Universität Stuttgart

Analytic Computing
Institut für Parallele und Verteilte Systeme

Universitätsstraße 32
70569 Stuttgart

Bachelorarbeit

Multistep Prediction of Vehicle States using Transformers

Stefan Bolz

Studiengang: B.Sc. Informatik

Prüfer: Prof. Dr. Steffen Staab

Betreuer: Alexandra Baier, M.Sc.

begonnen am: 08.10.2020

beendet am: 08.06.2021

Abstract

We develop a transformer architecture for multistep prediction of vehicle states. Multistep prediction is the prediction of states based on initial states and a series of control inputs. Research in natural language processing (NLP) promises advantages w.r.t. training time and prediction accuracy for the transformer architecture compared to a state-of-the-art LSTM model. We investigate the benefits of positional encoding and non-linear input embeddings for our multi-step transformer. Due to the inherent causality in dynamical systems, an explicit inclusion of time information via positonal encodings might not be necessary. Learning non-linear embeddings might improve prediction accuracy by extracting relevant data from the original input similar to word embeddings used with transformers in NLP. We evaluate our method on the use case of predicting ship motion under environmental disturbances. We cannot prove any benefits w.r.t. training time compared to LSTMs, transformers do however achieve similar prediction accuracy. We show that positional encodings have a detrimental effect on the prediction accuracy of multistep transformers, which proves that the transformer infers causal information from the input. Non-linear embeddings of control inputs and initial state yield a negligible improvement of prediction accuracy.

Kurzfassung

Wir entwickeln eine Transformer-Architektur für die mehrstufige Vorhersage von Fahrzeugzuständen. Mehrstufige Vorhersage ist die Vorhersage von Zuständen auf der Grundlage von Ausgangszuständen und einer Reihe von Steuereingaben. Die Forschung im Bereich der natürlichen Sprachverarbeitung (NLP) verspricht für die Transformer-Architektur Vorteile in Bezug auf Trainingszeit und Vorhersagegenauigkeit im Vergleich zu einem modernen LSTM-Modell.

Wir untersuchen die Vorteile von Positionskodierungen und nicht-linearen Eingabeeinbettungen für unseren mehrstufigen Transformer. Aufgrund der inhärenten Kausalität in dynamischen Systemen ist eine explizite Einbeziehung von Zeitinformationen über Positionskodierungen möglicherweise nicht notwendig. Das Lernen nichtlinearer Einbettungen könnte die Vorhersagegenauigkeit verbessern, indem relevante Daten aus der ursprünglichen Eingabe extrahiert werden, ähnlich wie bei Worteinbettungen, die für Transformer in NLP verwendet werden.

Wir evaluieren unsere Methode am Anwendungsfall der Vorhersage von Schiffsbewegungen unter Umwelteinflüssen. Wir können mit unserem Transformer keine Vorteile in Bezug auf die Trainingszeit im Vergleich zu LSTMs nachweisen, erreichen aber eine ähnliche Vorhersagegenauigkeit. Wir zeigen, dass Positionskodierungen einen nachteiligen Effekt auf die Vorhersagegenauigkeit von Transformern für mehrstufige Vorhersagen haben, was beweist, dass der Transformer kausale Informationen direkt aus der Eingabe ableitet. Nichtlineare Einbettungen von Steuereingängen und Anfangszustand führen zu einer vernachlässigbaren Verbesserung der Vorhersagegenauigkeit.

Contents

1 Introduction **7**

2 Neural Networks **9**
2.1 Feedforward Neural Networks 9
2.2 Recurrent Neural Networks 11
2.3 Transformers 13

3 Related Work **17**
3.1 LSTMs for Multistep Prediction 17
3.2 Transformers 17
3.3 Variants of Transformers 18

4 Multistep Transformer **19**
4.1 Problem Definition 19
4.2 Research Questions 20
4.3 Models 21

5 Evaluation **23**
5.1 Dataset 23
5.2 Evaluation Process 24
5.3 Results 24

6 Conclusion **27**

Bibliography **29**

1 Introduction

Multistep prediction is the prediction of multiple future states in a time series based on its previous values and additional external control inputs [5]. Applications for multistep prediction with such control inputs include simulation, predictive maintenance or fault detection of technical systems [8, 19]. Another use case is model-predictive control (MPC), where the immediate future of a system like a vehicle is predicted to choose and control its path. MPC has high performance requirements regarding the predictive models speed and accuracy. Because MPC uses the predictive model to make multiple predictions with different control inputs to find the best one to get the vehicle or system to a desired state, the predictions have to be much quicker than for a simulation scenario where prediction accuracy is usually much more important than speed. More accurate and time-efficient multistep prediction models can enable the simulation and control of complex systems, which is of particular interest in the development towards more automation in many industries.

Current approaches for multi-step prediction use different types of models, such as physically accurate simulation models which are usually very difficult to create for nontrivial systems and often require vast amounts of computational resources [27]. They are therefore in most cases not suited for model-predictive control. Alternatively, linear models allow cheap but inaccurate approximations of the system, which are suitable for some less complicated environments[7].

Deep learning is used for approximation when the system dynamics are non-linear and external disturbances have a strong impact on the dynamics [18, 19, 21]. An example for such a complex problem is the prediction of the movement of a ship in harsh weather conditions. Here the nonlinear and stochastic processes of the hydrodynamic interactions of the ship hull with the surrounding water can often only be approximated. Other environmental influences cannot be measured at all. The current state-of-the-art deep learning method for multistep prediction is the use of an LSTM network. While this network architecture fares better than older models like feed-forward networks or linear models in terms of prediction accuracy [19], it has to be trained sequentially [12], without the possibility to parallelize the process which results in long training times.

This thesis proposes the use of transformers for multistep prediction. Recent results in natural language processing [28] suggest that the transformer model may also be able to achieve better multistep prediction accuracy than an LSTM. We adjust the standard transformer model to include control inputs that are independent from the states at each time step.

Our implementation of a transformer model was not able to outperform an LSTM overall but it did have a better prediction accuracy for one of the predicted variables.

