Department of Mathematics
TUM School of Computation, Information and Technology
Technical University of Munich

Explaining Transformers through Dynamical Systems Theory

Nikita Okorokov

Thesis for the attainment of the academic degree

Master of Science

at the TUM School of Computation, Information and Technology of the Technical University of Munich

Examiner:
Prof.Dr.Felix Dietrich

Submitted:
Munich, May 15th, 2024

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

Munich, May 15th, 2024 Nikita Okorokov

Abstract

This thesis explores the operations of Transformer blocks through the lens of Koopman Operator theory, aiming to provide a deeper understanding of their inner dynamics. Leveraging Koopman Operator theory, we conceptualize Transformer operations as dynamical processes, uncovering underlying structures governing the processing of sequential data. Our investigation begins with the decoder blockof the Transformer model, treating the self-attention mechanism as a dictionary function and the feed-forward layer as an inverse function. We identify the first weight matrix in the feed-forward layer as a Koopman Operator responsible for transitioning the system to the next state in the dictionary space. Extending our inquiry, we explore the encoder-decoder architecture, incorporating parametric Koopman Operator theory to elucidate information flow dynamics within the Transformer model. Empirical validation through practical experiments demonstrates alignment of Transformer operations with Koopman Operator theory, particularly in the machine translation task using the Anki dataset. Despite initial discrepancies, refinements in model architecture and loss functions successfully realign Transformer operations without compromising performance, as evidenced by sustained BLEU scores for machine translations.

This research bridges the gap between machine learning and dynamical systems theory, laying the groundwork for a deeper understanding of Transformer models and the development of more interpretable and controllable machine learning architectures in NLP.

Acknowledgement

I begin by expressing my deepest gratitude to Prof. Felix Dietrich for providing me an opportunity to work on this thesis and for his invaluable guidance and feedback throughout its development. His mentorship has been instrumental in shaping my understanding and approach.

I extend my heartfelt appreciation to my wife, Tatyana, whose unwavering support and belief in me have been a constant source of strength. Her encouragement has been a guiding light, enabling me to navigate through challenges with resilience and determination. To my newborn son, Artem, I am endlessly grateful for the pure love and joy he brings into our lives. His presence has imbued me with newfound motivation and purpose, spurring me onward in my academic and professional pursuits.

I am indebted to my friends and family for their support and encouragement. Their belief in my abilities has been a source of inspiration, fueling my aspirations and propelling me toward success.

Lastly, to each and every individual who has played a part in my journey, whether through encouragement, support, or simply being there, I offer my heartfelt thanks. Your unwavering belief in me has been the cornerstone of my personal and academic growth.

Contents

1 Introduction

Natural Language Processing or NLP for short is a branch of artificial intelligence focused on enabling computers to interpret, analyze and generate human language. Main tasks of language modeling are machine translation, sentiment analysis, summarization, question answering and text classification. Traditional approaches to natural language processing often relied on handcrafted features and statistical models. However, the advent of deep learning marked a watershed moment in NLP.

One of the most significant developments in recent years is the introduction of the Transformer model, a type of architecture that has revolutionized the way we approach tasks in natural language processing and beyond. The Transformer model, introduced by Vaswani et al. in the paper "Attention is All You Need", has become a cornerstone in the field due to its ability to handle long-range dependencies in sequence data, outperforming previous state-of-the-art models in a wide range of NLP tasks [47].

Tasks like language translation, sentiment analysis, text summarization, and named entity recognition have witnessed significant advancements due to the transformer's ability in handling sequential data. Pre-trained transformer models, such as BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer), have become benchmarks in the field. Despite its popularity in language modeling, Transformer architecture has been successfully applied in fields like computer vision [20], modeling of graph-structured data [51], music generation [15] and numerous other domains.

Transformer model consists of an encoder and a decoder, each of which is made up of multiple identical layers. Each layer in the encoder processes the input sequence and generates a continuous representation of the sequence. This representation is then passed to the decoder, which generates the output sequence. The Transformer model is based on the concept of attention mechanism, which scores the importance or relevance of each input in relation to the others. This allows the model to focus on different parts of the input sequence when producing each element of the output sequence, thereby capturing the dependencies between elements regardless of their distance in the sequence. Despite its success, the operations of the Transformer model, particularly its self-attention mechanism, are often considered as a black box due to their complex nature.

