problem

#### Overview

Before introducing the physics-based loss function aimed at guiding the network towards learning physically sound predictions, it is important to present the weather prediction problem in isolation. Consider an $N \times N$ grid of locations, where at each location there are $F$ weather attributes measured at an hourly interval for $T$ hours. The proposed model takes an input array of dimension (F,T,N,N), and predicts an array of the same size. This prediction problem can be approached in various ways with various architectures, but ideally the proposed method should explicitly address both the spatial and temporal interactions between variables.

#### Proposed Architecture

Convolutional architectures are designed specifically to process spatial data, and with the recently demonstrated abilities of TCN's to efficiently and effectively process temporal data, a spatially-aware multivariate TCN is a natural architecture to address this problem. Traditionally, TCN's are built around causal 1D convolutions and thus consider only one time series. However, to spatially process $N^2$ multivariate time series at once a 1D implementation is insufficient. Instead, a 3D causal convolution with $F$ channels and causality enforced in the time dimension, is used. The architecture details are outlined in Table 4. Inter-TCN-block details are not outlined explicitly, but dropout layers (20%) were used within each TCN block and batch normalization and ReLU activation functions were used between the first and second TCN blocks. For construction of the network itself the PyTorch TCN layer module created by Bai [5] et al. was used with modification for the purposes of 3D causal convolution.

| Spatially-Aware Multivariate TCN | | | | |
|---|---|---|---|---|
| **block** | **data shape (in):** | **data shape (out):** | **dilations (time axis)** | **kernel sizes (t,n,n)** |
| **TCN 1** | (-, 4, 24, 16, 16) | (-, 64, 24, 16, 16) | [1,2,4,8] | (5,3,3) |
| **TCN 2** | (-, 64, 24, 16, 16) | (-, 64, 24, 16, 16) | [1,2,4,8] | (5,3,3) |
| **TCN 3** | (-, 64, 24, 16, 16) | (-, 4, 24, 16, 16) | [1,2,4,8] | (5,3,3) |

**Table 4. TCN architecture for weather prediction. The kernel sizes are along the time and x-y spatial dimensions respectively.**

### 3.4.2 Physics Loss

Equations 91-95 below were derived and explained in Chapter II. They are a reformulated version of the Navier-Stokes equations involving the following five variables: temperature, pressure (or geopotential height), u-wind, v-wind, and z-wind.

$$\rho g_x - \frac{\partial p}{\partial x} + \mu(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}) = \rho(\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z}) \tag{91}$$

$$\rho g_y - \frac{\partial p}{\partial y} + \mu(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 v}{\partial z^2}) = \rho(\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial v}{\partial z}) \tag{92}$$

$$\rho g_z - \frac{\partial p}{\partial z} + \mu(\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} + \frac{\partial^2 w}{\partial z^2}) = \rho(\frac{\partial w}{\partial t} + u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y} + w\frac{\partial w}{\partial z}) \tag{93}$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0 \tag{94}$$

$$\rho\hat{C}_p(\frac{\partial T}{\partial t} + u\frac{\partial T}{\partial x} + v\frac{\partial T}{\partial y} + w\frac{\partial T}{\partial z}) = k(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2}) \tag{95}$$

For use as constraints in the network, a couple of preliminary adjustments and assumptions were made. First, second order terms would be excluded from the analysis since they tend to be extremely small and in this case, given the amount of error

associated with just the model data alone, would fail to be meaningful. Second, only the $x - y$ plane is included. This was done to eliminate unnecessary complexity in exploring the concept of physics based deep weather prediction since the proposed methodology can adequately be explored in 2D and easily extended to 3D in the future. Third, gravity in the $x$ and $y$ directions is considered negligible. Finally, the physics based loss function being proposed is centered around the principle of conservation. So in this case Equation 94, the continuity equation, is not used because after losing its time dependent terms to scale analysis it becomes a balancing equation instead of a conservation equation like the others. Given all previous considerations the resulting conservation equations are:

$$\frac{\partial p}{\partial x} + \rho(\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}) = 0 \tag{96}$$

$$\frac{\partial p}{\partial y} + \rho(\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y}) = 0 \tag{97}$$

$$\rho\hat{C}_p(\frac{\partial T}{\partial t} + u\frac{\partial T}{\partial x} + v\frac{\partial T}{\partial y}) = 0, \tag{98}$$

where Equation 96 corresponds to solutions for u-wind, Equation 97 corresponds to solutions for v-wind and Equation 98 corresponds to solutions for temperature. These equations correspond to the total energy of the system, comprising of internal energy via temperature and kinetic energy via wind. To represent conservation of energy the three energy terms are isolated using the following reformulation:

$$\partial u = -\partial t(\frac{1}{\rho}\frac{\partial p}{\partial x} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}) \tag{99}$$

$$\partial v = -\partial t(\frac{1}{\rho}\frac{\partial p}{\partial y} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y}) \tag{100}$$

$$\partial T = -\partial t(u\frac{\partial T}{\partial x} + v\frac{\partial T}{\partial y}), \tag{101}$$

and then summed together. However, since $\partial u$, $\partial v$, and $\partial T$ are not in the same units

they must be manipulated further so they can be added. Specifically, since we are considering energy, they will all be converted to Joules. To do this, the $\partial u$ and $\partial v$ terms are squared and the absolute value of $\partial T * \hat{C}_p$ is taken, where $\hat{C}_p$ is the specific heat constant of 1004. So, the core of the physics loss is centered around the quantity $(\partial u)^2 + (\partial v)^2 + |1004 * \partial T|$. Finally, this physics loss would ideally be implemented in a closed system, understanding that conservation of energy and change of total energy in the system equaling zero are only absolutely valid in that case.

The implementation of this physics loss function on the predicted data cube will be presented by first considering this calculation at a single grid point and single point in time. At this candidate data point, there are eight partial derivatives that must be approximated, which in this case is done using first order finite differencing. In the most general sense this amounts to defining $\frac{\partial T}{\partial x_i}$ as approximately equal to the difference in temperature at the point to the right of $x_i$ and the point to the left of $x_i$ divided by the total distance separating $x_{i+1}$ and $x_{i-1}$. Since it desired that the loss function assess how physically wrong the predictions are with respect to our proposed loss, the first difference calculations are done using the weather attribute predictions. Note that because a first order finite difference approximation to the partial derivatives is used this results in the outer edges of the data cube being excluded from the physics loss function since they would not have predicted values on either side of the candidate value with which to perform the calculations.

Since the data is taken from a constant pressure level as will be discussed in the next section, geopotential height (Z) is used as the pressure related attribute. Therefore with the model outputting geopotential height, but the physics loss function requiring partial derivatives of pressure, a transformation was done within the loss

function to convert $\frac{\partial Z}{\partial x}$, and $\frac{\partial Z}{\partial y}$ to $\frac{\partial p}{\partial x}$ and $\frac{\partial p}{\partial y}$ respectively using the following formulas:

$$\frac{\partial p}{\partial x} = \rho g \frac{\partial Z}{\partial x} \tag{102}$$

$$\frac{\partial p}{\partial y} = \rho g \frac{\partial Z}{\partial y}, \tag{103}$$

The density calculation for this transformation, and also for Equations 99 and 100, was done using:

$$\rho = \frac{p}{RT} = \frac{700 \text{ hPa}}{287 * T}, \tag{104}$$

where pressure remains a constant at 700 hPa, R is the gas constant, and T is temperature. Here the true temperature at each point is used to calculate densities, though this is the only time a true value for one of the attributes is used as opposed to a predicted value as was previously mentioned.

Finally, calculating the quantity $(\partial u)^2 + (\partial v)^2 + |1004 * \partial T|$ for all interior points of the predicted data hyper-cube of shape $(B, F, T, N, N)$, where $B$ is the batch size, yields a cube of shape $(B, T-2, N-2, N-2)$. Obtaining a final scalar physics loss value for a given batch is done by averaging across both spatial dimensions $(N-2, N-2)$, then across the batch dimension $(B)$, and finally summing along the temporal dimension $(T-2)$.

### 3.4.3 Data

Given previous discussion, the data is unsurprisingly a collection of four weather attributes measured over time and space. As with weather clustering, the data comes from the HRRR models from October 2018 - November 2019. However, a much smaller geographic area, centered over Kansas and including parts of bordering states, was considered. The four weather attributes were temperature (K), geopotential height (m), u-wind (m/s), and v-wind (m/s). Geopotential height is often used in

place of directly measuring pressure, and it is simple to convert between the two. Instead of measuring pressure at a fixed altitude, geopotential height fixes pressure and measures the height at which the given pressure level occurs. In this case, the chosen pressure level was 700hPa.

The data was not usable directly out of the HRRR model dataset because the model predicts on a curvilear grid as opposed to a rectilinear grid. This means sets of points are not directly east-west and north-south of each other which is problematic for calculating partial derivatives with the finite difference method that assumes a rectilinear grid. To address this the data was interpolated to a rectilinear grid. Figure 26 illustrates the HRRR data that was collected to perform the interpolation overlayed on the new rectilinear interpolated grid boundaries.

An attempt was made to generate a final grid that remained approximately true to the original 3km resolution. This informed the choosing of the interpolation grid boundaries. Specifically, a bounding box for the interpolation grid that has approximately the same distance on all sides was desired. Clearly this is not possible due to the shape of the earth though, and one edge was necessarily longer. Further in pursuit of this goal, it would help to interpolate an approximately equal number of points in the interpolation grid as there are HRRR points that happen to intersect that same location. To accomplish this, the number of HRRR data points that fell within the bounding box created by the rectilinear grid boundaries as shown in Figure 26 were tabulated and found to be $28,170$. So, a final grid with 160 latitude and 160 longitude levels was chosen resulting in $25,600$ grid points.