The contributions of this paper can be summarized as follows:

1. The transformer architecture is adapted for multistep prediction with control inputs.

2. Different variants of the transformer model are evaluated and compared to determine whether an nonlinear embedding is beneficial and whether the model is able to infer the order of causally linked states in a sequence without positional encoding.
3. The transformer model is compared to a state-of-the-art LSTM w.r.t. prediction accuracy at the example of the prediction of a ship state

2 Neural Networks

Artificial neural networks have become the de-facto standard for most of machine learning applications. This chapter explains the different types of neural networks that we will use in our implementation or that will be relevant as a comparison baseline for this thesis. Section 2.1 explains feedforward neural networks and how neural networks are trained. Section 2.2 describes recurrent neural networks like LSTMs. Finally in section 2.3 we describe the standard transformer architecture.

2.1 Feedforward Neural Networks

Neural Networks are comprised of one or multiple layers. A typical layer consists at least of a weight matrix W and a bias vector b. The input x of a layer is multiplied by the matrix and the result subsequently added to the bias vector. This results in the output y of the layer:

$$y = Wx + b$$

Because this is an affine linear operation, networks with such layers would be limited to approximating only linear functions. To prevent this, some linear layers use a nonlinear **activation function** which is applied to the output of the linear operation. The most commonly used activation function today is the rectifier function f_{ReLU}, which returns zero for all negative inputs and is the identity function for all positive inputs:

$$f_{ReLU}(x) = max(0, x)$$

A unit in a network that employs this function is called a 'rectified linear unit' (ReLU).

Other common activation functions are the sigmoid function $\sigma(x)$ and $tanh(x)$.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

The sigmoid function maps values to the interval (0,1) while $tanh(x)$ maps to the interval (-1,1).

Because ReLUs are partwise linear, gradients computed for networks using ReLUs scale mostly linearly, which is useful when training the network. In most cases, ReLUs keep the training simple and efficient while sigmoid and $tanh$ activation functions are used when the output of those functions is used in further calculations that require a bounded value interval.

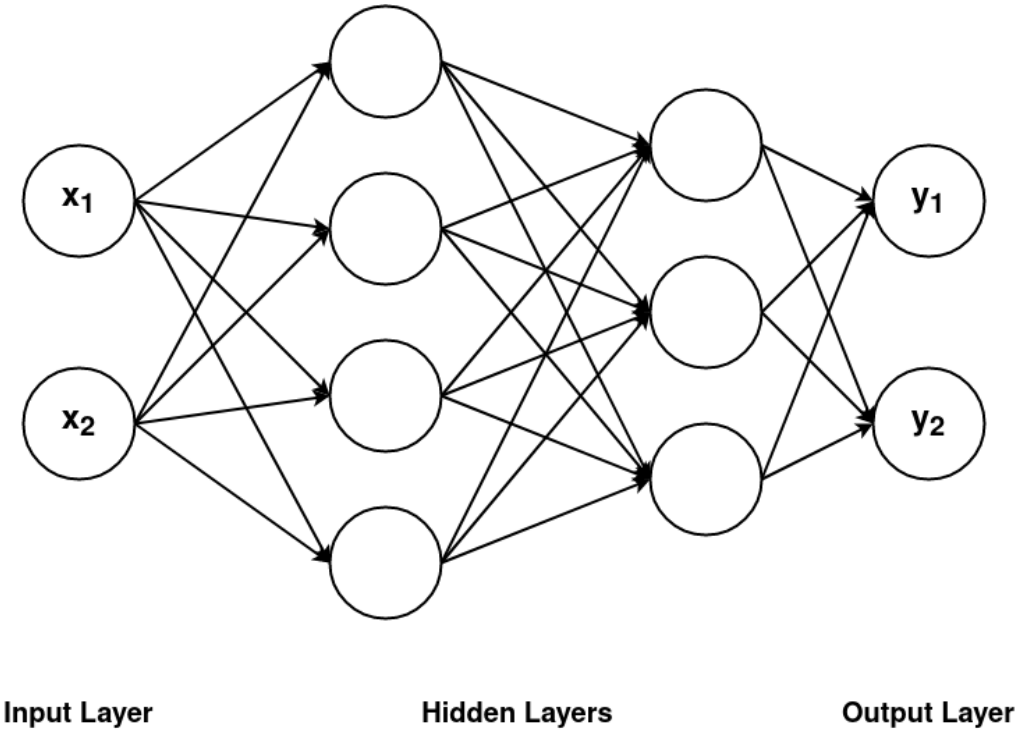

Figure 2.1: Example architecture of a feedforward neural network

Neural nets are comprised of multiple layers, that feed into each other. Depending on their intended purpose, networks differ in the size, number, type and order of their layers. A **Feedforward Neural Network (FFN)** combines multiple linear layers of which most use nonlinear activation. Figure 2.1 shows how multiple linear layers form a feedforward neural network. It is common to use nonlinear activation in the hidden layers but not the output layer.

Depending on the weights of a model its output changes. A loss function $\mathcal{L}$ calculates a distance metric between the output of a model and a desired ground truth y. A widely used loss function is the mean squared error (MSE) function which sums the squared differences between our desired output y and our actual output $\hat{y}$ over each of their n dimensions:

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

The goal of training a model is to minimize the loss function which requires the weights to be modified in a way that the model output $\hat{y}$ is closer to the desired output y. For this purpose a training dataset containing pairs of inputs and desired outputs is required.

In general, the optimization objective of a neural network implementing the function N_θ with weights θ, a training dataset S and a loss function L is defined as:

$$\theta^* = \underset{\theta \in \Theta}{\operatorname{argmin}} \mathcal{L}(y, N_\theta(x)) \quad \forall (x, y) \in S$$

The standard method of optimizing a loss function is the **gradient descent** method. Gradients for gradient descent are computed via **backpropagation** and used to update the collective weights θ weighted with a so-called learning rate λ step by step according to the following formula:

$$\theta_{t+1} = \theta_t - \lambda \nabla \mathcal{L}(y, N_\theta(x))$$

Backpropagation feeds the training input x into the model to compute the models predicted output $\hat{y}$ and the resulting loss. We then compute partial derivatives regarding the loss for each of the weights which together form a gradient.

