Alma Mater Studiorum · University of Bologna

Department of Physics and Astronomy
Department of Biological, Geological and Environmental Sciences

Master Degree in Science of Climate

MACHINE LEARNING METHODS FOR SEASONAL FORECASTS OF CLIMATE VARIABLES

Supervisor:
Prof. Antonio Navarra

Submitted by:
Nicolò Landi

Academic year 2023/2024

Abstract

The purpose of this thesis work is to give an overview of the most prominent and successful machine learning models in the field of climate forecasts, to exhibit some of the most important methods used in their implementation, and to showcase their performance through the use of a few simple example models. We first go through the inner workings of the LSTM and transformer models, highlighting their strengths and shortcomings. We then go on to the data preprocessing phase, which in our case included the EOF decomposition of our input fields, particularly the tropical sea surface temperature and surface air temperature. We also list some practical methods useful during the training process. Finally, we present the performance of our example LSTM and transformer models.

Contents

Introduction

When dealing with the climate system and in particular its evolution, we are mostly dealing with physical laws that can be described by partial differential equations, like the fundamental Navier-Stokes equations. Since these equations cannot be solved analytically, one needs to use approximations to solve the forecasting problem. From the first onset of the computer, weather prediction models started to obtain valid solutions with numerical methods, such as the finite differences method [1]. From those earliest years much development has occurred, with the increase of computational power allowing for extremely complex models, with a myriad of equations representing all kinds of physical, biological, and even social processes that occur on Earth. Nevertheless, these models are still fundamentally based on the same numerical methods used in the first forecasts.

In recent years, with the advent of big data, efficient supercomputers with Graphics Processing Units (GPU), and scientific interest in emerging new methods [2], a new way of solving the partial differential equations governing the climate system has emerged through the use of deep learning, leveraging the universal approximation theorem [3], which establishes that given a family of neural networks, for each function that we want to approximate, there exists a sequence of neural networks from that family such that the sequence converges to that function. In general, deep neural networks could approximate any high-dimensional function, provided sufficient training data are supplied. Although many methods are known from the 1960s and have been examined in detail in many studies since then, recent years, with unprecedented increases in data volume and computer power, are seen as the golden era for artificial intelligence and machine learning. For climate scientists, the most interesting group of techniques was found to be supervised learning, with many thematic articles detailing various such methods and their classification [4][5][6].

Today, a plethora of various neural network architectures have been developed, each with its own strengths and caveats. Among these, a broad type of network is the feed-forward neural network (FNNs), which are characterized by the uni-dimensional flow of information between their layers. This means that the information runs through the model only in one direction, forward through the hidden and output nodes without ever having information going back through cycles or loops [7]. This type of network has broad applications, but lacks a fundamental feature that makes it difficult to operate our case study: Since these types of models can only process one input at a time and produce an output, it is harder for them to capture patterns that arise in sequential data and that drive this type of statistical forecast of a time series evolution [8]. Nevertheless, some methods have been developed to make use of these types of networks in time series forecasting, by using particular setups for the input data and hybridizing with other methods [9][10].

An alternative to feedforward networks are recurrent neural networks (RNNs), which have a bi-directional flow as they contain loops in their architecture that make it so that a certain hidden state can be influenced by both current input and the previous hidden state, essentially creating a form of memory [11]. This type of mechanism renders this type of model very well suited to processing text, speech, and time series [12].

A more modern evolution of RNN is the Long Short-term memory model (LSTM) [13]. This type of architecture almost totally took over the standard RNNs by managing to deal with their most glaring shortcoming, the vanishing gradient problem [14]. LSTMs have found great success in the field of climate forecasting, especially on monthly to seasonal time scales [15][16].

Recently, another particularly attractive machine learning architecture has emerged, particularly when dealing with sequential inputs such as time series data: the transformer model [17]. Transformers have achieved superior performances in many tasks in natural language processing and computer vision, which also triggered great interest in the time series community. Among the multiple advantages of Transformers, the ability to capture long-range dependencies and interactions is especially attractive for time series modeling, leading to exciting progress in various time series applications such as the field of climate forecasting [18].

In this work, we take a look at the details behind the functioning of these neural networks, with a focus on best practices to use both in general and in particular when dealing with climate data. We also show examples of simple LSTM and transformer models and their performance when dealing with the forecast of basic climate variables such as sea surface temperature and surface air temperature. In particular, we will use the statistical method of EOF decomposition to prepare our data.

Chapter 1

Data and Methods

1.1 LSTM Architecture

1.1.1 Recurrent Neural Networks

A Recurrent Neural Network (RNN) is a type of neural network that is able to detect patterns in a sequence of data. We are interested in numerical time series, but these sequences could also represent handwriting or genomes [8]. The main difference between Recurrent Neural Networks and Feedforward Neural Networks, or Multi-Layer Perceptrons (MLPs), lies in the way information gets passed through the network. RNNs have cycles, and they transmit information back into themselves, allowing them to extend the functionality of Feedforward Neural Networks to also include not only the current input, but also previous ones. At a high level this difference can be visualized in figure 1.1. We can describe this process through the use of mathematical notation [19]. For that, we denote the hidden state and the input at time step t respectively as $\mathbf{H}_t$ and $\mathbf{X}_t$. Further, we use a weight matrix $\mathbf{W}_{xh}$, a hidden-state-to-hidden-state matrix $\mathbf{W}_{hh}$, a bias $\mathbf{b}_h$ and an activation function ϕ. Putting them together we get the equations for the hidden variables and the output variable.

$$\mathbf{H}_t = \phi_h(\mathbf{X}_t\mathbf{W}_{xh} + \mathbf{H}_{t-1}\mathbf{W}_{hh} + \mathbf{b}_h) \tag{1.1}$$

$$\mathbf{O}_t = \phi_o(\mathbf{H}_t\mathbf{W}_{ho} + \mathbf{b}_o) \tag{1.2}$$

Since $\mathbf{H}_t$ recursively includes $\mathbf{H}_{t-1}$ the RNN includes traces of all the hidden states that preceded $\mathbf{H}_{t-1}$ as well as $\mathbf{H}_{t-1}$ itself.

If we compare these equations with one made with similar notation for Feedforward Neural Networks we can clearly notice the differences between the two as described earlier.

$$\mathbf{H} = \phi_h(\mathbf{X}\mathbf{W}_{xh} + \mathbf{b}_h) \tag{1.3}$$

$$\mathbf{O} = \phi_o(\mathbf{H}\mathbf{W}_{ho} + \mathbf{b}_o) \tag{1.4}$$

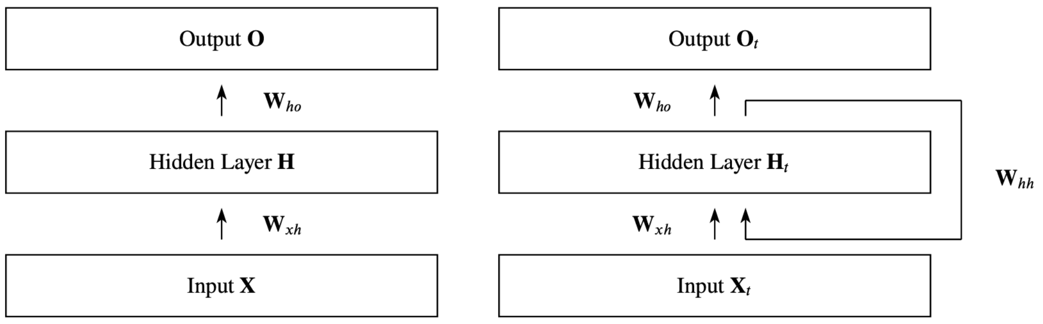


Figure 1.1: Visualisation of differences between Feedforward NNs (left) and Recurrent NNs (right). The option of having multiple hidden layers is aggregated into the **H** block [8].

1.1.2 LSTM

One problem with standard RNNs consists in not being able to handle long sequences correctly. This is due to the fact that back-propagated error signals tend to shrink or grow at every time step, so that over many cycles the error typically vanishes or blows up [20]. With a vanishing error learning takes an extremely long amount of time, or may not even work at all, while blown-up errors lead to oscillating weights and unstable learning. A Long Short-Term Memory (LSTM) [13] is a method that addresses this vanishing gradient problem, being explicitly designed to retain long term dependencies. LSTMs, like standard RNNs, have a chain of repeating modules of neural network, but instead of having a single neural network layer there are four, interacting in a particular way (figure 1.2).

The core innovation of LSTMs is the introduction of the cell state [21]. It runs through the entire sequence with only some minor linear interaction, allowing the network to preserve information over many time steps since it is easy for the information to flow along unchanged. The LSTM has the ability to remove or add information to the cell state through structures called gates, which regulate what information can pass through.

The self connection of a basic LSTM network would have a fixed weight set to '1' to preserve the cell state over time. Unfortunately, the cell states $\mathbf{C}_t$ tend to grow linearly with the progression of a time series during a continuous input stream [20]. This way the cell loses its memorizing capability, and tends to function like an ordinary RNN network neuron. To address this problem we can attach a forget gate to the self connection [22]. Forget gates can learn to reset the state of the cell when stored information is no longer needed, basically deciding how much information needs to be retained or thrown away from the previous time steps. It consists of a sigmoid layer that takes as input $\mathbf{h}_{t-1}$ and $\mathbf{x}_t$ and outputs a number between 0 and 1 to multiply to each number in the cell state $\mathbf{C}_{t-1}$. This way a number multiplied by 0 would be completely canceled or "forgotten".

$$\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f) \tag{1.5}$$

The next step consists in deciding what new information is going to be stored in the cell state. To achieve this, first a sigmoid layer called the input gate decides which values to update. Next, a *tanh* layer generates new values $\tilde{\mathbf{C}}_t$ that may be added to the cell state.

$$\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i) \tag{1.6}$$

$$\tilde{\mathbf{C}}_t = tanh(\mathbf{W}_C[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_C) \tag{1.7}$$

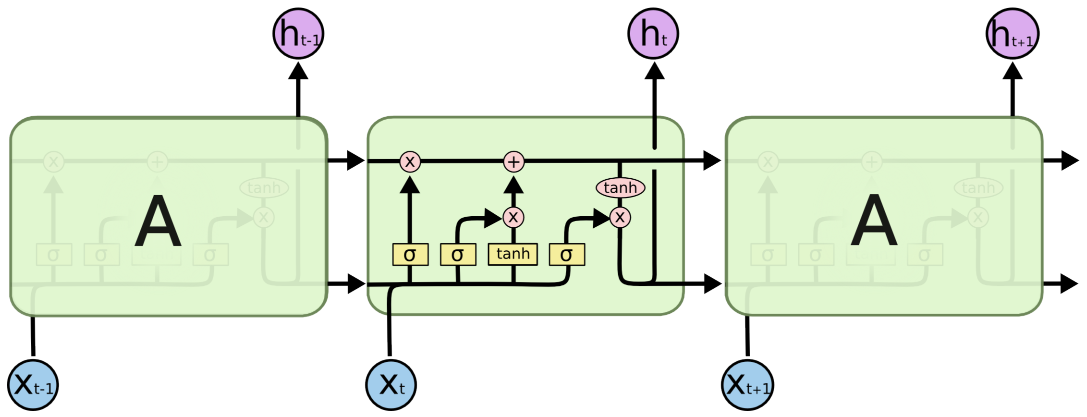


Figure 1.2: Visualization of the repeating module of the standard LSTM architecture, containing the four interacting layers and the connections to previous and successive modules [21].

Now we can update the cell state by multiplying the old state by $\mathbf{f}_t$, forgetting what was decided earlier, and adding the new values $\tilde{\mathbf{C}}_t$ scaled by the values that the input gate $\mathbf{i}_t$ decided to update.

$$\mathbf{C}_t = \mathbf{f}_t * \mathbf{C}_{t-1} + \mathbf{i}_t * \tilde{\mathbf{C}}_t \tag{1.8}$$

Finally, we need the output $\mathbf{h}_t$ of the whole module. We push the cell state through a *tanh* layer to get values between -1 and 1, and then multiply it by one last sigmoid function, the output gate.

$$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o) \tag{1.9}$$

$$\mathbf{h}_t = \mathbf{o}_t * tanh(\mathbf{C}_t) \tag{1.10}$$

We can initialize the bias weights of input and output gates with negative values, and the weights of the forget gate with positive ones. This way at the beginning of training the forget gate activation will be close to '1', preventing the memory cell from starting to forget right away, before even learning anything.

1.2 Transformer Architecture

A transformer is a deep neural network that makes use a self attention mechanism to comprehend relationships within sequential data. Interest in this model first sparked in the field of Natural Language Processing (NLP), but as for other neural network models capable of sequential data processing such as LSTM, the application to time series prediction is a natural evolution. Transformers excel in handling long dependencies, and are more tailored to do so even than modern versions of RNNs such as the LSTM. As we discussed LSTMs already managed to solve underlying problems of the more general RNNs, especially for the gradient problems that relegated previous models to working with short sequences only. However, LSTMs still struggle with processing long sequences, which hinders the extraction of the actual long term patterns [23]. Transformers are a type of deep neural network (DNN) that aims to offer a solution to some limitations of sequence-to-sequence architectures. In particular, they tackle the problem with long sequences and the sequential processing of inputs, that hinders the possibility of parallel training of the networks. Different from traditional recurrent methods, they manage to learn from an entire segment of a sequence through the use of the multi-head self-attention. Moreover, they end up being faster than other counterparts when dealing with large inputs since they can work in parallel, managing to utilize the full computational potential of Graphical Processing Units (GPUs).

Since the conception of the original transformer in 2017 [17] many different models have been developed from it, with some of them drastically changing its base structure. This means that the performance of transformer based models and the task they are suitable to tackle can vary significantly depending on the specific architecture employed. Nevertheless, the key components of transformer models is the self-attention mechanism, which is essential to their functionality.