Each observation is comprised of the four weather attributes over 24 hours, across a 16x16 grid, yielding input and truth arrays of size $(B, 4, 24, 16, 16)$, where B is the batch size. The truth and input arrays are also accompanied by $\partial x$, and $\partial y$ arrays containing the x and y distances respectively in meters between grid points, as these are


**Figure 26. Map showing the region of HRRR data collected (blue), and the region to which a rectilinear grid was interpolated using the HRRR data (yellow) [15].**

required for finite difference computations within the physics loss. Building the input and truth arrays such that no data is shared spatially or temporally from observation to observation, the interpolated grid yields $18,200$ observations. These observations were further segmented into training, validation, and test sets at an 80/10/10% split respectively. As is typical, to aid in neural network convergence each attribute is min-max normalized into the range [0,1]. Finally, since the partial derivative calculations are performed under the expectation that the data is in it's naturally occurring range of values the physics loss function un-normalizes the predictions before performing any calculations.

### 3.4.4 Model Fitting, Evaluation and Analysis Strategy

To assess the effect of the physics loss on predictive power, the prediction error for a model with both physics and MSE loss functions (physics-based model) will be compared to the prediction error of a model with just an MSE loss function (traditional model). Particular attention is paid to encouraging standardization between the two models so that differences in outcome with respect to prediction error can be attributed to the differences in loss functions. Specifically, the models are given the same initial starting weights, both optimized with ADAM at it's default learning rate of 0.001, and trained in the same loop so that they experience the same random batch shuffling. Additionally, each model is trained for a predetermined 35 epochs with a batch size of 32. Three instances of this experiment are performed, each time providing the models with a new random weight initialization for a more robust analysis of the results.

For model training, the physics loss was heavily discounted to ensure it does not overpower the MSE loss term. This is particularly important given that the MSE calculation is done on the min-max normalized data, while computation of the physics

loss function is performed back in the original data range. Given this, the total loss function for the physics-based model is:

$$\mathcal{L}_{total} = \mathcal{L}_{MSE} + 10^{-9}\mathcal{L}_{physics}. \tag{105}$$

# IV. Testing, Results, and Analysis

## 4.1 Clustering Results

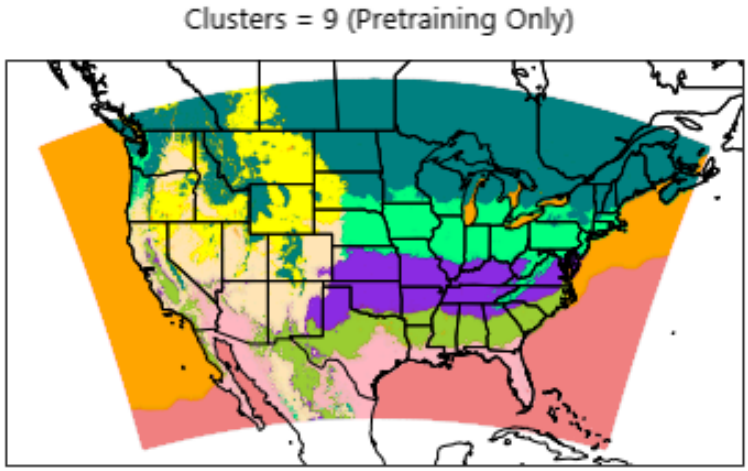
Since the literature review yeilded no existing work on clustering across geography given long weather time series data, there is no directly applicable work with which to compare the results. Instead there will be an inter-algorithm comparison of clustering results, comparison to an expert generated climate map, and comparison to various other maps such as terrain, elevation, mean annual temperature and sea surface temperature (SST) maps.

### 4.1.1 Inter-algorithm comparison

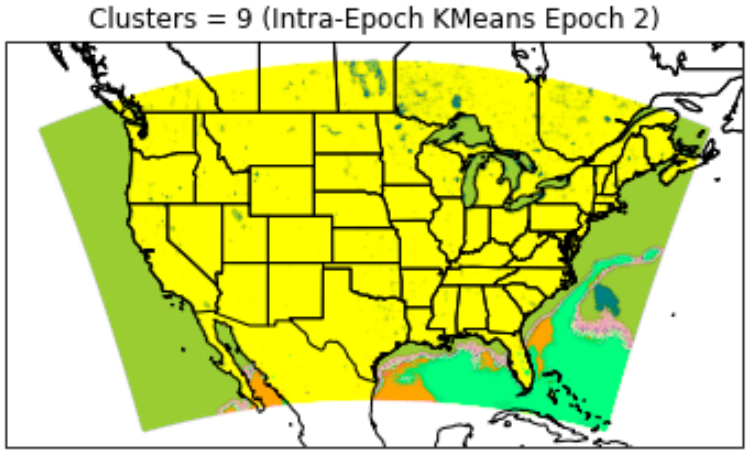
To begin, the cluster maps generated by each algorithm will be presented. Figure 27a shows the baseline cluster map from after pretraining but before any cluster specific training and Figures 27b and 27c show the cluster results after training for 2 epochs using SDTW-KM-DRC and SDTW-KLD-DRC respectively. It is immediately noted that SDTW-KM-DRC struggles significantly on this task compared to SDTW-KLD-DRC, and in fact acts better as a classifier of land versus water than as a clustering tool in this instance. Intuitively it appears that SDTW-KM-DRC considered the differences in temperature time series occurring over the water as more varied or different with respect to soft-DTW distance than that of those occurring over land and therefore chose to allocate all land-based temperatures to one cluster resulting in nearly all segmentation occurring over water-based points. Noting that this was not an issue in the cluster maps generated immediately after pretraining indicates further that in this case the relative importance given to differences in water-based locations as opposed to land-based locations arose, at least in part, as an artifact of the cluster training phase. It is suspected that this has to do with the hard clustering

assignment paired with the potential for imperfect centroids and cluster membership information generated via SDTW-KMeans since Lloyd's algorithm is only guaranteed to converge to a local minimum. It is worth noting that though this was not the expected outcome, the algorithm does a very good job of correctly segmenting water and land-based points. Especially considering that the baseline cluster map in Figure 27a had many water and land-based points incorrectly clustered - specifically and most notably the Great Lakes.

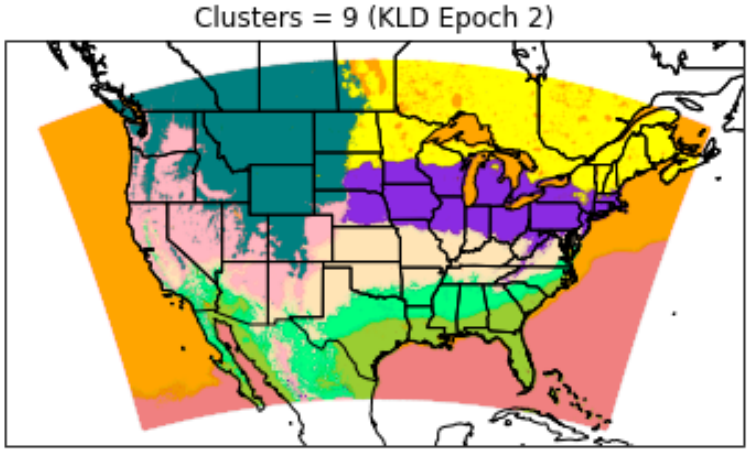
SDTW-KLD-DRC yielded results that were much closer to expected. Specifically, it should be noted that SDTW-KLD-DRC was also able to resolve many, if not all, incorrectly clustered water-based locations. This is particularly evident in the Great lakes region where initially Lake Superior was entirely grouped into a land-based cluster along with large parts of Lakes Michigan and Huron, and then after training with SDTW-KLD-DRC the lakes were correctly grouped with other bodies of water. Additionally, the SDTW-KLD-DRC generated clusters that roughly track with terrain features and follow latitude bands. These type of analyses are best done by comparing the cluster maps directly to external maps.




(a)




(b)




(c)

**Figure 27. Cluster maps for k=9. The initial cluster results coming from the autoencoder alone are re-displayed in (27a), and the SDTW-KM-DRC and SDTW-KLD-DRC results after training for two epochs are shown in (27b) and (27c) respectively.**

### 4.1.2 External Map Comparisons

As is evident from the previous section the cluster map generated by SDTW-KM-DRC did not yield cluster results conducive to comparison against various external feature maps, so all comparisons will be done against the cluster results generated via SDTW-KLD-DRC.

#### Geographical, Elevation, and Temperature Maps

This section will compare the cluster results from SDTW-KLD-DRC against maps representing the physical features and temperature of the region. First, consider the temperature map in Figure 28. This will serve a reference point for further comparison of the SDTW-KLD-DRC cluster results alongside elevation and geographical maps. In addition to changes in temperature based on elevation, primarily in the Western U.S. that will be discussed shortly, there are other noteworthy similarities between the temperature map and the SDTW-KLD-DRC generated cluster map. The Southern and Mid-Atlantic regions of the country have nearly the same breakdown in the temperature map and cluster map, provided that some of the temperature map categories are collapsed to closer match the cluster map resolution. This points to the idea that SDTW-KLD-DRC picked up heavily on "typical" temperature behavior across locations, among other things.