Dynamical systems theory is an area of mathematics that studies the behavior of systems that change over time. These systems can be anything from simple pendulums to complex networks of interacting particles. Theory of dynamical systems provides a framework for understanding how such systems evolve over time and under different conditions. It is widely used in various fields such as physics, engineering, economics and computer science.

Dynamical systems, especially nonlinear ones, exhibit a richness of behaviors that often pose significant challenges in understanding and predicting their dynamics. These systems, characterized by sensitivity to initial conditions and nonlinearity, can display chaotic behavior or intricate patterns that resist simple analytical solutions. Understanding and modeling such systems demand sophisticated mathematical tools and computational approaches. The Koopman operator provides a powerful lens, aiming to uncover underlying structures in these seemingly complex and unpredictable systems.

Developed by Bernard Koopman in the 1930s, the Koopman operator provides a powerful mathematical tool for analyzing the evolution of nonlinear dynamical systems [21]. It operates in an infinite-dimensional function space, offering a linear perspective on the dynamics of nonlinear systems. By transforming the

dynamics of a system into linear operations, the Koopman operator facilitates analysis, prediction, and comprehension of the underlying structure and behavior of these systems [4].

In addition to its theoretical importance, the Koopman operator also has practical applications in the fields of control, data analysis and machine learning. For instance, it can be used to learn models from complex, real-world data sets, enabling state-of-the-art prediction and control. The greater interpretability and lower computational costs of these models, compared to traditional machine learning methodologies, make Koopman learning an especially appealing approach. The Koopman Operator has been successfully applied in various fields, including fluid dynamics, power systems and robotics, to analyze and control complex systems. In the context of machine learning, the Koopman Operator can potentially provide a theoretical basis for understanding the operations of complex models based on neural networks.

Operations of transformer block can also be considered as a dynamical system, where next state is computed with the help of attention mechanism and feed-forward layer neural networks. In this thesis we will try to employ the Koopman operator to describe the operations of transformer block and provide a deeper understanding of the transformer models as dynamical systems. The main challenge will be to express the action of a transformer block as an evolution of a function in Hilbert space, exactly as in the Koopman operator setting.

The expected outcome of this research is a mathematical explanation of the transformer block's operations using the Koopman operator. By representing the operations of the Transformer block through the Koopman Operator framework, we can gain insights into how the model processes and transforms the input data, and how it learns to focus on different parts of the input sequence. Potentially this can lead to new optimization techniques or improvements in transformer models.

In conclusion, this thesis aims to bridge the gap between the fields of machine learning and dynamical systems by applying the Koopman Operator theory to the Transformer model. We believe that this interdisciplinary approach can lead to a deeper understanding of the Transformer model and contribute to the development of more interpretable and controllable machine learning models.

Thesis text is divided into three chapters and has the following structure.

Chapter 2 This foundational chapter lays the groundwork for understanding the theoretical underpinnings of our research. It begins with a comprehensive overview of the evolution of language modeling, tracing its journey from rule-based systems to the sophisticated architectures of transformers. Specifically, Section 2.1 delves into the historical progression, starting with rule-based modeling in subsection 2.1.1, and culminating in the introduction of transformer architectures in subsection 2.1.3. This section sets the stage for the exploration of transformers as a pivotal development in natural language processing. Following this, Section 2.2 delves into the core concepts of dynamical systems, providing a solid foundation for the subsequent chapters. It introduces the Koopman operator, a powerful mathematical tool that linearizes nonlinear dynamical systems, thereby enabling their analysis and manipulation. This introduction to the Koopman operator is crucial for understanding its application in machine learning and deep learning, as discussed in Section 2.3. This section further explores the application of the Koopman operator across various subfields, with a particular emphasis on its role in neural networks. By the end of this chapter, readers will have a robust understanding of the theoretical background necessary for grasping the novel approach of applying Koopman operator theory to transformer models.

Chapter 3 Chapter 3 transitions from theory to practice, focusing on the operational aspects of transformer blocks through the lens of the Koopman operator. This chapter is structured to bridge the gap between theoretical knowledge and practical application. It begins by outlining the operations of transformer blocks, detailing how these components function within the architecture. This theoretical under-

standing is then tested through a series of experiments, as described in this chapter. The experiments aim to determine whether the operations within the decoder block of transformers can be aligned with the principles of Koopman operator theory. Additionally, this chapter explores the potential for modifying the training process of transformers to achieve alignment with Koopman operator theory. Through these experiments, we seek to validate the applicability of Koopman operator theory to transformer models, providing empirical evidence of its effectiveness.