1.2.1 Attention Mechanism

The basis of the transformer model consists in finding associations or relationships between different segments within the same input sequence. Let $\{\mathbf{x}_i\}_{i=1}^{n}$ be a set of data points of length n in a single sequence. The self-attention operation is basically the weighted dot product of the input vectors $\mathbf{x}_i$ with each others [24] . We can actually divide this operation in two steps. The first one consists in computing the normalized dot product between all pairs of data points within the input sequence. The normalization is achieved through a softmax operator, which makes it so that given a certain set of numbers, the outputs will sum up to the unity value. These normalized correlations w_{ij} are computed between the input $\mathbf{x}_i$ and all other input vectors $\mathbf{x}_j$ with $j = 1, ..., n$:

$$w_{ij} = \text{softmax}(\mathbf{x}_i^T \mathbf{x}_j) = \frac{e^{\mathbf{x}_i^T \mathbf{x}_j}}{\sum_k e^{\mathbf{x}_i^T \mathbf{x}_k}}, \tag{1.11}$$

with $\sum_{j=1}^n w_{ij} = 1$ and $1 \leq i, j \leq n$. Then we compute a new representation $\mathbf{z}_i$ for each input segment $\mathbf{x}_i$ by doing a weighted sum of all inputs:

$$\mathbf{z}_i = \sum_{j=1}^n w_{ij} \mathbf{x}_j = \sum_{j=1}^n \text{softmax}(\mathbf{x}_i^T \mathbf{x}_j) \mathbf{x}_j. \tag{1.12}$$

In actuality the self attention operation is not applied directly onto the inputs $\{\mathbf{x}_i\}_{i=1}^n$, but to three different vectors obtained as linearly weighted vectors from the inputs. These vectors are the query $\mathbf{q}$, the key $\mathbf{k}$ and the value $\mathbf{v}$. For a certain input they can be computed as:

$$\mathbf{q}_i = W_q \mathbf{x}_i, \quad \mathbf{k}_i = W_k \mathbf{x}_i, \quad \mathbf{v}_i = W_v \mathbf{x}_i, \tag{1.13}$$

where W_q, W_k and W_v represent learnable weight matrices. The outputs $\{\mathbf{z}_i\}_{i=1}^n$ are computed similarly to equation 1.12, but with these three new vectors substituting each instance of the input $\mathbf{x}$. A visualization of the self attention process can be seen on the right of figure 1.3. Moreover, since the softmax operator is sensitive to large values, we scale the attention weights by the square root of the size of the vectors d:

$$\mathbf{z}_i = \sum_{j=1}^n \text{softmax}(\frac{\mathbf{q}_i^T \mathbf{k}_j}{\sqrt{d}}) \mathbf{v}_j. \tag{1.14}$$

Since the input data can contain many different levels of correlation between the

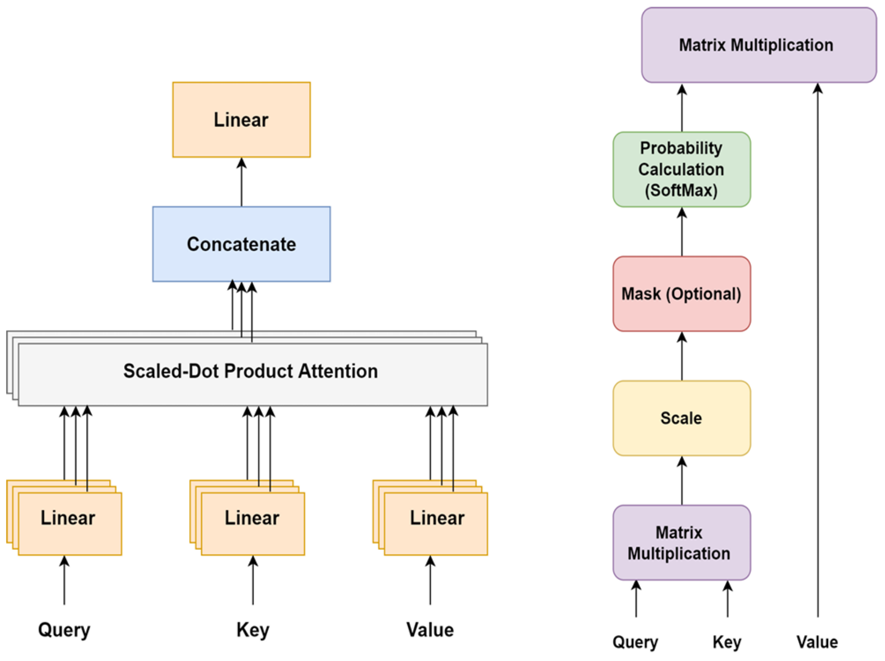


Figure 1.3: On the left the general structure of the Multi-head attention operation, with several attention layers running in parallel. On the right the detailed structure of a single Scaled Dot-Product Attention [23].

various data points, transformers benefit from multiple self-attention heads. These different heads operate on the same input in parallel and utilize different learnable weights W_q, W_k and W_v. In this way the model can manage to extract several distinct levels of correlation between the input data. The operations that are involved in this ”multi-head self-attention” [23] are schematized in figure 1.3. We basically take a fixed number of queries, keys and values and run them all in parallel through the scaled dot product operations. Then, we concatenate all the outputs z from all the heads, and lastly we linearly combine them using a final learnable weight matrix W_o. We remark that at the end of this process the dimension of the input X an the output Z are the same.

1.2.2 Decoder-Encoder Mechanism

The self-attention mechanism is the fundamental building block of the transformer architecture, but now we get to include it in a bigger structure as in figure 1.4.

The first block is called the encoder. It consists of a multi-head self-attention layer followed by a feed-forward layer, linked by residual connections and normalization layers. When training deep neural networks, residual connections and normalization layers are widely used since they help with training stabilization [25] and lead to faster convergence of the model [26].

The second block is the decoder, which contains similar operations and layers to the encoder. However, a decoder block receives two inputs. One input is the target sequence, but ”shifted right” by one position, with the purpose of masking the future prediction at each time step and avoid future information to spill into the prediction process. The second input is just the output of the previous encoder. Inside the decoder there are three layers. They consist in a multi-head self-attention, an encoder-decoder attention layer, and a feed-forward layer. The novelty with respect to an encoder is found in the second layer, which is a multi-head attention layer where the key and value are produced from the output of the last encoder and the query is created from the output of the preceding self-attention layer. Just like for the encoder, residual connection and normalization layers are included.

1.2.3 Positional Embedding

Since we are dealing with sequential data and the self-attention processes do not include information on the order of the input data in the given sequence, we need to somehow include that information. Transformers make use of positional encoding (PE) to be able to retain positional information for the inputs and meanwhile being able to process the modified input in parallel. This is in contrast to how RNNs work, since they intrinsically include sequential information by proceeding one step at the time in order along the sequence. Positional encoding works by calculating n PE vectors that do not include any learnable parameters, and adding them to the inputs $\{\mathbf{x}_i\}_{i=1}^{n}$ before going

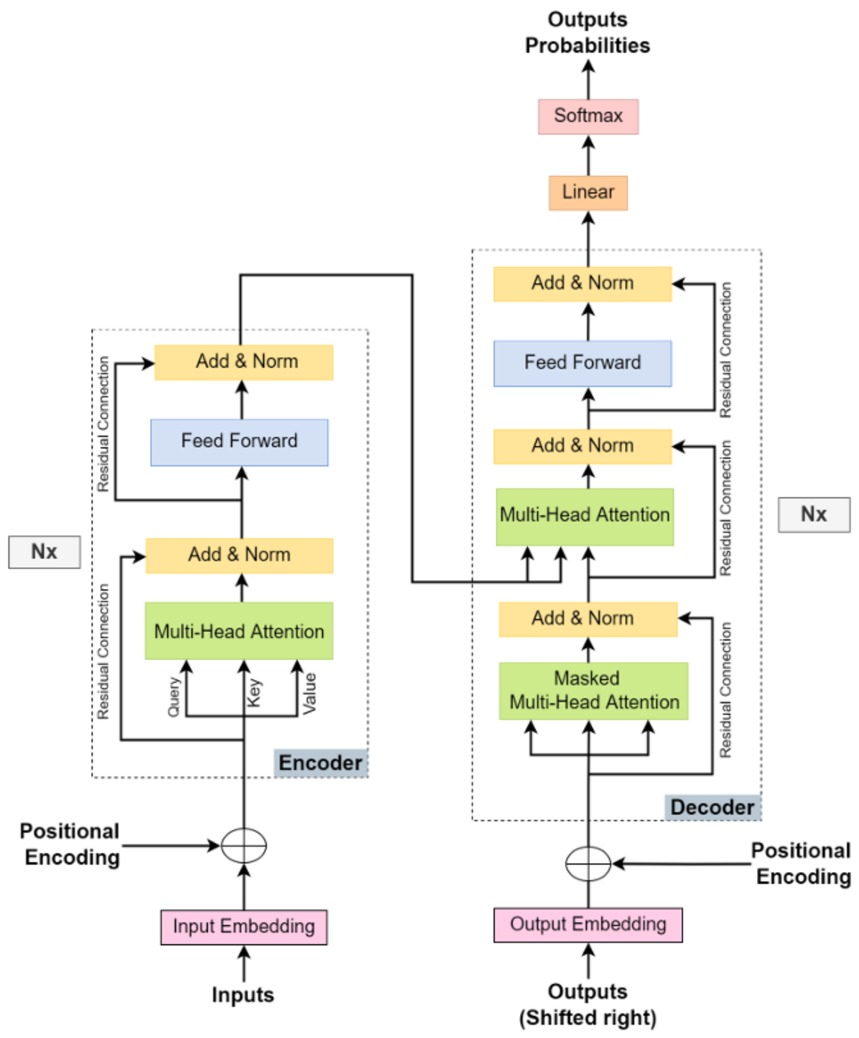


Figure 1.4: Transformer architecture, comprising the encoder and decoder blocks, with their respective internal structures and connections [23].

through the rest of the model as seen in figure 1.4. For the original transformer model [17] a sinusoidal function was used to compute the PE vectors.

1.3 Data Preprocessing

1.3.1 ERA5 Dataset

The data that was used throughout this work comes from the ERA5 reanalysis dataset from ECMWF . A reanalysis is a collection of past climatic variables created by merging simulated model states with historical observational data, using data assimilation to create a physically consistent new dataset. Reanalyses in Climate science give us extremely valuable data sources on the past states of components of the Earth system, like the atmosphere and ocean [27]. Reanalyses find themselves in many applications in Earth system sciences, from gauging progress in modelling and assimilation capabilities, to obtaining state-of-the-art climatologies to evaluate forecast-error anomalies [28].

The spatial grid resolution of the ERA5 global dataset is of 31 km for the horizontal coordinates, but it is available on regular latitude-longitude grids at $0.25°$ x $0.25°$ resolution. Vertically it has 137 pressure levels. Regarding the temporal dimension, it has a resolution of 1 hour, from 1940 to the present day. In this work we make use of only the variable regarding sea surface temperature (SST), so we do not utilize the vertical levels. We also only use monthly mean data instead of the full hourly case, since we focus on seasonal forecasts. However, we do use the full horizontal resolution of $0.25°$ x $0.25°$ of latitude-longitude.

1.3.2 EOF Analysis

The datasets we used use as input of our machine learning models are not the raw variables taken the from ERA5 reanalysis. We first apply Empirical orthogonal function analysis (EOF), which is a statistical tool that is used to identify spatial patterns of climate variability and how they change with time. In particular each EOF represents a spatial pattern that is orthogonal to all the others, with each one of them having an associated time series describing the evolution of that particular pattern through time [29].

To obtain the EOFs, we first need the de-trended anomaly of the data. To obtain the anomaly we take the climatology of our variable, which is the mean of monthly values over the full time period, and subtract it from the full field. This way we are excluding from our analysis seasonal variability. To de-trend the data we compute the linear trend of our variable over the time period and then subtract its effect. Moreover, we can divide by the standard deviation of each point so that they all have unit variance. When we have the de-trended anomaly of our variable we compress the 3D field into a 2-dimensional one. We do this by collapsing the two spatial dimensions into one and obtaining a matrix Z in which each column represents all the points at a single time step and each row is the whole time series for a single point in space. Then ZZ^T will be a covariance matrix for which we will solve an eigenvalue problem. Since this is a symmetrical matrix we get an orthogonal basis of eigenvectors which are our EOFs, while the associated eigenvalues represent the variance explained by each EOF. Each EOF represents a spatial mode of variability for the variable in question, with no information on how it evolves through time. If we sort the eigenvalues in descending order we can arrange the EOFs from most variance explained to least, so that the first EOF will always be the most important mode of variability. If we then project the full field onto this orthogonal basis we obtain the Principal Components (PCs), which represent the time series associated with each EOF and describe how the amplitude of each mode changes over time (figure 1.5).

This statistical method is first and foremost extremely useful in identifying the most important spatial patterns in climate variability, since by just looking at the first few EOFs, which contain most of the variance, one can get a sense of how our variable is varying in space. Moreover, it is a phenomenal method of data compression, since most of the variance will be explained by the first N EOFs, where N will be much smaller than the number of time steps, e.g. N≈20 if we are talking about an atmospheric variable. So to get the full field you just need to store N 2-dimensional spatial maps and N time series, instead of the full 3-dimensional field. Then, one can always recreate the full field by:

$$\text{Field}(x,t) = \sum_{i=1}^{N} EOF(x)_i * PC(t)_i \tag{1.15}$$

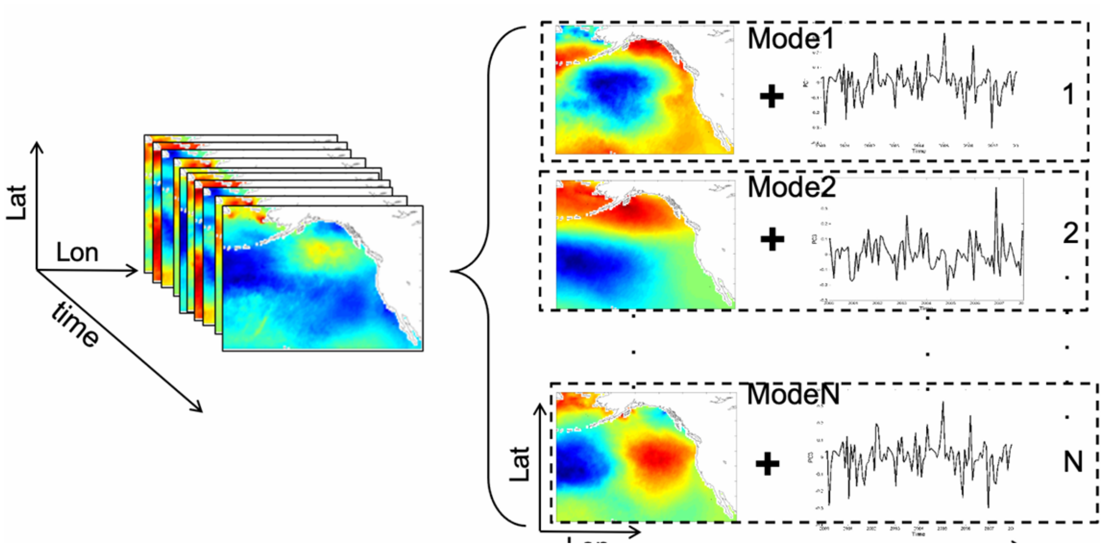


Figure 1.5: Visualization of EOF decomposition process. The data starts as a 3-dimensional field, and is compressed into a series of N 2-dimensional EOFs with their associated 1-dimensional time series [30].

1.3.3 Data Splitting and Sequencing

After having extracted EOFs and PCs from our variables, we can prepare the data to go into our machine learning models. Our aim is to use these models to make time-series forecasts, so to predict the value of our climate variable in the future. Since we care about time evolution we can leave behind the actual EOFs and take through our models only the respective time series, the PCs. So our models will be trained via supervised learning only on the PCs, and will learn to predict the time steps following a given input sequence. By excluding all the spatial data, the inputs of the models result much smaller in size, greatly increasing computational efficiency without compromising accuracy, since at the end we can reconstruct the full field by simply combining the forecast time series with the EOFs computed before by using equation 1.15.

If we were using all the obtained modes the reconstruction would be exact, but we actually wouldn't gain anything in terms of computational efficiency. The strength of this method lies in the fact that we can retain a much smaller number of EOFs, while still explaining most of the variance. In particular, the number of retained modes will correspond to the number of features of our datasets.

We then split the data into the training, validation, and testing sets, with a typical distribution of 80%/10%/10%. Note that to avoid future contamination of our data for the forecasting, the EOF decomposition was obtained by excluding the testing time frame from the calculations. The data in each set is then organized into sequences, where the sequence length is one of the hyperparameters of our model. Specifically, we retain source sequences that will serve as the input to our models and associated target sequences that refer to the following months, which the model will forecast. Both the number of features and the sequence length may vary between input and output sequences. We finally divide the data into batches, allowing the model to process the data bit by bit and reducing memory usage, while also allowing to leverage parallel processing. Using smaller batches is also a key in stochastic gradient descent, since computing the gradient on each batch introduces some noise into the learning process, which can help the model generalize better by avoiding local minima.