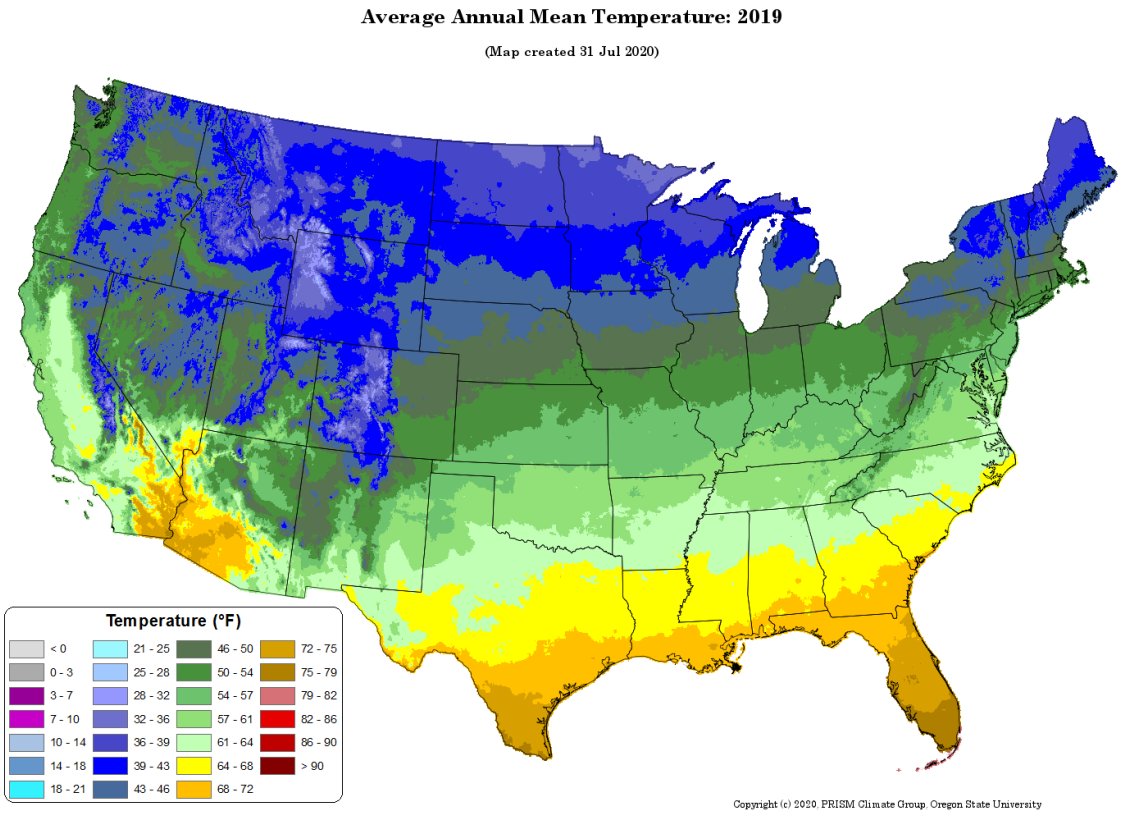



**Figure 28. Mean annual temperature (Fahrenheit) [16].**

Figure 29 provides an elevation specific comparison. It is immediately evident that the cluster results pick up on differences in elevation. This is not surprising since temperatures frequently change with respect to elevation, with higher elevations being lower in temperature and vice-versa. The teal/blue and pink clusters in the western U.S. from Figure 29b seem to correspond with mountainous regions. Within those two clusters it appears that the pink represent slightly warmer regions than the those represented by teal/blue. For example, in southwest Idaho there is a valley that clearly evidences this known relationship between temperature and elevation and is reflected in the cluster results. Figure 28 shows all of Idaho as having an average temperature between 32-50 degrees Fahrenheit except for this specific valley. The elevation map in Figure 29a shows a clear U-shaped region of low elevation, and the cluster map mirrors this with a corresponding U-shaped portion of the pink cluster

extending into Idaho. The same pattern can be seen surrounding the Cascade Range extending down from Canada into western Washington and Oregon as well as the Appalachian Mountains.

More broadly, there is a clear division that occurs in the cluster map along the Continental Divide. There are some clusters that span this divide, though they track almost precisely from east to west along the southern edge of the Rocky Mountains. Specifically, the tan, green and light green clusters, which primarily represent the South and parts of the midwest, extend along the southern border of the Rocky Mountains and into parts of southern Arizona, New Mexico, and California. This relationship is also substantiated in the temperature map (Figure 28).

Figure 30 provides comparison with respect to a slightly different map. Though it still provides some idea of elevation it focuses more on geographical and topographical characteristics. California's central valley is an interesting region for discussion as it has a vastly different geography from the rest of the state and yields significant agricultural production unlike most other areas in the Western U.S [95]. Considering this, note that California's central valley region in the SDTW-KLD-DRC cluster map is a combination of the light green, tan, and pink clusters and therefore bears some similarity to parts of the south.

There is one cluster in specific who's boundary does not have a clear reason - the teal/blue cluster in the Northwest region of the country. As previously discussed, the teal/blue and pink clusters represent much of the West and all of the Northwest region. The majority of the area they cover makes sense in regards to what locations each cluster represents. Specifically, the portion of the pink cluster in Nevada corresponds to The Great Basin, and the portion near the Four Corners of Arizona, Colorado, Utah, and New Mexico corresponds to the Colorado Plateau. Then nearly all of the teal/blue cluster that borders the pink cluster corresponds to either the

Cascade mountains in the Northwest or the Rockies which, in referring to Figures 29 and 30, have an easternmost point near the center of Colorado and then cut Northwest across Wyoming and central Montana. However, from that same point in Colorado the teal/blue cluster instead extends Northeast through the centers of North and South Dakota. The reason for this is suspected to be that even though Eastern Montana, Wyoming and North and South Dakota do not necessarily share the same mountainous terrain, they are more similar with respect to temperature and other dynamical features identified by SDTW-KLD-DRC to the other locations in the teal/blue cluster than to its neighboring clusters. It is also clear that different values of $k$, specifically higher ones, would add expressiveness to the model and possibly have resulted in the aforementioned region being assigned it's own cluster.

### Climate Classification Systems

The final cluster comparisons will be done by comparing the SDTW-KLD-DRC generated cluster map with the Koppen-Geiger climate classification system. Originally proposed by Wladimir Koppen in 1900 and later updated by Rudolph Geiger, the Koppen-Geiger climate classification system remains the most widely used to this day [96]. The system is built by combining three different factors: biome or vegetation zones , precipitation, and temperature. The Koppen-Keiger classification map is shown in Figure 31.

The Koppen-Geiger climate map mirrors much of what is seen in SDTW-KLD-DRC's cluster map, specifically in regard to latitude bands in the eastern part of the country and division occurring around the Continental Divide. Additionally, the area around the Appalachian Mountains in western Virginia and West Virginia that extends down from the purple cluster into the tan cluster from the SDTW-KLD-DRC map is mirrored in the blue, light blue, and light green colored areas in Figure 31.

In most of the map however a one-to-one comparison is not possible because of the different number of groupings used in each method, particularly in the western United States. The broadest characterization that can be made with respect to the $\approx 6$ classes Koppen-Geiger assigns to the western United States (excluding the coast) is that it has a dry/cold biome with varying temperatures. So, according to Koppen-Geiger, and in combining the characterizations of the $\approx 6$ classes, the pink and blue/teal clusters generated by SDTW-KLD-DRC would also be considered dry/arid and with a wide temperature range. Koppen-Geiger breaks up the eastern United States into two main groups that are divided approximately down the middle latitude of the tan cluster from the SDTW-KLD-DRC results. Below that line is considered to have a temperate biome and hot summers while above that line has a cold biome and hot summers.

#### Sea Surface Temperature Map

Though the primary point of interest in this research is land-based clusters, it is worth quickly analyzing whether the water-based clusters also had a similar structure to any external maps. Figure 32 shows the sea surface temperature around North America. The overall pattern of cooler water extending further down in the Pacific and being more similar to the Northern Atlantic waters is precisely the breakdown that is seen in the SDTW-KLD-DRC cluster maps.

### 4.1.3 Land-based only SDTW-KM-DRC

Since SDTW-KM-DRC did not perform as expected, it was run on a subset of land-based only data to see how it would behave without the presence of water. The algorithm was run for 33 iterations, where again an iteration consists of 500 batches of 32 points (i.e., $16,000$ points) and the results then checked to see if in fact the outcome

is different after removing the water-based data points. Since the two water-based clusters were removed, $k = 7$.

