Politecnico di Torino

Ingegneria Informatica (Computer Engineering)
A.Y. 2024/2025
Graduation Session July 2025

Artificial intelligence for climate teleconnections: a graph neural network approach with interpretable attention mechanisms

Supervisors:
Daniele Apiletti
Simone Monaco

Candidate:
Marco De Luca

Abstract

Climate teleconnections are long-range relationships among climate phenomena that underpin the foundation for explaining the climate dynamics of the planet and forecasting extreme events. Although there has been remarkable progress in climate science, many teleconnections remain inadequately characterized or even unidentified, partly due to the limitations of conventional statistical methods that cannot capture complex spatio-temporal dependencies and causality. A machine learning model can address these limitations. One approach to representing spatio-temporal relationships in climate data is through graphs, where nodes are points in space and time projected onto a spherical approximation of Earth, and edges connect each node at a specific time step to its neighbors in the subsequent time step. In this way, a model is directed to learn causal directions of influence. Graph neural networks (GNNs), especially graph attention networks (GATs), can extract causal patterns in the graphs through message-passing mechanisms. However, we need a reliable approach to assess the significance of those patterns. A straightforward solution is to assume that attention weights computed by the model yield estimates of causal strength. However, teleconnection analysis lacks ground truth data, making it difficult to determine whether model choices are random or reflect genuine causal relationships. Therefore, this represents an unsupervised task that requires modeling the probability of inference correctness. To guide the neural network in learning significant graph relationships, we employ a GAT as the encoder in a variational graph autoencoder (VGAE) architecture, with the training objective of learning meaningful data representations while minimizing the entropy of attention weight distributions. If the model chooses a same long-range connection at different scales of input data, then we can assume this is a real teleconnection pattern. Results show that the model can potentially detect known teleconnection patterns on ERA5 reanalysis dataset. However, current limitations - including low spatial resolution, neglect of inter-variable relationships, and absence of appropriate causality tests - indicate clear directions for future work in discovering new teleconnections.

Keywords: climate teleconnections, graph neural networks, causal inference

Table of Contents

List of Tables

List of Figures

Glossary

3D-Var	3D Variational Data Assimilation
4D-Var	4D Variational Data Assimilation
Adam	Adaptive Moment Estimation
AI	Artificial Intelligence
ANN	Artificial Neural Network
ANOVA	ANalysis Of VAriance
AO	Arctic Oscillation
AP	Average Precision
ARIMA	AutoRegressive Integrated Moving Average
AUC	Area Under the Curve
CDR	Climate Data Record
CDS	Copernicus Data Store
CNN	Convolutional Neural Network
DMD	Dynamic Mode Decomposition
DOE	Department of Energy
EBM	Energy Balance Model
ECMWF	European Centre for Medium-Range Weather Forecasts
ELBO	Evidence Lower Bound
EMIC	Earth System Model of Intermediate Complexity
ENSO	El Niño Southern Oscillation
EOF	Empirical Orthogonal Function
ERA	European Reanalysis
ESM	Earth System Model
FPR	False Positive Rate
GAE	Graph Autoencoder

GAT	Graph Attention Network
GCM	General Circulation Model
GD	Gradient Descent
GMAO	Global Modeling and Assimilation Office
GNN	Graph Neural Network
IoU	Intersection over Union
JMA	Japan Meteorological Agency
JRA	Japanese Reanalysis
KL	Kullback-Leibler divergence
LIDAR	Light Detection and Ranging
LOESS	LOcally Estimated Scatterplot Smoothing
MCMC	Markov Chain Monte Carlo
MERRA	Modern-Era Retrospective Analysis for Research and Applications
MJO	Madden-Julian Oscillation
MK	Mann-Kendall trend test
ML	Machine Learning
MLP	Multilayer Perceptron
MME	Multi-Model Ensemble
MPNN	Message Passing Neural Network
MSE	Mean Squared Error
MSSA	Multivariate Singular Spectrum Analysis
NAO	North Atlantic Oscillation
NASA	National Aeronautics and Space Administration
NCAR	National Center for Atmospheric Research
NCEI	National Centers for Environmental Information
NCEP	National Centers for Environmental Prediction
NetCDF	Network Common Data Form
NOAA	National Oceanic and Atmospheric Administration
ONI	Oceanic Niño Index

PCA	Principal Component Analysis
PDO	Pacific Decadal Oscillation
RadCM	Radiative-Convective Model
RCM	Regional Climate Model
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
SAR	Synthetic Aperture Radar
SARIMA	Seasonal AutoRegressive Integrated Moving Average
SDG	Sustainable Development Goal
SGD	Stochastic Gradient Descent
SK	Seasonal Kendall trend test
SSA	Singular Spectrum Analysis
SST	Sea Surface Temperature
STL	Seasonal-Trend decomposition using LOESS
SVD	Singular Value Decomposition
tanh	Hyperbolic Tangent
TPR	True Positive Rate
VAE	Variational Autoencoder
VGAE	Variational Graph Autoencoder
XAI	Explainable Artificial Intelligence

Chapter 1

Introduction

1.1 Background

1.1.1 Teleconnections and climate challenges

The term *teleconnection* originates from the combination of two components: *tele-*, a Greek-derived prefix meaning *distant* or *far*, and *connection*, referring to a link or relationship between entities. Thus, *teleconnection* literally means a connection over a distance. In climatology, teleconnections are significant relationships or links between weather phenomena at widely separated locations on earth, which typically entail climate patterns that span thousands of miles. Many teleconnection patterns behave like a seesaw, with atmospheric mass/pressure shifting back and forth between two distant locations — an increase in, say, atmospheric pressure in one location results in a decrease in pressure somewhere far, far away [1].

Teleconnections are important because they highlight climate variability and anomaly conditions at a global scale, thus allowing to understand the interconnections between different regions and their climate systems. They can influence weather patterns, precipitation, and temperature anomalies across vast distances, affecting agriculture, water resources, and ecosystems. They can also play a role in understanding and predicting extreme weather events, such as droughts, floods, and hurricanes. They are often associated with atmospheric circulation patterns, such as the El Niño-Southern Oscillation (ENSO), the North Atlantic Oscillation (NAO), and the Arctic Oscillation (AO). By analyzing such systems, scientists gain insights into the underlying dynamics of climate variability, which is fundamental for developing predictive models. These models are particularly vital in the context of a changing climate, as they provide valuable tools for forecasting future trends, managing natural resources, and preparing for climate-related risks.

1.1.2 A tribute to SDG 13: Climate action

The United Nations (UN) Sustainable Development Goal 13 (SDG 13) aims to combat climate change and its impacts, recognizing the urgent need for global action to address this pressing issue. Climate change poses significant threats to ecosystems, human health, and economies worldwide, making it imperative to take immediate and effective measures to mitigate its effects. SDG 13 emphasizes the importance of reducing greenhouse gas emissions, enhancing resilience to climate-related hazards, and promoting sustainable practices across all sectors. It calls for increased investment in renewable energy, sustainable transportation, and climate-resilient infrastructure, as well as the integration of climate considerations into national policies and strategies. By fostering international cooperation and knowledge sharing, SDG 13 aims to build a more sustainable and resilient future for all, ensuring that future generations can thrive in a stable and healthy environment [2].

In this context, the study of teleconnections becomes particularly relevant. The ability to recognize and model distant climate relationships supports efforts to anticipate and respond to climate variability, enhancing preparedness for extreme events and long-term shifts. This thesis, by leveraging artificial intelligence (AI) to explore and interpret teleconnection patterns, contributes to the broader objectives of SDG 13. Through improved understanding of global climate interdependencies, AI-driven approaches can inform climate-resilient policies, optimize resource management, and ultimately foster more effective climate action—advancing the shared goal of a stable and sustainable future.

1.1.3 Climate indices

Climate indices are numerical values derived from climate data that summarize and quantify specific climate phenomena, patterns, cycles. They are essential tools for understanding and monitoring climate variability, trends, and anomalies [3]. There are different examples of climate teleconnections that can be explained through climate indices, and they can be classified into different categories based on their characteristics and the phenomena they represent [4].

For instance, atmospheric pressure-based climate indices include North Atlantic Oscillation (NAO), which is based on the surface sea-level pressure difference between the Subtropical (Azores) High and the Subpolar Low. Fluctuations of this difference lead to an atmospheric cycle characterized by positive and negative phases. The positive phase of the NAO reflects below-normal heights and pressure across the high latitudes of the North Atlantic and above-normal heights and pressure over the central North Atlantic, the eastern United States and western Europe. The negative phase reflects an opposite pattern of height and pressure anomalies over these regions. Both phases of the NAO are associated with basin-wide changes in the intensity and location of the North Atlantic jet stream and storm track, and in large-scale modulations of the normal patterns of zonal and meridional heat and moisture transport, which in turn results in changes in temperature and precipitation patterns often extending from eastern North America to western and central Europe. Both phases alternate regularly, but the climate index can be used to identify periods of persistent or strong positive or negative phases. Strong positive phases of the NAO tend to be associated with above-normal temperatures in the eastern United States and across northern Europe and below-normal temperatures in Greenland and oftentimes across southern Europe and the Middle East. They are also associated with above-normal precipitation over northern Europe and Scandinavia and below-normal precipitation over southern and central Europe. Opposite patterns of temperature and precipitation anomalies are typically observed during strong negative phases of the NAO. During particularly prolonged periods dominated by one particular phase of the NAO, abnormal height and temperature patterns are also often seen extending well into central Russia and north-central Siberia. The NAO exhibits considerable interseasonal and interannual variability, and prolonged periods (several months) of both positive and negative phases of the pattern are common [5]. Other atmospheric indices include Arctic Oscillation (AO), Pacific North America pattern (PNA), Eastern Pacific Oscillation (EPO), Western Pacific Oscillation (WPO), Scandinavian Pattern (SCAND), East Atlantic / Western Russia pattern (EAWR), Southern Oscillation Index (SOI), Antarctic Oscillation (AAO).

Sea surface temperature (SST) based climate indices include the El Niño-Southern Oscillation (ENSO), which is characterized by periodic fluctuations in sea surface temperatures and atmospheric conditions in the tropical Pacific Ocean. The ENSO has two main phases[1]: El Niño, characterized by warmer-than-average sea surface temperatures in the central and eastern tropical Pacific, and La Niña, characterized by cooler-than-average sea surface temperatures in the same region. These phases have significant impacts on global weather patterns, including changes in precipitation, temperature, and storm activity across various regions. The ENSO is often monitored using the Oceanic Niño Index (ONI), which is based on sea surface temperature anomalies in the central equatorial Pacific (Niño 3.4 region). Positive values of the ONI indicate El Niño conditions, while negative values indicate La Niña conditions. The ENSO is a key driver of interannual climate variability and has far-reaching effects on ecosystems, agriculture, and water resources worldwide [6]. Other sea surface temperature-based indices include Pacific Decadal Oscillation (PDO), Atlantic Multidecadal Oscillation (AMO), Indian Ocean Dipole (IOD),

[1]Actually, there is a third phase called *neutral* phase, which is characterized by near-average sea surface temperatures in the central and eastern tropical Pacific.

North Pacific Gyre Oscillation (NPGO), Tropical North Atlantic Index (TNA), Tropical Southern Atlantic Index (TSA).

Subseasonal climate indices include the Madden-Julian Oscillation (MJO), which is a tropical atmospheric phenomenon characterized by eastward-propagating disturbances in tropical convection and circulation. The MJO consists of two main phases: the active phase, characterized by enhanced convection and rainfall over the Indian Ocean and western Pacific, and the suppressed phase, characterized by reduced convection and drier conditions over the same regions. The MJO has significant impacts on global weather patterns, including changes in precipitation, temperature, and storm activity across various regions. It is often monitored using the Real-time Multivariate MJO Index (RMM), which combines information from multiple atmospheric variables to provide a comprehensive view of the MJO state. The MJO is a key driver of subseasonal climate variability and has far-reaching effects on ecosystems, agriculture, and water resources worldwide [7].

Climate indices can be used to quantify climate dipoles, which refer to patterns of climate variability characterized by opposing anomalies (e.g., temperature or precipitation) in two spatially distinct regions. However, the term *climate dipole* has a different meaning from *climate index*, because the former implies that there are necessarily two opposing regions with respect to a specific phenomenon, while the latter can also involve a single region.

1.2 Research question

1.2.1 Motivation and challenges

Understanding climate teleconnections remains one of the most pressing challenges in contemporary climate science. While well-documented patterns such as ENSO, NAO, and AO have provided valuable insights into global climate dynamics, the complexity of Earth's climate system suggests that numerous teleconnections remain undiscovered or inadequately characterized. This knowledge gap represents a significant limitation in our ability to predict climate variability and extreme events, particularly in the context of ongoing climate change. Traditional statistical approaches to teleconnection analysis face fundamental constraints when dealing with modern climate datasets: they assume linear relationships between climate variables and struggle to capture the non-linear dynamics inherent in atmospheric and oceanic systems. These approaches also lack the ability to establish causal directionality, making it difficult to distinguish between genuine physical connections and spurious statistical associations. The increasing availability of high-resolution, multi-dimensional climate datasets has exacerbated these methodological limitations. As the volume and complexity of climate data continue to grow, traditional statistical techniques become computationally prohibitive and fail to exploit the rich spatio-temporal information contained in modern climate observations. This creates a paradox where our access to climate data has never been greater, yet our ability to extract meaningful teleconnection patterns remains constrained by outdated analytical tools.

The emergence of machine learning presents unprecedented opportunities to overcome these limitations. Advanced neural network architectures have demonstrated remarkable success in capturing complex patterns in high-dimensional data across various scientific domains. However, the application of machine learning to teleconnection discovery faces unique challenges that distinguish it from typical pattern recognition tasks. The absence of labeled training data means that supervised learning approaches cannot be directly applied, requiring novel unsupervised methodologies that can identify significant patterns while quantifying their reliability. Furthermore, the physical nature of teleconnections imposes additional constraints on machine learning approaches. Any method for teleconnection discovery must respect the spherical geometry of Earth, account for the temporal evolution of climate patterns, and provide interpretable results that can inform physical understanding of climate processes. These requirements necessitate specialized architectures and training procedures that go beyond standard machine learning applications.

The potential impact of advancing teleconnection research extends far beyond academic interest. Improved understanding of global climate connections directly supports the objectives of SDG 13 by enhancing our ability to predict climate variability, assess climate risks, and develop adaptive strategies. Better teleconnection models could inform seasonal forecasting, improve extreme event prediction, and support climate-resilient planning across multiple sectors.

1.2.2 Research objectives

We therefore set the research question for this work as follows:

> *How can we leverage machine learning to discover and interpret teleconnections in high-dimensional, spatio-temporal climate datasets?*

To address this research question, the primary goal of the present work is to develop and test machine learning methods for finding and understanding climate teleconnections in observational climate data, addressing the problems of traditional statistical methods while providing reliable identification of important climate relationships. This research has several connected objectives that work together to advance climate teleconnection analysis. The first objective involves creating better ways to represent climate data that can capture the complex relationships in space and time within Earth's climate system. This includes building structural frameworks that keep the round geometry of global climate observations and handle changes over time, while allowing analysis of long-distance climate connections, without needing labeled training data. The data representation should also encourage causal relationships, enabling the identification of significant teleconnections in a way that is computationally efficient and interpretable. The second objective focuses on using advanced neural network methods to find patterns of climate connectivity from these data representations. This involves using attention-based mechanisms that can focus on relevant climate relationships across different spatial and temporal scales, allowing the extraction of meaningful teleconnection patterns from high-dimensional climate datasets while testing the framework's ability to find known teleconnections and exploring the potential for finding previously unknown or poorly understood climate relationships. The third objective centers on building strong methods for interpreting and checking the climate relationships identified by machine learning models. This includes creating quantitative measures for evaluating how significant discovered teleconnections are across different dataset resolutions and time periods, ensuring that identified teleconnections represent real physical connections rather than statistical artifacts.

This thesis aims to explore a scientific approach to address the described objectives, and it should be considered as a starting point for future research in this area. The methods and findings presented here are intended to provide a foundation for further exploration and refinement of machine learning techniques for teleconnection discovery, with the ultimate goal of enhancing our understanding of global climate dynamics.

1.3 Thesis structure

The thesis is structured as follows. Chapter 2 provides an review of relevant literature and existing methods. Chapter 3 outlines the methodology used in the study, including data collection, preprocessing, and analysis techniques. Chapter 4 presents the results of the research, including key findings and insights. Chapter 5 discusses the implications of the findings, addresses limitations, and suggests directions for future research. Finally, the thesis concludes with a summary of the main contributions.

Chapter 2

Literature review

In the following, we review the existing literature on climate teleconnections, focusing on the methods and approaches used in previous studies. We start considering the traditional statistical methods and how they are applied in climate science, then we explore data-driven decomposition methods, which are increasingly used to analyze complex climate data. Finally, we discuss the use of artificial intelligence and neural networks. In general, we highlight features, advantages and limitations of each approach, providing a comprehensive overview of the state of the art in climate teleconnection analysis.

2.1 Statistical methods for climate science

A traditional approach to climate science is based on statistical methods, which include regression analysis, time series analysis, and hypothesis testing [8]. These methods are based on mathematical models that describe the relationships between variables and can be used to identify trends, correlations, and anomalies in climate data. Regression analysis is used to model the relationship between a dependent variable and one or more independent variables [9].

2.1.1 Regression analysis

Regression analysis can be used to examine the impact of various factors on climate variables, such as temperature or precipitation. For example, multiple linear regression can be used to assess the influence of greenhouse gas concentrations on global temperatures, while logistic regression can be employed to analyze the likelihood of extreme weather events based on historical data. There are different kinds of regression analysis, and they are described in Appendix A.1.

Traditional regression analysis relies on several key assumptions that may be challenging to satisfy in climate science applications [10]. While simpler regression models assume data independence, this assumption is particularly difficult to achieve in climate data due to inherent temporal and spatial correlations in climate variables. All regression approaches, in different forms, assume some form of linearity: linear regression assumes a direct linear relationship between independent and dependent variables; polynomial regression accommodates non-linear relationships within a polynomial framework; logistic regression assumes linearity between log-odds and predictors [11]. Additionally, these methods typically require normally distributed errors with constant variance (homoscedasticity). In climate science, complex non-linear relationships between variables, presence of outliers, and non-normal distributions often violate these assumptions, potentially leading to biased estimates and incorrect conclusions [12]. While temporal and spatial correlations pose significant challenges for traditional approaches, modern methods such as Graph Neural Networks (GNNs) have emerged to explicitly address these spatial-temporal dependencies in climate data [13, 14]. These developments, alongside time series analysis techniques, provide alternative frameworks for analyzing climate data that can better accommodate the complex correlation structures inherent in environmental systems.

2.1.2 Time series analysis

Time series analysis involves analyzing data points collected or recorded at specific time intervals to identify trends, seasonal patterns, and cyclic behavior. Time series analysis can be used to forecast future values based on historical data, making it valuable for climate prediction. Common techniques include AutoRegressive Integrated Moving Average (ARIMA) models [15], Seasonal-Trend decomposition using LOESS (STL) [16], and exponential smoothing [17]. Please see Appendix A.2 for a detailed description of these techniques.

The problem of time series analysis is that it assumes stationarity, i.e. mean and variance constant over time, which is a property that allows for more reliable predictions. However, climate data is often non-stationary due to trends, seasonality, and other factors [18]. This non-stationarity can lead to biased estimates and inaccurate predictions if not properly addressed. To overcome this issue, researchers often apply transformations or differencing techniques to make the data stationary before applying time series models, but this operation can sometimes remove important information from the data. Moreover, classical time series analysis is not effective at handling non-linearity and complex interactions between variables, which are common in climate data. For example, teleconnections can involve multiple climate variables interacting in complex ways, making it difficult to isolate their individual effects. Additionally, time series analysis may struggle with high-dimensional data, where the number of variables exceeds the number of observations, leading to overfitting and unreliable predictions [19]. STL tries to overcome these limitations, and it is actually meant for handling non-stationary data, but it is still based on linear assumptions. It also assumes that the seasonal component is constant over time, which may not hold true in all cases. Moreover, STL relies on the choice of smoothing parameters, which can significantly impact the results because it can be challenging and may require trial and error or optimization techniques [16]. Additionally, STL may not perform well with short time series or when the seasonal patterns are weak or irregular. Therefore, researchers also consider the approach of hypothesis testing.

2.1.3 Hypothesis testing

Hypothesis testing is a statistical method used to determine whether there is enough evidence to support a specific hypothesis about a population based on sample data [20]. In climate science, hypothesis testing can be used to assess the significance of observed trends, correlations, or differences between groups. Different kinds of hypothesis tests are available, and two main approaches can be identified: frequentist and bayesian [21].

Frequentist hypothesis testing relies on the concept of p-values, which represent the probability of observing the data given that the null hypothesis is true. A low p-value (typically less than 0.05) indicates that the observed data is unlikely under the null hypothesis, leading to its rejection in favor of the alternative hypothesis. The frequentist approach assumes that the null hypothesis is true until proven otherwise and focuses on the long-run frequency of events. It does not incorporate prior knowledge or beliefs into the analysis, making it less flexible in certain situations. Frequentist methods can be parametric or non-parametric. Parametric tests assume that the data follows a specific distribution, usually normal distribution, they are suitable for continuous data and can be used to compare means or variances between groups, while non-parametric tests do not make such assumptions and are suitable for ordinal data or when the assumptions of parametric tests are not met. Non-parametric tests are generally more robust to outliers and skewed distributions, making them suitable for analyzing climate data, which can often exhibit non-normal characteristics. Parametric tests include t-tests [22], F-tests [23], ANOVA [23] and some correlation tests, like Pearson's correlation coefficient [24]. Non-parametric tests include Mann-Kendall (MK) [25], Sen's slope [26], Pettitt test [27] and some correlation tests, like Spearman's rank correlation coefficient [28]. See Appendix A.3.1 for details.

The problem of p-values is that they do not provide a direct measure of the strength of evidence against the null hypothesis, and they can be influenced by sample size, leading to false conclusions. This is the main limitation of frequentist hypothesis testing, which can result in Type I and Type II errors. Type I error occurs when the null hypothesis is incorrectly rejected, while Type II error occurs when the null hypothesis is incorrectly accepted. This limitation leads to consider the Bayesian approach.

Bayesian hypothesis testing incorporates prior knowledge and beliefs, such as expert opinions or historical data, into the analysis. It allows for a more flexible and intuitive interpretation of results, as it provides a posterior probability distribution for the parameters of interest. This enables researchers to quantify uncertainty and make probabilistic statements about the hypotheses. Bayesian hypothesis testing uses Bayes' theorem to update the prior probability of a hypothesis based on observed data. The posterior probability is calculated as:

$$P(H|D) = \frac{P(D|H)P(H)}{P(D)}$$

where $P(H|D)$ is the posterior probability of the hypothesis H given the data D, $P(D|H)$ is the likelihood of the data given the hypothesis, $P(H)$ is the prior probability of the hypothesis, and $P(D)$ is the marginal likelihood of the data. The posterior probability can be interpreted as the degree of belief in the hypothesis after observing the data. Bayesian hypothesis testing allows for a more nuanced understanding of uncertainty and can provide more reliable conclusions in complex situations. Bayesian methods can also be used to estimate parameters, make predictions, and assess model fit. For example, Bayesian regression can be used to model the relationship between climate variables while accounting for uncertainty in the parameter estimates. Bayesian methods can also incorporate prior information about the parameters, allowing for more robust predictions and better handling of small sample sizes or missing data. There are different kinds of Bayesian hypothesis tests, such as Bayes factor, Bayesian ANOVA, and Bayesian regression (please see Appendix A.3.2).

Here the advantage is also the main challenge: the choice of prior distributions can significantly influence the results and it can be subjective, so it can lead to biased conclusions if the priors are not carefully chosen or if they do not accurately reflect the underlying processes. Additionally, Bayesian methods can be computationally intensive, requiring specialized software or algorithms for implementation. This can make them less accessible to researchers who may not have the necessary expertise or resources. Furthermore, Bayesian methods may not always provide clear guidance on how to interpret the results, particularly when it comes to making decisions based on posterior probabilities.

2.1.4 Usage for climate teleconnections and limitations

Statistical methods described above have been widely used in climate science to analyze and model climate data, including teleconnections. For instance, multiple linear regression was employed by Rust et al. [29] to investigate the relationship between the indices of twelve atmospheric teleconnection patterns and temperature in Europe. Time series analysis, specifically ARIMAX (AutoRegressive Integrated Moving Average with eXogenous variables), a variation from standard ARIMA, was used by Rahman et al. [30] to improve rainfall predictions involving climate teleconnection indices as predictors. MK and Sen's slope tests were applied by Abdullahi et al. [31] to detect trends in hydro-meteorological data. We observed that all the statistical methods have important limitations, and additionally they also may not be able to handle missing data or outliers effectively, which can impact the accuracy of predictions. As a result, researches often employ statistical methods in conjuction with other techniques.

2.2 Data-driven decomposition methods for climate science

Data-driven decomposition methods are a class of techniques used to analyze complex datasets by breaking them down into simpler components. These methods are particularly useful in climate science, where data can be high-dimensional and exhibit non-linear relationships. Data-driven decomposition methods can help identify patterns, trends, and relationships in the data, making it easier to understand and model complex phenomena. Some common data-driven decomposition methods include empirical orthogonal functions (EOF), singular spectrum analysis (SSA) and dynamic mode decomposition (DMD).

2.2.1 Empirical orthogonal functions

Empirical orthogonal functions (EOFs) [32] are a statistical technique used to analyze spatial and temporal patterns in climate data. They decompose a dataset into orthogonal modes, which represent the dominant patterns of variability in the data. EOF analysis is particularly useful for identifying spatial patterns in climate variables, such as temperature or precipitation, and can help reveal underlying processes driving climate variability. See Appendix B.1 for a mathematical description of EOF analysis. Some experiments show the usage of EOF analysis to study teleconnections and other climate phenomena. For example, Huang et al. [33] employed EOF to investigate the spatial patterns of the North Atlantic Oscillation (NAO) and its relationship with temperature and precipitation in Europe. Although powerful, EOF analysis suffers from a variety of limitations, for instance temporal mixing: EOF modes often mix different temporal scales and physical processes, so a single EOF mode might capture both trends and oscillations, making physical interpretation difficult. EOF provides a snapshot view without explicitly capturing the temporal evolution of patterns. EOF modes are forced to be orthogonal, which can lead to artificial splitting of coherent patterns that are not naturally orthogonal in the data. EOF also assumes the covariance structure is stationary over time, which often is not realistic for climate and geophysical data. These limitations lead to explore alternative frameworks, like singular spectrum analysis.

2.2.2 Singular spectrum analysis

Singular spectrum analysis (SSA) [34] is a data-driven decomposition method used to analyze time series data by decomposing it into a sum of components, including trends, oscillations, and noise. SSA is particularly useful for identifying periodic patterns and trends in climate data, making it a valuable tool for studying teleconnections and other climate phenomena. A mathematical description of SSA is provided in Appendix B.2. In climate teleconnections analysis, SSA proves particularly powerful for identifying and characterizing the dominant modes of variability that link distant regions, so each component of the decomposition can correspond to different teleconnection patterns. SSA is able to extract quasi-periodic oscillations with time-varying amplitudes and frequencies, which are characteristic of climate teleconnections. Unlike traditional spectral analysis methods that assume stationarity, SSA can capture the non-stationary nature of climate oscillations, revealing how teleconnection patterns evolve over time. Moreover, SSA enables the identification of trend components that may be masked by high-frequency variability, providing insights into long-term climate changes. The method can separate externally forced trends from internal climate variability, helping researchers distinguish between anthropogenic climate change signals and natural climate fluctuations. SSA can be also applied to multiple climate time series simultaneously (MSSA), allows for the identification of common modes of variability across different locations or climate variables. This extension is particularly valuable for teleconnections analysis, as it can reveal spatially coherent patterns and their temporal evolution, providing a comprehensive view of how climate anomalies propagate across different regions and time scales. Groth and Ghil [35] applied both SSA and MSSA to identify weak oscillatory behavior in high-dimensional data and show shared mechanisms of variability between the Gulf Stream and the North Atlantic Oscillation in the interannual frequency band. Manta et al. [36] applied MSSA to the ERA5 data to discover the main oscillators, in particular the South Atlantic Dipole (SAD), which is probably related to PDO and ENSO. Like EOF, also SSA has important limitations. For example, the mathematical nature of SSA imposes a significant computational complexity, which grows with the length of time series. Although different from EOF, SSA still assumes the time series is approximately stationary over the analysis window, limiting its effectiveness for highly nonstationary or transient phenomena. The window length is an important hyperparameter that can significantly affect the results: too short misses important patterns, too long creates computational burden and can mix unrelated dynamics. SSA is also difficult to interpret, because the components extracted may not correspond to physically meaningful processes. A better option is to consider dynamic mode decomposition.

2.2.3 Dynamic mode decomposition

Dynamic mode decomposition (DMD) [37] is a data-driven technique for analyzing complex dynamical systems by extracting coherent spatial-temporal patterns from time-series data. Unlike traditional methods that separate spatial and temporal analysis, DMD simultaneously identifies spatial patterns and their associated temporal evolution, making it particularly valuable for understanding oscillatory phenomena in climate systems. DMD assumes that the dynamics of a system can be locally approximated by a linear operator, even when the underlying system exhibits nonlinear behavior. Mathematical details are available at Appendix B.3.

In climate teleconnections analysis, DMD offers several advantages over traditional methods. First, temporal coherence: unlike EOF analysis, DMD modes are coherent in time, meaning each spatial pattern evolves according to a single temporal frequency, making it ideal for identifying oscillatory climate patterns such as ENSO, NAO, and PDO. Second, frequency decomposition: the method naturally separates patterns based on their temporal frequencies, enabling the isolation of interannual, decadal, and multidecadal climate variability without prior assumptions about the underlying dynamics. Third, predictive capability: since DMD modes evolve according to known frequencies and growth rates, they can be extrapolated forward in time for short-term climate prediction, providing a data-driven forecasting framework. The linear superposition property allows the system state to be expressed as $\mathbf{x}(t) = \sum_{i=1}^{r} b_i \boldsymbol{\phi}_i e^{\omega_i t}$, where b_i are the mode amplitudes determined by initial conditions [38]. Ferré et al. [39] developed a non-stationary DMD that captures seasonal and ENSO-related modes in sea surface temperature data from 1990-2016, showing distinct *turn on/off* spatial amplitude patterns during El Niño and La Niña years. Despite its advantages, DMD has important limitations that must be considered in climate applications. The linear approximation assumption may not capture strongly nonlinear climate dynamics, particularly during extreme events or regime transitions, although this limitation is often acceptable for large-scale circulation patterns that exhibit quasi-linear behavior. The method requires sufficient temporal resolution and length to capture the frequencies of interest, typically necessitating monthly or seasonal data over several decades for meaningful climate analysis. DMD assumes that the underlying dynamics are stationary over the analysis period, which may be violated by climate change and natural variability in very long time series. The identification of physically meaningful modes versus numerical artifacts requires careful analysis of eigenvalue magnitude, spatial patterns, and temporal evolution, as spurious modes can arise from noise or insufficient data. Additionally, the method's performance can be sensitive to the choice of time lag and the number of snapshots, requiring careful parameter selection based on the specific climate phenomena under investigation. These limitations have led to the development of various extensions, including sparsity-promoting DMD for noise reduction, multi-resolution DMD for multiple time scales, and kernel DMD for handling nonlinear dynamics, making it an increasingly sophisticated tool for climate teleconnection analysis.

While data-driven decomposition methods represent a significant advancement over traditional statistical approaches, they still face fundamental limitations when dealing with the full complexity of climate systems. The linear assumptions underlying EOF analysis, the stationarity requirements of SSA, and the local linearity constraints of DMD all restrict their ability to capture the highly non-linear, multiscale, and interactive nature of climate teleconnections. Moreover, as climate datasets grow in size, resolution, and complexity—encompassing satellite observations, reanalysis products, and high-resolution model outputs—the need for more flexible and powerful analytical frameworks becomes increasingly apparent.

2.3 Machine learning methods for climate science

The above mentioned challenges have led to the adoption of machine learning (ML) methods in climate science, which offer several key advantages over traditional approaches. ML methods can capture complex non-linear relationships without requiring explicit mathematical formulation of these relationships; they can handle high-dimensional datasets where the number of variables exceeds the number of observations, a common scenario in modern climate science; many ML algorithms are designed to work with non-stationary data and can adapt to changing patterns over time. Finally, ML methods can automatically identify interactions between variables that might be missed by traditional approaches, potentially revealing new insights into climate teleconnections and their underlying mechanisms. Please see Appendix C for a detailed description on how AI and ML methods work.

2.3.1 Evolution of machine learning approaches over time

The evolution of machine learning approaches in climate teleconnection analysis can be traced through several key developments, beginning with early neural network applications in the late 1990s and progressing to sophisticated deep learning architectures in recent years.

Neural networks first entered climate science through Tangang et al. [40], who pioneered the integration of artificial neural networks with Extended EOF analysis for ENSO forecasting. Breaking away from conventional linear statistical approaches, this research harnessed the non-linear pattern recognition strengths of neural networks. Their hybrid framework initially employed EOF analysis to compress sea surface temperature data dimensionality, followed by a feed-forward neural network for predicting ENSO indices with 12-month lead times. This breakthrough demonstrated neural networks' ability to capture intricate non-linear ENSO dynamics, matching or exceeding the performance of established statistical techniques. Nevertheless, the research faced constraints including its narrow focus on a single teleconnection pattern, reliance on basic neural network architecture, and failure to address overfitting concerns typical in neural network implementations.

Expanding upon these foundational advances, Gershunov and Barnett [41] broadened neural network applications to California's regional precipitation forecasting, emphasizing long-range predictions tied to large-scale atmospheric circulation dynamics. Their research employed MLPs to forge connections between Pacific Ocean sea surface temperature anomalies and California precipitation patterns, extending forecast horizons to several months. The study's primary achievement lay in showcasing neural networks' capacity to translate global climate signals into actionable regional precipitation forecasts—a vital bridge toward practical teleconnection applications. Results indicated that neural networks consistently outperformed conventional regression techniques in modeling the non-linear ocean-precipitation relationships. Yet the investigation remained geographically constrained, offered limited exploration of alternative neural network designs, and lacked thorough validation using independent datasets.

A pivotal methodological leap emerged through Hsieh et al. [42], who leveraged neural network models to uncover nonlinear winter atmospheric teleconnection patterns linked to ENSO and AO. This research marked a fundamental shift toward applying neural networks for understanding intricate climate relationships beyond mere forecasting objectives. The investigators utilized neural network frameworks to detect nonlinear teleconnections across surface air temperature, precipitation, sea level pressure, and 500 hPa geopotential height, revealing quadratic relationships with ENSO and AO indices. Remarkably, these nonlinear teleconnections demonstrated enhanced perturbation propagation compared to their linear counterparts. The investigation's significance lies in providing tangible evidence that nonlinear methodologies could expose teleconnection patterns invisible to traditional approaches, specifically establishing that climate indices exhibit quadratic rather than purely linear relationships with atmospheric fields. Furthermore, the research enhanced understanding of climate influence spatial extent, revealing that nonlinear teleconnections propagate climate signals across greater distances than previously recognized through linear analysis. Despite these advances, the study remained confined to winter atmospheric patterns without exploring seasonal variations or temporal evolution of nonlinear relationships.

Neural network sophistication escalated during the early 2010s with Spellman

[43], who engineered specialized neural network models to examine large-scale circulation patterns' influence on regional temperature dynamics. This research advanced the field methodologically by implementing more refined neural network architectures and training protocols than predecessor studies. The investigators concentrated on deciphering how global circulation patterns, especially the NAO and PDO, shape regional temperature variations across diverse geographical areas. Their primary accomplishment involved establishing that neural networks could effectively model intricate, non-linear relationships between large-scale circulation indices and regional temperature patterns, yielding insights into climate variability mechanisms at regional scales. The research also enhanced comprehension of these relationships' temporal stability and seasonal fluctuations. Nonetheless, the study's scope remained restricted to temperature variables exclusively, without investigating potential interactions among multiple climate variables.

Parallel innovations in convolutional neural networks materialized through Hoyos et al. [44], who engineered technology for detecting climate patterns during flood events using GCM data with CNNs[1]. This research tackled the formidable challenge of linking global climate patterns to regional extreme events, with particular emphasis on flood occurrences. The researchers implemented CNN architectures to automatically recognize spatial patterns in climate model output associated with elevated flood risk, constituting a major leap in pattern recognition capabilities for climate science. Their central achievement involved proving that CNNs could effectively learn complex spatial relationships in climate data without manual feature engineering, resulting in enhanced identification of flood-prone climate states. The investigation also advanced understanding of how global climate patterns manifest regionally during extreme events. However, the research remained constrained by its specific focus on one extreme event type and did not investigate the temporal evolution of identified patterns.

Among the most recent breakthroughs, Papacharalampous et al. [46] introduced the modeling of spatial asymmetries in teleconnected extreme temperatures through advanced statistical and machine learning methodologies. This investigation represents current state-of-the-art understanding of complex spatial structures in climate teleconnections, particularly examining how extreme temperature events

[1]A CNN (Convolutional neural network) is a type of deep learning architecture specifically designed to process grid-like data, such as images or spatially distributed climate variables. It employs convolutional layers to automatically learn spatial hierarchies of features, making it particularly effective for tasks involving spatial patterns and relationships. The architecture typically consists of multiple convolutional layers followed by pooling layers, which reduce dimensionality while preserving important spatial information. The final layers are usually fully connected layers that output predictions based on the learned features. CNNs have been widely used in computer vision and image analysis, but they have also found applications in climate science, especially for tasks like pattern recognition and classification of climate phenomena [45].

display asymmetric patterns across different geographical regions. The researchers constructed sophisticated statistical models capable of capturing spatial heterogeneity in teleconnection responses, addressing a substantial limitation of earlier approaches that frequently assumed spatial homogeneity. Their fundamental contribution involved establishing that spatial asymmetries in teleconnection patterns constitute meaningful information about underlying physical processes governing climate variability rather than mere noise. The study further enhanced comprehension of how extreme events propagate through the climate system and exhibit location-dependent characteristics. Nevertheless, the investigation remained limited to temperature extremes without exploring potential applications to other climate variables, and lacked comprehensive causal inference methods to establish the directionality of identified teleconnection relationships.

Contemporary developments in the field have been characterized by adopting cutting-edge deep learning architectures, exemplified by Ham et al. [47], who pioneered Graph Neural Networks (GNNs) for enhanced El Niño forecasting. This investigation marked a conceptual transformation by treating climate data as graph-structured information, where spatial relationships between regions are explicitly modeled as graph edges. The researchers constructed a GNN-based framework capable of capturing both local and remote influences on ENSO evolution, yielding substantial improvements in forecast accuracy over traditional methods. Their essential contribution involved proving that explicit spatial relationship modeling through graph structures could enhance climate teleconnection representation and boost predictive capability. The GNN approach delivered state-of-the-art performance in ENSO forecasting with lead times reaching 18 months, constituting a remarkable advancement over previous methodologies. Yet the study concentrated primarily on ENSO prediction without exploring broader GNN method applicability to other teleconnection patterns. The research also lacked detailed interpretability analysis to identify which graph connections were most crucial for forecasting performance.

2.3.2 Challenges and opportunities

Throughout the described chronological progression, persistent limitations surface consistently across the literature. In general, the state of the art in machine learning for climate science is more focused on predicting weather and climate patterns rather than understanding the underlying physical mechanisms driving teleconnections. This means that most approaches assume the existence of some known climate indices, and base their observations on these indices, rather than exploring the data to discover new teleconnections. This can lead to a limited understanding of the complex interactions between different climate variables and their impact on teleconnections. Perhaps most critically, the overwhelming majority of investigations lack rigorous causal inference methodologies, depending instead on correlation-based approaches that cannot definitively establish climate relationship directionality. This constitutes a substantial gap in the field, as comprehending causal relationships proves essential for developing reliable climate prediction systems and understanding physical mechanisms underlying teleconnections. Furthermore, numerous studies suffer from inadequate statistical significance testing, especially regarding multiple comparisons and high-dimensional climate data.

Some attempts in literature have been made to address these limitations. For example, Kawale et al. [48] proposed a method for causal discovery in climate data using a combination of statistical tests and machine learning techniques, which can help identify causal relationships between climate variables. Their approach integrated Granger causality tests[2] with conditional independence testing to distinguish between spurious correlations and genuine causal relationships in high-dimensional climate datasets. The methodology employed a two-stage framework: first applying dimensionality reduction techniques to manage the curse of dimensionality inherent in climate data, followed by systematic causal inference testing that accounts for confounding variables and temporal dependencies. The authors demonstrated their approach on sea surface temperature and atmospheric pressure data, successfully identifying known causal pathways while filtering out spurious associations that traditional correlation-based methods would incorrectly classify as causal. This work represented an important step toward more rigorous causal analysis in climate science, though it remained limited by computational constraints when applied to very high-dimensional datasets and did not fully address the challenge of identifying time-varying causal relationships that may evolve seasonally or during extreme climate events.

The insufficient exploration of model interpretability represents another area demanding future attention. While the progression from basic neural networks to sophisticated deep learning architectures illustrates the field's growing capacity to capture complex non-linear relationships in climate data, this increased complexity has frequently compromised interpretability, creating a performance-understanding trade-off regarding underlying physical processes. The interpretability problem extends beyond mere academic curiosity, but it poses significant practical challenges for climate science applications. Climate scientists require understanding of not just what a model predicts, but why it makes specific predictions, which physical mechanisms it has learned to recognize, and whether its decision-making process aligns with established climate physics. Traditional "black box" approaches, while

[2]Granger causality tests are a statistical hypothesis test for determining whether one time series can predict another time series. In the context of climate data, this can help identify whether changes in one climate variable (e.g., sea surface temperature) precede changes in another variable (e.g., atmospheric pressure), suggesting a potential causal relationship [49].

potentially offering superior predictive performance, fail to provide the mechanistic insights necessary for advancing scientific understanding or building confidence in model predictions among domain experts and policymakers.

Recent developments in Explainable Artificial Intelligence (XAI) offer promising avenues for addressing these interpretability challenges in climate applications. XAI encompasses a suite of techniques designed to make machine learning models more transparent and interpretable, including attention mechanisms that highlight which input regions most influence predictions, gradient-based attribution methods that identify the importance of individual features, and layer-wise relevance propagation techniques that trace prediction pathways through neural network layers. In the climate domain, these approaches could potentially reveal which spatial patterns, temporal sequences, or physical variables drive teleconnection predictions, thereby bridging the gap between statistical performance and physical understanding [50]. Several studies have begun applying XAI techniques to climate problems. Attention-based neural networks have been employed to identify which geographical regions contribute most to climate predictions, while gradient attribution methods have been used to understand how seasonal cycles and interannual variability influence model decisions [51]. Shapley values and other game-theoretic approaches have shown promise in quantifying the contribution of different climate variables to extreme event predictions. However, these applications remain in their infancy, and significant challenges persist in adapting XAI methods to the unique characteristics of climate data, including its high dimensionality, complex spatio-temporal dependencies, and the need for interpretations that align with physical processes rather than purely statistical patterns [52, 53]. Recent comprehensive evaluation frameworks have been developed to guide the selection and ranking of appropriate XAI methods for specific climate science applications, addressing the challenge of choosing among the growing number of available techniques [51].

Based on the above discussion, an important research gap can be addressed: the concept of attention as a mechanism for explainable neural networks, integrated in the GNN architecture, with a careful arrangement of data structure to represent spatio-temporal relationships and recover causal patterns, and a proper interpretation of the results, can be a powerful tool for climate teleconnection analysis and discovery without any prior knowledge on existing teleconnections, and constitutes the foundation of the present work.

This concludes the discussion on the state of the art in climate teleconnection analysis, highlighting the evolution of methods from traditional statistical approaches to advanced machine learning techniques. In the next chapter, we will present the methodology employed in this research, which builds upon these advancements to address the identified limitations and explore new avenues for understanding climate teleconnections.

Chapter 3

Methodology

In this chapter, we describe the methodology used in this project, including the dataset choice, data processing, model architecture, training objectives, and evaluation metrics.

The first step in the project execution was to choose a dataset that is suitable for the purpose of the project, i.e. to analyze and discover climate teleconnections. Various sources provide climate data, including paleoclimate proxies, in-situ observations, remote-sensed data, climate models, reanalysis data [54]. In Appendix D, each of them is analyzed in detail. All them have their own strengths and weaknesses. Specifically, each of them is suitable to a specific research question: paleoclimate proxies are useful for long-term climate trends, in-situ observations are valuable for local and regional studies, satellite data provides a global perspective, climate models are essential for simulating complex interactions, and reanalysis datasets offer a comprehensive view of the climate system. Choosing a dataset is the basis for making any analysis, but then models and methods must be identified in order to correctly analyze the data. For the purpose of this work, we assume to use a reanalysis dataset, because we need full spatio-temporal information, gridded data and long-range temporal coverage. After choosing the dataset, the next step in the project execution was to analyze the dataset, to understand its structure and the relationships between the variables. This task has been accomplished by visualizing the data using various techniques, such as plotting the monthly averages of the variables. This allowed to identify patterns and trends in the data, as well as to detect any anomalies or missing values.

3.1 Graph construction

3.1.1 Coordinates encoding

After analyzing the dataset, a proper data structure was identified in order to represent the data in a way that is suitable for the neural network. We assume that the dataset is gridded, meaning that the data is projected on a grid that covers the entire globe, with each grid cell representing a specific geographical area. The dataset is also spatio-temporal, because for each spatial location there are multiple measurements taken at different times. First, some coordinates were defined, to determine the meaning of each data sample. Since the objective is to analyze climate teleconnections, which involves dealing with spatio-temporal information, coordinates were defined as latitude and longitude for the spatial dimension, and month and year for the temporal dimension. However, this information cannot be directly used as input for the neural network, because, except for the year that progresses linearly in the integer domain, the rest of the coordinates are defined into a finite or periodic range. Let $\mathcal{C}$ be the set of coordinates, then:

$$\mathcal{C} = \{(y, m, \ell, g) \mid y \in \mathbb{N},\, m \in [1, 12],\, \ell \in [-90°, +90°],\, g \in [-180°, +180°]\}$$

where y is the year, m is the month, ℓ is the latitude and g is the longitude. For what concerns years, we just normalize them considering the range of years in the dataset, namely $Y = y_1, y_1, \ldots, y_n$, where y_1 is the first year in the dataset and y_n is the last year in the dataset. The normalization is performed as follows:

$$\phi_y = \frac{y - y_1}{y_n - y_1}$$

This way, the year is mapped to a value in the range $[0, 1]$, which can be used as input for the neural network. For the rest, a proper way to handle information in m, ℓ, g is to use a cyclic encoding, which is not strictly necessary in the case of latitude, because it has a continuous domain, but it is useful to avoid discontinuities in the case of longitude, where $-180°$ and $+180°$ are the same point on the globe. The advantage of this kind of encoding is easy to visualize in the case of months, because January and December are adjacent in the month domain, but they are not adjacent in the integer domain. In general, a cyclic encoding can be defined as follows:

$$\psi(x, T) = \frac{x}{T} \cdot 2\pi$$

where x is the value to be encoded and T is the period of the cyclic domain. For months, we can use the following mapping:

$$\psi(m, 12) = \frac{m - 1}{12} \cdot 2\pi \in [0, 2\pi]$$

For latitude and longitude, we first need to convert them from degrees to radians, to map them to the ranges, respectively, $[-\pi/2, +\pi/2]$ for latitude and $[-\pi, +\pi]$ for longitude. In other words:

$$\begin{aligned}\psi(\ell, 180) &= \frac{\ell}{180} \cdot 2\pi \in \left[-\frac{\pi}{2}, +\frac{\pi}{2}\right] \\ \psi(g, 180) &= \frac{g}{180} \cdot 2\pi \in [-\pi, +\pi]\end{aligned}$$

Then, for more expressiveness of the coordinates, we can use a sine and cosine encoding, which allows to represent the cyclic nature of the coordinates in a more compact way. Therefore, the full mapping is the following:

$$\begin{aligned}\phi_m &= \left[\cos\left(\frac{m-1}{12} \cdot 2\pi\right), \sin\left(\frac{m-1}{12} \cdot 2\pi\right)\right] \\ \phi_\ell &= \left[\cos\left(\frac{\ell}{180} \cdot 2\pi\right), \sin\left(\frac{\ell}{180} \cdot 2\pi\right)\right] \\ \phi_g &= \left[\cos\left(\frac{g}{180} \cdot 2\pi\right), \sin\left(\frac{g}{180} \cdot 2\pi\right)\right]\end{aligned} \tag{3.1}$$

where ϕ is the mapping function that transforms the coordinates into a two-dimensional vector. Thus, a point in space and time is identified by the following vector:

$$\mathbf{c} = [\phi_y, \phi_m, \phi_\ell, \phi_g]$$

The point has associated a vector of features $\mathbf{f} = [f_1, f_2, \ldots, f_k]$, where f_i is the value of the i-th variable at that point in space and time, for a total of k variables, selected in the dataset analysis. Hence, any point is represented by the tuple:

$$\mathbf{p} = [\mathbf{c}, \mathbf{f}]$$

The outcome of this step is a mapping between a data entry $\mathbf{d}$ from the original dataset $\mathcal{D}$ and a point $\mathbf{p}$ in the new representation $\mathcal{P}$. We can think about $\mathcal{D}, \mathcal{P}$ as two different spaces, and:

$$\begin{aligned}\text{enc} : \quad & \mathcal{D} \to \mathcal{P} \\ & \mathbf{d} \mapsto \mathbf{p}\end{aligned}$$

as a function that maps a data entry from the original dataset to a point in the new representation. The mapping is one-to-one, meaning that each data entry corresponds to a unique point in the $\mathbf{p}$-space, and vice versa. Therefore, let $D = |\mathcal{D}|$ be the cardinality of the original dataset, and $P = |\mathcal{P}|$ be the cardinality of the $\mathbf{p}$-space, then we have $D = P$. In other words, we have a bijective mapping between the two spaces, which preserves the information contained in the original dataset.