This process exploits the fact that a neural network with n layers and an activation function f_{act} implements a nested function chain:

$$\hat{y} = W_1 f_{act}(W_2 f_{act}(...W_n x + b_n...) + b_2) + b_1$$

Using the chain rule we can compute the gradient for one layer at a time and propagate the loss gradient through the whole network [15]. A layer at depth i has a layer input $in_i = W_i o_{i-1} + b_i$ and a layer output $o_i = f_{act}(in_i)$. Backpropagation calculates the gradients as follows [15]:

$$\frac{\partial \mathcal{L}_i}{\partial W_i} = \frac{\partial \mathcal{L}_i}{\partial o_i} \cdot \frac{\partial o_i}{\partial in_i} \cdot o_{i-1}$$
$$\frac{\partial \mathcal{L}_i}{\partial o_{i-1}} = W_i^T \cdot \frac{\partial \mathcal{L}_i}{\partial in_i}$$

Adding more layers increases the number of calculations that is required during training but enhance the models ability to approximate complex functions [17] by learning different abstractions of input features. This means that a network may learn to recognize basic attributes about the input in its first few layers and use this knowledge to make more complex distinctions at a higher level of abstraction. Learning with a multilayered neural network is also called 'deep learning'. Because the gradient that is calculated for the weights of each layer has a more direct changing impact on the last layers of the network compared to earlier layers, very deep networks with many layers are difficult to train. This can be prevented by integrating skip connections, that take the output of one layer and feed it not into the next layer but add onto the output of a later layer. Besides fixing the vanishing gradient problem this also has the added benefit that the model can learn to ignore layers that are not necessary for the problem which increases the models accuracy.

Besides the basic linear layers, activation function and skip connections, there are several other building blocks neural networks use for specific purposes.

2.2 Recurrent Neural Networks

A widespread machine learning approach to calculate an output sequence from an input sequence are Recurrent Neural Networks (RNNs) whose structure is designed to work on sequential data. While calculating the predicted output RNNs keep a copy h_t of this output vector, which serves as a

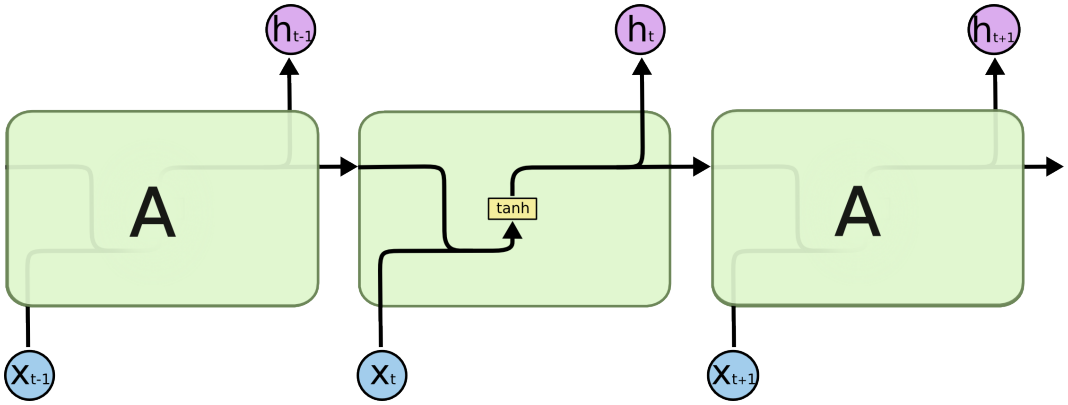

Figure 2.2: Architecture of an RNN cell [22]

memory of past inputs and is used together with a new input to generate the output of the next time step. This enables the model to calculate the output not only based on the last input but also on several inputs before that [10, 12]:

$$h_t = \tanh(W_h \cdot [h_{t-1}, x_t] + b_h)$$

The state-of-the-art RNN architecture for sequential prediction tasks is the **Long Short-Term Memory neural network (LSTM)** [25]. LSTM cells calculate a memory cell vector C_t and a hidden state vector h_t which are fed back into the LSTM in the next timestep alongside the new input. The following equations describe the forward computation of the LSTM cell:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.1}$$
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2.2}$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{2.3}$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{2.4}$$
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{2.5}$$
$$h_t = o_t * \tanh(C_t) \tag{2.6}$$

An LSTM cell performs multiple learned separate calculations on those vectors. First the output of the last step and the new input are concatenated. Then a layer with a sigmoid activation function calculates a mask vector f_t of values between zero and one (2.1). This vector is then multiplied element wise with the memory vector C_{t-1} coming from the last (2.4). Doing this effectively 'forgets' parts of the memory which were multiplied by values close to zero while other parts are multiplied by values close to 1 and are therefore retained.

Two other layers are fed the same inputs, one activated by a sigmoid (2.2) and one activated by a tanh function (2.3). The resulting vectors are multiplied to again retain only the parts that the sigmoid output i_t doesn't set to zero and then added to the memory vector (2.4). The memory vector C_t is now fully updated and is fed forward for the calculation of the next timestep. A copy of

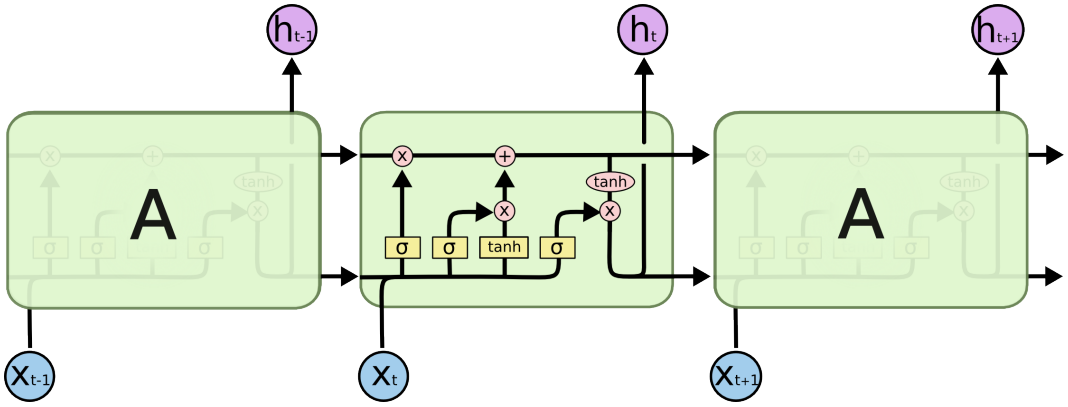

Figure 2.3: Architecture of an LSTM cell [22]

this memory vector is however multiplied with another sigmoid output o_t of the two other inputs. The resulting vector h_t is the actual output and hidden state of the LSTM for this timestep. The combination of these calculation steps enables the model to control exactly which parts of the inputs and the memory are combined.

