Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Deep Learning applied to Anomaly Detection in Meteorological Time Series

Bachelor's Thesis

2023
Júlia Lichmanová

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Deep Learning applied to Anomaly Detection in Meteorological Time Series

Bachelor's Thesis

Study Programme:	Computer Science
Field of Study:	Computer Science
Department:	Department of Computer Science
Supervisor:	MSc. Philipp Roberto Miotti

Bratislava, 2023
Júlia Lichmanová

70681272

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Júlia Lichmanová
Študijný program:	informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor:	informatika
Typ záverečnej práce:	bakalárska
Jazyk záverečnej práce:	anglický
Sekundárny jazyk:	slovenský

Názov: Deep Learning Applied to Anomaly Detection in Meteorological Time Series
Hlboké učenie aplikované na detekciu anomálií v meteorologických časových radoch

Anotácia: Cieľom bakalárskej práce je demonštrovať realizovateľnosť a praktické použitie metód hlbokého učenia na detekciu abnormálneho senzorického správania v prístrojoch merajúcich meteorologické premenné reprezentované ako viacrozmerný časový rad. Rýchla a presná analýza údajov v solárnom energetickom priemysle je veľmi žiadaná, keďže v súčasnosti trh so solárnou energiou rýchlo rastie.Využitie natrénovaného modelu by umožnilo označovať anomálie vo veľkom množstve údajov časových radov efektívnejšie v porovnaní s tradičnými metódami strojového učenia. Naviac, implementácia daného modelu hlbokého učenia by mohla byť využitá pri zisťovaní anomálií takmer v reálnom čase.

Vedúci:	MSc. Philipp Roberto Miotti
Katedra:	FMFI.KI - Katedra informatiky
Vedúci katedry:	prof. RNDr. Martin Škoviera, PhD.
Dátum zadania:	27.10.2022

Dátum schválenia: 31.10.2022

doc. RNDr. Dana Pardubská, CSc.
garant študijného programu

....

študent vedúci práce

70681272

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Júlia Lichmanová
Study programme:	Computer Science (Single degree study, bachelor I. deg., full time form)
Field of Study:	Computer Science
Type of Thesis:	Bachelor´s thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Deep Learning Applied to Anomaly Detection in Meteorological Time Series

Annotation: The goal of this bachelor's thesis is to demonstrate the feasibility and practicability of using Deep Learning methods for detecting abnormal sensory behavior in instruments measuring meteorological variables represented as a multivariate time series. Nowadays, fast and precise analysis of data in the solar energy industry is in high demand as the solar energy market is growing rapidly. Utilizing this proof of concept could make it possible to label anomalies in large amounts of time series data more efficiently compared to traditional Machine Learning methods. Furthermore, implementing such a Deep Learning model could be used for detecting anomalies in near real-time.

Supervisor:	Master of Science Philipp Roberto Miotti	
Department:	FMFI.KI - Department of Computer Science	
Head of department:	prof. RNDr. Martin Škoviera, PhD.	
Assigned:	27.10.2022	
Approved:	31.10.2022	doc. RNDr. Dana Pardubská, CSc. Guarantor of Study Programme

....
Student

....
Supervisor

Acknowledgments: I am incredibly thankful for the unwavering support and patience of my supervisor, MSc. Philipp Miotti. Throughout my journey, he has been a constant source of guidance, patiently addressing all my misunderstandings and offering expert advice. Also, I would like to thank Solargis for providing me with labeled data. Their provision of essential hardware has played a significant role in enabling me to carry out my work effectively. I am truly grateful for their support and the invaluable experience gained through this collaboration.

Abstrakt

Kontrola kvality je dôležitou súčasťou, ktorá zabezpečuje určité štandardy, presnosť a spoľahlivosť údajov v rôznych oblastiach riadených dátami. V kontexte systémov solárnej energie zohráva kontrola kvality kľúčovú úlohu pri overovaní satelitných a pozemne meraných dát. Súčasné metódy kontroly kvality primárne využívajú štatistické kontroly a detekciu výstredných hodnôt na základe limitov. Pokrok v oblasti výpočtových prostriedkov a dostupnosť veľkého množstva dát vytvorili základ pre nástroje strojového učenia a hlbokého učenia, ktoré by mohli potenciálne zlepšiť proces kontroly kvality. Táto práca skúma použitie metód hlbokého učenia, konkrétne Anomaly Transformer a TranAD založených na architektúre transformer, pre detekciu nereálnych hodnôt v meteorologických dátach meraných na zemi.

Kľúčové slová: hlboké učenie, detekcia anomálií, kontrola kvality, transformer, meteorologické časové rady

Abstract

Quality control is an important part that ensures certain standards, accuracy and reliability in data of various data-driven domains. Particularly in the context of solar energy systems quality control plays a key role in validating satellite-based and ground-measured data. Current quality control methods primarily employ statistical checks and limit-based outlier detection. The advancement of computational resources and availability of large amounts of data have created the foundation for machine learning and deep learning tools that can potentially improve the quality control process. This thesis explores the application of deep learning methods, specifically Anomaly Transformer and TranAD based on transformer architecture, for detecting unrealistic values in ground-measured meteorological data.

Keywords: deep learning, anomaly detection, quality control, transformer, meteorological time series

Contents

List of Figures

Introduction

Quality control (QC) is an important process in any domain where value is retrieved from data. It assures that the collected data adhere to a certain standard for further processing. The process of QC consists of a sequence of operations on the data involving domain experts and information technology tools. Especially, data from solar energy systems must be quality controlled before further use [4]. For example, it is an important part of the solar energy system design, validating satellite-based solar resource data and data assimilation of forecasting models.

In general, QC aims to uncover unrealistic values in measured data of various resources. Unrealistic values can be categorized by the source of error into two classes: equipment errors and operational-related errors [5]. Operational-related errors are caused by insufficient control of the measurement process, wrong calibration of the instrument, or accumulation of environmental material on measuring instruments. However, device errors are caused by malfunctioning of the measuring instrument or components.

In the last years, new statistical, machine learning and deep learning methods have been applied to QC. The combination of advanced high computing technology and domain knowledge brings several solutions to QC. Existing tools for QC of irradiation data are based mainly on applying limits on irradiance values from overcast and clear-sky models, and some of them also implement basic statistical tests for outlier detection [5]. A similar approach is done to wind data [6] and other meteorological variables [7], where step, consistency, temporal and spatial checks are applied. In recent years with the advance of fast, powerful processing units and large amounts of data, many new algorithms have been designed based on statistical principles, machine learning algorithms and deep learning methods. These high-performance tools in combination with domain expert knowledge have the potential to leverage the QC process.

In this thesis, we aim to demonstrate the practicability of deep learning methods used for detecting unrealistic values in meteorological data. To our knowledge, the concept of such deep learning techniques applied to anomaly detection in meteorological time series has not been explored in recent literature. Specifically, we train deep learning models such as Anomaly Transformer [2] and TranAD [8] based on the Transformer architecture [1]. Since the Transformer architecture has shown great potential in natural language processing, they have brought solutions to other fields as well. By using the ability to capture long-range relations in data, these models achieved so state-of-the-art results in anomaly detection.

Chapter 1

Background

In this chapter, we explain concepts such as machine learning, deep learning and basic data science techniques for a better understanding of the problem we are dealing with. We give an introduction to the basics of machine learning in Sec. 1.1, where we present a holistic overview of the machine learning landscape, the necessary steps regarding data preparation and the tools to evaluate machine learning models. In Sec. 1.2 we present an important technique how to extract insightful information from high-dimensional datasets. The basics of deep learning and artificial neural networks including the famous Transformer architecture are introduced in Sec. 1.4. We conclude this chapter by defining time series and anomaly detection in Sec. 1.5 and Sec. 1.6, respectively.

1.1 Machine Learning

Machine learning is a subfield of artificial intelligence that aims at studying algorithms that allow machines to learn on their own from training data. In other words, this field studies models that can be trained with specific data on a specific task. A machine learning algorithm aims to build a model based on observed data and uses the model to approximate a function that describes a relationship in the data.

Machine learning can be divided into several categories based on different characteristics [9]. We divide them by whether or not human supervision was used during the training phase, by the ability of the model to learn incrementally on new data, and by the strategy of the models to make predictions based on the similarity of new data points to known data points or on learned patterns. By the degree of human supervision machine learning models can be divided into four groups supervised learning, semi-supervised learning, unsupervised learning and reinforcement learning. The first three groups differ in the availability of labeled data. Supervised learning refers to training with only labeled data, whereas semi-supervised learning combines labeled and unlabeled data. Labeling data requires a lot of human work, so in the case of small amounts of labeled data semi-supervised or unsupervised are the only options. Unsupervised learning uses no labeled data but needs a much bigger amount of training data. With reinforcement learning, models learn by observing the environment and are rewarded in the case of correct behavior and penalized in the opposite case. By the ability of the model to be trained incrementally or not, we differentiate between online learning and batch learning. Batch learning refers to training on a fixed data set in one phase, whereas online learning refers to training incrementally on new data as they come in.

Machine learning models are useful in situations where a solution for a task is too complex to be described in a classical programming paradigm. We can write a classical algorithm that will recognize hand-written digits, but there will be almost infinitive combinations of vectors that fit a given digit. Such algorithms will not be executable in real time. Since the machine learning algorithm generates a model based on input data, we must be able to determine the performance of the model, or whether the model adequately describes the data. To accomplish this, there are various metrics to determine the performance of the model.

1.1.1 Performance

Many factors have an impact on the performance of machine learning models. First of all, when the data is of poor quality, contains too many outlier values, noise, missing values or labeled data contains erroneous labels even the best models cannot perform correctly. Poor data quality significantly reduces the accuracy of the model and the ability to learn patterns. To avoid this it is important to apply data-cleaning steps. For example, removing outliers, imputing missing values, etc. As irrelevant features can cause an inability of learning, it is very important to have a good set of features to train on. To prepare the features for the model we use certain data preprocessing techniques called feature engineering. Especially with a large number of features, it is important to deliberately analyze them in order to select and extract the most significant information. Feature engineering consists of two steps: feature selection and feature extraction. Feature selection is analyzing the usefulness of each feature and selecting the most relevant ones. The second step is feature extraction, which creates new features by combining existing features to create more essential ones. Even for the most sophisticated machine learning models, training data play a crucial role in the ability of machine learning algorithms to generate a model that can be used to predict future unseen data.