1.4 Training Process

After the data preprocessing phase is complete we can initialize our model and start with the training process. The weights and biases are initialized with a simple uniform distribution centered around 0, and limited to small numbers to avoid exploding gradient problems. Then we start to feed our input data to the network. For each input sequence in the training dataset the model will generate a prediction for the time step following the last one of the sequence. A loss value will typically be computed by deriving the Mean Square Error value (MSE) between the output time step and the target one, though other loss functions can also be used. Then the models computes the gradients of the loss with respect to the weights and biases using the technique of backpropagation through time (BPTT). This method propagates the gradients backward through the sequence of time steps.

1.4.1 Optimizer

Once the gradients are computed we get to actually update the parameters, both weights and biases, of our model. A simple yet effective update method, or optimizer method, is the Stochastic Gradient Descent (SGD). According to SGD we can update the parameters θ at each iteration by just following the gradient scaled by the step size or learning rate α, until we get to a minimum of the Loss function f, as can be seen in figure 1.6.

$$\theta_{t+1} = \theta_t - \alpha \nabla f(\theta_t) \tag{1.16}$$

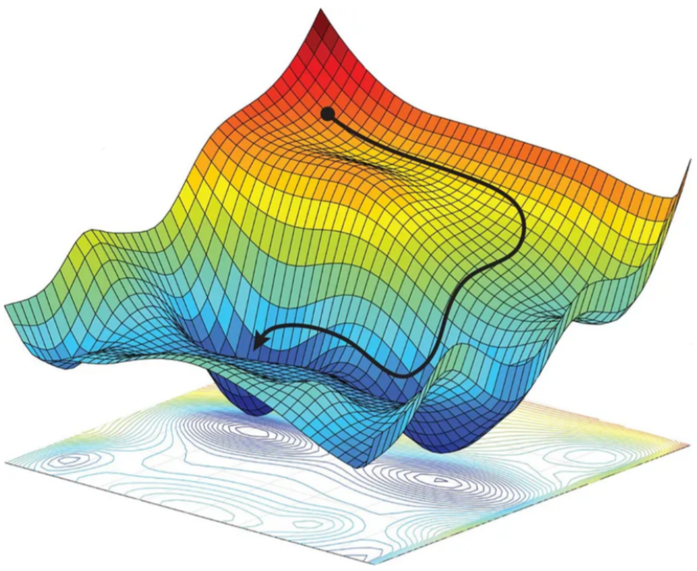


Figure 1.6: Example of a Stochastic Gradient Descent path for a 2-d parameters vector. The black arrow indicates the values of the parameters that get gradually updated during the training process. The process stops when they reach the values that correspond to the minimum of the loss function [31].

However, when working with deep learning models Stochastic Gradient Descent may not be the optimal choice. Another popular algorithm in the field of deep learning is Adam [32]. SGD has only a single learning rate value for all weight updates, and this learning rate does not change during training. Adam, on the other hand, computes individual adaptive learning rates for all the various parameters, and they are adapted as the learning progress unfolds [33]. In particular, the parameter learning rates are updated based on the values of both the average first moment of the gradient, or the mean, and the average of the second moment of the gradient, the uncentered variance.

$$\theta_{t+1} = \theta_t - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{1.17}$$

For a standard Adam algorithm (appendix 1), a weight decay is implemented with an L2 regularization term. In general, weight decay is a regularization method used to prevent overfitting and to improve the generalization of models. Its aim is to penalize excessively large weights, and it does so by adding a penalty term to the loss function, effectively reducing the magnitude of weights over time. With an L2 regularization term the new loss function would be like equation 1.18, where $||\theta||^2$ is the L2 norm of the weights and λ is the factor of the weight decay.

$$L_{new}(\theta) = L(\theta) + \lambda \cdot ||\theta||^2 \tag{1.18}$$

The L2 regularization used by Adam is intrinsically tied to the learning rate. It has been demonstrated that by decoupling the weight decay from the gradient update and applying the decay directly to the weights substantially improves Adam generalization performance [34]. We can notice that the update equation (1.19) for this new algorithm is essentially the same as equation 1.17 with the addition of the weight decay factor. This new algorithm takes the name of AdamW (appendix 2), and is generally preferred to standard Adam when weight decay would be used.

$$\theta_{t+1} = \theta_t - \alpha \cdot (\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \lambda\theta_t) \tag{1.19}$$

1.4.2 Scheduler and Gradient Clipping

We also take some other precautions during the training process to avoid stability issues, like using a learning rate scheduler and gradient clipping.

A learning rate scheduler is a method that generally makes it so the learning rate can be gradually lowered during the training process. This has the aim to make the model make large updates at the beginning of learning, when the parameters are far from their optimized values, and consequently make smaller updates later, allowing for more stability and fine-tuning in the last steps of reaching the optimal solution [35]. In particular, we use a method from pytorch called "ReduceLROnPlateau" [36]. This method applies a step decay, so it reduces the learning rate by a given factor, when a monitored value stops improving for a specified number of epochs. In our case the monitored value is the training loss.

Gradient clipping is another way of directly addressing the problem with exploding gradients, in addition to weight decay which also helps. The objective function, especially for very deep networks or RNNs, may often contain big non-linearities. These non-linearities will make it so that some regions in the parameter space have very high derivatives. When during training the parameters get close to these regions, following the gradient during the update phase may bring the new values of the parameters very far, possibly even losing most of the learning progress that was done up to that point. When applying gradient clipping, the error derivative the error derivative during back-propagation is tied to a certain threshold, and this clipped variant is used to update the weights. This method helps the model during gradient descent when in the vicinity of extremely steep cliffs, limiting the model's reaction to the extreme gradient and avoiding being flung away from the solution like in figure 1.7. In particular, we use the pytorch method "clip_grad_norm_" that limits the gradient's norm to a set value. If the norm surpasses this value, then the gradients will be proportionally rescaled until their norm is within the specified maximum value.

1.4.3 Dropout

As we already discussed, large neural networks with relatively small training datasets may result in overfitting, for example by learning the statistical noise in the training data, resulting in poor performance when encountering new data. In theory, the best way to regularize a model would be to average the predictions of all neural networks with

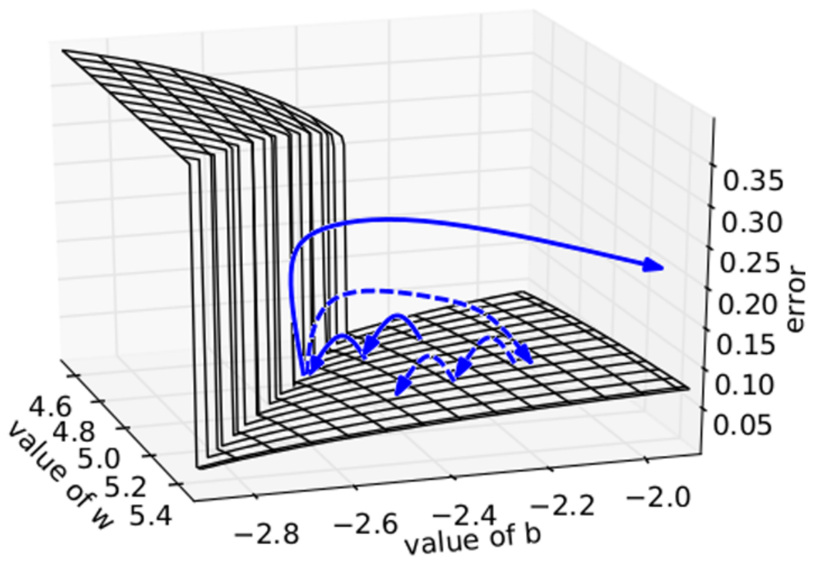


Figure 1.7: The error surface of a single hidden unit of a recurrent network is plotted, the highlight being the presence of a very steep wall. The solid blue line represents a standard trajectory that the gradient descent may follow after encountering the cliff. The dashed line shows a possible trajectory that would be followed if gradient clipping is applied [37].

all the the possible settings of the parameters, weighting each setting by its posterior probability computed on the training data [38]. This would be obviously possible only with unlimited computing power. An alternative is to use not an unlimited number of models but an ensemble. This approach can work, but it still requires a very large amount of computational power to store and train all the models in the ensemble if they are large.

This is where dropout comes in. Dropout can be seen as an approximation this ensemble training process. It operates by temporarily removing, or ”dropping out”, a random unit from the network, along with all its connections both incoming and outgoing (figure 1.8). In practice, using dropout during a training process of a neural network is like sampling a thinned version of it, consisting in all the units that survived dropout [38]. For each training case a new thinned network is sampled, and the training happens only on this sampled one. So training a neural network with dropout can be seen as training a collection of thinned networks that extensively share weights, but where each thinned network learns very rarely or even never. Once training is complete, to get a prediction during the testing phase the most direct way would be to average the predictions from all the thinned models. Since this is not computationally feasible we use a simple approximation of the averaging method. We use a single neural network without dropout, whose weights are scaled down versions of the weights obtained during the training phase. The weights get simply multiplied by the probability of being retained of their unit. This way the expected output of a hidden unit is the same as the actual output during testing.

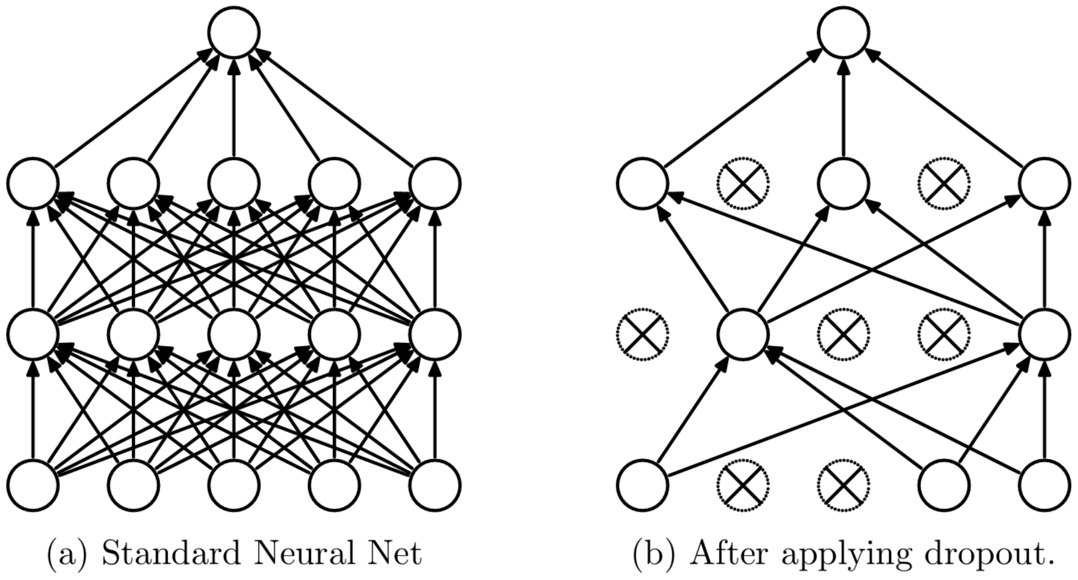


Figure 1.8: A schematic visualization of the process of dropout on a neural network. On the left (a) we see a standard network with all its neurons always connected. Meanwhile on the right (b) we see an example of what happens to the network after dropout is applied, with several connections being cut, effectively removing those neurons from the network for that step [38].

1.4.4 Validation Phase

After a full epoch of training is passed, and the mean loss is computed for all the samples in the training dataset, we go on to the validation step. This step is basically composed by a stripped down version of the training process. The model goes through all the sequences in the validation dataset, computing the prediction for the time step following the sequence, and then calculating the loss versus the target value. During this phase we turn off all gradient computations to speed up the process. The gradients are not needed since now we are no longer updating the parameters of the model. We also save the mean validation loss for each epoch.

We can compare the values of the loss for the training and for the validation steps to check model performance throughout the whole process. If the model is working as intended we of course expect the training loss to go down with each epoch until it reaches a low value, since in theory with time the model should be able to learn to represent the data that was feed into it for the supervised learning process. However, since the model could not learn from the validation data, the comparison between the two losses gives us already a measure on how well the model is able to generalize throughout the epochs. The training process may go on until a maximum number of epochs is reached, but in reality we can make it so it never reaches that point by using an ”Early stopping” function. This method simply checks the current mean validation loss, and if this value stays the same or even begins to grow for a certain fixed number of epochs, then it stops the training process. In this way, we can avoid cases in which the model learns better and better how to represent the training data, signified by a continuing decrease in the training loss, but gets worse at generalizing to other data. This is one of the various ways in which we limit overfitting.

Chapter 2

Results

We now show the outputs of the models and the skill of their forecasts. The main focus will be put on the sea surface temperature (SST) variable, as it is the most important variable that alone can help us study and predict climate patterns at seasonal time scales. This is because sea surface temperature and generally oceanic variables vary with longer periods with respect to atmospheric ones. In particular, in the former case the timescale is on the order of months, while in the latter it is more on the order of weeks. Moreover, sea surface temperature influences the air temperature and in general the whole circulation pattern above it, with seasonal variations in air temperature often being just the result of interactions with the ocean.

We also put a focus on the tropical region, as extending the forecast to the whole globe with a method such as this would yield very imprecise results, and it was not the focus of the experiment. We even take a look at the forecast for the tropical Pacific region only, to show a case with less noisy data.

2.1 Obtaining the Forecast

2.1.1 Greedy Inference

Once the model is trained we can test its skill for seasonal forecasts. The model was trained to minimize the loss for the prediction of a single month in the future, but that does not mean we cannot utilize it for forecasting longer times. We apply the concept of "Greedy inference". It consists in firstly letting the model run on an input sequence and predict the following time step. Then we update the input sequence by dropping the first value and adding the predicted one at the end of the sequence. This way we have a sequence of the same length but shifted one time step in the future, where the last time step is the forecast one, and all the others are still from the input data. We can theoretically apply this step as many times as we want, each time dropping an input time step and adding a predicted one, but of course the accuracy of the prediction will drop as we keep going, as the errors on each prediction will progressively stack up. We will choose a certain forecast window to test our data, and since we are interested in seasonal forecasts the window may be a few months long, but generally less than one year.

2.1.2 Recomposition and Testing

During the data pre-processing phase we applied EOF decomposition. This means that throughout the training and prediction phases the data that went through the model were just the time series, or the Principal Components. This basically means the models did not learn anything about the spatial patterns of the variables in question directly. To obtain the full field of our forecast variable we still need to combine the output sequences with their respective EOFs that contain the spatial information. The re-composition is simply computed through equation 1.15, with the number of retained features influencing the accuracy of the reconstruction.

To test the accuracy of the forecast we need to compare it with the test data. We make the comparison by using two somewhat exchangeable metrics. The first one consists in computing the root mean square error (RMSE) of the residuals between the forecast values and the observed ones across all spatial points. The second one is the correlation between the two fields. Both of these metrics are not averaged over the prediction time window, but are instead computed separately for each time step, and we store these result to compare the evolution of the skill of the forecast for longer prediction times.

2.2 LSTM

The learnable parameters of the model get optimized and assigned naturally during the training process of a neural network. However there are plenty of so-called hyperparameters, many of which were already presented in chapter 1, that have to be manually assigned by the user before the model enters training and are then fixed. This means we have to train and test the models with a variety of choices for these hyperparameters, and then manually decide what are the optimal values for each one of them. This optimization process is called tuning. We now illustrate the outputs and skills of our LSTM model for a variety of values for some hyperparameters.