3.1.2 Grid2mesh

The encoding process described before is a first step towards a data representation that takes into account the shape of the Earth. In fact, the Earth can be approximated as a sphere, but a gridded dataset collects measurements assuming a flat, rectangular Earth. As a result, points at a same time step but in different grid cells do not necessarily correspond to different locations on the globe: the identity holds for all locations at the equator, but as we get closer to the poles, the distance between two points at the same latitude but different longitudes decreases, until it becomes zero. A proof of this is shown in Appendix E.1. The grid distortion leads to an overrepresentation of information at the poles, and a proper data handling mechanism should take care of this. Therefore, we chose to adopt a *grid2mesh* strategy, which consists in mapping the gridded data onto a triangular mesh that approximates the spherical shape of the Earth. In general, a polygon mesh is a collection of vertices, sides, and faces that defines the shape of a 3D object in space. In the case of a triangular mesh, each face is a triangle defined by three vertices. A polygon mesh can approximate, up to a specified refinement level, complex shapes and curved surfaces more effectively than a regular grid, and triangular mesh is particularly suitable to the purpose of representing the Earth's surface. The refinement level is defined as the number of times the mesh is subdivided into smaller triangles, starting from a base shape, and can be interpreted as the resolution of the mesh. Refinement can be performed through different strategies, but for triangular meshes it is quite common to use the midpoint subdivision method, which consists in connecting the midpoints of each side of the triangles to create new vertices, and then forming new triangles with these vertices. Theoretically, the mesh could have higher resolution than the grid, but that would mean that the same feature values are replicated across multiple mesh vertices, because the mesh cannot extract more information than the grid already contains. In the following, we assume that the mesh resolution is always lower than or equal to the grid resolution. The simplest form of triangular mesh for a spherical surface is the icosahedron, which is a polyhedron with $F = 20$ triangular faces, $V = 12$ vertices and $E = 30$ sides. Assuming a unit sphere, i.e. a sphere with radius equal to 1, let:

$$\varphi = \frac{1+\sqrt{5}}{2} \approx 1.6180339887$$

be the golden ratio, i.e. the limit of the ratio of consecutive Fibonacci numbers[1]. Then, assuming cartesian coordinates (x, y, z), a common way to define the vertices of the icosahedron is the following:

$$\begin{array}{llll} V_1 = (-1, \varphi, 0) & V_2 = (1, \varphi, 0) & V_3 = (-1, -\varphi, 0) & V_4 = (1, -\varphi, 0) \\ V_5 = (0, -1, \varphi) & V_6 = (0, 1, \varphi) & V_7 = (0, -1, -\varphi) & V_8 = (0, 1, -\varphi) \\ V_9 = (\varphi, 0, -1) & V_{10} = (\varphi, 0, 1) & V_{11} = (-\varphi, 0, -1) & V_{12} = (-\varphi, 0, 1) \end{array}$$

Each face can be represented as a triplet of vertices:

$$\begin{array}{llll} F_1 = (V_1, V_{12}, V_6) & F_2 = (V_1, V_6, V_2) & F_3 = (V_1, V_2, V_8) & F_4 = (V_1, V_8, V_{11}) \\ F_5 = (V_1, V_{11}, V_{12}) & F_6 = (V_2, V_6, V_{10}) & F_7 = (V_6, V_{12}, V_5) & F_8 = (V_{12}, V_{11}, V_3) \\ F_9 = (V_{11}, V_8, V_7) & F_{10} = (V_8, V_2, V_9) & F_{11} = (V_4, V_{10}, V_5) & F_{12} = (V_4, V_5, V_3) \\ F_{13} = (V_4, V_3, V_7) & F_{14} = (V_4, V_7, V_9) & F_{15} = (V_4, V_9, V_{10}) & F_{16} = (V_5, V_{10}, V_6) \\ F_{17} = (V_3, V_5, V_{12}) & F_{18} = (V_7, V_3, V_{11}) & F_{19} = (V_9, V_7, V_8) & F_{20} = (V_{10}, V_9, V_2) \end{array}$$

This structure corresponds to a mesh without any refinement, i.e. the refinement level is 0. To get to refinement level 1, each triangle is subdivided into smaller triangles, by connecting the midpoints of each side of the icosahedron, and so on. Figure 3.1 shows this process. Starting from this mesh, we need to map grid points to mesh points. Latitude ℓ and longitude g are geographical coordinates that inherently hold a spherical nature, in fact they are angular distances respectively from the equator and the prime meridian. However, they need to be converted to cartesian coordinates to be represented on a mesh. The conversion is performed as follows:

$$\begin{aligned} x_\gamma &= \cos(\ell) \cdot \cos(g) \\ y_\gamma &= \cos(\ell) \cdot \sin(g) \\ z_\gamma &= \sin(\ell) \end{aligned}$$

[1]The Fibonacci sequence is defined as:

$$F(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F(n-1) + F(n-2) & \text{if } n \geq 2 \end{cases}$$

where $F(n)$ is the n-th Fibonacci number.
The golden ratio is defined as:

$$\varphi = \lim_{n \to \infty} \frac{F(n)}{F(n-1)} = \frac{1+\sqrt{5}}{2}$$

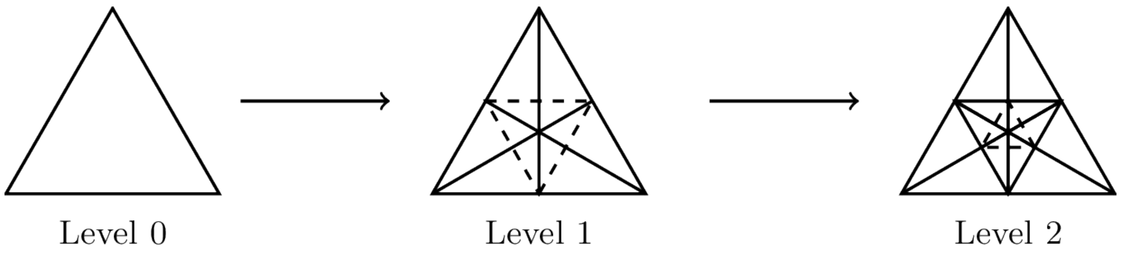


Figure 3.1: Mesh refinement process: starting from a single triangle (Level 0), each triangle is subdivided into four smaller triangles by connecting the midpoints of its sides (Level 1). The process can be repeated recursively for higher refinement levels (Level 2).

where $(x_\gamma, y_\gamma, z_\gamma)$ are the cartesian coordinates of the grid point on the sphere corresponding to the geographical coordinates (ℓ, g), expressed in radians. From Equation 3.1, we know a sine-cosine encoding of latitude and longitude, so we can use the following mapping:

$$\begin{aligned} x_\gamma &= \phi_\ell[1] \cdot \phi_g[1] \\ y_\gamma &= \phi_\ell[1] \cdot \phi_g[2] \\ z_\gamma &= \phi_\ell[2] \end{aligned}$$

where $\phi_k[i]$ is the i-th component of the encoding of the coordinate k. Unless lucky cases, the grid point does not correspond to any vertex of the mesh, so we need to find the closest vertex to the grid point, and this implies computing a distance metric. Let $\mathbf{g} = (x_\gamma, y_\gamma, z_\gamma)$ be the grid point, and $\mathbf{m}_i = (x_i, y_i, z_i)$ be the i-th vertex of the mesh. One could think to use the Euclidean distance:

$$d(\mathbf{g}, \mathbf{m}_i) = \sqrt{(x_\gamma - x_i)^2 + (y_\gamma - y_i)^2 + (z_\gamma - z_i)^2}$$

where V_i is the i-th vertex of the mesh, and (x_i, y_i, z_i) are its cartesian coordinates. However, for the spherical assumption, this is wrong, and we need an angular distance metric, rather than a linear, plain one. The angular distance can be computed using:

$$d(\mathbf{g}, \mathbf{m}_i) = \arccos\left(\frac{\mathbf{g} \cdot \mathbf{m}_i}{\|\mathbf{g}\| \cdot \|\mathbf{m}_i\|}\right) \tag{3.2}$$

where $\mathbf{g} \cdot \mathbf{m}_i$ is the dot product between the two vectors, and $\|\mathbf{g}\|$ and $\|\mathbf{m}_i\|$ are their respective norms. The norm is necessary because the dot product is not normalized, so it can lead to values outside the range $[-1, 1]$, on which the arccos function is defined. The closest vertex is the one with the minimum distance:

$$\mathbf{m}_\mathbf{g} = \arg\min_{\mathbf{m}_i} d(\mathbf{g}, \mathbf{m}_i)$$

Using the logic of *the closest one*, it is possible, and it is more likely as the mesh refinement level is lower, that two or more grid points are mapped to the same mesh vertex. One could think to just average pooling the features over the grid points that are mapped to the same vertex, i.e.:

$$\mathbf{f}_k = \frac{1}{|\mathcal{G}_k|} \sum_{\mathbf{g} \in \mathcal{G}_k} \mathbf{f}_\mathbf{g}$$

where $\mathcal{G}_k$ is the set of grid points mapped to the vertex $\mathbf{m}_k$, and $\mathbf{f}_\mathbf{g}$ is the vector of features associated to the grid point $\mathbf{g}$. However, this does not solve the overrepresentation at the poles, because in the original grid each cell has the same size, but the spherical area of a cell near the poles is much less than near the equator. To avoid this, we can use an area-weighted average pooling, where the weights are proportional to the actual area of the grid cells on the sphere:

$$\mathbf{f}_k \approx \frac{\sum_{\mathbf{g}\in\mathcal{G}_k} \cos(\ell_{\mathbf{g}}) \cdot \mathbf{f}_{\mathbf{g}}}{\sum_{\mathbf{g}\in\mathcal{G}_k} \cos(\ell_{\mathbf{g}})}$$

A proof of this approximation is provided in Appendix E.2. This means that we can approximate the area-weighted average pooling with a cosine-weighted average pooling, where the weights are proportional to the cosine of the latitude of the grid cell. This approach is then applied to both spatio-temporal information and climatology, in order to extract, from the mesh representation, feature anomalies, that are computed as the difference between the feature value at a given point and the climatology value at that point. Climatology was obtained by averaging the values of each variable over a predefined time period, resulting in a single value for each grid cell and for each month, regardless of the year. Let $\mathbf{f}_{t,s}$ be the feature vector at a given point in space and time, and $\mathbf{f}^{\text{clim}}_{m_t,s}$ be the climatology feature vector at that spatial location for the month of the year m_t associated to time step t, then the anomaly is defined as:

$$\mathbf{f}^{\text{anomaly}}_{t,s} = \mathbf{f}_{t,s} - \mathbf{f}^{\text{clim}}_{m_t,s}$$

In the following, anomalies are used in place of the original features, because they are more informative and allow to highlight the deviations from the climatology.

This step, as anticipated, does not preserve the cardinality of the original dataset, because an aggregation is performed over the grid points mapped to the same mesh vertex. Therefore, the mapping is not one-to-one anymore, but it is many-to-one. Let $\mathcal{M}$ be the space of the mesh representation and $\mathbf{m}$ a mesh point. The following function:

$$\begin{aligned} \text{grid2mesh} : \quad & \mathcal{P} \to \mathcal{M} \\ & \mathbf{p} \mapsto \mathbf{m} \end{aligned}$$

maps each grid point to a mesh vertex. Let $M = |\mathcal{M}|$ be the cardinality of the mesh representation, then we have $P \geq M$. In other words, the mapping is not injective anymore, and it is not a bijection anymore, because the information is virtually lost in the aggregation process. We can say that:

$$M = \frac{1}{\alpha} P$$

where $\alpha \geq 1$ is the average number of grid points mapped to each mesh vertex, and depends on the mesh refinement level, the grid resolution and distance metrics. This average can be expressed as function of the latitude:

$$\alpha(\ell) = \frac{2\bar{\alpha}}{\pi \cdot \cos(\ell)}$$

where $\bar{\alpha}$ is the mean value of the integral over the surface of the Earth, described in Appendix E.3. At the equator, $\ell = 0$, so:

$$\alpha(0) = \frac{2\bar{\alpha}}{\pi}$$

At the poles, $\ell = \pm\frac{\pi}{2}$, so:

$$\lim_{\ell \to \pm\frac{\pi}{2}} \alpha(\ell) = \lim_{\ell \to \pm\frac{\pi}{2}} \frac{2\bar{\alpha}}{\pi \cdot \cos(\ell)} = +\infty$$

This means that, if the grid resolution is high enough, the average number of grid points mapped to each mesh vertex is very high at the poles, and very low at the equator.

3.1.3 Time windowing

Now that we have a mesh representation of the data, we can construct a graph from it. A graph is defined as a set of nodes and edges, where each node represents a point in space and time, and each edge represents a relationship between two nodes. In our case, the nodes are built using the encoding of year and month as temporal coordinates, the mesh vertices as spatial coordinates and the feature anomalies as features. In other words, each node is defined as:

$$\mathbf{v}_i = \left[\phi_y, \phi_m, \mathbf{m}_i, \mathbf{f}_i^{\text{anomaly}}\right]$$

We now introduce this notation[2]:

$$\begin{aligned}
\mathbf{v}_{t,s} &= [\mathbf{c}_{t,s}, \mathbf{f}_{t,s}] \\
\mathbf{c}_{t,s} &= [\phi_{y_t}, \phi_{m_t}, \mathbf{m}_s] \\
\mathbf{f}_{t,s} &= \mathbf{f}_{t,s}^{\text{anomaly}}
\end{aligned}$$

where t is the time step, s is the spatial location, ϕ_{y_t} is the encoding of the year at time step t, ϕ_{m_t} is the encoding of the month at time step t, and $\mathbf{m}_s$ is the mesh vertex corresponding to the spatial location s. Making explicit the dependence on time and space is necessary because we want a graph representation that captures both dimensions. In fact, considering the broader objective to analyze climate teleconnections, we need to account for variation of features over time, and above

[2]Beware that $\mathbf{f}_{t,s}$ is actually a redefinition, because it was already used to denote the feature vector at a given point in space and time, but here it is used to denote the anomaly feature vector.

all the neural network architecture must be able to learn these variations effectively. From the point of view of graph construction, there are several ways to represent data, by creating a graph where nodes represent the measurements collected at either:

(s1) one spatial location

(s2) multiple spatial locations but not all together

(s3) all spatial locations

for the spatial setting, and at either:

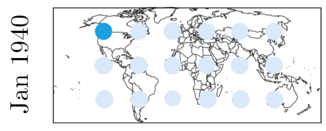
(t1) one time step

(t2) multiple time steps but not all together

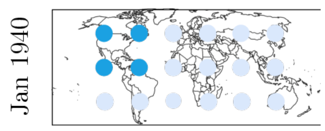
(t3) all time steps

In Figure 3.2, a graphical example of all the 9 possible configurations is provided. The nodes are represented as circles on the planet map for simplicity, but from Section 3.1.2 we know that they are actually mesh vertices, and therefore they are uniformly distributed on the sphere but not on a planar projection. Each node incorporates measurements collected at a specific point in space and time. Shown edges are just a subset of the possible ones, to highlight the different configurations, and here they are meant exclusively as connecting spatial locations at consecutive time steps, but it is not the only possible choice. Each map is associated with a point in time, whose unit of measure, in our convention, is the month of the year. Therefore, a graph should be intended as made up of one map, or more maps connected each other, depending on the configuration. From this list, we can actually exclude the cases whose spatial setting is (s1), because the resulting graph would not capture the spatial relationships between different locations. Similarly, we can also filter out the cases whose temporal setting is (t1), because they do not account for temporal relationships.

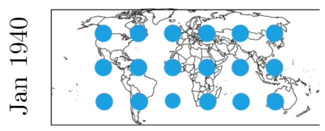
For the moment, we focus on the choice of the better configuration with respect to the spatial setting. (s2) can be a good approach if the network is able to keep track of the spatial views observed for each input data. However, these views must be carefully designed. In fact, if each spatial view is obtained by selecting near locations on the Earth, the result is a graph that does not capture the global relationships between distant locations, so the neural network should combine information from different spatial views to have a comprehensive understanding of the data: for each spatial view, the network can elaborate a meaningful representation of that portion of the Earth, and then connect representations belonging to different spatial views to capture the global relationships. This approach is reasonable if the spatial



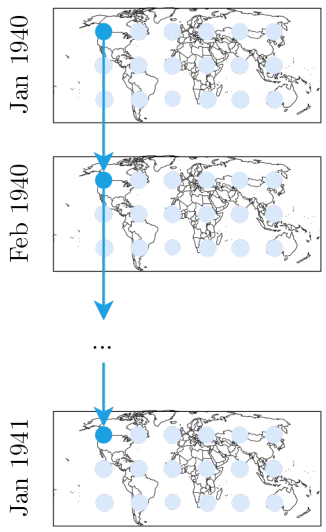
(a) (s1, t1)



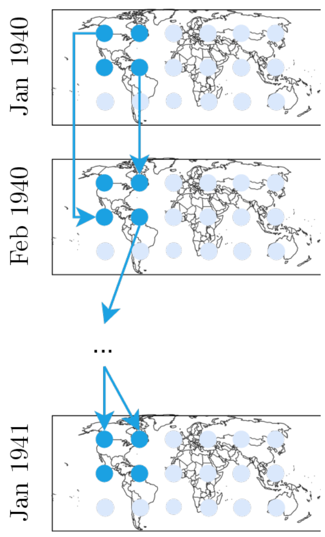
(b) (s2, t1)



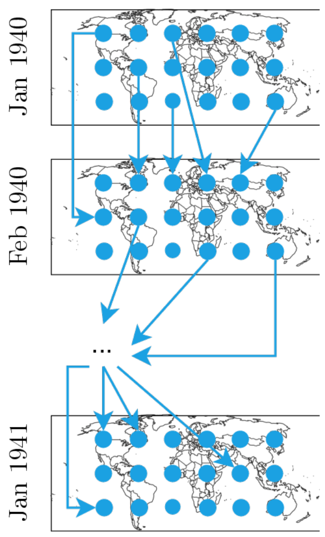
(c) (s3, t1)



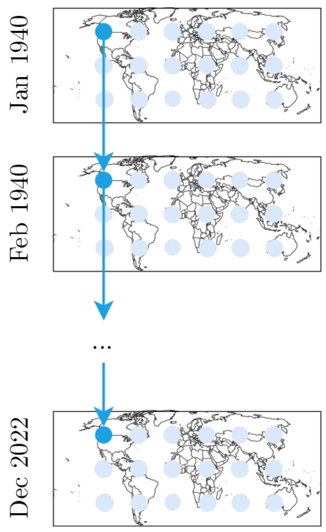
(d) (s1, t2)



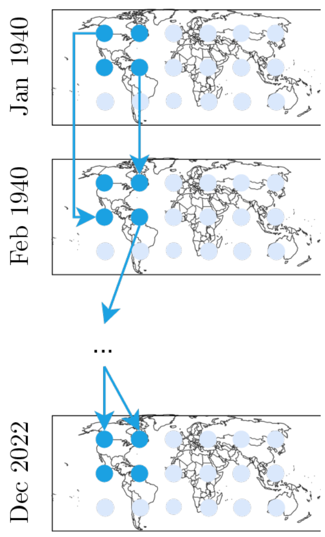
(e) (s2, t2)



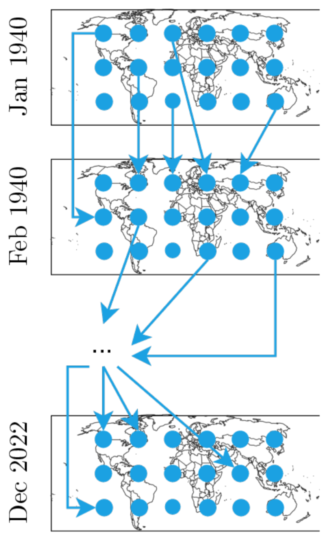
(f) (s3, t2)



(g) (s1, t3)



(h) (s2, t3)



(i) (s3, t3)

Figure 3.2: Spatio-temporal graph configurations: all combinations of spatial (s1, s2, s3) and temporal (t1, t2, t3) settings. The labels indicate the spatial and temporal settings used in each configuration.

complexity of the dataset is high, so it is not feasible for the network to learn the relationships between all locations at once. A possible network architecture for this case, assuming that for any input the time view is the same, is a shared encoder, combined with a set-based aggregator. For instance, a GNN encoder can be applied on each spatial view to learn separate representations, and then a Set Transformer[3] combines all them into a single representation, to be used for the final prediction. Instead, if each spatial view contains random locations, the network can learn the correlation between the features of those locations, but it is not guaranteed that the resulting representation is meaningful. In fact, the network could learn to associate locations that are not actually related, leading to a poor generalization. For what concerns the case (s3), if the dataset is not too large, it is the most straightforward approach, because the network can learn the relationships between all locations at once, without the need to combine representations from different spatial views.

We now focus on the temporal setting. (t2) is a good choice if the network is able to keep track of the temporal views observed for each input data. Again, the effectiveness of this configuration depends on how time views are defined. It does not really make sense to build graphs with information coming from random time steps, because we have no guarantee that edges connecting nodes at different time steps are significant. Instead, if the time view is obtained by selecting some consecutive time steps, the resulting graph captures meaningful temporal relationships, and hopefully, depending on how edges are defined, they could also express causal patterns. The challenge, similarly to the case of (s2), is to design a network capable to extract such relationships, taking into account different inputs. If the objective is to model complex temporal dynamics, the network should be able to learn from multiple time steps simultaneously, potentially using techniques such as temporal convolutions or recurrent layers. In other words, assuming that the spatial view is the same for any input, a possible network architecture is a shared encoder, combined with a temporal aggregator. The case (t3) can be feasible only if the dataset's temporal dimension is not too large, but teleconnection analysis implies a long-term perspective, so it is not the best choice. In fact, if the dataset contains many time steps, the network could struggle to learn meaningful relationships, because it would be overwhelmed by the amount of information.

Considering these insights, we can conclude that the best choice is to use the configuration (s3) and (t2): this allows the network to learn meaningful relationships between spatial locations, while still capturing temporal dynamics. We then introduce the concept of time window $w_\eta = \{t_1, t_2, \ldots, t_\eta\}$, which is a set of consecutive time steps, where t_1 is the first time step, t_η is the last one

[3]A Set Transformer is a neural network architecture designed to process sets of elements, where the order of the elements does not matter. It uses self-attention mechanisms to learn relationships between elements in the set, and it can be used to aggregate information from different sets [55].

and η is the number of time steps in the window. Time windows can partially overlap, meaning that the same time step can be included in multiple time windows. However, we assume that a total overlap is not possible, as it would generate identical time windows. Let $\mathcal{T}$ be the set of time steps and $\mathcal{W}_\eta$ be the set of all time windows of length η. The following function:

$$\begin{aligned} \text{time2window}: \quad & \mathcal{T} \to \mathcal{W}_\eta \\ & t \mapsto w_\eta \end{aligned}$$

maps each time step to a time window, where t is the time step and w_η is the time window containing that time step. Let $W_\eta = |\mathcal{W}_\eta|$, $T = |\mathcal{T}|$. We assume that each time window is shifted by one time step, so is trivial to show that, given η, $W_\eta = T - \eta + 1$.

As anticipated, we want to build a graph G whose nodes $v_{t,s}$ are points in space and time. We state that each graph represents a time window, so it should be identified by the first and last time steps of the window. We can introduce the following notation:

$$G_{i,\eta} = (V_{i,\eta}, E_{i,\eta})$$

where $G_{i,\eta}$ is the graph that starts in month i and ends after η time steps, so it represents the time window $\{i, i+1, \ldots, i+\eta\}$, $V_{i,\eta}$ is the set of vertices of the graph and $E_{i,\eta}$ is the set of edges. The set of vertices is defined as:

$$V_{i,\eta} = \{\mathbf{v}_{t,s} \mid t \in \{i, i+1, \ldots, i+\eta\}, s \in \{1, 2, \ldots, M\}\}$$

This better highlights the fact that a same spatial location appears in every time step, and so for each time window. Hence, $S = \eta \cdot M$ is the number of points in space and time in each time window and $N_\eta = W_\eta \cdot S$ is the total number of points in all time windows. The following function:

$$\begin{aligned} \text{spatiotemporal2graph}: \quad & \mathcal{W}_\eta \to \mathcal{G}_\eta \\ & w_\eta \mapsto G_\eta \end{aligned}$$

maps each time window to a graph. The set of graphs $\mathcal{G}_\eta$ is then defined as:

$$\mathcal{G}_\eta = \{G_{i,\eta} \mid i \in \{1, 2, \ldots, T - \eta\}\}$$

where T is the total number of time steps. For each time window, a corresponding graph is built, so the total number of graphs is $|\mathcal{G}_\eta| = W_\eta$, each graph contains S nodes and the total number of nodes in all graphs is N_η.

3.1.4 Temporal influence

The edges can be defined as connections between nodes at consecutive time steps, or more complex relationships depending on the specific task. In this case we deal with task of teleconnection analysis, so we seek for causal relationships. Therefore, edges should be directed, otherwise the network could just recover the strength of the correlation, but not the direction of influence. A generalist approach is to define a fully connected graph, where each node is connected to all other nodes (see Figure 3.3).

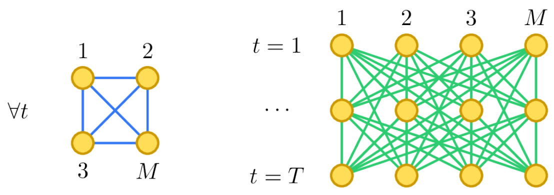


Figure 3.3: Fully connected graph representation with bidirectional edges using a mesh of $M = 4$ points and $T = 3$ time steps. Arrows are omitted for clarity. On the left, intra-temporal edges, connecting nodes at the same time step, are shown in blue. On the right, the geometry of the mesh is ignored, mesh points belonging to the same time step are disposed on a line, and inter-temporal edges, connecting nodes at different time steps, are shown in green.

However, trivial domain knowledge suggests that it does not make sense to connect a point in space and time to points belonging to previous time steps, because the present or future cannot influence the past. Therefore, a more plausible approach is to define a graph where edges are directed from past to future time steps, possibly changing spatial location, and from one spatial location to another one at the same time step (see Figure 3.4).

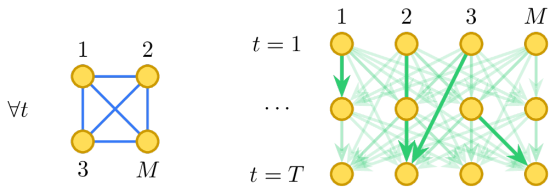


Figure 3.4: Graph representation with directed edges using a mesh of $M = 4$ points and $T = 3$ time steps. Arrows are omitted on the left for clarity. On the right, some edges are highlighted to show all the possible kinds of temporal relationships: from t to $t+1$, from t to $t+k$ $\forall k$, with and without changing spatial location. See Figure 3.3 for the meaning of shapes and colors.

We ask ourselves whether this graph structure is the best choice, thinking about how a neural network should interpret the graph to obtain a meaningful representation. Since, over the entire training, the network sees all time windows, it can learn to associate spatial locations with their temporal dynamics, even if not all times are connected each other. In other words, given t, s, instead of connecting node $v_{t,s}$ to node $v_{t+k,s}$ $\forall k$, it is enough to connect it to node $v_{t+1,s}$: on one hand, putting connections for all k would guide the model to analyze all possible temporal causal paths, but on the other hand, it would introduce a lot of redundant connections, which could lead to overfitting. Therefore, we can simplify the representation as shown in Figure 3.5.

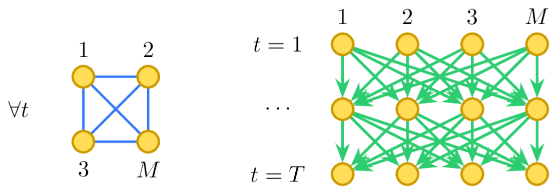


Figure 3.5: Graph representation with directed edges using a mesh of $M = 4$ points and $T = 3$ time steps, where edges are directed only to the same and next time step. Arrows are omitted on the left for clarity. See Figure 3.3 for the meaning of shapes and colors.

3.1.5 Spatial influence

Still keeping in mind the objective of teleconnection analysis, we can further simplify the graph structure by removing edges connecting nodes at the same time step: although they inherently represent teleconnections, since they associate climate features in different, even far, locations, they are not useful for causal inference, because they do not provide any information about temporal relationships. In particular, we can assume that a current event can only influence future events, and not other current events. Hence, with respect to Figure 3.5, we just ignore the left diagram. The current configuration assumes that any node $v_{t,s}$ can influence all nodes at the next time step, regardless of their spatial location. However, although in general each phenomenon is never totally independent from the others, it is reasonable to assume that a node $v_{t,s}$ can influence only nodes at the next time step that are close in space. This is also justified by the fact that the time granularity is one month, which is a relatively little time step for most climate phenomena, so it is unlikely that an event can spread across the globe in just 30 days: some conditions become evident in years or decades.

We can assume that a node $v_{t,s}$ can influence only nodes $v_{t+1,s'}$, where $s' \in \mathcal{N}_s$, i.e. the mesh point s' is a neighbor of the mesh point s in the mesh representation. Consequently, a node $v_{t,s}$ can be influenced only by nodes $v_{t-1,s'}$, where $s' \in \mathcal{N}_s$. A neighbor s' is a point adjacent to the point s in the mesh representation, i.e. the polyhedron based on which the mesh is built has a side between s and s'. Figure 3.6 shows this concept of influence on a mesh without refinement. This

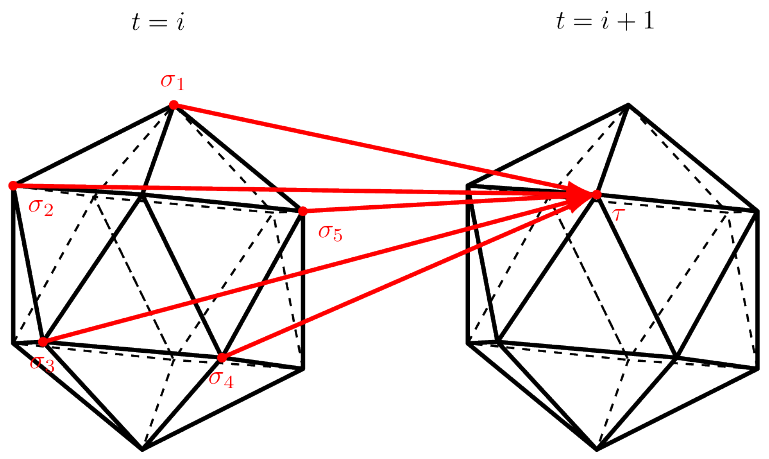


Figure 3.6: Graph representation of an icosahedron mesh with $M = 12$ points and $T = 2$ time steps. First time step is shown on the left, second time step is shown on the right. Since the mesh is not refined, node τ has 5 neighbors σ_i, edges are shown in red and highlight the present-future relationships. The same rule is applied to every other node.

assumption is very strong and could seem to exclude the possibility of long-range teleconnections. However, again we consider the fact that climate phenomena take long time to manifest on a global scale. Therefore, single edges, that traverse a time step and move in the neighborhood of the starting spatial location, cannot express teleconnection patterns. We then introduce the concept of *path* of length k as a sequence of k edges that connect a node $v_{t,s}$ to a node $v_{t+k,s'}$ through $k-1$ intermediate nodes. s, s' are related by the following condition:

$$s' \in \mathcal{N}_s^{(k)} = \bigcup_{i=1}^{k} \mathcal{N}_s^{(i)} \tag{3.3}$$

where $\mathcal{N}_s^{(i)}$ is the set of neighbors of s at distance i, i.e. the set of mesh points that are reachable from s by traversing i sides of the mesh. As anticipated, each edge of the path connects a node at time t to a node at time $t+1$, so a path inherently captures spatio-temporal relationships and can express long-range teleconnections by connecting distant nodes through intermediate neighbors. A path is defined as:

$$P_{t,s,s'}^{(k)} = (e_{v_{t,s},v_{t+1,s^{(1)}}}, \ldots, e_{v_{t+k-1,s^{(k-1)}},v_{t+k,s'}})$$

where t is the time step of the first node, s is the spatial location of the first node, s' is the spatial location of the last node, and $s^{(i)}$ is the spatial location of the i-th intermediate node. Each edge is defined as:

$$e_{v_{t^*,s^{(i)}},v_{t^*+1,s^{(j)}}} = \left(v_{t^*,s^{(i)}}, v_{t^*+1,s^{(j)}}\right)$$

where t^* is the time step of the source node, $s^{(i)}$ is the spatial location of the source node and $s^{(j)}$ is the spatial location of the destination node. An example of path is shown in Figure 3.7. Considering Equation 3.3, we can imagine that, as the length of the path increases, given a destination node $v_{t+k,s'}$, the number of possible source nodes $v_{t,s}$ increases, because the set of neighbors $\mathcal{N}_s^{(k)}$ grows. In other words, the longer the path, the more distant the source node can be from the destination node. This has also another interpretation: the longer the path, the bigger is the area of the Earth that can influence a single node. We can express this concept by introducing the notion of *cone-shaped relationship* between points in space and time, where the apex of the cone is the destination node and the base is the set of source nodes that reach it by traversing a path of length k. The cone is defined as:

$$C_{t,s}^{(k)} = \left\{v_{t-k,s'} \mid s' \in \mathcal{N}_s^{(k)}\right\}$$

where t is the time step of the destination node, s is the spatial location of the destination node, and s' is the spatial location of the source node. The cone-shaped relationship is shown in Figure 3.8.

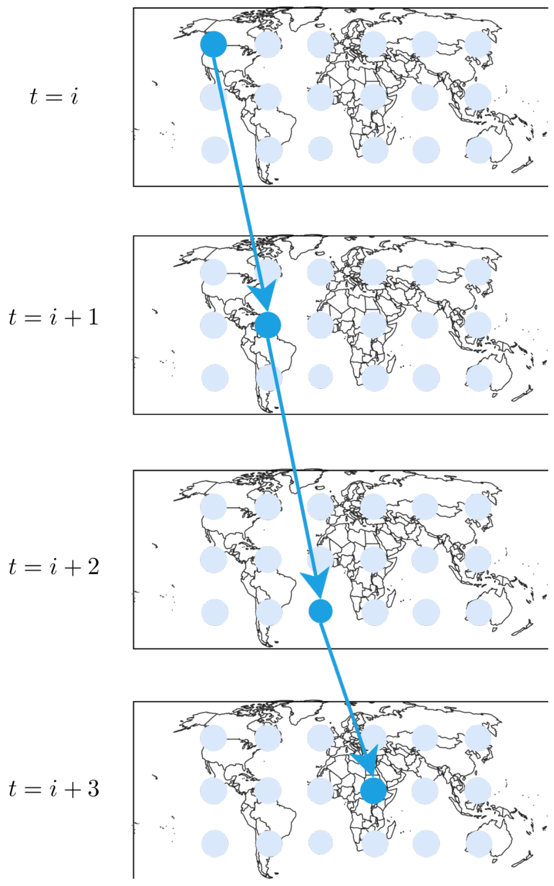


Figure 3.7: Graph representation of a path of length $k = 3$ over the planet connecting North America with Central Africa with $T = 4$ time steps. The position of mesh points is not exact and is just to illustrate that, at each time step, the spatial location moves to a geographical neighbor.

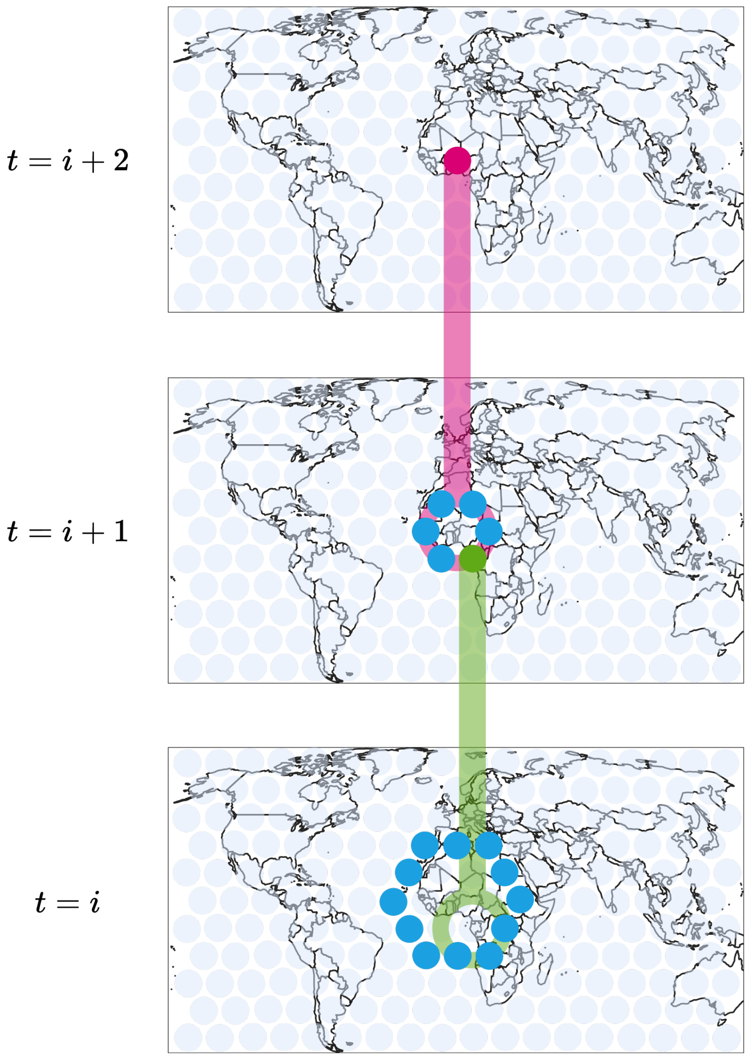


Figure 3.8: Cone-shaped relationship between mesh nodes on a graph with $T = 3$ time steps. For simplicity, a *cone* is represented as a *magnifying glass*, where the dot at the top represents the destination node, and the circle at the bottom represents the set of source nodes that can reach it by traversing a path of length $k = 1$, i.e. the neighbors of the destination node. Backward in time, the exploration of the neighbors continues just for one node, to avoid excessive graphical complexity. In red is indicated the cone of influence of the destination node with respect to the previous time step. In green is shown the cone of influence of one of the intermediate nodes of the path. In blue is indicated the extension of the cone of influence in a specific time step, i.e. the farthest nodes that can influence the destination node at that time step.

Please note that, since we are accounting for neighbors, we are actually excluding the temporal relationships for the same spatial location, so a mesh node does not influence itself at the next time step. We do not explicitly constrain in any way the distance between the source and destination mesh nodes along a path, so it is possible that the mesh node at the beginning of the path is the same as the mesh node at the end of the path. However, we force that, for any edge $e_{v_{t,s},v_{t+1,s'}}$ in the path, the source and destination nodes must be different, i.e. $s \neq s'$. This setup can guide the neural network to search for paths that explore the space, potentially connecting distant locations, but if for any reason the network prefers a self-connecting path, this is not excluded by the graph structure, and can be analyzed for interpretability purposes.

There is an additional aspect to consider: the above discussion assumes the capability of input graphs to represent spatial relationships regardless of the mesh refinement level. In fact, an edge connecting two neighbor points in a coarse mesh cannot connect the same points in the refined mesh, because they are not neighbors anymore, and they require more edges, i.e. a path, to reach connection. From Equation 3.3, we know that, given any mesh refinement level m, there exists a path between two mesh points s and s' of length k. We can intuitively think that, each time we refine the mesh, the length of a path connecting the same points increases. Since the refinement process uses the midpoint strategy, let's imagine a straight line connecting two points: at refinement level 0, the line is made up of two points, so the path length is 1, at refinement level 1, the line is made up of three points, so the path length is 2, at refinement level 2, the line is made up of five points, and the path length is 4, because divisions are computed recursively. Therefore, each time the mesh is refined, the path length doubles, so:

$$k^{(m)} = 2^m$$

If we want to make sure that the graph structure is able to represent the relationships of the coarse mesh in the refined one, we can define edges considering not simply the neighborhood, but the k-order neighborhood, i.e. the set of mesh points that can be reached from a given mesh point s by traversing k edges, and precisely the 2^m-order neighborhood. Let $d_m(s, s')$ be the minimum number of edges needed to connect s and s' in a m-refined mesh, then we can redefine:

$$\mathcal{N}_s^{(m)} = \{s' \mid d_m(s, s') \leq 2^m\}$$

and from now on we can assume that an edge in the graph is:

$$e_{v_{t,s},v_{t+1,s'}} = (v_{t,s}, v_{t+1,s'}) \quad \forall s' \in \mathcal{N}_s^{(m)}$$

It can finally be trivially adapted the definition of the cone-shaped relationship. In Appendix E.4, some properties of the neighborhood are shown, which can be useful to understand the relationships between mesh points in a graph.

3.1.6 Edge features

As anticipated, each input graph is characterized by vertices and edges. We want also to include in the graph construction some additional information that can be useful to the neural network to learn spatio-temporal relationships. In particular, we assume that the neural network does not have any prior knowledge about the spatio-temporal structure of the data. In fact, as from Section 3.1.3, the vertices incorporate spatio-temporal coordinates, but the edges have all the same meaning, regardless of the spatio-temporal distance. Therefore, we add edge features that include a spatial distance and a temporal distance. Given two points in space and time $v_{t,s}$ and $v_{t',s'}$, we define the spatial distance following Equation 3.2:

$$\Delta_\sigma(v_{t,s}, v_{t',s'}) = d(\mathbf{m}_s, \mathbf{m}_{s'}) = \arccos\left(\frac{\mathbf{m}_s \cdot \mathbf{m}_{s'}}{\|\mathbf{m}_s\| \cdot \|\mathbf{m}_{s'}\|}\right)$$

i.e. the angular distance between the mesh points s and s'. Instead, the temporal distance is defined as:

$$\Delta_\tau(v_{t,s}, v_{t',s'}) = |t - t'|$$

where t and t' are known in months, so the result is an integer number of months. The edge feature vector is then defined as:

$$\mathbf{e}_{v_{t,s}, v_{t',s'}} = [\Delta_\sigma(v_{t,s}, v_{t',s'}), \Delta_\tau(v_{t,s}, v_{t',s'})]$$

3.2 Neural architecture

The data structure has been defined without explicitly choosing the neural network architecture, but assuming that it should rely on processing graph-structured input data and thus making some assumptions about how the model should interpret the graph. Therefore, the model should be based on GNN (Graph Neural Network). We want that the network learns spatio-temporal relationships by exploring paths of various lengths and directions, selects relevant paths and thus provides causal patterns over the globe and the time window. As anticipated, each input graph represents a different time window, and each of them are shifted by one month, so the network can learn repetitive patterns. Moreover, different mesh refinement levels can be given as input to the model to analyze consistency between outputs, i.e. if the model learns the same relationships regardless of the mesh resolution.

3.2.1 Graph neural networks

GNNs are a class of neural networks specifically designed to operate on graph-structured data. Differently from traditional neural networks, which typically operate on fixed-size inputs, GNNs can handle variable-sized graphs and learn representations that capture the structure and relationships within the graph. Depending on how the model uses information from the graph, transductive and inductive GNNs can be distinguished. Transductive GNNs are trained on a specific graph and can only make predictions for nodes within that graph. In contrast, inductive GNNs can generalize to unseen graphs or nodes, allowing them to make predictions on new data that was not present during training. This is particularly useful in applications where the graph structure may change over time or when new nodes are added. Also, a classification can be made depending on the task of the GNN. Node-level tasks involve predicting labels for individual nodes, such as classifying the type of a node based on its features and its connections. Edge-level tasks focus on predicting the existence or type of edges between nodes, such as determining whether two nodes are connected or predicting the weight of an edge. Graph-level tasks involve predicting properties of the entire graph, such as classifying the graph into different categories based on its structure and features. In general some variants of GNNs can be identified [56]: GAT (Graph Attention Network), GCN (Graph Convolutional Network) [57] and GraphSAGE (Graph Sample and Aggregate) [58]. However, all of them are based on the same principles of message passing and aggregation, with different mechanisms for computing messages and updating node representations. The foundational framework is indeed MPNN (Message Passing Neural Network).