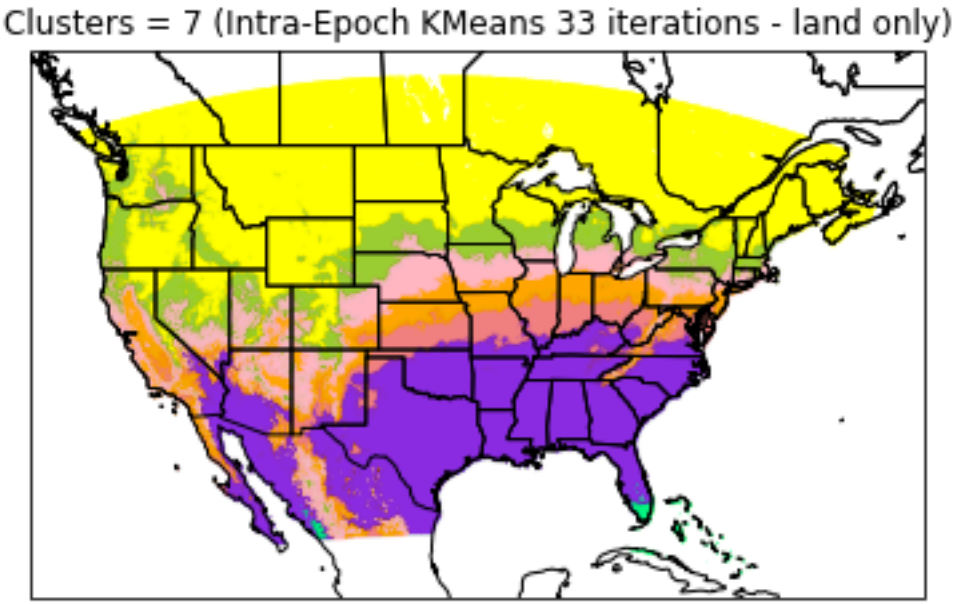


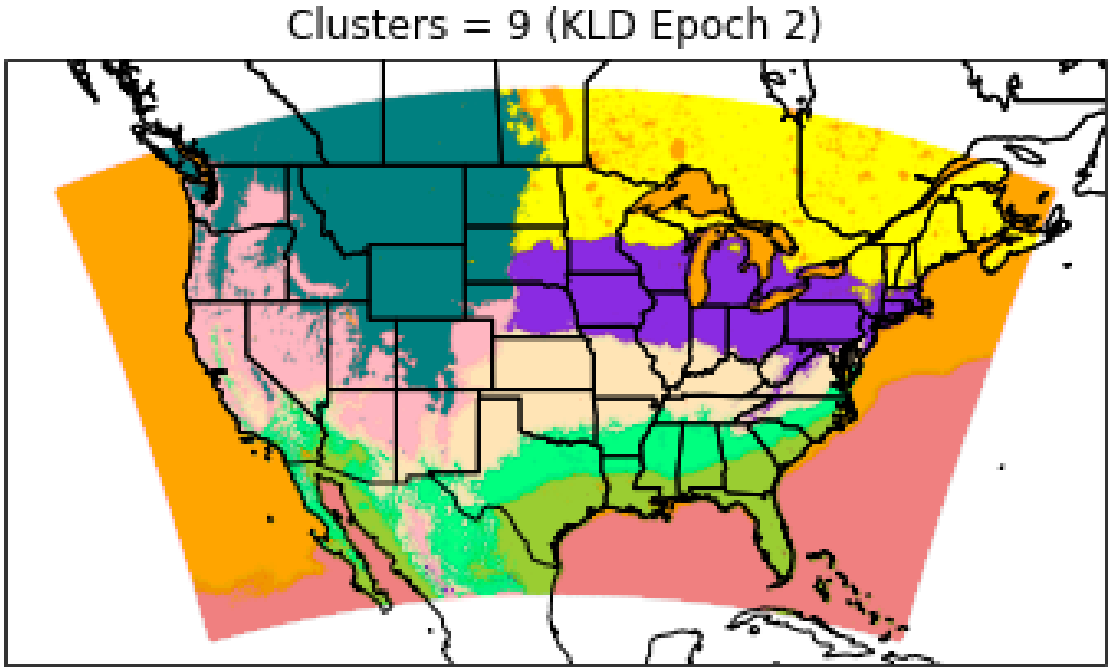

**Figure 33. SDTW-KM-DRC test after removal of water-based clusters.**

The clustering map shown in Figure 33 displays some of the same characteristics as the clusters resulting from full analysis with SDTW-KLD-DRC, but has some notable differences as well. The western United States, particularly where the Rocky Mountains border the eastern edge of the Great Basin, is quite similar in each map. Additionally, The Appalachian Mountain region, specifically in West Virginia and western Virginia belongs to the clusters north of them as opposed to the cluster in purple cluster in the same latitude band.

Of all the external maps, Figure 33 has the most characteristic/feature overlap with Figure 28, the mean annual temperature map. This particular level of similarity was not evidenced in the SDTW-KLD-DRC result. Specifically, notice how while

there is a clear point near the Continental Divide at which the cluster formation starts being influenced by mountainous terrain, the clusters from Figure 33 span the entire country which was not the case with SDTW-KLD-DRC, but is the pattern displayed in the mean annual temperature map. This particular similarity could indicate that SDTW-KM-DRC picked up heavily on the average of the time series as one of its predominant features.

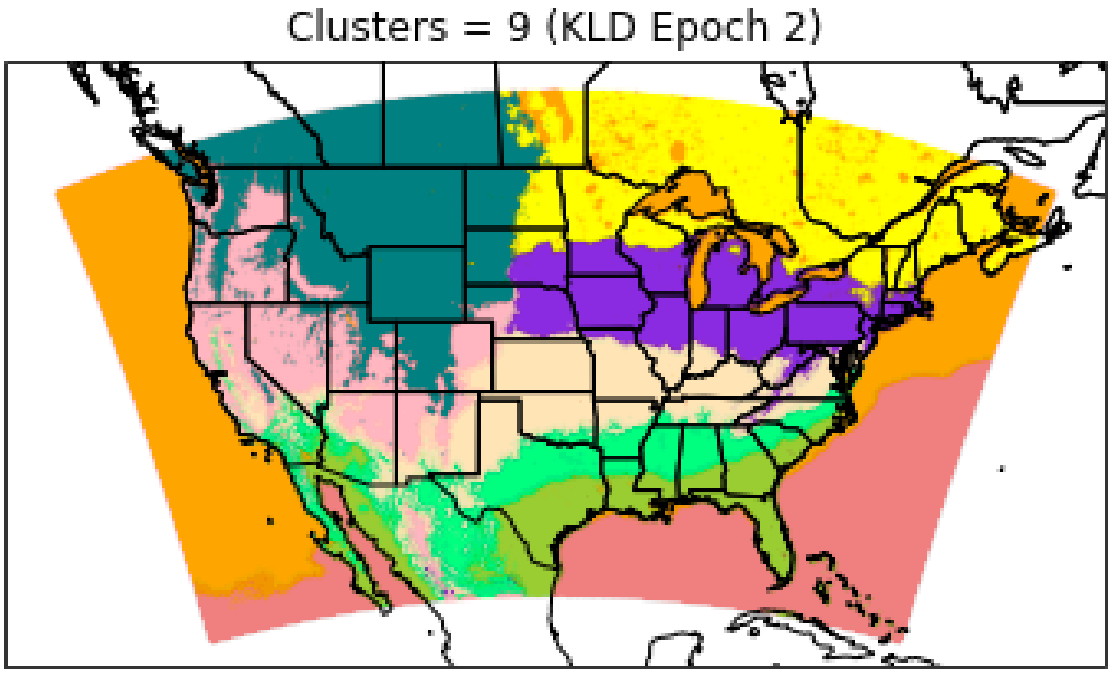

(a) Elevation map [93].




(b) SDTW-KLD-DRC cluster map.

**Figure 29. Elevation comparison.**


(a) Geographical map [94].




(b) SDTW-KLD-DRC cluster map.