The choice of a machine learning model for a given set of data can significantly influence the ability of a machine learning algorithm to learn patterns in the data. To choose the right model a deep understanding of the data is required. For example, for

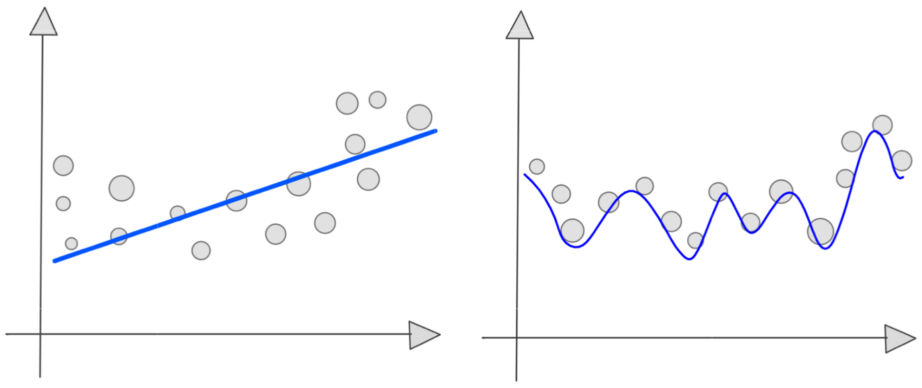


Figure 1.1: The left part of this figure shows a case where a linear model is underfitting the data. The right part of this figure shows a case where the polynomial model is overfitting the data.

data that have a linear trend a linear model would be the right choice for a regression algorithm. On the other hand, choosing a higher-order polynomial model for linear data in general leads to overfitting on the right side in Fig. 1.1. The model is too sensitive to small fluctuations in data and can fit even noise rather than linear trends. As a result, the model is performing well on training data but poorly on test data. In that case, we say that such a model does not generalize well, i.e., the model is said to be too complex for describing the data. On the contrary, when we choose a too-simple model on highly complex data, the model will not have the capacity to learn the underlying patterns. For example, a linear model is not able to generalize data with a sinusoidal periodicity. This situation is called underfitting on the left side in Fig. 1.1. When choosing a model for any given data, we encounter a so-called bias-variance tradeoff, where simple and complex models tend to have high bias and high variance, respectively. Bias represents a difference between the approximate function and the real function describing data, and variance represents the amount of change in the approximate function due to changes in the data.

1.1.2 Evaluation

Evaluating a model is a process, where the model's performance is measured while running on new unseen data. To evaluate a model, several scoring metrics are used. Before defining them we explain some basic concepts.

Comparing the model's predicted values with the target values, i.e., labels, we identify four possible classes: true positives, true negatives, false positives and false negatives. A true positive (TP) corresponds to a correctly predicted positive value, and a true negative (TN) corresponds to a correctly predicted negative value. TP and TN occur when predicted values match their respective labels. Contrarily, when a model is incorrect there are two types of errors: a false positive (FP) occurs when the model's prediction of a positive value is incorrect. Also, a false negative (FN) occurs when the model's prediction of a negative value is incorrect. Based on these four classes we can construct various metrics. One of the most broadly used metrics, called accuracy, is defined in Eq. (1.1). Accuracy represents the ability of the model to correctly predict values. Specifically, it says about the proportion of correctly predicted values with respect to all values. Other metrics are more specific to the type of prediction error. Precision, defined in Eq. (1.2) expresses how many of the predicted positive values were correct, i.e., the model's ability not to predict a value as positive that is labeled as negative. Recall, defined in Eq. (1.3), expresses how many true positive values were correctly predicted, i.e., the ability to find all the positive values. By taking the harmonic mean between precision and recall, we get another metric called the F1-score, see Eq. (1.4). This metric summarizes the trade-offs between the two error types FP and FN [9].

$$Accuracy = \frac{TruePositive + TrueNegative}{TotalPrediction} \tag{1.1}$$

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \tag{1.2}$$

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative} \tag{1.3}$$

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} = \frac{2TP}{2TP + FP + FN} \tag{1.4}$$

Measuring the accuracy of a model is an essential part of its development, so the above metrics are very necessary. In order to be able to direct the training of the model in the right direction a sample is selected from training data called validation data, with which the accuracy of the model is tested. This phase of measuring the accuracy of the model is called model validation. After the model has been developed and the training is over, the final model is evaluated on unseen data.

1.2 Dimensionality Reduction

As technology progresses, a large amount of data is generated and accumulated. Data does not expand only in length, but also in breadth as more interrelated data is measured. With the increasing number of features, data are more complex and difficult to comprehend. Since a deep understanding of data in machine learning is required, many techniques that reduce the complexity of data have been developed.

1.2.1 Principal Component Analysis

Principal component analysis (PCA) is a technique that reduces the dimensionality of multivariate datasets. It builds a coordinate system that maximizes the variance of the studied data along its coordinate axes. In order to reduce the dimensionality of the dataset the axes with the least amount of variance are removed, which minimizes the information loss.

Let's suppose that our studied data follow a multivariate Gaussian distribution with different scales. Specifically, assume that this cluster of data is elongated along a specific direction. The aim of PCA used for dimensionality reduction is to identify a hyperplane that is closest to the data while maximizing its variance. PCA works by iteratively choosing axes that capture the maximum variance of the data. First, it chooses an axis along which the data have the largest variance. Then identifies the second axis with the largest remaining variance, orthogonal to the first one. Then the third one is orthogonal to the first and the second one. There are as many orthogonal axes as the number of dimensions of data. Our multidimensional space is defined by this mentioned set of orthogonal axes. A subset of axes that contains the least variance is removed from the multidimensional space and this lower-dimensional space defines a hyperplane. Each axis of the hyperplane minimizes the mean square error between the original data and its projection to the hyperplane. As a result of PCA, we get the hyperplane. Projecting the original data to this hyperplane a lower-dimensional representation of the data is obtained. PCA helps to uncover relations and patterns, that might be difficult to observe in the original high-dimensional data.

One of the methods to apply PCA is to decompose the input data matrix using Singular Value Decomposition (SVD). SVD is a standard matrix factorization technique that decomposes the input matrix M into three matrices. Matrix M can be expressed using the equation $M = U\Sigma V^T$ where Σ is a rectangular diagonal matrix with singular values on its diagonal. These values are ordered from the largest to the smallest. U and V^T are orthogonal matrices. Matrix U contains as columns left singular vectors. These left singular vectors represent eigenvectors of MM^T, where M is the original matrix. Likewise, matrix V^T contains as columns right singular vectors. These right singular vectors represent eigenvectors of M^TM. In the previously mentioned concept, the coordinate axes of the transformed coordinate system correspond to the eigenvectors of the matrix, and each eigenvalue corresponds to the variance of the data along its eigenvector [10].

Another method is based on computing the covariance matrix C from the input matrix M, where C has the same number of columns and rows. The diagonal values of C represent the variances of each feature in M, and the off-diagonal elements of C represent covariances between two different features. Covariance is a measure of the linear relationship between two variables, i.e., proportionality. For real-valued data the covariance matrix C of data matrix M is defined by,

$$C = \frac{1}{n-1} M^T M,$$

where n represents number of rows in M matrix. When the covariance between two features is positive and increasing, the two features tend to be more positively proportional, whereas, when the covariance is negative and decreasing, the two features tend to be more negatively proportional. A zero value for the covariance number corresponds to no linear relationship between two features. To find principal components, PCA performs eigenvalue decomposition of the covariance matrix, where eigenvectors and eigenvalues are computed. Eigenvalues represent the amount of variance of each feature. Eigenvectors represent directions where data spread is the largest.

A second application of PCA is for visualization purposes. The approach here is to choose two or three coordinate axes with the highest variance instead of discarding the axes with the least variance. Such a visualization helps us to get better insights from the data and get a deeper understanding of the machine-learning task. With PCA we can simplify complex information and interpret data in a human-understandable way.

1.3 Deep Learning

Deep learning is a machine learning technique to generate models based on artificial neural networks (ANN) using multiple layers of interconnected nodes. Generally, machine learning is a field that builds learning algorithms that are able to learn from data, without being explicitly programmed in contrast to traditional algorithms. The name "deep" refers to using a high number of layers in the network. More layers enable deep learning networks to better uncover the patterns in data. In general, this increases the accuracy of a model but also increases training time, computational resources and the amount of data necessary for training.

The high number of layers allows the model to learn the input data hierarchy. The hierarchy of the input data represents the levels of the features. For example, in an image classification task of a human face, lower levels represent edges and lines, middle levels represent the nose and eyes, and higher levels represent facial structures. The hierarchy of the input data in classical machine learning techniques is hand-engineered by a human, whereas deep learning is able to learn this hierarchy by itself. Because of that, deep learning requires larger amounts of data for training. With a larger amount of data and a higher number of layers the prediction power increases.

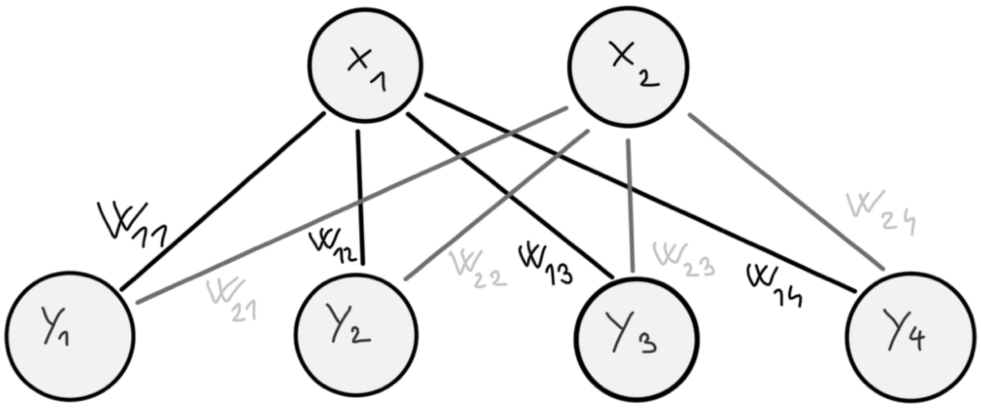


Figure 1.2: This figure represents a simple neural network, where x_i represent input neurons, y_i represent output neurons, w_{ij} represent weights.

An ANN consists of interconnected nodes called neurons. A simple layer of interconnected nodes is shown in Fig. 1.2. Each connection is assigned a value - weight, where weight represents the strength of the connection. The stronger the connection's weight, the greater the impact the neuron has on that connection. A very important part of ANN is its activation function, that represents a non-linear transformation in the model. It enables the model to solve more complex tasks, e.g., approximate non-linear functions. Without the activation function, ANNs are only able to represent matrix multiplication.

Generally, let's assume that input data are multidimensional, input into ANN is a real value vector, where each value corresponds to one feature. Mostly is one input neuron x_i for each input feature. In the next layers, each neuron x_i in the input is multiplied by its associated weight w_i and summed. Subsequently, onto weighted sum is applied activation function. Different neural networks vary by the number of neurons per layer, called width, and by the number of stacked layers, called depth of a neural network.

Deep learning training algorithms consist of an iterative learning process that updates the weights of the neural network. The process of the algorithm may differ according to the type of optimization algorithm used. But in general, we can divide this iterative process into four steps: forward pass, calculation of the loss function, backward pass and updating the weights. The first step is the forward pass. In this step, input data are fed into the model, and are processed by each layer. The second step consists of calculating the loss function. The loss function is a metric for evaluating how precise a deep learning model is. It computes an error between the predicted output of the neural network and the expected output. We can imagine the loss function as a convex function. The main goal of training a deep learning model is to minimize the loss function. The third step is the backward pass. The backward pass computes gradients of the loss function with respect to the model's weights. In modern neural network architectures, the gradients are computed using an algorithm called backpropagation, which is basically a highly effective way to compute the gradients using the chain rule for partial derivation. The fourth step consists of updating the weights. The weights are updated according to an algorithm using the gradients computed in the previous step. With the aim of minimizing the loss function, we want to converge to the point where the gradient of the loss function is zero. These four steps constitute one iteration of the training process with several data instances. The number of instances that are processed in one iteration is defined by the batch size of the model. This iteration repeats until all training data are processed. This iterative process constitutes one epoch of training.