3.2.2 MPNNs and message passing

MPNNs are a general framework for designing GNNs that can operate on arbitrary graph structures. They provide a flexible and powerful way to model graph-structured data by allowing nodes to exchange information through a message passing mechanism, which enables nodes in a graph to iteratively exchange and update information based on their neighbors. The key components of MPNNs include the message function, the aggregation function, and the update function. The message function computes messages based on the features of neighboring nodes, the aggregation function combines these messages to form a new representation for each node, and the update function updates the node's representation based on the aggregated messages. The process can be expressed abstractly as follows:

$$h_v^{(k)} = \text{Update}\left(h_v^{(k-1)}, \text{Aggregate}\left(\{M(h_u^{(k-1)}) \mid u \in \mathcal{N}_v\}\right)\right)$$

where $h_v^{(k)}$ is the representation of node v at iteration k, $\mathcal{N}_v$ is the set of neighbors of node v, and M is the message function that computes messages from neighboring nodes. The process can be repeated for multiple iterations, allowing nodes to exchange information with their neighbors over several hops in the graph. A more concrete representation, that considers that nodes appear as vectors of features and even edges can have features, is the following:

$$\mathbf{x}_i^{(k)} = \gamma^{(k)} \left(\mathbf{x}_i^{(k-1)}, \bigoplus_{j \in \mathcal{N}_i} \phi^{(k)} \left(\mathbf{x}_i^{(k-1)}, \mathbf{x}_j^{(k-1)}, \mathbf{e}_{j,i} \right) \right)$$

where $\mathbf{x}_i^{(k)}$ is the feature vector of node i at iteration k, $\mathcal{N}_i$ is the set of neighbors of node i, $\phi^{(k)}$ is the message function that computes messages from neighboring nodes, $\mathbf{e}_{j,i}$ is the edge feature between nodes j and i, and $\gamma^{(k)}$ is the update function that combines the node's previous features with the aggregated messages. The aggregation function $\bigoplus$ can be any permutation-invariant operation[4], such as summation, mean, or max pooling[5]. The choice of aggregation function can significantly impact the performance of the GNN. In Figure 3.9 the concept of message passing is shown graphically. Message passing can be interpreted as a generalization of the convolution operation to irregular domains, like in this case graph-structured data. In fact, in traditional CNNs, the convolution operation is applied to local neighborhoods of pixels in an image. In GNNs, the message passing operation allows nodes to exchange information with their neighbors, effectively performing a convolution-like operation on the graph structure [61]. In an MPNN, each iteration of message passing is performed by a different layer of the network, so if K is the number of layers, it is also the number of message passing iterations. Moreover, each message passing iteration allows a node to gather information from its immediate neighbors. By stacking multiple message passing layers (i.e., increasing K), each node incrementally receives information from nodes that are further away in the graph. After K layers, a node's representation will incorporate information from nodes up to K hops away. This layered structure enables the model to capture broader contextual dependencies within the graph. However, in the real world if K is too large, experimentally bigger than 3, the model may struggle to learn meaningful representations, as the noise from distant nodes can overwhelm the signal from closer ones. In other words, deep MPNN architecture suffer from oversquashing: when many messages from a large neighborhood must be aggregated into a fixed-size representation, which constitutes a bottleneck, crucial

[4]Permutation-invariant operations are operations that yield the same result regardless of the order of the inputs.

[5]Pooling consists of aggregating information from a set of inputs to produce a single output, often used in neural networks to reduce the dimensionality of the data and capture important features. In the context of GNNs, pooling can be applied to aggregate information from neighboring nodes or edges, allowing the model to learn hierarchical representations of the graph structure. Common pooling operations include max pooling, average pooling, and sum pooling, each of which computes a summary statistic from the input set [59].

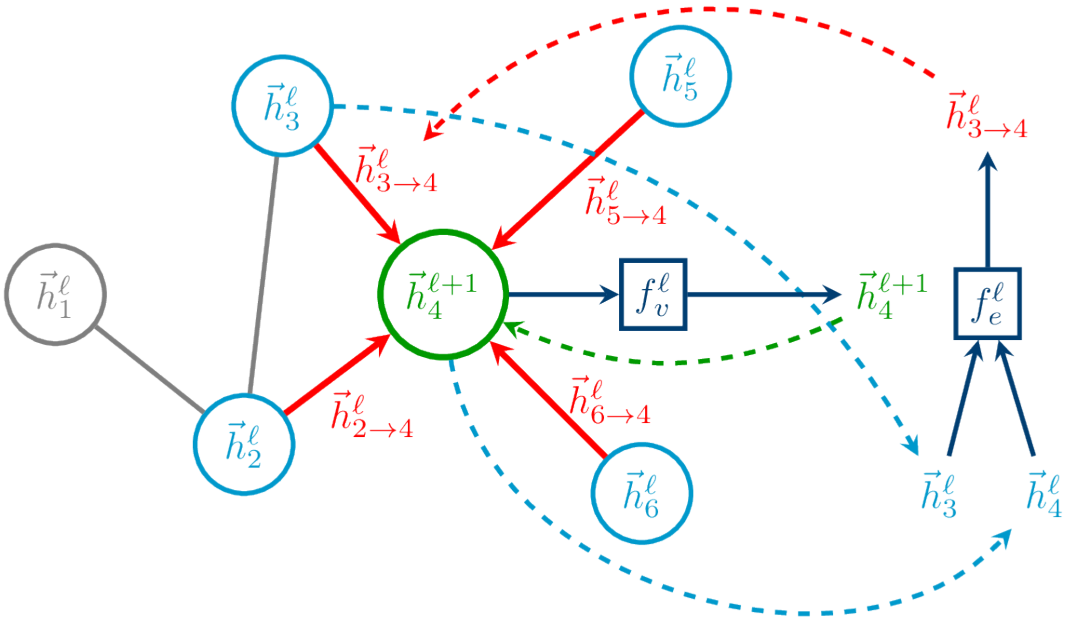


Figure 3.9: Message passing in a graph neural network. Each node receives messages from its neighbors, aggregates them, and updates its representation based on the aggregated information. The ℓ indicates the current level of the message passing, while the $\ell + 1$ indicates the next level. The red arrows indicate the messages sent from one node to another [60].

information can be *squashed*, i.e., compressed in a way that prevents the model from capturing long-range dependencies. They also suffer from oversmoothing: as the number of message passing iterations increases, the representations of nodes become increasingly similar, leading to a loss of discriminative power, as all nodes converge to similar embeddings. This is because the aggregation operation tends to average out the features of neighboring nodes, causing the representations to converge towards a common value. To mitigate this issue, techniques such as residual connections, skip connections, or attention mechanisms can be employed to preserve information from earlier layers and prevent oversmoothing [62, 63].

3.2.3 Graph attention networks

GATs [64] are a type of GNN that incorporates attention mechanisms to weigh the importance of neighboring nodes when aggregating their features. The key idea behind GATs is to assign different attention scores to different neighbors, allowing the model to focus on the most relevant edges in the graph. This is particularly useful in scenarios where the graph structure is heterogeneous or when some neighbors are more informative than others. The attention mechanism in GATs can be expressed as follows:

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^T\left[\mathbf{W}\mathbf{x}_i || \mathbf{W}\mathbf{x}_j\right]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\mathbf{a}^T\left[\mathbf{W}\mathbf{x}_i || \mathbf{W}\mathbf{x}_k\right]\right)\right)}$$

where α_{ij} is the attention score for the edge $e_{j,i}$, $\mathbf{a}$ is a learnable weight vector, $\mathbf{W}$ is a learnable weight matrix, $\mathbf{x}_i$ and $\mathbf{x}_j$ are the feature vectors of nodes i and j, respectively, and $||$ denotes concatenation. The attention scores are normalized across all neighbors of node i to ensure that they sum to 1. The final node representation can be computed as follows:

$$\mathbf{h}_i = \sigma\left(\sum_{j\in\mathcal{N}_i}\alpha_{ij}\mathbf{W}\mathbf{x}_j\right)$$

where σ is a non-linear activation function (e.g., ReLU). GAT layers can be stacked to form a multi-layer architecture, allowing the model to learn hierarchical representations of the graph. GATs also allow for multi-head attention, where multiple attention mechanisms are applied in parallel, and their outputs are concatenated or averaged to produce the final node representation. This can help to capture different aspects of the graph structure and improve the model's performance. In this case, final node representation can be computed as follows:

$$\mathbf{h}_i = \sigma\left(\bigoplus_{k=1}^{K}\sum_{j\in\mathcal{N}_i}\alpha_{ij}^{(k)}\mathbf{W}^{(k)}\mathbf{x}_j\right)$$

where K is the number of attention heads, $\alpha_{ij}^{(k)}$ is the attention score for the k-th head, and $\mathbf{W}^{(k)}$ is the weight matrix for the k-th head. The aggregation operation $\bigoplus$ can be either concatenation or averaging, depending on the specific implementation. In Figure 3.10 an example of how attention works in a GAT is shown: The image highlights that, among the neighbors of node i, node i itself is

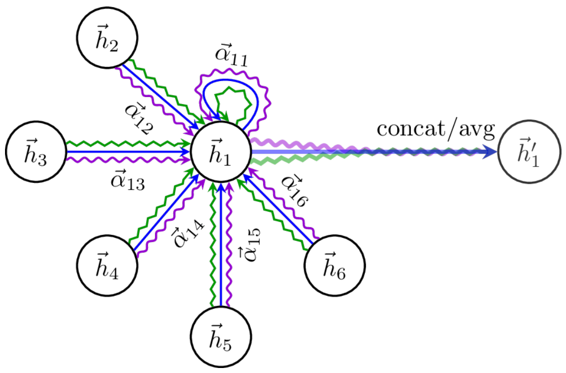


Figure 3.10: Graph attention mechanism. In this case, 3 attention heads are defined, respectively represented by the blue, green, and mauve arrows. The final node representation is computed by aggregating the outputs of all attention heads [64].

considered, so GATs are designed to handle self-loops, i.e. edges that connect a node to itself. This is useful to stabilize learning, since it helps preserve identity and prevents over-dependence on the local neighborhood. Moreover, it allows the model to incorporate information from the node's own features when computing attention scores and updating its representation. However, in our case, we do not want to consider self-loops, because we are interested in inter-node causality and influence propagation patterns, so we will not consider them in the attention mechanism. The attention mechanism in GATs is similar to the one used in natural language processing tasks, where the model learns to weigh the importance of different words in a sentence based on their context [64]. In our case, GAT-based architectures are fundamental to learn the relationships between nodes and edges in the graph, because the attention mechanism allows recovering causal patterns in the data: if the model assigns high attention scores to certain edges, it indicates that those edges are more relevant to explain feature values of destination nodes of those edges, and therefore a potential causal path between source and destination nodes.

Depending on the implementation, GATs can incorporate additional features, such as edge features. In this case, which is our goal, the attention mechanism can be extended to consider edge features when computing attention scores. The attention score between nodes i and j can be computed as follows:

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^T\left[\mathbf{W}\mathbf{x}_i||\mathbf{W}\mathbf{x}_j||\mathbf{W}_e\mathbf{e}_{ij}\right]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\mathbf{a}^T\left[\mathbf{W}\mathbf{x}_i||\mathbf{W}\mathbf{x}_k||\mathbf{W}_e\mathbf{e}_{ik}\right]\right)\right)}$$

where $\mathbf{e}_{ij}$ is the edge feature vector between nodes i and j, and $\mathbf{W}_e$ is a learnable weight matrix for the edge features. In other words, the GAT can incorporate edge features into the attention mechanism by concatenating them with the node features.

A variation from standard GAT is represented by GATv2, that introduces some modifications to the attention mechanism to improve its performance. The authors of GATv2 motivate the need for a new version stating that the traditional GAT layer computes only static attention. In other words, for any set of nodes V and a trained GAT layer, the attention function α defines a constant ranking ($argsort$) of the nodes, unconditioned on the query nodes i. Instead, GATv2 introduces a dynamic attention mechanism, where the attention scores are conditional on their query nodes, allowing for a strictly more expressive attention mechanism. Therefore, let $e : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be a score function that computes the attention score between two nodes based on their features. In the two versions of GAT, the attention scores are computed as follows:

$$\begin{aligned} \text{GAT}: \quad & e\left(\mathbf{h}_i, \mathbf{h}_j\right) = \text{LeakyReLU}\left(\mathbf{a}^\top \cdot \left[\mathbf{W}\mathbf{h}_i \| \mathbf{W}\mathbf{h}_j\right]\right) \\ \text{GATv2:} \quad & e\left(\mathbf{h}_i, \mathbf{h}_j\right) = \mathbf{a}^\top \text{LeakyReLU}\left(\mathbf{W} \cdot \left[\mathbf{h}_i \| \mathbf{h}_j\right]\right) \end{aligned}$$

where $\mathbf{h}_i$ and $\mathbf{h}_j$ are the feature vectors of nodes i and j, respectively, $\mathbf{W}$ is a learnable weight matrix, $\mathbf{a}$ is a learnable weight vector, and $\|$ denotes concatenation. Edge features are ignored for simplicity, but the reasoning is analogue to standard GAT. This different formulation is justified by the fact that the application of consecutive linear operations in original GAT on $\mathbf{W}, \mathbf{a}$ can be collapsed into a single linear layer, which leads to the model's choices be independent by the query node i.

Based on these considerations, we can conclude that GATv2 is more suitable for our task and we will use GATv2 as the main building block of our neural network architecture. In particular, we employ a single attention head and some consecutive graph attention layers. Each layer independently computes the attention scores and updates the node representations based on the aggregated messages from their neighbors, and the output of each layer is a new set of node representations, which can be used as input to the next layer. Therefore, each layer chooses a different set of neighbors to aggregate information from, and if we combine single choices of different layers, that represent causal relationships, we can obtain a more complex causal path, that can define a spatio-temporal teleconnecton pattern. 2 consecutive layers are interleaved by a ReLU activation function, because it allows for the standard practice of introducing non-linearity in the model.

3.2.4 Autoencoders

Thanks to GATv2 architecture, given an input graph the model can learn a latent representation of the node features, compute for each node the attention scores with respect to its neighbors and therefore highlight the most relevant incoming edges. Our objective is that the model learns to effectively assign different weights to different edges, but the latent representation must also be meaningful. The main challenge of GNN-based architectures is that they are generally designed for supervised tasks, like classification, so they usually require a ground truth label for each node or edge in the graph. However, since we deal with known and unknown teleconnection patterns, we do not have any ground truth data, and we want the model to learn the relationships between nodes and edges without any supervision. Therefore, a simple GNN architecture is not sufficient, and we need a more general framework that can encapsulate the advantages of GATv2 while also making sure that the model understands data structure and is sensible to feature variations.

To this purpose, we can introduce the concept of autoencoders. An autoencoder is a type of neural network architecture that learns to encode input data into a lower-dimensional representation (latent space) and then reconstruct the original data from this representation. Therefore, it is meant for unsupervised learning tasks, like ours. The main components of an autoencoder are the encoder and the decoder. The encoder maps the input data to a latent space:

$$\mathbf{Z} = f_{\text{enc}}(\mathbf{X})$$

where $\mathbf{Z} \in \mathbb{R}^{N \times F}$ is the latent representation of the input data, f_{enc} is the encoder network, $\mathbf{X} \in \mathbb{R}^{N \times D}$ is the input data matrix, N is the number of samples, D is the dimensionality of the input data, and F is the dimensionality of the latent space. The decoder then reconstructs the original data from the latent representation:

$$\hat{\mathbf{X}} = f_{\text{dec}}(\mathbf{Z}) \tag{3.4}$$

where $\hat{\mathbf{X}} \in \mathbb{R}^{N \times D}$ is the reconstructed data and f_{dec} is the decoder network. The encoder and decoder are typically implemented as neural networks, where the encoder compresses the input data into a lower-dimensional latent space, and the decoder reconstructs the original data from the latent representation. Autoencoders can be classified into deterministic and probabilistic.

Deterministic autoencoders learn a fixed mapping from the input data to the latent space and they are mainly used for dimensionality reduction or anomaly detection. In a deterministic autoencoder, the encoder is defined as:

$$\mathbf{Z} = \mathbf{W}\mathbf{X} + \mathbf{b}$$

where $\mathbf{W} \in \mathbb{R}^{F \times D}$ is the weight matrix, $\mathbf{b} \in \mathbb{R}^{F}$ is the bias vector, and $\mathbf{Z}$ is the latent representation of the input data. The decoder is defined as:

$$\hat{\mathbf{X}} = \mathbf{W}^T\mathbf{Z} + \mathbf{b}$$

where $\mathbf{W}^T$ is the transpose of the weight matrix [65]. The training process involves minimizing a reconstruction loss. In case of continous data, it is the MSE between the original input data and the reconstructed data:

$$\mathcal{L}(\mathbf{X}, \hat{\mathbf{X}}) = \|\mathbf{X} - \hat{\mathbf{X}}\|^2$$

where $\mathbf{X}$ is the original input data, $\hat{\mathbf{X}}$ is the reconstructed data, and $\|\cdot\|^2$ is the squared Euclidean norm. Instead, for binary data, it is the binary cross-entropy (BCE):

$$\mathcal{L}(\mathbf{X}, \hat{\mathbf{X}}) = -\frac{1}{N}\sum_{i=1}^{N}\left(x_i \log(\hat{x}_i) + (1 - x_i)\log(1 - \hat{x}_i)\right)$$

where x_i is the i-th element of the input data $\mathbf{X}$, $\hat{x}_i$ is the i-th element of the reconstructed data $\hat{\mathbf{X}}$, and N is the number of samples. See Section C.3.1 for reference.

Probabilistic autoencoders learn a distribution over the latent space, allowing for sampling and generation of new data points. Variational autoencoders (VAE) are a type of probabilistic autoencoder that learns a distribution over the latent space using variational inference [66]. The encoder outputs the parameters of a Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$:

$$\boldsymbol{\mu}, \log \boldsymbol{\sigma}^2 = f_{\text{enc}}(\mathbf{X})$$

where f_{enc} is the encoder network and $\boldsymbol{\mu}, \boldsymbol{\sigma}^2 \in \mathbb{R}^F$ are the mean and variance of the approximate posterior. A latent variable $\mathbf{Z}$ is sampled using the reparameterization trick:

$$\mathbf{Z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{3.5}$$

where $\odot$ denotes element-wise multiplication, and $\boldsymbol{\epsilon}$ is a random noise vector sampled from a standard normal distribution. The decoder then reconstructs the input, following Equation 3.4. The VAE architecture does not constrain the nature of the encoder and decoder networks, so they can be implemented using any type of neural network. This is an important foundation for what we are going to build. The objective function of a VAE is to maximize the evidence lower bound (ELBO) on the log-likelihood of the data, which can be expressed as:

$$\text{ELBO}(\mathbf{X}, \hat{\mathbf{X}}, \boldsymbol{\mu}, \boldsymbol{\sigma}) = \mathbb{E}_{q_\phi(\mathbf{Z}|\mathbf{X})}[\log p_\theta(\mathbf{X}|\mathbf{Z})] - D_{\text{KL}}(q_\phi(\mathbf{Z}|\mathbf{X}) \,\|\, p(\mathbf{Z}))$$

where $q_\phi(\mathbf{Z}|\mathbf{X})$ is the approximate posterior distribution, $p_\theta(\mathbf{X}|\mathbf{Z})$ is the likelihood of the data given the latent variable, and $p(\mathbf{Z})$ is the prior distribution over the latent variable. The first term is the expected log-likelihood of the data given the latent variable, which encourages the model to reconstruct the input data accurately. The second term is the Kullback-Leibler (KL) divergence between the approximate posterior and the prior distribution, which encourages the latent space to follow a known prior distribution, typically a standard normal distribution. The KL divergence is defined as:

$$D_{\mathrm{KL}}(q \,\|\, p) = \mathbb{E}_q \left[\log \frac{q}{p} \right] = \int q(x) \log \frac{q(x)}{p(x)} dx \tag{3.6}$$

where q is the approximate posterior and p is the prior distribution. The KL divergence measures how much the approximate posterior deviates from the prior distribution [67]. Since a neural network is trained by minimizing a loss function, we can take as loss function the negative ELBO:

$$\mathcal{L}(\mathbf{X}, \hat{\mathbf{X}}, \boldsymbol{\mu}, \boldsymbol{\sigma}) = -\mathbb{E}_{q_\phi(\mathbf{Z}|\mathbf{X})}[\log p_\theta(\mathbf{X}|\mathbf{Z})] + D_{\mathrm{KL}}(q_\phi(\mathbf{Z}|\mathbf{X}) \,\|\, p(\mathbf{Z}))$$

We can rewrite:

$$\begin{aligned}
\mathcal{L}_{\mathrm{recon}}(\mathbf{X}, \hat{\mathbf{X}}) &= -\mathbb{E}_{q_\phi(\mathbf{Z}|\mathbf{X})}[\log p_\theta(\mathbf{X}|\mathbf{Z})] \\
\mathcal{L}_{\mathrm{KL}}(\boldsymbol{\mu}, \boldsymbol{\sigma}) &= D_{\mathrm{KL}}(q_\phi(\mathbf{Z}|\mathbf{X}) \,\|\, p(\mathbf{Z})) \\
\mathcal{L}(\mathbf{X}, \hat{\mathbf{X}}, \boldsymbol{\mu}, \boldsymbol{\sigma}) &= \mathcal{L}_{\mathrm{recon}}(\mathbf{X}, \hat{\mathbf{X}}) + \mathcal{L}_{\mathrm{KL}}(\boldsymbol{\mu}, \boldsymbol{\sigma})
\end{aligned}$$

and realize that, with respect to deterministic autoencoders, the loss function of a probabilistic autoencoder can be seen as a regularized version of its reconstruction loss.

3.2.5 Graph autoencoders

With respect to the general purpose of autoencoders, our goal is slightly different. In fact, we want our model to learn a meaningful latent representation of data, but we do not impose that the latent space has smaller dimensionality than the input space. Instead, it can be much higher, because capturing spatio-temporal relationships requires a lot of information. Moreover, we need a way to account for both node features and edge features, and a regular autoencoder is not able to do that. Therefore, we need to adapt the autoencoder architecture to the graph structure, and this is where graph autoencoders (GAE) come into play. GAEs are a type of autoencoder specifically designed to work with graph-structured data. They extend the concept of autoencoders to graphs by incorporating the graph structure into the encoding and decoding processes. The main components of a GAE are the encoder, which learns a latent representation of the graph, and the decoder, which reconstructs the graph structure from the latent representation. The encoder can be implemented using various types of GNNs which learn to aggregate information from neighboring nodes and edges, while the decoder can be implemented using another GNN that reconstructs the graph structure based on the learned embeddings, or a simple MLP, or even an inner product operation. In the following, just for explaination purposes, we assume that, for each graph $G = (V, E)$, the set of edges E is passed to the GAE as an adjacency matrix $\mathbf{A}$, which is a square matrix where the entry (i, j) indicates the presence or absence of an edge between nodes i and j. The adjacency matrix could also be weighted, meaning that the entries can take real values instead of just 0 or 1. However, our edge features are not meant as weights, but rather as additional information about the edges (they could not be used as weights, since they are made of more than one feature). Therefore, we use the standard definition of adjacency matrix:

$$\mathbf{A}_{i,j} = \begin{cases} 1 & \text{if } e_{i,j} \in E \\ 0 & \text{otherwise} \end{cases}$$

The output of the encoder can be described as follows:

$$\mathbf{Z} = f_{\text{enc}}(\mathbf{X}, \mathbf{A})$$

where $\mathbf{Z} \in \mathbb{R}^{N \times F}$ is the latent representation of the graph, $\mathbf{X} \in \mathbb{R}^{N \times D}$ is the node feature matrix, $\mathbf{A} \in \mathbb{R}^{N \times N}$ is the adjacency matrix of the graph, N is the number of nodes, D is the dimensionality of the node features, and F is the dimensionality of the latent space. As seen for GAT, depending on the neural network chosen for the encoder, edge features can be also incorporated into the encoding process. Again for simplicity, we assume that the edge features are passed to the GAE as a 3D tensor $\mathbf{E} \in \mathbb{R}^{N \times N \times D_e}$, where D_e is the dimensionality of the edge features. Each entry (i, j, k) corresponds to the k-th feature of the edge between nodes i and j. Therefore, the encoder can be extended to incorporate edge features as follows:

$$\mathbf{Z} = f_{\text{enc}}(\mathbf{X}, \mathbf{A}, \mathbf{E})$$

While the reconstruction performed by standard autoencoders focuses on input data (nodes), in GAE reconstructing the graph structure from the latent representation means that the decoder must learn to predict the adjacency matrix $\hat{\mathbf{A}}$ from the latent representation $\mathbf{Z}$. The output of the decoder is:

$$\hat{\mathbf{A}} = f_{\text{dec}}(\mathbf{Z}) \tag{3.7}$$

where $\hat{\mathbf{A}} \in \mathbb{R}^{N \times N}$ is the reconstructed adjacency matrix of the graph. In the most simple case of an inner product decoder, the reconstruction can be expressed as:

$$\hat{\mathbf{A}} = \sigma(\mathbf{Z}\mathbf{Z}^T)$$

where σ is a non-linear activation function (e.g., sigmoid) applied element-wise to the resulting matrix. This operation computes the pairwise similarity between the latent representations of nodes, effectively reconstructing the adjacency matrix of the graph. The result can be interpreted as the predicted probabilities of edges between nodes based on their latent representations. Unlike standard autoencoders, since $\mathbf{A}$ is a binary matrix, the loss function is a binary cross-entropy loss:

$$\mathcal{L}(\mathbf{A}, \hat{\mathbf{A}}) = -\frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\Big(\mathbf{A}_{i,j}\log(\hat{\mathbf{A}}_{i,j}) + (1-\mathbf{A}_{i,j})\log(1-\hat{\mathbf{A}}_{i,j})\Big) \tag{3.8}$$

The training task of a GAE aims to learn a meaningful latent representation of the graph while reconstructing the adjacency matrix as accurately as possible. While this is a good starting point, we are actually interested in something more: given a node i, we want that the attention weights $\alpha_{i,j}\,\forall j \in \mathcal{N}_i$ computed by the GATv2 follow a certain distribution, such that we reduce the risk that all attention weights are uniformly distributed. First of all, we can constrain the latent space distribution, so that the model learns to encode the graph structure in a more meaningful way. We can think to use a VGAE (Variational graph autoencoder), which, as expected, extends the concept of VAE to graphs. Again, the outcome of a VGAE is a probabilistic graph representation, allowing for more expressive modeling of the graph structure with respect to standard GAE. The encoder of VGAE outputs the parameters of a Gaussian distribution:

$$\boldsymbol{\mu}, \log\boldsymbol{\sigma}^2 = f_{\text{enc}}(\mathbf{X}, \mathbf{A}, \mathbf{E})$$

where f_{enc} is the encoder network, $\boldsymbol{\mu}$ is the mean, and $\boldsymbol{\sigma}^2$ is the variance of the latent space distribution. A latent variable $\mathbf{Z}$ is sampled using the reparameterization trick described in Equation 3.5. The reconstruction of the adjacency matrix is performed like in Equation 3.7. The VGAE loss function is the same as the one of VAE, but applied to the adjacency matrix:

$$\begin{aligned}
\mathcal{L}_{\text{recon}}(\mathbf{A}, \hat{\mathbf{A}}) &= -\mathbb{E}_{q_\phi(\mathbf{Z}|\mathbf{X},\mathbf{A},\mathbf{E})}[\log p_\theta(\mathbf{A}|\mathbf{Z})] \\
\mathcal{L}_{\text{KL},\mathbf{Z}}(\boldsymbol{\mu}, \boldsymbol{\sigma}) &= D_{\text{KL}}(q_\phi(\mathbf{Z}|\mathbf{X}, \mathbf{A}, \mathbf{E}) \,\|\, p(\mathbf{Z})) \\
\mathcal{L}(\mathbf{A}, \hat{\mathbf{A}}, \boldsymbol{\mu}, \boldsymbol{\sigma}) &= \mathcal{L}_{\text{recon}}(\mathbf{A}, \hat{\mathbf{A}}) + \mathcal{L}_{\text{KL},\mathbf{Z}}(\boldsymbol{\mu}, \boldsymbol{\sigma})
\end{aligned} \tag{3.9}$$

Using VGAE implies a slight variation on how the different GATv2 layers are stacked. Let K be the number of GATv2 layers in a standard GAE architecture, and let K' be the number of GATv2 layers in a VGAE model. In order to get the same number of transformations of the latent space, we need to set $K' = K + 2$, because the first K layers are used to compute the latent representation of the graph, while the last 2 layers are used to compute the mean and variance of the latent space distribution. More precisely, for both GAE and VGAE the output of the k-th layer is:

$$\mathbf{Z}^{(k)} = f^{(k)}_{\text{GATv2}}(\mathbf{Z}^{(k-1)}, \mathbf{A}, \mathbf{E}) \quad \forall k \in [1, K]$$

where $\mathbf{Z}^{(0)} = \mathbf{X}$ and $f^{(k)}_{\text{GATv2}}$ is the computation performed by the k-th GATv2 layer. But then, assuming that the $k+1$-th layer is used to compute the mean of the latent space distribution, and the $k+2$-th layer is used to compute the variance, we have:

$$\boldsymbol{\mu} = f^{(K+1)}_{\text{GATv2}}(\mathbf{Z}^{(K)}, \mathbf{A}, \mathbf{E})$$
$$\log \boldsymbol{\sigma}^2 = f^{(K+2)}_{\text{GATv2}}(\mathbf{Z}^{(K)}, \mathbf{A}, \mathbf{E})$$

Please note that these two layers are not stacked one after the other, but they both branch from the output of the K-th layer, so they can be computed in parallel. The outputs of the $K+1$-th and $K+2$-th layers are then used to sample the latent variable $\mathbf{Z}$ using the reparameterization trick, as described previously. Like the other layers, they also produce attention weights. However, for the causal path discovery task, we are not interested in them, because they are not associated to some edges of the path, but rather to the latent space distribution. Figure 3.11 shows the difference between GAE and VGAE architectures, highlighting the additional layers used to compute the latent space distribution in VGAE. VGAE is a powerful framework that, thanks to the probabilistic modeling, improves generalization capabilities of GAE and prevents overfitting. Its stochastic nature introduces a kind of implicit data augmentation: a node is represented by a distribution of latent vectors, rather than a single point in the latent space. Of course, this comes at the cost of increased computational complexity, but still manageable [68]. Equation 3.9 highlights that, as expected, the loss function of VGAE is a regularized version of the reconstruction loss of GAE. A target distribution defined for the latent space is fine, but we also need a regularization for the attention weights computed by each GATv2 layer.

3.2.6 Loss function

First of all, with respect to Equation 3.9, we need to consider that the adjacency matrix $\mathbf{A}$, in real-world cases, is usually sparse, meaning that most of the entries are zero. We call *positive edges* the edges that are present in the graph, and *negative edges* the edges that are not present in the graph. We know that A is a binary matrix, for positive edges $\mathbf{A}_{i,j} = 1$ and for negative edges $\mathbf{A}_{i,j} = 0$. Therefore, the VGAE reconstruction loss can be rewritten as:

$$\mathcal{L}_{\text{recon}}(\mathbf{A}, \hat{\mathbf{A}}) = -\frac{1}{N^2}\left(\sum_{e_{i,j}\in E} \log(\hat{\mathbf{A}}_{i,j}) + \sum_{e_{i,j}\notin E} \log(1 - \hat{\mathbf{A}}_{i,j})\right)$$

where E is the set of positive edges in the graph. However, $\mathbf{A}$ is also strongly imbalanced, i.e. the number of negative edges vastly exceeds the number of positive edges. This training objective leads to poor generalization and uncontrolled overfitting. We then introduce a negative sampling strategy to address this issue.

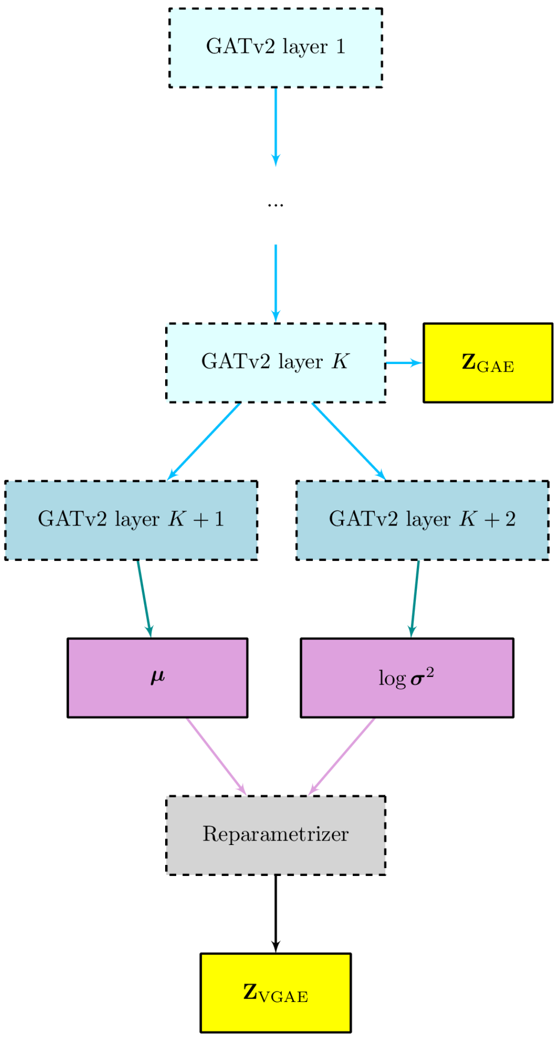


Figure 3.11: Latent space computation comparison between GAE and VGAE architectures. $\mathbf{Z}_{\text{GAE}}$ is the latent representation of the graph computed by GAE, while $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are the mean and variance of the latent space distribution computed by VGAE. The reparameterization trick is used to sample the latent variable $\mathbf{Z}$ from the distribution defined by $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$.

In other words, we randomly select negative edges from the graph, and we add them to the loss function, in place of all the negative edges. The number of negative edges is chosen to be equal to the number of positive edges, so that the loss function is balanced. Let $\bar{E}$ be the set of negative edges sampled from the graph, then:

$$\mathcal{L}_{\text{recon}}(\mathbf{A}, \hat{\mathbf{A}}) = -\frac{1}{2|E|}\left(\sum_{e_{i,j}\in E} \log(\hat{\mathbf{A}}_{i,j}) + \sum_{e_{i,j}\in \bar{E}} \log(1 - \hat{\mathbf{A}}_{i,j})\right)$$

because $|E| = |\bar{E}|$.

For what concerns the VGAE KL divergence term, we can assume a standard normal distribution as prior, i.e. $p(\mathbf{Z}) = \mathcal{N}(0, \mathbf{I})$. Therefore, the KL divergence can be computed as:

$$\mathcal{L}_{\mathrm{KL},\mathbf{Z}}(\boldsymbol{\mu}, \boldsymbol{\sigma}) = D_{\mathrm{KL}}(q_\phi(\mathbf{Z}|\mathbf{X}, \mathbf{A}, \mathbf{E}) \,\|\, p(\mathbf{Z})) = \mathbb{E}_{q_\phi(\mathbf{Z}|\mathbf{X},\mathbf{A},\mathbf{E})}\left[\log \frac{q_\phi(\mathbf{Z}|\mathbf{X}, \mathbf{A}, \mathbf{E})}{p(\mathbf{Z})}\right]$$

which can be computed as:

$$\mathcal{L}_{\mathrm{KL},\mathbf{Z}}(\boldsymbol{\mu}, \boldsymbol{\sigma}) = -\frac{1}{2N}\sum_{i=1}^{N}\left(1 + \log(\sigma_i^2) - \mu_i^2 - \sigma_i^2\right)$$

where μ_i and σ_i^2 are the mean and variance of the latent space distribution for the i-th node, respectively.

Finally, we introduce a regularization term for the attention weights computed by each GATv2 layer. We can use again Equation 3.6 as reference, but we assume as target distribution:

$$p(x) = \mathcal{U}(x) = \begin{cases} \frac{1}{N} & \text{if } x \in [0, 1] \\ 0 & \text{otherwise} \end{cases}$$

i.e. a uniform distribution over the interval $[0, 1]$ - the same on which attention weights are defined - where N_i is the number of neighbors of node i. Let $\boldsymbol{\alpha}^k \in \mathbb{R}^{N \times N}$ be the attention matrix computed by the k-th GATv2 layer, and N is the number of nodes in the graph. An attention matrix is a square matrix where the entry (i, j) indicates the attention weight assigned by node i to node j for the edge $e_{j,i}$. The distribution of attenton weights is indicated as α_i^k. Let $\boldsymbol{\alpha}$ be the set of all attention matrices, i.e., $\boldsymbol{\alpha} = \{\boldsymbol{\alpha}^k \mid k = 1, \ldots, K\}$. For the moment, we assume just one layer of GATv2, for simplicity. The KL divergence between the attention weights distribution α_i and the target distribution can be computed as follows:

$$\begin{aligned}
D_{\mathrm{KL},\boldsymbol{\alpha}}(\alpha_i \,\|\, \mathcal{U}) &= \mathbb{E}_{\alpha_i}\left[\log \frac{\alpha_i}{\mathcal{U}(\alpha_i)}\right] \\
&= \mathbb{E}_{\alpha_i}\left[\log \alpha_i - \log \mathcal{U}(\alpha_i)\right] \\
&= \mathbb{E}_{\alpha_i}\left[\log \alpha_i - \log \frac{1}{N_i}\right] \\
&= \mathbb{E}_{\alpha_i}\left[\log \alpha_i + \log N_i\right] \\
&= \mathbb{E}_{\alpha_i}\left[\log \alpha_i\right] + \log N_i
\end{aligned}$$

where $\mathbb{E}_{\alpha_i}$ is the expectation with respect to the attention weights distribution α_i. The term $\log N_i$ is constant for each node i, so we can ignore it in the loss function. What remains is strictly linked to the concept of entropy:

$$H(\alpha_i) = -\mathbb{E}_{\alpha_i}\left[\log \alpha_i\right]$$

The entropy of a distribution measures the uncertainty or randomness of the distribution. In other words, it quantifies how much information is contained in the distribution. The higher the entropy, the more uncertain or random the distribution is. The entropy is maximized when the distribution is uniform. Therefore:

$$D_{\mathrm{KL},\boldsymbol{\alpha}}(\alpha_i \,\|\, \mathcal{U}) = -H(\alpha_i) + \log N_i$$

Maximizing the KL divergence, which means making the distribution of attention weights as less uniform as possible, is equivalent to minimizing the entropy of the attention weights. Therefore, we can define the entropy loss as:

$$\mathcal{L}_{\mathrm{entropy}}(\alpha_i) = H(\alpha_i) = -\mathbb{E}_{\alpha_i}\left[\log \alpha_i\right] = -\sum_{j \in \mathcal{N}_i} \alpha_{i,j} \log \alpha_{i,j}$$

Since attention weights are computed separately by each GATv2 layer, we can define the entropy loss for the k-th layer as:

$$\mathcal{L}^k_{\mathrm{entropy}}(\alpha^k_i) = -\sum_{j \in \mathcal{N}_i} \alpha^k_{i,j} \log \alpha^k_{i,j}$$

where $\alpha^k_{i,j}$ is the attention weight computed by the k-th GATv2 layer on the edge $e_{j,i}$. Consequently, the average entropy loss for all K layers is:

$$\mathcal{L}_{\mathrm{entropy}}(\alpha_i) = \frac{1}{K} \sum_{k=1}^{K} \mathcal{L}^k_{\mathrm{entropy}}(\alpha^k_i)$$

The entropy loss should act as a regularization term. Hence, let $\lambda > 0$ be a hyperparameter that controls the trade-off between the VGAE loss and the entropy loss. The overall loss function for the VGAE with entropy regularization can be defined as:

$$\mathcal{L}(\mathbf{A}, \hat{\mathbf{A}}, \boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\alpha}) = \mathcal{L}_{\mathrm{recon}}(\mathbf{A}, \hat{\mathbf{A}}) + \mathcal{L}_{\mathrm{KL},\mathbf{Z}}(\boldsymbol{\mu}, \boldsymbol{\sigma}) + \lambda \cdot \mathcal{L}_{\mathrm{entropy}}(\boldsymbol{\alpha}) \tag{3.10}$$

The defined loss is one of the main metrics to evaluate the performance of the model during training, validation and testing. Some additional measurements are performed to assess the generalization capabilities of the model: the AP (Average Precision) and the AUC (Area Under the Curve) [69]. The AP assumes that the edges are sorted by their predicted probabilities, and it computes the average precision at each rank. Let $y_{(k)}$ be the *ground truth* label for the edge at rank k, i.e., $y_{(k)} = 1$ if the edge is positive, so it really exists, and 0 otherwise. Let E' be the total number of edges predicted by the model, and E be the total number of positive edges in the graph. The AP is defined as:

$$\mathrm{AP} = \frac{1}{E} \sum_{k=1}^{E'} P(k) \cdot y_{(k)}$$

where $P(k)$ is the precision at rank k, defined as:

$$P(k) = \frac{1}{k} \sum_{j=1}^{k} y_{(j)}$$

The AUC is defined as the area under the receiver operating characteristic (ROC) curve, which is a plot of the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The TPR is defined as:

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{TP}}{E}$$

where TP is the number of true positives, i.e, the number of positive edges correctly predicted by the model, and FN is the number of false negatives, i.e., the number of positive edges not predicted by the model. The FPR is the following:

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} = \frac{\text{FP}}{\bar{E}}$$

where FP is the number of false positives, i.e., the number of negative edges incorrectly predicted as positive by the model, and TN is the number of true negatives, i.e., the number of negative edges correctly predicted as negative by the model. Assuming that TPR is a function of FPR, the AUC is then defined as:

$$\text{AUC} = \int_0^1 \text{TPR}(f)\, d\text{FPR}(f)$$

3.2.7 Final model architecture

As anticipated, edges can be represented as an adjacency matrix, but also as an edge list, which is a common standard if the graph is sparse. In our case, graphs contain many edges, but the edge configuration is far from being fully connected, as described previously. Therefore, we prefer to use the edge list. Moreover, the network is trained by providing in the each epoch more input graphs that are packed into a batch. Please see Appendix C.3.3 for more insights on how to train a neural network.

The neural network architecture proposed for this work is shown in Figure 3.12. Different colors are used to highlight different components in the network, and also identify inputs and outputs of each component. Rectangles with solid borders contain any kind of information and also report the size of data, while rectangles with dashed borders perform transformations on input data. Dashed borders are used also to organize components in some modules: input data, encoder, latent representation, decoder, loss and metrics. Edge labels report the size of data flowing through some transformations. For simplicity, in the decoder module positive and negative edges are provided together and the sigmoid outputs two different sets of edge probabilities. Actually, these are two distinct computations that are both needed to compute loss and metrics. S_G, T_G are respectively the spatial and temporal cardinalities of the gridded raw dataset, while D is the number of features. B is the batch size, E is the number of edges in the graph, S_M is the number of spatial locations on the mesh, T_W is the number of time steps in each window, F is the number of features in the latent space, and K is the number of GATv2 layers. This diagram assumes some implementation details. For instance, edge features have 2 items in the last dimension because we store separately spatial and temporal distance. Positive and negative edges have 2 rows to represent one edge, because source and target node are stored separately.

We now recap the flow of data textually:

1. The raw dataset is transformed through the Grid2mesh approach described in Section 3.1.2 to produce a graph representation of the data.
2. The graph is passed to the encoder GATv2 layers. The first K layers process input graphs, while the last 2 layers compute the mean and variance of the latent space distribution. All GATv2 layers compute attention scores.
3. Mean and variance are then reparametrized to produce the latent representation.
4. From positive edges, negative edges are sampled.
5. The latent representation is then passed to the decoder, which computes probabilities separately for positive and negative edges.
6. The entropy loss is computed by combining the attention weights computed by each of the first K GATv2 layers.
7. The KL loss is computed using the mean and variance of the latent space distribution.
8. The reconstruction loss combines the reconstruction error on positive and negative edge probabilities.
9. AUC and AP are computed from edge probabilities.

3.3 Model outputs interpretation

3.3.1 Computation tree

As anticipated, our objective is to discover causal paths in the graph, which are built by traversing the edges in the graph that are highlighted by the attention

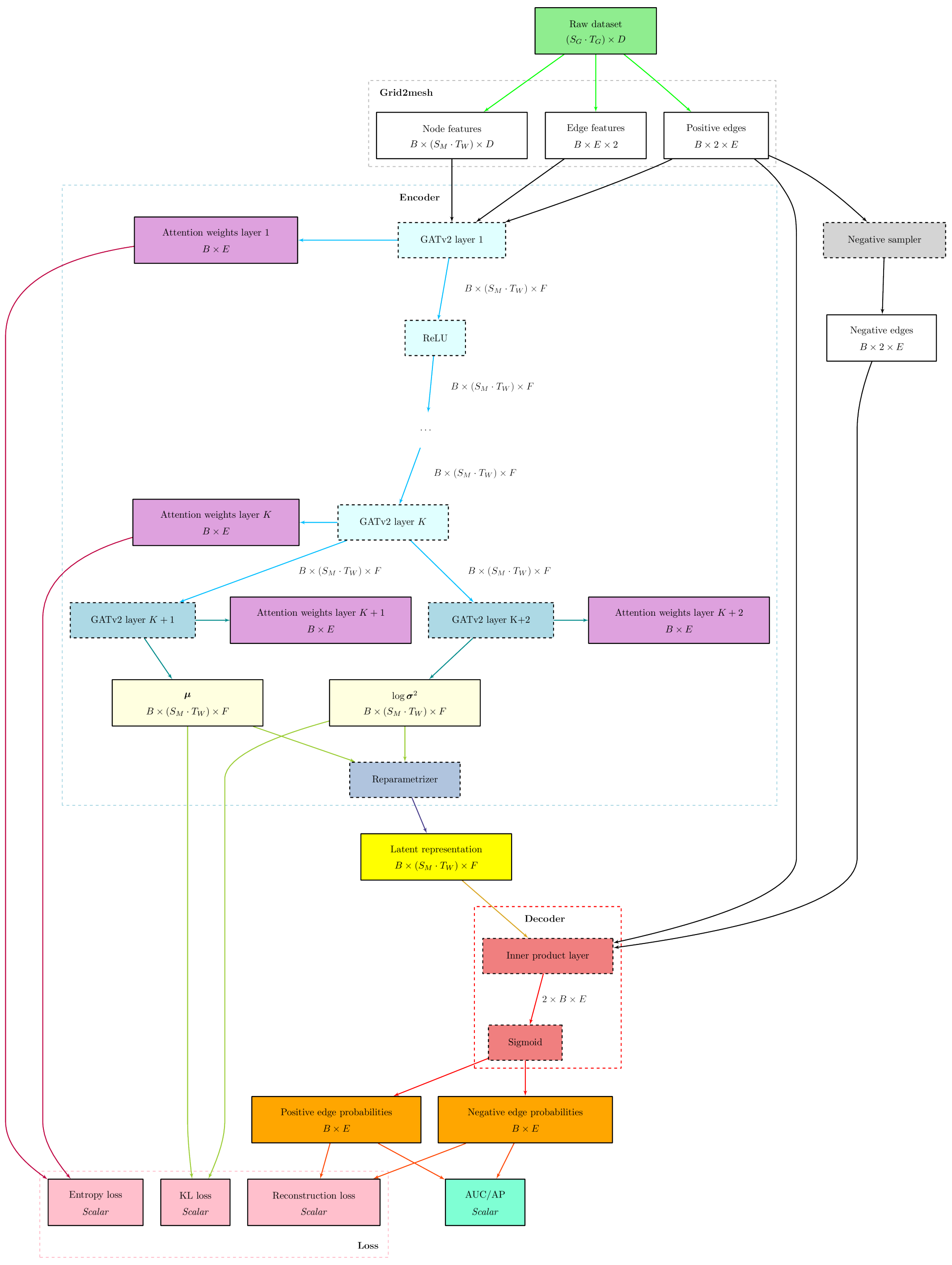

Raw dataset
$(S_G \cdot T_G) \times D$
Grid2mesh
Node features
$B \times (S_M \cdot T_W) \times D$
Edge features
$B \times E \times 2$
Positive edges
$B \times 2 \times E$
Encoder
Attention weights layer 1
$B \times E$
GATv2 layer 1
Negative sampler
$B \times (S_M \cdot T_W) \times F$
ReLU
Negative edges
$B \times 2 \times E$
$B \times (S_M \cdot T_W) \times F$
$\dots$
$B \times (S_M \cdot T_W) \times F$
Attention weights layer K
$B \times E$
GATv2 layer K
$B \times (S_M \cdot T_W) \times F$
$B \times (S_M \cdot T_W) \times F$
GATv2 layer $K+1$
Attention weights layer $K+1$
$B \times E$
GATv2 layer K+2
Attention weights layer $K+2$
$B \times E$
$\boldsymbol{\mu}$
$B \times (S_M \cdot T_W) \times F$
$\log \boldsymbol{\sigma}^2$
$B \times (S_M \cdot T_W) \times F$
Reparametrizer
Latent representation
$B \times (S_M \cdot T_W) \times F$
Decoder
Inner product layer
$2 \times B \times E$
Sigmoid
Positive edge probabilities
$B \times E$
Negative edge probabilities
$B \times E$
Entropy loss
Scalar
KL loss
Scalar
Reconstruction loss
Scalar
AUC/AP
Scalar
Loss

Figure 3.12: Proposed architecture for the neural network. See Section 3.2.7 for a detailed description of the meaning of shapes and colors.

weights distribution. Since each GATv2 layer of the neural network computes a set of attention weights, one could think to just select, for each layer and for each target node, the *best* incoming edge according to the attention weights. Nevertheless, this can only highlight local choices of the model and is not meaningful to discover causal paths. The main challenge is how to combine the attention weights of different layers. As described in Section 3.2.2, each layer operates over the same graph structure and therefore aggregates information from the same set of direct neighbors. However, while the graph topology remains unchanged, successive layers expand the receptive field of each node, effectively allowing it to incorporate information from progressively more distant parts of the graph. Let $h_i^{(0)}$ denote the initial feature vector of node i at the input of the first layer. Then, $h_i^{(1)}$ - the output of the first layer - depends on $h_i^{(0)}$ and the features $h_j^{(0)}$ of its immediate neighbors $j \in \mathcal{N}_i$. Similarly, $h_i^{(2)}$, computed by the second layer, depends on $h_i^{(1)}$ and the updated features $h_j^{(1)}$ from its neighbors, which themselves already encode information from their own neighbors in the previous layer, and so on. In other words, remembering that the GATv2 update rule at layer k is:

$$h_i^{(k)} = \sigma\left(\sum_{j\in\mathcal{N}_i} \alpha_{ij}^{(k)} \cdot \mathbf{W}^{(k)} h_j^{(k-1)}\right)$$

where σ is a non-linear activation function, $\mathbf{W}^{(k)}$ is the weight matrix of the k-th GATv2 layer, and $\alpha_{ij}^{(k)}$ is the attention weight computed by the k-th GATv2 layer on the edge (j, i), we can expand the representation of node i at layer k recursively. Let $k = 2$:

$$h_i^{(2)} = \sigma\left(\sum_{j\in\mathcal{N}_i} \alpha_{ij}^{(2)} \cdot \mathbf{W}^{(2)} \cdot \sigma\left(\sum_{\ell\in\mathcal{N}_j} \alpha_{j\ell}^{(1)} \cdot \mathbf{W}^{(1)} h_\ell^{(0)}\right)\right)$$

The recursion can be continued for any k, finding that at every step of the expansion the neighborhood of the node i is expanded by one hop, and the representation $h_i^{(k)}$ at layer k incorporates information from all nodes within a k-hop neighborhood of node i. This means that we can make an insightful analysis on the relation between the attention weights computed by each layer. Shin et al. [70] took care about the challenge of combining attention weights across layers. They introduced the concept of computation tree, which is a conceptual unrolling of the message-passing layers of a GAT into a tree structure starting from a target node. Each layer of the GAT corresponds to one step in the tree, and each node in the tree corresponds to a node in the original graph that can influence the target node's representation through a path of length equal to the number of layers. The tree shows all possible *computation paths* from the neighbors, neighbors-of-neighbors, etc., that contribute to the representation of the target node after multiple layers. The concept of computation tree is useful to highlight some observations. In fact, identical edges can appear multiple times in the computation tree, because they can be traversed by different paths. Moreover, nodes do not appear uniformly in the computation tree, because those ones that are closer to the target node are traversed by more computation paths, whereas those ones that are farther away are traversed by fewer of them, and only if the number of layers is large enough.