**Figure 30. Geographical comparison.**

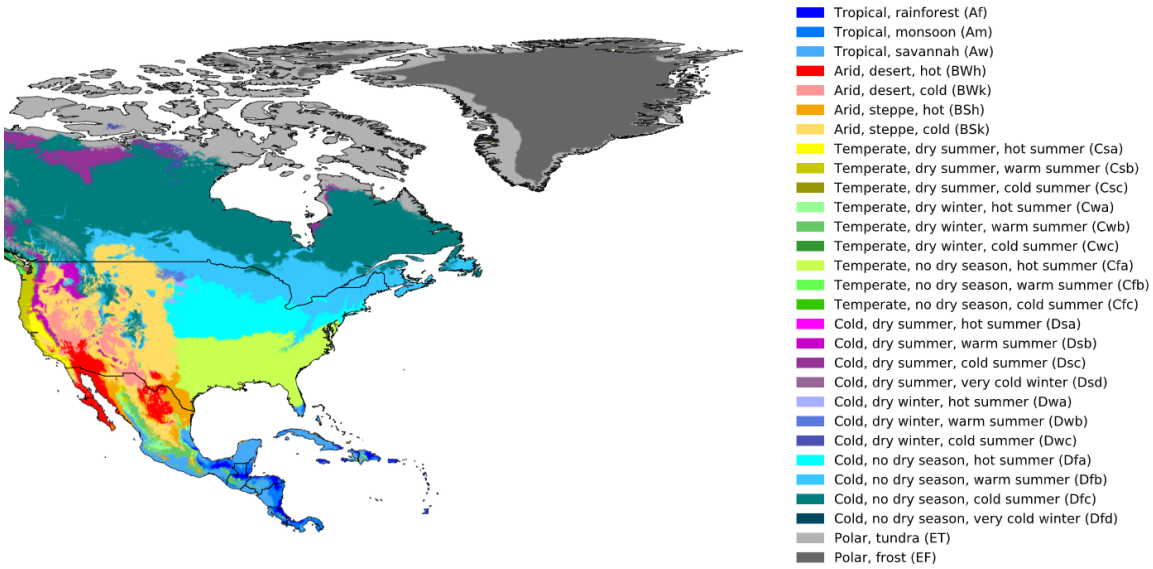



**Figure 31. Koppen-Geiger climate classification map for North America from 1980-2016 [17]**

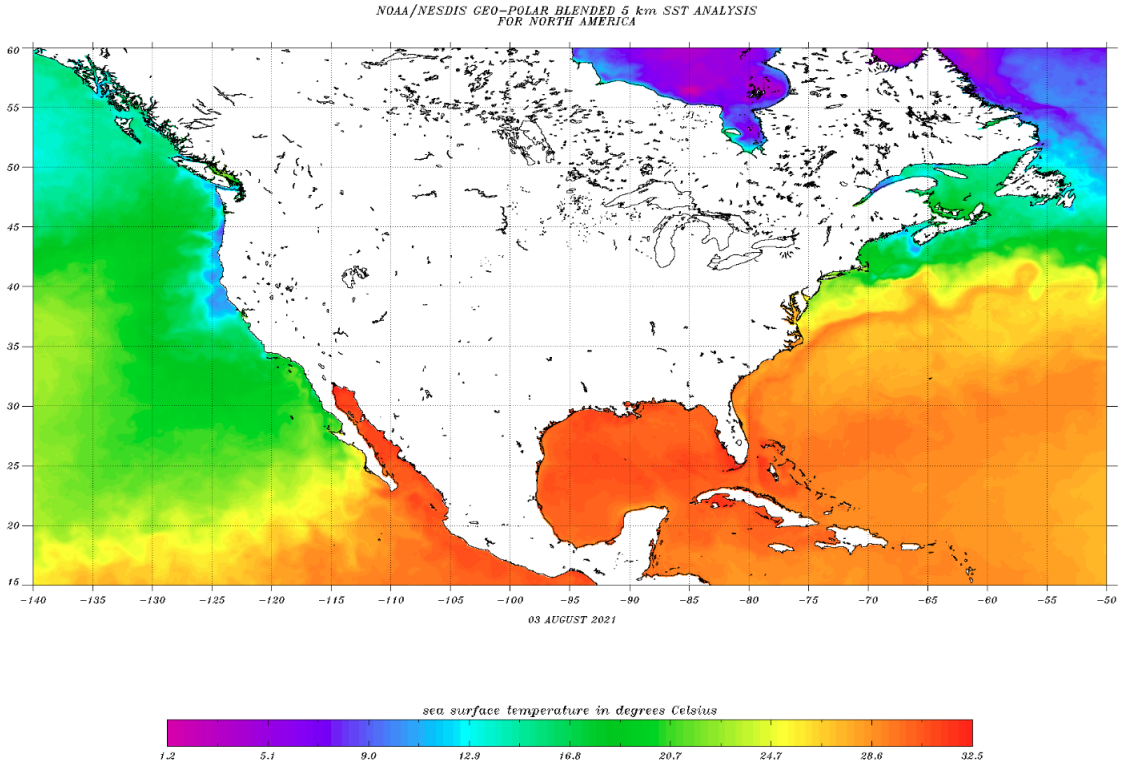



**Figure 32. North American SST's [18].**

### 4.2 Physics Informed Deep Weather Forecasting

Each instance of running standardized physics-based and traditional models yielded very similar results. At the beginning of training there was no distinct advantage shown by either model, but by the $20^{th}$ epoch the physics based models consistently started outperforming the traditional models with respect to both validation and training errors. Figure 34 presents these results. The left column of this figure provides the raw results over the 35 epochs of training, while the right column gives a zoomed in view starting at epoch 5 to present the results on an easier to view scale.

Figure 34 is a good indication that the addition of the physics loss to a traditional model, when compared to the traditional model alone, is beneficial with respect to both training and validation error. However, to properly estimate the true out-of-sample performance the models are applied to the test dataset. Specifically, within each instance the physics and traditional model weights corresponding to the minimum validation error are loaded into the model and then applied to the test data. Table 5 displays these results.

The first instance was picked for some extended analysis. Specifically, over the pre-determined training period of 35 epochs the physics models outperformed across the board, but the possibility of these results changing over a longer training horizons was considered. To investigate this Instance 1 was run for an additional 40 epochs. It was found that the original result holds with the physics model indeed outperforming

| **Test Set MSE Comparison** | | |
|---|---|---|
| | **Physics-based model** | **Traditional model** |
| **Instance 1** | **0.00313** | 0.00376 |
| **Instance 2** | **0.00303** | 0.00425 |
| **Instance 3** | **0.00274** | 0.00425 |

**Table 5. Comparison of physics-based and traditional neural network test MSE across all three instances. Note that the physics-based model achieves a lower test MSE across all three instances.**

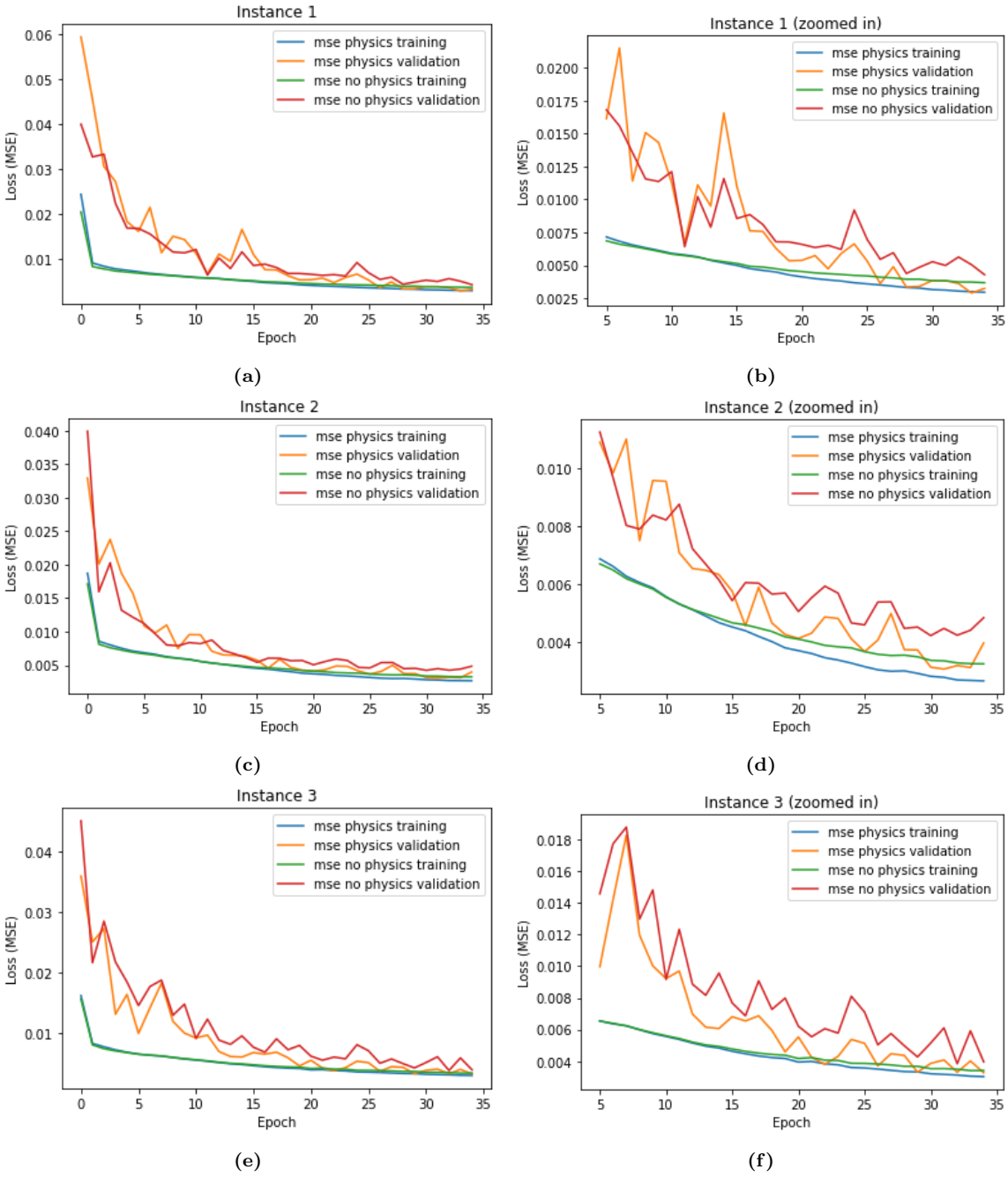



**Figure 34. Physics-based model loss vs traditional model loss instances.**

over a longer training horizon as well. Figure 35 illustrates this on a zoomed in scale showing epochs 20-75. Table 6 reports the test set MSE's over the 75 epoch extended analysis which again indicates that the physics model outperforms the traditional over a longer training horizon with respect to out-of-sample testing.

Finally, it is worth noting that not only does the physics-based model outperform with respect to validation and test errors, i.e., generalize well, it also outperforms on the training data itself. This is important because if it only performed better with respect to validation and test errors it could be suspected that the physics loss was somehow merely acting as a regularizer and depressing the network weights. Moreover, regularizers can often hurt training error and at best typically have no effect on it. In this case though, the training errors for the physics-based models are consistently lower than their traditional counterparts beginning at around the $15^{th}$ epoch. This suggests that the physics loss function is actually helping the network learn.

| **Test Set MSE Extended Analysis** | | |
|---|---|---|
| | **Physics-based model** | **Traditional model** |
| **Instance 1 (75 epochs)** | **0.00208** | 0.00297 |