2.2.1 Tropical SST

We now take a look at the evolution of the loss value during the training process and at the skill of the forecasts for the tropical sea surface temperature, with various choices of some hyperparameters of the model. In figure 2.1 we can see how the model performs with different values of the number of features. In our case the number of features corresponds to the number of EOFs retained during the data pre-processing step as discussed in section 1.3.2. Switching at the second plot, we can see the skill of the prediction with respect to the full field, represented by both the RMSE and the correlation. The skill of the model is compared to the skill of the simplest forecast, the persistence, that consists in taking the value of the last time step as the prediction for the next one, and then all the future ones. On the third picture we see the skill of the forecasts not against the full field but only on the reconstruction with the respective number of features.

Choosing a number of features equal to 10, various other iterations of the model went through the tuning process focusing on a variety of other hyperparameters. In figures 2.2 and 2.3 we see some examples dealing with the complexity of the model. In the first case we show the tuning process of the dimension of the hidden layer of the model, and in the second case the number of layers.

2.2.2 Pacific SST

The previous section dealt with the sea surface temperature of the tropics of the whole world, while here we focus only on the tropical Pacific. This region still captures El Niño, the most important climate mode and the one that appears in the first few EOFs, and discards other regions that tend to add a lot of noise. In figures 2.4 and 2.5 we can see the loss and skill values pertaining to the number of features and dimension of the hidden state for a model trained on the Pacific region only.

2.2.3 Tropical T2M

We now take a look at how the model performs with a different variable, the air temperature at 2 meters above the surface. It is meant to represent surface air temperature, while avoiding a part of the interference with the ground due to turbulence and heat fluxes. In figures 2.6, 2.7 and 2.8 we can see the outcome of the tuning processes of some hyperparameters as seen in the previous sections. They show respectively the tuning of the number of sequences, the dimension of the hidden state and the number of layers.

2.2.4 T2M over Sea

In this section we show the output of the model with a modified version of the temperature 2 meters above the surface. In particular, we masked the field over land, effectively making this variable represents the air temperature only above the ocean, discarding the rest. In figure 2.9 we can see the loss and skill of the model when trained to forecast this variable, with a varying number of features

2.3 Transformer

The transformer model also went through similar testing scenarios and tuning process. In figures 2.10, 2.11 and 2.12 we can see the loss and skill of the model for the standard utilized dataset, so for a forecast of the sea surface temperature of the whole tropics. The model complexity of transformers depends on different parameters to those of the LSTM. In these plots we show the performance of the model with different values for the number of heads of the Multi-head attention mechanism, and the hidden dimension, which is the dimension of both the input and output vectors of the encoder and decoder. The hidden dimension divided by the number of heads represents the dimension of each single head of attention.

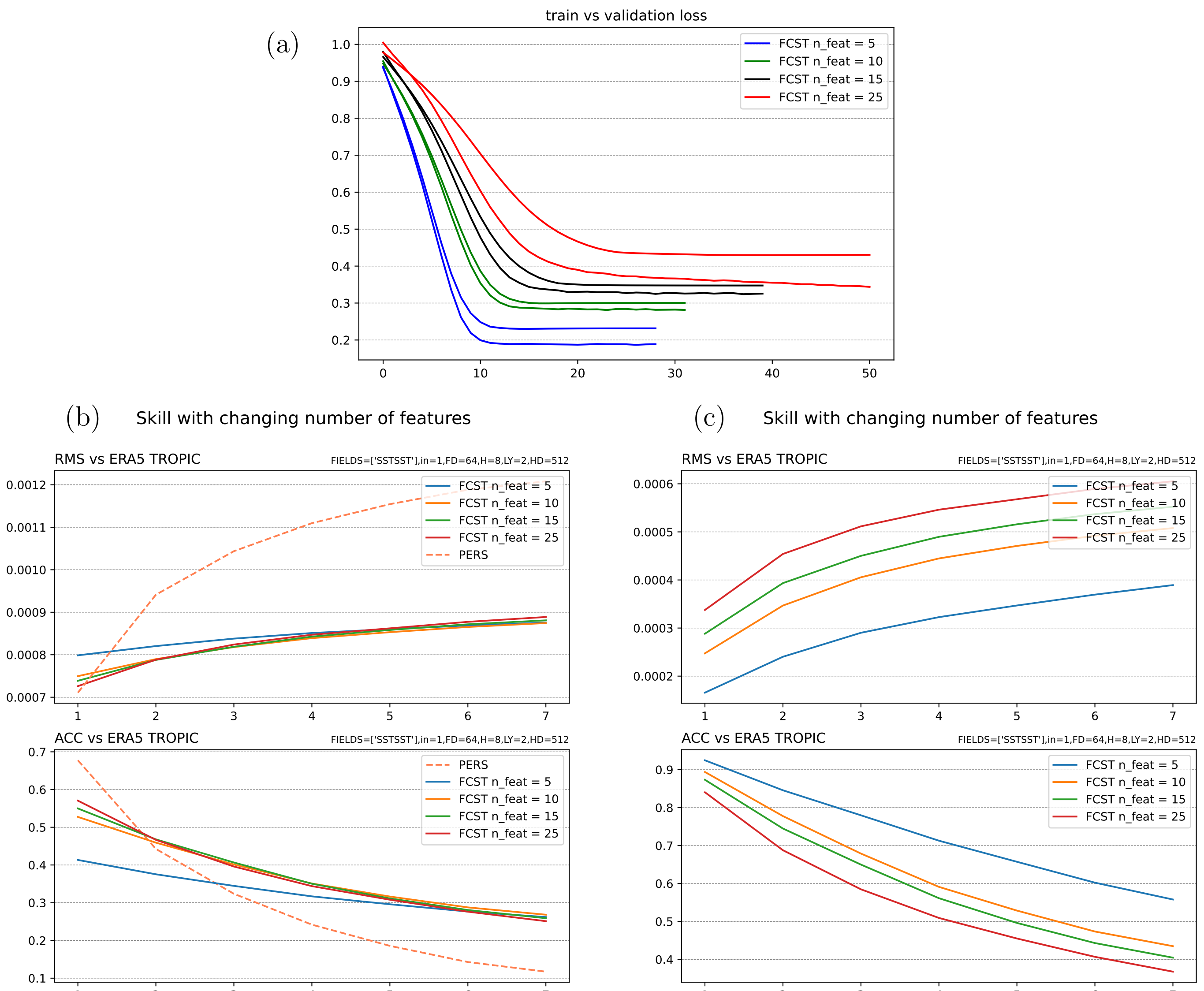
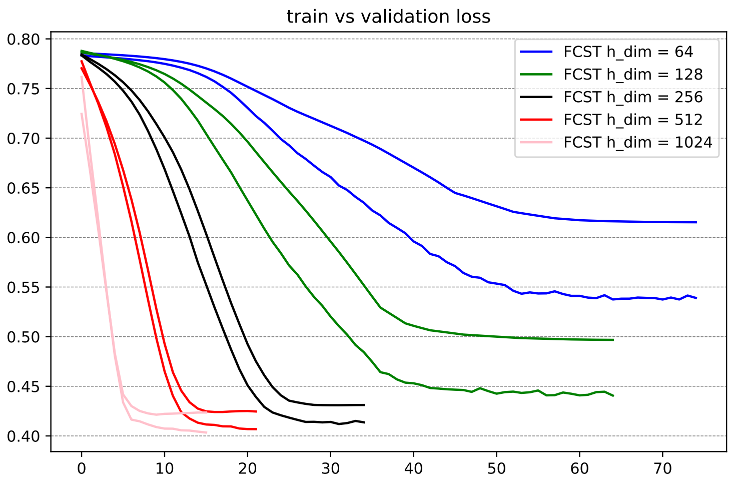


Figure 2.1: Loss values and forecast skill of the LSTM for the tropical sea surface temperature. (a) The plot shows the training and validation loss values, with each color representing a different number of features and the top line for each color representing the validation loss, while the bottom line represents training loss; the x axis shows the epoch reached during training. (b) The plots show the skill of the forecast, compared against the full original field. For the solid lines each color represents a different number of features, while the dashed line shows the skill of the persistence. The plot on the top shows the RMSE, while the one on the bottom the correlation. The x axis represents the number of time steps in the future with respect to the input sequence for the given forecast. (c) The plots again show the skill of the forecast, but it is compared against the field reconstructed with the respective number of features instead of the full field.

(a)



(b)

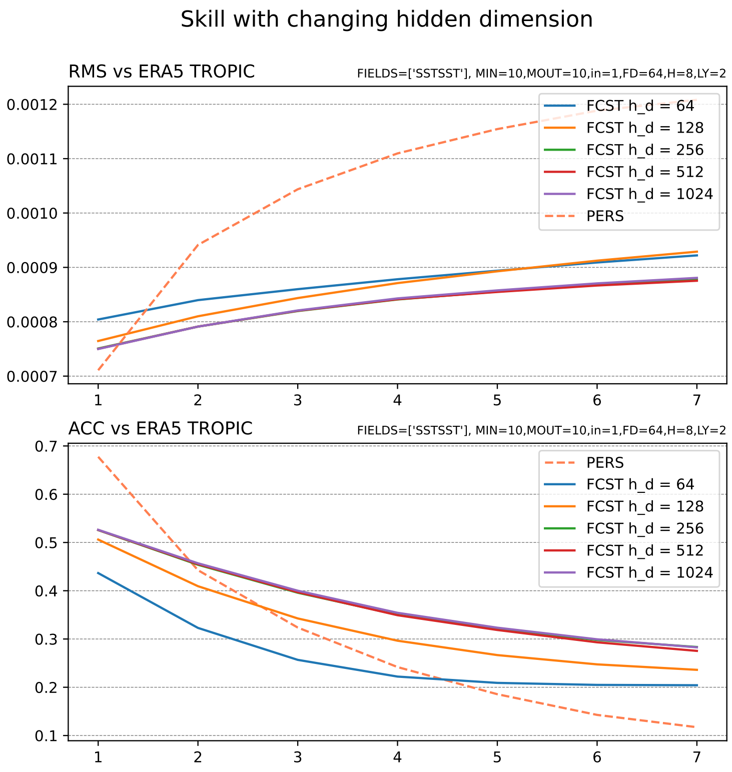
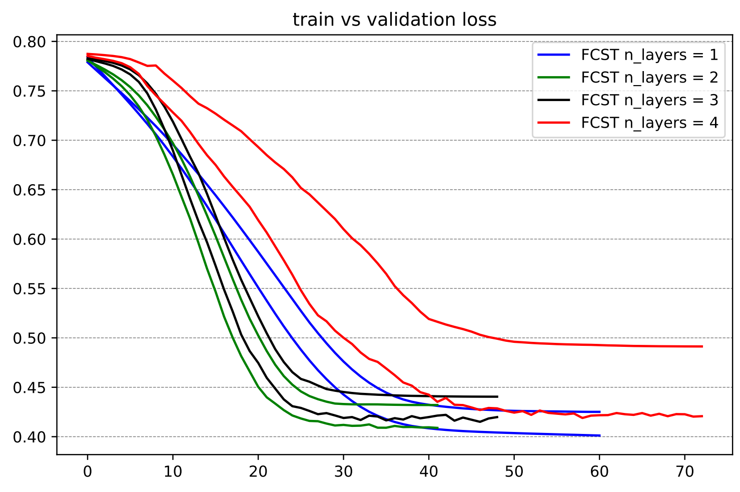


Figure 2.2: Loss values and forecast skill of the LSTM for the tropical sea surface temperature. (a) The plot shows the training and validation loss values, with each color representing a different hidden dimension and the top line for each color representing the validation loss, while the bottom line represents training loss; the x axis shows the epoch reached during training. (b) The plots show the skill of the forecast, compared against the full original field. For the solid lines each color represents a different hidden dimension, while the dashed line shows the skill of the persistence. The plot on the top shows the RMSE, while the one on the bottom the correlation. The x axis represents the number of time steps in the future with respect to the input sequence for the given forecast.

(a)



(b)

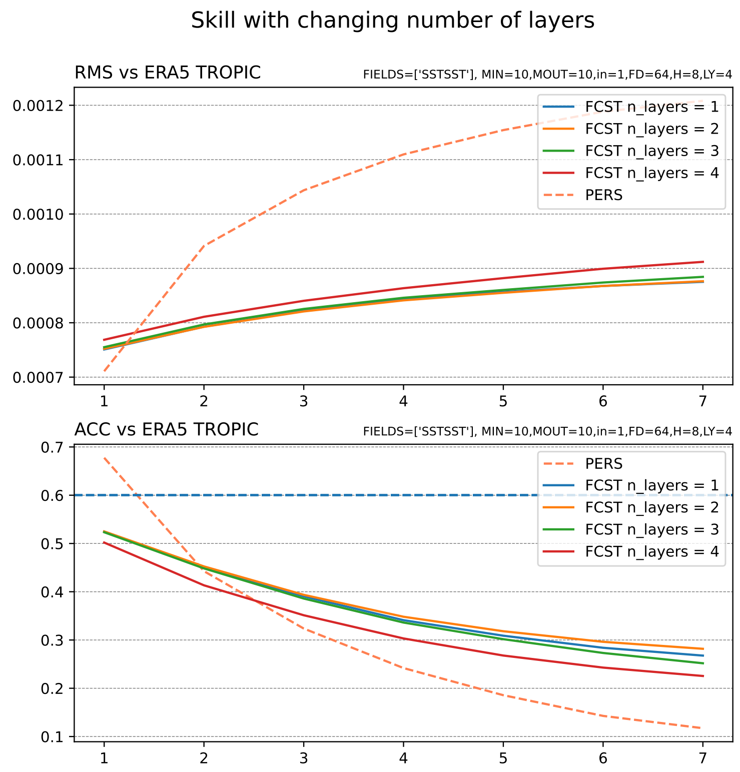
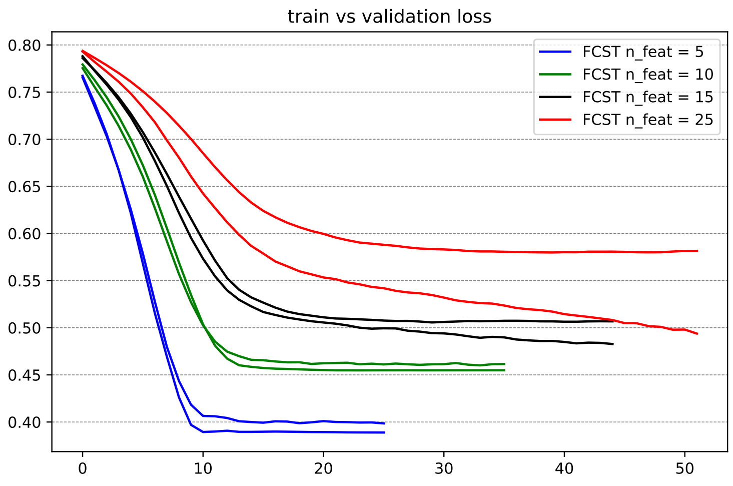


Figure 2.3: Loss values and forecast skill of the LSTM for the tropical sea surface temperature. (a) The plot shows the training and validation loss values, with each color representing a different number of layers and the top line for each color representing the validation loss, while the bottom line represents training loss; the x axis shows the epoch reached during training. (b) The plots show the skill of the forecast, compared against the full original field. For the solid lines each color represents a different number of layers, while the dashed line shows the skill of the persistence. The plot on the top shows the RMSE, while the one on the bottom the correlation. The x axis represents the number of time steps in the future with respect to the input sequence for the given forecast.