ANNs include various architectures that have proven to be very powerful. One of the simplest architectures are feedforward neural networks (FNNs), which process data only in the forward direction. In an FNN architecture, all neurons from one layer are connected to neurons from the next layer. Another very powerful architecture types are convolutional neural networks (CNN) which achieve state-of-the-art in various computer vision tasks. CNNs are able to hierarchically capture the structure of an object. Recurrent neural networks (RNNs) are built for sequential tasks, and are able to capture short-term dependencies. On the other hand, RNNs struggle with long sequences to capture long-term dependencies, in such cases transformer architectures achieve state-of-the-art performance.

1.4 Transformer

The Transformer is a deep learning architecture designed by an AI research team at Google to solve sequential tasks. In their published paper "Attention Is All You Need" [1] they demonstrate that the Transformer architecture significantly improves the capabilities of models in natural language processing (NLP). Their new solution solves the short-term memory problem of recurrent neural networks (RNN) by capturing long-range dependencies between input elements of a sequence using an attention mechanism. Furthermore, it enables parallel processing of its inputs, which significantly speeds up the training process. The Transformer model contains two parts an encoder and a decoder. Specifically, the encoder is composed of two main parts: the Multi-Head Attention layer and a feedforward neural network (FNN). The decoder has almost the same structure but in contrast with the encoder, the decoder has another Multi-Head Attention layer in the middle.

1.4.1 Multi-Head Attention Mechanism

The Multi-Head Attention mechanism is a fundamental building block of the Transformer that allows for selective focus on different parts of input and output sequences. It simulates attention in the input sequence by assigning weights to each input object. The higher the weight, the stronger the relative importance or significance of the input object.

Apriory, three different matrices are given as input to the attention layer. The common convention is to name these matrices Q, K and V, which stand for query matrix, key matrix, and value matrix, respectively. Then these three matrices are fed into a dense layer, which does not contain an activation function. In other words, multi-head attention first applies a linear transformation to each matrix. The linear transformation aims to divide the whole feature space into different subspaces, where each subspace focuses on different features of the input. Each division can be regarded as a projection onto different subspaces. Next, the output matrix of the linear transformation is split into different blocks, where each block of the matrix represents a different subspace. Subsequently, each block is further processed by the Scaled Dot-Product Attention function,

$$\text{Attention}\left(Q_b, K_b, V_b\right) = \text{softmax}\left(\frac{Q_b K_b^T}{\sqrt{d_{\text{keys}}}}\right) V_b, \tag{1.5}$$

where Q_b is a $n_{\text{queries}} \times d_{\text{keys}}$ matrix, K_b is a $n_{\text{keys}} \times d_{\text{keys}}$ matrix and V_b is a $n_{\text{keys}} \times d_{\text{values}}$ matrix. Each Q_b, K_b, V_b represent the block of the input matrices Q, K, V after the linear transformation, respectively. The meaning of the matrix multiplication between Q_b and K_b^T is to create an attention matrix and d_{keys} acts as a scaling factor for the attention matrix. The functionality of the softmax function is to normalize the attention weights such that all rows in the attention matrix sum up to one. We can view the normalized attention matrix as an attention map between keys and queries. Each query-key pair of an attention map refers to a similarity score between the matrices Q_b and K_b. Subsequently, the attention map is multiplied by the value matrix V_b. Based on the attention map, the values, i.e. rows, of the value matrix V_b are linearly combined into resulting values. Finally, the results of each head are concatenated and embedded back into the original space. Specifically, from the point of view of embedding, it is thought that the concatenated results pass through a linear layer, or by linear transformation.

1.4.2 Architecture

Now we are going to describe the basic components of the Transformer architecture shown in Fig. 1.3. As input into the encoder is a raw sequence of objects. Since this representation is incomprehensible for the Transformer model, the input is encoded into vectors. First, each object is embedded considering the meaning and the position of the

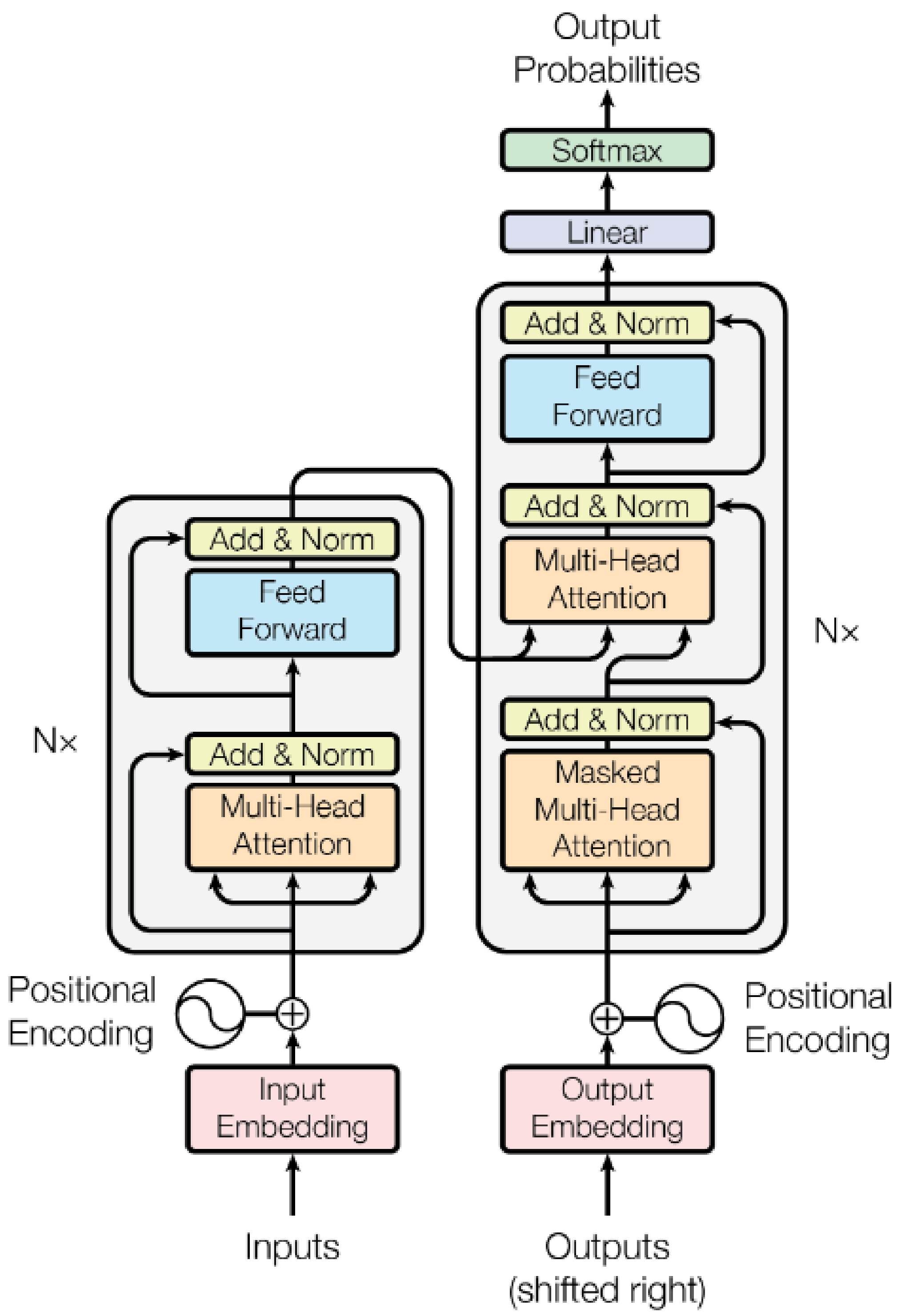


Figure 1.3: This figure represents the architecture of the Transformer model described in [1].

object, using word embedding and positional embedding. Specifically, word embedding is a process where each object in the sequence is embedded into a multidimensional vector based on its features. As a result, similar objects have similar vector representations. In algebraic representation, the similarity of vectors is represented as a distance from each other. Next, positional embedding encodes into each vector information that describes the absolute and relative position of the object in the sequence. The outputs of the embedding are unique dense vectors. Positional embedding is mostly implemented using a sine/cosine positional embedding matrix. This type of positional embedding has the advantage that it can be extended into arbitrarily long sentences. Combining word and positional embedding allows the transformer network to capture both, the relative position in the input sequence and the meaning of the object.

After the input has been encoded with the embedding layer it is passed into the encoder. The encoder consists of two types of neural network layers. First, it is sent through the Multi-Head Self-Attention layer and afterward through the FFN. In the attention layer weights are calculated between each vector pair of the input. The higher the weight between two vectors, the more relevant the vector. The main purpose of this attention layer is to create an attention map that contains the correlation values (attention weights) between each pair of vectors. Based on its weight, each object is given some attention from another object in the input itself. That's why this layer is called the self-attention layer. Next, the output of the layer is added to the output of the previous embedding layer. This type of connection linking non-adjacent layers is called a skip connection. This connection positively affects the stabilization of the training. Subsequently, the output of the skip connection is normalized, which also stabilizes the training. Next, the output of the normalization layer is passed through the FFN. Similarly, a skip connection and a normalization layer are applied to the output of the FFN in the same way as for the attention layer. The part of the encoder that consists of the Multi-Head Self-Attention and FFN is stacked n times in a row, and the stack's final output is fed into the decoder. The only parameters that are updated in the encoder are the weights in the FFN and in the linear transformation applied at the beginning of the Multi-Head Attention layer.

The input into the decoder during the training phase is a target sequence shifted one step to the right and a start token is placed at the beginning. Similarly, as in the encoder word and position embeddings are applied to the input, followed by a masked Multi-Head Self-Attention layer. This masked attention layer is almost the same as in the encoder, except that each object in the target sequence is allowed to attend only to the preceding objects in the sequence. The mask is necessary during training because, during the inference phase, the input to the decoder does not have a known remainder of the input, unlike in the training phase. The output of this masked self-attention layer, i.e. query matrix, together with the value and key matrix from the encoder are taken as input to the Multi-Head Attention layer. The combination of these outputs from the encoder and from the decoder's multi-head self-attention can get information about the encoder's input sequence and the context of the current decoder state. With this combination, the Multi-Head Attention layer enables the decoder to target the relevant parts of the input sequence while generating the next object in the output sequence. Next, the output of the attention layer is added to a skip connection, and a normalization layer is applied to the addition before being passed onto the last FFN. Finally, a skip connection, a normalization layer and a linear transformation with a softmax function are applied to the final result. The attention mechanism has found its way into almost all types of deep learning architectures and applications, not only NLP but also in image processing, generative learning, time series analysis and anomaly detection.

1.5 Time series

Generally, a series of data is a list of ordered data points. Time series data is a sequence of data points collected over time with a fixed sampling frequency. Each index is represented by a time measure, e.g., date, hours, and seconds. Time series data is unambiguously arranged in a timeline from the past to the future. Typically time series data are used everywhere where information is correlated in time, e.g., stock market, real-time measurements. Time series can be divided into two types based on a number of dimensions - univariate and multivariate. Univariate time series have only one variable measured over time. Each measured value has its timestamp. Multivariate time series contain multiple variables measured at each timestamp. Attributes of time series data can be divided into two types based on dependencies - contextual and behavior. Contextual attributes define the context of the data. For example, a timestamp in time series refers to a contextual attribute. Timestamp mostly represents the date and time when the record was measured. It gives the context of each measure altogether and it is an essential component of time series.