**Table 6. Test set MSE for Instance 1 analysis extended to 75 epochs.**

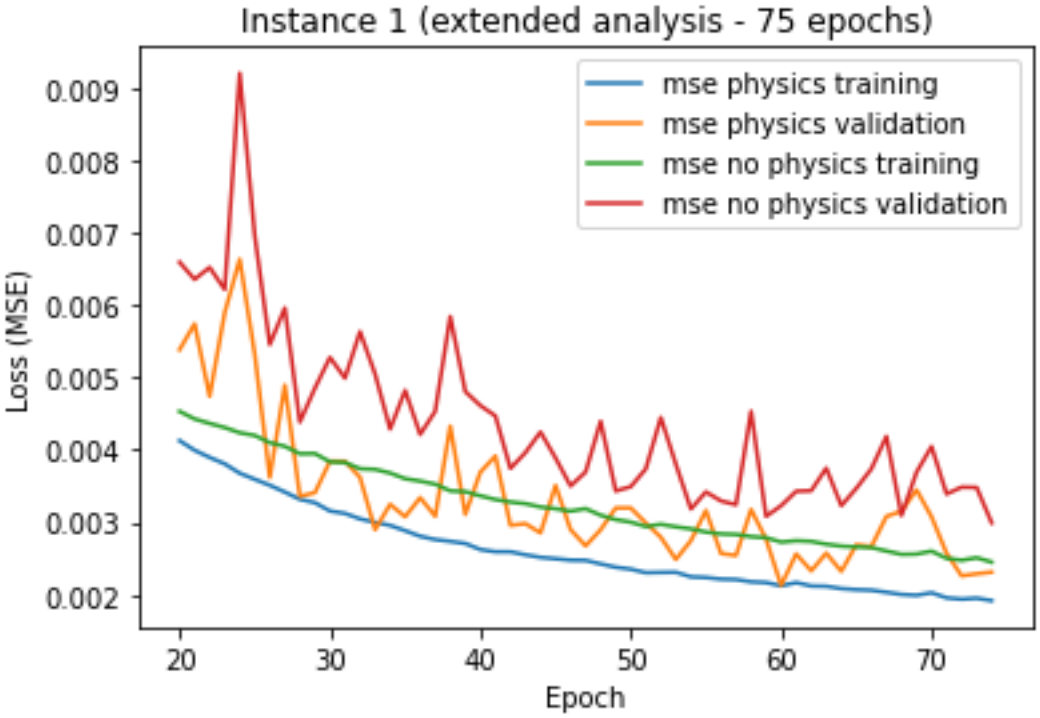



**Figure 35. Training and validation error for physics-based model over extended analysis 75 epoch training period for Instance 1 (zoomed in).**

# V. Conclusions and Future Work

The research objectives will be discussed in relation to the results presented for each contribution as well as future work that could be done.

## 5.1 Clustering

### 5.1.1 SDTW-DRC Algorithm Conclusions

SDTW-KM-DRC and SDTW-KLD-DRC address the need for a time series appropriate DRC algorithm. Soft-DTW, a differentiable version of the DTW operator takes the place of Euclidean based distance metrics in the clustering layers of each algorithm. The differentiable nature of soft-DTW as a loss function allows the autoencoder weights in SDTW-KM-DRC and SDTW-KLD-DRC to update in a way that yields a cluster friendly latent spaces with respect to the shape and dynamics of the latent embeddings. After an autoencoder pretraining phase to provide a meaningful latent space at the onset of cluster specific training, the two algorithms offer slightly different approaches to the same problem. Throughout training with SDTW-KM-DRC, the soft-DTW distance between embedded data points and their respective centroids is minimized while there are periodic updates to both the cluster centroids and cluster membership using SDTW-KMeans. Additionally, once training is completed the SDTW-KMeans algorithm must be run one more time to yield the final clustering result. On the other hand, SDTW-KLD-DRC learns both the cluster centroids and cluster membership vector internally. Specically, the SDTW-KLD-DRC autoencoder has an additional set of weights off of the bottleneck that represent the cluster centroids and are thus learned directly. Moreover, the cluster membership information is a direct output of the network in the form of probabilities ($q_{i,:}$) associated with each embedded data point belonging to a given centroid.

### 5.1.2 Weather Clustering Conclusions

As was evidenced in Chapter IV, SDTW-KM-DRC and SDTW-KLD-DRC yeilded very different results. Given the inclusion of both water and land-based locations in the dataset, SDTW-KM-DRC acted more as a discriminator of land and water than as a clustering algorithm. Intuitively it is suspected that throughout training, and as embedded data points are pressured to align with their centroids, SDTW-KM-DRC considered the differences among water-based locations as being more prominent than the differences between land-based locations and thus relegated all land-based locations to one cluster to allow for maximal segmentation among water-based locations. SDTW-KLD-DRC did not suffer this issue, and yielded cluster map results along the lines of what was expected. The precise reason for this difference is not known though it is suspected that the extra degree of subtlety inherent to SDTW-KLD-DRC due to the fact that it uses soft cluster assignments as opposed to hard cluster assignments is a contributing factor. The SDTW-KLD-DRC cluster maps were compared to a number of external maps to analyze their meaning, specifically geographical/terrain, elevation, mean annual temperature, sea surface temperature, and expert generated climate maps. The SDTW-KLD-DRC cluster map roughly followed terrain and elevation maps while also resolving mis-classifications of water-based locations that were evident in the initial autoencoder only pre-trained cluster map. Specifically, there was a clear division along the Continental Divide and along prominent mountain ranges there was notable cluster separation. Additionally, in the eastern part of the United States the clusters followed latitude bands. Furthermore, with respect to the widely used Koppen-Geiger climate classification map, the SDTW-KLD-DRC clusters were fairly similar in that the eastern part of the United States is characterized by a series of horizontal clusters, while on the western side of the Continental Divide the clusters are more interspersed due to significant variation in terrain. Finally, the water-based

clusters generated by SDTW-KLD-DRC very closely mirror sea surface temperature (SST). Taken as a whole, these results indicate that SDTW-KLD-DRC applied to the task of geographical clustering provides meaningful insight in regards to broad climatic characterizations of the United States.

### 5.1.3 Clustering Future Work

Since SDTW-KM-DRC did not perform as expected in this application of large-scale weather clustering, there are a couple of things within the existing framework that could be explored. First, within the SDTW-KMeans step that employs Lloyd's algorithm, multiple iterations could be done from random starting centroids to help avoid using a result stemming from a local minimum. Additionally, if computationally feasible, the subsampling size and number of samples could be increased, or done away with altogether in favor of using the entire dataset for clustering at each cluster update step. Alternatively an entirely different clustering algorithm could be dropped in place of Lloyd's algorithm. Specifically, something like k-medoids or a kernel based method could be explored.

In terms of the specific application of clustering weather data, there are a number of things that could still be examined. First, it could be more beneficial to use a de-trended and de-seasonalized version of the time series. This would force the DRC algorithms to look at the residuals of the time series data, and thus pick up on the similarity in how different geographical locations tend to deviate from their normal behavior. This could in fact be an even better way to focus on the dynamics of the weather at different locations. Additionally, specific choices regarding the architecture and model tuning could be explored. For instance, the latent embedding space could be made smaller, and various hyperparameters like learning rate, loss component weights, and the soft-DTW smoothing parameter $\gamma$ could be tuned. Finally, this

analysis focused solely on temperature, but making this a multivariate analysis to include other weather attributes like pressure, wind-speeds, dew point, etc. could be very valuable.

## 5.2 Physics Informed Deep Weather Forecasting

### 5.2.1 Physics Informed Deep Weather Forecasting Conclusions

A physics-based loss function that relies on the notion of conservation of energy and was derived from the Navier-Stokes equations was used as a means for encoding prior physical knowledge into the training of a deep neural network for weather prediction. To assess the effectiveness of the proposed physics-based loss function a series of three experiments were run. Specifically, three instances of the physics-based and traditional models having the same initial starting weights, hyperparameter values, and batch training order were trained for a predetermined 35 epochs. The physics-based model outperformed the traditional model in all three experimental instances with respect to both validation and training error. Likewise, in all three instances the out-of-sample test set error was lower. To further assess the robustness of this result across a longer training horizon, the models from Instance 1 were trained for an additional 40 epochs. The result remained valid over the longer 75 epoch training horizon. Specifically, the physcis based model validation and training errors outperformed the traditional model errors, and the out-of-sample test error was also lower. Furthermore, lower training errors for the physics based models across all three instances indicates that the physics loss is not merely acting as a regularizer and is instead actually helping the network learn. These results indicate that the proposed physics based loss function is effective at both helping a model learn and generalize to the task of weather prediction when compared to a model equipped with only an MSE loss function.

### 5.2.2 Physics Informed Deep Weather Forecasting Future Work

Since the proposed loss function was merely a proof of concept, there a number of things that could be examined in the future. The observation sizes were chosen to be a $16 \times 16$ grid, though for particular applications and even for the purposes of examining how a broader spatial region affects performance, this could be changed. Similarly, the 24-hour in 24-hour out prediction scheme could also be re-examined. It is quite possible that more input data could help the network pick up on broader long-range trends which could aid in predictive power. The spatially-aware TCN architecture and model training protocol could also be further tuned with respect to kernel sizes, number of filters, dropout, batch size, learning rate, etc. Other architectures like ConvLSTM's would also be appropriate for this task and could yield interesting results. Finally, it would be interesting to look specifically at the effect that the weight applied to the physics loss has on training. Specifically, at what level does this weight provide maximum benefit. Clearly it provides benefit at the level used for this proof of concept, but would increasing it yield further improvements and what happens when it starts to become equal to or greater in magnitude than MSE?

# Bibliography


1. P. Bauer, A. Thorpe, and G. Brunet, "The quiet revolution of numerical weather prediction," *Nature*, vol. 525, no. 7567, p. 47, 2015.

2. I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

3. P. Zhang, C. Lan, J. Xing, W. Zeng, J. Xue, and N. Zheng, "View adaptive recurrent neural networks for high performance human action recognition from skeleton data," 03 2017.

4. M. Sugiyama, H. Sawai, and A. Waibel, "Review of tdnn (time delay neural network) architectures for speech recognition," pp. 582 – 585 vol.1, 07 1991.

5. S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *arXiv preprint arXiv:1803.01271*, 2018.

6. E. de Bézenac, A. Pajot, and P. Gallinari, "Deep learning for physical processes: incorporating prior scientific knowledge," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2019, p. 124009, dec 2019.

7. X. Jia, J. Willard, A. Karpatne, J. Read, J. Zwart, M. Steinbach, and V. Kumar, "Physics guided rnns for modeling dynamical systems: A case study in simulating lake temperature profiles," in *Proceedings of the 2019 SIAM International Conference on Data Mining*, pp. 558–566, SIAM, 2019.