Chapter 4 The final chapter of this thesis, Chapter 4, encapsulates the conclusions drawn from the research and outlines potential future work. This chapter serves as a summary of the findings, highlighting the key contributions of the thesis to the field. It also identifies areas where further research is needed, suggesting directions for future studies. By synthesizing the theoretical and practical aspects of the research, this chapter provides a comprehensive overview of the thesis's impact and its implications for the broader field of machine learning and natural language processing. This chapter is crucial for readers to understand the significance of the research and its potential to advance the field.

2 Theoretical Background

This opening chapter introduces essential concepts crucial for the subsequent sections of the thesis. Primarily, it comprises a thorough analysis of findings from existing literature, organized and presented to enhance the clarity of the underlying theory.

2.1 Evolution of Natural Language Processing

In this section, we provide a comprehensive exploration of the evolutionary trajectory of Natural Language Processing (NLP). From early linguistic theories to modern deep learning approaches, this research illuminates the dynamic evolution that has propelled NLP to the forefront of artificial intelligence and language understanding.

2.1.1 Rule-Based Methods, Statistical Models and Word Embeddings

In this subsection, we delve into the multifaceted landscape of Natural Language Processing (NLP), analyzing three distinct approaches that have played pivotal roles in shaping the field: Rule-Based Methods, Statistical Models, and Word Embeddings. Rule-Based methods, based on linguistic principles, have been foundational in structuring language understanding systems. Statistical Models introduced a paradigm shift by leveraging vast corpora to extract patterns and relationships. The emergence of Word Embeddings revolutionized NLP by capturing semantic nuances and contextual intricacies. Through a nuanced examination of these approaches, we aim to research their individual contributions, strengths, and limitations, providing a comprehensive background for understanding the dynamic evolution of NLP methodologies.

Rule-Based Methods

Rule-based methods in Natural Language Processing were among the earliest approaches used to process and understand language. These methods relied on predefined linguistic rules, heuristics and pattern matching to analyze and interpret textual data. Linguists and experts meticulously crafted sets of rules based on linguistic principles and grammatical structures. These rules aimed to codify the syntax, semantics and morphology of language.

One of the most popular examples from this period is the natural language processing system called "Eliza". It was developed by Joseph Weizenbaum in 1966 [48]. Eliza represented a basic chatbot designed to mimic discussions with a psychotherapist. Although Eliza's replies were scripted in advance, individuals found the interaction unexpectedly captivating and believed they were engaging with a real human being.

Despite their structured approach and interpretability, rule-based methods face notable challenges. They were labor-intensive, requiring expertise in linguistics and continual refinement of rules for different languages and domains. Scalability was a significant issue, as accommodating new linguistic patterns or adapting to variations in language proved cumbersome. These systems often struggled with ambiguity, context sensitivity and the inherent dynamism of language. These limitations forced the development of more sophisticated approaches, such as statistical methods and word embeddings.

Statistical Methods

In the 1990s statistical approaches emerged in language modeling to overcome the drawbacks of rule-based systems. These techniques employ statistical models such as N-grams language models, Markov Models, Naive Bayes, SVM and Logistic Regression for the analysis and generation of human language. One of the main applications of statistical methods in NLP involves predicting a word's probability based on its context. Additionally, these methods played a crucial role in machine translation, facilitating the creation of models that could translate text between languages.

N-grams language modeling N-grams are a sequence of N contiguous words or characters extracted from a text corpus. They are widely used in language modeling, capturing the probability of word sequences occurring together [18]. For instance, in a bigram model (2-grams), the probability of the word "good" following "very" might be learned from the frequency of the phrase "very good" in the corpus. N-grams provide context-based information about word sequences and are instrumental in predicting the likelihood of a word given its context [45].

Markov Models The idea that a word's probability depends only on a small number of preceding words is called a Markov assumption [18]. Markov models encompass the group of probabilistic models that propose the ability to anticipate the likelihood of the future word without looking too far into the previous text. Markov models, especially Hidden Markov Models (HMMs), are prevalent in speech recognition, part-of-speech tagging and named entity recognition. For instance, in part-of-speech tagging, an HMM might learn the probability of a word being a noun, verb or adjective based on its context and the sequence of parts of speech observed in the corpus [38].