Based on the behavior we can decompose the time series into four components - trend, season, cycle and irregular. Each component expresses the aspects of the pattern of the time series. A secular trend refers to a global long-term, gradually changing pattern of the data. Seasonal variations represent a pattern that repeatedly returns itself over some period, e.g., month, year. Cyclical fluctuations don't repeat fixed patterns in periods. Irregular variations represent randomly behaving patterns with unexpecting events. If some data points or subsequences of time series do not have any of these aspects it refers to anomalies.

1.6 Anomaly detection

Anomalies are single data points or consecutive data with unusual behavior with respect to the majority of data, called point anomalies or collective anomalies, respectively. Anomaly data occur in a very small portion of the data and are significantly different from the majority of data. Point anomalies and collective anomalies can be classified into several groups depending on the magnitude, shape, periodicity and trend of the anomaly values.

Point anomalies represent values that are significantly different from their neighbors. They show up as one unexpected value over time. Specifically, point anomalies based on the magnitude of the anomaly value are classified as global and contextual. A global anomaly is a significant point with unrealistic value on a global scale. It significantly deviates from the usual patterns in the data. We can imagine it as an out-of-range of certain proximity value. This type of anomaly is easy to observe. However, a contextual anomaly is a point that is observable on small scales. This value does not take an extreme value but in the context does not make sense. Contextual anomaly deviates from its neighbors but takes on a value within a certain range of proximity.

Collective anomalies based on the other characteristics mentioned above are classified as shapelet, seasonal, and trend anomalies. In the case of shapelet anomalies, the pattern is not consistent with similar patterns in the area. With seasonal anomalies, the periodicity deviates from the normal course. Trend anomalies are specified when the mean value in a given time window has suddenly changed.

Anomaly detection is an essential part of the measured data for their use. Since there are many different types of anomalies, finding anomalies is a very complex process, also is an actual research topic in the field of AI, machine learning, deep learning, etc.

Chapter 2

Related Work

Anomaly detection is being applied in several fields using methods from classical machine learning, deep learning, signal analysis, data mining and statistics. Anomaly detection techniques can be categorized according to the ability to work on multiple dimensions - multi-dimensional time series or one dimension - univariate time series. Anomaly detection techniques can also be categorized by the amount of needed supervision: unsupervised, semi-supervised and supervised. Unsupervised techniques are able to detect anomalies without prior knowledge, i.e., labeled data, assuming that anomalies are abnormal and rare. They aim to learn patterns of normal data, which has the effect to differentiate and detect anomalous data. Supervised techniques require prepared data with labels for each value if it is an anomalous value or normal value. This technique is not practical and is very rarely used because human labeling is expensive and anomalies can have various shapes. Semi-supervised techniques combine the techniques of unsupervised and supervised learning, where input data are partially labeled. Models can be categorized by their learning algorithm used into: forecasting, distribution, reconstruction, distribution, encoding, distance and tree-based methods.

Forecasting methods

The main purpose of forecasting methods is to predict future values from learned data. In the case of anomaly detection, it is based on observation of the differences between forecast and observed values. Wu et al. [11] demonstrate an effective forecast solution for unsupervised learning for anomaly detection in time series, using Local Trend Inconsistency and RNN. Also, Munir et al in. [12] present a deep learning model called DeepAnT, which is based on CNN and consists of two parts: a time series predictor and an anomaly detector.

Tree-based methods

Tree-based methods are very intuitive and based on simple principles adopted from computer science. One of the first efforts to detect anomalies using tree-based methods called isolation forest was published by Liu et al. [13]. Nowadays, it is one of the most popular tree-based methods for anomaly detection. The method is based on an algorithm that applies binary trees. It splits the data space recursively using orthogonal lines, where anomalies correspond to the points that have the lowest number of splits. In other words, the anomalies are represented by very short branches. In recent years, many follow-up publications, i.e. Hariri et al. [14], extending the original isolation forest algorithm have been published.

Reconstruction methods

Reconstruction methods aim to detect patterns in a lower-dimensional representation of the input series, and use these patterns to create a reconstruction of the original series. Subsequently, the reconstructed values are compared with the observed values, and anomalies are detected based on these differences. Zhang et al. in [15] propose a solution that detects anomalies and provides a diagnosis in which dimension anomaly has occurred based on Multi-Scale Convolutional Recurrent Encoder-Decoder (MSCRED). Also, Su et al. [16] propose an OmniAnomaly model that is based on stochastic variable connection and planar normalizing form.

Encoding methods

The encoding method is very similar to the reconstruction technique, though in encoding the input is encoded into a lower dimensional space, and the anomalies are detected from this representation. Gao et al. in their work [17] propose a method based on ensemble learning for anomaly detection. Also, Boniol and Palpanas in [18] propose an unsupervised solution for anomaly detection that encodes the input into a lower dimensional representation, and represents this encoded input in a graph.

Distance methods

Distance-based techniques work by comparing distances between different points or subsequences of a time series. Mostly, anomalies are detected by having unusually long distances compared to normal data. Song et al. in [19] proposes a hybrid semi-supervised anomaly detection model that is based on an autoencoder and k-nearest neighbor algorithm. Boniol et al. [20] proposed the SAND model which is adaptive to the distribution drifts and omits unused data.

Distribution methods

Distribution techniques are based on estimating the distribution of the data. Anomaly detection then corresponds to identifying points that significantly differ from the estimated data distribution, or searching for discrepancies in the frequency of occurrence of a subsequence. Boniol, et al. presented a distribution technique for anomaly detection called COPOD [21], that constructs copula for modeling distribution. Also, Hochenbaum et al. in [22] proposed a solution based on statistical learning using seasonal decomposition and seasonal components to detect anomalies.

Chapter 3

Transformer models used

In our investigation of anomaly detection techniques, we arrived at the decision to use deep learning models based on the Transformer architecture for anomaly detection. As the transformer architecture has shown potential in many areas, it has also proven to be very powerful in anomaly detection. We have selected two specific models: Anomaly Transformer[1] and TranAD[2] described in Sec. 3.1 and Sec 3.2, respectively. Both of these models have consistently showcased state-of-the-art performance on distinct datasets.

3.1 Anomaly Transformer

In general, it can be observed that the majority of data is characterized by normal patterns, whereas anomalies are infrequent occurrences. Consequently, anomalies face considerable difficulty in establishing robust associations with the entirety of the data. The term "anomaly" itself suggests a unique, abnormal, or rare event. This particular feature was used as the basic idea when building the Anomaly Transformer by Xu et al. [8]

The Anomaly Transformer is based on the Transformer architecture [1]. It uses the Multi-Head Self-Attention mechanism and a feedforward neural network. Since the Anomaly Transformer focuses on the detection of anomalies in time series, the architecture differs from the Transformer architecture described in Sec. 1.4 of the Background chapter. The multi-head attention layer consists of two main parts Series-association S and Prior-association P which we are going to elaborate on in more detail.

The Series-association corresponds to the multi-head self-attention mechanism. The name self indicates that the input sequence is paying attention to itself. As input into the Series-association are two matrices K and Q. Next, the linear transformation is

[1]https://github.com/thuml/Anomaly-Transformer

[2]https://github.com/imperial-qore/TranAD

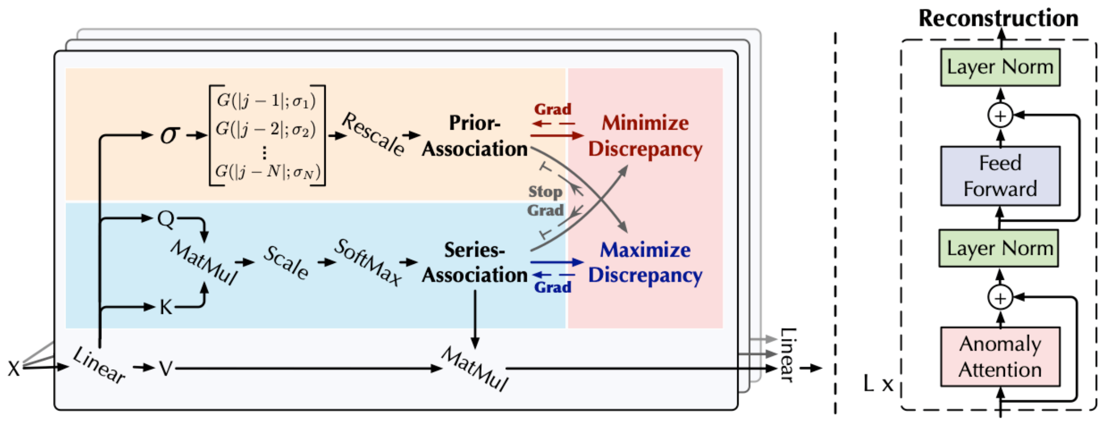


Figure 3.1: This figure represents the architecture of the Anomaly Transformer model described in [2].

applied to these matrices. Since this attention mechanism is multi-headed, the output matrix of a linear transformation is split into different blocks, and each block is fed into one head of the attention layer. In each head, the Scaled Dot-Product Attention is applied (see Eq. (1.5)), which is used to create a reconstruction of the input. The Series-association between the points of the input sequence is just the attention matrix of the formula in Eq. (1.5). In each head, we also compute the Prior-association of the input. Basically, it is a Gaussian kernel with learnable parameters σ_i. The Prior-association assigns one parameter σ_i for each timestamp i of the input sequence. As an output of the Prior-association is a Gaussian distribution where the parameter σ_i corresponds to the standard deviation of the Gaussian distribution. In other words, σ_i represents how spread the peak area is. With these parameters, the Gaussian kernel is adaptive to various patterns in the input sequence and is able to capture different durations of anomalies. Subsequently, the output, i.e. reconstruction of each head is concatenated into one output of the Multi-Head Attention layer.

From the output of the Series-association and of the Prior-association with Kullback-Leibler divergence (KL) is computed the Association discrepancy,

$$\mathrm{AssDis}\left(P,S,X\right)=\left[\frac{1}{L}\sum_{l=1}^{L}\left(\mathrm{KL}\left(P_{i,:}^{l}\|S_{i,:}^{l}\right)+\mathrm{KL}\left(S_{i,:}^{l}\|P_{i,:}^{l}\right)\right)\right]_{i=1,\ldots,N}, \tag{3.1}$$

where L corresponds to the total number of stacked layers of the Attention mechanism and FNN, and X represents the input series. The Association discrepancy is a metric that quantifies the difference between two distributions - the Series-association and the Prior-association. The Kullback-Leibler divergence is employed to calculate this discrepancy, with P and S representing the Prior-association and Series-association, respectively. The discrepancy is evaluated for each timestamp i, where N represents the length of the input sequence (see Eq. (3.1)). A higher association discrepancy indicates a greater disparity between the two distributions. Conversely, a smaller value for the Association discrepancy indicates a higher level of similarity between the two distributions. The Gaussian curve has a characteristic bell-shaped curve that resembles a shape of an anomaly. As a result, a small number of the Association discrepancy will indicate the anomaly.