8. F. Ferstl, M. Kanzler, M. Rautenhaus, and R. Westermann, "Time-hierarchical clustering and visualization of weather forecast ensembles," *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 1, pp. 831–840, 2017.

9. S. Totz, E. Tziperman, D. Coumou, K. Pfeiffer, and J. Cohen, "Winter precipitation forecast in the european and mediterranean regions using cluster analysis," *Geophysical Research Letters*, vol. 44, no. 24, pp. 12–418, 2017.

10. W. S. Yackinous, "Chapter 10 - fundamentals of nonlinear dynamics," in *Understanding Complex Ecosystem Dynamics* (W. S. Yackinous, ed.), pp. 177 – 191, Boston: Academic Press, 2015.

11. S. Wallot and G. Leonardi, "Analyzing multivariate dynamics using cross-recurrence quantification analysis (crqa), diagonal-cross-recurrence profiles (dcrp), and multidimensional recurrence quantification analysis (mdrqa) – a tutorial in r," *Frontiers in Psychology*, vol. 9, 12 2018.

12. S. Aghabozorgi, A. S. Shirkhorshidi, and T. Y. Wah, "Time-series clustering–a decade review," *Information Systems*, vol. 53, pp. 16–38, 2015.

13. X. Guo, L. Gao, X. Liu, and J. Yin, "Improved deep embedded clustering with local structure preservation.," in *IJCAI*, pp. 1753–1759, 2017.

14. P. Remy, "Temporal convolutional networks for keras." `https://github.com/philipperemy/keras-tcn`, 2020.

15. Google Earth, "Map outlining interpolation over kansas." `https://earth.google.com/web/`.

16. PRISM Climate Group, "Average annual mean temperature: 2019." `https://prism.oregonstate.edu/recent/monthly.php`.

17. H. E. Beck, N. E. Zimmermann, T. R. McVicar, N. Vergopolan, A. Berg, and E. F. Wood, "Present and future köppen-geiger climate classification maps at 1-km resolution," *Scientific data*, vol. 5, no. 1, pp. 1–12, 2018.

18. N. Oceanic and A. A. (NOAA), "Noaa/nesdis sst map north america." `https://www.ospo.noaa.gov/Products/ocean/sst/contour/`.

19. M. A. Nielsen, *Neural networks and deep learning*. Morgan Kaufmann Publishers, 3 ed., 2012.

20. C.-K. Peng, S. Havlin, H. E. Stanley, and A. L. Goldberger, "Quantification of scaling exponents and crossover phenomena in nonstationary heartbeat time series," *Chaos: an interdisciplinary journal of nonlinear science*, vol. 5, no. 1, pp. 82–87, 1995.

21. P. Lynch, "The origins of computer weather prediction and climate modeling," *Journal of Computational Physics*, vol. 227, no. 7, pp. 3431–3444, 2008.

22. A. Chattopadhyay, P. Hassanzadeh, and S. Pasha, "Predicting clustered weather patterns: A test case for applications of convolutional neural networks to spatio-temporal climate data," *Scientific Reports*, vol. 10, 2020.

23. M. Cuturi and M. Blondel, "Soft-dtw: A differentiable loss function for time-series," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, p. 894–903, JMLR.org, 2017.

24. F. G. Shuman, "History of numerical weather prediction at the national meteorological center," *Weather and Forecasting*, vol. 4, no. 3, pp. 286–296, 1989.

25. Joint Chiefs of Staff, "Meteorological and Oceanographic Operations," 2010.

26. M. A. Nielsen, *Neural networks and deep learning*, vol. 2018. Determination press San Francisco, CA, USA:, 2015.

27. E. P. Willis and W. H. Hooke, "Cleveland abbe and american meteorology, 1871–1901," *Bulletin of the American Meteorological Society*.

28. V. Bjerknes, "The problem of weather prediction, considered from the viewpoints of mechanics and physics," *Meteorologische Zeitschrift - METEOROL Z*, vol. 18, pp. 663–667, 12 2009.

29. R. A. Pielke Sr, T. Matsui, G. Leoncini, T. Nobis, U. S. Nair, E. Lu, J. Eastman, S. Kumar, C. D. Peters-Lidard, Y. Tian, *et al.*, "A new paradigm for parameterizations in numerical weather prediction and other atmospheric models," *National Weather Digest*, vol. 30, pp. 93–99, 2006.

30. M. Ghil and P. Malanotte-Rizzoli, "Data assimilation in meteorology and oceanography," vol. 33 of *Advances in Geophysics*, pp. 141 – 266, Elsevier, 1991.

31. Z. Zhang and M. Sabuncu, "Generalized cross entropy loss for training deep neural networks with noisy labels," in *Advances in neural information processing systems*, pp. 8778–8788, 2018.

32. A. H. Waibel, T. Hanazawa, G. E. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 37, pp. 328–339, 1989.

33. F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *arXiv e-prints*, pp. arXiv–1511, 2015.

34. X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," *Advances in neural information processing systems*, vol. 28, 2015.

35. M. Dalto, J. Matuko, and M. Vaak, "Deep neural networks for ultra-short-term wind forecasting," *2015 IEEE International Conference on Industrial Technology (ICIT)*, pp. 1657–1663, 2015.

36. P. Hewage, M. Trovati, E. Pereira, and A. Behera, "Deep learning-based effective fine-grained weather forecasting model," *Pattern Analysis and Applications*, vol. 24, no. 1, pp. 343–366, 2021.

37. S. Jeong, I. Park, H. S. Kim, C. H. Song, and H. K. Kim, "Temperature prediction based on bidirectional long short-term memory and convolutional neural network combining observed and numerical forecast data," *Sensors*, vol. 21, no. 3, p. 941, 2021.

38. S. Rasp and N. Thuerey, "Data-driven medium-range weather prediction with a resnet pretrained on climate simulations: A new model for weatherbench," *Journal of Advances in Modeling Earth Systems*, vol. 13, no. 2, p. e2020MS002405, 2021.

39. M. Schultz, C. Betancourt, B. Gong, F. Kleinert, M. Langguth, L. Leufen, A. Mozaffari, and S. Stadtler, "Can deep learning beat numerical weather prediction?," *Philosophical Transactions of the Royal Society A*, vol. 379, no. 2194, p. 20200097, 2021.

40. P. Balachandran, *Fundamentals of compressible fluid dynamics*. PHI Learning Pvt. Ltd., 2006.

41. R. P. Chhabra, "Non-newtonian fluids: an introduction," in *Rheology of complex fluids*, pp. 3–34, Springer, 2010.

42. B. R. Bird, W. E. Steward, and E. N. Lightfoot, *Transport Phenomena*. Johh Wiley & Sons, Inc., 2002.

43. M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations," *arXiv preprint arXiv:1711.10561*, 2017.

44. M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics informed deep learning (part II): data-driven discovery of nonlinear partial differential equations," *CoRR*, vol. abs/1711.10566, 2017.

45. N. A. K. Doan, W. Polifke, and L. Magri, "Physics-informed echo state networks for chaotic systems forecasting," in *International Conference on Computational Science*, pp. 192–198, Springer, 2019.

46. E.-H. Han, G. Karypis, V. Kumar, and B. Mobasher, "Hypergraph based clustering in high-dimensional data sets: A summary of results," *IEEE Data Eng. Bull.*, vol. 21, no. 1, pp. 15–22, 1998.

47. D. Xu and Y. Tian, "A comprehensive survey of clustering algorithms," *Annals of Data Science*, vol. 2, no. 2, pp. 165–193, 2015.

48. A. Chattopadhyay, E. Nabizadeh, and P. Hassanzadeh, "Analog forecasting of extreme-causing weather patterns using deep learning," *Journal of Advances in Modeling Earth Systems*, vol. 12, no. 2, p. e2019MS001958, 2020.

49. A. Chattopadhyay, P. Hassanzadeh, and S. Pasha, "Predicting clustered weather patterns: A test case for applications of convolutional neural networks to spatio-temporal climate data," *Scientific reports*, vol. 10, no. 1, pp. 1–13, 2020.

50. K. A. McKinnon, A. Rhines, M. Tingley, and P. Huybers, "Long-lead predictions of eastern united states hot days from pacific sea surface temperatures," *Nature Geoscience*, vol. 9, no. 5, pp. 389–394, 2016.

51. A. Zarnani, P. Musilek, and J. Heckenbergerova, "Clustering numerical weather forecasts to obtain statistical prediction intervals," *Meteorological Applications*, vol. 21, no. 3, pp. 605–618, 2014.

52. K. Steinhaeuser, N. V. Chawla, and A. R. Ganguly, "Comparing predictive power in climate data: Clustering matters," in *International symposium on spatial and temporal databases*, pp. 39–55, Springer, 2011.

53. P. Pons and M. Latapy, "Computing communities in large networks using random walks," in *J. Graph Algorithms Appl*, Citeseer, 2006.

54. H. Kantz and T. Schreiber, *Nonlinear time series analysis*. Cambridge university press, 2 ed., 2004.

55. K. T. Alligood, T. D. Sauer, and J. A. Yorke, *Chaos*. Springer, 1996.

56. V. A. Zorich and R. Cooke, *Mathematical analysis I*. Springer, 2004.

57. A. J. Zomorodian, *Topology for computing*, vol. 16. Cambridge university press, 2005.

58. T. Sauer, J. A. Yorke, and M. Casdagli, "Embedology," *Journal of statistical Physics*, vol. 65, no. 3-4, pp. 579–616, 1991.

59. F. Takens, "Detecting strange attractors in turbulence," in *Dynamical systems and turbulence, Warwick 1980*, pp. 366–381, Springer, 1981.