Naive Bayes It is a probabilistic classifier based on the Bayes theorem and the assumption of feature independence. Despite its "naive" assumption of independence between features, it performs remarkably well in text classification tasks like sentiment analysis, spam filtering and document categorization. Naive Bayes classifiers are extensively used in text classification, especially when dealing with large volumes of text data. They are efficient, easy to implement and provide decent accuracy in tasks where independence assumptions hold reasonably well [37].

Logistic Regression Despite its name, logistic regression is a classification algorithm that models the probability of a binary outcome. In NLP, it is applied to text classification tasks, assigning a probability to each class and making predictions based on these probabilities. Logistic regression is used in sentiment analysis, document classification and spam detection. It is valued for its simplicity, interpretability and capability to handle linearly separable problems in text classification [18].

Support Vector Machine SVM is a powerful supervised learning algorithm used for classification and regression tasks. In language modeling, it is predominantly applied to text classification tasks by finding the optimal hyperplane that separates classes in a high-dimensional space. SVMs are employed in text classification tasks such as sentiment analysis, text categorization and document classification. They are valued for their ability to handle high-dimensional data efficiently and find complex decision boundaries in text [18].

These statistical methods played a crucial role in various NLP tasks, providing methods to model language, classify text and extract meaningful insights from textual data. However, they do not take into account a sequential nature of texts and may struggle with capturing complex patterns in data that can be better modeled by more sophisticated architectures like neural networks.

Word Embeddings

To apply machine learning algorithms for textual data, we need to encode text into numeric representation. Two of the main popular methods for it are One-Hot-Encoding and term frequency-inverse document frequency (TF-IDF).

In the case of One-Hot-Encoding we start by creating a vocabulary, containing all unique words from the text corpus. Each word in the vocabulary is represented as a unique binary vector, where all elements are 0 except for one element corresponding to the index of that word, which is set to 1.

TF-IDF is a numerical statistic used to evaluate the importance of a word in a document relative to a collection of documents (corpus). It measures the relevance of a word by balancing its frequency within a document (TF) against its frequency across all documents (IDF) [41]. These statistics are computed as follows

$$TF(t,d) = \frac{\text{Number of times the term } t \text{ appears in the document } d}{\text{Total number of terms in the document } d}, \tag{2.1}$$

$$IDF(t,D) = \log\left(\frac{\text{Total number of documents in corpus } D}{\text{Number of documents in the corpus containing the term } t}\right), \tag{2.2}$$

where t represents a term (word), d represents a specific document and D represents the entire corpus (collection of documents). Finally, the TF-IDF score is obtained by multiplying TF and IDF for each term in a document.

However, these methods have significant limitations when it comes to creating word embeddings. They provide basic representations of words, but lack the ability to encode semantic meaning, context or relationships between words, which are crucial aspects for various Natural Language Processing tasks. To address these issues more advanced techniques like Word2Vec, Doc2Vec and Glove were developed.

Word2Vec Word2Vec, introduced by a team at Google, learns word embeddings by predicting surrounding words in a context window. It operates on two architectures: Continuous Bag-of-Words (CBOW) and Skip-gram. CBOW predicts a target word based on its context (surrounding words), while Skip-gram predicts surrounding words given a target word. Word2Vec learns vector representations that capture semantic relationships between words. It encodes similarity, such that words with similar meanings have similar vector representations [30, 31].

Doc2Vec Doc2Vec, also known as paragraph embeddings or paragraph vectors, is an extension of Word2Vec that allows the generation of fixed-length feature representations for variable-length pieces of texts, such as sentences, paragraphs or documents. The fundamental idea behind Doc2Vec is to extend the Word2Vec model to learn continuous representations not only for words but also for entire documents. This technique aims to capture the semantic meanings and contextual information of entire documents in fixed-length vectors [22].

GloVe Global Vectors for Word Representation (GloVe) is an unsupervised learning algorithm for generating word embeddings, developed by researchers at Stanford University. GloVe constructs word embeddings by leveraging global word-word co-occurrence statistics across the entire corpus. It emphasizes global context relationships. It assigns vectors to words based on the probability of word co-occurrences in a global context, emphasizing word relationships beyond local contexts [36].

In essence, statistical models and basic machine learning techniques, complemented by word embeddings, marked a significant shift towards data-driven approaches in NLP, enabling better semantic representation and understanding of language.