(a)



(b)

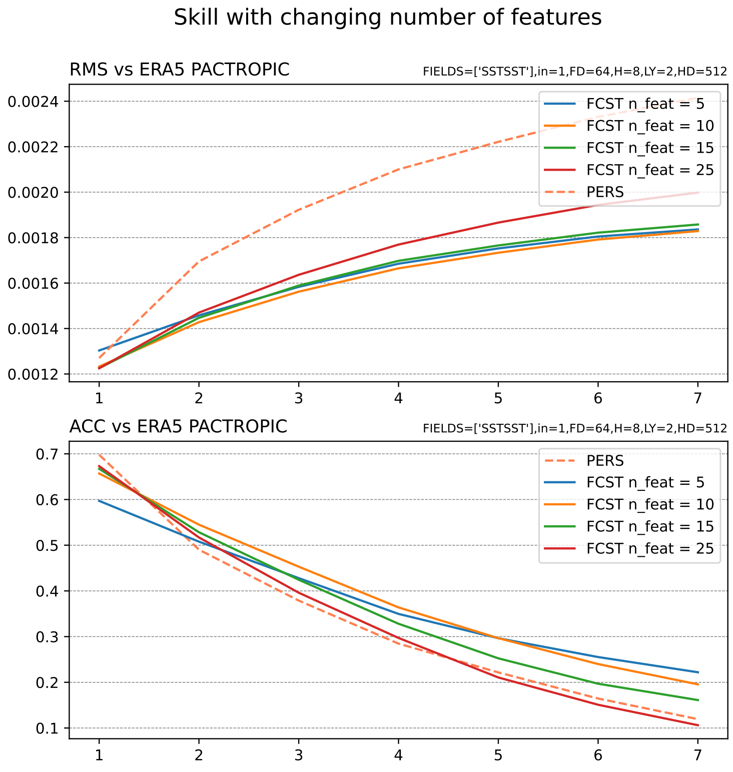
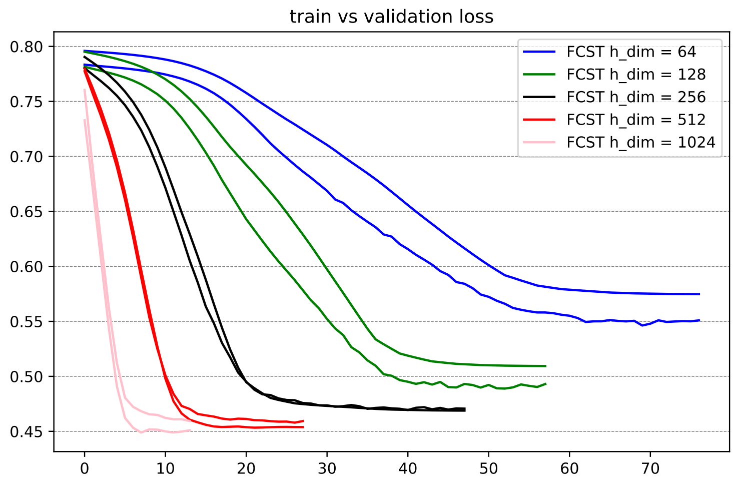


Figure 2.4: Loss values and forecast skill of the LSTM for the tropical Pacific sea surface temperature. (a) The plot shows the training and validation loss values, with each color representing a different number of features and the top line for each color representing the validation loss, while the bottom line represents training loss; the x axis shows the epoch reached during training. (b) The plots show the skill of the forecast, compared against the full original field. For the solid lines each color represents a different number of features, while the dashed line shows the skill of the persistence. The plot on the top shows the RMSE, while the one on the bottom the correlation. The x axis represents the number of time steps in the future with respect to the input sequence for the given forecast.

(a)



(b)

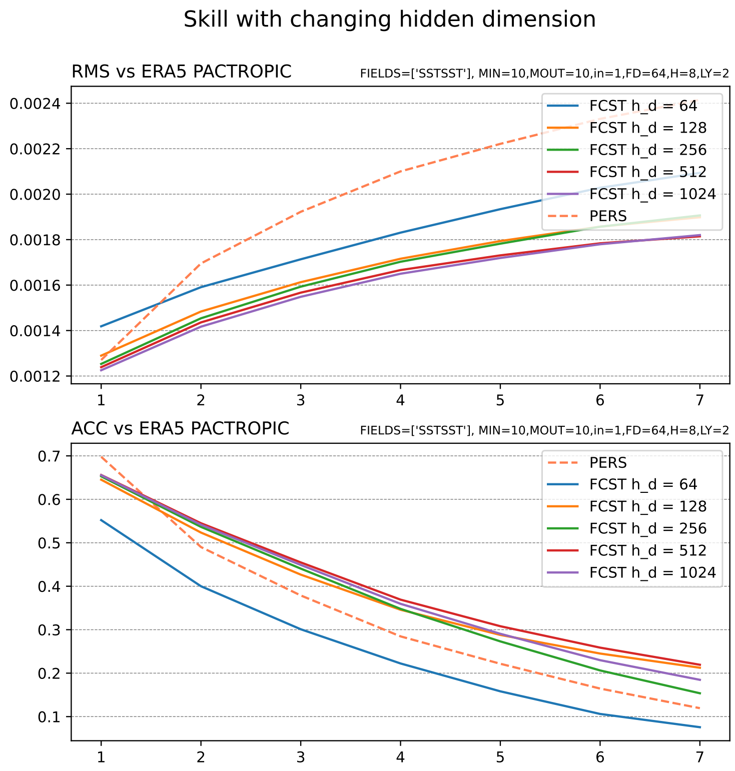
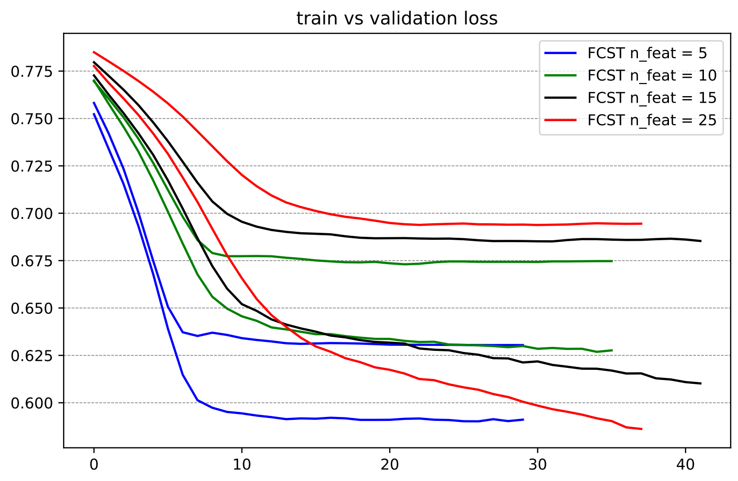


Figure 2.5: Loss values and forecast skill of the LSTM for the tropical Pacific sea surface temperature. (a) The plot shows the training and validation loss values, with each color representing a different hidden dimension and the top line for each color representing the validation loss, while the bottom line represents training loss; the x axis shows the epoch reached during training. (b) The plots show the skill of the forecast, compared against the full original field. For the solid lines each color represents a different hidden dimension, while the dashed line shows the skill of the persistence. The plot on the top shows the RMSE, while the one on the bottom the correlation. The x axis represents the number of time steps in the future with respect to the input sequence for the given forecast.

(a)



(b)

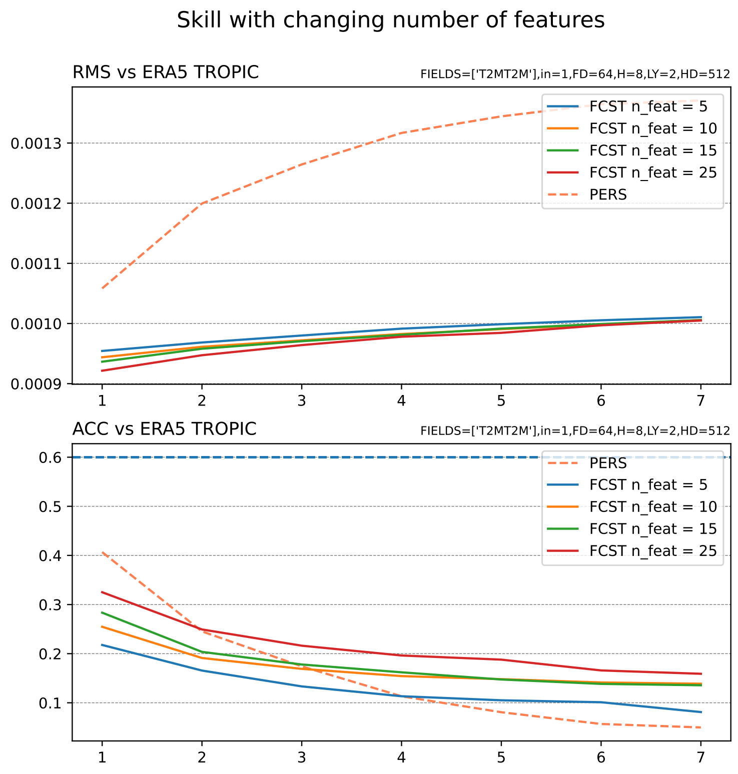
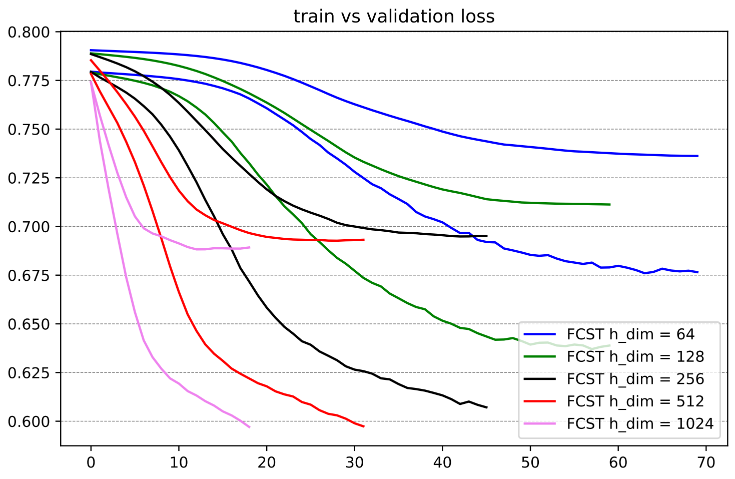


Figure 2.6: Loss values and forecast skill of the LSTM for the tropical surface air temperature. (a) The plot shows the training and validation loss values, with each color representing a different number of features and the top line for each color representing the validation loss, while the bottom line represents training loss; the x axis shows the epoch reached during training. (b) The plots show the skill of the forecast, compared against the full original field. For the solid lines each color represents a different number of features, while the dashed line shows the skill of the persistence. The plot on the top shows the RMSE, while the one on the bottom the correlation. The x axis represents the number of time steps in the future with respect to the input sequence for the given forecast.

(a)



(b)

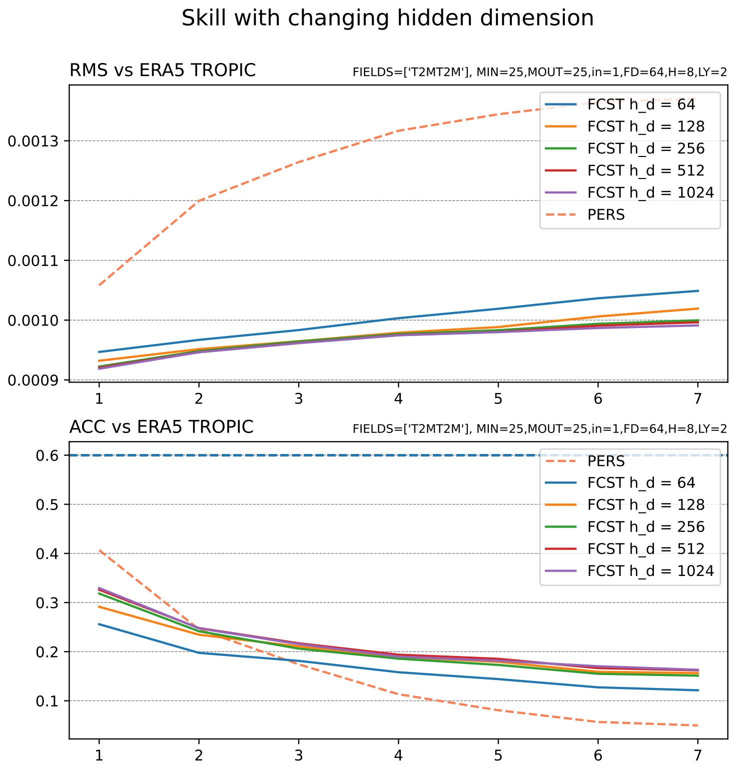
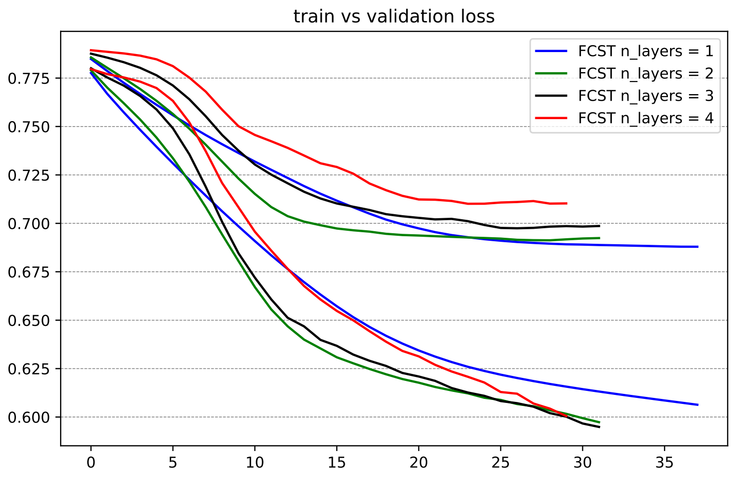


Figure 2.7: Loss values and forecast skill of the LSTM for the tropical surface air temperature. (a) The plot shows the training and validation loss values, with each color representing a different hidden dimension and the top line for each color representing the validation loss, while the bottom line represents training loss; the x axis shows the epoch reached during training. (b) The plots show the skill of the forecast, compared against the full original field. For the solid lines each color represents a different hidden dimension, while the dashed line shows the skill of the persistence. The plot on the top shows the RMSE, while the one on the bottom the correlation. The x axis represents the number of time steps in the future with respect to the input sequence for the given forecast.