In order for the algorithm to be able to direct training in the right direction, it is necessary to determine the loss function that enables to evaluate the model's performance. The lower the value of the loss function, the more accurate the model's predictions are. The goal during training is to upgrade the parameters to minimize this loss function. The following loss function is used to evaluate the model during training,

$$\text{Loss}_{\text{Total}}\left(\hat{X}, P, S, \lambda, X\right) = \|X - \hat{X}\|_F^2 - \lambda \|\text{AssDis}(P, S, X)\|_1, \tag{3.2}$$

where $\hat{X}$ represents the reconstruction of input X, λ determines the loss terms, $\|\cdot\|_F$ is the Frobenius norm and $\|\cdot\|_k$ represents the k-norm, where k-norm for a vector $x = (x_1, ..., x_n)$ is defined as, $\|x\|_p := \left(\sum_{i=1}^{n} |x_i|^p\right)^{\frac{1}{p}}$. Finally, the above concepts will be used in anomaly detection. The model detects anomalies based on a combination of reconstruction error and the Association discrepancy.

$$\text{AnomalyScore}\,(X) = \text{Softmax}\left(-\text{AssDis}(P, S, X)\right) \odot \left[\|X_{i,:} - \hat{X}_{i,:}\|_2^2\right]_{i=1,\ldots,N} \tag{3.3}$$

Anomalies reduce the Association discrepancy, but due to the fact that the Association discrepancy is negated in the equation, it will drive a higher anomaly score. Since the reconstruction error represents the difference between the actual and predicted values, the value of the reconstruction error will be higher if there are anomalies in the input sequence.

3.2 TranAD

TranAD is a deep learning model proposed by Tuli et al. [3] for detecting anomalies using the attention mechanism together with an adversarial training style in a two-phase approach. It was created by extending the transformer architecture in [1], and combining it with the training principles of a generative adversarial network (GAN). One significant benefit of TranAD is its ability to not only identify a timestamp at which an anomaly arises but also determine the exact dimension where the anomaly occurred.

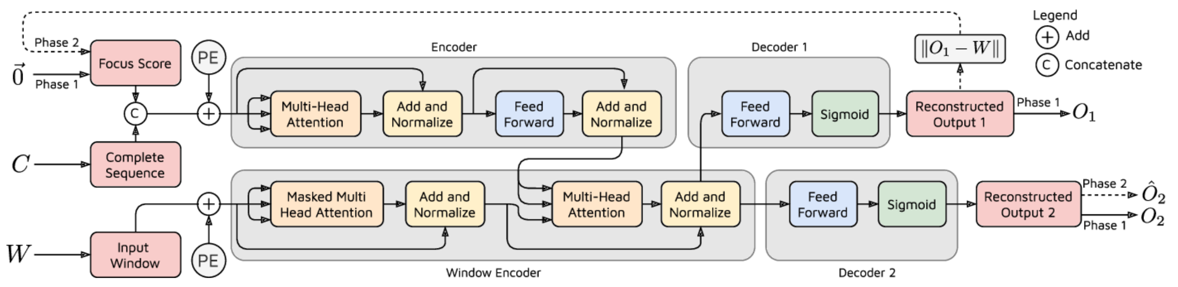


Figure 3.2: This figure represents the architecture of the TranAD model described in [3].

First, the raw univariate or multivariate time series $T = \{x_1, ..., x_N\}$, where N represents the length of the input series, is processed into smaller time series $C_t = \{x_1, ..., x_t\}$, where $x_i \in \mathbb{R}^m$, and $W_t = \{x_{t-K+1}, ..., x_t\}$, where K is the window length, for each $t \in \{1, ..., N\}$. In the univariate case $m = 1$, and in the multivariate case $m > 1$. At each training step, the model gets as input one pair of C_t and W_t at a time together with their respective positional encodings. In addition to these two inputs, the model also processes another input $F \in \mathbb{R}^{K \times m}$, called focus score. The encoder takes as input the concatenation of C_t and F, and generates key and value matrices for the window encoder using a multi-head attention layer and a feedforward neural network (FNN). The value and key matrices offer comprehensive global pattern and temporal trend insights from the entire input sequence C_t. The window encoder creates an abstract representation of W_t by applying a masked multi-head attention layer, and yields a query matrix that is further processed in another multi-head attention layer together with the key and value matrices from the first encoder. Due to the similarities between this transformer architecture and the architecture of the Transformer in [1] we can draw an analogy with querying a lookup table: the window encoder queries local patterns if they are present in the global pattern. Combining the query, key and value matrices provides sufficient information to facilitate reconstruction in two identical decoders. Each decoder is composed of an FNN and a sigmoid activation function, which non-linearly transform the output into the range $[0, 1]$. This step is essential for ensuring that the normalized input window is properly matched. The outputs of each encoder are then used to evaluate the model's loss function.

Encoder

The encoder in Fig. 3.2 takes the complete time series C up to a specific timestamp t, concatenated with the focus score F, as input. The focus F is initialized as a $K \times N$ matrix, where K corresponds to the window size and N corresponds to the dimension of the input series. This model tries to detect anomalies in each dimension in the time series. Therefore, the output of the loss function is a vector, where each value corresponds to the value returned by the loss function of one single dimension. Similar to the encoder in the Transformer architecture mentioned in Sec. 1.4 of the background chapter works this encoder. The encoder first preprocesses the input with positional encoding. Next, a multi-head attention layer is applied to the input sequence, which applies weights to the input. The output of the attention layer is added to a skip connection and then normalized. Finally, the result is passed through a feedforward neural network (FFN) and normalized again.

Window Encoder

The process involves the utilization of a fixed-length input window, denoted as W, which is input into the window encoder (see Fig. 3.2). The input window is first encoded using positional encoding, and subsequently passes through the masked multi-head self-attention. The application of masked multi-head self-attention serves to prevent the decoder from accessing future timestamps, as the encoder and window encoder work in parallel across all inputs. The resultant output is added to a skip connection and normalized. Next, the output query matrix from the window encoder, together with the value and key matrices obtained from the initial encoder, are processed by the multi-head attention layer. The value and key matrices offer comprehensive global pattern and temporal trend insights from the entire input sequence. The combination of the window encoder query matrix and encoder matrices provides sufficient information to facilitate reconstruction prediction. The multi-head attention mechanism produces an output that is combined with a skip connection and subsequently normalized. The resulting output is then fed into two identical decoders. Each decoder is composed of an FFN and a sigmoid activation function, which non-linearly transform the output to the range of $[0, 1]$. This step is essential for ensuring that the normalized input window is properly matched.

Training phase

The training step of the model is done in two phases. In the first phase, the input sequence C (until the specific data point x_t) concatenated with the focus score F, here initialized to zero, is fed into the encoder. The encoder identifies global patterns and temporal trends of the input data and provides this information to the window encoder. This information is retrieved from the attention map of the multi-head self-attention layer. Also in this first phase, the window encoder is fed with the input window W. The role of the masked multi-head self-attention layer is to provide information from the attention map about the local patterns and trends of each particular window. Due to the similarities between this transformer architecture and the architecture of the Transformer in [1] we can apply an analogy of querying a lookup table. The window encoder is querying local patterns if they are present in the global pattern. Next, the output is fed into two decoders. Both decoders independently create reconstructions of the input windows O_1 and O_2,

$$L1 = \|O_1 - W\|_2,$$

$$L2 = \|O_2 - W\|_2,$$

where $L1$ and $L2$ represent reconstruction losses. The reconstruction loss $L1$ is set as the focus score with zero padding to match the dimension of the input sequence C. The second phase begins by concatenating the focus score F with the input sequence C. Similarly, the encoder and the window encoder process the inputs. The fact that the loss function is incorporated into the second phase is to emphasize the differences that the model was not able to learn in the first phase. Specifically, this type of training is called adversarial training. The output of the window encoder is fed into the second decoder and reconstructs an output $\hat{O}_2$. Here the second encoder tries to maximize the difference between the reconstruction of the input window $\hat{O}_2$ and the input window W, whereas, in the first phase, the decoders aim to minimize the difference. This adversarial training procedure can be represented as two loss functions,

$$\hat{L}1 = +\|\hat{O}_2 - W\|_2,$$

$$\hat{L}2 = -\|\hat{O}_2 - W\|_2,$$

where $\hat{L}1$ and $\hat{L}2$ represent adversarial losses, $\hat{O}_2$ represents input reconstruction of the second phase. Combining reconstruction loss and adversarial loss from both phases, we get the following equations,

$$L1 = \epsilon^{-n}\|O_1 - W\|_2 + (1 - \epsilon^{-n})\|\hat{O}_2 - W\|_2,$$

$$L2 = \epsilon^{-n}\|O_2 - W\|_2 - (1 - \epsilon^{-n})\|\hat{O}_2 - W\|_2,$$

where ϵ represents a training hyperparameter controlling the weight of the individual losses from each phase. This combination ensures, the stabilization of the training. When in the first phase reconstruction is very poor, then the focus score in the second phase will be unreliable. Due to this, the adversarial loss gives low weight in the initial part. On the other hand when the focus score is reliable, the weight to the adversarial loss increases.

We now describe the anomaly detection procedure of TranAD [3]. When running the model on new time series inputs, the model first generates reconstructions O_1 and $\hat{O}_2$, where the first and second outputs are the reconstructions of the first decoder in the first phase and of the second decoder in the second phase, respectively. Based on these reconstructions the anomaly score vector s for the TranAD [3] is defined as,

$$s = \frac{1}{2}\|O_1 - \tilde{W}\|_2 + \frac{1}{2}\|\hat{O}_2 - \tilde{W}\|_2,$$

where $\tilde{W}$ represents a sequence of unseen data for which we want to detect anomalies. The anomalies of each time series dimension are then based on a threshold function, as well as the anomalies for each timestamp across all dimensions.

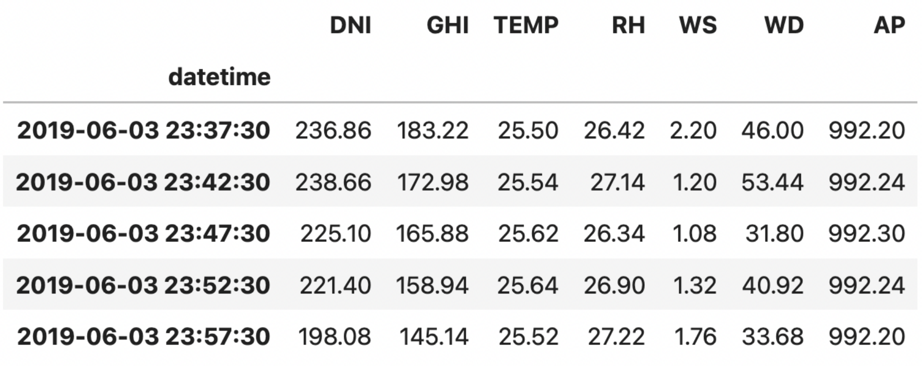
Chapter 4

Experimental Setup

In this chapter, we describe in Sec. 4.1 the dataset we used for training the models presented in the previous chapter. Then, we proceed in Sec. 4.2 by explaining the various data exploration and preprocessing steps that were necessary before feeding the data into the models. Finally, we present the details about their hyperparameter configurations in Sec. 4.3

4.1 Dataset

In our experiment, we use meteorological time series data collected by meteorological instruments measuring various weather variables worldwide. The data sources are in table A.1, where each site name corresponds to the specific provider responsible for collecting the data. In total, the dataset comprises nearly 17 million records, with a sampling frequency of 5 minutes. Data includes seven dimensions, that provide valuable information about meteorological conditions. The first dimension corresponds to global horizontal irradiance (GHI). GHI indicates the total amount of sunlight energy received from all directions by a surface of unit area per second. The second dimension is direct normal irradiance (DNI) which corresponds to the portion of sunlight that is incident on a unit area perpendicular to the direction of the sunlight coming in a straight line from the sun. In our dataset, GHI and DNI are quantified in watts per square meter (W/m^2). Air temperature 2m above ground level (TEMP) is measured in degrees Celsius. Wind speed (WS) represents the speed of air movement, and is quantified in meters per second (m/s). Wind direction (WD), specifies the compass direction of wind velocity from which the wind is blowing, and is quantified in degrees in range (0, 360). Relative humidity (RH) represents the amount of moisture in the air relative to its maximum at the same temperature. RH is quantified in percentage. Atmospheric pressure (AP) is the pressure of air mass inside a column of the atmosphere exerted on the Erth's surface quantified in $mbar$.