2.1.2 Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU)

In this subsection, we provide a detailed exploration of three foundational neural network approaches for language modeling: Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, and Gated Recurrent Unit (GRU) networks. RNNs, characterized by their ability to process sequential data, are instrumental in capturing temporal dependencies within text data. LSTM networks, a variant of RNNs, address the vanishing gradient problem by incorporating memory cells, enabling them to retain long-term dependencies. Similarly, GRU networks offer a streamlined architecture with gated mechanisms, providing an efficient alternative for modeling sequential data. Through an in-depth analysis of these architectures, we aim to elucidate their unique functionalities, strengths and limitations for language modeling, laying the groundwork for subsequent discussions on advanced language processing techniques.

Texts and natural language has a temporal or sequential relationship between the words. However, basic machine learning models process each input independently, disregarding the order in which the data is presented. Therefore, they struggle to capture dependencies or patterns based on the sequence of words. To overcome this limitation, Recurrent Neural Networks and their variants such as Long Short-Term Memory and Gated Recurrent Unit neural networks were developed.

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a class of neural networks designed for handling sequential data by retaining information in memory across time steps [39]. They process sequences of inputs while maintaining an internal state that captures information about previous inputs. The architecture of a simple Recurrent Neural Network consists of the following elements:

1. Inputs. At each time step t, an RNN receives an input vector x_t.
2. Hidden State. RNN maintains a hidden state h_t at each time step t, which acts as the memory capturing information from previous inputs. This hidden state is updated at each time step.
3. Output. RNN produces output y_t based on the current input vector x_t and the hidden state h_t.

The calculation of the hidden state h_t and output y_t in a basic RNN can be represented by

$$h_t = \text{activation}(W_{hx} \cdot x_t + W_{hh} \cdot h_{t-1} + b_h), \tag{2.3}$$
$$y_t = W_{hy} \cdot h_t + b_y, \tag{2.4}$$

where x_t is the input vector at timestamp t, h_t is the hidden state at timestamp t, y_t is the output at timestamp t, W_{hx} is the learnable weight matrix for the input, W_{hh} is the learnable weight matrix for the hidden state, W_{hy} is the learnable weight matrix for the output, b_h and b_y are bias terms, $activation$ is some activation function. Figure 2.1 shows a cell structure in RNN block.

Figure 2.2 shows an architecture of the simple RNN.

This recurrent nature of using the previous hidden state to calculate the current hidden state allows RNNs to retain information across time steps and process sequential data. However, basic RNNs can struggle capturing long-range dependencies due to the vanishing and exploding gradient problems. More advanced architectures like LSTMs and GRUs were developed to address these issues [43].

Long Short-Term Memory Network

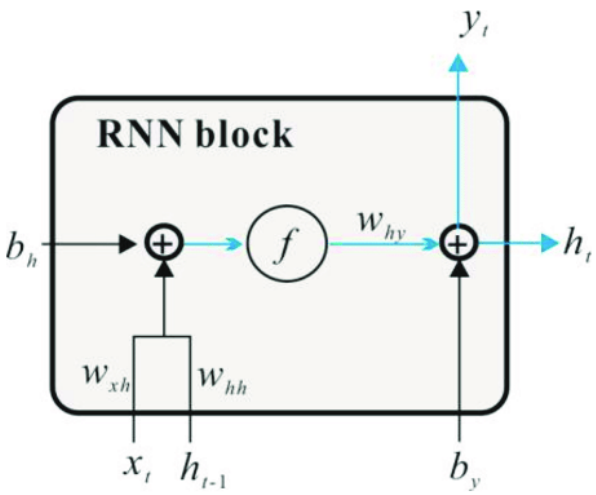


Figure 2.1 RNN cell structure. Taken from [52].