The main benefit of an LSTM over conventional RNNs is its ability to decide whether it forgets some parts of the memory and whether it adds its output to the memory. This increases the span of inputs which the model can effectively keep in its memory, which enables the LSTM to capture long-term dependencies in sequential data.

The controlled forgetting also prevents the buildup of large errors by blocking irrelevant inputs and noise from entering the cell, improving the training convergence [10, 13]. The memory of an LSTM is however still not perfect as the capacity of the hidden state vector is limited and over time most information about past inputs is displaced by newer information.

2.3 Transformers

Transformer models use self-attention to learn dependencies between different inputs. Self attention works by first calculating a key, query and value for every input. This is achieved by learning three matrices W_Q, W_K and W_V, which together make up an attention head. W_Q is multiplied with an input vector x to generate the query q, W_k is multiplied with an input vector x to generate the key k and W_V is multiplied with an input vector x to generate the value v.

$$q = W_Q x$$
$$k = W_K x$$
$$v = W_V x$$

Transposing these vectors into a row, combining all q vectors into a Matrix Q and doing the same for the k and v vectors yields the Matrices Q, K, V.

The output of one attention head is calculated using a softmax function and normalization by dividing with the square root of the key dimension:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{dim_k}})V$$

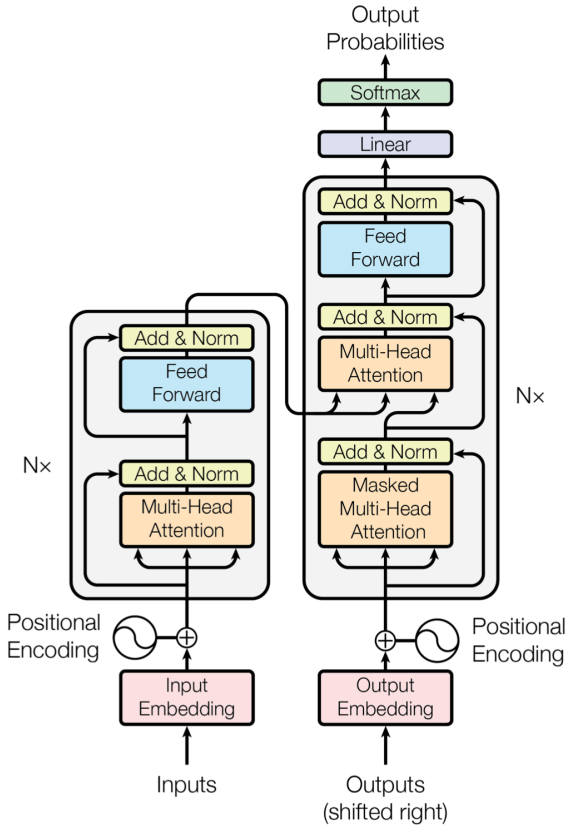

Figure 2.4: Scheme of the transformer architecture [28]

Figure 1 shows the original transformer architecture. The elements of the input sequences, which are words of a sentence in the NLP case, are embedded into a vector space. In language processing this is often achieved by a pretrained embedding network.

Out transformer models will instead use a vector embedding that they learn themself with either a simple linear transformation layer or a full FFN which is trained together with the rest of the model.

After that a positional encoding (PE) is added, which retains data about the relative position of the elements in the input sequence vector to each other [28]. For NLP models, the 'position' (pos) means the place of the word in the input sequence. Here this positional encoding is crucial for model functionality, because the meaning of a sentence can change completely when the same

words are put in a different order. The following is added as positional encoding to all features at position n in the embedded input vectors of length d at the position pos in the sequence:

$$PE(pos, n) = \begin{cases} sin(pos/10000^{\frac{2i}{d}}) & n = 2i, \quad i \in \mathbb{N} \\ cos(pos/10000^{\frac{2i}{d}}) & n = 2i + 1, \quad i \in \mathbb{N} \end{cases}$$

The encoded vectors feed into the main components of the transformer, the encoder and decoder blocks, which each contain a feed forward layer and a multi-head attention layer [28]. In each 'head' of the multi-head attention layer the attention is computed separately in parallel. This allows different attention distributions on different parts of the input vector to be passed on simultaneously. Every time step and all words are compared with each other. The output of the attention blocks feed in an FFN and finally in a softmax layer that outputs the normalised probabilities for the possible predictions.

Our implementation of the transformer network differs from this implementation as described in section 4.3.

3 Related Work

This chapter will summarize the previous scientific work and the current state-of-the-art for multistep prediction and the transformer architecture.

3.1 LSTMs for Multistep Prediction

Recurrent architectures, such as LSTMs, are prominently used for modeling of time series including multistep prediction. Successful applications of LSTMs for multistep prediction are found for example in financial time series prediction [26] and have also been used for system identification and the prediction of dynamic systems [1, 19]. Depending on the specific problem, the original LSTM architecture is augmented with techniques like hidden state initialization, where the initial hidden state is computed via a separate neural network rather than initialized as zero or randomly. Stacking multiple LSTMs in hidden layers provides additional abstraction capabilities and gives the model an inherent view on time frames of different sizes [12].

3.2 Transformers

The transformer architecture has shown great improvements in terms of accuracy and training time for natural language processing tasks compared to RNNs [28]. Because it does not share the basic recurrent structure of RNNs, it is more readily parallelised and therefore provides great improvements in training time. This acceleration is possible without a trade-off in terms of prediction accuracy, with the transformer based models GPT-2 and its successor GPT-3 being two of the best generative natural language models to date [3, 24]. Its main advantage over other neural networks lies in the multi-head attention layers, that generate a mapping of relevance between different parts of the input. This happens regardless of the actual distance in the input vector, which is why the transformer can relate inputs to each other over longer time frames than an RNN.

Although the transformer architecture has in the three years since its first publication mainly been recognized for its breakthrough performance in NLP, it has already been applied on several other problems. These include image generation [23], where the local self attention aspect of the encoder and decoder blocks was modified to be fed the individual pixel channels of an image. Another recent application is the time series forecasting of the prevalence of influenza-like illness [29]. Here the original transformer architecture remained unchanged.