	DNI	GHI	TEMP	RH	WS	WD	AP
datetime							
2019-06-03 23:37:30	236.86	183.22	25.50	26.42	2.20	46.00	992.20
2019-06-03 23:42:30	238.66	172.98	25.54	27.14	1.20	53.44	992.24
2019-06-03 23:47:30	225.10	165.88	25.62	26.34	1.08	31.80	992.30
2019-06-03 23:52:30	221.40	158.94	25.64	26.90	1.32	40.92	992.24
2019-06-03 23:57:30	198.08	145.14	25.52	27.22	1.76	33.68	992.20

Figure 4.1: This figure illustrates a sample of the utilized dataset.

4.2 Data preprocessing

In the previous section, we introduced the dataset and its features. In this section, we apply feature transformations and explore deficiencies of the dataset through data preprocessing. This step involves examining the data and handling imperfections, as the measurements are not in a form suitable for neural networks.

In the raw dataset, wind direction is a time series, i.e., a function of time, whose range values are represented in angle degrees relative to the geographic North Pole in Fig. 4.2. The transition between 359 degrees and 0 degrees may be considered an anomaly by machine learning models as it represents a significant jump in this representation of wind direction. To avoid this type of discontinuity, we decided to utilize two separate columns. One column now represents the sine value of the angle degree, while the second column represents the cosine value. This new approach allows for a more meaningful and comprehensive representation of wind direction in our data.

The measured data include missing values represented as NaN. In the case of the TranAD model, we decided to replace NaN values with the mean value of the corresponding column of the dataset. On the other hand, for the Anomaly Transformer, we replaced the NaN values with zero values because the authors found this to be the best solution for other datasets.

Each model has its unique characteristics and operates differently, necessitating the use of different data normalization. For the TranAD model, we apply min-max normalization following the recommendation of the authors in their publication [3]. The equation for this normalization is,

$$x_{ij} \leftarrow \frac{x_{ij} - min(T_j)}{max(T_j) - min(T_j) + 0.0001},$$

where T_j represents one feature of the input time series and x_i represents the point that is normalized. In the case of the Anomaly Transformer model, we apply standardization feature scaling,

$$x_i \leftarrow \frac{x_i - u}{s},$$

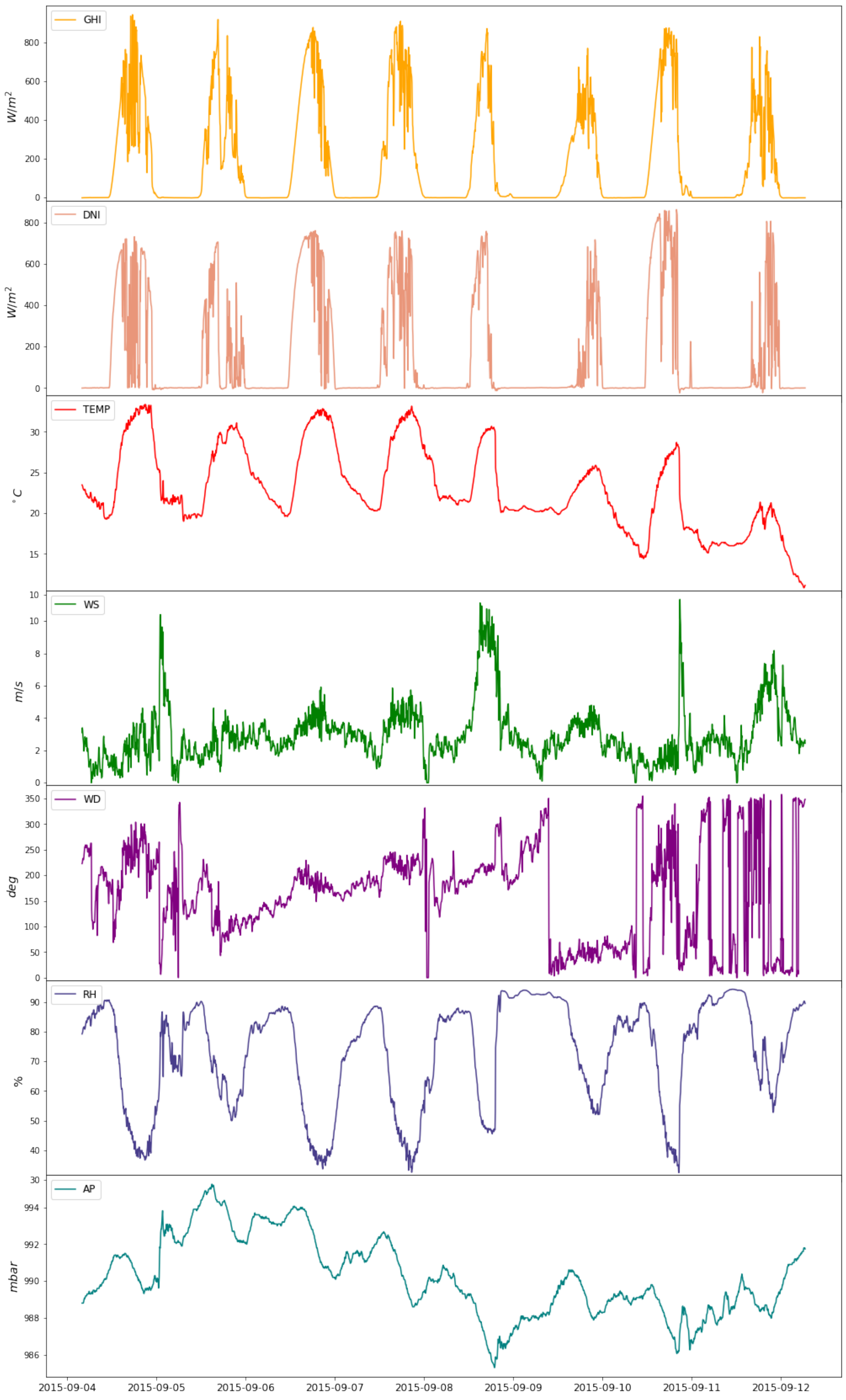


Figure 4.2: This figure illustrates a sample plot of all dimensions.

where u represents the mean and s represents the standard deviation of the one feature of the input time series.

4.3 Implementation details

The entire thesis, from a software perspective, was written in the Python programming language [23]. Specifically, for data preprocessing and experimental data analysis, we utilized the following libraries: NumPy [24], Pandas [25], and Scikit-learn [26]. For loading data into the models, we made use of the PyTorch library [27]. All plots in our thesis were generated using Matplotlib library[28].

The training of the Anomaly Transformer and TranAD models was carried out with distinctive approaches to hardware and software configurations. For training the Anomaly Transformer model, we leveraged the processing power of the NVIDIA GeForce RTX 3090 GPU. This model was configured with a window size of 400, ran for 3 epochs, used a learning rate of 0.0001, and was processed in batches of 32. Conversely, to train the TranAD model, we utilized the processing capabilities of a cluster of CPUs. The parameters for TranAD included a window size of 10, 8 epochs of training, a similar learning rate of 0.0001, but processed data in larger batches of 128.

Chapter 5

Experiment and Results

In this chapter, we present the outcomes of our comprehensive data analysis and deep learning models' training. The results from utilizing PCA on our data are presented in Sec. 5.1. In Sec. 5.2 we present our training results. Additionally, we highlight the distinct advantages of each model.

5.1 Feature Analysis

Generally, deep learning models take a lot of resources to train. Training them can last for several days and require a few gigabytes of storage. Some deep learning models that take time-series data as input cannot process raw time-series data. Their values are highly time-dependent. For that reason, they need to be preprocessed. Each time record needs to be fed into the model also with a certain number of preceding and following time records. The total amount of records is called window size. With increasing window size the amount of input data that a deep learning model gets at a time increases. This has a negative impact on the total training time, which can last for days. On the other side when the window size is too small the model gets not enough information from the time series because the values are correlated in time. This can reduce the model's ability to make good predictions. Our task is now to find the optimal window size that balances the trade-off between training time and prediction strength.

For this task, we sampled the dataset in 60-min frequency steps. This dataset has different scales, and in order to process them with PCA we need to bring them on the same scale using standardization. Next, we perform several steps to create Hankel matrices [29] for different lag numbers l. In the context of time series analysis, Hankel matrices are often used to structure the data in a way that reveals temporal relationships, making it easier to apply techniques such as PCA. When a Hankel matrix is used in time series analysis, each row of the matrix typically represents a different time-shifted version of the data. In this context, the sequence used to construct the

Hankel matrix is the time series data itself. For instance, if we have a time series of data points x_1, x_2, x_3, x_4, x_5, a Hankel matrix with a lag of $l = 3$ would look like this:

$$\begin{bmatrix} x_1 & x_2 & x_3 \\ x_2 & x_3 & x_4 \\ x_3 & x_4 & x_5 \end{bmatrix}$$

Each row in the matrix represents a "window" of l consecutive data points in the time series. This construction captures the temporal ordering of the data in a simple matrix format. In our case, we apply PCA on the lag-covariance matrix formed from the Hankel matrix for l ranging from 10 to 500 to project the data to a lower-dimensional space defined by the leading components. These leading components explain at most 95 percent of the total variance. Running this PCA for multiple lags we observe a relationship between the number of leading components and the number of lags as can be seen in the figure below. By referring to the analysis presented in Fig. 5.1, a notable