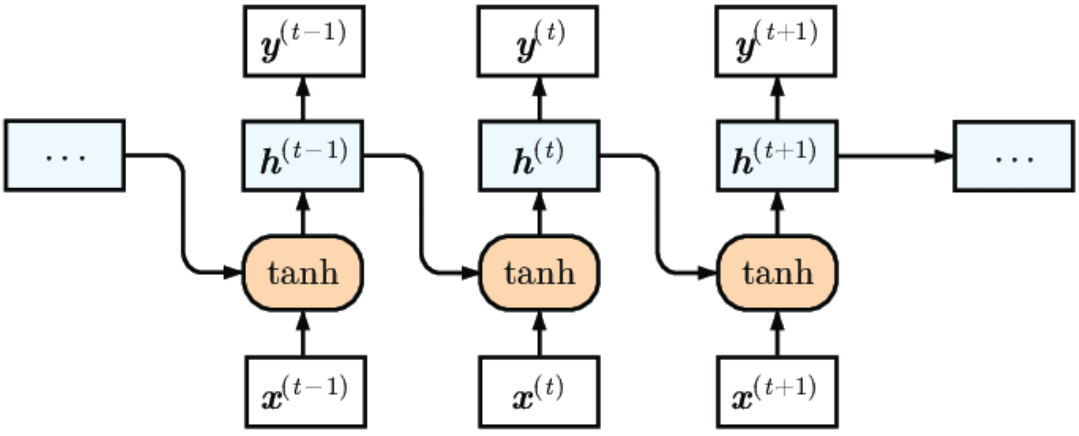


Figure 2.2 Vanilla RNN architecture. Taken from [32].

Long Short-Term Memory networks (LSTMs) were developed to overcome issues with vanishing and exploding gradients in recurrent neural networks by introducing a more sophisticated cell structure. Figure 2.3 shows structure of LSTM cell.

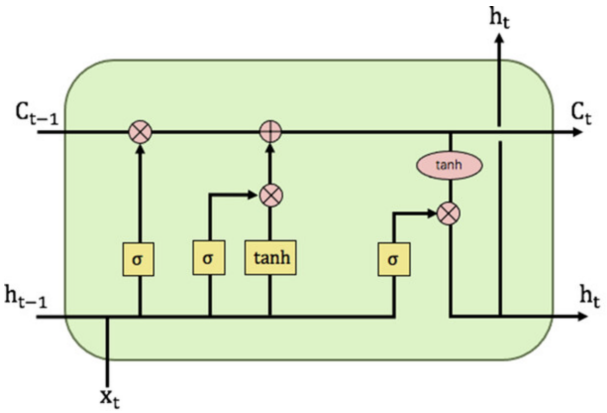


Figure 2.3 LSTM cell structure. Taken from [46].

The key innovation lies in the architecture of the LSTM cell, which includes specialized mechanisms known as gates. Their names are forget gate, input gate and output gate. These gates play a crucial role in controlling the flow of information through the cell, allowing for the effective management of gradients during the training process [14].

Forget Gate The forget gate decides what information from the previous cell state should be discarded. It takes into account both the previous hidden state h_{t-1} and the current input vector x_t. Then by applying a sigmoid function, the forget gate outputs values between 0 and 1, indicating the proportion of information to forget (0 - completely discard, 1 - keep complete information). This is done by

$$f_t = \sigma(W_{fx} \cdot x_t + b_{fx} + W_{fh} \cdot h_{t-1} + b_{fh}), \tag{2.5}$$

where x_t is the current input vecor, h_{t-1} is the previous hidden state, W and b represent learnable weights and biases.

Input Gate The input gate determines what new information will be stored in the cell state. It consists of a sigmoid layer and a tanh layer. The sigmoid layer decides which values will be updated and the tanh layer creates new candidate values that could be added to the state. This process is described as

$$i_t = \sigma(W_{ix} \cdot x_t + b_{ix} + W_{ih} \cdot h_{t-1} + b_{ih}), \tag{2.6}$$
$$g_t = \tanh(W_{ig} \cdot x_t + b_{ig} + W_{hg} \cdot h_{t-1} + b_{hg}), \tag{2.7}$$

where x_t is the current input vecor, h_{t-1} is the previous hidden state, W and b represent learnable weights and biases.

Cell State Update The cell state is updated based on the decisions from the input and forget gates. It forgets the information as decided by the forget gate and adds the new candidate values scaled by the importance values from the input gate. The update process of the cell state is represented by

$$c_t = f_t \cdot c_{t-1} + i_t \cdot g_t, \tag{2.8}$$

where f_t is the result of forget gate, i_t and g_t are results of input gate and c_{t-1} is the cell state from previous step.