(a)



(b)

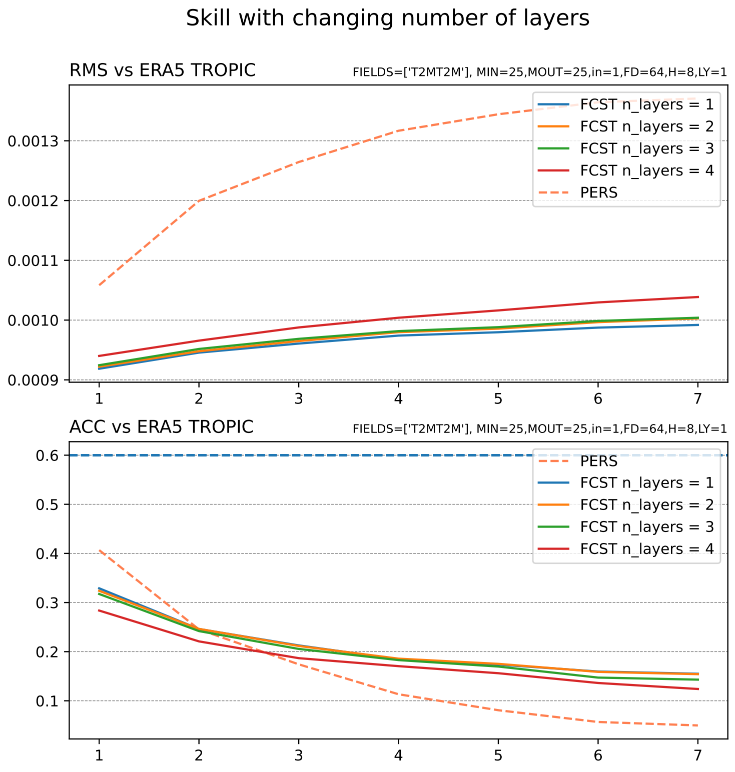
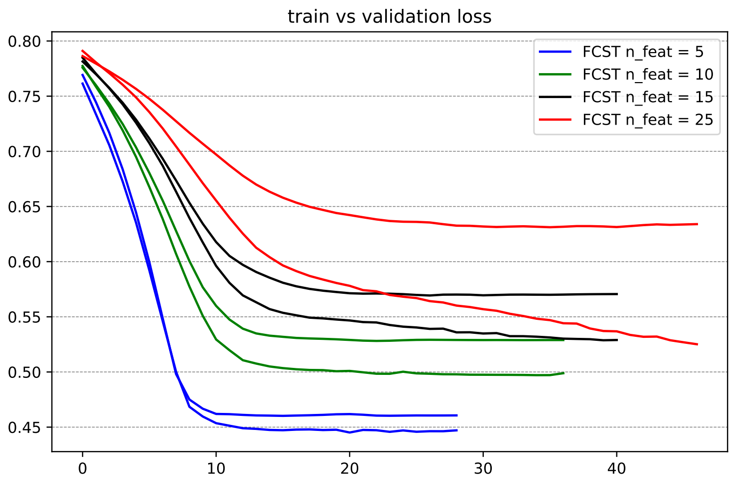


Figure 2.8: Loss values and forecast skill of the LSTM for the tropical surface air temperature. (a) The plot shows the training and validation loss values, with each color representing a different number of layers and the top line for each color representing the validation loss, while the bottom line represents training loss; the x axis shows the epoch reached during training. (b) The plots show the skill of the forecast, compared against the full original field. For the solid lines each color represents a different number of layers, while the dashed line shows the skill of the persistence. The plot on the top shows the RMSE, while the one on the bottom the correlation. The x axis represents the number of time steps in the future with respect to the input sequence for the given forecast.

(a)



(b)

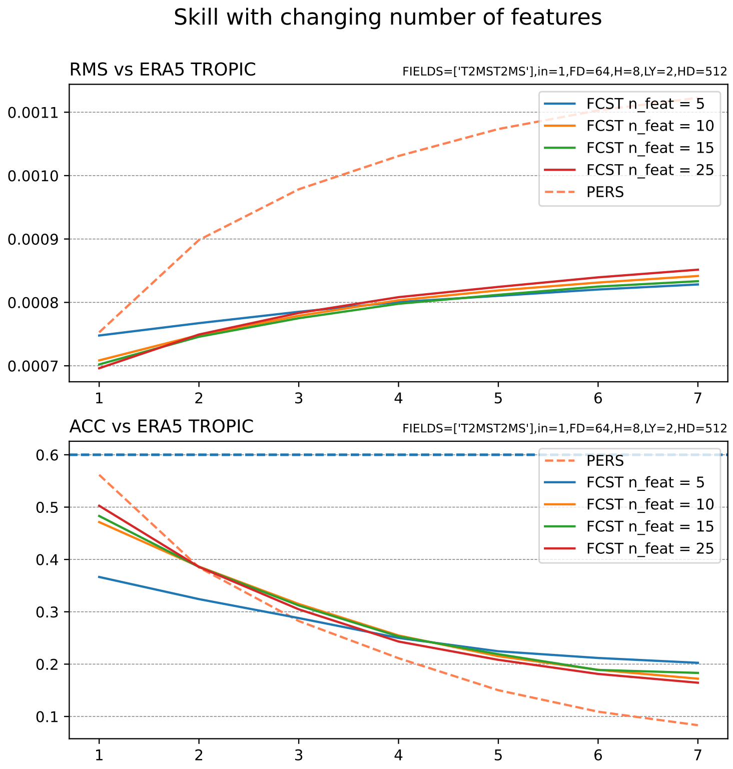
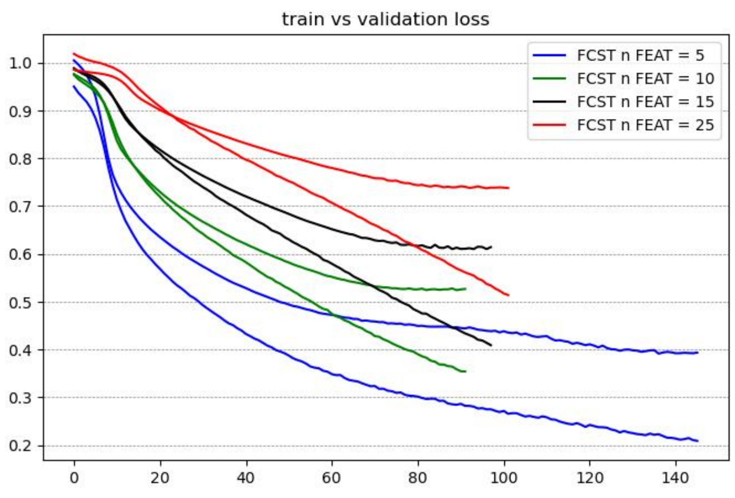


Figure 2.9: Loss values and forecast skill of the LSTM for the tropical surface air temperature over the ocean. (a) The plot shows the training and validation loss values, with each color representing a different number of features and the top line for each color representing the validation loss, while the bottom line represents training loss; the x axis shows the epoch reached during training. (b) The plots show the skill of the forecast, compared against the full original field. For the solid lines each color represents a different number of features, while the dashed line shows the skill of the persistence. The plot on the top shows the RMSE, while the one on the bottom the correlation. The x axis represents the number of time steps in the future with respect to the input sequence for the given forecast.

(a)



(b)

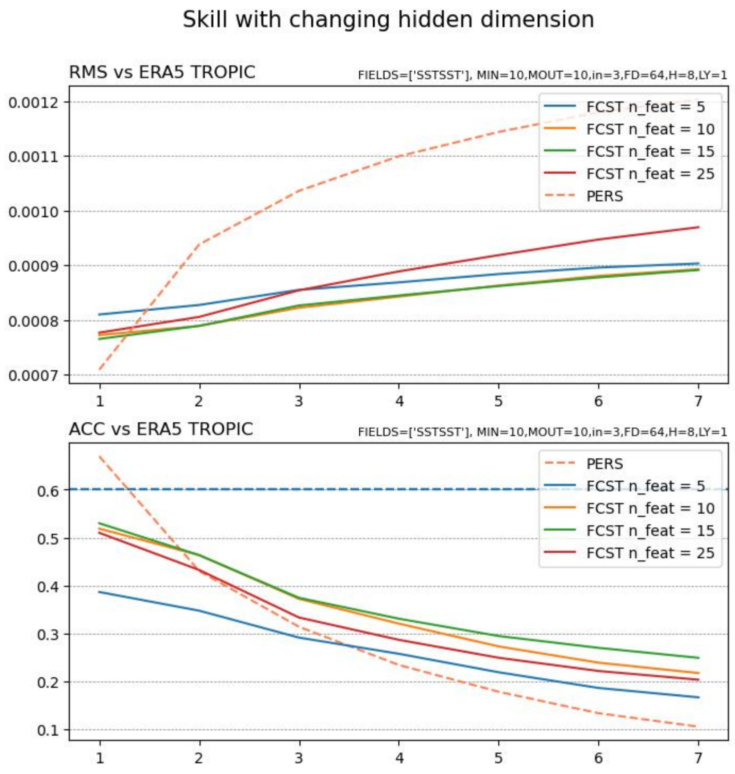
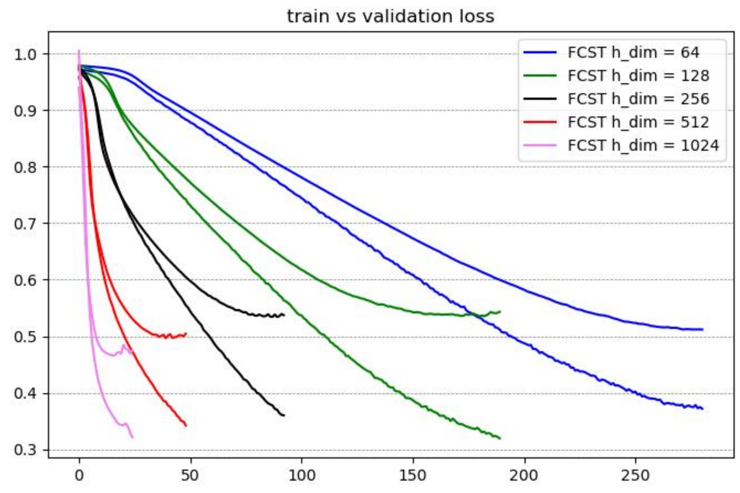


Figure 2.10: Loss values and forecast skill of the transformer model for the tropical sea surface temperature. (a) The plot shows the training and validation loss values, with each color representing a different number of features and the top line for each color representing the validation loss, while the bottom line represents training loss; the x axis shows the epoch reached during training. (b) The plots show the skill of the forecast, compared against the full original field. For the solid lines each color represents a different number of features, while the dashed line shows the skill of the persistence. The plot on the top shows the RMSE, while the one on the bottom the correlation. The x axis represents the number of time steps in the future with respect to the input sequence for the given forecast.

(a)



(b)

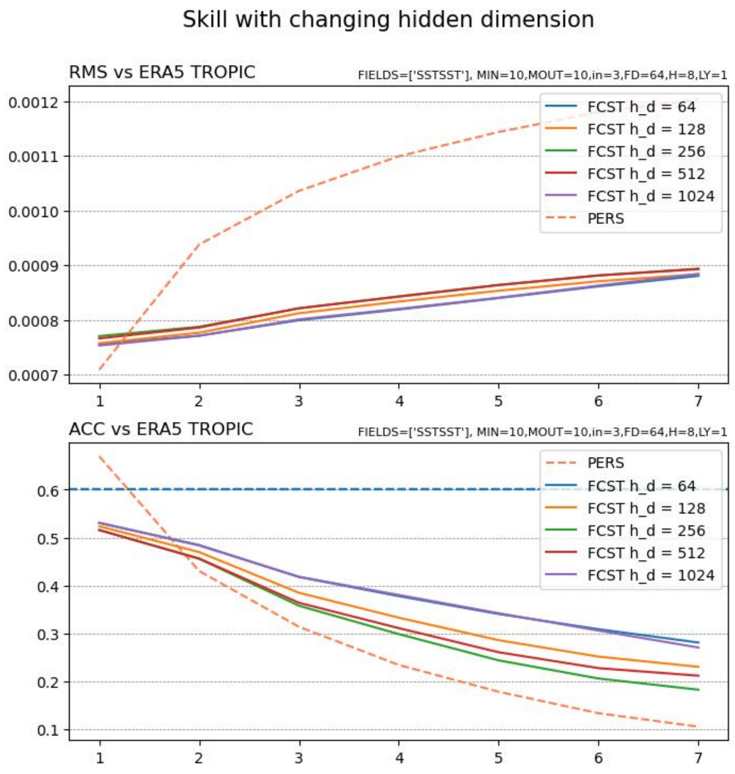
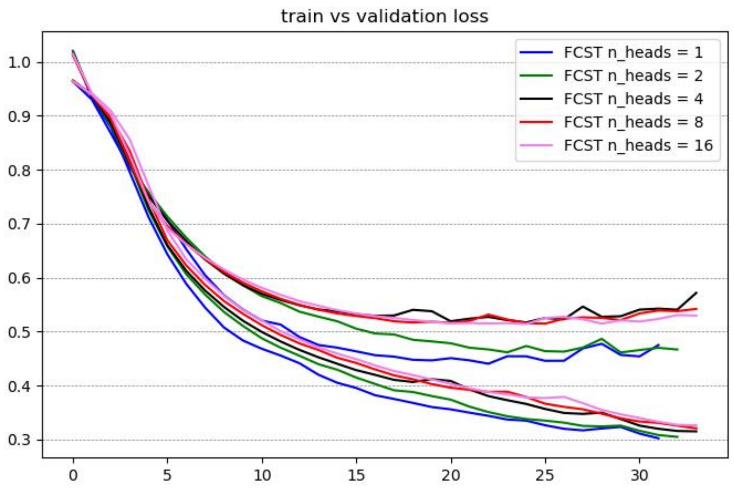


Figure 2.11: Loss values and forecast skill of the transformer model for the tropical sea surface temperature. (a) The plot shows the training and validation loss values, with each color representing a different hidden dimension and the top line for each color representing the validation loss, while the bottom line represents training loss; the x axis shows the epoch reached during training. (b) The plots show the skill of the forecast, compared against the full original field. For the solid lines each color represents a different hidden dimension, while the dashed line shows the skill of the persistence. The plot on the top shows the RMSE, while the one on the bottom the correlation. The x axis represents the number of time steps in the future with respect to the input sequence for the given forecast.

(a)



(b)

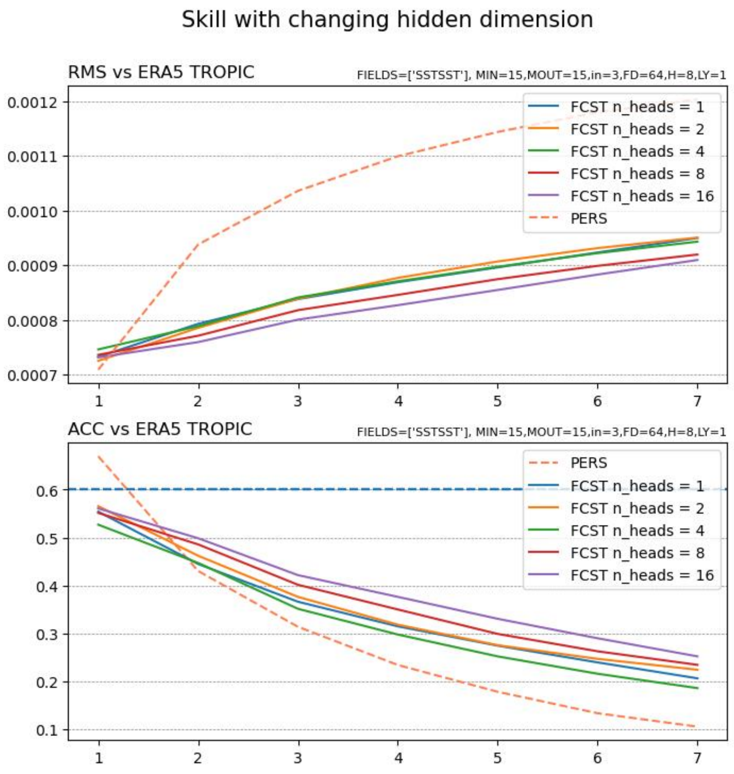


Figure 2.12: Loss values and forecast skill of the transformer model for the tropical sea surface temperature. (a) The plot shows the training and validation loss values, with each color representing a different number of heads and the top line for each color representing the validation loss, while the bottom line represents training loss; the x axis shows the epoch reached during training. (b) The plots show the skill of the forecast, compared against the full original field. For the solid lines each color represents a different number of heads, while the dashed line shows the skill of the persistence. The plot on the top shows the RMSE, while the one on the bottom the correlation. The x axis represents the number of time steps in the future with respect to the input sequence for the given forecast.