Transformers have also been used for traffic prediction [4], where they replaced a combination of convolutional and recurrent neural networks as the state-of-the-art.

Transformers have also been combined with generative adversarial networks (GANs) for time series forecasting [30]. The Transformer serves as the generator and can attain smaller error accumulation on sequential data than a traditional GAN.

The original transformer implementation has been used to predict the trajectory of human social behaviour [11] where it outperformed an LSTM model.

3.3 Variants of Transformers

A more efficient variant of the transformer, the reformer, builds upon the transformer architecture but achieves similar results with much smaller time and memory requirements for long sequences [14]. In a similar attempt Li et al. restricted the attention layers to only local attention which improved local awareness and memory requirements [16]. While this is important for language applications, where the input sequences might consist of hundreds of words, the runtime improvements are negligible for the comparably short input sequences of our vehicle state prediction task.

Another proposed improvement to the transformer architecture is the adaptively sparse transformer which aimed to deliver the same performance as the regular transformer while lowering space requirements [6]. This is achieved by replacing the softmax functions in the transformer with an α-entmax function which dynamically sets very low weights to zero.

The bulk of the research regarding transformer networks has been on natural language processing. And while some successful work promises a wide array of other beneficial applications, so far the research is thin on multistep prediction with exogenous inputs. This thesis is aimed to clear up whether or not transformer models can replace LSTMs in this respect.

4 Multistep Transformer

In this chapter we describe our developed multistep transformer and baselines. In section 4.1 we formally define multistep prediction, in section 4.2 we list the research questions we wish to answer with this thesis and in section 4.3 we explain all our models that will be compared in the evaluation.

4.1 Problem Definition

In (singlestep-)prediction a nonlinear prediction function F with exogenous inputs relates a predicted future state $\hat{s}_{t+1}$ in a time series to both recent states s_i of this time series and recent states c_i of an series of control inputs.

$$\hat{s}_{t+1} = F(s_{t-n}, s_{t-n+1}, ..., s_t, c_{t-n}, c_{t-n+1}, ..., c_t)$$

In prediction problems this function F has to be identified. It can be any linear or nonlinear function and therefore may be replaced by anything from a simple lookup table to a range of different machine learning models like the proposed transformer model. For multistep prediction or system identification frequently used models have been physics engines and RNNs.

To use F for multistep prediction, one can follow an approach called multi-stage prediction where the outputs of F are used as another input state in a series of additional prediction steps, which forms a feedback loop that theoretically can predict the state at any future time step.

$$\hat{s}_{t+2} = F(s_{t-n+1}, s_{t-n+2}, ..., s_t, \hat{s}_{t+1}, c_{t-n+1}, c_{t-n+2}, ..., c_t, c_{t+1})$$
$$\hat{s}_{t+3} = F(s_{t-n+2}, s_{t-n+3}, ..., \hat{s}_{t+1}, \hat{s}_{t+2}, c_{t-n+2}, c_{t-n+3}, ..., c_{t+1}, c_{t+2})$$
$$\hat{s}_{t+4} = ...$$

Because the estimation of F likely yields some error for any prediction, the accuracy of the predictions will decrease for a larger number of prediction steps, when multiple prediction errors accumulate. To construct our transformer model for multistep prediction with control inputs we will first retain as much of the structure used for the language generation or the influenza prediction as possible, which means the code provided in the original transformer paper [28] will be only slightly altered to fit in the evaluation framework.

The optimisation goal for all our models is to minimize their loss function. We consider two different loss functions, starting with the widely used mean squared error (MSE). The waves in our ship scenario cause oscillations in the state values that the models have to predict, especially the roll angle of the ship. Using the MSE can lead to models predicting a constant value in these outputs and ignore its oscillating properties. To prevent this, we can compute the MSE on the gradient of the predicted values instead of on the values themselves. This mean squared gradient error (MSGE)

incentivises the model to capture the dynamics of oscillations more thoroughly [2]. The MSE and the MSGE are calculated for a prediction horizon of H future states as follows:

$$MSE = \frac{1}{H}\sum_{t=1}^{H}(s_t - \hat{s}_t)^2$$

$$MSGE = \frac{1}{H}\sum_{t=1}^{H}\big((s_t - s_{t-1}) - (\hat{s}_t - \hat{s}_{t-1})\big)^2$$

Preliminary testing showed no benefit for using the MSGE to train our transformer models so they all use the MSE as a loss function. The MSGE is used to train our baseline LSTM model [2].

4.2 Research Questions

This thesis aims to answer the following research questions:

1. Does the transformer architecture perform better than a LSTM for multistep prediction? We expect it to have comparable or better accuracy and to its parallel architecture we expect the transformer model to also have shorter training times.
2. Does a nonlinear transformation of the input (by a conventional feed-forward neural network (FFN)) provide any benefit over a simple linear mapping to the hidden dimension of the model? In the original use case of transformers for NLP pretrained embeddings were used in the input embedding layer. As such embeddings do not exist for our use case, we want to determine whether FFN serve as a viable replacement for the input embedding. Equal or better performance than the baseline can be expected because a FFN can learn to behave the same way as any linear mapping while also being able to learn nonlinear transformations.
3. Is a positional encoding of the input vectors necessary in a prediction setting of a physical system? In other words, can a transformer learn physical causality from data? The underlying physical regularities of the ship simulation are causal in the mathematical sense, meaning a value is dependant solely on values and events that occurred before it. The relative temporal position of the input states towards each other may therefore already be encoded in their capture of physical events, without applying the additional positional encoding. This means it is possible that the model learns to put the inputs in a temporal order by itself without loss in prediction performance. For this we expect anything from equal to far worse performance without positional encoding.

Comparing the baseline model and the FFN variant will provide an answer to the research question 2 and the results of the No-PE variant will be compared to the baseline to answer research question 3. The best preforming of the four transformer models will be compared to the LSTM model to answer the research question 1. The results allow us to conclude if the transformer architecture performs favorably in this setting and if the positional encoding is beneficial in this case.

4.3 Models

This section describes the linear and LSTM model used as baseline, as well as our multistep transformer and its variants.

4.3.1 Linear Model

We use a simple linear model as a baseline to compare to the performance of our more complex models. This model consists of only one linear layer as defined in section 2.1 with no activation function.