3.3.2 Attention matrix

In the following, for simplicity, we will use GAT to refer to GATv2.

Before discussing how to combine attention weights across layers, we need to analyze how attention weights can be represented. An attention matrix $A(k)$ is the matrix of attention weights computed by the k-th GAT layer on the incoming edges with respect to any target node i. More precisely:

$$A(k) = \begin{bmatrix} \alpha^k_{1,1} & \alpha^k_{1,2} & \cdots & \alpha^k_{1,n} \\ \alpha^k_{2,1} & \alpha^k_{2,2} & \cdots & \alpha^k_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha^k_{n,1} & \alpha^k_{n,2} & \cdots & \alpha^k_{n,n} \end{bmatrix}$$

where $\alpha^k_{i,j} \geq 0$ is the attention weight computed by the k-th GAT layer on the edge (j, i), and n is the number of nodes in the graph. The attention matrix $A(k)$ is a square matrix of size $n \times n$, where n is the number of nodes in the graph. We know that the attention weights are computed as follows:

$$\alpha^k_{i,j} = \frac{\exp(e^k_{i,j})}{\sum_{l \in \mathcal{N}_i} \exp(e^k_{i,l})} \quad \forall i, j \in \{1, \ldots, n\}$$

where $e^k_{i,j}$ is the attention score computed by the k-th GAT layer on the edge (j, i), and $\mathcal{N}_i$ is the set of neighbors of node i. Therefore:

$$\sum_{j \in \mathcal{N}_i} \alpha^k_{i,j} = 1 \quad \forall i \in \{1, \ldots, n\}$$

This means that the attention weights $\alpha^k_{i,j}$ are normalized. This property can be reflected on the attention matrix $A(k)$. In fact, in each row i, attention weights $\alpha^k_{i,j}$ are related to any edge (existing or not) having as destination node i. We say:

$$\alpha^k_{i,j} \begin{cases} > 0 & \text{if } j \in \mathcal{N}_i \\ = 0 & \text{if } j \notin \mathcal{N}_i \end{cases} \quad \forall i, j \in \{1, \ldots, n\}$$

Hence:

$$\sum_{j=1}^{n} \alpha^k_{i,j} = \sum_{j \in \mathcal{N}_i} \alpha^k_{i,j} + \sum_{j \notin \mathcal{N}_i} \alpha^k_{i,j} = 1 + 0 = 1 \quad \forall i \in \{1, \ldots, n\}$$

so each row in the attention matrix $A(k)$ sums to 1. This leads us to consider $A(k)$ a row-stochastic matrix [71], i.e. it could be interpreted as a transition matrix of a Markov chain, where the nodes are the states of the Markov chain and the attention weights are the transition probabilities between the states.

3.3.3 Correlation matrix

We want to combine different attention matrices. Shin et al. [70] defined the correlation matrix as a way to account for the influence of multiple layers in the GAT architecture and it captures how attention weights from different layers interact and contribute to the final representation of nodes in the graph. The correlation matrix $\mathbf{C}_L(k)$ for the k-th layer of a GAT network with L[6] layers can be defined recursively as:

$$\mathbf{C}_L(k) = \begin{cases} \mathbf{I} & \text{if } k = 1 \\ \mathbf{C}_L(k-1) \cdot \mathbf{A}(L-k+2) & \text{if } k > 1 \end{cases} \tag{3.11}$$

In other words, the k-th correlation matrix for L layers is the product of the attention matrices $A(k)$ of the last k layers. Therefore, the product is computed in reverse order, starting from the last layer, i.e. the layer near the output, and going back to the first one. $\mathbf{C}_L(k)$ is a square matrix of size $n \times n$, where n is the number of nodes in the graph. The correlation matrix can be interpreted as a transition matrix of a Markov chain, where the nodes are the states of the Markov chain and the attention weights are the transition probabilities between the states. We are interested to know if the correlation matrix $\mathbf{C}_L(k)$ is a row-stochastic matrix. In general, let A, B be two row-stochastic matrices. Then, the product $C = A \cdot B$ is also a row-stochastic matrix [72]. A proof for this for the correlation matrix is provided in Appendix E.5.

Shin et al. [70] use correlation matrix and attention matrix to calculate GATT:

$$\phi^v_{i,j} = \sum_{m=1}^{L} [\mathbf{C}_L(L-m)]_{v,i} [\mathbf{A}(m)]_{i,j} \tag{3.12}$$

where $\phi^v_{i,j}$ is the edge attribution of an edge $e_{i,j}$ with respect to node v in a L-layer GAT network, $[\mathbf{C}_L(L-m)]_{v,i}$ is the v-th row and i-th column of the correlation matrix $\mathbf{C}_L(L-m)$, and $[\mathbf{A}(m)]_{i,j}$ is the i-th row and j-th column of the attention matrix $\mathbf{A}(m)$. This metric accounts for the proximity effect - because the closer the edge to the target node, the higher is its impact for the computation of the target

[6]From now on, L has the meaning of number of layers of the GAT network, and it is not related to the mesh refinement level anymore.

node's representation - and adjusts the contribution of an edge by its position with respect to a target node. However, it depends on the choice of a target node v, so we cannot use it because we are interested in the whole graph and not just a single node. From Equation 3.11, we can easily understand that each term of the sum in Equation 3.12, fixed the target node v, multiplies the correlation matrix by the attention weight representing the *missing* edge in the computation tree. Let's make an example, assuming that $L = 3$:

$$\begin{aligned}
&\mathbf{C}_3(1) = \mathbf{I} \\
&\mathbf{C}_3(2) = \mathbf{C}_3(1) \cdot \mathbf{A}(3) = \mathbf{I} \cdot \mathbf{A}(3) = \mathbf{A}(3) \\
&\mathbf{C}_3(3) = \mathbf{C}_3(2) \cdot \mathbf{A}(2) = \mathbf{A}(3) \cdot \mathbf{A}(2)
\end{aligned}$$

Let:

$$\phi^v_{ij,m} = [\mathbf{C}_L(L-m)]_{v,i}[\mathbf{A}(m)]_{i,j}$$

Then, we can rewrite Equation 3.12 as:

$$\phi^v_{i,j} = \sum_{m=1}^{L} \phi^v_{ij,m}$$

where $\phi^v_{ij,m}$ is the edge attribution of an edge $e_{i,j}$ with respect to node v in the m-th layer of a L-layer GAT network. Since $L = 3$, we have:

$$\begin{aligned}
\phi^v_{ij,1} &= [\mathbf{C}_3(3)]_{v,i}[\mathbf{A}(1)]_{i,j} = [\mathbf{A}(3) \cdot \mathbf{A}(2)]_{v,i}[\mathbf{A}(1)]_{i,j} \\
\phi^v_{ij,2} &= [\mathbf{C}_3(2)]_{v,i}[\mathbf{A}(2)]_{i,j} = [\mathbf{A}(3)]_{v,i}[\mathbf{A}(2)]_{i,j} \\
\phi^v_{ij,3} &= [\mathbf{C}_3(1)]_{v,i}[\mathbf{A}(3)]_{i,j} = [\mathbf{I}]_{v,i}[\mathbf{A}(3)]_{i,j}
\end{aligned}$$

We can see that the last term in the product of each $\phi^v_{ij,m}$ is the attention weight computed by the farthest layer from output for a path of length m. Therefore, we can think of extending this concept to the whole graph, by defining a new correlation matrix $\mathbf{R}_L$ that includes also the *missing* layer:

$$\mathbf{R}_L(k) = \begin{cases} \mathbf{I} & \text{if } k = 0 \\ \mathbf{R}_L(k-1) \cdot \mathbf{A}(L-k+1) & \text{if } k > 0 \end{cases}$$

In this way, we defined a correlation matrix that captures the relationships between different layers of the graph. Therefore, each entry c^k_{ij} of $\mathbf{R}_L(k)$ represents the strength of the attention weights of the edges involved in a path from node j to node i of length k, that, as explained, traverses k layers in the GAT network.

3.3.4 Paths

We are interested to show that paths computed in the computation tree correspond to paths in the spatio-temporal graph. From now on, we use the term *correlation* *matrix* to identify R_L. Let $c^k_{i,j}$ be the i-th row and j-th column of the correlation matrix $\mathbf{R}_L(k)$. Then:

$$\mathbf{R}_L(k) = \begin{bmatrix} c^k_{1,1} & c^k_{1,2} & \cdots & c^k_{1,n} \\ c^k_{2,1} & c^k_{2,2} & \cdots & c^k_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ c^k_{n,1} & c^k_{n,2} & \cdots & c^k_{n,n} \end{bmatrix}$$

We know that each node of any input graph is a point in space and time. Therefore, n is the number of nodes over all time steps and spatial points considered in the input data. Let T be the number of time steps and S be the number of spatial points. Then, we can rewrite the correlation matrix as divided into $T \times T$ blocks:

$$\mathbf{R}_L(k) = \begin{bmatrix} \mathbf{R}_{L,1,1}(k) & \mathbf{R}_{L,1,2}(k) & \cdots & \mathbf{R}_{L,1,T}(k) \\ \mathbf{R}_{L,2,1}(k) & \mathbf{R}_{L,2,2}(k) & \cdots & \mathbf{R}_{L,2,T}(k) \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{R}_{L,T,1}(k) & \mathbf{R}_{L,T,2}(k) & \cdots & \mathbf{R}_{L,T,T}(k) \end{bmatrix}$$

where each block $\mathbf{R}_{L,i,j}(k)$ is a $S \times S$ correlation matrix for the i-th and j-th time steps:

$$\mathbf{R}_{L,i,j}(k) = \begin{bmatrix} c^{k,L}_{[i,1],[j,1]} & c^{k,L}_{[i,1],[j,2]} & \cdots & c^{k,L}_{[i,1],[j,S]} \\ c^{k,L}_{[i,2],[j,1]} & c^{k,L}_{[i,2],[j,2]} & \cdots & c^{k,L}_{[i,2],[j,S]} \\ \vdots & \vdots & \ddots & \vdots \\ c^{k,L}_{[i,S],[j,1]} & c^{k,L}_{[i,S],[j,2]} & \cdots & c^{k,L}_{[i,S],[j,S]} \end{bmatrix}$$

where $c^{k,L}_{[i,l],[j,m]}$ is the k-order correlation computed by a L-layer GAT between the l-th spatial point at the i-th time step and the m-th spatial point at the j-th time step. To facilitate the transition between the standard notation (that uses two indices to identify an element of the matrix) and the one defined for our dataset (that uses two pairs of indices to identify an element of the matrix), we define the following functions:

$$\text{time}(v_p) = i$$
$$\text{space}(v_p) = l$$

where $p = (i-1) \cdot S + l$ is the index of the node v_p in the input graph, i is the time step of the node v_p, and l is the spatial point of the node v_p. This mapping can be trivially extended also to attention weights and edges. Let:

$$P_{v_1 \to v_2 \to \cdots \to v_m \to v_{m+1}} = (e_{v_1,v_2}, e_{v_2,v_3}, \ldots, e_{v_m,v_{m+1}})$$

be a path of length:

$$|P_{v_1 \to v_2 \to \cdots \to v_m \to v_{m+1}}| = m \leq L$$

from node v_1 to node v_{m+1} passing through nodes $v_2, \dots, v_m$, where v_i refers to a generic node in the graph, without any specific spatial and temporal meaning, that takes the i-th position in the path. The path is a defined sequence of edges, where $e_{v_i,v_{i+1}}$ is the edge from node v_i to node v_{i+1}, and crosses m layers of the GAT network. A single edge in the path is identified by:

$$P_{v_1 \to v_2 \to \dots \to v_m \to v_{m+1}}[l] = e_{v_l,v_{l+1}} \quad \forall l \in \{1, 2, \dots, m\}$$

The following functions are useful:

$$\begin{aligned} \text{first}(P_{v_1 \to v_2 \to \dots \to v_m \to v_{m+1}}) &= v_1 \\ \text{last}(P_{v_1 \to v_2 \to \dots \to v_m \to v_{m+1}}) &= v_{m+1} \end{aligned}$$

Therefore, given p^m any path of length m:

$$\Pi^m_{v_1,v_{m+1}} = \{p^m \mid \text{first}(p^m) = v_1 \wedge \text{last}(p^m) = v_{m+1}\}$$

is the set of all paths having length m from node v_1 to node v_{m+1}. Because of how graphs are defined in our dataset, each edge $e_{v_i,v_{i+1}}$ is characterized by:

$$\text{time}(v_{i+1}) = \text{time}(v_i) + 1 \tag{3.13}$$

$$\text{space}(v_{i+1}) \in \mathcal{N}_{\text{space}(v_i)} \tag{3.14}$$

Let also $\alpha^k_{v_{i+1},v_i}$ be the attention weight computed by the k-th layer of the GAT network on the edge $e_{v_i,v_{i+1}}$. Since $\mathbf{C}_L(k)$ is obtained as a product of attention matrices, each term of the correlation matrix involves products of attention weights. In particular, the correlation $c^{k,L}_{i,j}$ is the sum of the products of attention weights along all paths of length k from node v_i to node v_j. This means that:

$$c^{k,L}_{i,j} = \sum_{p^k \in \Pi^k_{v_i,v_j}} \prod_{l=1}^{k} \alpha^{L-k+l}_{p^k[l]}$$

where the sum is taken over all paths p^k of length k from node v_i to node v_j, and $\alpha^l_{p^k[l]}$ is the attention weight computed by the l-th layer of the GAT network and associated with edge $p^k[l]$, i.e. the l-th step in the path p^k. A proof of this statement is provided in Appendix E.6. In particular, each element of the correlation matrix $\mathbf{C}_L(k)$ is the sum of the products of attention weights along all paths of length k from node $[t, s]$ to node $[t', s']$, so the correlation matrix, if properly analyzed, can be used to extract *significant* paths of length k.

3.3.5 Entropy

As anticipated, the attention matrix $\mathbf{A}(m)$ can be interpreted as a transition matrix of a Markov chain. To understand how much information is contained in the attention weights, we can compute the entropy of the attention matrix. The entropy of a Markov chain is defined as:

$$H(X) = -\sum_{i=1}^{n}\sum_{j=1}^{n} p_{i,j}\log(p_{i,j})$$

where X is the Markov chain, n is the number of states in the Markov chain, and $p_{i,j}$ is the transition probability from state i to state j [73]. A per-row entropy can instead be defined as:

$$H_i(X) = -\sum_{j=1}^{n} p_{i,j}\log(p_{i,j})$$

where $H_i(X)$ is the entropy of the i-th row of the transition matrix. While the matrix-level entropy $H(X)$ measures the uncertainty of the transition probabilities of the entire Markov chain, the per-row entropy $H_i(X)$ measures the uncertainty of the transition probabilities from a specific state i to all other states. The per-row entropy is useful to understand how much information is contained in the transition probabilities from a specific state, while the matrix-level entropy is useful to understand the quantity of information contained in the transition probabilities of the entire Markov chain. A high entropy value indicates that the transition probabilities are uniformly distributed, and so the model is more uncertain and less interpretable, while a low entropy value indicates that the transition probabilities are concentrated on a few states, so the model is more certain and more interpretable. In our case, the transition probabilities are the attention weights $\alpha_{i,j}^{m}$ computed by the m-th layer of a GAT network. Therefore, we can define the row-level entropy of the attention matrix $\mathbf{A}(m)$ as:

$$H_i(\mathbf{A}(m)) = -\sum_{j=1}^{n} \alpha_{i,j}^{m}\log(\alpha_{i,j}^{m})$$

This entropy tells how concentrated or dispersed the attention weights are for each target node i in the m-th layer of the GAT network, i.e. if all incoming edges have similar attention weights or if some edges are much more important than others. We can extend this definition to the correlation matrix $\mathbf{R}_L(k)$:

$$H_i(\mathbf{R}_L(m)) = -\sum_{j=1}^{n} c_{i,j}^{m,L}\log(c_{i,j}^{m,L})$$

We can easily understand that the meaning is the same of the row-level entropy of the attention matrix, but instead of analyzing the distribution of attention weight on incoming edges, it considers the distribution of attention weights of all paths of length m directed to node i. Since our GAT network is made of L layers, we can repeat this computation for each layer (or length) m. This means we are free to choose the length of the paths we want to analyze. In the following, we will assume to analyze the longest paths possible in our network, i.e. $m = L$. To use the entropy as a filter for the paths, we start considering the entropy of a uniform probability distribution U over n elements:

$$U = \left\{\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}\right\}$$

where $|U| = n$. The corresponding entropy is:

$$H_U = -\sum_{i=1}^{n} \frac{1}{n} \log\left(\frac{1}{n}\right) = \log(n)$$

Let now consider a Dirac distribution D over n elements [74]:

$$D = \{1, 0, \ldots, 0\}$$

where again $|D| = n$ and the non-zero element can take any position in the vector. The entropy of the Dirac distribution is:

$$H_D = -1 \cdot \log(1) - 0 \cdot \log(0) - \ldots - 0 \cdot \log(0) = 0$$

where $0 \cdot \log(0)$ is defined to be 0, because:

$$\lim_{x \to 0} x \log(x) = 0$$

For any other distribution, the entropy will be between 0 and $\log(n)$, i.e.:

$$0 \leq H \leq \log(n)$$

Assuming that only one path is significant for each target node, the ideal distribution for a perfect, significant teleconnection is a Dirac distribution, where the attention weight of the significant path is 1 and all other attention weights are 0. However, this is rarely the case in practice: in real-world scenarios, the attention weights are often distributed among multiple paths, leading to a non-zero entropy value. Therefore, we can use as threshold a positive value $t < \log(n)$, where n is the number of incoming paths for a target node, to filter out the paths that are not significant. The problem of this approach is that constraining the entropy does not guarantee any property about the shape of the distribution of attention weights, and moreover the entropy is not a linear function, so it is not possible to define a threshold that works for all cases. We can introduce the following distribution:

$$Q(\beta) = \left\{\beta + \frac{1-\beta}{n}, \frac{1-\beta}{n}, \ldots, \frac{1-\beta}{n}\right\}$$

where $\beta \in [0, 1]$ is a parameter that controls the concentration of the distribution around the first value, which is greater or equal than the others. The entropy of this distribution is:

$$H_Q(\beta) = -\left(\beta + \frac{1-\beta}{n}\right) \log\left(\beta + \frac{1-\beta}{n}\right) - (n-1) \cdot \frac{1-\beta}{n} \log\left(\frac{1-\beta}{n}\right)$$

The relationship between the entropy and the parameter β is shown in Figure 3.13. As β increases, the entropy decreases, meaning that the distribution becomes more concentrated around the first value. The entropy is 0 when $\beta = 1$, i.e. when the distribution is a Dirac distribution, and it approaches $\log(n)$ as β approaches 0, i.e. when the distribution becomes uniform. Choosing a suitable value for β is a

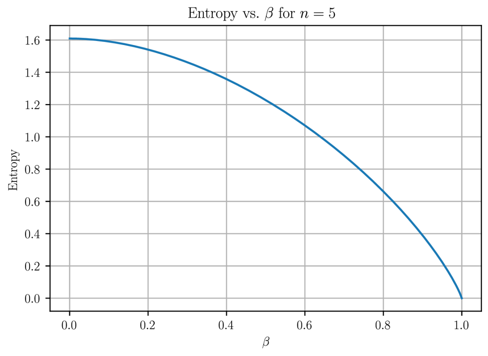


Figure 3.13: Relationship between the entropy $H_Q(\beta)$ and the parameter β for different values of β. As β increases, the entropy decreases, indicating a more concentrated distribution.

more straightforward way to control the concentration of the distribution around the first value. The impact of this choice can be visualized in Figure 3.14, where a distribution of $n = 5$ numbers summing to 1 is shown for different values of β. After choosing a proper β value for the dataset, we can filter out the paths that do not satisfy the condition:

$$H_i(\mathbf{R}_L) < H_Q(\beta) \tag{3.15}$$

This means that the computation is performed separately for each target node, and therefore using the definition of per-row entropy. The assumption that only one path is significant for each target node is not always true, and it can happen that multiple paths have more similar, high attention weights for the same target node. However, we remember that the training objective of the neural network regularizes the distribution of attention weights to be far from the uniform distribution, so we can expect that the network, properly trained, already assigns the highest attention weight to one edge per target node, except for few cases. Consequently, for each

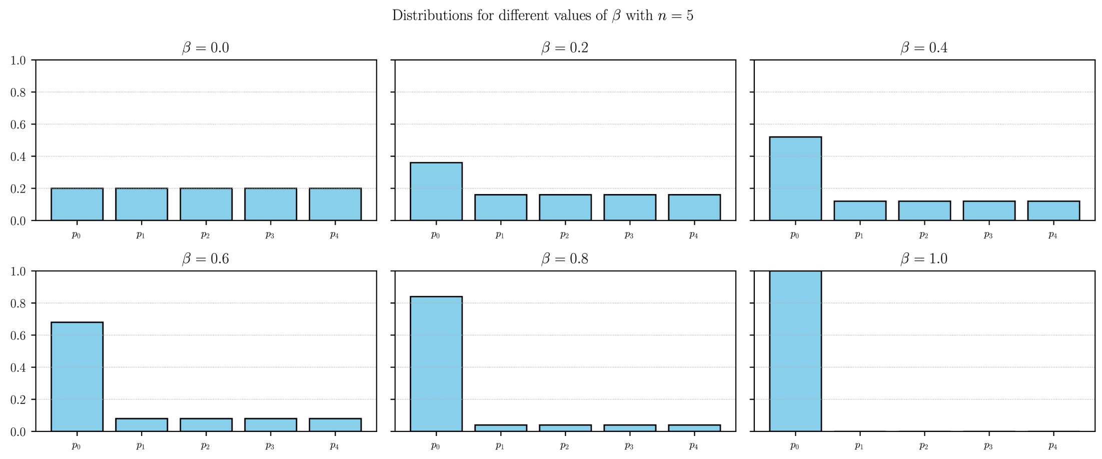


Figure 3.14: Impact of the parameter β on the distribution of values. As β increases, the distribution becomes more concentrated around the first value, leading to a lower entropy. p_i are just placeholders for the values of the distribution, and they are not related to the paths in the graph.

target node we select the source node linked to it through the path with the highest attention weight. Let:

$$P_i = \{p^L \mid \text{first}(p^L) = v_j \wedge \text{last}(p^L) = v_i\}$$

be the set of paths from v_j to v_i of length L, where i is one of the rows resulting from Equation 3.15. From this set, we can extract the path with the highest attention weight:

$$p_i = \arg\max_{p^L \in P_i} \prod_{l=1}^{L} \alpha_{p^L[l]}^{L-l+1}$$

3.3.6 Fine2coarse mapping

Now that we have one path for each target node, we want to assess the robustness of model predictions. In Section 3.1.2, we described the process of mesh refinement and how it can impact the attention mechanisms in the model. In particular, the latent representation of input data of a same temporal period can be completely different depending on the mesh resolution. This means that the paths we identified may not be stable across different levels of mesh refinement, and we need to evaluate their significance accordingly. For this purpose, assuming that the neural network has been trained on a dataset with a mesh refinement level $M > 0$, we can evaluate the model on input data with any mesh refinement M' such that $0 \leq M' \leq M$. We can also assume that the output of the selection of high-attention paths is represented through a binary mask B, that has the same shape of the correlation matrix $\mathbf{R}_L$, where each element is 1 if the corresponding path is candidate for significance and 0 otherwise. Since $\mathbf{R}_L$ has shape $(T \cdot S) \times (T \cdot S)$, and S depends on M', we cannot directly compare the masks obtained for different mesh refinement levels. To this purpose, we can set the mesh refinement level 0 as baseline and map any binary mask to the baseline. For simplicity, we start defining a matrix that maps spatial points in the refined mesh to the ones in the coarse mesh. Let $S^{(0)}$ be the number of spatial points in the coarse mesh, and $S^{(M')}$ be the number of spatial points in the M'-level refined mesh. We remember that the technique used to refine the spherical mesh is the midpoint triangulation, so starting from the coarse mesh, i.e. the 0-level refined mesh, each point generates 5 points in the 1-level refined mesh, as can be easily visualized in Figure 3.15. Again, each point in the 1-level refined mesh generates 5 points in the 2-level refined mesh, and so on. All the points in the $l + 1$-level refined mesh that are generated by a same point in the l-level refined mesh have the same distance from that point, and can be considered as *neighbors*. For simplicity, we include in these points also the coarser one, and we therefore define $\mathcal{N}_i^{l,l+1}$ as the set of the finer points generated by the i-th point in the l-level refined mesh, including the point itself. The *neighborhood*

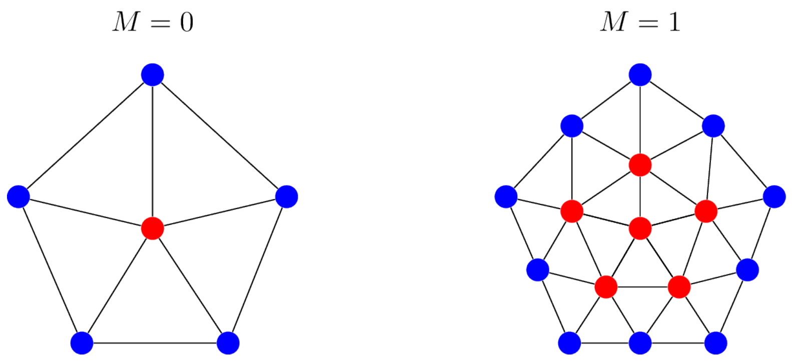


Figure 3.15: Midpoint triangulation of a spherical mesh. Each point in the coarse mesh generates 5 points in the refined mesh. In red is highlighted how one single point in the coarse mesh is triangularized in the refined mesh. M indicates the mesh refinement level.

grows as more refinement levels are involved, and we can generalize that:

$$\mathcal{N}_i^{a,b} = \bigcup_{l=a}^{b-1} \mathcal{N}_i^{l,l+1}, \quad \forall a < b$$

The cardinality of this set can be computed by considering that $\mathcal{N}_i^{a,b}$ exploration is a descendance tree, and so using the formula for the partial sum of a geometric series:

$$|\mathcal{N}_i^{a,b}| = \frac{r^{b-a+1} - 1}{r - 1} = \frac{5^{b-a+1} - 1}{5 - 1} = \frac{5^{b-a+1} - 1}{4}$$

where r is the branching factor of the tree, which in this case is 5. Assuming $a = 0$:

$$|\mathcal{N}_i^{0,1}| = \frac{5^2 - 1}{4} = 6$$
$$|\mathcal{N}_i^{0,2}| = \frac{5^3 - 1}{4} = 31$$

and so on. We can define the mapping matrix $\mathbf{M}^{(l)}$ from the l-level refined mesh to the coarse mesh as a matrix of shape $S^{(0)} \times S^{(l)}$. For each entry of the matrix:

$$\mathbf{M}_{i,j}^{(l)} \begin{cases} > 0 & \text{if } j \in \mathcal{N}_i^{0,l} \\ = 0 & \text{otherwise} \end{cases}$$

Since all points in $\mathcal{N}_i^{0,l}$ have the same distance from the i-th point in the coarse mesh, they should uniformly contribute to the coarse point. We can assume that the contribution of each point is a probabilistic event, and therefore each row in $\mathbf{M}_l$ should sum to 1:

$$\mathbf{M}_{i,j}^{(l)} = \begin{cases} \frac{1}{|\mathcal{N}_i^{0,l}|} & \text{if } j \in \mathcal{N}_i^{0,l} \\ 0 & \text{otherwise} \end{cases}$$

$\mathbf{R}_L$ is computed for each time step in the time window, but of course the structure of the mapping matrix $\mathbf{M}^{(l)}$ will always be the same. In order to apply the mapping to the binary mask B, we can use the Kronecker product[7] between the identity

[7]The Kronecker product is a mathematical operation that takes two matrices and produces a block matrix. It is defined as follows: given two matrices $\mathbf{A}$ of shape $m \times n$ and $\mathbf{B}$ of shape $p \times q$, the Kronecker product $\mathbf{A} \otimes \mathbf{B}$ is a matrix of shape $mp \times nq$, where each element a_{ij} of $\mathbf{A}$ is multiplied by the entire matrix $\mathbf{B}$, resulting in a block matrix [75]:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \cdots & a_{2n}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & a_{m2}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{bmatrix}$$

matrix $\mathbf{I}_T$, where T is the number of time steps in the time window, and the mapping matrix $\mathbf{M}^{(l)}$:

$$\mathbf{M}^{(l),T} = \mathbf{I}_T \otimes \mathbf{M}^{(l)} = \begin{bmatrix} \mathbf{M}^{(l)} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{M}^{(l)} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{M}^{(l)} \end{bmatrix}$$

The resulting matrix $\mathbf{M}^{(l),T}$ has shape $(T \cdot S^{(0)}) \times (T \cdot S^{(l)})$. Let $B^{(l)}$ be the binary mask of shape $(T \cdot S^{(l)}) \times (T \cdot S^{(l)})$. We can now project $B^{(l)}$ to the coarse mesh in the following way:

$$\Pi = \mathbf{M}^{(0),T} \times B^{(l)} \times \left(\mathbf{M}^{(0),T}\right)^\top$$

The both-sided multiplication ensures that both the rows and columns of the binary mask are correctly mapped to the coarse mesh, and each item in the resulting matrix can be interpreted as how many paths from the l-level refined mesh contribute to the corresponding point in the coarse mesh. This means that the output is not binary anymore. However, we can assume that just one contribution from the l-level refined mesh is enough to consider the corresponding point in the coarse mesh as significant, so we can define the items of the binary mask $\mathbf{B}^{(0),l}$ as:

$$\mathbf{B}^{(0),l}_{i,j} = \begin{cases} 1 & \text{if } \Pi_{ij} > 0 \\ 0 & \text{otherwise} \end{cases}$$

and this output is specific for the l-level refined mesh and can be compared with the binary masks obtained from other levels of mesh refinement.

3.3.7 Significance test

There are many ways to compare binary masks. For example, we could compute the union:

$$\bigcup_{l=0}^{M} \mathbf{B}^{(0),l}$$

which represents the points that are significant in at least one level of mesh refinement. However, this does not guarantee that the points are significant across all levels of mesh refinement, and therefore it does not provide a robust assessment of the model predictions. We could think about taking the average, but this would not be meaningful in the context of binary masks. We could compute the cross-entropy between the binary masks, but we are not interested in assessing the reliability of model predictions at a certain refinement level with respect to the others, and we also do not have any ground truth to compare with. We can define a Pearson correlation metric [24] between all masks and, since the masks are binary, it corresponds to the Phi coefficient [76]. We introduce:

$$\begin{aligned}
n_{11}^{(l_1,l_2)} &= \#\{(i,j) \mid \mathbf{B}_{i,j}^{(0),l_1} = 1 \land \mathbf{B}_{i,j}^{(0),l_2} = 1\} \\
n_{10}^{(l_1,l_2)} &= \#\{(i,j) \mid \mathbf{B}_{i,j}^{(0),l_1} = 1 \land \mathbf{B}_{i,j}^{(0),l_2} = 0\} \\
n_{01}^{(l_1,l_2)} &= \#\{(i,j) \mid \mathbf{B}_{i,j}^{(0),l_1} = 0 \land \mathbf{B}_{i,j}^{(0),l_2} = 1\} \\
n_{00}^{(l_1,l_2)} &= \#\{(i,j) \mid \mathbf{B}_{i,j}^{(0),l_1} = 0 \land \mathbf{B}_{i,j}^{(0),l_2} = 0\}
\end{aligned}$$

where l_1 and l_2 are the levels of mesh refinement. The values $n_{11}^{(l_1,l_2)}$, $n_{10}^{(l_1,l_2)}$, $n_{01}^{(l_1,l_2)}$, and $n_{00}^{(l_1,l_2)}$ represent the number of points in the binary masks that agree or disagree across the two levels of mesh refinement. The Phi coefficient is a measure of association between two binary variables, and the Phi coefficient matrix Φ is defined as a matrix of elements:

$$\Phi_{ij} = \frac{n_{11}^{(i,j)} n_{00}^{(i,j)} - n_{10}^{(i,j)} n_{01}^{(i,j)}}{\sqrt{(n_{11}^{(i,j)} + n_{10}^{(i,j)})(n_{11}^{(i,j)} + n_{01}^{(i,j)})(n_{00}^{(i,j)} + n_{10}^{(i,j)})(n_{00}^{(i,j)} + n_{01}^{(i,j)})}}$$

assuming that the denominator is 0, otherwise $\Phi_{ij} = 0$. This matrix could be useful to assess how much the binary masks agree, but we are actually interested in the predictions that are common to all masks. Therefore, we just compute the intersection between the binary masks:

$$\mathbf{B}^{(0)} = \bigcap_{l=0}^{M} \mathbf{B}^{(0),l}$$

This intersection gives us a binary mask that contains only the points that are significant across all levels of mesh refinement, and therefore can be considered as robust. Please note that in Section 3.1.5 we defined the spatial structure of input data in a way that self-loops are not possible, but cyclic paths are not avoided: if a path starts in node $v_{s,t}$, it could end in node $v_{s,t+k}$, where k is the length of the path. Moreover, it could happen that a non-cyclic path in the refined mesh contributes to a cyclic path in the coarse mesh, if source and target node in the path in the fine mesh are mapped to the same point in the coarse mesh. Therefore, we will likely find cyclic contributions in the final binary mask $B^{(0)}$. We can reasonably discard these contributions, because they do not represent a significant teleconnection. If we switch to an edge-list representation of the binary mask:

$$\{(v_{s,t}, v_{s',t'}) \mid \mathbf{B}_{[s',t'],[s,t]}^{(0)} = 1\}$$

we can remove the cyclic paths by filtering out the edges where $s = s'$:

$$\mathbf{E}^{(0)} = \{(v_{s,t}, v_{s',t'}) \mid \mathbf{B}_{[s',t'],[s,t]}^{(0)} = 1 \land s \neq s'\}$$

This result is the list of significant teleconnections, and the procedure described in this section can be seen as a significance test for the teleconnections identified by the model. Assuming that this procedure is applied to the output of a GAT network for every test sample, we can produce proper statistics, such as the number of teleconnections and their distribution across time and space. For visualization purposes, nodes can be converted back from mesh to grid representation, so that the teleconnections can be plotted on a grid map. We can also compute the intersection over union (IoU), a.k.a. Jaccard similarity [77], which is usually defined for pairs of matrices:

$$\mathrm{IoU}(\mathbf{B}^{(0),l_1}, \mathbf{B}^{(0),l_2}) = \frac{\sum_{i,j} \min(\mathbf{B}^{(0),l_1}_{i,j}, \mathbf{B}^{(0),l_2}_{i,j})}{\sum_{i,j} \max(\mathbf{B}^{(0),l_1}_{i,j}, \mathbf{B}^{(0),l_2}_{i,j})}$$

Where the min operation corresponds to the intersection and the max operation corresponds to the union. However, we can extend this definition to directly compare all the matrices:

$$\mathrm{IoU}(\mathbf{B}^{(0)}) = \frac{\sum_{l=0}^{M} \sum_{i,j} \min(\mathbf{B}^{(0),l}_{i,j})}{\sum_{l=0}^{M} \sum_{i,j} \max(\mathbf{B}^{(0),l}_{i,j})}$$

A high IoU value indicates that the model predictions are consistent across different levels of mesh refinement, while a low IoU value indicates that the model predictions are not robust and may depend on the mesh resolution.

We now briefly recap the steps of the proposed approach:

1. The neural network is trained on a dataset with a certain mesh refinement level $M > 0$.

2. The model is evaluated on test data with any mesh refinement level M' such that $0 \leq M' \leq M$.

3. The attention matrix $\mathbf{A}(m)$ is computed for each layer m of the GAT network.

4. The correlation matrix $\mathbf{R}_L(k)$ is computed for each layer L and path length k.

5. The entropy $H_i(\mathbf{R}_L(k))$ is computed for each target node i and layer L.

6. The paths are filtered based on the entropy condition $H_i(\mathbf{R}_L(k)) < H_Q(\beta)$, where β is a parameter that controls the concentration of the distribution of attention weights.

7. For each target node i, the path with the highest attention weight is selected.

8. The binary mask $B^{(l)}$ is computed for each level of mesh refinement l.

9. The binary masks are mapped to the coarse mesh using the mapping matrix $\mathbf{M}^{(l),T}$.
10. The binary masks are intersected to obtain the final binary mask $\mathbf{B}^{(0)}$ that contains only the points that are significant across all levels of mesh refinement.
11. The cyclic paths are removed from the final binary mask $\mathbf{B}^{(0)}$ to obtain the edge list $\mathbf{E}^{(0)}$ of significant teleconnections.

This concludes the exploration of the methodological aspects of the proposed approach. In the next chapter, we will present the results obtained by applying this method to a real-world dataset.

Chapter 4

Results

In this chapter, we present the results of the experiments conducted to evaluate the proposed approach for teleconnection discovery. We begin by detailing the experimental setup, including the dataset used and the analysis performed. Subsequently, we discuss the application of the methodology previously described, and we highlight our findings.

4.1 Experimental setup, dataset choice and analysis

4.1.1 Environment and tools

The experiments were conducted using two different environments: a local and a remote machine. The local machine was equipped with an Apple M1 @ 3.20 GHz processor, 8 cores and 8 GB of RAM. The remote machine was a server with an Intel Xeon Gold 6130 CPU @ 2.10 GHz processor, 16 cores, 64 GB of RAM, and a GPU NVIDIA Tesla V100-PCIE with 16GB of VRAM. The project was developed using Python 3.11.10 [78] and the following libraries: `numpy`, `torch`, `pandas`, `torch_geometric`, `xarray`, `lightning`, `plotly`, `matplotlib`, `basemap`, `wandb`, `netcdf4`, `kaleido`, `scikit-learn`, `trimesh`,
`cartopy`, and `networkx`. The dataset was processed through *Xarray* [79], which allows for efficient manipulation of multi-dimensional arrays and supports NetCDF [80] files. Data modelling through graphs was performed using *PyTorch Geometric* [61], which provides implementations of various graph neural network layers and utilities for working with graph-structured data. In fact, it was also employed to build the neural network itself, together with *PyTorch* [81], a widely used deep learning framework. Models were trained using the *PyTorch Lightning* framework [82], which simplifies the training process and allows for easy integration with

Weights and Biases [83] for experiment tracking and visualization. Charts were generated using *Plotly* [84], a powerful library for creating interactive visualizations, and *Matplotlib* [85], a widely used library for static visualizations. For geospatial mapping and analysis, *Basemap* and *Cartopy* were used. Mesh-related operations and 3D geometry processing were handled using *Trimesh*, while *NetworkX* was used for additional graph-related utilities and visualizations.

4.1.2 Dataset

After a comparison of the available datasets, the one chosen for the experiments is the ERA5 reanalysis dataset [86, 87], because it is one of the most complete in terms of variables and temporal coverage.

ERA5, developed by ECMWF and released through the Copernicus Climate Change Service (C3S), is the fifth-generation ECMWF reanalysis and represents a significant advancement over its predecessors. According to [88], ERA5 provides hourly estimates of a broad range of atmospheric, ocean-wave, and land-surface variables from 1940 to the present, with updates provided on a daily basis and a latency of approximately five days. The data assimilation system[1] used is a 12-hour four-dimensional variational method[2] (4D-Var), integrated within ECMWF's Integrated Forecasting System (IFS), and enhanced with a 10-member Ensemble of Data Assimilations (EDA). This ensemble approach enables the generation of uncertainty estimates every three hours, helping users assess the confidence in the data based on variability among ensemble members. ERA5 data are offered at high spatial resolution: $0.25° \times 0.25°$ for the main reanalysis and $0.5° \times 0.5°$ for uncertainty fields ($1° \times 1°$ for ocean-wave data). It includes 137 vertical model levels, extending from the surface to 0.01 hPa (approximately 80 km altitude), which facilitates detailed analysis of atmospheric processes from the troposphere to the mesosphere [86]. The reanalysis is available in multiple formats, including General Regularly distributed Information in Binary form (GRIB) [91] and NetCDF[3], and can be accessed via the Copernicus Climate Data Store (CDS). ERA5 comprises

[1]The data assimilation system is a set of algorithms and techniques used to combine observational data with numerical models to produce a consistent and accurate representation. In other words, real-time observations and model forecasts are combined to create a best estimate of the current state [89].

[2]4D-Var is a data assimilation technique that uses a variational approach to minimize the difference between the model forecast and the observations, taking into account the uncertainties in both. It considers four dimensions: three spatial dimensions (latitude, longitude, and altitude) and time [90].

[3]NetCDF (Network Common Data Form) is a set of software libraries and machine-independent data formats that support the creation, access, and sharing of array-oriented scientific data. Data in NetCDF is self-describing, portable, scalable, appendable, sharable, archivable [80].

four main data subsets: hourly and monthly products, each provided for single levels (representing surface and near-surface conditions) and pressure levels (representing upper-air conditions). Pre-computed monthly averages are available for convenience, though ensemble mean and spread are not included in these summaries.

The CDS allows selecting the desired variables, the time range, the geographical area and the data format. The dataset was downloaded in the early December 2024. Considering the general purpose of discovering and analyzing climate teleconnections, a set of variables has been selected to consider the overall climate system, including atmospheric, oceanic and land surface variables. In Table 4.1, for each chosen variable the name, the description and the unit of measurement are reported [86]. For what concerns time range, data was downloaded from January

Variable	**Description**	**Unit**
`d2m`	2 m dewpoint temperature	K
`msl`	Mean sea level pressure	Pa
`skt`	Skin temperature	K
`sp`	Surface pressure	Pa
`sst`	Sea surface temperature	K
`t2m`	2 m temperature	K
`tp`	Total precipitation	m
`u10`	10 m u-component of wind	m/s
`v10`	10 m v-component of wind	m/s

Table 4.1: List of selected ERA5 variables used in the experiments.

1st, 1940 to December 1st, 2023, and, for each month for each year, the first day at 00:00 UTC was selected. The selected geographical area is the whole globe. For portability and ease of use, the data was downloaded in NetCDF format and regridded to a resolution of 10 degrees, to analyze big-scale climate phenomena: although such regridding produces a considerable loss of detail, it allows to reduce the computational cost of the experiments and to focus on the most relevant climate teleconnections. Moreover, to properly balance the dataset size for training, validation and testing, only data until December 2022 was used for the experiments. The regridding was performed using the *Xarray* library and following a bilinear interpolation method, which is a common technique for resampling gridded data: it calculates the value of a new grid point by taking a weighted average of the four nearest grid points in the original grid, thus preserving the overall trend and structure of the data while reducing its resolution. However, it is important to note that this method may introduce some artifacts and smoothing effects, especially in areas with high variability or sharp gradients. Also, it is not always the appropriate technique: the majority of variables considered in the experiments are mean-preserving, meaning that the average value of the variable is preserved during the regridding process, but some variables, such as `tp`, are additive, so during the regridding process the result is the sum of the values of the original grid points, which is not mean-preserving. For simplicity, every variable was regridded using the same method, because the very coarsed grid is intentional to show the climatological characteristics of a geographical area, rather than the local variability of the variable. The regridded dataset was then saved in NetCDF format for further processing and analysis.

4.1.3 A data example

Data in ERA5 has a lot of expressive power when it comes to representing climate phenomena. In particular, it allows to analyze how different variables interact with each other, and how they change over time and space. In Figure 4.1, the monthly average of all the variables for a time step taken as example (January 1990) is shown, to give an idea of the data structure and the spatial distribution of the variables. The figure shows the global distribution of each variable, highlighting the differences in temperature, pressure, and precipitation patterns across the globe. The data is presented in a grid format, with each cell representing a specific geographical area and the color intensity indicating the value of the variable at that location. What can be observed is that the dataset is generally clean, with only some `NaN` value in the SST variable, because it is only computed over the ocean and not on the land. It is also evident that using the same coarsening strategy for all the variables is not always the best choice, since some variables are more sensitive to spatial resolution than others. For example, temperature and pressure are generally well represented at a coarse resolution, while precipitation is more sensitive to spatial resolution and requires a finer grid to capture local variations. In fact, if we look at the precipitation variable, we can see that it is more concentrated in specific areas, such as tropical regions and coastal areas, where it is more likely to occur, but values are very low, because rain happens only in specific points in space and time, not for an entire region or month, so averaging the values is not the proper way to represent this information. However, for the experiments, normalized anomalies are considered, so the magnitude of variables does not cause any loss of information. Wind speed variables show that air flows are generally stronger in the tropics and weaker in the polar regions. Other variables reflect elementary notions on climatology: pressure is lower on the mountains and higher in the valleys, temperature is lower at higher altitudes, and so on. However, it is not so immediate to visualize the relationships between the variables or geographical regions.