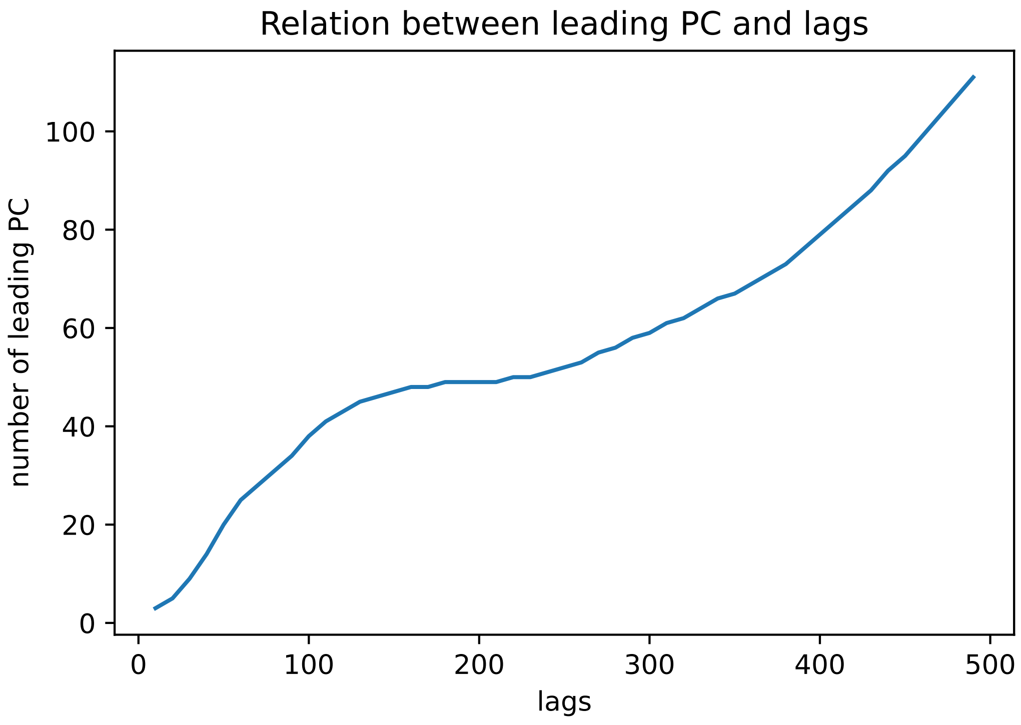


Figure 5.1: Plot between the number of leading components on the y-axis against the number of lags on the x-axis.

observation is that the function curve stabilizes or levels off around the value of 150. Consequently, we can deduce that the best approach for maximizing information while maintaining an optimal window size lies within the range of 100 to 200, in the case of 5-min. sampling frequency it represents range of 1200 to 2400. Additionally, we also analyze the spatial linear correlations between the data features.

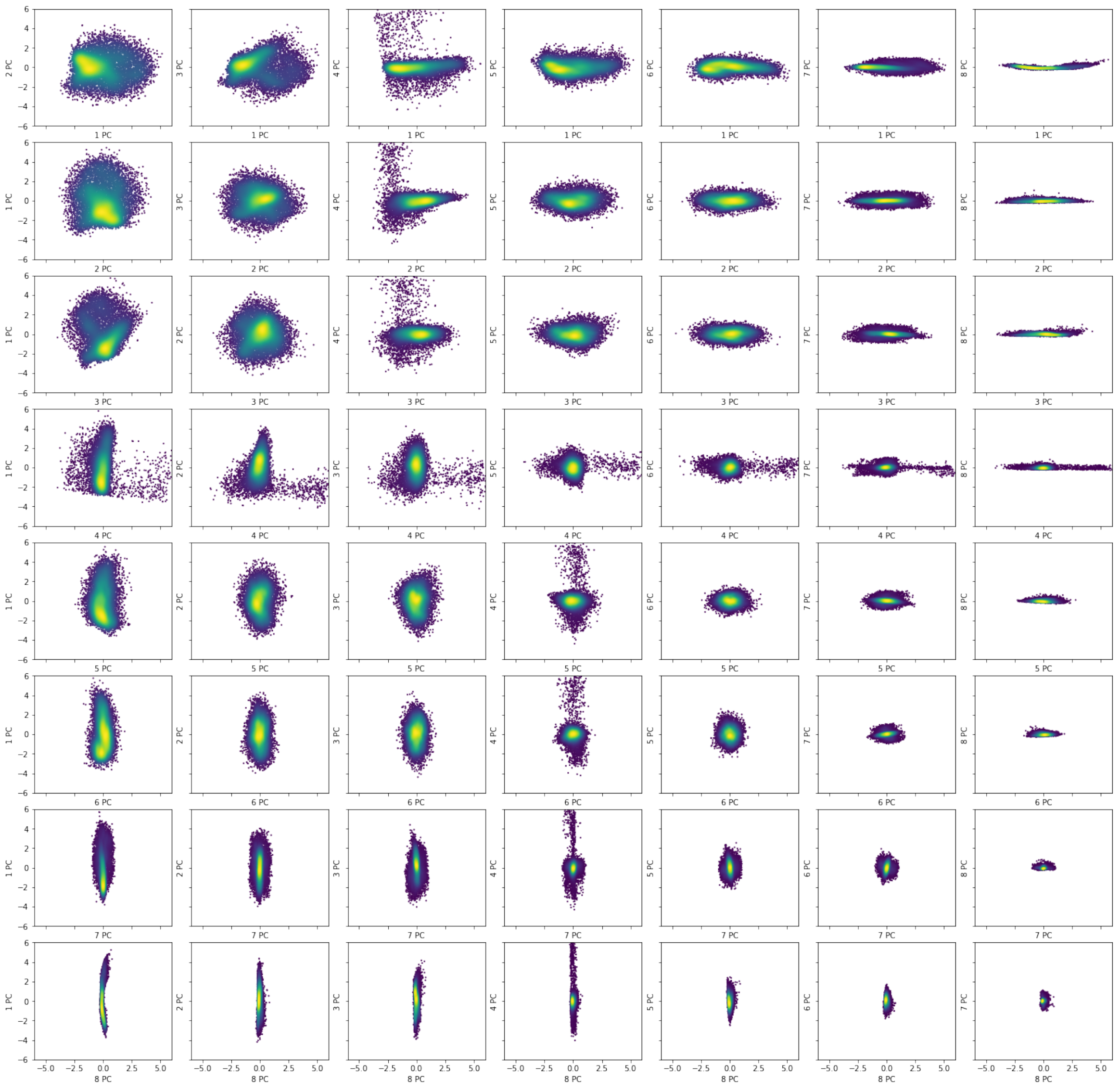


Figure 5.2: This figure shows heatmap plots of all data points of the Bonville meteo site (see Tab. A.1) projected on a 2-dimensional hyperplane formed by each pair of principal components.

Analyzing 2-dimensional projections of the datasets onto principal components in Fig. 5.2, we observe a notable dispersion of data points along the principal components. From these dispersions, we conclude that only roughly 2/3 of the features carry linear correlations between each other. The plot incorporates a heatmap technique, which visualizes data by assigning warmer colors to densely populated areas and colder colors to sparsely populated points.

5.2 Training Results

We performed several training runs of the models and present only the most notable experiment results that we were able to achieve in a restricted time and computational environment. Each experiment with the Anomaly Transformer and TranAD, processing over 16 million time records, took more than 3 days and around 6 hours of training time, respectively. This large difference is mainly addressable to the fact that TranAD utilizes a pre-trained model.

When looking at Tab. 5.1 we see a large difference between the models' performance. Anomaly Transformer seems to be more conservative in labeling data points as anomalies, which can be seen from the relatively low recall performance but decent high precision. Whereas TranAD is labeling much more time records than is truly necessary. This can be observed from precision being extremely high, and at the same time recall being extremely low. We suspect that the case of its bad performance might root in the fact that the threshold function is well calibrated for our type data. One advantage of TranAD is to pinpoint the exact dimension in which the anomaly has occurred. We also present its ability to detect anomalies for each dimension in Tab. 5.2 that confirm the overall poor performance of TranAD in our case. Since our training run for a long time it was unfeasible to run it with our suggested window size from the previous section.

Model	**Precision**	**Recall**	**F1-score**
Anomaly Transformer	0.7416	0.3567	0.4817
TranAD	0.1392	0.9615	0.2432

Table 5.1: Overall performance validation metrics for each trained model across all features, i.e., disjunction.

However, when we observe the results of Fig. 5.3, we see that TranAD demonstrates impressive predictive capabilities for GHI, DNI, sin(WD), and cos(WD) features. However, inaccuracies in predicting other features can be attributed to a range of factors, primarily because these features depend on the specific measuring location.

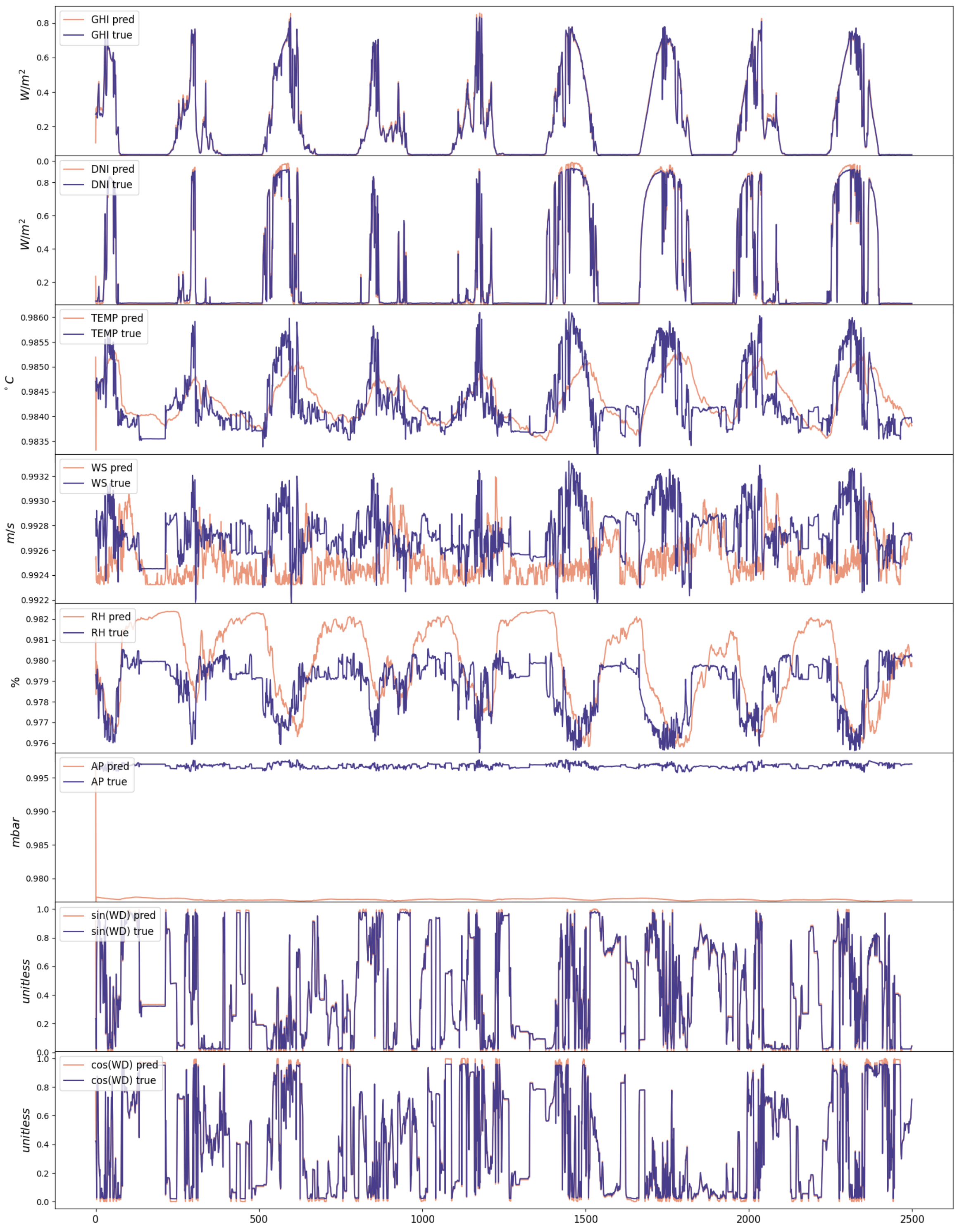


Figure 5.3: This figure represents a small sequence of the reconstructed time series returned by TranAD in blue against its original time series in orange for each dimension.