The input of this linear model consists of several states $s_{i-m}, ..., s_i$ and control inputs $c_{i-m}, ..., c_i$ for an input window of size m, which are all concatenated into one single vector. The model output is the predicted next state s_{i+1}. The linear model consists only of a weight matrix W_l and bias vector b_l and uses no activation function:

$$s_{t+1} = W_l \left[s_{t-m}, ..., s_t, c_{t-m}, ..., c_t \right]^\top + b_l$$

We train our linear model for 500 epochs using the Adam optimizer. We use gradient descent instead of analytical solvers to get a more comparable training time baseline for our evaluation.

4.3.2 LSTM Model

As a second baseline for comparison and a representative for the state-of-the-art we use an LSTM model as described by Mohajerin et al. [20]. We apply the specific implementation and hyperparameters from Baier er al. [2]. This approach extends the basic LSTM architecture with an initialization mechanism. A second LSTM computes the initial hidden state h_t and cell state C_t given an initial window of control input and states.

The function F_{LSTM} that represents this recurrent model looks as follows:

$$s_{t+1}, h_{t+1}, C_{t+1} = F_{LSTM}(s_t, c_t, h_t, C_t)$$

The depth of this LSTM model is 2 layers and it has a hidden dimension of 192. [2] The loss function MSGE (defined in Section 4.1) is used together with Adam optimizer.

4.3.3 Transformer models

Initially our intention for our transformer models was to closely follow the approach by Vaswai et al. in their paper on the original transformer architecture [28]. The encoder-decoder architecture however proved difficult to train for our use case and models would not converge. We simplified the transformer model to only contain an encoder block. With this modification the model was not too deep to converge effectively. Additionally we added a linear output layer to the encoder that concatenates all individual outputs of the multi-head attention layers and transforms them into our

output dimension. Now the transformer model can be utilized similar to the linear model with the following representative Function $F_{transformer}$:

$$s_{t+1} = F_{transformer}(s_{t-m}, ..., s_t, c_{t-m}, ..., c_t)$$

Here the inputs are again the previous m states and control inputs, however each state/control-pair is fed into the model as part of a sequence, which is processed in parallel.

There will be four variations of the transformer model:

1. **Baseline transformer**

   An encoder-only transformer model as described above. Additionally the input embedding is not performed by a pretrained network like it is common in NLP models but instead by a linear layer that is trained with the rest of the model.

2. **FFN variant**

   A variant of the baseline model where the linear layer is replaced by an FFN, which enables a similar mechanism to the embedding used in many NLP models. This model will be used to answer research question 2.

3. **No-PE variant**

   A variant of the baseline model without the positional encoding, where the inputs are fed in the embedding layer without any preprocessing. This means that all of the previous states that make of the input of this calculation step are fed simultaneously into the model without explicitly encoding the order they occurred in. The model will only be able to make sense of the inputs if it can place them into their correct order by relying on the causal nature of the dynamical system. This will be useful to determine the utility of a positional encoding in this use case and answer research question 3.

4. **FFN+No-PE variant**

   A variant that combines the modifications of the FFN variant and the No-PE variant in a single model.

All possible combinations of our variants are therefore represented in their own model. The best performing transformer variant is then compared to the baseline models. Each transformer variant is trained for 500 epochs and uses the Adam optimizer.

5 Evaluation

This chapter describes the dataset used for training and evaluation, how hyperparameters of the transformer model variants were optimized and the evaluation process and its results. We evaluate the research questions stated in Section 4.2 given our evaluation results.

5.1 Dataset

Our dataset is a set of simulations of a ship in open water generated by Baier et al. [2]. To enable modeling wave-induced motion these simulation use a 4-DOF ship model. We only train a 3-DOF model on this data because waves are not measurable or predictable for long prediction horizons and predicting them is not the focus of our implementation. The ship executes random maneuvers and is subject to environmental disturbances in the form of wind and waves. Each simulation is initialized with a different wave height and wind speed. Every second, the ship state is captured by sensors measuring the ship surge velocity u, sway velocity v and yaw rate r as shown in Figure 5.1

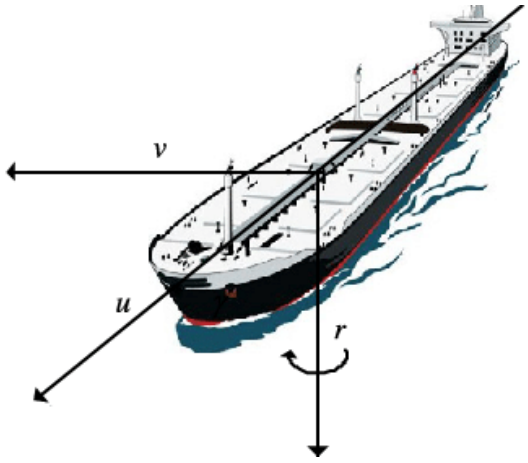

Figure 5.1: The body-fixed coordinate frame of a ship for movement in a plane. Our models predict the surge velocity u, the sway velocity v and the yaw rate r of the ship. [9]

Additionally the control inputs for the angle and rotation speed of two rudder motors are also captured every second. The ship states together with the control inputs over 1 hour form one simulation with 3600 discrete data points.

The dataset consists of 96 simulations of 1 hour each. This is further split up into a training dataset of 86 simulations and a test dataset of 10 simulations. 26 of the simulations in the training dataset were used for validation during training.

Before training and inference the data is normalized using z-score normalization $z(x) = \frac{x-\mu}{\sigma}$. The means μ and standard deviations σ used for the normalization are calculated using only the training dataset.

5.2 Evaluation Process

Our models predict the 3 state variables for a prediction horizon of 60 seconds (which translates to 60 predicted states) with an input window of 20 seconds. We separate the test dataset into sections of 60 seconds and for each of these sections we compare our prediction to the ground truth. This is done by calculating the sum of the mean absolute errors (MAE) over all 60 predicted states for all 3 state variables. The metric we use for our evaluation is the average of these summed MAEs. The MAE for a ground truth y and a prediction $\hat{y}$ of dimension n is defined as follows:

$$MAE(y, \hat{y}) = \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{n}$$

We search for optimal hyperparameters of the baseline transformer model using grid search. The hyperparameters optimised by this search are the number of multi-head attention layers (model depth) and the hidden dimension (model width) of the transformer network. The performance measure used to determine the best model is the sum of the MAEs of all 3 state variables over a separate dataset. To weigh the MAEs of each of the state variables equally the errors were calculated on the z-score-normalized data.