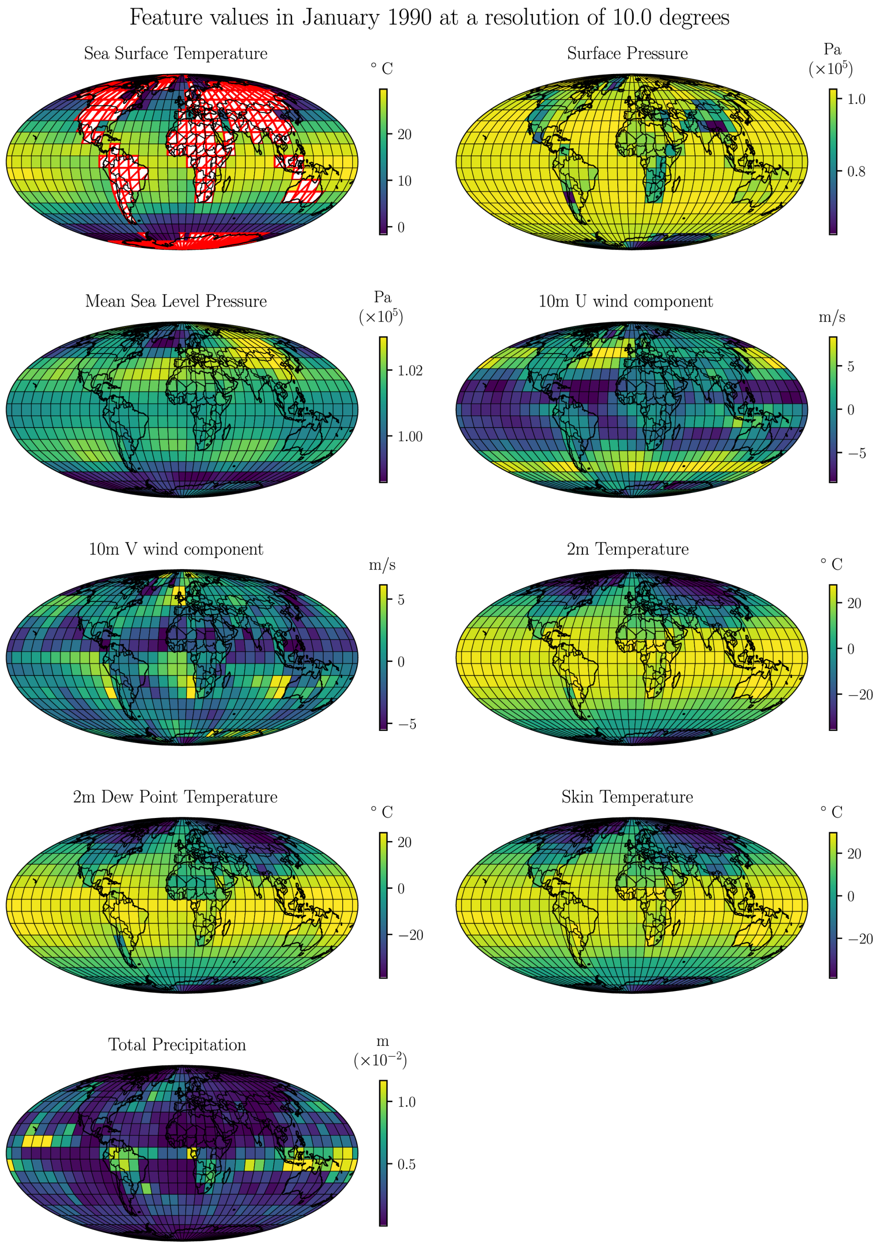


Figure 4.1: Monthly average of the selected ERA5 variables for January 1990. In case of `NaN` values, the cell is colored in white, with red stripes. Each subplot represents a different variable, with the color scale on the right indicating the value range. Unit of measurement is shown in the legend of each subplot. In case of temperature, values are shown in Celsius for better readability, but the original values are in Kelvin. The grid resolution is 10×10 degrees, covering the whole globe.

4.1.4 Statistical distribution of the variables

To better understand the dataset, it is useful to analyze the statistical distribution of the variables over time. In Figure 4.2, the monthly average of each variable is shown for the entire time range considered in the experiments, from January 1940 to December 2022. We can make some considerations on how the variables differ across the globe. For instance, temperature variables show a general increase over time, with some fluctuations due to seasonal variations; precipitation in Europe is much less than in any other macro-area; winds are stronger in Australia; sea level pressure in the northern hemisphere is generally higher than in the southern hemisphere.

4.2 Graph construction and dataloader

4.2.1 Climatology

Once the dataset was regridded and variables selected, a preprocessing step was needed to make data suitable to be fed to the neural network. First of all, climatology was computed. To take into account recent climate change phenomena, the time period chosen for the climatology computation is from January 1970 to December 1999 and therefore lasts 30 years. Each NetCDF file contains one time step (month of a specific year), 19 distinct latitudes and 36 distinct longitudes, for a total of $1 \times 19 \times 36 = 684$ rows. All the NetCDF files belonging to that period were merged into a unique `DataFrame` object, which leads to $(30 \times 12) \times 684 = 246{,}240$ rows. Data was then grouped by latitude, longitude and month, regardless of the year, resulting into $12 \times 19 \times 36 = 8{,}208$ groups. For each group, the mean and standard deviation were computed for each variable, and the resulting matrices were saved as tensors to be employed for data anomaly conversion.

4.2.2 Coordinates encoding

Tensor transformation is applied in the same way to both data and climatology. Each tensor has N rows and D columns. N depends if the tensor is generated by the data or the climatology: if from data, $N = 684$, otherwise $N = 8{,}208$. Instead, $D = C + F$ is fixed, where C is the number of coordinates and F is the number of features. Coordinates are, in order:

1. Normalized year
2. Sine component of the month
3. Cosine component of the month

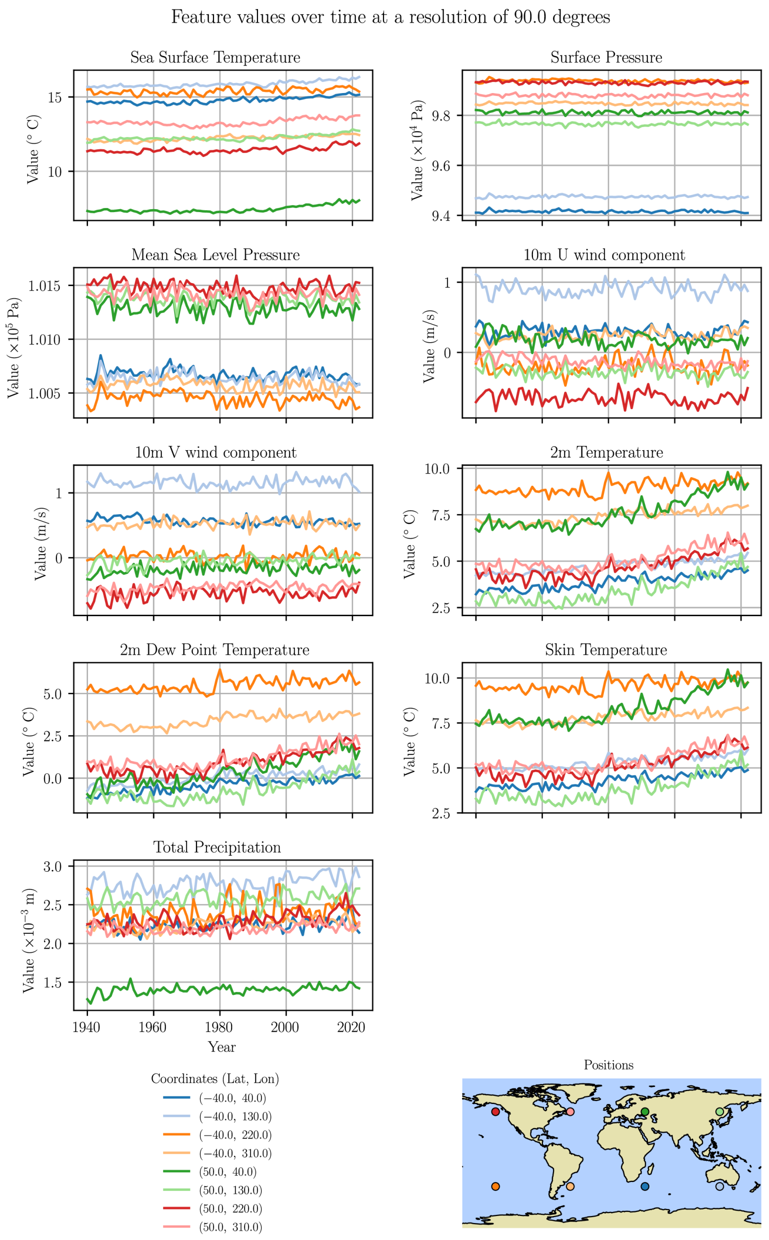


Figure 4.2: Statistical distribution of the selected ERA5 variables over time. Each color represents measurements collected at different macro-areas. The x-axis represents the time, while the y-axis represents the value of the variable. The grid resolution is 90 × 90 degrees, covering the whole globe, to show trends of macro-areas in a clear way. Positions should be intended as entire areas, not single points.

4. Sine component of the latitude
5. Cosine component of the latitude
6. Sine component of the longitude
7. Cosine component of the longitude

Since the period considered for the experiments is from January 1940 to December 2022, the normalized year is computed as follows:

$$\frac{\text{year} - 1940}{2023 - 1940} = \frac{\text{year} - 1940}{83}$$

Therefore, $C = 7$. As from Section 4.1.2, $F = 9$ is the number of features, so $D = 16$.

4.2.3 Grid2mesh

We remember from Section 4.1.3 that there is always chance for missing values in data. Therefore, we substitute `NaN` values with the average of non-missing values of the same feature in the tensor.

Tensors all present the same coordinates but differ in the values of the features. This means that the *grid2mesh* mapping can be done indifferently on any of the tensors. Starting from the cosine and sine components of latitude and longitude, we recover 3D cartesian coordinates, obtaining an auxiliary grid tensor of shape $N \times 3$, where N again depends if the tensor is generated by the data or the climatology. We generate an icosahedron mesh and we refine it up to the desired level: to the purpose of this work, we consider 2 successive refinements. Each time that the mesh is refined, we keep track of the fine mesh points generated by each coarse point, in order to define the *coarse2fine* mapping to be used subsequently. The mesh object contains vertices and faces, stored in proper matrices: the vertex matrix has shape $V \times 3$, because points are known in 3D cartesian coordinates, and the faces matrix has shape $F \times 3$[4], because each face is defined by 3 vertices. Experiments have been conducted using 0, 1 and 2 levels of mesh refinement, in order to assess the impact of the mesh resolution on the model performance, but at the same time to keep the computational cost manageable. In case of no refinement, $V = 12$ and $F = 20$, after the first refinement, $V = 42$ and $F = 80$, while after the second refinement, $V = 162$ and $F = 320$. While faces are used for the mesh refinement and visualization, vertices are used to define the graph nodes. A spherical distance

[4]Please do not confuse the F in the faces matrix with the F in the features, they are different.

is then computed between each row in the mesh vertex matrix and each row in the grid tensor. These distances are stored as a tensor of shape $N \times V$, and the closest mesh node to each grid point is determined by an `argmin` operation, which returns an array of shape N, where each element is the index of the closest mesh node to the corresponding grid point.

At this point, for each input tensor (data or climatology) we can perform the *grid2mesh* mapping: we isolate the feature columns from the tensor, we multiply each row by the cosine component of the corresponding latitude (latitude weights), and, through an `index_add` operation, we sum for each mesh point the values of the features of the corresponding grid points, we do the same thing for the latitude weights, and we divide the summed features by the summed latitude weights, such that we can average features over the cosine of the latitude and obtain a proper mesh representation of the data. The resulting tensor has shape $V \times F$. We finally build a tensor by packing the following coordinates:

1. Normalized year
2. Sine component of the month
3. Cosine component of the month
4. x coordinate of the mesh point
5. y coordinate of the mesh point
6. z coordinate of the mesh point

for a total of $C = 6$ columns, which, added to the $F = 9$ adapted features, leads to a mesh representation of shape $V \times 15$. Given this tensor for both data and climatology, we can now compute the anomalies. We should just calculate the difference between the data and climatology tensors. However, for numerical stability purposes, we prefer to use the climatology mean and standard deviation to normalize the data. Therefore, we subtract the mean and divide by the standard deviation for each feature.

4.2.4 Time windowing and edges construction

Once we have the mesh representation for anomalies, we want to construct graphs where information from consecutive time steps is included. After some experiments, we found that a time window of $T = 13$ months, because we want to let edges make $T - 1 = 12$ traversals in time, it is also a good compromise between the amount of information and the computational cost. Therefore, we concatenate the anomalies of 13 consecutive months through a batching strategy, which leads to a tensor of shape $(13 \cdot V) \times 15$. We need edges, which connect each node in a specific time step to its spatial 2^m-neighbors in the following time step, where the definition of *neighbors* is given in Section 3.1.5 and m is the mesh refinement level. Neighbors are stored as a key-value dictionary, where the key is the index of the spatial node and the value is a list of indices of the neighboring nodes. Starting from this dictionary, which only includes spatial information, we integrate temporal information to cover the entire time window. Therefore, edges are computed using as source the key of the dictionary and as target any of the values associated to that key. This step is repeated for each value of the key, for each key and for each time step in the window. Please note that nodes in the first time step do not have any incoming edge, and edges are the same for every time step, so they are repeated $T - 1$ times. Therefore, the resulting edges are stored in a tensor of shape $2 \times (12 \cdot E)$, where E depends on the number of neighbors, and therefore on the mesh refinement level. We also need edge features, which, as from Section 3.1.6, include a spatial distance and a temporal distance. For the spatial distance, which is the same regardless of the time step, we compute a distance matrix $V \times V$ between all nodes of the mesh, and for each edge we properly reference the values of this matrix by using the source and target indices. For the temporal distance, we simply compute the difference between the time steps of the source and target nodes, which is always 1 in this case, because we are considering consecutive time steps. The resulting edge features tensor has shape $(12 \cdot E) \times 2$, where the first column is the spatial distance and the second column is the temporal distance. Finally, the graph is constructed using the following tensors:

- `x`: the node features tensor of shape $(13 \cdot V) \times 15$, which contains the coordinates and the features of each node;

- `edge_index`: the edge index tensor of shape $2 \times (12 \cdot E)$, which contains the source and target indices of each edge;

- `edge_attr`: the edge features tensor of shape $(12 \cdot E) \times 2$, which contains the spatial and temporal distances for each edge.

For numerical stability purposes, both the `x` and `edge_features` tensors are normalized. In Table 4.2, the dimensions of the tensors as a function of the mesh refinement level are reported. The first column indicates the mesh refinement level in meters, while the other columns show the shapes of the tensors. As we can see, the number of edges increases significantly with the mesh refinement level, because of the 2^m-neighborhood rule and the fact that edges at a certain resolution include edges at lower resolutions.

m	V	E	`x`	`edge_index`	`edge_attr`
0	12	60	156×15	2×720	720×2
1	42	$600 + 60 = 660$	546×15	$2 \times 7{,}920$	$7{,}920 \times 2$
2	162	$8040 + 660 = 8{,}700$	$2{,}106 \times 15$	$2 \times 104{,}400$	$104{,}400 \times 2$

Table 4.2: Tensor dimensions as a function of mesh refinement level. m is the mesh refinement level, V, E are the number of vertices and edges assuming a single time step. Tensor shapes are computed considering the entire time window.

4.2.5 Dataloader

To handle the graph data and feed it to the neural network, a *PyTorch Geometric* dataloader was created. The dataloader takes as input the graph tensors and constructs a dataset object, which can be used to iterate over the data in batches. When training a neural network, we usually divide the dataset into three subsets: training, validation and testing. The training set is used to train the model, the validation set is used to tune the hyperparameters and the testing set is used to evaluate the model performance on unseen data. This is necessary to avoid overfitting and to ensure that the model generalizes well to new data. In this case, we decided to use a time-based split, i.e. once generated the graphs for all time windows, starting from the first month of the dataset and shifting by one month at a time, we split the dataset into three subsets based on the time period in which the graph starts. The training set includes all the graphs where the first time step is between January 1940 and December 1989; for the validation set, the range is between January 1991 and December 2000; for the test set, the graphs start between January 2002 and December 2021. The fact that some months are *skipped* is intentional, because of the length of the time window considered: a time window of 13 steps that starts in the beginning of December 1989 - the last one of the training set - ends in the end of December 1990, and the following one - the first one of the validation set - starts in January 1991. Therefore, the entire time period of the dataset is covered, but it is ensured that there is no overlapping between the subsets, which is crucial to avoid that the results are biased. In Figure 4.3, the temporal extension of the training, validation and testing sets is shown graphically, with the time periods and the number of samples for each subset. The model is trained on past data and evaluated on future data, which is a common practice in time series analysis. The dataloader also allows for batching and shuffling of the data, which is useful for training the model in mini-batches and improving generalization. Since we work with graph-structured data, the batching strategy consists in generating, from more graphs (`Data` objects), a unique graph (`Batch` object) where `x` and `edge_index` and `edge_attr` are concatenated over all graphs. However, indices in `edge_index` are adjusted to account for the concatenation, so that they refer to the correct nodes in the batch. In particular, since node indices start from 0 for each graph, *PyTorch Geometric* offsets the indices in `edge_index` for each graph based on the cumulative number of nodes in previous graphs. The `Batch` object also contains a `batch` attribute, which is an array of length equal to the number of nodes in the batch, where each element indicates the index of the graph to which the corresponding node belongs. Batching is enabled for both training, validation and testing sets, to better exploit the computational resources and to speed up the training process. For the experiments, batch size was set to $B = 16$. Instead, shuffling is only enabled for the training set, to ensure that

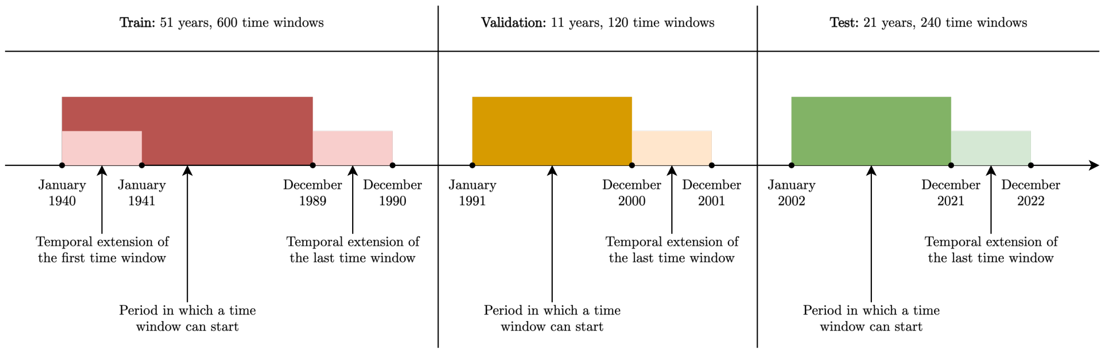

Train: 51 years, 600 time windows
Validation: 11 years, 120 time windows
Test: 21 years, 240 time windows
January 1940
January 1941
December 1989
December 1990
January 1991
December 2000
December 2001
January 2002
December 2021
December 2022
Temporal extension of the first time window
Temporal extension of the last time window
Temporal extension of the last time window
Temporal extension of the last time window
Period in which a time window can start
Period in which a time window can start
Period in which a time window can start

Figure 4.3: Extension of training, validation and testing sets over time. Each color represents a different subset: red for training, orange for validation, and green for testing. The left side of a rectangle indicates the first time step, included from the beginning of that month, whereas the right side indicates the last time step, included until the end of that month. For each subset, the number of years (including the last month of the last time window) is reported, as well as the total number of graphs in that subset.

the model does not learn any temporal patterns from the data, but only from the features and edges. The validation and testing sets are not shuffled, to preserve the temporal order of the data and to evaluate the model performance on unseen data.

4.3 Neural network

After defining the graph construction mechanism, the neural network architecture was built. We use a VGAE architecture, with a GATv2 encoder and an inner product decoder.

4.3.1 Model

The encoder is made of $L + 2$ GATv2 layers, where L is defined when the encoder is instanciated, and the 2 additional layers are required by the VGAE architecture to compute the mean and log variance[5] of the latent space. Experiments have been conducted using from $L = 2$ to $L = 6$ layers, to observe how model performance changes and if oversmoothing and oversquashing damage results. For each layer, the number of input and output channels is provided. We always assume that there is one attention head and self-loops are excluded from attention computation. For the first layer, which takes as input a graph object, input dimension is the same as the number of features in the node features tensor, i.e. 15. The output dimension of the first layer is set to 256, to allow the latent space to have a good representation power. This is the latent space dimension, and therefore is preserved in the following layers, since we want our model to learn paths traversing layers and edges, so, for all the remaining layers, including the ones responsible for computing the mean and log variance of the latent space, input dimension is equal to output dimension. When input data enters the encoder, `x`, `edge_index` and `edge_attr` tensors are passed to the first layer, which computes the attention weights for that layer, of size E, and the output features for each node, of shape $(13 \cdot V) \times 256$. These ones are then passed to a ReLU function and then to the next layer. After the L-th layer, the ReLU function is not activated and the output features are still a tensor of shape $(13 \cdot V) \times 256$, At this point, both the mean and log variance layers take this tensor as input and both produce a representation of the same shape. With the reparametrization trick, the two tensors are combined into a single latent space tensor of the same shape. The resulting latent space tensor is then passed to the decoder. The decoder indexes the latent space tensor using the `edge_index`

[5]We know from theory that the VGAE computes $\log \sigma^2$, but for numerical stability purposes we prefer to compute $\log \sigma$ and then square it, so that we can avoid numerical issues when computing the exponential function.

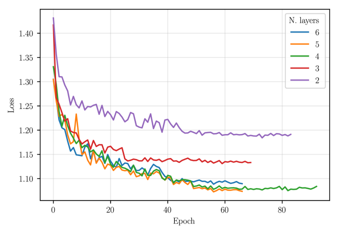
tensor, which contains the source and target indices of each edge, and computes the inner product between the features of the source and target nodes in the latent space. This results in a tensor of shape $12 \cdot E$, which represents the predicted edge weights for each edge in the graph. Finally, a sigmoid function is applied to the predicted edge weights, to obtain a probability distribution over the edges.

4.3.2 Loss function, backpropagation and metrics

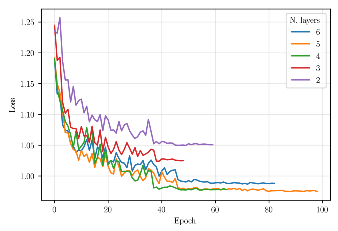
In each step of the training, validation or testing, input data is passed to the model and then outputs are processed to compute the loss and the metrics.

First of all, we take the output of the decoder to compute the reconstruction loss on positive edges. Then, we perform a negative sampling from positive edges to obtain negative edges, that have the same shape of positive edges, because we want a perfect class balance. We use negative edges with `z` to compute the reconstruction loss on negative edges. We sum the two losses to obtain the total reconstruction loss. The tensors of mean and log variance are used to compute the KL loss. The VGAE loss is then the sum of the reconstruction loss and the KL loss, which is divided by the number of nodes in the graph. For what concerns the entropy loss, in this case, separately for each of the L layers, we compute the entropy of attention weights, which is a scalar. Then, the L entropies are averaged to compute the entropy loss, which is multiplied by the hyperparameter λ. After some experiments, we found out that $\lambda = 0.5$ is a good value to balance the contributions of the various losses. In fact, the reconstruction loss and the KL loss (VGAE loss terms) are usually smaller than the entropy loss, and the difference can be even one order of magnitude. However, training the model against the VGAE loss is easier than training it against the entropy loss, because the latter is more sensitive to the attention weights and it is more difficult to optimize. Therefore, the entropy loss must give a higher contribution to the total loss, but not too high, in a way that the order of magnitude of both loss terms is similar. Finally, the total loss is computed as the sum of the reconstruction loss, the KL loss and the entropy loss. All loss terms are scalars, so the total loss is also a scalar.

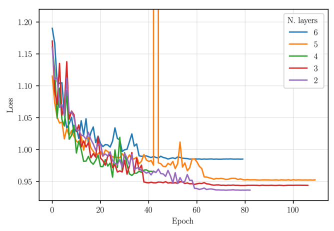
Loss is needed for backpropagation on the model parameters. This is performed using the Adam optimizer (please see Appendix C.3.3 for more details on how it works) with a learning rate of 0.001 and no weight decay. The scheduler is `ReduceOnPlateau`, which reduces the learning rate by a factor of 0.1 if the validation loss does not improve for 5 epochs. The model is trained for a maximum of 400 epochs, but training stops early if the validation loss does not improve for 10 epochs. In Figure 4.4, the validation loss curves over epochs for different loss terms and mesh refinement levels are shown. The curves show how the loss changes over time, and how the model converges to a minimum loss value. We know that the total loss is dominated by the entropy loss, so the curves for the entropy loss



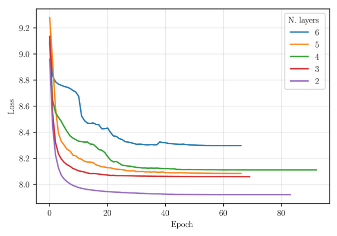
(a) VGAE loss, $m = 0$



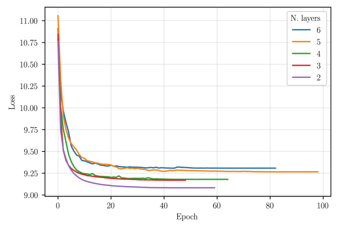
(b) VGAE loss, $m = 1$



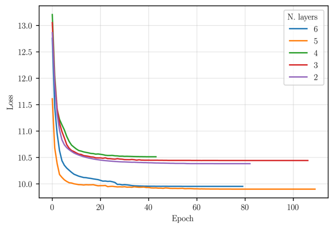
(c) VGAE loss, $m = 2$



(d) Entropy loss, $m = 0$



(e) Entropy loss, $m = 1$



(f) Entropy loss, $m = 2$

Figure 4.4: Validation loss curves over epochs for different loss terms, mesh refinement levels and number of layers.

are more stable and have a lower variance than the curves for the VGAE loss. The curves for the VGAE loss show more fluctuations, but we can take them as good because they are likely due to the randomness introduced by the reparametrization trick. In general, the VGAE loss converges at a good value, but the difference with respect to the initial value is not so high. We should pay attention to Figure 4.4c, because for $L = 5, m = 2$ at epoch 43 the loss diverges significantly, reaching the value of 1,406. This is an anomaly that comes from the KL divergence, and is probably due to a numerical issue when computing the `logstd` tensor under certain conditions. In Figure 4.4a, the best performing configuration (lowest value) is for $L = 5$, which is true also for Figure 4.4b. Instead, in Figure 4.4c, the loss is minimized for $L = 2$. For what concerns the entropy loss, $L = 2$ is the best performing configuration for Figure 4.4d and Figure 4.4e, while for Figure 4.4f the optimal choice is $L = 5$. We should take into account that both losses contribute to the total loss with specific weights, so the best performing configuration for the VGAE loss is not necessarily the best performing one for the entropy loss. Although we anticipated that deep GNN models are prone to oversmoothing and oversquashing, we can see that in this case the VGAE, and therefore the random noise in the latent space, helps to mitigate these issues. In fact, even though configurations with $L = 6$ are not the best performing ones, the related results are not dramatically worse than the others. This is an important discovery, because we are interested in long paths traversing the graph and not being limited to 2 hops.

Test losses are shown in Table 4.3. We notice that the VGAE loss on the test

VGAE loss				**Entropy loss**			
L	$m = 0$	$m = 1$	$m = 2$	L	$m = 0$	$m = 1$	$m = 2$
2	1.162	1.047	**0.933**	2	**7.961**	**9.131**	10.430
3	1.114	1.025	0.945	3	8.100	9.213	10.488
4	**1.064**	**0.971**	0.945	4	8.157	9.225	10.559
5	63.645	0.972	0.952	5	8.135	9.309	**9.899**
6	1.079	0.982	0.985	6	8.336	9.350	9.951

Table 4.3: Loss comparison on the test set between mesh refinement levels 0, 1 and 2. For visual clarity, the best values for each loss and refinement level are shown in bold.

set for the 5-layer model with no refinement is extremely high, which is an anomaly compared to the other values. When performing the reparametrization trick, the `logstd` tensor is adjusted with the `clamp` function by limiting the maximum value to 10.0. However, when computing the KL loss, the magnitude of the result is dominated by the `logstd.exp()**2` operation, which leads to high values for `logstd` values greater than 1.0. In the worst case, if a `logstd` value is 10.0, the result of the KL loss is $e^{20} \approx 485{,}165{,}195.4$, which is extremely high. It turns out that, for this specific configuration, although training and validation perform well, the model is not able to generalize properly on the test set, and a noticeable amount of `logstd` values are greater than 1.0, and even reach 10.0. By the way, again the insight that the VGAE helps to mitigate oversmoothing and oversquashing is confirmed by the performance on the test set, regardless of the mesh refinement level. In general, the entropy loss worsens as the mesh refinement level increases, but this is expected, because the number of edges increases significantly with the mesh refinement level, and therefore the entropy loss is more sensitive to the attention weights.

The resulting total test loss is shown in Table 4.4. We can see that setups with

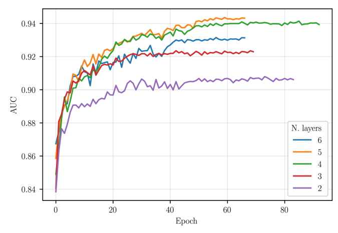
Total loss			
L	$m = 0$	$m = 1$	$m = 2$
2	5.143	5.612	6.148
3	5.164	5.632	6.188
4	**5.143**	**5.584**	6.224
5	67.713	5.626	**5.902**
6	5.247	5.657	5.960

Table 4.4: Total loss comparison on the test set between mesh refinement levels 0, 1 and 2. For visual clarity, the best values for each loss and refinement level are shown in bold.

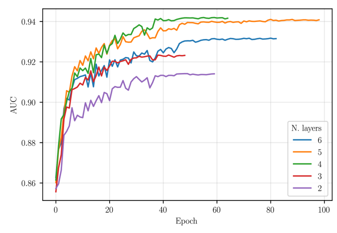
$m = 0$ and $m = 1$ agree on the best performing neural network depth, but the same depth is the worst one for $m = 2$.

After computing the loss, we also compute AUC and AP, by taking the output of the decoder on positive edges and negative edges, and comparing them with a tensor of ones and zeros respectively. The functions `roc_auc_score` and `average_precision_score` from *scikit-learn* are used to compute these metrics. In Figure 4.5, we can see the respective validation curves. In general, AUC is higher than AP, but values of AP are still acceptable: regardless of the mesh refinement level, the model makes good predictions and the latent space is able to capture the relationships between nodes and edges. From these charts, we can see that for $m = 2$ the configuration with $L = 6$ performs worse (lower metrics) than the others, while for $m = 0$ and $m = 1$ it can be comparable to other configurations. Conversely, for $m = 0$ and $m = 1$, the metrics are worse for $L = 2$, which instead is the best performing for $m = 2$. AUC and AP show similar trends.

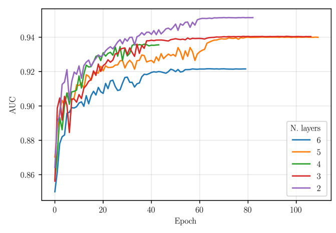
Test values for AUC and AP are shown in Table 4.5.



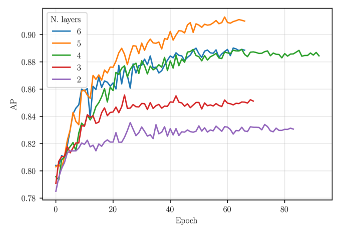
(a) AUC, $m = 0$



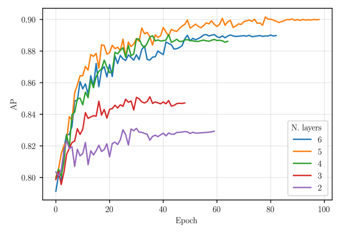
(b) AUC, $m = 1$



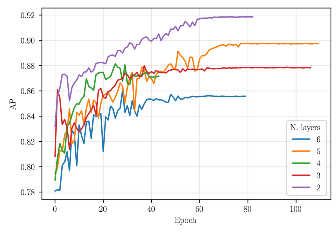
(c) AUC, $m = 2$



(d) AP, $m = 0$



(e) AP, $m = 1$



(f) AP, $m = 2$

Figure 4.5: Validation AUC and AP curves over epochs for different mesh refinement levels and number of layers.

AUC				**AP**			
L	$m = 0$	$m = 1$	$m = 2$	L	$m = 0$	$m = 1$	$m = 2$
2	0.913	0.915	**0.952**	2	0.834	0.831	**0.919**
3	0.926	0.922	0.940	3	0.856	0.847	0.881
4	0.942	**0.943**	0.938	4	0.889	0.886	0.876
5	**0.942**	0.941	0.940	5	**0.910**	**0.900**	0.897
6	0.931	0.932	0.921	6	0.891	0.889	0.855

Table 4.5: AUC and AP comparison on the test set between mesh refinement levels 0, 1 and 2. For visual clarity, the best values for each metric and refinement level are shown in bold.

We want to choose the best neural network depth according to all the test metrics. There is not a unique answer, but we can trace out a rank for each metric and each mesh refinement level, where 1 means that the model is the best with respect to that metric and mesh refinement level, and 4 means that the model is the worst. This is shown in Table 4.6. The total column shows the sum of ranks across

	Loss			**AUC**			**AP**			
$L \setminus m$	0	1	2	0	1	2	0	1	2	**Total**
2	1	2	2	4	4	1	4	4	1	23
3	3	3	3	3	3	2	3	3	2	25
4	1	1	4	1	1	3	2	2	3	**18**
5	-	-	-	-	-	-	-	-	-	-
6	4	4	1	2	2	4	1	1	4	23

Table 4.6: Ranking of neural network depths by performance metrics across different mesh refinement levels (m). Lower ranks indicate better performance.

all metrics, excluding the anomalous $L = 5$ configuration, which is not considered in the ranking, because of the anomaly on the loss function that could damage the results. The lower the total rank, the better the model performance. Therefore, we can reasonably conclude that the best model has $L = 4$ layers, and we will use it for the following experiments. This means that paths can have a maximum length of 4 hops.

All the described operations are performed almost in the same way for training, validation and testing sets. The only differences are that, for the training set, the model parameters are updated using backpropagation, while for the validation and testing sets, the model parameters are not updated. Also, the random noise employed in the reparametrization trick is not used for the validation and testing sets, because we want to evaluate the model performance on the deterministic output of the decoder. Therefore, in this case only the mean tensor is employed for the reparametrization.

4.4 Model outputs interpretation

To make predictions robust, we want to assess the model's performance on inputs with different mesh refinement levels. One strategy can be to train the model following the dataset split scheme defined in Section 4.2.5 for all the desired mesh refinement levels. However, a more straightforward approach is to use different models for different mesh refinement levels, because it is difficult to generalize the latent representation across different resolutions.

4.4.1 Correlation matrix

Given a specified mesh refinement level and the depth of the encoder (L), we are interested in computing the correlation matrix. For any layer of the network, the attention weights have shape $16 \cdot 12 \cdot E = 192 \cdot E$, so each attention weight is associated to a specific edge in the `edge_index` tensor. This one is basically a list of edges, but we need a matrix representation of the edges. Therefore, for every layer an attention matrix of shape $(16{\cdot}13{\cdot}V)\times(16{\cdot}13{\cdot}V) = (208{\cdot}V)\times(208{\cdot}V)$ is generated, using the index of the target node of the edge as row and the index of the source node of the edge as column. Then, correlation matrix is computed by defining an identity matrix of the same shape and performing a matrix multiplication between the attention matrix and the identity matrix. The result is iteratively repeated for every layer, and intermediate results are stored in a dictionary, such that it is possible to collect correlation matrices at every depth, i.e. for every path length less or equal than L. Let's assume we selected a desired path length. When analyzing correlation matrices, it is better to isolate the attention weights of edges belonging to different batches, because they are not related to each other. Therefore, we compute the correlation matrix for each batch separately, and this is possible by extracting diagonal blocks from the correlation matrix, obtaining 16 matrices of shape $(13 \cdot V) \times (13 \cdot V)$.

4.4.2 Entropy

Now let's focus on a single batch. We remember from Section 3.3.3 that the correlation matrix is row-stochastic, because the attention matrices are row-stochastic. However, this is true for all the nodes that have any incoming edge, and this does not occur for the first time step. Consequently, in order to perform any analysis on the correlation matrix, we need to filter out the rows that sum to 0, because no attention weight is assigned to them. This problem occurs for any underlying attention matrix. Therefore, depending on the chosen path length, we can have a different number of rows to filter out. For example, for a path length of 1, the first time step has no incoming edges, so the first V rows are filtered out, while for a path length of 2, the first $2 \cdot V$ rows are filtered out. In general, if the path length is p, the first $p \cdot V$ rows in the correlation matrix are filtered out. Filtering operations impose computing a mask, which has the same shape of the correlation matrix. For the remaining rows, we compute the element-wise entropy and we sum over the columns to compute a per-row entropy, obtaining an array of shape $13 \cdot V$. Then, we are interested to filter out the rows that have an entropy greater than a certain threshold. The threshold must take into account the number of positive items of the row, which depends on the path length but also on the mesh refinement level. For example, for a mesh refinement level of 0 and a path length of 1, the number of incoming edges for each target node is 5. If N is the number of positive items in the row, we set as maximum entropy $\log N$, which corresponds to a uniform distribution over the positive items. We are interested in distributions far from uniform, so the threshold must be between $\log N$ and 0. However, we described in Section 3.3.5 that it is difficult to directly determine a good entropy threshold, and therefore we set a threshold in a β-space where we can control how much the distribution of the entropy values is concentrated, regardless of the mesh refinement level and the path length. After some experiments, we found out that a good value for β is 0.8. In Table 4.7, we show the entropy threshold values for different mesh refinement levels and path lengths.

m	k	n	Entropy threshold
0	1	5	0.661
0	2	11	0.893
0	3	12	0.916
0	4	12	0.916
1	1	16	0.990
1	2	39	1.202
1	3	42	1.219
1	4	42	1.219
2	1	55	1.279
2	2	143	1.483
2	3	162	1.509
2	4	162	1.509

Table 4.7: Entropy threshold values for different mesh refinement levels and path lengths. k should be intended as the path length. n is the number of incoming paths per target node.

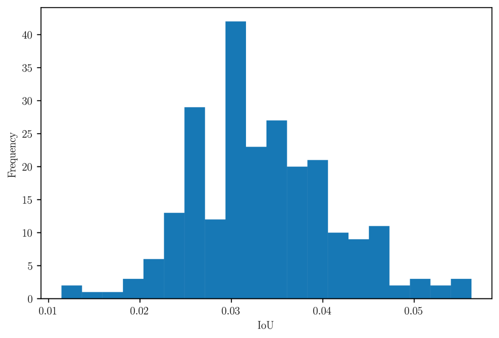
We then filter out the rows with an entropy greater or equal to the threshold, using another mask. Finally, through an `argmax` operation, we select the highest attention weight for each row. The obtained mask, namely `fine_mask` is binary and has `True` values for any edge set as candidate for the significance test.

4.4.3 Fine2coarse mapping

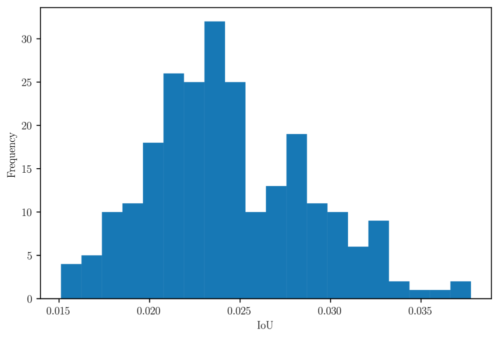
In case we are dealing with a mesh refinement level higher than 0, we must project this mask on the coarse space. To do this, we need a binary mask, namely `coarse_constraint`, that has `True` values for the items corresponding to paths that are possible in the coarse space for the specified path length, i.e. source-target pairs that are connected by a path of that length. Moreover, we need a mapping matrix from fine to coarse space. This mapping is the same regardless of the input data, and is defined using a dictionary populated during the mesh refinement process, as described in Section 4.2.3. This data structure contains, for each node at a specified mesh refinement level, the indices of the corresponding nodes in the successive refinement. Therefore, by hierarchically traversing the dictionary, we can know, for each node at refinement level 0, all the generated nodes at the desired refinement level. We then build a matrix of shape $V_{\text{coarse}} \times V_{\text{fine}}$, where, for each row, we assign uniform scores, summing to 1, to all the columns whose indices are the nodes in the fine mesh recursively generated by the coarse node represented by the row. We decide to perform all the evaluations using the mesh refinement 0 as the baseline coarse mesh, so $V_{\text{coarse}} = 12$. The mapping matrix is defined for each time step. We use the `kron` function to compute the Kronecker product between the mapping matrix and the identity matrix of shape 13×13, resulting into a matrix of shape $(13 \cdot 12) \times (13 \cdot V_{\text{fine}}) = 156 \times (13 \cdot V_{\text{fine}})$. The projection of our binary mask on the coarse space is computed through the matrix multiplication `mapping_matrix @ fine_mask @ mapping_matrix.T`. Since the result is not boolean anymore, we create a new mask, namely `coarse_mask` that has `True` values for the items that are greater than 0, because we assume that just one edge contribution in the fine space is enough to consider it in the coarse space. `coarse_mask` is then combined with `coarse_constraint` using a logical `and` operation, to obtain the final mask. Final masks are collected starting from each mesh refinement level, for each path length and test sample.

4.4.4 Significance test

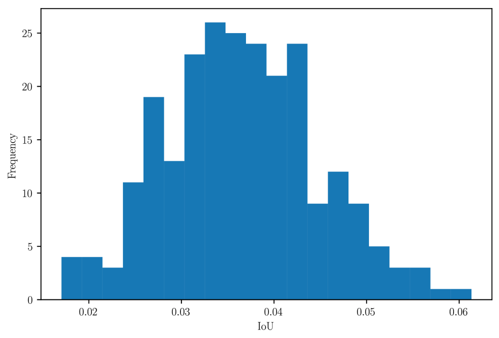
For each path length and test sample, processed binary masks of different mesh refinement levels, all having the same shape 156×156, are combined by computing their intersection (logical `and`) and union (logical `or`). IoU is calculated by dividing the sum of intersection over all the items of the mask and the sum of union over all the items. Therefore, IoU is a scalar, while intersection can be visualized on the map, by associating, for each `True` entry, the involved nodes and their spatio-temporal position. In Figure 4.6, the distribution of the IoU values over the test set is shown for different path lengths k. We can notice that IoU values are very low, and this suggests that, although the attempts to make the model generalize well on any mesh resolution, models trained on different mesh refinement levels have a totally different understanding of data relationships, so a better approach to represent edges in input graphs is needed, or maybe a different neural network architecture. However, this also highlights that some paths may be more easily transferable between different mesh resolutions than others, and they could really represent causal relationships. Although the magnitude of IoU values, if we want to visualize the significant paths across the entire test set this would still result into a lot of paths to be shown. Paths that are significant for one test sample are not necessarily



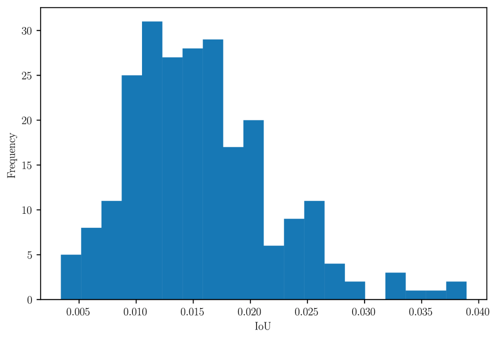
(a) $k = 1$



(b) $k = 2$



(c) $k = 3$



(d) $k = 4$

Figure 4.6: IoU histograms for different path lengths (k). The x-axis represents the IoU values, while the y-axis shows the number of occurrences for each value over the test set.

significant for the others, and this variability is fundamental to prove that the model learns a representation which depends on input data. Also, remembering that each path traverses multiple time steps, we expect that, depending on the starting time step we are considering, the paths may be different. Nevertheless, considering the overall test set, the model's choices appear less heterogeneous, and visualizations showing the significant paths at different starting time steps are more similar to each other. Figure 4.7 shows an example of this situation. Slight differences on

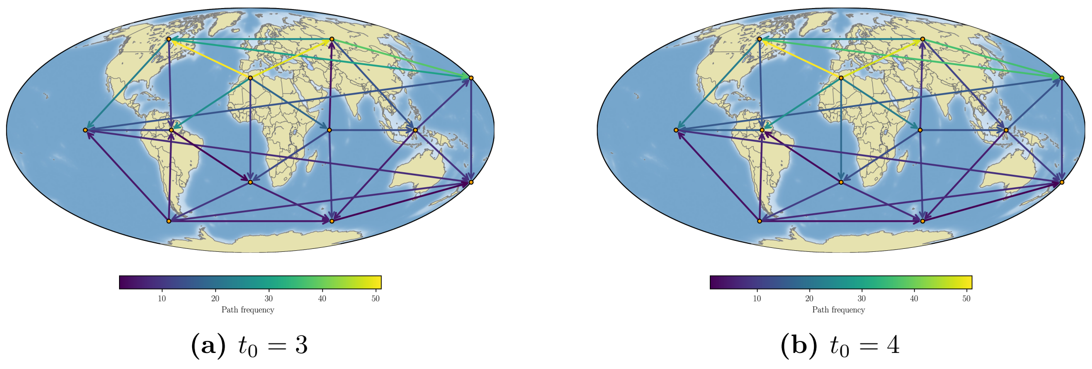
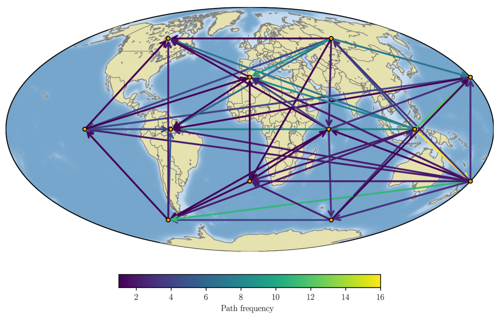


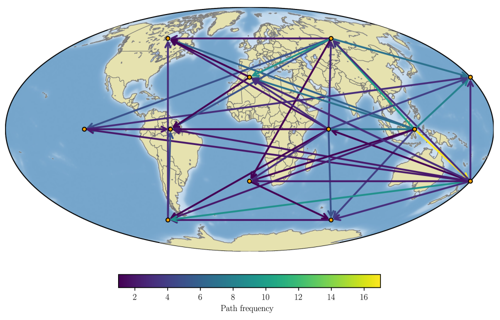
Figure 4.7: Comparison of significant paths of length $k = 1$ for different starting time steps (t_0) over the entire test set. The paths are shown on the map, where nodes are represented as circles and edges as arrows. For visual clarity, paths starting and ending in the same node are not show. Moreover, if two paths have opposite directions, the one with highest frequency is shown. For each path, the color indicates the frequency in which it appears in the test set.

paths of length $k = 1$ propagate over longer paths. Therefore, analyzing paths of length $k = 4$ can lead to more significant results, and different across different starting time steps (see Figure 4.8). Some paths are consistently frequent in the test set, regardless of the starting time step. For instance, the path connecting Indonesia to New Zealand is very frequent, but it is difficult to associate a specific meaning to detected paths: they could correspond to unknown teleconnection patterns, or they could be just noise, considering the low IoU values. To the extent of this work, although the promising methodology, there is no practical way to determine the physical meaning of detected paths, and therefore we cannot draw any conclusion about the existence of teleconnection patterns in the data. However, we can say that the model is able to learn a representation of the data that is able to capture relationships between nodes and edges, and this is a good starting point for future work.