Chapter 3

Discussion

3.1 LSTM

3.1.1 Sea Surface Temperature

We now focus on how the model deals with the number of retained EOFs, or the number of features used. Tracking how the model manages EOF is crucial, since this statistical decomposition is the fundamental process that we applied to the original data, and leaves the model to deal with inputs that are completely different in structure and meaning to what the full original dataset was.

We can go back to figure 2.1 to understand how the model forecast is influenced by this process. If we take a look at the evolution of the values of the loss during training, we can see that they starkly decrease when the number of features is lower. This is to be expected since having less features directly implies there is less to learn so we expect a lower error. But, since we ultimately want a good forecast for the original variable, we also have to remember that retaining less EOFs means we are looking at a lower fraction of the total variance of the full field. So being better at predicting the time series evolution of the first few PCs, does not necessarily contribute to better general results. In particular, these are the chosen test values for the number of features and their respective explained variance for the tropical sea surface temperature case:

Number of features	Explained variance
5	43%
10	55%
15	63%
25	72%

This means that even with a perfect prediction, reconstructing the field from only 5 EOFs would only result in a description of 43% of the full initial field. So even if the loss for the lowest number of features is low, it does not mean that the forecast will be the most accurate. Nevertheless, we can see that at 25 features the model has a bigger problem with overfitting, as the validation and training loss diverge significantly from one another. It is actually the 10 features case the one where the two losses are closest.

It is not that surprising to see a high accuracy for the first time step of the persistence, since in climatology the sea surface temperature is known to have a long timescale of variability, and it is normal to see it not change much in a single month. Moreover, the persistence takes into account the full field as its starting point, while the forecast will be at an inherent disadvantage due to the variability lost during the EOF process. Nevertheless, we can see that at longer prediction times the model overtakes the persistence. When comparing the various number of features between them, we can see that even if the one with 5 retained EOFs has lower loss than the others, it happens to have the lowest skill, meaning that the retained variance is too small to accurately describe the full field. Moreover, we can see that the others go more or less together, meaning that for the computational cost, our model excels at 10 features retained.

In the third plot of figure 2.1, we can see the skill of the model when compared to the field reconstructed only from the respective number of EOFs. This result follows more closely what we see in the loss values, as the cases with less retained EOFs manage to have a higher skill. In any case we can notice that at least for the forecast of said number of features, every test of the model has much higher skill than against the full field, and this is to be expected.

However, the model being good at forecasting the first few EOFs is very useful, even more so that the skill against the full field may imply. For example, in the optimal case with 10 features we retain 55% of the variance of the full field. This percentage may seem small, but we have to remember that the dataset consists in the sea surface temperature for the whole tropical zone, and depicting every single detail in every sea and every ocean could not be in the scope of such a simple model. Moreover, the single most impactful variability mode at these timescales is El Niño. and by accurately predicting it, one can make good assumptions on a number of other phenomena and anomalies especially in the tropics and the Pacific, but also on the entire world, since El Niño drives seasonal variability even at very long distances, through processes called teleconnections, some of which are shown if figure 3.1 [39][40]. All of this to say that since the EOF decomposition ranks the modes in terms of variance explained, most of El Niño evolution is present in just the first few modes. This means that we can at least say that these models are good at forecasting the evolution of El Niño, and from that other assumptions could be made on other regions and phenomena, even if this effects may not be directly present in the outputs of the model.

Moreover, to capture this small number of features, the models don't need to be extremely complex. As we can see in figures 2.2 and 2.3, of course with a very small dimension of the hidden state the model does not have the capacity to capture the necessary patterns to make a good forecast, but after a certain value there is not a lot to gain by increasing it further. Furthermore, even if LSTMs and deep networks in general may often benefit by increasing the number of layer, in our case the model has a very high skill even with just one layer, even if by a small margin the case with 2 layers still performs better. In any case, by further increasing the complexity we notice that the model just tends to overfit more and more, as it probably starts to capture too much noise as if it was the pattern it was trying to learn, and has then trouble with generalizing to new data.

To further analyse the model capacity to forecast El Niño in particular, we can look at how it performs if we cut the input data to only the tropical pacific area. The first thing to notice is that this way we are cutting much of the small local variability and noise that is included in the whole dataset, so the model has way less patterns to learn to still be able to recompose the whole field. We can see this directly by looking at the explained variance by the first few EOFs of the tropical pacific:

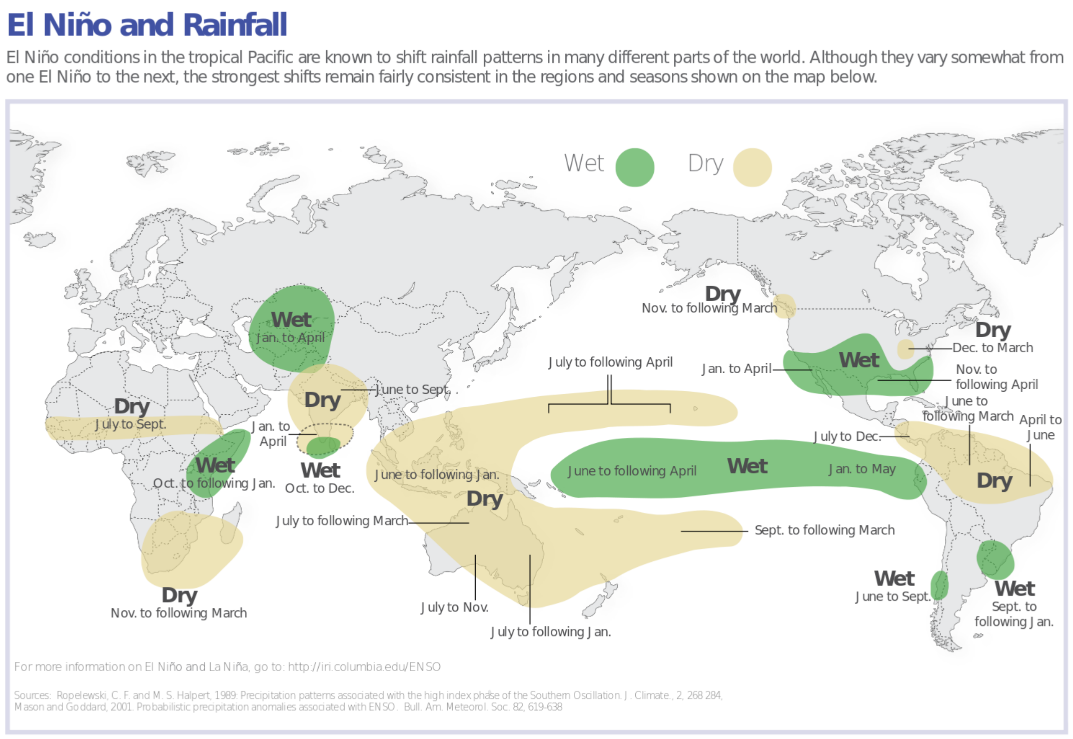

El Niño and Rainfall
El Niño conditions in the tropical Pacific are known to shift rainfall patterns in many different parts of the world. Although they vary somewhat from one El Niño to the next, the strongest shifts remain fairly consistent in the regions and seasons shown on the map below.
Wet
Dry
Wet Jan. to April
Dry June to Sept.
Dry July to Sept.
Jan. to April
Wet Oct. to Dec.
Wet Oct. to following Jan.
Dry Nov. to following March
June to following Jan.
Dry July to following March
July to Nov.
July to following Jan.
July to following April
Dry Nov. to following March
Jan. to April
Wet
Dry Dec. to March
Nov. to following April
June to following March
April to June
July to Dec.
June to following April
Wet
Jan. to May
Dry
Sept. to following March
Wet June to Sept.
Wet Sept. to following Jan.
For more information on El Niño and La Niña, go to: http://iri.columbia.edu/ENSO
Sources: Ropelewski, C. F. and M. S. Halpert, 1989. Precipitation patterns associated with the high index phase of the Southern Oscillation. J. Climate, 2, 268-284. Mason and Goddard, 2001. Probabilistic precipitation anomalies associated with ENSO. Bull. Am. Meteorol. Soc. 82, 619-638

Figure 3.1: Typical rainfall patterns during El Niño events. Such teleconnections are likely to occur during El Niño events, but not certain [41][42].

Number of features	Explained variance
5	74%
10	82%
15	86%
25	89%

As we can see the first few EOFs account for almost the full variance of whole region, as adding 10 more EOFs after the fifteenth only adds a mere 3% variance. This means that we can train the model to predict only the first 10 PCs like in the previous case, and the resulting skill will be much greater, as the reconstruction naturally explains more processes. This is reflected in figure 2.4 where we can see that in general the skill is much higher than in the full tropical case. For similar reason the skill of the persistence is also higher to match, but in comparison the model still performs better than in the previous case, as even the first month forecast manages to almost reach the skill of the persistence, and then overtakes it easily in the second month.

3.1.2 Surface Air Temperature

When looking at the air temperature 2 meters above the surface (t2m) the story is much different. First and foremost, in figure 2.6 we can see that the loss value is much greater with respect to the sea surface temperature case. Even more crucial is the fact that the difference between the training and validation loss is very large, indicating severe overfitting. This means that the model is focusing too much on learning the specific evolution of the training data, but cannot generalize to other data. This time the problem does not lie within the EOF decomposition, as if we look at the variance retained by the first few EOFs, we can see that the values are very close to the sea surface temperature case:

Number of features	Explained variance
5	40%
10	54%
15	63%
25	74%

Nevertheless, the problem is not totally accounted for just by the model being unfit to predict this type of data. This is because if we look at the skill of the prediction we see something much more similar to the sea surface temperature case. The skill is overall much lower, but comparing it to the persistence we can see the same pattern, with the forecast of the model overtaking the persistence skill after a few months.

This goes to show the big disparity between trying to forecast an oceanic variable versus an atmospheric one. The atmosphere evolves at a much higher frequency as seen in figure 3.2, and since we are trying to make seasonal forecasts, this fast variability falls beneath our time resolution of one month and gets lost, while at the same time making the data much more noisy [44]. The overfitting seen by the model can be explained by the fact that while it is trying to learn the underlying long term patterns within the

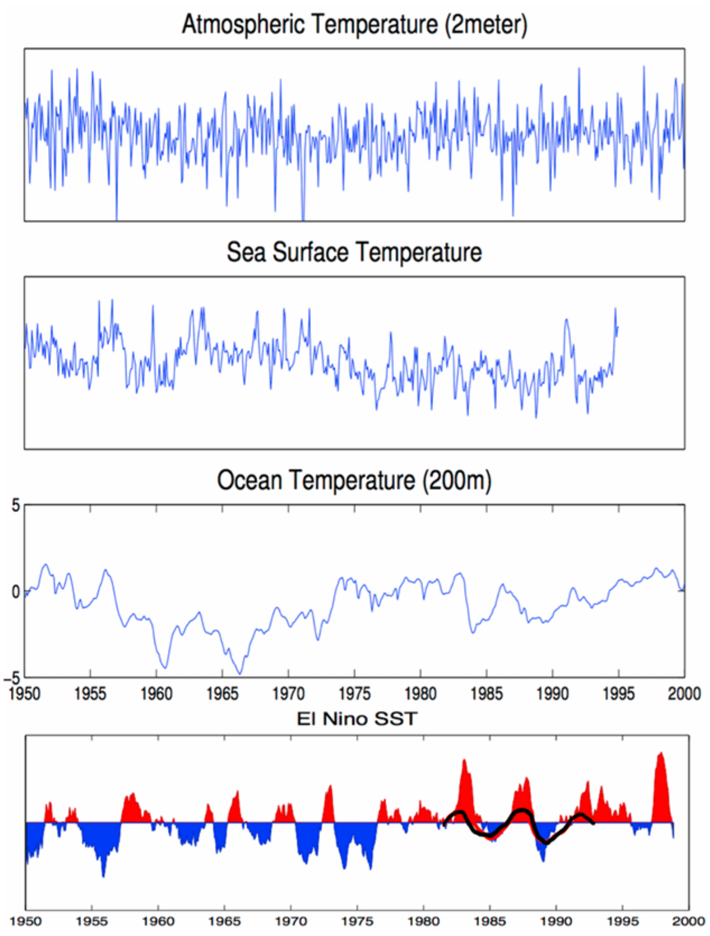


Figure 3.2: Example time series for air temperature 2 meters above sea level, sea surface temperature, and ocean temperature at 200 meters of depth. At the end the El Niño sst time series is added for comparison [43].

input data, it instead captures this noise as if it was a real pattern, and fails to predict new data.

However, even if the atmosphere is this unpredictable and varies on such fast time scales, we can see that some of the skill of the forecast is still retained even if we are predicting a much longer time window. This is because even if we are looking at air temperature, the long term variability of the surface air temperature over the ocean is basically fully driven by the oceanic one. In figure 2.9 we can see this effect, as it shows at the performance of the model on the air temperature only above the oceans, and the plot looks almost identical to the one in figure 2.1 regarding the sea surface temperature. This also goes to show that the decrease in the skill for the t2m case is concentrated over land, where our model does cannot manage to account for the fast variability of the atmosphere.

3.2 Transformer

If we compare the performance of our LSTM to that of our Transformer, the first thing we can notice is that the transformer suffers from a worse case of overfitting. So even if the the loss value relative to the training dataset is smaller, the validation loss end up being higher than the previous model. We can also notice that with standard settings the transformer takes longer to train in terms of epochs passed.

In spite of the glaring problems with the evolution of the loss during training, the skill tends to somewhat match the one from the LSTM, with some variation on the trend of the curve with respect to the number of months predicted.

Regarding the complexity, the transformer seems to benefit more from having a deeper, more complex model, as the skill seems to be at its highest with a higher number of heads and hidden dimension. It also seems to favor a slightly higher number of features with respect to the LSTM.

Summary and Conclusions

The purpose of this thesis work is to first give a broad overview of the most successful machine learning models in the field of climate forecasts, but also to show some practical methods and processes essential to their implementation and provide examples through the use of some experiments.

We first explained the inner workings and building blocks of two of the most popular and useful machine learning architectures in the field of time series prediction, the LSTM and the transformer, showing their key advantages and strong points, and their shortcomings.

We then provided a detailed description of the setup and methods used for our representative experiments. We went through an overview of the ERA5 dataset, followed by the description of the EOF decomposition method, and the creation of the input sequences used during training. We also described the training process step by step, highlighting useful methods and precautions that were applied.

Finally, we presented the results obtained by our various experiments and evaluated the performance of our LSTM and transformer models on the forecasts of sea surface temperature and surface air temperature for the tropical region. The models provided valuable insights regarding their behavior with different number of retained EOFs and on the handling of oceanic versus atmospheric data.

The number of retained EOFs, which are represented by the number of features of our models, plays a crucial role in the performance of the models. We observed that while retaining too few EOFs limits the accuracy of the forecasts, through the loss of critical variance information, retaining too many also causes problems. In particular, it leads to overfitting, with the model not being able to keep track of all the patterns that arise and not being able to generalize well to other data. In our case, the optimal number of EOFs retained for the sst forecast was around 10, which for the whole tropical region accounted for 55% of the full variance. This is also a testament to the strengths of the EOF analysis in this field, by enabling us to get consistent results while massively increasing computational efficiency. For example, in this case going from the full 3-dimensional tropical field to the EOF decomposed one, while the time dimension stays the same length, on the spatial side we go from around 10^5 points to just 10 features.