The hyperparameter search showed a slight performance benefit for larger model widths up to 512 and no benefit for larger model depth. For more than 4 layers the model failed to converge during training. The best performing model had only one multi-head attention layer and a hidden dimension of 512. These hyperparameters are also used for the other transformer variants.

We compare the best transformer model to our linear model as a simple baseline and the LSTM model as a state-of-the-art deep learning baseline.

5.3 Results

All four transformer variants are compared to find the best transformer model. We use the above mentioned average unnormalized MAE over the whole test dataset as a performance metric. This time the MAEs of the three state values of a prediction are calculated after denormalization and evaluated separately instead of in a sum like in the hyperparameter search. The best performing transformer variant is afterwards compared to the linear model and the LSTM model.

Variant	u [m s^{-1}]	v [m s^{-1}]	r [rad s^{-1}]
`Baseline transformer`	0.1897	0.0762	**0.0022**
`FFN variant`	0.1409	0.0645	0.0026
`No-PE variant`	0.2069	0.0692	0.0029
`FFN+No-PE variant`	**0.1226**	**0.0601**	**0.0022**

Table 5.1: MAE of the 3 state variables for four transformer variants

5.3.1 Ablation Study

We compare the four transformer variants with regards to their prediction accuracy to determine whether the modifications present in the variants are of any benefit. Table 5.1 shows the rounded MAE of the 3 predicted state variables for all four transformer variants we tested.

Both variants without positional encoding performed better than the equivalent models with positional encoding, with the FFN+No-PE variant being the best performing transformer model out of the four. This strongly suggests that a positional encoding of the input vectors is unnecessary and therefore answers research question 3.

This result is unexpected because positional encoding is essential when used in transformer NLP models. Here it seems to be unnecessary and even a hindrance for the models to learn the order of the state and control inputs. This is indicative of the models capability to infer the order of the input sequence solely based on the physical causalities in the input data.

The FFN variant performed worse than the baseline transformer model while the FFN+No-PE variant performed slightly better than the No-PE variant. We cannot conclude either way on the benefit of the FFN as input embedding and therefore cannot answer research question 2. However the embeddings appear to be less important for model performance in this case than in NLP transformer models.

5.3.2 Comparison to baseline

The FFN+No-PE variant is now compared to our linear model and the LSTM model. Table 5.2 shows the MAE of the predictions for all 3 model types over the test dataset.

Model	u [m s^{-1}]	v [m s^{-1}]	r [rad s^{-1}]
`Linear model`	0.3108	0.0999	0.0037
`LSTM model`	**0.0893**	0.0818	**0.0013**
`FFN+No-PE variant`	0.1226	**0.0601**	0.0022

Table 5.2: MAE of the 3 state variables for the FFN+No-PE variant, our linear model and the LSTM model

Both our transformer model and the LSTM model performed significantly better than the linear model across all predicted values. These results are expected because these models are much larger as seen in table 5.3 and unlike the linear model capable of learning nonlinear dependencies.

For two of three state variables (surge velocity u and yaw rate r) the LSTM produced more accurate predictions than the transformer model. For one of the three (sway velocity v) however the transformer models predictions were more accurate. While this result still places the LSTM model as overall better performing, is is possible that further research and optimization may improve the transformer model beyond the performance of the LSTM.

Table 5.3 shows the parameter count, training epochs and training time of all models.

Model	parameter count	epochs	training time [min]
`Linear model`	603	500	30
`LSTM model`	898950	1000+400	9
`FFN+No-PE variant`	3452419	500	84

Table 5.3: Parameter count, training epochs and training time of the FFN+No-PE variant, the linear baseline and the LSTM baseline. All models were trained on a Nvidia A100 GPU. The training epochs for the LSTM are given as 400 for the initializer network and 1000 for the predictor network respectively. Because the LSTM uses a different formatting for the training data with less redundant data, the training time for the LSTM cannot be reasonably compared to that of the other models.

The transformer model has almost 4 times as many parameters as the LSTM model and more than 5000 times as many parameters as the simple linear model. The training time for the LSTM was much shorter than either of the other models, this is however due to a difference in the sampling of the training data: Each epoch the same training data was used to train all models, but the linear and transformer model used it with much more redundancy than the LSTM model. This difference allows no conclusions about the benefit in training time a transformer model for multistep prediction may have over a LSTM model.

Training time of the linear model could have been reduced with an analytical optimization approach instead of gradient descent, but having the same training method and the same number of epochs the transformer model can be compared easily to the linear model. Despite having more than 5000 times more parameters the transformer model only had a 2.8 times longer training time. This is likely due to the parallelization that is possible for training transformer models.

Our results provide no definitive answer to research question 1. In future work it would be beneficial to compare the two model architectures across additional different multistep prediction domains. Both LSTM and transformer models seem to be viable options for multistep prediction applications.

6 Conclusion

We adjusted the transformer architecture for multistep prediction. Our implementation replaced the standard encoder-decoder architecture and featured only an encoder with variants using nonlinear input embedding and positional encoding. We evaluated the performance of our transformer models against a linear model and an LSTM model by comparing their multistep prediction accuracy for a ship simulation.

We can answer our research questions as follows:

1. The transformer architecture is a viable candidate for multistep prediction. Although our encoder-only transformer could not outperform the state-of-the-art LSTM model overall, it was able to predict the sway velocity v more accurately than the LSTM. This suggests that with further optimization a better result with a transformer than with the LSTM might be achievable.
2. Using a FFN as nonlinear input embedding did offer a small improvement over the linear embedding, but only when combined with a removal of the positional encoding. The variant with positional encoding and FFN performed the worst of all variants. Compared to the importance of embeddings in transformer models for NLP, the effect of the nonlinear embedding was negligible for multistep prediction.
3. Contrary to what is seen in NLP applications, a positional encoding did decrease our model performance consistently. The transformer model was able to infer the order of its input sequence without any positional encoding using only the physical causality present in the data. This finding indicates that transformer models for most physical simulation or modelling applications likely also do not require positional encoding.

Future work could examine whether other forms of positional encoding can yield performance benefits and how well the ability to reconstruct time sequences based on temporal causality translates to other physical systems.