This concludes the discussion about the results obtained by applying the previously described methodology. The next chapter will summarize the findings and



(a) $t_0 = 0$



(b) $t_0 = 1$

Figure 4.8: Comparison of significant paths of length $k = 4$ for different starting time steps (t_0) over the entire test set. See Figure 4.7 for a description of the visualization.

provide conclusions about the work done in this thesis.

Chapter 5

Conclusion

In this chapter, we outline some limitations of the current approach and suggest possible future research directions. Finally, we summarize the main contributions of this work.

5.1 Considerations on the approach

This research investigated climate teleconnection patterns using a graph-based deep learning approach applied to the ERA5 reanalysis dataset. The study utilized global climate data spanning 1940-2023 across 9 variables, which was regridded to 10-degree resolution and preprocessed to compute climatological anomalies relative to a 1970-1999 baseline. The climate data was represented as graphs using refined icosahedral meshes at three levels of complexity (12, 42, and 162 nodes), with edges connecting both spatial and temporal neighbors. The modeling framework employed a VGAE with GATv2 layers to learn latent representations and infer teleconnection patterns. The researchers systematically tested different architectural configurations, varying model depth from 2 to 6 layers and mesh refinement levels. The training objective combined VGAE loss (reconstruction plus KL divergence) with entropy loss on attention weights. Performance evaluation revealed that configurations with 4 layers and mesh refinement levels $m = 0$ or $m = 1$ achieved optimal results. The best-performing models demonstrated strong discriminative capability with AUC scores of approximately 0.95 and AP scores around 0.92 on the test set. However, entropy loss increased with mesh refinement due to growing edge complexity, and some configurations (particularly $L = 5$ with $m = 0$) showed numerical instability in KL divergence calculations. The model's attention mechanism enabled identification of potential teleconnection pathways through the graph structure, with influence paths extending up to 4 steps. Significant paths were extracted using entropy filtering and analyzed through intersection/union operations across different mesh refinements. While the low IoU scores between different mesh levels indicated limited direct transferability of detected patterns, certain spatial pathways appeared consistently across configurations, suggesting potentially robust climate connections that warrant further physical interpretation.

5.1.1 Limitations

As anticipated, the present work does not provide a definitive solution to the problem of teleconnection discovery, but it is a starting point for future research. Some limitations can be identified in the current approach, taking into account the methodology, the results and the comparison with the state of the art.

Although innovative with respect to present literature, the discussed approach does not really provide a solid foundation for proper significance testing of the discovered teleconnections. Data representation is structured in a way that edges express causal relationships in space and time, but not necessarily in a statistically significant way. In fact, we assumed that each spatio-temporal point has an influence on the neighbors, but this is not necessarily true: it is a first approximation that, thanks to the model outputs and domain knowledge, can be refined in the future, defining just a static set of points that are relevant for one node according to some tangible criteria, different for each node. The same assumption is used to interpret the attention weights of the model: since the edges are causal, the attention weights are interpreted as the strength of the causal relationship between two nodes. However, once switched to a static set of points, attention weights can effectively represent the strength of the relationship. The significance test is made of a set of conditions that must be satisfied: the entropy of the attention weights must be low, the attention weight should be the highest among all incoming edges, the same edge should be selected in all the mesh resolutions. However, we are not guaranteed that the selected edges correspond to a statistically significant relationship, but rather the model consistently prefers those edges over others. In other words, we are trusting that the model has learned a meaningful representation of the data, but we do not have a solid statistical foundation to support this assumption. At the same time, since we deal with an unsupervised problem, there are limited opportunities for verifying the model representation and its outputs: checking the model outputs against known teleconnections is a good way to verify the model, but still not sufficient as a general validation method. We could think to incorporate domain knowledge to guide the model, for instance a proper edge weight initialization based on known causal relationships, or a constraint enforcement to prevent the model from learning implausible causal relationships. However, this would require a careful selection of the domain knowledge to be used, and a proper integration into the model. When dealing with edge selection, also the *fine2mesh* approach is not guaranteed to be the best one: we could just select the most relevant edges for a specific mesh resolution, but then, when comparing results with other mesh resolutions, we would probably find that the selected edges do not exist in other resolutions, because they are coarser, so a mapping is still needed to find the corresponding edges in other resolutions. The main problem of the chosen approach is that it assumes that just one edge in the fine mesh is enough to activate it in the coarser mesh, but in general there are a lot of activations, so we could set a threshold to compute the binary mask. However, this would potentially exclude some relevant edges, so we would need to find a good threshold that balances the number of edges selected and the relevance of the relationships.

Other limitations of the proposed approach come from data representation choices. Although *grid2mesh* is a good way to represent the data, the batching strategy to create a temporal window of the data is not optimal: linking nodes of one time step to nodes of the next time step is fine because it avoids redundant information, but we are not guaranteed it is the best way to represent relationships. Moreover, selected node features were chosen as the most relevant for the problem, but they do not necessarily represent the most informative features for the model, so a more thorough feature selection process could be performed to identify the most useful features. For computational complexity reasons, we also packed together all the features of one node in a single vector, but this imposes that we cannot analyze the relationships between the different features of the same node, but only between the nodes. Also, while the edge spatial feature is informative, we can easily discover that the edge temporal feature is not informative at all, since it is always equal to 1 (the difference in time between two consecutive time steps). However, node features include spatial and temporal information, so in general edge features are not strictly necessary and the model could be trained without them. Another problem with current data representation is that the computational complexity explodes as the mesh resolution increases, because 2^m-neighbors are considered for each node. This was intentional to ensure that models trained on fine meshes can be used to discover teleconnections on coarse meshes, and in general have a view on data that just extends views in coarser meshes, and does not lose any information.

Finally, the model is not optimized for performance: hyperparameters were chosen based on a preliminary analysis, but a more thorough hyperparameter tuning process could be done to improve the model performance, especially for the relative weights of the loss terms. The limitation of this model is the length of paths that can be learned: since the neural network is based on a GNN, it cannot have a high number of layers, otherwise it would compromise the ability of the model to learn meaningful representations of data.

5.1.2 Future work

Based on the limitations of the current approach, we can identify some possible future research directions to improve the model and the results. The main goal is to provide a more solid statistical foundation for teleconnection discovery, so that discovered relationships can be considered statistically significant and not just a consequence of the model representation.

The current approach offers a good starting point, but there is still room for significant improvement. One of the main challenges is how to better integrate meteorological and climatological knowledge into the model. Right now, the graph structure is built without considering physical constraints, and this means that the model might learn relationships that are not plausible from a scientific point of view. A better solution would be to initialize the graph edges using known atmospheric dynamics and teleconnection patterns. This would provide the model with a more realistic starting point. We could also think of adding constraints during training to prevent the model from learning connections that don't make sense physically. Another possibility is to include domain knowledge through Bayesian priors or regularization terms, which would help the model focus on meaningful relationships and reduce the risk of picking up random correlations from noisy, high-dimensional data. In the long run, it might also be useful to include expert systems to flag patterns that contradict well-known scientific mechanisms and highlight others that are potentially interesting and worth deeper investigation.

From an architectural point of view, traditional GNNs struggle with long-range dependencies, which are very important in climate modeling. Alternatives like Graph Transformers could help overcome this limitation thanks to their attention mechanisms, which are better suited for capturing long-distance relationships. Recurrent Graph Networks could also be useful, especially when trying to model how climate patterns evolve over time, since they are able to retain memory of past states. Similarly, Hierarchical Graph Networks might help by learning from both local and global structures at the same time, which is important because teleconnections appear at different spatial and temporal scales. Recent attention-based architectures with skip connections could also be explored, since they are able to focus on the most relevant patterns while reducing the risk of information loss through deeper layers.

Another important limitation of the current model is how it deals with mesh resolution. Right now, the mesh is static and has the same resolution everywhere, but climate data is not uniform: some regions have more complex dynamics than others. Using adaptive mesh refinement could allow the model to focus on regions where teleconnections are more complex, while using lower resolution elsewhere. This would save computational resources and improve the model's ability to learn from the data. It also suggests a better way to combine information from multiple resolutions: instead of training separate models, we could develop multi-scale models that ensure coherence across different resolutions. This would require changes in the loss function, making it aware of resolution and capable of balancing information from each level. The way we select edges should also change: instead of a static approach, we could develop a dynamic system that adjusts to the resolution and preserves important connections across different scales.

Feature engineering is another area that needs improvement. So far, different variables like temperature, pressure, and wind have been simply concatenated, but this does not capture how they interact. A more advanced integration could reveal patterns that only appear when considering multiple variables together. The temporal dimension also needs better treatment: currently, the model uses a fixed-size time window, but climate phenomena operate on different timescales, from days to decades. A more flexible approach would allow the window size to adapt based on the type of pattern being learned. Teleconnections also often involve time lags: a change in one region might affect another region weeks or months later, and the model should learn these delays automatically.

When it comes to validation, more robust methods are needed. Checking model outputs against known teleconnections is useful but not sufficient. Using synthetic data from climate simulations with known patterns could provide a more controlled environment for testing the model's ability to recover meaningful relationships.

Finally, computational complexity is still a major bottleneck. As the mesh resolution increases, the number of nodes and edges grows rapidly, making the model expensive to train and run. We could address this by using approximate nearest neighbors to limit the number of considered connections, or by applying hierarchical clustering to group similar nodes and reduce graph size. Sparse representations that keep only the most relevant edges would also help reduce memory and computation requirements, making the results easier to interpret. At the same time, distributed computing frameworks could allow us to scale up the model to larger datasets, enabling near real-time analysis of global climate patterns.

5.2 Main contributions

In the following, we summarize the main contributions of this work:

- unsupervised approach to climate teleconnections discovery;
- graph-structured data representation using *grid2mesh*, time windows and topological neighborhood to represent spatio-temporal relationships in climate data;
- VGAE model architecture with a GATv2 encoder to learn the graph structure and highlight the most relevant edges through the attention mechanism;

- custom loss function to guide the model towards an optimal distribution of the attention weights;
- statistical heuristics to select the most relevant edges from the attention weights, based on their entropy and consistency across different mesh resolutions.

This concludes the present work on the discovery of teleconnections using graph neural networks. The proposed approach provides a new perspective on how these complex relationships can be modeled and understood, and opens up new avenues for future research in this area. The results obtained demonstrate just the potential of graph neural networks for discovering teleconnections in climate data, without any expectation on the quality of the actual inference, and highlight the importance of considering the spatial and temporal dimensions of the data in order to capture the complex relationships that exist between different regions of the Earth. Code and data are available at: `https://github.com/markdeluk/GNNTeleconnections`.

Appendix A

Statistical methods for climate data analysis

A.1 Regression analysis

Linear regression is used to model the relationship between a single independent variable and a dependent variable using a linear equation. The goal is to find the best-fitting line that minimizes the difference between the predicted and actual values. The linear regression model can be represented as:

$$y = \beta_0 + \beta_1 x + \epsilon$$

where y is the dependent variable, x is the independent variable, β_0 is the intercept, β_1 is the slope of the line, and ϵ is the error term. The slope represents the change in the dependent variable for a one-unit change in the independent variable. The intercept represents the value of the dependent variable when the independent variable is zero. The error term accounts for the variability in the dependent variable that cannot be explained by the independent variable.

An extension of linear regression is multiple linear regression, which generalizes standard linear regression at multiple independent variables:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n + \epsilon$$

where y is the dependent variable, $x_1, x_2, \ldots, x_n$ are the independent variables, β_0 is the intercept, $\beta_1, \beta_2, \ldots, \beta_n$ are the coefficients for the independent variables, and ϵ is the error term. The coefficients represent the change in the dependent variable for a one-unit change in each independent variable while holding all other variables constant. Multiple linear regression allows for the examination of the combined effects of multiple factors on a dependent variable.

Polynomial regression is another extension of linear regression that allows for non-linear relationships between the independent and dependent variables. It models the relationship using a polynomial equation, which can capture more complex patterns in the data. The polynomial regression model can be represented as:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \ldots + \beta_n x^n + \epsilon$$

where y is the dependent variable, x is the independent variable, β_0 is the intercept, $\beta_1, \beta_2, \ldots, \beta_n$ are the coefficients for the polynomial terms, and ϵ is the error term. The degree of the polynomial determines the complexity of the model: a higher degree allows for more flexibility in capturing non-linear relationships.

Logistic regression is used for binary classification problems, where the dependent variable is categorical. It models the probability of the dependent variable belonging to a specific class based on one or more independent variables. The logistic regression model can be represented as:

$$p(y = 1|x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

where $p(y = 1|x)$ is the probability of the dependent variable being equal to 1 given the independent variable x, β_0 is the intercept, and β_1 is the coefficient for the independent variable. The logistic function maps the linear combination of the independent variables to a probability value between 0 and 1.

Logistic regression can be extended to multiple independent variables using the following equation:

$$p(y = 1|x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n)}}$$

where $p(y = 1|x)$ is the probability of the dependent variable being equal to 1 given the independent variables $x_1, x_2, \ldots, x_n$, β_0 is the intercept, and $\beta_1, \beta_2, \ldots, \beta_n$ are the coefficients for the independent variables. The coefficients represent the change in the log-odds of the dependent variable for a one-unit change in each independent variable while holding all other variables constant.

A.2 Time series analysis

ARIMA models [15] are used to model time series data by combining three main components: autoregression (AR), differencing (Integrated, I), and moving average (MA). AR indicates that the current value of the series is based on its previous values. For instance, today's temperature may be influenced by temperatures from the past few days. I represents the differencing of the series, which is used to make the time series stationary by removing trends and seasonality. MA captures the relationship between the current value and past errors. The ARIMA model is characterized by three parameters: p, d, and q, where p is the order of the autoregressive part (number of lag observations included in the model), d is the degree of differencing (number of times that the raw observations are differenced), and q is the order of the moving average part (size of the moving average window). The ARIMA model can be represented as:

$$y_t = \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \ldots + \theta_q \epsilon_{t-q} + \epsilon_t$$

where y_t is the current value of the time series, $\phi_1, \phi_2, \ldots, \phi_p$ are the coefficients for the autoregressive part, $\theta_1, \theta_2, \ldots, \theta_q$ are the coefficients for the moving average part, and ϵ_t is the error term. The ARIMA model can be used to forecast future values by iteratively applying the model to generate predictions. It can also be extended to include seasonal components, resulting in Seasonal ARIMA (SARIMA) models, which are particularly useful for modeling seasonal time series data. The SARIMA model [92] incorporates seasonal autoregressive and moving average terms, allowing it to capture both short-term and long-term dependencies in the data. The SARIMA model can be represented as:

$$\begin{aligned} y_t = {} & \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} \\ & + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \ldots + \theta_q \epsilon_{t-q} \\ & + Y_{t-s} + \Phi_1 Y_{t-s-1} + \Phi_2 Y_{t-s-2} + \ldots + \Phi_P Y_{t-s-P} \\ & + \Theta_1 \epsilon_{t-s-1} + \Theta_2 \epsilon_{t-s-2} + \ldots + \Theta_Q \epsilon_{t-s-Q} \\ & + \epsilon_t \end{aligned} \tag{A.1}$$

where Y_{t-s} is the seasonal component, $\Phi_1, \Phi_2, \ldots, \Phi_P$ are the coefficients for the seasonal autoregressive part, and $\Theta_1, \Theta_2, \ldots, \Theta_Q$ are the coefficients for the seasonal moving average part. The seasonal period s represents the number of observations in each season (e.g., 12 for monthly data with yearly seasonality). The SARIMA model can be used to forecast future values by iteratively applying the model to generate predictions.

STL [16] decomposes a time series into three components: trend, seasonality, and residuals. The trend component represents the long-term movement in the data, while the seasonal component captures the repeating patterns that occur at regular intervals. The residuals represent the random noise in the data. STL can be used to identify and remove trends and seasonality from the data, making it easier to analyze and model, and thus improving forecasting accuracy. To estimate the trend and seasonal components, STL uses the LOESS smoother, a non-parametric regression method that fits local polynomial regressions to the data, allowing for flexible modeling of non-linear relationships. The trend component is estimated by applying the LOESS smoother to the entire time series, which captures the underlying trend without being overly influenced by short-term fluctuations. The seasonal component is estimated by applying the LOESS smoother to the seasonal subseries, which are obtained by grouping the data by season (e.g., month or quarter). The residuals are calculated by subtracting the trend and seasonal components from the original time series. The STL decomposition can be visualized as a plot showing the original time series, trend component, seasonal component, and residuals, allowing for a clear understanding of the underlying patterns in the data. The STL decomposition can be represented mathematically as:

$$y_t = T_t + S_t + R_t$$

where y_t is the observed value at time t, T_t is the trend component, S_t is the seasonal component, and R_t is the residual component.

Exponential smoothing [17] is a family of forecasting methods that apply weighted averages of past observations to make predictions. The weights decrease exponentially as the observations get older, giving more importance to recent data. Exponential smoothing can be used for both univariate and multivariate time series data, making it a versatile tool for climate prediction. The simplest form of exponential smoothing is single exponential smoothing, which is suitable for time series data without trends or seasonality. It can be represented as:

$$\hat{y}_{t+1} = \alpha y_t + (1 - \alpha)\hat{y}_t$$

where $\hat{y}_{t+1}$ is the forecasted value for the next time step, y_t is the observed value at time t, $\hat{y}_t$ is the forecasted value at time t, and α is the smoothing parameter ($0 < \alpha < 1$). The smoothing parameter determines the weight given to the most recent observation compared to the previous forecast. A higher value of α gives more weight to recent observations, while a lower value gives more weight to past forecasts. The choice of α can significantly impact the accuracy of the forecasts, and it is often determined through optimization techniques or cross-validation.

A.3 Hypothesis testing

A.3.1 Frequentist approach

A Student's t-test [22] is a parametric hypothesis test used to compare the means of two groups (or datasets) to determine if there is a significant difference between them. It can be applied to independent samples (two-sample t-test) or paired samples (paired t-test). The t-test can be represented as:

$$t = \frac{\bar{x}_1 - \bar{x}_2}{s_p\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}}$$

where t is the t-statistic, $\bar{x}_1$ and $\bar{x}_2$ are the sample means, s_p is the pooled standard deviation, and n_1 and n_2 are the sample sizes. The pooled standard deviation is calculated as:

$$s_p = \sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}}$$

where s_1 and s_2 are the sample standard deviations. The t-test assumes that the data is normally distributed and that the variances of the two groups are equal. However, its application in climate science is limited due to the autocorrelated nature of climate data, which can lead to misleading conclusions if not properly addressed.

F-test [23] is a statistical parametric test used to compare the variances of two or more groups to determine if there are significant differences between them, so it is useful to compare variability in climate variables. It is often used as a preliminary test before conducting an ANOVA to assess the homogeneity of variances assumption. The F-test calculates an F-statistic, which is the ratio of the variances of the groups being compared. The F-statistic can be represented as:

$$F = \frac{s_1^2}{s_2^2}$$

where F is the F-statistic, s_1^2 is the variance of the first group, and s_2^2 is the variance of the second group. The F-test assumes independent observations and random sampling from normally distributed populations.

An analysis of variance (ANOVA) [23] extends F-test to compare the means of three or more groups to determine if there are significant differences between them. It helps understanding differences in climate variables across different regions or time periods. ANOVA tests the null hypothesis that all group means are equal against the alternative hypothesis that at least one group mean is different. The ANOVA F-statistic is calculated as the ratio of the variance between groups to the variance within groups:

$$F = \frac{\text{MS}_{\text{between}}}{\text{MS}_{\text{within}}}$$

where F is the F-statistic, $\text{MS}_{\text{between}}$ is the mean square between groups, and $\text{MS}_{\text{within}}$ is the mean square within groups. The F-statistic is then compared to a critical value from the F-distribution to determine the p-value. The ANOVA test assumes that the variances of the groups are equal (homoscedasticity), and that the observations are independent. ANOVA can decompose the total variance into components attributable to different sources, such as treatment effects and random error. One-way ANOVA is used when there is one independent variable with three or more levels, and is expressed as:

$$y_{ij} = \mu + \alpha_i + \epsilon_{ij}$$

where y_{ij} is the dependent variable, μ is the overall mean, α_i is the effect of the i-th group, and ϵ_{ij} is the error term. The one-way ANOVA tests whether there are significant differences between the means of the groups. In contrast, two-way ANOVA is used when there are two independent variables, and it can assess both main effects and interaction effects between the independent variables. The two-way ANOVA can be expressed as:

$$y_{ijk} = \mu + \alpha_i + \beta_j + \gamma_{ij} + \epsilon_{ijk}$$

where y_{ijk} is the dependent variable, μ is the overall mean, α_i is the effect of the i-th group of the first independent variable, β_j is the effect of the j-th group of the second independent variable, γ_{ij} is the interaction effect between the two independent variables, and ϵ_{ijk} is the error term.

Pearson's correlation coefficient [24] is a parametric test used to assess the strength and direction of the linear relationship between two continuous variables, such as temperature and CO_2 levels. The Pearson correlation coefficient (r) ranges from -1 to 1, where -1 indicates a perfect negative correlation, 0 indicates no correlation, and 1 indicates a perfect positive correlation. The Pearson correlation coefficient can be calculated as:

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2}\sqrt{\sum(y_i - \bar{y})^2}}$$

where r is the Pearson correlation coefficient, x_i and y_i are the individual data points, $\bar{x}$ and $\bar{y}$ are the means of the x and y variables, respectively. The Pearson correlation test assumes that the relationship between the variables is linear, homoschedasticity and there are no outliers.

Mann-Kendall (MK) test [25] is a non-parametric test used to assess the significance of trends in time series data. It is particularly useful for detecting monotonic trends in non-normally distributed data. The MK test calculates a test statistic based on the number of pairs of observations that are in increasing or decreasing order. The test statistic can be represented as:

$$S = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \operatorname{sgn}(x_j - x_i)$$

where S is the test statistic, n is the number of observations, and $\operatorname{sgn}(x_j - x_i)$ is the sign function that returns 1 if $x_j > x_i$, -1 if $x_j < x_i$, and 0 if $x_j = x_i$. The MK test can also be extended to account for seasonality and autocorrelation in the data. In particular, the SK (Seasonal Kendall) test [93] is a modification of the MK test that accounts for seasonal patterns in the data. The SK test calculates the test statistic by comparing the ranks of observations within each season, allowing for a more accurate assessment of trends in seasonal data. The SK test can be represented as:

$$S = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \operatorname{sgn}(x_j - x_i) \cdot \operatorname{sgn}(t_j - t_i)$$

where S is the test statistic, n is the number of observations, $\operatorname{sgn}(x_j - x_i)$ is the sign functmaion that returns 1 if $x_j > x_i$, -1 if $x_j < x_i$, and 0 if $x_j = x_i$, and $\operatorname{sgn}(t_j - t_i)$ is the sign function that returns 1 if $t_j > t_i$, -1 if $t_j < t_i$, and 0 if $t_j = t_i$. Mann-Kendall test assumes that the data is independent and identically distributed (i.i.d.), meaning that the observations are drawn from the same distribution and are not correlated with each other.

Sen's slope [26] is a non-parametric method used to estimate the slope (regression coefficient) of a linear trend in time series data, i.e. a measure of the rate of change. It can be used in conjunction with the MK test to assess the significance of trends. The Sen's slope estimator can be calculated as:

$$\hat{Q} = \frac{\operatorname{median}\left(\frac{x_j - x_i}{j - i}\right)}{i < j}$$

where $\hat{Q}$ is the estimated slope, x_i and x_j are the observations at times i and j, respectively. Sen's slope assumes independence of observations and monotonic trend over time.

The Pettitt test [27] is a non-parametric test used to detect abrupt changes in the mean level of a time series, so it can identify shifts in climate variables that may be associated with teleconnections or other climate phenomena. The Pettitt test calculates a test statistic based on the number of observations that are greater than or less than the median of the time series. The test statistic can be represented as:

$$U_{t,T} = \sum_{i=1}^{t} \sum_{j=t+1}^{T} \operatorname{sgn}(x_i - x_j)$$

where t ranges from 1 to $T - 1$, and T is the total number of observations. The sign function $\operatorname{sgn}(x)$ returns 1 if $x > 0$, 0 if $x = 0$, and -1 if $x < 0$. The significante test is assessed by calculating:

$$p \approx 2 \exp\left(-\frac{6K_T^2}{T^3 + T^2}\right)$$

where K_T is the maximum absolute value of $U_{t,T}$ over all possible t, and T is the number of observations. The Pettitt test can be applied to both univariate and multivariate time series data, making it a versatile tool for climate analysis. However, it is mainly meant for univariate data, and extending it to multivariate data requires modifications or adaptations to handle the complexity of data itself. The Pettitt test assumes independence of observations, continuous data and no long-term trends. It also assumes a single change-point in time series: in other words, there is an abrupt change in the mean level of the time series at a specific point in time, and the data before and after this point are independent. This assumption may not hold true in all cases, especially in climate data, where multiple change-points or gradual changes may occur.

The Spearman's rank correlation coefficient [28] is a non-parametric test used to assess the strength and direction of the monotonic relationship between two continuous or ordinal variables. It is based on the ranks of the data rather than the actual values, making it less sensitive to outliers and non-normal distributions. The Spearman's rank correlation coefficient (r_s) can be calculated as:

$$r_s = 1 - \frac{6\sum d_i^2}{n(n^2-1)}$$

where r_s is the Spearman's rank correlation coefficient, d_i is the difference between the ranks of the two variables for each observation, and n is the number of observations. Like Pearson, the Spearman's rank correlation coefficient ranges from -1 to 1, where -1 indicates a perfect negative monotonic correlation, 0 indicates no correlation, and 1 indicates a perfect positive monotonic correlation. The Spearman's rank correlation test assumes that the data is ordinal or continuous and that the relationship between the variables is monotonic.

A.3.2 Bayesian approach

The Bayes factor [94] is a Bayesian approach to hypothesis testing that quantifies the strength of evidence in favor of one hypothesis over another. It is calculated as the ratio of the posterior probabilities of the two hypotheses, given the observed data. The Bayes factor can be represented as:

$$BF_{10} = \frac{P(D|H_1)}{P(D|H_0)}$$

where BF_{10} is the Bayes factor comparing hypothesis H_1 to hypothesis H_0, $P(D|H_1)$ is the likelihood of the data given hypothesis H_1, and $P(D|H_0)$ is the likelihood of the data given hypothesis H_0. A Bayes factor greater than 1 indicates evidence in favor of hypothesis H_1, while a Bayes factor less than 1 indicates evidence in favor of hypothesis H_0. The strength of evidence can be interpreted using a scale which categorizes Bayes factors into different ranges:

- $1:1$ to $3:1$: Weak evidence
- $3:1$ to $10:1$: Substantial evidence

- $10:1$ to $30:1$: Strong evidence
- $30:1$ to $100:1$: Very strong evidence
- $> 100:1$: Decisive evidence

The following holds:

$$BF_{01} = \frac{1}{BF_{10}} = \frac{P(D \mid H_0)}{P(D \mid H_1)}$$

i.e., the Bayes factor comparing hypothesis H_0 to hypothesis H_1 is the inverse of the Bayes factor comparing hypothesis H_1 to hypothesis H_0. The Bayes factor can be used to compare multiple hypotheses simultaneously, allowing for a more comprehensive assessment of the evidence in favor of different models or explanations. It can also be used to update prior beliefs about the hypotheses as new data becomes available, making it a powerful tool for hypothesis testing in climate science.

Bayesian ANOVA [95] is a Bayesian approach to analyze variance in data, allowing for the comparison of means across multiple groups while accounting for uncertainty in the parameter estimates. It provides a posterior distribution for the group means and allows for the calculation of Bayes factors to assess the strength of evidence in favor of different hypotheses. Bayesian ANOVA can be used to analyze climate data with multiple factors, such as temperature, precipitation, and other climate variables, while accounting for uncertainty in the estimates. The mathematical representation of Bayesian ANOVA is similar to frequentist ANOVA.

Bayesian regression [96] is a Bayesian approach to regression analysis. It can be used to model the relationship between climate variables while accounting for uncertainty in the parameter estimates, providing a more robust understanding of climate variability. The Bayesian regression model can be represented as:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} + \epsilon_i$$

where y_i is the dependent variable for observation i, β_0 is the intercept, $\beta_1, \beta_2, \ldots, \beta_p$ are the coefficients for the independent variables $x_{i1}, x_{i2}, \ldots, x_{ip}$, and ϵ_i is the error term. The model can be extended to include random effects, hierarchical structures, or other covariates, allowing for more complex analyses. The Bayesian regression model can be estimated using Markov Chain Monte Carlo (MCMC) methods, which are a class of algorithms used to sample from complex probability distributions, allowing for the estimation of posterior distributions in Bayesian analysis. They are particularly useful for high-dimensional parameter spaces and can be applied to various Bayesian models, including regression, hierarchical models, and latent variable models. MCMC methods generate samples from the posterior distribution by constructing a Markov chain[1] that converges to the target distribution. The most common MCMC algorithm is the Metropolis-Hastings algorithm, which uses a proposal distribution to generate candidate samples and accepts or rejects them based on their likelihood.

[1]A Markov chain is a sequence of random variables where the future state depends only on the current state and not on the past states. The chain is constructed by iteratively proposing new samples based on the current sample and accepting or rejecting them based on their likelihood. The acceptance probability is determined by the ratio of the target distribution at the proposed sample to the target distribution at the current sample. This process continues until a sufficient number of samples have been generated, allowing for the estimation of posterior distributions and credible intervals [97].

Appendix B

Data-driven decomposition methods for climate data

B.1 Empirical orthogonal functions

The EOF method can be represented mathematically as:

$$X(t) = \sum_{i=1}^{N} a_i(t)\phi_i$$

where $X(t)$ is the observed data at time t, $a_i(t)$ are the principal component time series, ϕ_i are the EOFs (spatial patterns), and N is the number of EOFs. The EOFs are obtained by solving the eigenvalue problem of the covariance matrix of the data, allowing for the identification of the dominant modes of variability. The first EOF represents the mode with the highest variance, while subsequent EOFs represent modes with decreasing variance. EOF analysis can be used to identify teleconnections by examining the spatial patterns associated with different climate indices or phenomena [98]. EOF decomposition is strictly related to PCA (Principal Component Analysis), which is a dimensionality reduction technique that transforms a dataset into a new coordinate system, where the axes (principal components) are ordered by the amount of variance they explain. EOF analysis can be seen as a specific application of PCA to climate data, where the goal is to identify the dominant modes of variability in the data, and the main difference between EOF and PCA is that EOF focuses on spatial patterns, while PCA focuses on the overall structure of the data. However, most online sources use the two terms interchangeably [32].

B.2 Singular spectrum analysis

The mathematical foundation of SSA begins with the construction of a trajectory matrix from the original time series. Given a time series $X = \{x_1, x_2, \ldots, x_N\}$ of length N, and a window length L (where $1 < L < N$), the trajectory matrix $\mathbf{H}$ is constructed as a Hankel matrix[1] of size $L \times K$, where $K = N - L + 1$. Each row i of this matrix contains L consecutive elements of the time series starting from position i:

$$\mathbf{H}[i, j] = x_{i+j-1}$$

This embedding process transforms the univariate time series into a multidimensional trajectory space, capturing the temporal dependencies within the data. The decomposition phase applies singular value decomposition (SVD) to the trajectory matrix:

$$\mathbf{H} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

where $\mathbf{U}$ is an $L \times L$ matrix of left singular vectors, $\mathbf{\Sigma}$ is an $L \times K$ diagonal matrix of singular values $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r$, and $\mathbf{V}^T$ is a $K \times K$ matrix of right singular vectors. Each singular value represents the contribution of the corresponding mode to the total variance of the system. The trajectory matrix can then be expressed as:

$$\mathbf{H} = \sum_{i=1}^{r} \sigma_i \mathbf{U}_i \mathbf{V}_i^T$$

where each term $\sigma_i \mathbf{U}_i \mathbf{V}_i^T$ represents an elementary matrix capturing a specific temporal pattern. The reconstruction process involves grouping related elementary matrices and converting them back to time series format through diagonal averaging, also known as Hankelization. For a given elementary matrix $\mathbf{H}_i$, the reconstructed time series component is obtained by averaging the anti-diagonals of the matrix. This process ensures that the reconstructed components maintain the temporal structure of the original data while isolating specific dynamical features [34].

[1]A Hankel matrix is a square matrix in which each ascending skew-diagonal from left to right is constant [99]. For example:

$$\mathbf{H} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 \\ x_2 & x_3 & x_4 & x_5 & x_6 \\ x_3 & x_4 & x_5 & x_6 & x_7 \\ x_4 & x_5 & x_6 & x_7 & x_8 \\ x_5 & x_6 & x_7 & x_8 & x_9 \end{bmatrix}$$

B.3 Dynamic mode decomposition

The mathematical foundation of DMD begins with a sequence of spatial data snapshots $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m \in \mathbb{R}^n$, where each snapshot represents the state of a dynamical system at discrete time intervals. The fundamental assumption is that the system dynamics can be approximated by a linear operator $\mathbf{A}$:

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k$$

From this sequence, two data matrices are constructed:

$$\mathbf{X}_1 = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_{m-1} \end{bmatrix} \in \mathbb{R}^{n\times(m-1)}$$
$$\mathbf{X}_2 = \begin{bmatrix} \mathbf{x}_2 & \mathbf{x}_3 & \cdots & \mathbf{x}_{m} \end{bmatrix} \in \mathbb{R}^{n\times(m-1)}$$

with the relationship $\mathbf{X}_2 = \mathbf{A}\mathbf{X}_1$. Since the data matrix $\mathbf{X}_1$ may be rank-deficient, SVD is applied: $\mathbf{X}_1 = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$, where $\mathbf{U}$ contains the left singular vectors, $\mathbf{\Sigma}$ is a diagonal matrix of singular values, and $\mathbf{V}$ contains the right singular vectors. A reduced-order approximation of the linear operator is then constructed:

$$\tilde{\mathbf{A}} = \mathbf{U}^*\mathbf{X}_2\mathbf{V}\mathbf{\Sigma}^{-1}$$

The eigenvalues and eigenvectors of $\tilde{\mathbf{A}}$ provide the essential dynamic information: $\tilde{\mathbf{A}}\mathbf{w}_i = \lambda_i\mathbf{w}_i$, where λ_i are the eigenvalues and $\mathbf{w}_i$ are the corresponding eigenvectors. The DMD modes are computed as $\boldsymbol{\phi}_i = \mathbf{U}\mathbf{w}_i$, and each mode represents a spatial pattern that evolves according to $\boldsymbol{\phi}_i(t) = \boldsymbol{\phi}_i e^{\omega_i t}$, where $\omega_i = \ln(\lambda_i)/\Delta t$ is the continuous-time eigenvalue [100]. Each eigenvalue $\lambda_i = |\lambda_i|e^{i\theta_i}$ encodes both amplitude and phase information, providing the growth/decay rate $\sigma_i = \ln(|\lambda_i|)/\Delta t$ and oscillation frequency $f_i = \theta_i/(2\pi\Delta t)$. This dual characterization allows DMD to identify not only the spatial structure of climate patterns but also their temporal behavior, including stability properties. Modes with $|\lambda_i| > 1$ are unstable and grow exponentially, while modes with $|\lambda_i| < 1$ are stable and decay, and modes with $|\lambda_i| = 1$ are neutrally stable and maintain constant amplitude [37].

Appendix C

Artificial intelligence and neural networks

C.1 Artificial intelligence

Artificial intelligence (AI) refers to the capability of a digital computer or computer-controlled robot to perform tasks typically associated with human intelligence, such as reasoning, learning, problem-solving, perception, and natural language understanding. It encompasses the development of systems endowed with intellectual processes characteristic of humans, including the ability to reason, discover meaning, generalize, or learn from past experiences. AI encompasses several key branches, each focusing on specific aspects of replicating or enhancing human-like intelligence in machines. For instance, machine learning, natural language processing, computer vision, and robotics are all integral components of AI [101].

Machine learning (ML), in particular, involves the development of algorithms that enable computers to learn from data and improve their performance over time without being explicitly programmed. This branch of AI has gained significant attention due to its ability to analyze large datasets, identify patterns, and make predictions based on historical information.

ML algorithms can be categorized into supervised learning, unsupervised learning, semi-supervised learning and reinforcement learning, each with its own set of techniques and applications: supervised learning involves training a model on labeled data, where the input-output pairs are known; unsupervised learning deals with unlabeled data, seeking to identify hidden patterns or structures; semi-supervised learning combines both labeled and unlabeled data to improve learning efficiency; reinforcement learning focuses on training agents to make decisions by interacting with an environment and receiving feedback in the form of rewards or penalties [102].

The world of ML can be divided into two main approaches: shallow learning and deep learning. Shallow learning refers to traditional machine learning techniques that typically involve a single layer of processing, such as linear regression, decision trees, and support vector machines. These methods are effective for simpler tasks and smaller datasets but may struggle with more difficult problems. They are easier to interpret and require less computational power. However, they cannot capture complex relationships and they highly rely on manual feature engineering: starting from raw data, the user must identify and extract relevant features that can be used to train the model. This process can be time-consuming and requires domain expertise, as the success of shallow learning models often depends on the quality of the features selected [103].

On the other hand, deep learning is a subset of ML that employs multiple layers of processing to model intricate patterns and relationships in data. Examples are convolutional neural networks (CNNs) and recurrent neural networks (RNNs). These models can automatically learn hierarchical representations of data, without the need for extensive feature engineering. This is particularly useful in tasks such as image recognition, natural language processing, and speech recognition, where the data is often high-dimensional and unstructured. However, deep learning models are more computationally intensive and require larger datasets to achieve optimal performance. They can also be more challenging to interpret, as the decision-making process of deep neural networks is often opaque, making it difficult to understand how they arrive at specific predictions.

C.2 Neural networks

The basis for both shallow and deep learning is the concept of artificial neural network (ANN): it is a computational model inspired by the structure and function of the biological neural network in the human brain. ANNs consist of interconnected nodes (neurons) organized into layers, where each neuron processes input data and passes it to the next layer. The connections between neurons have associated weights that are adjusted during the training process to minimize the difference between predicted and actual outputs. This iterative process allows ANNs to learn complex patterns and relationships in data, making them powerful tools for various applications in AI. The structure of ANN is typically composed of an input layer, one or more hidden layers, and an output layer. The input layer receives the raw data, while the hidden layers perform computations and transformations on the data. The output layer produces the final predictions or classifications based on the learned representations from the hidden layers. Each neuron in a layer is connected to neurons in the subsequent layer, allowing information to flow through the network [104].

C.2.1 Perceptron

The simplest and oldest form of an ANN is the perceptron, which consists of a single layer of neurons. The perceptron takes multiple input signals, each multiplied by a weight, sums them up, and applies an activation function to produce an output. The activation function introduces non-linearity into the model. The perceptron can be represented mathematically as follows:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

where y is the output, f is the activation function, w_i are the weights, x_i are the input signals, and b is the bias term. The bias term allows the model to shift the activation function, enabling it to fit the data better. The perceptron can be visualized as a simple diagram, as shown in Figure C.1.

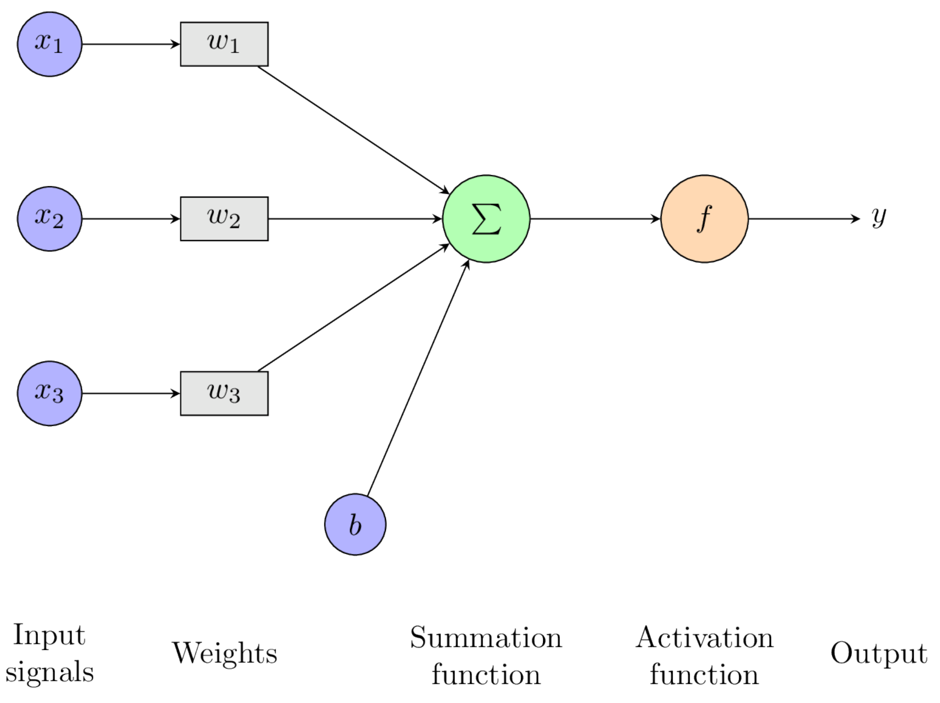


Figure C.1: Structure of a perceptron, a basic unit of an ANN. The input signals are multiplied by weights, summed, and passed through an activation function to produce the output.

The perceptron is characterized by its ability to learn linear decision boundaries, making it suitable for linearly separable problems. It uses a step function as the activation function, which produces binary outputs (0 or 1) based on whether the weighted sum of inputs exceeds a certain threshold:

$$f(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

where z is the weighted sum of inputs.

The perceptron learns by adjusting the weights based on the error between predicted and actual outputs using a simple learning rule:

$$w_i \leftarrow w_i + \eta(y - \hat{y})x_i$$

where w_i is the weight for input x_i, η is the learning rate, y is the actual output, and $\hat{y}$ is the predicted output. The learning rate determines how much the weights are adjusted during each update. A higher learning rate results in larger weight updates, while a lower learning rate leads to smaller updates. The perceptron continues to adjust its weights until it converges to a solution that minimizes the error [105].

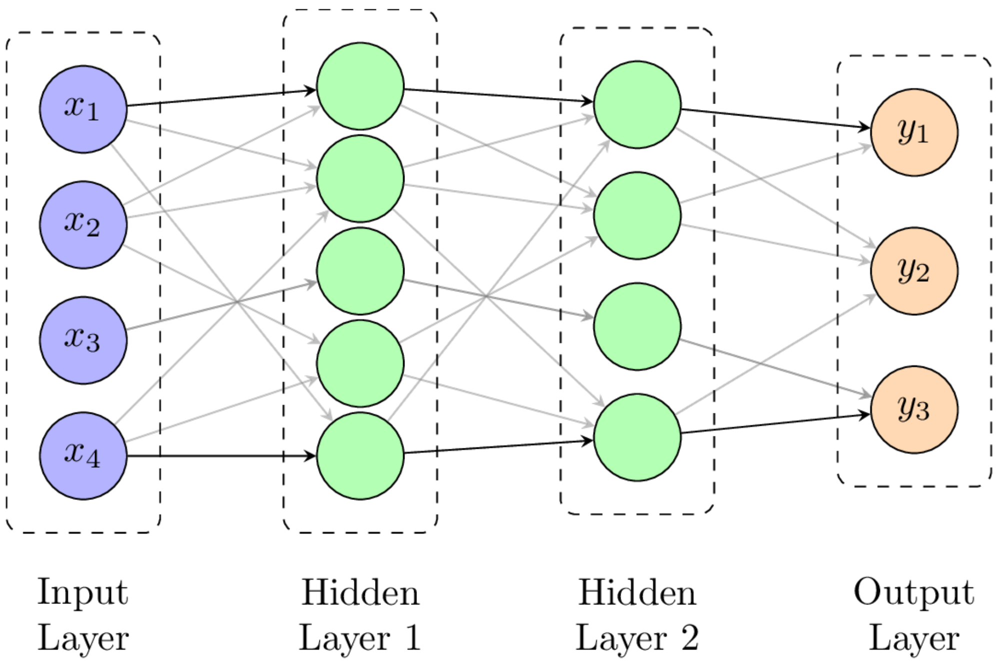
C.3 Multi-layer perceptron

The perceptron is a binary classifier, meaning it can only classify data into two categories. For complex tasks, it is not enough, and therefore multi-layer perceptrons (MLPs), some years afterwards, were introduced. MLPs consist of multiple layers of neurons, where each layer can have different numbers of neurons, and the connections between layers can be fully connected or partially connected. The structure of an MLP is shown in Figure C.2.

MLPs can learn complex relationships, and this is proven by the universal approximation theorem [106], which states that a feedforward neural network with a single hidden layer containing a finite number of neurons can approximate any continuous function on compact subsets of $\mathbb{R}^n$, given appropriate weights and activation functions. This means that MLPs can be useful for a wide range of tasks, including regression, classification, and time series prediction. The key to their success lies in the ability to learn non-linear decision boundaries by stacking multiple layers of neurons and using non-linear activation functions.

Depending on the task, the output layer can have one or more neurons. Consequently, the output layer's activation function can also vary. For example, in a binary classification task, a sigmoid activation function is often used in the output layer to produce a probability score between 0 and 1:

$$y = \frac{1}{1 + e^{-z}}$$



Note: For visual clarity, only a subset of connections is shown. In a fully connected MLP, each neuron in a layer is connected to every neuron in the adjacent layer.

Figure C.2: Structure of a Multi-Layer Perceptron (MLP) with four inputs, two hidden layers, and three outputs. Each layer is fully connected to the next layer, enabling the network to learn complex patterns and relationships in data.

where z is the weighted sum of inputs. The output can be interpreted as the probability of the input belonging to a specific class. If the output is greater than 0.5, the input is classified as one class; otherwise, it belongs to the other class.

In contrast, for multi-class classification tasks, a softmax activation function is typically employed to produce a probability distribution across multiple classes:

$$y_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

where y_i is the probability of class i, z_i is the weighted sum of inputs for class i, and k is the total number of classes. The softmax function ensures that the sum of probabilities across all classes equals 1, allowing for a clear interpretation of the model's predictions.

Other activation functions, such as the rectified linear unit (ReLU) and hyperbolic tangent (tanh), are commonly used in hidden layers. The ReLU activation function is defined as:

$$f(z) = \max(0, z)$$

where z is the weighted sum of inputs. The ReLU function introduces non-linearity into the model while maintaining computational efficiency. It is particularly effective in deep networks, as it helps mitigate the vanishing gradient problem - i.e. the issue of gradients becoming too small during backpropagation - allowing for faster convergence during training.

The tanh activation function is defined as:

$$f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

where z is the weighted sum of inputs. The tanh function produces outputs in the range of -1 to 1, making it suitable for tasks where negative values are meaningful. It is also a smooth and differentiable function, which is beneficial for gradient-based optimization methods.

Today every ANN is based on the MLP architecture.

The perceptron learning rule assumed a linear separation rule for the data, which limited its applicability to linearly separable problems. MLPs, instead, can learn non-linear decision boundaries by stacking multiple layers of neurons and using non-linear activation functions. Therefore, a new learning rule was needed to train MLPs: gradient descent (GD). While the perceptron learning rule adjusted weights only based on the error of the output neuron, GD computes the gradients of the loss function, which quantifies the difference between predicted and actual outputs, with respect to all weights in the network, that are adjusted iteratively to minimize the loss. Here is the formulation of the GD algorithm:

$$w_i \leftarrow w_i - \eta \frac{\partial L}{\partial w_i}$$

where L is the loss function, w_i is the weight for input x_i, and η is the learning rate. The gradients indicate how much each weight should be adjusted to minimize the loss. The learning rate determines the step size taken during weight updates, and it is crucial for convergence: a too high learning rate can lead to overshooting the minimum, while a too low learning rate can result in slow convergence [107].