60. L. Uzal, G. Grinblat, and P. Verdes, "Optimal reconstruction of dynamical systems: A noise amplification approach," *Physical review. E, Statistical, nonlinear, and soft matter physics*, vol. 84, p. 016223, 07 2011.

61. M. Casdagli, S. Eubank, J. D. Farmer, and J. Gibson, "State space reconstruction in the presence of noise," *Physica D: Nonlinear Phenomena*, vol. 51, no. 1-3, pp. 52–98, 1991.

62. A. Tsonis, J. Elsner, and K. Georgakakos, "Estimating the dimension of weather and climate attractors: important issues about the procedure and interpretation," *Journal of the atmospheric sciences*, vol. 50, no. 15, pp. 2549–2555, 1993.

63. P. GRASSBERGER, T. SCHREIBER, and C. SCHAFFRATH, "Nonlinear time sequence analysis," *International Journal of Bifurcation and Chaos*, vol. 01, no. 03, pp. 521–547, 1991.

64. S. Tsuji and K. Aihara, "Criterion for determining the optimal delay of attractor reconstruction using persistent homology," *Nonlinear Theory and Its Applications, IEICE*, vol. 10, no. 1, pp. 74–89, 2019.

65. S. H. Strogatz, *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*. New York: Addison-Wesley, 1994.

66. J. B. Dingwell, "Lyapunov exponents," *Wiley encyclopedia of biomedical engineering*, 2006.

67. M. Kumar, R. B. Pachori, and U. R. Acharya, "Automated diagnosis of myocardial infarction ecg signals using sample entropy in flexible analytic wavelet transform framework," *Entropy*, vol. 19, no. 9, p. 488, 2017.

68. W. Zhang, Y. Du, T. Yoshida, Q. Wang, and X. Li, "Samen-svr: using sample entropy and support vector regression for bug number prediction," *IET Software*, vol. 12, no. 3, pp. 183–189, 2018.

69. J. Theiler, "Estimating fractal dimension," *JOSA A*, vol. 7, no. 6, pp. 1055–1073, 1990.

70. J. C. Sprott and G. Rowlands, "Improved correlation dimension calculation," *International Journal of Bifurcation and Chaos*, vol. 11, no. 07, pp. 1865–1880, 2001.

71. C.-K. Peng, S. V. Buldyrev, S. Havlin, M. Simons, H. E. Stanley, and A. L. Goldberger, "Mosaic organization of dna nucleotides," *Physical review e*, vol. 49, no. 2, p. 1685, 1994.

72. R. Hardstone, S.-S. Poil, G. Schiavone, R. Jansen, V. Nikulin, H. Mansvelder, and K. Linkenkaer-Hansen, "Detrended fluctuation analysis: A scale-free view on neuronal oscillations," *Frontiers in Physiology*, vol. 3, p. 450, 2012.

73. T. W. Liao, "Clustering of time series data—a survey," *Pattern recognition*, vol. 38, no. 11, pp. 1857–1874, 2005.

74. C. Tang and C. Monteleoni, "On lloyd's algorithm: New theoretical insights for clustering in practice," in *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics* (A. Gretton and C. C. Robert, eds.), vol. 51 of *Proceedings of Machine Learning Research*, (Cadiz, Spain), pp. 1280–1289, PMLR, 09–11 May 2016.

75. D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series.," in *KDD workshop*, vol. 10, pp. 359–370, Seattle, WA, USA:, 1994.

76. M. Cuturi, J.-P. Vert, Ø. Birkenes, and T. Matsui, "A kernel for time series based on global alignments," *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*, vol. 2, pp. II–413–II–416, 2007.

77. Y. Bengio, A. C. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, pp. 1798–1828, 2013.

78. M. R. Karim, O. Beyan, A. Zappa, I. G. Costa, D. Rebholz-Schuhmann, M. Cochez, and S. Decker, "Deep learning-based clustering approaches for bioinformatics," *Briefings in bioinformatics*, vol. 22, no. 1, pp. 393–415, 2021.

79. E. Min, X. Guo, Q. Liu, G. Zhang, J. Cui, and J. Long, "A survey of clustering with deep learning: From the perspective of network architecture," *IEEE Access*, vol. 6, pp. 39501–39514, 2018.

80. P. Huang, Y. Huang, W. Wang, and L. Wang, "Deep embedding network for clustering," in *2014 22nd International Conference on Pattern Recognition*, pp. 1532–1537, 2014.

81. B. Yang, X. Fu, N. D. Sidiropoulos, and M. Hong, "Towards k-means-friendly spaces: Simultaneous deep learning and clustering," in *international conference on machine learning*, pp. 3861–3870, PMLR, 2017.

82. D. Sculley, "Web-scale k-means clustering," in *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, (New York, NY, USA), p. 1177–1178, Association for Computing Machinery, 2010.

83. Q. Ma, J. Zheng, S. Li, and G. W. Cottrell, "Learning representations for time series clustering," *Advances in neural information processing systems*, vol. 32, pp. 3781–3791, 2019.

84. H. Zha, X. He, C. Ding, M. Gu, and H. D. Simon, "Spectral relaxation for k-means clustering," in *Advances in neural information processing systems*, pp. 1057–1064, 2001.

85. D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," tech. rep., Stanford, 2006.

86. P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

87. J. Xie, R. Girshick, and A. Farhadi, "Unsupervised deep embedding for clustering analysis," in *International conference on machine learning*, pp. 478–487, PMLR, 2016.

88. C. Schölzel, "Nonlinear measures for dynamical systems," June 2019.

89. S. Bianchi, "fathon: A python package for a fast computation of detrendend fluctuation analysis and related algorithms," *Journal of Open Source Software*, vol. 5, no. 45, p. 1828, 2020.

90. M. Maghoumi, *Deep Recurrent Networks for Gesture Recognition and Synthesis*. PhD thesis, University of Central Florida Orlando, Florida, 2020.

91. M. Maghoumi, E. M. T. II, and J. J. L. Jr, "DeepNAG: Deep Non-Adversarial Gesture Generation," 2020.

92. R. Tavenard, J. Faouzi, G. Vandewiele, F. Divo, G. Androz, C. Holtz, M. Payne, R. Yurchak, M. Rußwurm, K. Kolar, and E. Woods, "Tslearn, a machine learning toolkit for time series data," *Journal of Machine Learning Research*, vol. 21, no. 118, pp. 1–6, 2020.

93. United States Geological Survey USGS, "Geographic face of the nation elevation." https://store.usgs.gov/product/114587.

94. United States Geological Survey USGS, "National geographic map - scale not given." https://www.arcgis.com/home/webmap/viewer.html?featurecollection=https%3A%2F%2Fbasemap.nationalmap.gov%2Farcgis%2Frest%2Fservices%2FUSGSShadedReliefOnly%2FMapServer%3Ff%3Djson%26option%3Dfootprints&supportsProjection=true&supportsJSONP=true.

95. USDA National Agricultural Statistics Service, "Quick stats." https://www.ers.usda.gov/data-products/chart-gallery/gallery/chart-detail/?chartId=58320.

96. M. Kottek, J. Grieser, C. Beck, B. Rudolf, and F. Rubel, "World map of the köppen-geiger climate classification updated," *Meteorologische Zeitschrift*, vol. 15, pp. 259–263, 05 2006.

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 0704–0188*



| 1. REPORT DATE *(DD–MM–YYYY)* | 2. REPORT TYPE | 3. DATES COVERED *(From — To)* |
|---|---|---|
| 27–09–2021 | PhD Dissertation | Sept 2018 — Sept 2021 |

**4. TITLE AND SUBTITLE**

Deep Learning for Weather Clustering and Forecasting

**5a. CONTRACT NUMBER**

**5b. GRANT NUMBER**

**5c. PROGRAM ELEMENT NUMBER**

**6. AUTHOR(S)**

Beveridge, Nathanael R., Capt, USAF

**5d. PROJECT NUMBER**

**5e. TASK NUMBER**

**5f. WORK UNIT NUMBER**

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

**8. PERFORMING ORGANIZATION REPORT NUMBER**

AFIT-ENS-DS-21-037

**9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)**

HAF/A3WR
Mr. Fred Fahlbusch
1480 Air Force Pentagon
Washington, DC 20330
571 256-8087

**10. SPONSOR/MONITOR'S ACRONYM(S)**

**11. SPONSOR/MONITOR'S REPORT NUMBER(S)**

**12. DISTRIBUTION / AVAILABILITY STATEMENT**

Distribution Statement A:
Approved for Public Release; Distribution unlimited.

**13. SUPPLEMENTARY NOTES**



**14. ABSTRACT**

Clustering weather data is a valuable endeavor in multiple respects. The results can be used within a larger weather prediction framework or could simply serve as an analytical tool for characterizing climatic differences of a particular region. This research proposes a methodology for clustering geographic locations based on the similarity in shape of their temperature time series. To this end an emerging and powerful class of clustering techniques that leverages deep learning, called deep representation clustering (DRC), are utilized. Moreover, a time series specific DRC algorithm is proposed that addresses a current gap in the field. Finally, deep learning based weather prediction is an increasingly common research topic as a means of obtaining more rapid predictions when compared to traditional numerical weather prediction (NWP). Since their are known physical equations that govern atmospheric behavior, namely the Navier-Stokes equations, the concept of reformulating these laws into a physics based loss function is explored with particular interest in whether a model trained with such a loss function can outperform it's baseline counterpart.

**15. SUBJECT TERMS**

deep learning, clustering, physics, dynamics, weather prediction

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT | b. ABSTRACT | c. THIS PAGE | | | Dr. Raymond R. Hill, AFIT/ENS |
| U | U | U | UU | 147 | 19b. TELEPHONE NUMBER *(include area code)* (937) 255-6565, x7469; raymond.hill@afit.edu |