Bibliography

[1] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, S. Savarese. "Social LSTM: Human trajectory prediction in crowded spaces". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 961–971 (cit. on p. 17).

[2] A. Baier, Z. Boukhers, S. Staab. "Hybrid Physics and Deep Learning Model for Interpretable Vehicle State Prediction". In: *CoRR* abs/2103.06727 (2021). arXiv: 2103.06727. URL: https://arxiv.org/abs/2103.06727 (cit. on pp. 20, 21, 23).

[3] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei. "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901. URL: https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf (cit. on p. 17).

[4] L. Cai, K. Janowicz, G. Mai, B. Yan, R. Zhu. "Traffic transformer: Capturing the continuity and periodicity of time series for traffic forecasting". In: *Transactions in GIS* 24.3 (2020), pp. 736–755 (cit. on p. 17).

[5] H. Cheng, P.-N. Tan, J. Gao, J. Scripps. "Multistep-ahead time series prediction". In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer. 2006, pp. 765–774 (cit. on p. 7).

[6] G. M. Correia, V. Niculae, A. F. Martins. "Adaptively sparse transformers". In: *arXiv preprint arXiv:1909.00015* (2019) (cit. on p. 18).

[7] J. G. De Gooijer, R. J. Hyndman. "25 years of time series forecasting". In: *International Journal of Forecasting* 22.3 (2006), pp. 443–473 (cit. on p. 7).

[8] P. Filonov, A. Lavrentyev, A. Vorontsov. "Multivariate industrial time series with cyber-attack simulation: Fault detection using an lstm-based predictive data model". In: *Thirtieth Conference on Neural Information Processing Systems (NIPS). Time Series Workshop*. 2016 (cit. on p. 7).

[9] T. I. Fossen. *Handbook of marine craft hydrodynamics and motion control*. John Wiley & Sons, 2011 (cit. on p. 23).

[10] F. Gers, J. Schmidhuber, F. Cummins. "Learning to forget: continual prediction with LSTM". In: *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*. Vol. 2. 1999, 850–855 vol.2. DOI: 10.1049/cp:19991218 (cit. on pp. 12, 13).

[11] F. Giuliari, I. Hasan, M. Cristani, F. Galasso. "Transformer networks for trajectory forecasting". In: *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE. 2021, pp. 10335–10342 (cit. on p. 18).

[12] A. Graves, A.-r. Mohamed, G. Hinton. "Speech recognition with deep recurrent neural networks". In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE. 2013, pp. 6645–6649 (cit. on pp. 7, 12, 17).

[13] S. Hochreiter, J. Schmidhuber. "LSTM can solve hard long time lag problems". In: *Advances in neural information processing systems*. 1997, pp. 473–479 (cit. on p. 13).

[14] N. Kitaev, L. Kaiser, A. Levskaya. "Reformer: The Efficient Transformer". In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL: https://openreview.net/forum?id=rkgNKkHtvB (cit. on p. 18).

[15] Y. A. LeCun, L. Bottou, G. B. Orr, K.-R. Müller. "Efficient backprop". In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48 (cit. on p. 11).

[16] S. Li, X. Jin, Y. Xuan, X. Zhou, W. Chen, Y.-X. Wang, X. Yan. "Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting". In: *arXiv preprint arXiv:1907.00235* (2019) (cit. on p. 18).

[17] S. Liang, R. Srikant. "Why deep neural networks for function approximation?" In: *arXiv preprint arXiv:1610.04161* (2016) (cit. on p. 11).

[18] N. Mohajerin, M. Mozifian, S. Waslander. "Deep learning a quadrotor dynamic model for multi-step prediction". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 2454–2459 (cit. on p. 7).

[19] N. Mohajerin, S. L. Waslander. "Multistep prediction of dynamic systems with recurrent neural networks". In: *IEEE transactions on neural networks and learning systems* 30.11 (2019), pp. 3370–3383 (cit. on pp. 7, 17).

[20] N. Mohajerin, S. L. Waslander. "State initialization for recurrent neural network modeling of time-series data". In: *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2017, pp. 2330–2337 (cit. on p. 21).

[21] O. Ogunmolu, X. Gu, S. Jiang, N. Gans. "Nonlinear systems identification using deep dynamic neural networks". In: *arXiv preprint arXiv:1610.01439* (2016) (cit. on p. 7).

[22] C. Olah. *Understanding LSTM Networks*. Aug. 27, 2015. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (cit. on pp. 12, 13).

[23] N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, N. Shazeer, A. Ku, D. Tran. "Image Transformer". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Ed. by J. G. Dy, A. Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 4052–4061. URL: http://proceedings.mlr.press/v80/parmar18a.html (cit. on p. 17).

[24] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever. "Language models are unsupervised multitask learners". In: *OpenAI Blog* 1.8 (2019), p. 9 (cit. on p. 17).

[25] A. Sagheer, M. Kotb. "Time series forecasting of petroleum production using deep LSTM recurrent networks". In: *Neurocomputing* 323 (2019), pp. 203–213 (cit. on p. 12).

[26] S. Siami-Namini, A. S. Namin. "Forecasting economics and financial time series: ARIMA vs. LSTM". In: *arXiv preprint arXiv:1803.06386* (2018) (cit. on p. 17).

[27] J. Tompson, K. Schlachter, P. Sprechmann, K. Perlin. "Accelerating eulerian fluid simulation with convolutional networks". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 3424–3433 (cit. on p. 7).

[28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin. “Attention is all you need”. In: *Advances in neural information processing systems*. 2017, pp. 5998–6008 (cit. on pp. 7, 14, 15, 17, 19, 21).

[29] N. Wu, B. Green, X. Ben, S. O’Banion. “Deep Transformer Models for Time Series Forecasting: The Influenza Prevalence Case”. In: *arXiv preprint arXiv:2001.08317* (2020) (cit. on p. 17).

[30] S. Wu, X. Xiao, Q. Ding, P. Zhao, Y. Wei, J. Huang. “Adversarial Sparse Transformer for Time Series Forecasting”. In: *Advances in Neural Information Processing Systems* 33 (2020) (cit. on p. 18).

All links were last followed on June 8, 2021.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Druck-Exemplaren überein.

Datum und Unterschrift: 08.06.2021, S. B

Declaration

I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted hard copies.

Date and Signature: 08.06.2021, S. B