C.3.1 Loss function

The loss function can be seen as a measure of how well the model is performing. In other words, it is a scalar value that indicates how far off the model's predictions are from the true values. By minimizing this loss during training, the model learns to make more accurate predictions. The choice of loss function depends on the specific task at hand. For example, in regression tasks, mean squared error (MSE) is commonly used, while in classification tasks, cross-entropy loss is often employed.

MSE is defined as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

where n is the number of samples, y_i is the true value, and $\hat{y}_i$ is the predicted value. MSE measures the average squared difference between predicted and actual values, penalizing larger errors more heavily.

Instead, cross-entropy loss is defined as:

$$\text{Cross-Entropy} = -\frac{1}{n}\sum_{i=1}^{n}\left(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right)$$

where y_i is the true label (0 or 1) and $\hat{y}_i$ is the predicted probability of the positive class. Cross-entropy loss measures the dissimilarity between the true distribution and the predicted distribution, encouraging the model to output probabilities that closely match the true labels.

C.3.2 Backpropagation

Backpropagation is the algorithm used to compute the gradients of the loss function with respect to the weights in an ANN. It is a key component of the GD algorithm and allows for efficient training of deep networks. The backpropagation algorithm consists of two main steps: forward propagation and backward propagation. During forward propagation, the input data is passed through the network, layer by layer, until it reaches the output layer. The output is compared to the true label using the loss function. The loss is then propagated backward through the network to compute the gradients of the loss function with respect to each weight. During backward propagation, the gradients are computed using the chain rule of calculus. The algorithm can be summarized in the following steps:

1. Initialize the weights and biases of the network.
2. For each training sample:
   (a) Perform forward propagation to compute the output of the network.
   (b) Compute the loss using the loss function.
   (c) Perform backward propagation to compute the gradients of the loss with respect to the weights and biases.
   (d) Update the weights and biases using the computed gradients and the learning rate.
3. Repeat steps 2-4 for a specified number of epochs or until convergence.

The backpropagation algorithm is efficient because it reuses the computed activations from the forward pass to calculate the gradients during the backward pass. This reduces the computational cost and allows for training deep networks with many layers. The algorithm can be applied to various types of neural networks, including feedforward networks, CNNs, and RNNs.

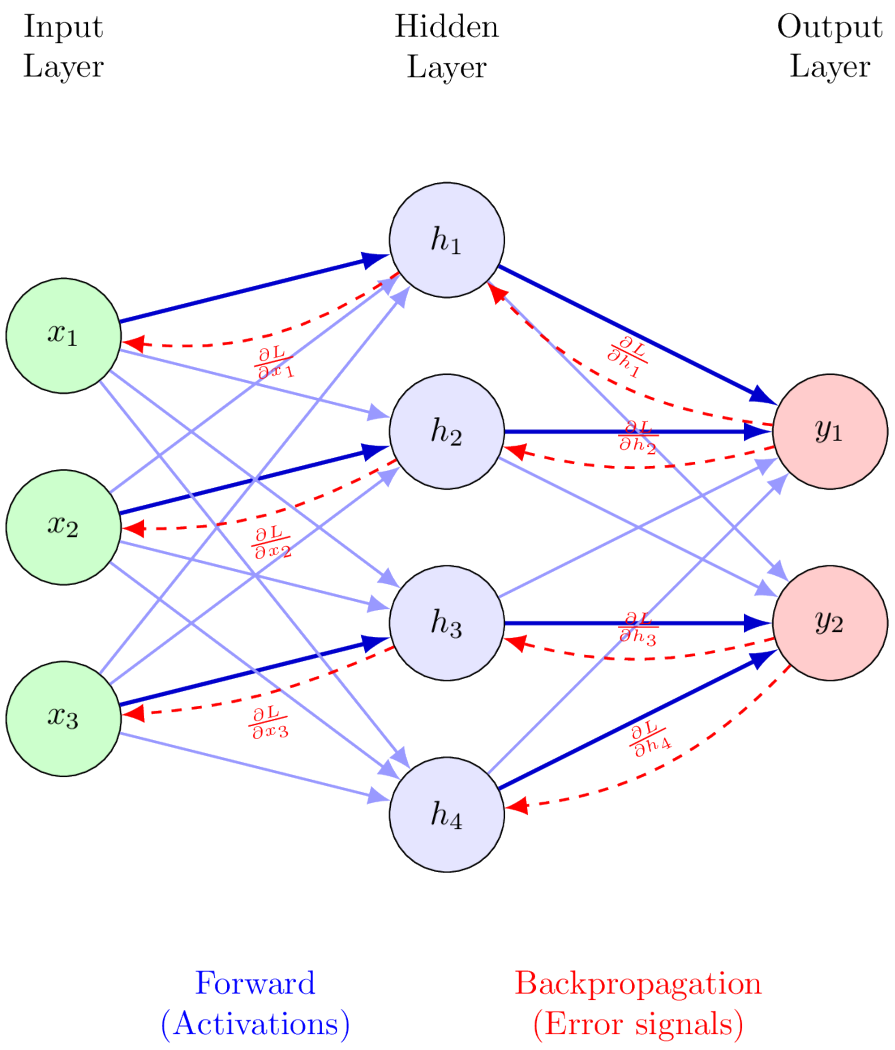
The backpropagation algorithm can be visualized as a computational graph, where the forward pass computes the output and the backward pass calculates the gradients of the loss function with respect to the weights. The graph illustrates how the gradients are propagated backward through the network, allowing for efficient weight updates during training [108]. Figure C.3 shows a simplified version of the backpropagation algorithm, highlighting the key steps involved in training an ANN.

C.3.3 Learning process

In the GD approach, weights are updated using an optimization algorithm. The most common optimization algorithm is stochastic gradient descent (SGD), which updates the weights based on a small batch of training samples rather than the entire dataset. This approach was introduced to limit the risk of the network getting stuck in local minima, which can occur when using the entire dataset for weight updates. By using just a portion of the dataset at a time, the model can explore different regions of the loss landscape and potentially find better solutions. The SGD algorithm updates the weights as follows:

$$w_i \leftarrow w_i - \eta \frac{1}{m} \sum_{j=1}^{m} \frac{\partial L}{\partial w_i}$$

where m is the batch size, and $\frac{\partial L}{\partial w_i}$ is the gradient of the loss function with respect to the weight w_i. The batch size determines how many samples are used in each iteration of the weight update. A smaller batch size leads to more frequent updates,


Input Layer
Hidden Layer
Output Layer
x_1
x_2
x_3
h_1
h_2
h_3
h_4
y_1
y_2
$\frac{\partial L}{\partial x_1}$
$\frac{\partial L}{\partial x_2}$
$\frac{\partial L}{\partial x_3}$
$\frac{\partial L}{\partial h_1}$
$\frac{\partial L}{\partial h_2}$
$\frac{\partial L}{\partial h_3}$
$\frac{\partial L}{\partial h_4}$
Forward (Activations)
Backpropagation (Error signals)

Note: Only a subset of backpropagation arrows is displayed for clarity.

Figure C.3: Backpropagation algorithm: the forward pass computes the output, while the backward pass calculates the gradients of the loss function with respect to the weights.

while a larger batch size results in fewer updates but with more accurate estimates of the gradients.

SGD is a powerful optimization algorithm that can converge faster than traditional gradient descent methods, especially in large datasets. However, it can also introduce noise into the weight updates, which can lead to oscillations and slower convergence. To mitigate this issue, various techniques have been developed, such as momentum, learning rate scheduling, and adaptive learning rates [109].

Momentum helps to smooth out the updates by considering the previous weight updates, allowing the model to build up speed in the direction of the optimal solution. It is calculated as follows:

$$v_i \leftarrow \beta v_i + (1-\beta)\frac{\partial L}{\partial w_i}$$

where v_i is the velocity term, β is the momentum coefficient (typically set to 0.9), and $\frac{\partial L}{\partial w_i}$ is the gradient of the loss function with respect to the weight w_i. The weights are then updated using the velocity term:

$$w_i \leftarrow w_i - \eta v_i$$

where w_i is the weight for input x_i, and η is the learning rate. The momentum term helps to accelerate the convergence process by allowing the model to maintain its direction of movement, reducing oscillations and improving stability.

Learning rate scheduling adjusts the learning rate over time, starting with a higher rate and gradually decreasing it as training progresses. This approach allows the model to explore the loss landscape more aggressively in the beginning and then fine-tune the weights as it converges to a solution. There are a lot of different learning rate scheduling techniques, each with its own advantages and disadvantages. The choice of learning rate schedule can significantly impact the convergence speed and overall performance of the model. Common learning rate scheduling techniques include step decay, exponential decay, and cosine annealing.

Step decay reduces the learning rate by a fixed factor after a certain number of epochs. For example, the learning rate can be halved every 10 epochs. Its formulation is as follows:

$$\eta_t = \eta_0 \cdot \gamma^{\left\lfloor \frac{t}{T} \right\rfloor}$$

where η_t is the learning rate at epoch t, η_0 is the initial learning rate, γ is the decay factor (e.g., 0.5), and T is the number of epochs after which the learning rate is reduced [110].

Exponential decay decreases the learning rate exponentially over time. It is defined as:

$$\eta_t = \eta_0 \cdot e^{-\lambda t}$$

where η_t is the learning rate at epoch t, η_0 is the initial learning rate, and λ is the decay rate [111].

Cosine annealing adjusts the learning rate using a cosine function, allowing for periodic increases and decreases in the learning rate. It is defined as:

$$\eta_t = \frac{1}{2}\left(1 + \cos\left(\frac{t}{T}\pi\right)\right) \cdot (\eta_0 - \eta_{\min}) + \eta_{\min}$$

where η_t is the learning rate at epoch t, η_0 is the initial learning rate, $\eta_{\min}$ is the minimum learning rate, and T is the total number of epochs. This approach allows for a gradual decrease in the learning rate while also providing opportunities for exploration [112].

Adaptive learning rates adjust the learning rate for each weight based on its historical gradients, allowing for more efficient convergence. One popular adaptive learning rate algorithm is Adam (Adaptive Moment Estimation), which combines the benefits of momentum and adaptive learning rates. Adam is an extension of SGD that computes the first and second moments of the gradients, allowing it to adaptively adjust the learning rates for each weight based on their historical gradients. The Adam update rule is as follows:

$$\begin{aligned}
m_i &\leftarrow \beta_1 m_i + (1 - \beta_1)\frac{\partial L}{\partial w_i} \\
v_i &\leftarrow \beta_2 v_i + (1 - \beta_2)\left(\frac{\partial L}{\partial w_i}\right)^2 \\
\hat{m}_i &= \frac{m_i}{1 - \beta_1^t} \\
\hat{v}_i &= \frac{v_i}{1 - \beta_2^t} \\
w_i &\leftarrow w_i - \eta\frac{\hat{m}_i}{\sqrt{\hat{v}_i} + \epsilon}
\end{aligned}$$

where m_i is the first moment (mean), v_i is the second moment (uncentered variance), β_1 and β_2 are the decay rates for the first and second moments (typically set to 0.9 and 0.999, respectively), $\hat{m}_i$ and $\hat{v}_i$ are the bias-corrected first and second moments, t is the current time step, and ϵ is a small constant added for numerical stability (typically set to 10^{-8}).

The Adam optimizer has become one of the most widely used optimization algorithms in deep learning due to its efficiency and effectiveness in training complex models. It is particularly well-suited for large datasets and high-dimensional parameter spaces, making it a popular choice for training deep neural networks. The combination of momentum and adaptive learning rates allows Adam to converge faster than traditional SGD methods, while also being more robust to noisy gradients and sparse data [113].

Appendix D

Climate data types

D.1 Paleoclimate proxies

Paleoclimate proxies are indirect indicators of past climate conditions, in fact they can provide valuable insights into long-term climate trends and variations over thousands of years. Paleoclimate proxies can be classified into biological, chemical and physical.

Biological proxies involve the remains of living organisms, such as pollen and spores, plant microfossils, charcoal, foraminifera and ostracodes, diatoms, corals, that reflect past environmental conditions. Pollen and spores are microscopic reproductive structures from plants, preserved in sediments, indicating past vegetation; plant microfossils are visible plant remains like leaves and seeds, providing insights into historical plant communities; charcoal residues from ancient fires, helping reconstruct past fire regimes; foraminifera and ostracodes are microscopic marine organisms whose shell compositions reflect past water temperatures and salinity; diatoms are silica[1]-based algae sensitive to environmental changes, useful for reconstructing past aquatic conditions; corals are marine invertebrates with growth patterns and chemical compositions that record sea surface temperatures and ocean chemistry.

Chemical proxies, e.g. stable isotopes, elemental analyses, biomarkers, biogenic silica, derive from the chemical composition of natural materials, reflecting environmental parameters. Stable isotopes are ratios of isotopes like oxygen-18 to oxygen-16 in materials such as ice cores and marine sediments, indicating past temperatures and precipitation; elemental analyses involve measuring the concentrations of elements like iron and phosphorus in sediments, revealing information

[1]Silica, or silicon dioxide (SiO_2), is a naturally occurring compound composed of silicon and oxygen [114].

about erosion, productivity, and land use changes; biomarkers are organic molecules from specific organisms, preserved in sediments, indicating past biological activity and environmental conditions; biogenic silica, i.e. silica from organisms like diatoms, is used to infer past productivity and environmental changes in aquatic systems.

Physical proxies, like sediment composition, texture and structure, color, density, magnetic properties, involve the physical characteristics of sediments and geological formations. Sediment composition, i.e. types of minerals and fossils in sediments, inform about past salinity and temperature; texture and structure, i.e. grain size and layering in sediments, indicate transportation processes and depositional environments; sediment color variations can reflect changes in organic content, oxidation conditions, and source materials; density (mass per unit volume of sediments) provides clues about composition and depositional conditions; magnetization of segments is useful for identifying sediment sources and dating through Earth's magnetic field reversals [115].

Examples of datasets based on paleoclimate proxies include NOAA National Centers for Environmental Information (NCEI) Paleoclimatology Data [116], Past Global Changes (PAGES) 2k Network [117] and Neotoma Paleoecology Database [118].

While paleoclimate data are invaluable, they present some challenges. The interpretation of paleoclimate proxies can be complex, as they often require careful calibration and validation against modern observations. Additionally, the dating of sediment cores and other geological materials can introduce uncertainties, particularly when relying on radiometric dating techniques or stratigraphic correlations. Moreover, paleoclimate proxies may have limitations in terms of spatial coverage and temporal resolution, for instance they may not capture short-term fluctuations or recent changes in climate.

D.2 In-situ observations

In-situ observations, or ground-based measurements, are direct measurements collected at the location of interest, providing essential data for understanding and monitoring the Earth's climate system. They are a more straightforward way to collect climate data, compared to paleoclimate proxies, and are crucial for calibrating satellite data, validating climate models, and informing policy decisions. Measurements are collected using various instruments, which can be classified in marine/oceanographic, atmospheric, and terrestrial. Marine and oceanographic instruments can monitor surface temperature, ocean temperature, salinity, currents and wave height. They include include moored buoys, which are anchored to the sea floor, drifting buoys, that move with ocean currents, argo floats, which can go up to a depth of 2000 meters; AUVs (Autonomous Underwater Vehicle), that can traverse the ocean, research vessels, i.e. ships equipped with advanced sensors, and tide gauges, that are installed along coastlines and measure sea level changes over time [119]. Atmospheric platforms include weather stations, radiosondes and air quality monitors. Weather stations are fixed installations measuring parameters like temperature, humidity, wind speed, and precipitation; radiosondes are balloon-borne instruments that ascend through the atmosphere, recording vertical profiles of temperature, pressure, and humidity; air quality monitors are devices that track pollutants such as nitrogen dioxide (NO_2), particulate matter ($PM_{2.5}$), and ozone levels. Terrestrial platforms comprehend hydrological stations, cryospheric measurements and vegetation monitoring stations. Hydrological stations monitor river discharge, groundwater levels, and soil moisture; cryospheric[2] measurements assess snow depth, ice thickness, and glacier movement; vegetation monitoring stations evaluate land cover, biomass, and photosynthetic activity. Examples of datasets based on in-situ observations include United States Climate Reference Network (USCRN) [121] and European Climate Assessment (ECA) [122]. Although in-situ observations provide valuable data, they also have limitations. The spatial coverage of in-situ measurements can be uneven, particularly in remote or inaccessible regions, leading to gaps in data availability. Additionally, in-situ measurements can be affected by local environmental conditions, such as urban heat islands or land use changes, which may introduce biases in the data. Furthermore, the maintenance and calibration of ground-based instruments are essential to ensure data quality and reliability.

D.3 Satellite data

Satellite remote sensing involves using instruments aboard satellites to collect data about Earth's surface and atmosphere without direct contact. Therefore, with respect to in-situ measurements, satellite data is collected far from the location of interest, but it can provide a more comprehensive view of large areas and insights at different spatial and temporal resolutions, thus complementing ground-based measurements. Satellite instruments capture electromagnetic radiation (such as visible light, infrared, and microwave) reflected or emitted by Earth's features.

They can be divided into passive and active. Passive sensors detect natural energy emitted or reflected by objects on Earth. They rely on external sources of energy, primarily the Sun, to illuminate the target. They can cover large areas quickly and are useful for monitoring surface temperatures, vegetation and ocean color, but they depend on sunlight limits observations to daytime and measurements

[2]The cryosphere encompasses the portions of Earth's surface where water exists in solid form [120].

can be disturbed by cloud cover. Active sensors emit their own energy towards the Earth and measure the reflected or backscattered signals. This allows them to collect data regardless of external lighting conditions. They operate day and night, regardless of weather conditions, and provide high-resolution data on surface and atmospheric features, but typically cover smaller areas compared to passive sensors and are more complex and expensive to operate [123].

Sensors can also be classified based on the type of data they collect. Optical sensors detect sunlight reflected from Earth's surface. They can be panchromatic, i.e. capture images in a single broad band, producing high-resolution black-and-white images, multispectral, which capture data in multiple discrete bands (typically 3-10), including visible and near-infrared wavelengths, or hyperspectral, that capture data in hundreds of narrow, contiguous spectral bands, allowing for detailed analysis of materials and land cover. Radar sensors can be Synthetic Aperture Radar (SAR), i.e. they emit microwave signals and measure their return, enabling imaging regardless of weather or lighting conditions, and Interferometric SAR (InSAR), which combine multiple SAR images to detect ground surface changes, useful for monitoring subsidence, earthquakes, and glacier movements. Thermal sensors measure emitted infrared radiation to assess surface temperatures, aiding in studies of urban heat islands, wildfires, and energy efficiency. Light Detection and Ranging (LIDAR) sensors use laser pulses to generate high-resolution 3D models of Earth's surface, valuable for topographic mapping and vegetation analysis [124].

Examples of datasets based on passive sensors include the NOAA Climate Data Record (CDR) of Advanced Very High Resolution Radiometer (AVHRR) Reflectance [125], the Moderate Resolution Imaging Spectroradiometer (MODIS) Land Surface Temperature and Emissivity (MOD11) [126], and the Visible Infrared Imaging Radiometer Suite (VIIRS) Surface Reflectance CDR [127]. Examples of datasets based on active sensors include instead Sentinel-1 Synthetic Aperture Radar (SAR) Data [128], Global Ecosystem Dynamics Investigation (GEDI) LIDAR [129], and the Advanced Microwave Scanning Radiometer (AMSR) Unified L3 Daily [130].

Although satellite data provides valuable information, it also has limitations. The fact that satellite data is available at different spatial and temporal resolutions is both an advantage and a limitation, because it can lead to inconsistencies in data quality and interpretation. For example, high-resolution satellite imagery can capture fine details of land cover changes, while coarser resolution data may miss important features or variations. This can affect the accuracy of analyses and models that rely on satellite data. Additionally, the temporal resolution of satellite data can vary, with some sensors providing frequent observations while others have longer revisit times. This can impact the ability to monitor rapid changes in climate or land cover, such as extreme weather events or deforestation. Moreover, satellite records may have temporal gaps, they are limited to the period since satellite observations began, and ensuring consistency across different satellite instruments and missions requires careful calibration and validation. However, compared to paleoclimate proxies and in-situ observations, satellite data is much more used by researchers.

D.4 Climate models

Climate models are mathematical representations of the Earth's climate system, simulating the interactions between various components, such as the atmosphere, oceans, land surface, and ice. They are used to understand past climate changes, assess current conditions, and project future climate scenarios. Climate models can be classified into different categories based on their complexity and spatial resolution: Energy Balance Models (EBMs), Radiative-Convective Models (RadCMs), Earth system Models of Intermediate Complexity (EMICs), General Circulation Models (GCMs), Regional Climate Models (RCMs) and Earth System Models (ESMs).

EBMs are simple representations of the Earth's energy budget, focusing on the balance between incoming solar radiation and outgoing thermal radiation; RadCMs extend this concept by incorporating vertical temperature profiles and convection processes in the atmosphere [131]; EMICs are simplified models that capture essential climate processes while reducing computational complexity, making them suitable for long-term climate simulations [132]; GCMs are more complex models that simulate the three-dimensional circulation of the atmosphere and oceans, including interactions between different components of the climate system; RCMs are used to downscale GCM projections to finer spatial resolutions, providing more localized climate information; ESMs are the most comprehensive models, integrating physical, chemical, and biological processes in the climate system, such as the carbon cycle and vegetation dynamics [133, 134].

Most of the climate models are some computational limitations, so they cannot resolve all physical processes explicitly. Therefore, they rely on parameterizations, which involve representing small-scale processes, like cloud formation and convection, through simplified relationships, which is crucial for capturing their effects on the larger-scale climate system [135]. Moreover, some models have coarse-resolution grids, and they are translated into finer-resolution grids using downscaling techniques. These techniques include statistical downscaling, which uses relationships between large-scale climate variables and local observations to create high-resolution projections, and dynamical downscaling, which involves running a high-resolution model nested within a coarser-resolution model to capture regional climate features [136].

Climate models are useful to make in-depth analysis of single climatical processes, but they are subject to individual biases and uncertainties and may not capture the full range of possible climate outcomes. Therefore, today researchers tend to prefer Multi-Model Ensembles (MMEs). MMEs involve running several climate models under the same initial conditions and external forcings (such as greenhouse gas concentrations). The individual model outputs are then combined, typically through averaging, to produce a consensus forecast. MMEs capture a range of possible outcomes, reflecting the uncertainties inherent in climate modeling. Therefore, they provide a more robust assessment of future climate scenarios. MMEs can be constructed using different approaches, such as simple averaging, weighted averaging based on model performance, or more sophisticated statistical techniques. MMEs are widely used in climate research and policy-making. For example, the Coupled Model Intercomparison Project (CMIP) provides a framework for comparing outputs from various global climate models, facilitating the development of comprehensive climate assessments [137]. Similarly, the NCEI North American Multi-Model Ensemble (NMME) is employed for seasonal climate forecasting, integrating models from multiple institutions to improve prediction accuracy [138]. Still, MMEs are not without limitations: they can be computationally expensive, requiring significant resources to run multiple models simultaneously; the choice of models included in the ensemble can influence the results, and there is no universally accepted method for selecting or weighting models; they may not fully capture the range of uncertainties associated with climate projections, particularly in regions with limited observational data or complex climate dynamics; if all models share similar biases, the ensemble mean may still be skewed.

D.5 Reanalysis datasets

The most used and appreciated datasets in climate science are reanalysis datasets. Reanalysis datasets are produced by assimilating observational data into numerical weather prediction models, resulting in a consistent and high-resolution representation of the atmosphere over time. They combine observations from various sources, such as satellites, weather stations, and buoys, with model simulations to create a comprehensive picture of the Earth's climate system. Reanalysis datasets are valuable for climate monitoring, research, policy making, risk assessment in commercial applications. Therefore, they are also useful for analyzing climate teleconnections. They could still be affected by biases, particularly in regions with sparse observational data or complex topography, but they are generally considered more reliable than individual observational datasets. Reanalysis datasets are produced by several institutions, including the European Centre for Medium-Range Weather Forecasts (ECMWF) [139], the National Oceanic and Atmospheric Administration (NOAA) [140], and the Japan Meteorological Agency (JMA) [141]. The most common reanalysis datasets include the ERA5 (European Reanalysis), NCEI

Reanalysis 1 and 2, MERRA-2 (Modern-Era Retrospective Analysis for Research and Applications) and JRA-55 (Japanese Reanalysis).

NCEP/NCAR Reanalysis 1 is a global atmospheric reanalysis dataset produced by the National Centers for Environmental Prediction (NCEP) and the National Center for Atmospheric Research (NCAR). Spanning from 1948 to the present, it offers a comprehensive view of the Earth's climate system. The dataset includes various atmospheric variables, such as temperature, humidity, wind speed, and precipitation, at different pressure levels and surface levels. The data is produced using a combination of numerical weather prediction models and observational data assimilation techniques. The dataset differs from ERA5 for the data assimilation methods: while NCEP/NCAR Reanalysis 1 employs a sequential data assimilation approach, integrating observational data into a forecast model to produce a consistent atmospheric analysis, ERA5 utilizes a more advanced data assimilation system, incorporating a 4D-Var approach that considers both spatial and temporal variations in the assimilation process. This results in ERA5 providing higher temporal and spatial resolution data compared to NCEP/NCAR Reanalysis 1. Additionally, ERA5 offers hourly estimates for a wide range of atmospheric, ocean-wave, and land-surface variables, with uncertainty estimates provided by a 10-member ensemble at three-hour intervals. In contrast, NCEP/NCAR Reanalysis 1 primarily provides daily and monthly mean data, with less frequent temporal resolution. Furthermore, ERA5 is updated daily with a delay of about five days, ensuring more timely data availability, while NCEP/NCAR Reanalysis 1 has a longer update cycle, which may result in less current data for analysis [142]. NCEP/DOE Reanalysis 2 is an updated version of NCEP/NCAR Reanalysis 1 realized by the Department of Energy (DOE). It has improved data assimilation techniques and a more comprehensive representation of the Earth's climate system. It covers the period from 1979 to the present, offering enhanced spatial and temporal resolution. The dataset includes a wider range of atmospheric variables and provides better estimates of precipitation and other hydrological variables. These datasets provide data with a resolution of $2.5° \times 2.5°$ and 17 vertical levels, extending from the surface to 10 hPa (approximately 30 km altitude) [143].

MERRA-2 is a comprehensive atmospheric reanalysis dataset produced by NASA's Global Modeling and Assimilation Office (GMAO). Covering the period from 1980 to the present, MERRA-2 offers high-resolution data on various atmospheric variables, including temperature, humidity, wind speed, and precipitation. MERRA-2 is provided in multiple collections tailored to different research needs: the INST collection delivers hourly instantaneous meteorological and aerosol-coupled fields, the FLX collection gives grid-box-average surface turbulent fluxes and related energy/water-budget quantities, and the LND collection isolates land-only model variables (e.g., soil moisture, vegetation states) without fractional-area weighting. All analyses are performed on NASA GMAO's cubed-sphere grid at

$0.5° \times 0.625°$ (~50 km) resolution with 72 hybrid-eta levels up to 0.01 hPa, and include representations of ice sheets over Greenland and Antarctica for improved surface coupling. Data assimilation in MERRA-2 uses the GEOS-5 (Goddard Earth Observing System) Data Assimilation System (version 5.12.4) with a 3D-Var[3] framework plus incremental analysis updates, jointly ingesting meteorological observations and space-based Aerosol Optical Depth (AOD) retrievals. This joint aerosol-meteorology assimilation, the first of its kind in a multi-decadal reanalysis, allows full radiative coupling of analyzed aerosol fields back into the climate model, significantly improving simulations of dust, smoke, and other particulates and their climate impacts [145]. New analyses are produced in near real time, with hourly meteorological fields and 3-hourly aerosol updates; monthly files undergo additional quality checks and are released around the 15th-20th of the following month. MERRA-2 also offers specialized collections such as RAD (radiation diagnostics), CLD (cloud parameters), MST (moist processes), TRB (turbulence), SLV (single-level fields), CHM (chemical forcings), and AER/ADG (aerosol diagnostics and extended diagnostics) [146].

JRA-55 is a global atmospheric dataset produced by JMA that spans from 1958 to the present, offering a comprehensive and homogeneous record of past climate conditions for research and operational applications. By assimilating an extensive array of observations—including radiosondes, surface stations, aircraft reports, and satellite radiances—within a 4D-Var framework, JRA-55 resolves many of the temporal inconsistencies and biases present in earlier reanalyses. The inception of JRA-55 builds upon the earlier JRA-25 product, which covered the period 1979-2004, and then was subject to a set of updates and improvements [147]. JRA-55 employs a spectral model at TL319[4] horizontal resolution with 60 vertical σ-levels[5] reaching up to 0.1 hPa. The data assimilation system integrates observations over a six-hour window using 4D-Var, augmented by a variational bias correction for satellite radiances to mitigate instrument and retrieval biases. This framework also incorporates time-varying concentrations of greenhouse gases (CO_2, CH_4, N_2O), improving the representation of radiative processes and long-term climate signals. Covering the full globe on a regular latitude-longitude grid, JRA-55 delivers three-hourly, six-hourly, daily, and monthly fields of core atmospheric variables, including temperature, winds, geopotential height, humidity,

[3]3D-Var is a data assimilation technique that considers data as a single time point. It is less computationally intensive than 4D-Var [144].

[4]TL319 refers to the triangular truncation of the spectral model, which determines the horizontal resolution of the model. The number 319 indicates that the model has a total of 319 wave numbers in each direction, resulting in a grid spacing of approximately 60 km [148].

[5]σ-levels are a coordinate system used in atmospheric models where the vertical coordinate is defined as the ratio of the pressure at a given level to the surface pressure [149].

and surface pressure. The dataset also provides surface and flux products—such as precipitation, radiation components, evaporation, and soil moisture—as well as diagnostics relevant for climate studies, including storm-track indices, tropical cyclone metrics, and Madden-Julian Oscillation diagnostics [150].

Appendix E

Mathematical proofs

E.1 Grid distortion

Let $\mathbf{p}_1 = [\mathbf{c}_1, \mathbf{f}_1]$ and $\mathbf{p}_2 = [\mathbf{c}_2, \mathbf{f}_2]$ be two points at the same time step, i.e. $y_1 = y_2$ and $m_1 = m_2$, but with different coordinates, i.e. $\ell_1 \neq \ell_2$ and $g_1 \neq g_2$. Then, the distance between the two points (assuming flat Earth) is given by:

$$d(\mathbf{p}_1, \mathbf{p}_2) = R\sqrt{(\ell_1 - \ell_2)^2 + (g_1 - g_2)^2}$$

where R is the radius of the Earth, which is approximately 6371 km. If $\ell_1 = \ell_2$, then the distance is:

$$d(\mathbf{p}_1, \mathbf{p}_2) = R|g_1 - g_2|$$

If $g_1 = g_2$, then the distance is:

$$d(\mathbf{p}_1, \mathbf{p}_2) = R|\ell_1 - \ell_2|$$

If both $\ell_1 = \ell_2$ and $g_1 = g_2$, then the distance is zero, i.e. the two points are the same. However, on a spherical surface the situation is different, and the Haversine formula can be used:

$$d(\mathbf{p}_1, \mathbf{p}_2) = 2R \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{\ell_1 - \ell_2}{2}\right) + \cos(\ell_1) \cdot \cos(\ell_2) \cdot \sin^2\left(\frac{g_1 - g_2}{2}\right)}\right)$$

At the equator, $\ell_1 = \ell_2 = 0$. Therefore, the distance becomes:

$$\begin{aligned}
d(\mathbf{p}_1, \mathbf{p}_2) &= 2R \cdot \arcsin\left(\sqrt{\sin^2(0) + \cos(0) \cdot \cos(0) \cdot \sin^2\left(\frac{g_1 - g_2}{2}\right)}\right) \\
&= 2R \cdot \arcsin\left(\sqrt{0 + 1 \cdot 1 \cdot \sin^2\left(\frac{g_1 - g_2}{2}\right)}\right) \\
&= 2R \cdot \arcsin\left(\sin\left(\frac{g_1 - g_2}{2}\right)\right) \\
&= 2R \cdot \frac{g_1 - g_2}{2} \\
&= R|g_1 - g_2|
\end{aligned}$$

so it is equivalent to the flat Earth case. Instead, at the poles $\ell_1 = \ell_2 = \frac{\pi}{2}$. Then:

$$\begin{aligned}
d(\mathbf{p}_1, \mathbf{p}_2) &= 2R \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{\pi}{2} - \frac{\pi}{2}\right) + \cos\left(\frac{\pi}{2}\right) \cdot \cos\left(\frac{\pi}{2}\right) \cdot \sin^2\left(\frac{g_1 - g_2}{2}\right)}\right) \\
&= 2R \cdot \arcsin\left(\sqrt{0 + 0 \cdot 0 \cdot \sin^2\left(\frac{g_1 - g_2}{2}\right)}\right) \\
&= 2R \cdot \arcsin(0) = 0
\end{aligned}$$

E.2 Area-weighted grid2mesh average pooling

The calculation of the area of a grid cell requires computing an integral:

$$A_{\mathbf{g}} = \int_{S_{\mathbf{g}}} dA$$

where $S_{\mathbf{g}}$ is the surface of the grid cell $\mathbf{g}$. The area can be computed as a double integral over the latitude ℓ and longitude g of the grid cell, considering that the area element on the sphere is given by:

$$dA = R^2 \cdot \cos(\ell)\, dg\, d\ell$$

Therefore:

$$A_{\mathbf{g}} = \int_{\ell_{\mathbf{g}-}}^{\ell_{\mathbf{g}+}} \int_{g_{\mathbf{g}-}}^{g_{\mathbf{g}+}} R^2 \cdot \cos(\ell)\; dg\, d\ell$$

where R is the radius of the Earth, $\ell_{\mathbf{g}-}$ and $\ell_{\mathbf{g}+}$ are the lower and upper latitudes of the grid cell, and $g_{\mathbf{g}-}$ and $g_{\mathbf{g}+}$ are the left and right longitudes of the grid cell. The result is:

$$A_{\mathbf{g}} = R^2 \cdot (g_{\mathbf{g}+} - g_{\mathbf{g}-})\,(\sin(\ell_{\mathbf{g}+}) - \sin(\ell_{\mathbf{g}-}))$$

where $\ell_{\mathbf{g}-}, \ell_{\mathbf{g}+}$ are the lower and upper latitudes, and $g_{\mathbf{g}-}, g_{\mathbf{g}+}$ are the left and right longitudes of the grid cell. The area-weighted average pooling is then defined as:

$$\mathbf{f}_k = \frac{\sum_{\mathbf{g}\in\mathcal{G}_k} A_{\mathbf{g}} \cdot \mathbf{f}_{\mathbf{g}}}{\sum_{\mathbf{g}\in\mathcal{G}_k} A_{\mathbf{g}}}$$

We can simplify this equation by assuming that, when computing $\mathbf{f}_k$, we are only considering adjacent grid cells, there is no need for very high spatial precision and variables change smoothly. Therefore, let:

$$\ell_{\mathbf{g}+} = \ell_{\mathbf{g}} + \frac{\Delta\ell_{\mathbf{g}}}{2}, \quad \ell_{\mathbf{g}-} = \ell_{\mathbf{g}} - \frac{\Delta\ell_{\mathbf{g}}}{2}$$

where $\ell_{\mathbf{g}}$ is a latitude considered as center of the area. Then, for the mean value theorem, we can approximate the difference between the sine of the upper and lower latitudes using the derivative of the sine function:

$$\begin{aligned}\sin(\ell_{\mathbf{g}+}) - \sin(\ell_{\mathbf{g}-}) &= \sin\left(\ell_{\mathbf{g}} + \frac{\Delta\ell_{\mathbf{g}}}{2}\right) - \sin\left(\ell_{\mathbf{g}} - \frac{\Delta\ell_{\mathbf{g}}}{2}\right) \\ &\approx \frac{d\sin(\ell_{\mathbf{g}})}{d\ell_{\mathbf{g}}} \cdot \Delta\ell_{\mathbf{g}} \\ &= \cos(\ell_{\mathbf{g}}) \cdot \Delta\ell_{\mathbf{g}}\end{aligned}$$

As a result, the area of the grid cell can be approximated as:

$$A_{\mathbf{g}} \approx R^2 \cdot \Delta\ell_{\mathbf{g}} \cdot \Delta g_{\mathbf{g}} \cdot \cos(\ell_{\mathbf{g}})$$

Since R, $\Delta\ell_{\mathbf{g}}$ and $\Delta g_{\mathbf{g}}$ are constant for all grid cells:

$$\mathbf{f}_k \approx \frac{\sum_{\mathbf{g}\in\mathcal{G}_k} R^2 \cdot \Delta\ell_{\mathbf{g}} \cdot \Delta g_{\mathbf{g}} \cdot \cos(\ell_{\mathbf{g}}) \cdot \mathbf{f}_{\mathbf{g}}}{\sum_{\mathbf{g}\in\mathcal{G}_k} R^2 \cdot \Delta\ell_{\mathbf{g}} \cdot \Delta g_{\mathbf{g}} \cdot \cos(\ell_{\mathbf{g}})} = \frac{\sum_{\mathbf{g}\in\mathcal{G}_k} \cos(\ell_{\mathbf{g}}) \cdot \mathbf{f}_{\mathbf{g}}}{\sum_{\mathbf{g}\in\mathcal{G}_k} \cos(\ell_{\mathbf{g}})}$$

E.3 Mean value of the grid2mesh cardinality

It is trivial to show that the area of the spherical approximation of the Earth is:

$$A = 4\pi R^2$$

Therefore, the area of a mesh cell is:

$$A_{\text{mesh}} = \frac{A}{M} = \frac{4\pi R^2}{M}$$

From previous considerations, we know that the area of a grid cell is:

$$A_{\text{grid}} = R^2 \cdot \Delta\ell_{\text{grid}} \cdot \Delta g_{\text{grid}} \cdot \cos(\ell_{\text{grid}})$$

By combining the two areas, we can express α as function of the latitude:

$$\alpha(\ell) = \frac{A_{\text{mesh}}}{A_{\text{grid}}} = \frac{4\pi R^2}{M} \cdot \frac{1}{R^2 \cdot \Delta\ell_{\text{grid}} \cdot \Delta g_{\text{grid}} \cdot \cos(\ell)} = \frac{4\pi}{M \cdot \Delta\ell_{\text{grid}} \cdot \Delta g_{\text{grid}} \cdot \cos(\ell)}$$

The mean value of the integral over a surface is defined as:

$$\bar{\alpha} = \frac{1}{A} \int_{\mathcal{S}} f \, dA$$

where f is a function defined on the surface $\mathcal{S}$, and A is the area of the surface. We know A and dA for the Earth, so we can rewrite:

$$\begin{aligned}
\bar{\alpha} &= \frac{1}{4\pi R^2} \int_0^{2\pi} \int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \alpha(\ell) \cdot R^2 \cdot \cos(\ell) \, d\ell \, dg \\
&= \frac{1}{4\pi R^2} \int_0^{2\pi} \int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \frac{4\pi}{M \cdot \Delta\ell_{\text{grid}} \cdot \Delta g_{\text{grid}} \cdot \cos(\ell)} \cdot R^2 \cdot \cos(\ell) \, d\ell \, dg \\
&= \frac{1}{M \cdot \Delta\ell_{\text{grid}} \cdot \Delta g_{\text{grid}}} \int_0^{2\pi} \int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} d\ell \, dg \\
&= \frac{1}{M \cdot \Delta\ell_{\text{grid}} \cdot \Delta g_{\text{grid}}} \cdot 2\pi \cdot \left(\frac{\pi}{2} - \left(-\frac{\pi}{2}\right)\right) \\
&= \frac{2\pi^2}{M \cdot \Delta\ell_{\text{grid}} \cdot \Delta g_{\text{grid}}}
\end{aligned}$$

E.4 Graph neighborhood

When handling an icosahedron geometry, i.e. before any mesh refinement, the properties of the graph can be inferred from the ones of the polyhedron itself. Let f be the number of edges per face and d the number of edges per vertex. Icosahedron is one of the five Platonic solids [151], therefore all its faces are equilater triangles ($f = 3$), all edges have the same length and the same number of faces and edges meet at each vertex (d is constant for all vertices). Since each edge connects 2 vertices and is shared by 2 faces, it is trivial to show that $f \cdot F = 2E$, and also $d \cdot V = 2E$. Therefore:

$$d = \frac{f \cdot F}{V}$$

Since $F = 20$ and $V = 12$, we can conclude that $d = 5$, i.e. each vertex is connected to 5 other vertices, which are its neighbors. However, this does not apply to refined meshes, because they do not represent Platonic shapes anymore, so we have no guarantee that the number of neighbors is constant for all mesh points. Given L the mesh refinement level, since all faces are still triangles, and each edge is shared by 2 faces:

$$E(L) = \frac{3}{2} F(L)$$

Starting from $F(0) = 20$, at every refinement level L the number of faces is multiplied by 4, so we can express the number of faces as:

$$F(L) = 20 \cdot 4^L$$

We use the Euler's formula for polyhedra [152]:

$$V - E + F = 2$$

From this, we can express the number of vertices as:

$$V(L) = E(L) - F(L) + 2 = \frac{1}{2}F(L) + 2 = 10 \cdot 4^L + 2$$

The degree $d(L)$ of a vertex at level L can then be expressed as:

$$d(L) = \frac{3F(L)}{V(L)} = \frac{60 \cdot 4^L}{10 \cdot 4^L + 2}$$

This function is strictly increasing, has a minimum in $d(0) = 5$, and tends to:

$$\lim_{L \to +\infty} d(L) = 6$$

When refining the mesh, points in the coarse mesh exist also in the fine one. It can be shown that they preserve their degree, and so $d(L)$ increases because all the added points in the fine mesh have degree 6.

What was discussed until now is the topological meaning of neighbors, i.e. the fact that two mesh points are neighbors if they are connected by an edge (side) in the mesh representation. However, we can also define a geometric meaning of neighbors, i.e. two mesh points are neighbors if they are close enough in space, regardless of the mesh structure. This is a more general definition, because it does not depend on the mesh representation, but only on the spatial distance between points. However, if we choose the geometric definition, we should statically define a k-NN (k-Nearest Neighbors) approach, i.e. always select the k nearest neighbors for each mesh point according to some distance metric, where k is a constant. Instead, we want to exploit the properties of our mesh representation, and therefore we opt for the topological definition. It can be empirically proven that a topological neighbor $v_{t,s'}$ of a mesh point $v_{t,s}$ is also a geometric neighbor if:

$$\delta(v_{t,s}, v_{t,s'})^{(L)} < \delta_{\min}^{(L)} \cdot \varphi \quad \forall s' \in \mathcal{N}_s$$

where $\delta(v_{t,s}, v_{t,s'})^{(L)}$ is the distance between the mesh points $v_{t,s}$ and $v_{t,s'}$ at level L, $\delta_{\min}^{(L)}$ is the minimum distance between two mesh points at level L. Of course, as L increases, $\delta_{\min}^{(L)}$ decreases, so the condition becomes more restrictive. In particular, the following holds:

$$\delta_{\min}^{(L)} = \frac{\delta_{\min}^{(0)}}{2^L}$$

E.5 Row-stochastic correlation matrix

- **Base case:** $k = 0$. In this case, we have:

$$\mathbf{C}_L(0) = \mathbf{I}$$

  where $\mathbf{I}$ is the identity matrix. The identity matrix is a row-stochastic matrix, since each row sums to 1.

- **Inductive step:** Assume that the statement holds for $k = m$, i.e. $\mathbf{C}_L(m)$ is a row-stochastic matrix. We want to prove that $\mathbf{C}_L(m+1)$ is also a row-stochastic matrix. We introduce the following notation: given a matrix $\mathbf{M}$, we denote by $[\mathbf{M}]_{i,j}$ the element in the i-th row and j-th column of $\mathbf{M}$. We have:

$$\mathbf{C}_L(m+1) = \mathbf{C}_L(m) \cdot \mathbf{A}(L-m)$$

  $\mathbf{A}(L-m)$ is a row-stochastic matrix:

$$\sum_{j=1}^{n} [\mathbf{A}(L-m)]_{i,j} = 1 \quad \forall i \in \{1, \dots, n\}$$

  $\mathbf{C}_L(m)$, as from the inductive hypothesis, is a row-stochastic matrix:

$$\sum_{j=1}^{n} [\mathbf{C}_L(m)]_{i,j} = 1 \quad \forall i \in \{1, \dots, n\}$$

  Then:

$$\begin{aligned}
\sum_{j=1}^{n} [\mathbf{C}_L(m+1)]_{i,j} &= \sum_{j=1}^{n} \left(\sum_{l=1}^{n} [\mathbf{C}_L(m)]_{i,l} \cdot [\mathbf{A}(L-m)]_{l,j} \right) \\
&= \sum_{l=1}^{n} [\mathbf{C}_L(m)]_{i,l} \cdot \left(\sum_{j=1}^{n} [\mathbf{A}(L-m)]_{l,j} \right) \\
&= \sum_{l=1}^{n} [\mathbf{C}_L(m)]_{i,l} \cdot 1 \\
&= \sum_{l=1}^{n} [\mathbf{C}_L(m)]_{i,l} \\
&= 1
\end{aligned}$$

  for all $i \in \{1, \dots, n\}$.

By the principle of mathematical induction, we conclude that $\mathbf{C}_L(k)$ is a row-stochastic matrix for all $k \geq 0$.

E.6 Path interpretation of the correlation matrix

- **Iteration 0:** $k = 0$. In this case, we have:

$$\mathbf{R}_L(0) = \mathbf{I}$$

  where $\mathbf{I}$ is the identity matrix. No consideration can be done at this point.

- **Iteration 1:** $k = 1$. In this case, we have:

$$\mathbf{R}_L(1) = \mathbf{A}(L)$$

  Therefore, the correlation $c_{i,j}^{1,L}$ is given by:

$$c_{i,j}^{1,L} = \alpha_{v_i,v_j}^{L}$$

  and represents a path of length 1 from node v_j to v_i, i.e.:

$$P_{v_j \to v_i} = (e_{v_j,v_i})$$

- **Iteration 2:** $k = 2$. In this case, we have:

$$\mathbf{R}_L(2) = \mathbf{A}(L) \cdot \mathbf{A}(L-1)$$

  The correlation $c_{i,j}^{2,L}$ is given by:

$$c_{i,j}^{2,L} = \sum_{l=1}^{n} \alpha_{i,l}^{L} \cdot \alpha_{l,j}^{L-1}$$

  We know that $\mathbf{R}_L(2)$ can be actually represented as a block matrix, where each block corresponds to a time step t and a spatial point s. Taking into account how input data is structured, we can rewrite the correlation as:

$$c_{[t,s],[t',s']}^{2,L} = \sum_{s''=1}^{S} \sum_{t''=1}^{T} \alpha_{[t,s],[t'',s'']}^{L} \cdot \alpha_{[t'',s''],[t',s']}^{L-1}$$

  As from Equation 3.13 and Equation 3.14, the attention weight $\alpha_{[t,s],[t'',s'']}^{L}$ is non-zero only if $t'' = t - 1$ and $s'' \in \mathcal{N}_s$. Therefore, we can rewrite the correlation as:

$$c_{[t,s],[t',s']}^{2,L} = \sum_{s'' \in \mathcal{N}_s} \alpha_{[t,s],[t-1,s'']}^{L} \cdot \alpha_{[t-1,s''],[t',s']}^{L-1}$$

where $t' = t - 2$ and $s' \in \mathcal{N}_{s''}$[1]. This sum takes into account all paths of length 2 from node $[t, s]$ to node $[t', s']$, where each path has the form:

$$P_{[t',s']\to[t-1,s'']\to[t,s]} = (e_{[t',s'],[t-1,s'']}, e_{[t-1,s''],[t,s]})$$

Therefore, the choice of s'' produces different paths.