For the sst in particular, despite the reduction in variance, the model demonstrated high skill for the prediction of the most dominant modes of variability, first and foremost managing to capture El Niño evolution, further demonstrated by the fact that limiting the forecast to the tropical Pacific area we get strong and consistent forecast. Although the performance for the whole tropical region was less satisfactory, with the models struggling to capture the granular variability of various areas directly, we can still make indirect assumptions on several areas from the knowledge of the state of El Niño.

The forecast of surface air temperature was systemically hampered by more significant challenges. In particular, by including the air temperature over land the predictions for the tropical area achieved low skill, a fact that is to be expected due to the faster time scale of atmospheric variability limiting the direct prediction of atmospheric variables on longer time scales. However, when limiting the model to the air temperature over the oceans, the skill climbed back up to the values for the sst experiments, in line with the well known fact that long term variability of the atmosphere over sea is fully driven by oceanic variability.

Overall, we noticed that increasing model complexity, such as adding more layers or increasing the dimension of the hidden state, did not consistently improve the performance, and it often exacerbated overfitting instead.

In terms of LSTM versus transformer performance, we noticed that even if LSTM models tend to be outperformed by transformers for bigger and more convoluted tasks, in our experiments the LSTM model somewhat outperformed the transformer, as the latter suffered from more severe overfitting despite achieving lower training loss. Nonetheless, the skills of both models were quite similar to each other. We may still favor the LSTM for our simple case studies due to its lower complexity and depth, while still managing to generalize better than its counterpart. This however, would not be the case for more complex tasks involving for example more variables or longer sequences.

In summary, both models demonstrated good forecast performances regarding the sea surface temperature variables, particularly for the prediction of the most important climate modes like El Niño. Both models also exhibited overfitting tendencies when dealing with greater complexity and depth and with a larger number of retained EOFs, highlighting the importance of balancing model complexity with the nature of the data being forecast.

Bibliography

[1] J. G. Charney, R. Fjörtoft, and J. Von Neumann. "Numerical Integration of the Barotropic Vorticity Equation". In: *Tellus A: Dynamic Meteorology and Oceanography* 2.4 (1950), pp. 237–254. ISSN: 1600-0870. DOI: 10.3402/tellusa.v2i4.8607. URL: https://account.a.tellusjournals.se/index.php/su-j-tadmo/article/view/4055 (visited on 06/17/2024).

[2] B. Bochenek and Z. Ustrnul. "Machine Learning in Weather Prediction and Climate Analyses—Applications and Perspectives". en. In: *Atmosphere* 13.2 (2022), p. 180. ISSN: 2073-4433. DOI: 10.3390/atmos13020180. URL: https://www.mdpi.com/2073-4433/13/2/180 (visited on 06/17/2024).

[3] K. Hornik, M. Stinchcombe, and H. White. "Multilayer feedforward networks are universal approximators". en. In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 08936080. DOI: 10.1016/0893-6080(89)90020-8. URL: https://linkinghub.elsevier.com/retrieve/pii/0893608089900208 (visited on 06/17/2024).

[4] A. Singh, N. Thakur, and A. Sharma. "A review of supervised machine learning algorithms". In: *2016 3rd international conference on computing for sustainable global development (INDIACom)*. Ieee. 2016, pp. 1310–1315.

[5] D. Dhall, R. Kaur, and M. Juneja. "Machine learning: a review of the algorithms and its applications". In: *Proceedings of ICRIC 2019: Recent innovations in computing* (2020), pp. 47–63.

[6] B. Mahesh. "Machine learning algorithms-a review". In: *International Journal of Science and Research (IJSR).[Internet]* 9.1 (2020), pp. 381–386.

[7] C. Ferrie. *Neural networks for babies*. Naperville: Sourcebooks Jabberwocky, 2019. ISBN: 978-1-4926-7120-6.

[8] R. M. Schmidt. *Recurrent Neural Networks (RNNs): A gentle Introduction and Overview*. arXiv:1912.05911 [cs, stat]. 2019. URL: http://arxiv.org/abs/1912.05911 (visited on 08/27/2024).

[9] S. A. Abdulkarim and A. P. Engelbrecht. "Time series forecasting with feedforward neural networks trained using particle swarm optimizers for dynamic environments". en. In: *Neural Computing and Applications* 33.7 (2021), pp. 2667–2683. ISSN: 0941-0643, 1433-3058. DOI: 10.1007/s00521-020-05163-4. URL: https://link.springer.com/10.1007/s00521-020-05163-4 (visited on 09/28/2024).

[10] Y.-G. Ham, J.-H. Kim, and J.-J. Luo. "Deep learning for multi-year ENSO forecasts". en. In: *Nature* 573.7775 (2019), pp. 568–572. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/s41586-019-1559-7. URL: https://www.nature.com/articles/s41586-019-1559-7 (visited on 09/28/2024).

[11] J. Schmidhuber. "Deep Learning in Neural Networks: An Overview". In: *Neural Networks* 61 (2015). arXiv:1404.7828 [cs], pp. 85–117. ISSN: 08936080. DOI: 10.1016/j.neunet.2014.09.003. URL: http://arxiv.org/abs/1404.7828 (visited on 09/28/2024).

[12] A. Tealab. "Time series forecasting using artificial neural networks methodologies: A systematic review". en. In: *Future Computing and Informatics Journal* 3.2 (2018), pp. 334–340. ISSN: 23147288. DOI: 10.1016/j.fcij.2018.10.003. URL: https://linkinghub.elsevier.com/retrieve/pii/S2314728817300715 (visited on 09/28/2024).

[13] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory". en. In: *Neural Computation* 9.8 (1997), pp. 1735–1780. ISSN: 0899-7667, 1530-888X. DOI: 10.1162/neco.1997.9.8.1735. URL: https://direct.mit.edu/neco/article/9/8/1735-1780/6109 (visited on 08/28/2024).

[14] S. Hochreiter. "Untersuchungen zu dynamischen neuronalen Netzen". In: (1991).

[15] C. Broni-Bedaiko et al. "El Niño-Southern Oscillation forecasting using complex networks analysis of LSTM neural networks". en. In: *Artificial Life and Robotics* 24.4 (2019), pp. 445–451. ISSN: 1433-5298, 1614-7456. DOI: 10.1007/s10015-019-00540-2. URL: http://link.springer.com/10.1007/s10015-019-00540-2 (visited on 09/28/2024).

[16] Q. Guo, Z. He, and Z. Wang. "Monthly climate prediction using deep convolutional neural network and long short-term memory". en. In: *Scientific Reports* 14.1 (2024), p. 17748. ISSN: 2045-2322. DOI: 10.1038/s41598-024-68906-6. URL: https://www.nature.com/articles/s41598-024-68906-6 (visited on 09/28/2024).

[17] A. Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[18] Q. Wen et al. *Transformers in Time Series: A Survey*. arXiv:2202.07125 [cs, eess, stat]. 2023. URL: http://arxiv.org/abs/2202.07125 (visited on 06/17/2024).

[19] A. Zhang et al. *Dive into Deep Learning*. arXiv:2106.11342 [cs]. 2023. URL: http://arxiv.org/abs/2106.11342 (visited on 08/27/2024).

[20] R. C. Staudemeyer and E. R. Morris. *Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks*. arXiv:1909.09586 [cs]. 2019. URL: http://arxiv.org/abs/1909.09586 (visited on 08/27/2024).

[21] C. Olah. *Colah's blog*. en. 2015. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/?source=post_page37e2f46f1714.

[22] F. A. Gers, J. Schmidhuber, and F. Cummins. "Learning to Forget: Continual Prediction with LSTM". en. In: *Neural Computation* 12.10 (2000), pp. 2451–2471. ISSN: 0899-7667, 1530-888X. DOI: 10.1162/089976600300015015. URL: https://direct.mit.edu/neco/article/12/10/2451-2471/6415 (visited on 08/29/2024).

[23] S. Islam et al. *A Comprehensive Survey on Applications of Transformers for Deep Learning Tasks*. arXiv:2306.07303 [cs]. 2023. URL: http://arxiv.org/abs/2306.07303 (visited on 09/13/2024).

[24] S. Ahmed et al. "Transformers in Time-series Analysis: A Tutorial". In: *Circuits, Systems, and Signal Processing* 42.12 (2023). arXiv:2205.01138 [cs], pp. 7433–7466. ISSN: 0278-081X, 1531-5878. DOI: 10.1007/s00034-023-02454-8. URL: http://arxiv.org/abs/2205.01138 (visited on 08/22/2024).

[25] C. Szegedy et al. *Going Deeper with Convolutions*. arXiv:1409.4842 [cs]. 2014. URL: http://arxiv.org/abs/1409.4842 (visited on 09/17/2024).

[26] J. L. Ba, J. R. Kiros, and G. E. Hinton. *Layer Normalization*. arXiv:1607.06450 [cs, stat]. 2016. URL: http://arxiv.org/abs/1607.06450 (visited on 09/17/2024).

[27] R. Baatz et al. "Reanalysis in Earth System Science: Toward Terrestrial Ecosystem Reanalysis". en. In: *Reviews of Geophysics* 59.3 (2021), e2020RG000715. ISSN: 8755-1209, 1944-9208. DOI: 10.1029/2020RG000715. URL: https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2020RG000715 (visited on 08/24/2024).

[28] H. Hersbach et al. "The ERA5 global reanalysis". en. In: *Quarterly Journal of the Royal Meteorological Society* 146.730 (2020), pp. 1999–2049. ISSN: 0035-9009, 1477-870X. DOI: 10.1002/qj.3803. URL: https://rmets.onlinelibrary.wiley.com/doi/10.1002/qj.3803 (visited on 08/24/2024).

[29] Zhang. *Mathematical and Physical Fundamentals of Climate Change*. en. Elsevier, 2015. ISBN: 978-0-12-800066-3. DOI: 10.1016/C2013-0-14403-0. URL: https://linkinghub.elsevier.com/retrieve/pii/C20130144030 (visited on 08/24/2024).

[30] G. Liguori. "Course on Tropical Air-Sea Interactions, Science of Climate, University of Bologna". In: (2023).

[31] C. .-. Medium. *Stochastic Gradient Descent*. URL: https://crossknight.medium.com/stochastic-gradient-descent-bfff0a7b3433.

[32] D. P. Kingma and J. Ba. *Adam: A Method for Stochastic Optimization*. arXiv:1412.6980 [cs]. 2017. URL: http://arxiv.org/abs/1412.6980 (visited on 08/31/2024).

[33] J. Brownlee. *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning*. URL: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

[34] I. Loshchilov and F. Hutter. *Decoupled Weight Decay Regularization.* arXiv:1711.05101 [cs, math]. 2019. URL: http://arxiv.org/abs/1711.05101 (visited on 08/30/2024).

[35] T. Martin. *A (Very Short) Visual Introduction to Learning Rate Schedulers (With Code).* URL: https://medium.com/@theom/a-very-short-visual-introduction-to-learning-rate-schedulers-with-code-189eddffdb00.

[36] *PyTorch: An Imperative Style, High-Performance Deep Learning Library.* arXiv:1912.01703 [cs, stat]. 2019. URL: http://arxiv.org/abs/1912.01703.

[37] R. Pascanu, T. Mikolov, and Y. Bengio. *On the difficulty of training Recurrent Neural Networks.* arXiv:1211.5063 [cs]. 2013. URL: http://arxiv.org/abs/1211.5063.

[38] N. Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[39] S. Zhao et al. "Explainable El Niño predictability from climate mode interactions". en. In: *Nature* 630.8018 (2024), pp. 891–898. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/s41586-024-07534-6. URL: https://www.nature.com/articles/s41586-024-07534-6 (visited on 09/26/2024).

[40] O. Alizadeh. "A review of ¡span style="font-variant:small-caps;"¿ENSO¡/span¿ teleconnections at present and under future global warming". en. In: *WIREs Climate Change* 15.1 (2024), e861. ISSN: 1757-7780, 1757-7799. DOI: 10.1002/wcc.861. URL: https://wires.onlinelibrary.wiley.com/doi/10.1002/wcc.861 (visited on 09/26/2024).

[41] S. J. Mason and L. Goddard. "Probabilistic Precipitation Anomalies Associated with ENSO". en. In: *Bulletin of the American Meteorological Society* 82.4 (2001), pp. 619–638. ISSN: 0003-0007, 1520-0477. DOI: 10.1175/1520-0477(2001)082<0619:PPAAWE>2.3.CO;2. URL: http://journals.ametsoc.org/doi/10.1175/1520-0477(2001)082%3C0619:PPAAWE%3E2.3.CO;2 (visited on 09/26/2024).

[42] NOAA. *National Oceanic and Atmospheric Administration, Understanding El Nino.* URL: https://www.noaa.gov/understanding-el-nino.

[43] G. Liguori. “Course on Climate Variability, Science of Climate, University of Bologna”. In: (2023).

[44] C. E. Bulgin, C. J. Merchant, and D. Ferreira. “Tendencies, variability and persistence of sea surface temperature anomalies”. en. In: *Scientific Reports* 10.1 (2020), p. 7986. ISSN: 2045-2322. DOI: 10.1038/s41598-020-64785-9. URL: https://www.nature.com/articles/s41598-020-64785-9 (visited on 09/28/2024).

Appendix

Algorithm 1 shows the full Adam optimization algorithm, while algorithm 2 shows AdamW.

Algorithm 1 Adam: Stochastic Optimization Algorithm [32]

```
Require: $\alpha$: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters $\theta$
Require: $\theta_0$: Initial parameter vector
Initialize: $m_0 \leftarrow 0$ (1st moment vector)
Initialize: $v_0 \leftarrow 0$ (2nd moment vector)
Initialize: $t \leftarrow 0$ (timestep)
 1: while $\theta_t$ not converged do
 2:     $t \leftarrow t + 1$
 3:     $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$    ▷ Get gradients w.r.t. stochastic objective at timestep $t$
 4:     $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$    ▷ Update biased first moment estimate
 5:     $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$    ▷ Update biased second raw moment estimate
 6:     $\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$    ▷ Compute bias-corrected first moment estimate
 7:     $\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$    ▷ Compute bias-corrected second raw moment estimate
 8:     $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}$    ▷ Update parameters
 9: end while
10: return $\theta_t$    ▷ Resulting parameters
```

Algorithm 2 Adam with L2 regularization and AdamW (decoupled weight decay) [34]

Given: $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, $\lambda \in \mathbb{R}$

Initialize: timestep $t \leftarrow 0$, parameter vector $\theta_0 \in \mathbb{R}^n$, first moment vector $m_0 \leftarrow 0$, second moment vector $v_0 \leftarrow 0$, schedule multiplier $\eta_0 \in \mathbb{R}$

```
1:  repeat
2:      $t \leftarrow t + 1$
3:      $f_t(\theta_{t-1}) \leftarrow \text{SelectBatch}(\theta_{t-1})$    ▷ Select batch and return the corresponding gradient
4:      $g_t \leftarrow \nabla f_t(\theta_{t-1}) + \lambda\theta_{t-1}$    ▷ Gradient with L2 regularization
5:      $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$    ▷ Update biased first moment estimate (element-wise)
6:      $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$    ▷ Update biased second raw moment estimate (element-wise)
7:      $\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$    ▷ Compute bias-corrected first moment estimate
8:      $\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$    ▷ Compute bias-corrected second raw moment estimate
9:      $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$    ▷ Can be fixed, decayed, or used for warm restarts
10:     $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon} + \lambda\theta_{t-1}\right)$    ▷ AdamW with decoupled weight decay
11: until stopping criterion is met
12: return optimized parameters $\theta_t$
```