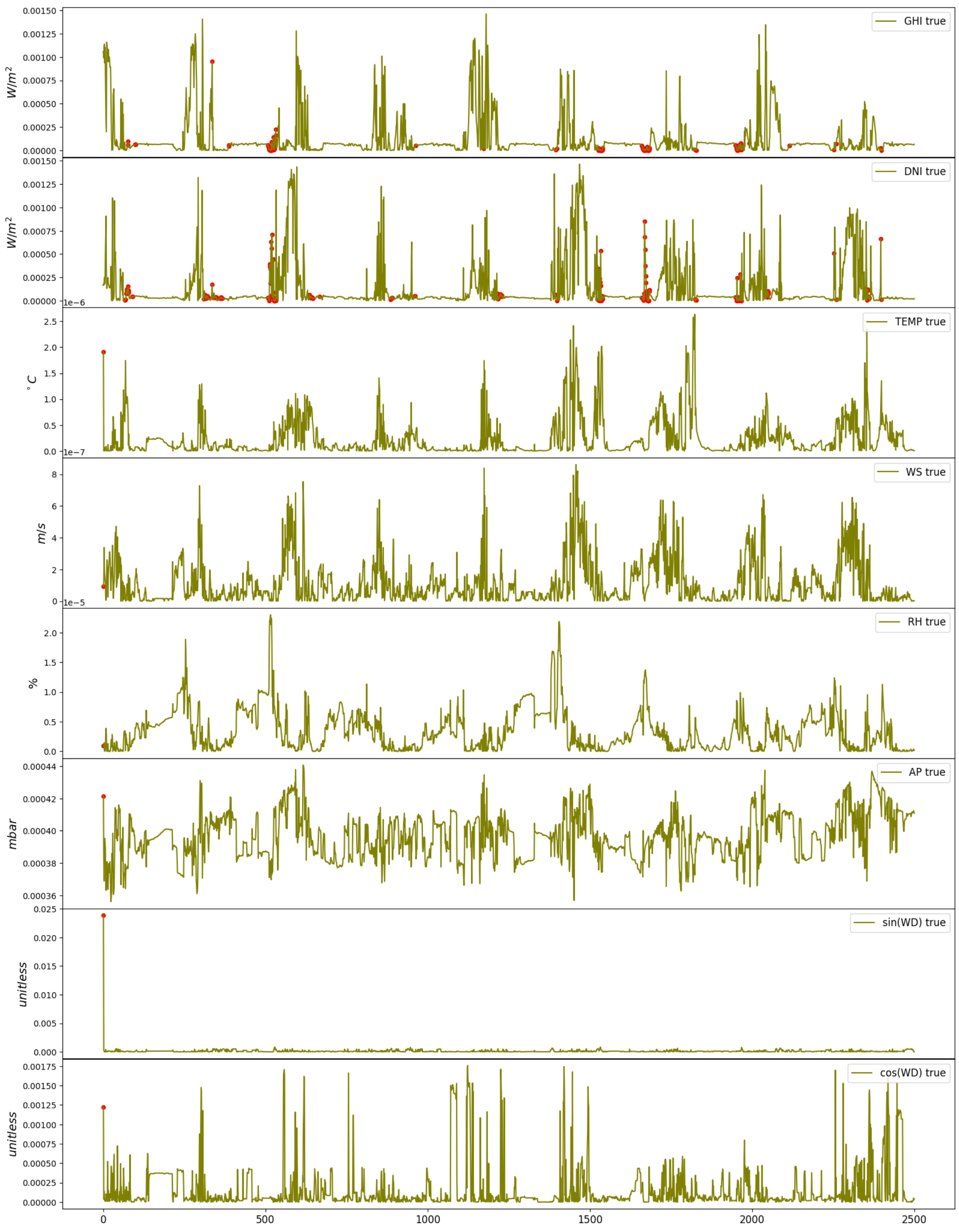


Figure 5.4: This figure represents the anomaly scores of a small time series sequence for each input dimension returned by TranAD, and the ground truth labels are shown as red dots.

Features	F1-score	Precision	Recall
GHI	0.2425	0.1659	0.4502
DNI	0.4882	0.3613	0.7524
TEMP	0.0122	0.0061	1.0000
WS	0.1031	0.0543	1.0000
RH	0.0168	0.0084	0.9971
AP	0.0167	0.0084	0.7669
sin(WD)	0.0035	0.0017	0.0806
cos(WD)	0.0043	0.0022	0.0806

Table 5.2: Performance validation metrics of the TranAD model for each individual feature of the input time series.

Conclusion

Deep learning tools possess remarkable power and complexity. However, their training process often requires substantial time, thereby limiting the extent of our observations. Notably, the TranAD model has demonstrated impressive accuracy in predicting time series for certain features. The Anomaly Transformer model has proven to be more cautious in its data labeling approach, whereas TranAD labeling exhibits a significantly higher degree of assertiveness. In light of these limitations and successes, we recommend exploring future avenues by experimenting with various hyperparameters during the training phase. This approach would enable us to delve deeper into the potential of the model and further enhance its performance.

Bibliography

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. volume 2017-December, 2017.

[2] Jiehui Xu, Haixu Wu, Jianmin Wang, and Mingsheng Long. Anomaly transformer: Time series anomaly detection with association discrepancy. 2022.

[3] Shreshth Tuli, Giuliano Casale, and Nicholas R. Jennings. Tranad. *Proceedings of the VLDB Endowment*, 15, 2022.

[4] Marcel Suri and Tomas Cebecauer. Satellite-based solar resource data: Model validation statistics versus user's uncertainty. volume 2, 2014.

[5] Tariq Muneer S. Younes R. Claywell. Quality control of solar radiation data: Present status and proposed new approaches. 30(9), 2005.

[6] Oliver Probst Roberto Chávez-Arroyo. Quality assurance of near-surface wind velocity measurements in Mexico. 22(2), 2015.

[7] F. Vejen and Norske meteorologiske institutt. *Quality Control of Meteorological Observations: Automatic Methods Used in the Nordic Countries*. Report 8/2002. Norwegian Meteorological Institute, 2002.

[8]

[9] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras and TensorFlow: concepts, tools, and techniques to build intelligent systems*. 2019.

[10] Jan J. Gerbrands. On the relationships between svd, klt and pca. *Pattern Recognition*, 14(1):375–381, 1981. 1980 Conference on Pattern Recognition.

[11] Developing an unsupervised real-time anomaly detection scheme for time series with multi-seasonality. *IEEE Transactions on Knowledge and Data Engineering*, 34, 2022.

[12] Mohsin Munir, Shoaib Ahmed Siddiqui, Andreas Dengel, and Sheraz Ahmed. Deepant: A deep learning approach for unsupervised anomaly detection in time series. *IEEE Access*, 7, 2019.

[13] Fei Tony Liu, Kai Ming Ting, and Zhi Hua Zhou. Isolation forest. *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 413–422, 2008.

[14] Sahand Hariri, Matias Carrasco Kind, and Robert J. Brunner. Extended isolation forest. *IEEE Transactions on Knowledge and Data Engineering*, 33, 2021.

[15] A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data. 2019.

[16] Ya Su, Rong Liu, Youjian Zhao, Wei Sun, Chenhao Niu, and Dan Pei. Robust anomaly detection for multivariate time series through stochastic recurrent neural network. 2019.

[17] Yifeng Gao, Jessica Lin, and Constantin Brif. Ensemble grammar induction for detecting anomalies in time series. volume 2020-March, 2020.

[18] Paul Boniol and Themis Palpanas. Series2graph: Graph-based subsequence anomaly detection for time series. *Proceedings of the VLDB Endowment*, 13, 2020.

[19] Hongchao Song, Zhuqing Jiang, Aidong Men, and Bo Yang. A hybrid semi-supervised anomaly detection model for high-dimensional data. *Computational Intelligence and Neuroscience*, 2017, 2017.

[20] Paul Boniol, John Paparrizos, Themis Palpanas, and Michael J. Franklin. Sand: Streaming subsequence anomaly detection. volume 14, 2021.

[21] Zheng Li, Yue Zhao, Nicola Botta, Cezar Ionescu, and Xiyang Hu. Copod: Copula-based outlier detection. volume 2020-November, 2020.

[22] Jordan Hochenbaum, Owen S. Vallis, and Arun Kejariwal. Automatic anomaly detection in the cloud via statistical learning. *CoRR*, abs/1704.07706, 2017.

[23] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[24] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren

Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[25] The pandas development team. pandas-dev/pandas: Pandas, February 2020.

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

[27] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[28] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[29] Wikipedia. Hankel matrix, 2023. Accessed: 2023-02-21.

[30] du Clou S. van Niekerk J.L. Gauche P. Leonard C. Mouzouris M.J. Meyer A.J. van der Westhuizen N. van Dyk E.E. Brooks, M.J. and F. Vorster. Sauran: A new resource for solar radiometric data in southern africa. 2015.

Appendix A

Meteorological sites

Site name	**Country**	**Source**	**No. records**	**Use case**
Alamosa	Colorado, USA	SUFRAD	201820	validation
Tri An	Vietnam	ESMAP	227630	validation
Bloemfontein	South Africa	SAURAN	361540	training
Bondville	Illinois, USA	SUFRAD	2203742	training
Boulder	Colorado, USA	SUFRAD	2311893	training
Desert Rock	Nevada, USA	SUFRAD	2201732	training
Fort peck	Montana, USA	SUFRAD	2308075	training
Goodwin Creek	Mississippi, USA	SUFRAD	2287549	training
Graaff-Reinet	South Africa	SAURAN	223001	training
Rock Springs	Pennsylvania, USA	SUFRAD	2273111	training
Sioux Falls	South Dakota, USA	SUFRAD	1842197	training
Vanrhynsdrorp	South Africa	SAURAN	296429	training

Table A.1: Table of all meteo sites used as data sources in our work. It contains the name of the meteo site its location by country, the data provider (SUFRAD - NOAA Earth System Research Laboratory, 1995: Surface Radiation Budget (SURFRAD) Network Observations. Datasets: Alamosa, BON, TBL, DRA, FPK, GWN, Rock Springs, SXF. NOAA National Centers for Environmental Information. Access date: 15.02.2023, SAURAN - The Southern African Universities Radiometric Network (SAURAN)[30], ESMAP - Data obtained from the "World Bank via ENERGYDATA.info, under a project funded by the Energy Sector Management Assistance Program (ESMAP)), the number of time records and the use case for our work.