- **Iteration 3:** $k = 3$. In this case, we have:

$$\mathbf{R}_L(3) = \mathbf{A}(L) \cdot \mathbf{A}(L-1) \cdot \mathbf{A}(L-2)$$

The correlation $c_{i,j}^{3,L}$ is given by:

$$\begin{aligned} c_{i,j}^{3,L} &= \sum_{l=1}^{n} \sum_{m=1}^{n} \alpha_{i,l}^{L} \cdot \alpha_{l,m}^{L-1} \cdot \alpha_{m,j}^{L-2} \\ &= \sum_{l=1}^{n} \alpha_{i,l}^{L} \cdot \left(\sum_{m=1}^{n} \alpha_{l,m}^{L-1} \cdot \alpha_{m,j}^{L-2} \right) \\ &= \sum_{l=1}^{n} \alpha_{i,l}^{L} \cdot c_{l,j}^{2,L-1} \end{aligned}$$

We can extend the previous reasoning, by rewriting the correlation as:

$$\begin{aligned} c_{[t,s],[t',s']}^{3,L} &= \sum_{s''=1}^{S} \sum_{t''=1}^{T} \alpha_{[t,s],[t'',s'']}^{L} \cdot c_{[t'',s''],[t',s']}^{2,L-1} \\ &= \sum_{s'' \in \mathcal{N}_s} \alpha_{[t,s],[t-1,s'']}^{L} \cdot c_{[t-1,s''],[t',s']}^{2,L-1} \end{aligned}$$

The choice of s'' determines the penultimate point traversed by the path, and from then on the path depends on the choice of a point $[t^*, s^*]$, and so on. Therefore, the sum takes all the paths of length 3 from node $[t, s]$ to node $[t', s']$, where each path has the form:

$$P_{[t',s']\to[t-1,s'']\to[t^*,s^*]\to[t,s]} = (e_{[t',s'],[t-1,s'']}, e_{[t-1,s''],[t^*,s^*]}, e_{[t^*,s^*],[t,s]})$$

- **Iteration k:** $k > 3$. At this point, it is trivial to show that:

$$\begin{aligned} c_{[t,s],[t',s']}^{k,L} &= \sum_{s'' \in \mathcal{N}_s} \alpha_{[t,s],[t-1,s'']}^{L} \cdot c_{[t-1,s''],[t',s']}^{k-1,L-1} \\ &= \sum_{s_1 \in \mathcal{N}_s} \alpha_{[t,s],[t-1,s_1]}^{L} \cdot \left(\sum_{s_2 \in \mathcal{N}_{s_1}} \alpha_{[t-1,s_1],[t-2,s_2]}^{L-1} \cdot (\cdots) \right) \\ &= \sum_{s_1 \in \mathcal{N}_s} \cdots \sum_{s_{k-1} \in \mathcal{N}_{s_{k-2}}} \alpha_{[t,s],[t-1,s_1]}^{L} \cdots \alpha_{[t-k+1,s_{k-1}],[t',s']}^{L-k+1} \end{aligned}$$

[1] In the following, we omit the dependence between $[t, s]$ and $[t', s']$, i.e. between the last and first points of the path, for the sake of simplicity.

Each product of attention weights corresponds to a path of length k from node $[t, s]$ to node $[t', s']$, where the choice of the intermediate points $[t - 1, s_1], \ldots, [t - k + 1, s_{k-1}]$ determines the specific path p^k. Therefore:

$$c^{k,L}_{[t,s],[t',s']} = \sum_{p^k \in \Pi^k_{[t,s],[t',s']}} \prod_{l=1}^{k} \alpha^{L-k+l}_{p^k[l]}$$

Bibliography

[1] ENSO Blog. *What are teleconnections? Connecting Earth's climate patterns via global information superhighways.* Accessed: 2025-04-03. 2022. URL: https://www.climate.gov/news-features/blogs/enso/what-are-teleconnections-connecting-earths-climate-patterns-global (cit. on p. 1).

[2] United Nations. *Goal 13: Take urgent action to combat climate change and its impacts.* Accessed: 2025-05-30. 2015. URL: https://www.un.org/sustainabledevelopment/climate-change/ (cit. on p. 2).

[3] Christian Pagé, Abel Aoun, and Alessandro Spinuso. *icclim: Calculating Climate Indices and Indicators Made Easy.* Accessed: 2025-05-30. 2025. URL: https://icclim.readthedocs.io/en/latest/explanation/climate_indices.html (cit. on p. 2).

[4] Jan Dutton. *What is a Climate Index: 6 Great Things You Need to Know.* Accessed: 2025-05-30. 2021. URL: https://www.worldclimateservice.com/2021/09/13/climate-index/ (cit. on p. 2).

[5] National Centers for Environmental Information. *North Atlantic Oscillation (NAO).* Accessed: 2025-05-30. 2025. URL: https://www.ncei.noaa.gov/access/monitoring/nao/ (cit. on p. 3).

[6] Michelle L'Heureux. *What is the El Niño-Southern Oscillation (ENSO) in a nutshell?* Accessed: 2025-05-30. 2014. URL: https://www.climate.gov/news-features/blogs/enso/what-el-ni%C3%B1o%E2%80%93southern-oscillation-enso-nutshell (cit. on p. 3).

[7] Jan Dutton. *What is the Madden-Julian Oscillation?* Accessed: 2025-05-30. Sept. 2021. URL: https://www.worldclimateservice.com/2021/09/20/madden-julian-oscillation/ (cit. on p. 4).

[8] Timothy M. DelSole and Michael K. Tippett. *Statistical Methods for Climate Scientists.* Cambridge, UK: Cambridge University Press, 2022. DOI: 10.1017/9781108659055. URL: https://doi.org/10.1017/9781108659055 (cit. on p. 7).

[9] Douglas C. Montgomery, Elizabeth A. Peck, and G. Geoffrey Vining. *Introduction to Linear Regression Analysis*. 6th ed. Hoboken, NJ: John Wiley & Sons, 2021. ISBN: 978-1-119-57875-8 (cit. on p. 7).

[10] Karen A McKinnon and Clara Deser. «Could detection and attribution of climate change trends be spurious regression?» In: *Climate Dynamics* 60.5-6 (2022), pp. 1291–1312 (cit. on p. 8).

[11] Daniel S Wilks. *Statistical methods in the atmospheric sciences*. Vol. 100. Academic Press, 2011 (cit. on p. 8).

[12] Chao-Lung Huang, Wen-Liang Huang, Hsueh-Yi Chang, Yang-Chieh Yeh, and Chuen-Tsai Tsai. «A novel framework for spatio-temporal prediction of environmental data using deep learning». In: *Scientific Reports* 10.1 (2020), p. 22243 (cit. on p. 8).

[13] Liangliang Li, Jihua Zhu, Hao Zhang, Hanlin Tan, Bangdi Du, and Qingling Ran. «A Graph Neural Network with Spatio-Temporal Attention for Multi-Sources Time Series Data: An Application to Frost Forecast». In: *Sensors* 22.4 (2022), p. 1486 (cit. on p. 8).

[14] Hanye Yu, Yongze Kang, Xun Shi, Jianxin Wang, Qingqing Zhang, and Fang Chen. «Spatial-temporal graph neural networks for groundwater data». In: *Scientific Reports* 14.1 (2024), p. 23956 (cit. on p. 8).

[15] Fatoumata Dama and Christine Sinoquet. «Time Series Analysis and Modeling to Forecast: a Survey». In: *arXiv preprint arXiv:2104.00164* (2021). URL: https://arxiv.org/abs/2104.00164 (cit. on pp. 8, 110).

[16] Robert B. Cleveland, William S. Cleveland, Jean E. McRae, and Irma Terpenning. «STL: A Seasonal-Trend Decomposition Procedure Based on Loess». In: *Journal of Official Statistics* 6.1 (1990), pp. 3–33. URL: https://www.proquest.com/scholarly-journals/stl-seasonal-trend-decomposition-procedure-based/docview/1266805989/se-2 (cit. on pp. 8, 9, 111).

[17] Rob J. Hyndman, Anne B. Koehler, Ralph D. Snyder, and Simone Grose. «A state space framework for automatic forecasting using exponential smoothing methods». In: *International Journal of Forecasting* 18.3 (2002), pp. 439–454. DOI: 10.1016/S0169-2070(01)00110-8. URL: https://www.sciencedirect.com/science/article/pii/S0169207001001108 (cit. on pp. 8, 112).

[18] James Douglas Hamilton. *Time series analysis*. Vol. 2. Princeton, NJ: Princeton University Press, 1994 (cit. on p. 8).

[19] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. 2nd. New York, NY: Springer Science & Business Media, 2009 (cit. on p. 9).

[20] S. E. Braun, K. D. Willis, S. N. Mladen, F. Aslanzadeh, A. Lanoye, J. Langbein, M. Reid, and A. R. Loughan. «The nuts and bolts of hypothesis testing». In: *Neuro-Oncology Practice* 9.6 (2022). Published online 2022 Jun 8; eCollection 2022 Dec, pp. 509–519. ISSN: 2052-2948. DOI: 10.1093/nop/npv052. URL: https://doi.org/10.1093/nop/npv052 (cit. on p. 9).

[21] Riko Kelter. «Bayesian and frequentist testing for differences between two groups with parametric and nonparametric two-sample tests». In: *WIREs Computational Statistics* 13.6 (2021). Open Access, e1523. DOI: 10.1002/wics.1523. URL: https://doi.org/10.1002/wics.1523 (cit. on p. 9).

[22] W. Gosset. «The Probable Error of a Mean». In: *Biometrika* 6.1 (1908). Originally published under the pseudonym "Student", pp. 1–25. DOI: 10.1093/biomet/6.1.1 (cit. on pp. 9, 112).

[23] R.A. Fisher. *Statistical Methods for Research Workers.* First edition; ISBN: 978-0-02-844740-7. Edinburgh: Oliver & Boyd, 1925 (cit. on pp. 9, 113).

[24] Karl Pearson. «Note on regression and inheritance in the case of two parents». In: *Proceedings of the Royal Society of London* 58 (1895), pp. 240–242. DOI: 10.1098/rspl.1895.0041. URL: https://royalsocietypublishing.org/doi/abs/10.1098/rspl.1895.0041 (cit. on pp. 9, 74, 114).

[25] Maurice G. Kendall. *Rank Correlation Methods.* 4th. London: Charles Griffin, 1975. ISBN: 978-0852641996 (cit. on pp. 9, 114).

[26] Pranab Kumar Sen. «Estimates of the Regression Coefficient Based on Kendall's Tau». In: *Journal of the American Statistical Association* 63.324 (1968), pp. 1379–1389. DOI: 10.1080/01621459.1968.10480934. URL: https://doi.org/10.1080/01621459.1968.10480934 (cit. on pp. 9, 115).

[27] A. N. Pettitt. «A Non-Parametric Approach to the Change-Point Problem». In: *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 28.2 (1979), pp. 126–135. DOI: 10.2307/2346729. URL: https://doi.org/10.2307/2346729 (cit. on pp. 9, 115).

[28] Charles Spearman. «The proof and measurement of association between two things». In: *The American Journal of Psychology* 100.3-4 (1987), pp. 441–471. DOI: 10.2307/1422689 (cit. on pp. 9, 116).

[29] Henning W. Rust, Andy Richling, Peter Bissolli, and Uwe Ulbrich. «Linking teleconnection patterns to European temperature - a multiple linear regression model». In: *Meteorologische Zeitschrift* 24.4 (2015), pp. 411–423. ISSN: 0941-2948. DOI: 10.1127/metz/2015/0642. URL: https://www.schweizerbart.de/papers/metz/detail/24/84724/Linking_teleconnection_patterns_to_European_temper?af=crossref (cit. on p. 11).

[30] M. A. Rahman, M. A. Bari, and M. A. Rahman. «Use of Teleconnections to Predict Western Australian Seasonal Rainfall Using ARIMAX Model». In: *Hydrology* 7.3 (2020), p. 52. DOI: 10.3390/hydrology7030052. URL: https://www.mdpi.com/2306-5338/7/3/52 (cit. on p. 11).

[31] Nura Idris Abdullahi, Shamsudden Mohammed, Yusuf Owoseni, Stephen Ijimdiya, and Khalid Suleiman. «A Non-Parametric Mann-Kendall and Sen's Slope Estimate as a Method for Detecting Trend Within Hydro-Meteorological Time Series: A Review». In: *Academy Journal of Science and Engineering* 13.2 (2023), pp. 1–12. ISSN: 2734-3898. URL: https://ajse.academyjsekad.edu.ng/index.php/new-ajse/article/view/272 (cit. on p. 11).

[32] Karen L. Smith. *Introduction to Principal Component Analysis*. Accessed: 2025-05-19. 2023. URL: https://kls2177.github.io/Climate-and-Geophysical-Data-Analysis/chapters/Week7/Intro_to_PCA.html (cit. on pp. 11, 119).

[33] B. Huang, W. Zhang, L. Zhang, J. Li, L. Wang, J. Li, and J. He. «How does the SST variability over the western North Atlantic Ocean control Arctic warming over the Barents–Kara Seas?» In: *Geophysical Research Letters* 46.24 (2019), pp. 13951–13960. DOI: 10.1029/2019GL085653. URL: https://doi.org/10.1029/2019GL085653 (cit. on p. 11).

[34] David S Broomhead and Gregory P King. «Extracting qualitative dynamics from experimental data». In: *Physica D: Nonlinear Phenomena* 20.2-3 (1986), pp. 217–236 (cit. on pp. 12, 120).

[35] A. Groth and M. Ghil. «Monte Carlo Singular Spectrum Analysis (SSA) Revisited: Detecting Oscillator Clusters in Multivariate Datasets». In: *Journal of Climate* 28.19 (2015), pp. 7873–7893. DOI: 10.1175/JCLI-D-15-0100.1 (cit. on p. 12).

[36] Gaston Manta, Eviatar Bach, Stefanie Talento, Marcelo Barreiro, Sabrina Speich, Michael Ghil, et al. «The South Atlantic Dipole via multichannel singular spectrum analysis». In: *Scientific Reports* 14 (2024), p. 15534. DOI: 10.1038/s41598-024-62089-w. URL: https://doi.org/10.1038/s41598-024-62089-w (cit. on p. 12).

[37] Jonathan H Tu, Clarence W Rowley, Dirk M Luchtenburg, Steven L Brunton, and J Nathan Kutz. «On dynamic mode decomposition: Theory and applications». In: *Journal of Computational Dynamics* 1.2 (2014), pp. 391–421 (cit. on pp. 13, 121).

[38] J Nathan Kutz, Steven L Brunton, Bingni W Brunton, and Joshua L Proctor. *Dynamic Mode Decomposition: Data-Driven Modeling of Complex Systems*. Philadelphia: SIAM, 2016. ISBN: 978-1-611974-49-2 (cit. on p. 13).

[39] John Ferré, Ariel Rokem, Elizabeth A. Buffalo, J. Nathan Kutz, and Adrienne Fairhall. «Non-Stationary Dynamic Mode Decomposition». In: *IEEE Access* 11 (Oct. 2023). Epub 2023 Oct 20; PMCID: PMC10705813, pp. 117159–117176. DOI: `10.1109/ACCESS.2023.3326412` (cit. on p. 13).

[40] Fredolin T. Tangang, William W. Hsieh, and Benyang Tang. «Forecasting ENSO Events: A Neural Network–Extended EOF Approach». In: *Journal of Climate* 11.1 (1998), pp. 29–41. DOI: `10.1175/1520-0442(1998)011<0029:FEEANN>2.0.CO;2` (cit. on p. 14).

[41] Alexander Gershunov and Tim P. Barnett. «Artificial Neural Networks and Long-Range Precipitation Prediction in California». In: *Journal of Applied Meteorology* 39.1 (2000), pp. 57–66. DOI: `10.1175/1520-0450(2000)039<0057:ANNALR>2.0.CO;2` (cit. on p. 15).

[42] William W. Hsieh, Aiming Wu, and Amir Shabbar. «Nonlinear atmospheric teleconnections». In: *Geophysical Research Letters* 33.3 (2006), p. L03711. DOI: `10.1029/2005GL025471` (cit. on p. 15).

[43] Guy Spellman. «Influence of Circulation Patterns on Temperature Behavior at the Regional Scale: A Case Study Investigated via Neural Network Modeling». In: *Journal of Climate* 26.11 (2013), pp. 3784–3792. DOI: `10.1175/JCLI-D-11-00551.1` (cit. on p. 15).

[44] Nicolás Hoyos, Alexander Correa-Metrio, and Julián Escobar. «Development of Technology for Identification of Climate Patterns during Floods Using Global Climate Model Data with Convolutional Neural Networks». In: *Water* 14.24 (2022), p. 4045. DOI: `10.3390/w14244045` (cit. on p. 16).

[45] Keiron O'Shea and Ryan Nash. «An Introduction to Convolutional Neural Networks». In: *arXiv preprint arXiv:1511.08458* (2015). URL: `https://arxiv.org/abs/1511.08458` (cit. on p. 16).

[46] Georgia Papacharalampous, Hristos Tyralis, and Andreas Langousis. «Modeling spatial asymmetries in teleconnected extreme temperatures». In: *arXiv preprint arXiv:2310.03220* (2024). arXiv: `2310.03220 [physics.ao-ph]` (cit. on p. 16).

[47] Yoo-Geun Ham, Jeong-Hwan Kim, and Jing-Jia Luo. «Graph Neural Networks for Improved El Niño Forecasting». In: *arXiv preprint arXiv:2012.01598* (2020). arXiv: `2012.01598 [physics.ao-ph]` (cit. on p. 17).

[48] Jaya Kawale, Snigdhansu Chatterjee, Dominick Ormsby, Karsten Steinhaeuser, Stefan Liess, and Vipin Kumar. «Testing the significance of spatio-temporal teleconnection patterns». In: *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* KDD '12. Beijing, China: Association for Computing Machinery, 2012, pp. 642–

650. ISBN: 9781450314626. DOI: `10.1145/2339530.2339634`. URL: `https://doi.org/10.1145/2339530.2339634` (cit. on p. 18).

[49] C. W. J. Granger. «Investigating causal relations by econometric models and cross-spectral methods». In: *Econometrica* 37.3 (July 1969), pp. 424–438. DOI: `10.2307/1912791` (cit. on p. 18).

[50] Amina Adadi and Mohammed Berrada. «Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI)». In: *ACM Computing Surveys* 51.5 (2018), pp. 1–42. DOI: `10.1145/3236009`. URL: `https://doi.org/10.1145/3236009` (cit. on p. 19).

[51] Benjamin A. Toms, Elizabeth A. Barnes, and Imme Ebert-Uphoff. «Finding the Right XAI Method—A Guide for the Evaluation and Ranking of Explainable AI Methods in Climate Science». In: *Artificial Intelligence for the Earth Systems* 3.3 (2024), e230074. DOI: `10.1175/AIES-D-23-0074.1` (cit. on p. 19).

[52] Antonios Mamalakis, Imme Ebert-Uphoff, and Elizabeth A. Barnes. «Using Explainable Artificial Intelligence to Quantify "Climate Distinguishability" After Stratospheric Aerosol Injection». In: *Geophysical Research Letters* 50.24 (2023), e2023GL106137. DOI: `10.1029/2023GL106137` (cit. on p. 19).

[53] Stefano Materia et al. «Artificial intelligence for climate prediction of extremes: State of the art, challenges, and future perspectives». In: *WIREs Climate Change* 15.4 (2024), e914. DOI: `10.1002/wcc.914` (cit. on p. 19).

[54] James H. Faghmous and Vipin Kumar. «Spatio-temporal Data Mining for Climate Data: Advances, Challenges, and Opportunities». In: *Data Mining and Knowledge Discovery for Big Data*. Ed. by Xiaohua Hu, Jian Pei, Wei Wang, and Philip S. Yu. Accessed: 2025-04-05. Springer, 2013, pp. 83–116. DOI: `10.1007/978-3-642-40837-3_3`. URL: `https://doi.org/10.1007/978-3-642-40837-3_3` (cit. on p. 21).

[55] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. «Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks». In: *International Conference on Machine Learning (ICML)*. Vol. 97. PMLR. 2019, pp. 3744–3753 (cit. on p. 32).

[56] Bharti Khemani, Shruti Patil, Ketan Kotecha, and Sudeep Tanwar. «A review of graph neural networks: concepts, architectures, techniques, challenges, datasets, applications, and future directions». In: *Journal of Big Data* 11.1 (2024), p. 18. DOI: `10.1186/s40537-023-00876-4`. URL: `https://journalofbigdata.springeropen.com/articles/10.1186/s40537-023-00876-4` (cit. on p. 42).

[57] Thomas N Kipf and Max Welling. «Semi-Supervised Classification with Graph Convolutional Networks». In: *arXiv preprint arXiv:1609.02907* (2017) (cit. on p. 42).

[58] William L. Hamilton, Rex Ying, and Jure Leskovec. *Inductive Representation Learning on Large Graphs*. 2017. arXiv: `1706.02216 [cs.SI]`. URL: `https://arxiv.org/abs/1706.02216` (cit. on p. 42).

[59] Mikael Henaff, Joan Bruna, and Yann LeCun. «Deep Convolutional Networks on Graph-Structured Data». In: *arXiv preprint arXiv:1506.05163* (2015). URL: `https://arxiv.org/abs/1506.05163` (cit. on p. 43).

[60] Petar Veličković. *TikZ: Complete collection of my PGF/TikZ figures*. Accessed: 2025-05-31. 2018. URL: `https://github.com/PetarV-/TikZ` (cit. on p. 44).

[61] Matthias Fey and Jan Eric Lenssen. *Fast Graph Representation Learning with PyTorch Geometric*. 2019. arXiv: `1903.02428 [cs.LG]`. URL: `https://arxiv.org/abs/1903.02428` (cit. on pp. 43, 77).

[62] Qimai Li, Zhichao Han, and Xiao-Ming Wu. «Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning». In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018 (cit. on p. 45).

[63] Uri Alon and Eran Yahav. «On the Bottleneck of Graph Neural Networks and its Practical Implications». In: *International Conference on Learning Representations (ICLR)*. 2021. URL: `https://openreview.net/forum?id=i80OPhOCVH2` (cit. on p. 45).

[64] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. «Graph Attention Networks». In: *International Conference on Learning Representations (ICLR)*. 2018. URL: `https://arxiv.org/abs/1710.10903` (cit. on pp. 45, 46).

[65] Geoffrey E. Hinton and Ruslan R. Salakhutdinov. «Reducing the Dimensionality of Data with Neural Networks». In: *Science* 313.5786 (2006), pp. 504–507. DOI: `10.1126/science.1127647` (cit. on p. 49).

[66] Diederik P Kingma and Max Welling. «Auto-Encoding Variational Bayes». In: *arXiv preprint arXiv:1312.6114* (2013) (cit. on p. 49).

[67] Solomon Kullback and Richard A Leibler. «On information and sufficiency». In: *The Annals of Mathematical Statistics* 22.1 (1951), pp. 79–86. DOI: `10.1214/aoms/1177729694` (cit. on p. 50).

[68] Thomas N. Kipf and Max Welling. «Variational Graph Auto-Encoders». In: *arXiv preprint arXiv:1611.07308* (Nov. 2016). DOI: `10.48550/arXiv.1611.07308`. arXiv: `1611.07308 [stat.ML]` (cit. on p. 53).

[69] Kendrick Boyd, Kevin H. Eng, and C. David Jr. Page. «Area under the Precision-Recall Curve: Point Estimates and Confidence Intervals». In: *Machine Learning and Knowledge Discovery in Databases.* Ed. by Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip Železný. Vol. 8190. Lecture Notes in Computer Science. Springer, 2013, pp. 451–466. ISBN: 978-3-642-40993-6. DOI: `10.1007/978-3-642-40994-3_29` (cit. on p. 56).

[70] Yong-Min Shin, Siqing Li, Xin Cao, and Won-Yong Shin. «Faithful and Accurate Self-Attention Attribution for Message Passing Neural Networks via the Computation Tree Viewpoint». In: *arXiv preprint arXiv:2406.04612* (2024). URL: `https://arxiv.org/abs/2406.04612` (cit. on pp. 60, 62).

[71] A. A. Markov. «Extension of the Law of Large Numbers to Dependent Events». Russian. In: *Izvestiya Fiziko-Matematicheskogo Obshchestva pri Kazanskom Universitete, Seriya 2* 15 (1906), pp. 135–156 (cit. on p. 62).

[72] Math Stack Exchange user. *Proving or disproving product of two stochastic matrices is stochastic.* Accessed: 2025-05-22. 2018. URL: `https://math.stackexchange.com/questions/2773442` (cit. on p. 62).

[73] Claude E. Shannon. «A Mathematical Theory of Communication». In: *The Bell System Technical Journal* 27.3 (July 1948), pp. 379–423. URL: `https://dl.acm.org/doi/pdf/10.1145/584091.584093` (cit. on p. 66).

[74] P. A. M. Dirac. «The Physical Interpretation of the Quantum Dynamics». In: *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character* 113.765 (1927), pp. 621–641. DOI: `10.1098/rspa.1927.0012` (cit. on p. 67).

[75] Roger A. Horn and Charles R. Johnson. *Topics in Matrix Analysis.* Revised. Cambridge, UK: Cambridge University Press, 1991, pp. viii+607. ISBN: 0-521-30587-X. DOI: `10.1017/CBO9780511840371` (cit. on p. 72).

[76] Jacob Cohen, Patricia Cohen, Stephen G. West, and Leona S. Aiken. *Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences.* 3rd. Mahwah, NJ: Lawrence Erlbaum Associates, 2003. ISBN: 978-0805822236 (cit. on p. 74).

[77] Gonzalo Travieso, Alexandre Benatti, and Luciano da F. Costa. «An Analytical Approach to the Jaccard Similarity Index». In: *arXiv preprint arXiv:2410.16436* (2024). v1 posted 21 Oct 2024. DOI: `10.48550/ARXIV.2410.16436`. URL: `https://arxiv.org/abs/2410.16436` (cit. on p. 75).

[78] Python Software Foundation. *Python 3.11.10 Release.* Accessed: 2025-06-03. Sept. 2024. URL: `https://www.python.org/downloads/release/python-31110/` (cit. on p. 77).

[79] S. Hoyer and J. Hamman. «xarray: N-D labeled arrays and datasets in Python». In: *Journal of Open Research Software* 5.1 (2017). DOI: `10.5334/jors.148`. URL: `https://doi.org/10.5334/jors.148` (cit. on p. 77).

[80] Unidata Program Center. *The NetCDF C Libraries and Utilities*. Version 4.9.2. 2025. URL: `https://www.unidata.ucar.edu/software/netcdf/` (cit. on pp. 77, 78).

[81] Adam Paszke et al. «PyTorch: An Imperative Style, High-Performance Deep Learning Library». In: *Advances in Neural Information Processing Systems* 32 (2019). URL: `https://arxiv.org/abs/1912.01703` (cit. on p. 77).

[82] William Falcon and The PyTorch Lightning team. *PyTorch Lightning*. Version 2.5.1.post0. 2024. URL: `https://lightning.ai/docs/pytorch/stable/` (cit. on p. 77).

[83] Weights & Biases. *Weights & Biases Quickstart Guide*. Accessed: 2025-06-03. 2025. URL: `https://wandb.ai/quickstart/` (cit. on p. 78).

[84] Plotly Technologies Inc. *Collaborative data science*. Accessed: 2025-07-03. 2015. URL: `https://plot.ly` (cit. on p. 78).

[85] J. D. Hunter. «Matplotlib: A 2D Graphics Environment». In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: `10.1109/MCSE.2007.55`. URL: `https://doi.org/10.1109/MCSE.2007.55` (cit. on p. 78).

[86] European Centre for Medium-Range Weather Forecasts. *ERA5: data documentation*. Accessed: 2025-05-02. 2025. URL: `https://confluence.ecmwf.int/display/CKB/ERA5%5C%3A+data+documentation` (cit. on pp. 78, 79).

[87] Adrian Simmons et al. *Global stratospheric temperature bias and other stratospheric aspects of ERA5 and ERA5.1*. Technical Memorandum 859. European Centre for Medium-Range Weather Forecasts (ECMWF), Jan. 2020. DOI: `10.21957/rcxqfmg0`. URL: `https://www.ecmwf.int/en/elibrary/81149-global-stratospheric-temperature-bias-and-other-stratospheric-aspects-era5-and` (cit. on p. 78).

[88] Copernicus Climate Change Service (C3S) Climate Data Store (CDS). *ERA5 hourly data on single levels from 1940 to present*. Accessed: 2025-05-01. 2025. URL: `https://cds.climate.copernicus.eu/datasets/reanalysis-era5-single-levels?tab=overview` (cit. on p. 78).

[89] Geir Evensen, Femke C. Vossepoel, and Peter Jan van Leeuwen. *Data Assimilation Fundamentals: A Unified Formulation of the State and Parameter Estimation Problem*. Springer Textbooks in Earth Sciences, Geography and Environment. Open Access. Springer, 2022. DOI: `10.1007/978-3-030-96709-3`. URL: `https://link.springer.com/book/10.1007/978-3-030-96709-3` (cit. on p. 78).

[90] P. Courtier, J.-N. Thépaut, and A. Hollingsworth. «A Strategy for Operational Implementation of 4D-Var Using an Incremental Approach». In: *Quarterly Journal of the Royal Meteorological Society* 120.519 (1994), pp. 1367–1387. DOI: `10.1002/qj.49712051912`. URL: `https://doi.org/10.1002/qj.49712051912` (cit. on p. 78).

[91] Anabelle Menochet and Michela Giusti. *What are GRIB files and how can I read them.* Copernicus Knowledge Base, ECMWF Confluence Wiki. Last modified on November 22, 2023. Nov. 2023. URL: `https://confluence.ecmwf.int/display/CKB/What+are+GRIB+files+and+how+can+I+read+them` (cit. on p. 78).

[92] Shabnam Kumari and P. Muthulakshmi. «SARIMA Model: An Efficient Machine Learning Technique for Weather Forecasting». In: *Procedia Computer Science* 235 (2024), pp. 656–670. DOI: `10.1016/j.procs.2024.04.064`. URL: `https://doi.org/10.1016/j.procs.2024.04.064` (cit. on p. 111).

[93] Robert M. Hirsch and James R. Slack. «A Nonparametric Trend Test for Seasonal Data with Serial Dependence». In: *Water Resources Research* 20.6 (1984), pp. 727–732. DOI: `10.1029/WR020i006p00727`. URL: `https://doi.org/10.1029/WR020i006p00727` (cit. on p. 114).

[94] Harold Jeffreys. *The Theory of Probability.* Third. Oxford University Press, 1961 (cit. on p. 116).

[95] Jeffrey N Rouder, Paul L Speckman, Dongchu Sun, and Richard D Morey. «Bayesian t tests for accepting and rejecting the null hypothesis». In: *Psychonomic Bulletin & Review* 16.2 (2009), pp. 225–237. DOI: `10.3758/PBR.16.2.225` (cit. on p. 117).

[96] Bruna Wundervald. *Bayesian Linear Regression.* Tech. rep. National University of Ireland, Maynooth, 2019. URL: `https://www.researchgate.net/publication/333917874_Bayesian_Linear_Regression` (cit. on p. 117).

[97] Aaron Plavnick. *The Fundamental Theorem of Markov Chains.* Accessed: 2025-05-02. 2008. URL: `https://www.math.uchicago.edu/~may/VIGRE/VIGRE2008/REUPapers/Plavnick.pdf` (cit. on p. 118).

[98] David B. Stephenson and Rasmus E. Benestad. «Empirical Orthogonal Function (EOF) Analysis». In: *Quantitative Methods of Data Analysis for the Physical Sciences and Engineering.* Accessed: 2025-05-19. Cambridge University Press, 2000. URL: `https://www.cambridge.org/core/books/quantitative-methods-of-data-analysis-for-the-physical-sciences-and-engineering/empirical-orthogonal-function-eof-analysis/19813B04BEC5F5BC670CD6119BF968E4` (cit. on p. 119).

[99] J. R. Partington. *An Introduction to Hankel Operators.* Vol. 13. London Mathematical Society Student Texts. Cambridge University Press, 1988. ISBN: 0-521-36791-3 (cit. on p. 120).

[100] Peter J Schmid. «Dynamic mode decomposition of numerical and experimental data». In: *Journal of Fluid Mechanics* 656 (2010), pp. 5–28. DOI: 10.1017/S0022112010001217 (cit. on p. 121).

[101] Britannica. *Artificial intelligence.* Accessed: 2025-04-04. 2025. URL: https://www.britannica.com/technology/artificial-intelligence (cit. on p. 123).

[102] NVIDIA. *SuperVize Me: What's the Difference Between Supervised, Unsupervised, Semi-Supervised and Reinforcement Learning?* Accessed: 2025-04-04. 2018. URL: https://blogs.nvidia.com/blog/supervised-unsupervised-learning/ (cit. on p. 123).

[103] Medium. *Shallow Learning vs Deep Learning: Is Bigger Always Better?* Accessed: 2025-04-04. 2024. URL: https://medium.com/%40hassaanidrees7/shallow-learning-vs-deep-learning-is-bigger-always-better-51c0bd21f059 (cit. on p. 124).

[104] Ben Kröse, B. Krose, Patrick van der Smagt, and Patrick Smagt. «An introduction to neural networks». In: *J Comput Sci* 48 (Jan. 1993). Accessed: 2025-04-04 (cit. on p. 124).

[105] Frank Rosenblatt. «The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain». In: *Psychological Review* 65.6 (1958). Accessed: 2025-04-04, pp. 386–408. DOI: 10.1037/h0042519. URL: http://dx.doi.org/10.1037/h0042519 (cit. on p. 126).

[106] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. «Multilayer Feedforward Networks are Universal Approximators». In: *Neural Networks* 2.5 (1989), pp. 359–366. DOI: 10.1016/0893-6080(89)90020-8. URL: https://www.sciencedirect.com/science/article/pii/0893608089900208 (cit. on p. 126).

[107] Fionn Murtagh. «Multilayer perceptrons for classification and regression». In: *Neurocomputing* 2.5-6 (July 1991). Accessed: 2025-04-04, pp. 183–197. DOI: 10.1016/0925-2312(91)90023-5. URL: https://doi.org/10.1016/0925-2312(91)90023-5 (cit. on p. 129).

[108] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. «Learning representations by back-propagating errors». In: *Nature* 323.6088 (1986), pp. 533–536. DOI: 10.1038/323533a0. URL: https://www.nature.com/articles/323533a0 (cit. on p. 130).

[109] Robert Mansel Gower, Nicolas Loizou, Xun Qian, Alibek Sailanbayev, Egor Shulgin, and Peter Richtárik. «SGD: General Analysis and Improved Rates». In: *Proceedings of the 36th International Conference on Machine Learning.* Vol. 97. Proceedings of Machine Learning Research. arXiv:1901.09401. PMLR, 2019, pp. 5200–5209. DOI: `10.48550/arXiv.1901.09401`. URL: `https://arxiv.org/abs/1901.09401` (cit. on p. 132).

[110] Rong Ge, Sham M. Kakade, Rahul Kidambi, and Praneeth Netrapalli. «The Step Decay Schedule: A Near Optimal, Geometrically Decaying Learning Rate Procedure For Least Squares». In: *arXiv preprint arXiv:1904.12838* (2019). Accessed: 2025-04-05. URL: `https://doi.org/10.48550/arXiv.1904.12838` (cit. on p. 132).

[111] Zhiyuan Li and Sanjeev Arora. «An Exponential Learning Rate Schedule for Deep Learning». In: *arXiv preprint arXiv:1910.07454* (2019). Accessed: 2025-04-05. URL: `https://doi.org/10.48550/arXiv.1910.07454` (cit. on p. 133).

[112] Ilya Loshchilov and Frank Hutter. «SGDR: Stochastic Gradient Descent with Warm Restarts». In: *arXiv preprint arXiv:1608.03983* (2016). Accessed: 2025-04-05. URL: `https://doi.org/10.48550/arXiv.1608.03983` (cit. on p. 133).

[113] Diederik P. Kingma and Jimmy Ba. «Adam: A Method for Stochastic Optimization». In: *arXiv preprint arXiv:1412.6980* (2014). Accessed: 2025-04-05. URL: `https://doi.org/10.48550/arXiv.1412.6980` (cit. on p. 133).

[114] ScienceDirect. *Silica - an overview.* Accessed: 2025-05-02. 2025. URL: `https://www.sciencedirect.com/topics/chemical-engineering/silica` (cit. on p. 135).

[115] U.S. Geological Survey. *Paleoclimate Proxies.* Accessed: 2025-04-29. 2025. URL: `https://www.usgs.gov/programs/ecosystems-land-change-science-program/science/paleoclimate-proxies` (cit. on p. 136).

[116] NOAA National Centers for Environmental Information. *Paleoclimatology Data.* Accessed: 2025-04-29. 2025. URL: `https://www.ncei.noaa.gov/products/paleoclimatology` (cit. on p. 136).

[117] PAGES 2k Consortium. «A global multiproxy database for temperature reconstructions of the Common Era». In: *Scientific Data* 4 (2017), p. 170088. DOI: `10.1038/sdata.2017.88`. URL: `https://www.nature.com/articles/sdata201788` (cit. on p. 136).

[118] John W. Williams et al. «The Neotoma Paleoecology Database, a multiproxy, international, community-curated data resource». In: *Quaternary Research* 89.1 (2018), pp. 156–177. DOI: `10.1017/qua.2017.105`. URL: `https://www.cambridge.org/core/journals/quaternary-research/article/neotoma-paleoecology-database-a-multiproxy-international-communitycurated-data-resource/1E1C9EB07ADFF01182BCB69A08E1C755` (cit. on p. 136).

[119] Copernicus Marine Service. *In Situ Platforms.* Accessed: 2025-04-29. 2025. URL: `https://marine.copernicus.eu/explainers/operational-oceanography/monitoring-forecasting/in-situ/platforms` (cit. on p. 137).

[120] National Snow and Ice Data Center. *What is the Cryosphere?* Accessed: 2025-05-02. 2025. URL: `https://nsidc.org/learn/what-cryosphere` (cit. on p. 137).

[121] *In situ observations of meteorological and soil variables from the US Climate Reference Network near the surface from 2006 to present.* Accessed: 2025-04-29. 2023. DOI: `10.24381/cds.d1f4864d`. URL: `https://cds.climate.copernicus.eu/cdsapp#!/dataset/insitu-observations-near-surface-temperature-us-climate-reference-network` (cit. on p. 137).

[122] A. M. G. Klein Tank et al. «Daily dataset of 20th-century surface air temperature and precipitation series for the European Climate Assessment». In: *International Journal of Climatology* 22.12 (2002), pp. 1441–1453. DOI: `10.1002/joc.773`. URL: `https://doi.org/10.1002/joc.773` (cit. on p. 137).

[123] NASA. *Remote Sensing.* Accessed: 2025-04-30. 2025. URL: `https://www.nasa.gov/directorates/somd/space-communications-navigation-program/remote-sensing/` (cit. on p. 138).

[124] SATPALDA. *Types of Satellite Imagery.* Accessed: 2025-04-30. 2024. URL: `https://satpalda.com/types-of-satellite-imagery/` (cit. on p. 138).

[125] NOAA National Centers for Environmental Information (NCEI). *NOAA Climate Data Record (CDR) of AVHRR Surface Reflectance, Version 5.* Accessed: 2025-04-30. 2021. URL: `https://www.ncei.noaa.gov/access/metadata/landing-page/bin/iso?id=gov.noaa.ncdc:C01557` (cit. on p. 138).

[126] NASA LP DAAC. *MODIS/Terra Land Surface Temperature and Emissivity Daily L3 Global 1km SIN Grid V061 (MOD11A1).* Accessed: 2025-04-30. 2022. URL: `https://modis.gsfc.nasa.gov/data/dataprod/mod11.php` (cit. on p. 138).

[127] Eric Vermote and NOAA CDR Program. *NOAA Climate Data Record (CDR) of VIIRS Surface Reflectance, Version 1.* Accessed: 2025-04-30. 2022. URL: https://catalog.data.gov/dataset/noaa-climate-data-record-cdr-of-viirs-surface-reflectance-version-11 (cit. on p. 138).

[128] NASA Earth Science Data Systems. *Sentinel-1 C-band Synthetic Aperture Radar (C-SAR).* Accessed: 2025-04-30. 2025. URL: https://www.earthdata.nasa.gov/data/instruments/sentinel-1-c-sar (cit. on p. 138).

[129] NASA Earth Science Data Systems. *GEDI (Global Ecosystem Dynamics Investigation) LiDAR.* Accessed: 2025-04-30. 2025. URL: https://www.earthdata.nasa.gov/data/instruments/gedi-lidar (cit. on p. 138).

[130] NASA National Snow and Ice Data Center Distributed Active Archive Center (NSIDC DAAC). *AMSR-E/AMSR2 Unified L3 Daily 12.5 km Brightness Temperatures, Sea Ice Concentration, Motion & Snow Depth Polar Grids V001.* Accessed: 2025-04-30. 2025. URL: https://catalog.data.gov/dataset/amsr-e-amsr2-unified-l3-daily-12-5-km-brightness-temperatures-sea-ice-concentration-motion (cit. on p. 138).

[131] Austin Gorens. *4.4.2. Radiative-Convective Models.* Accessed: 2025-04-30. 2018. URL: https://www.global-climate-change.org.uk/4-4-2.php (cit. on p. 139).

[132] Kari De Pryck and Mike Hulme, eds. *A Critical Assessment of the Intergovernmental Panel on Climate Change.* Open Access under CC BY-NC-ND 4.0. Cambridge University Press, 2022. ISBN: 9781009082099. DOI: 10.1017/9781009082099 (cit. on p. 139).

[133] NOAA Climate.gov. *Climate Models.* Accessed: 2025-04-30. 2021. URL: https://www.climate.gov/maps-data/climate-data-primer/predicting-climate/climate-models (cit. on p. 139).

[134] NOAA Geophysical Fluid Dynamics Laboratory. *Climate Modeling - Geophysical Fluid Dynamics Laboratory.* Accessed: 2025-04-30. 2009. URL: https://www.gfdl.noaa.gov/climate-modeling (cit. on p. 139).

[135] Hannah Christensen and Laure Zanna. «Parametrization in Weather and Climate Models». In: *Oxford Research Encyclopedia of Climate Science* (2022). Accessed: 2025-04-30. DOI: 10.1093/acrefore/9780190228620.013.826. URL: https://oxfordre.com/climatescience/display/10.1093/acrefore/9780190228620.001.0001/acrefore-9780190228620-e-826 (cit. on p. 139).

[136] Copernicus Climate Change Service. *What is Statistical and Dynamical Downscaling?* Accessed: 2025-04-30. 2021. URL: https://climate.copernicus.eu/sites/default/files/2021-01/infosheet8.pdf (cit. on p. 139).

[137] World Climate Research Programme. *Coupled Model Intercomparison Project (CMIP)*. Accessed April 30, 2025. 2025. URL: `https://wcrp-cmip.org` (cit. on p. 140).

[138] NOAA National Centers for Environmental Information. *North American Multi-Model Ensemble (NMME)*. Accessed April 30, 2025. 2025. URL: `https://www.ncei.noaa.gov/products/weather-climate-models/north-american-multi-model` (cit. on p. 140).

[139] European Centre for Medium-Range Weather Forecasts. *European Centre for Medium-Range Weather Forecasts (ECMWF)*. Accessed: 2025-05-01. 2025. URL: `https://www.ecmwf.int` (cit. on p. 140).

[140] National Oceanic and Atmospheric Administration. *National Oceanic and Atmospheric Administration (NOAA)*. Accessed: 2025-05-01. 2025. URL: `https://www.noaa.gov` (cit. on p. 140).

[141] Japan Meteorological Agency. *Japan Meteorological Agency (JMA)*. Accessed: 2025-05-01. 2025. URL: `https://www.jma.go.jp/jma/en/menu.html` (cit. on p. 140).

[142] Eugenia Kalnay et al. «The NCEP/NCAR 40-Year Reanalysis Project». In: *Bulletin of the American Meteorological Society* 77.3 (1996), pp. 437–471. DOI: `10.1175/1520-0477(1996)077<0437:TNYRP>2.0.CO;2`. URL: `https://doi.org/10.1175/1520-0477(1996)077%3C0437:TNYRP%3E2.0.CO;2` (cit. on p. 141).

[143] Masao Kanamitsu, Wesley Ebisuzaki, John Woollen, S-K Yang, Joseph J. Hnilo, Michael Fiorino, and Gerald L. Potter. «NCEP-DOE AMIP-II Reanalysis (R-2)». In: *Bulletin of the American Meteorological Society* 83.11 (2002), pp. 1631–1643. DOI: `10.1175/BAMS-83-11-1631`. URL: `https://doi.org/10.1175/BAMS-83-11-1631` (cit. on p. 141).

[144] Daichun Wang, Wei You, Zengliang Zang, Xiaobin Pan, Yiwen Hu, and Yanfei Liang. «A three-dimensional variational data assimilation system for aerosol optical properties based on WRF-Chem v4.0: design, development, and application of assimilating Himawari-8 aerosol observations». In: *Geoscientific Model Development* 15 (2022), pp. 1821–1840. DOI: `10.5194/gmd-15-1821-2022`. URL: `https://doi.org/10.5194/gmd-15-1821-2022` (cit. on p. 142).

[145] V. Buchard et al. «The MERRA-2 Aerosol Reanalysis, 1980 Onward. Part I: System Description and Data Assimilation Evaluation». In: *Journal of Climate* 30.17 (2017), pp. 6823–6850. DOI: `10.1175/JCLI-D-16-0609.1` (cit. on p. 142).

[146] NASA Global Modeling and Assimilation Office (GMAO). *MERRA-2: Modern-Era Retrospective analysis for Research and Applications, Version 2*. Accessed: 2025-05-02. 2015. URL: `https://gmao.gsfc.nasa.gov/reanalysis/MERRA-2/` (cit. on p. 142).

[147] Kazutoshi Onogi et al. «The JRA-25 Reanalysis». In: *Journal of the Meteorological Society of Japan. Ser. II* 85.3 (2007), pp. 369–432. DOI: `10.2151/jmsj.85.369`. URL: `https://doi.org/10.2151/jmsj.85.369` (cit. on p. 142).

[148] Japan Meteorological Agency. *JRA-55 Handbook: TL319 Spectral Model and 60-Level σ-Coordinate System*. Accessed: 2025-05-02. 2025. URL: `https://jra.kishou.go.jp/JRA-55/document/JRA-55_handbook_TL319_en.pdf` (cit. on p. 142).

[149] Z. I. Janjić, R. Gall, and M. E. Pyle. *Scientific Documentation for the NMM Solver*. Tech. rep. NCAR/TN-477+STR. Accessed: 2025-05-02. National Center for Atmospheric Research, 2010. URL: `https://web.archive.org/web/20110823082059/http://nldr.library.ucar.edu/collections/technotes/asset-000-000-000-845.pdf` (cit. on p. 142).

[150] Shinya Kobayashi et al. «The JRA-55 Reanalysis: General Specifications and Basic Characteristics». In: *Journal of the Meteorological Society of Japan. Ser. II* 93.1 (2015), pp. 5–48. DOI: `10.2151/jmsj.2015-001`. URL: `https://doi.org/10.2151/jmsj.2015-001` (cit. on p. 143).

[151] Plato. *Timaeus*. Ed. and trans. by Peter Kalkavage. Indianapolis: Hackett Publishing, 2008. ISBN: 9780872204461 (cit. on p. 148).

[152] Leonhard Euler. «Elementa doctrinae solidorum». Latin. In: *Novi Commentarii Academiae Scientiarum Petropolitanae* 4 (1758). Presented in 1750, published in 1758, pp. 109–140 (cit. on p. 149).