Output Gate Finally, the output gate determines what information is required for the current hidden state. This process is described by

$$o_t = \sigma(W_{ox} \cdot x_t + b_{ox} + W_{oh} \cdot h_t + b_{oh}), \tag{2.9}$$
$$h_t = o_t \cdot \tanh(c_t), \tag{2.10}$$

where x_t is the current input vector, h_t is the current hidden state, c_t is the current cell state, W and b represent learnable weights and biases.

Gated Recurrent Unit

Gated Recurrent Unit is another improvement over Recurrent Neural Networks, introduced by Kyunghyun Cho in 2014 [7]. GRU is very similar to LSTM. Both utilizes gates to control the flow of information, but GRUs have simpler architecture and some improvements over LSTMs while offering similar capabilities to handle long-range dependencies in sequential data. GRU architecture consists of two gates - the update gate and the reset gate. Figure 2.4 shows the architecture of Gated Recurrent Unit.

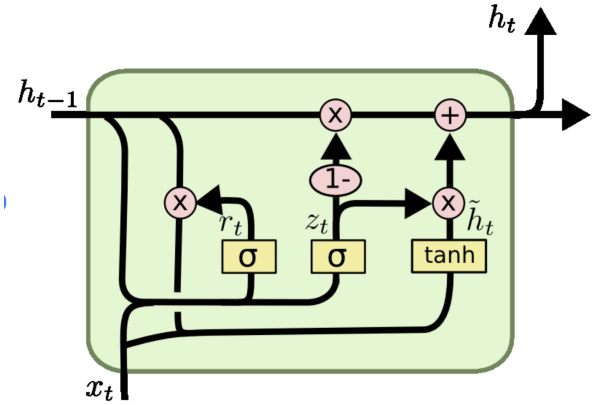


Figure 2.4 Gated Recurrent Unit (GRU). Taken from [1].

Update Gate (Long Term Memory) Decides how much of the information from the current input vector and previous hidden state to consider to update the current hidden state. The update gate is defined by

$$z_t = \sigma(W_{xz} \cdot x_t + W_{hz} \cdot h_{t-1} + b_z), \tag{2.11}$$

where x_t is the current input vector, h_{t-1} is the previous hidden state, W and b represent learnable weights and biases.

Reset Gate (Short Term Memory) Manages how much of the previous hidden state to forget or reset. This is done by

$$r_t = \sigma(W_{xr} \cdot x_t + W_{hr} \cdot h_{t-1} + b_r), \tag{2.12}$$

where x_t is the current input vector, h_{t-1} is the previous hidden state, W and b represent learnable weights and biases.

Finally, the output of the GRU is computed, using update and reset gates

$$h'_t = tanh(W_{xh} \cdot x_t + W_{hh}(r_t \odot h_{t-1}) + b_h), \tag{2.13}$$
$$h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot h'_t, \tag{2.14}$$

where x_t is the current input vector, h_{t-1} is the previous hidden state, r_t is the output of reset gate, z_t is the output of update gate, W and b represent learnable weights and biases.

RNNs, LSTMs and GRUs have transformed a landscape of sequential data modeling, particularly in natural language processing, time series analysis and various other fields [13, 40].

2.1.3 Attention and Transformers

In this subsection we describe two groundbreaking advancements in the field of Natural Language Processing: Attention mechanism and Transformers. Attention mechanism, inspired by human cognitive processes, enable models to focus on relevant parts of the input sequence, revolutionizing tasks such as machine translation and text summarization. Transformers, based on attention mechanism, introduce a novel architecture that eschews recurrent connections in favor of self-attention mechanisms, facilitating parallel computation and enabling the modeling of long-range dependencies more effectively. By delving into the intricacies of attention mechanism and transformers, we aim to describe their concepts and advantages over recurrent models.

Attention

Attention mechanism was introduced in 2016 in the seminal paper "Neural Machine Translation by Jointly Learning to Align and Translate" by Bahdanau et al. It revolutionized the landscape of neural machine translation. The attention mechanism, a fundamental component of sequence-to-sequence models, overcame the limitations of traditional models by enabling dynamic alignments between source and target sequences during translation [3].

RNN encoder-decoders was state-of-the-art approaches for the task of Neural Machine Translation (NMT) before attention. The encoder reads and encodes a source sentence into a fixed-length vector. A decoder then generates a translation from the encoded vector. The whole encoder-decoder system is jointly trained to maximize the probability of a correct translation given a source sentence. However, the fixed-length vector from the encoder is the bottleneck of such systems, because it needs to contain all information from the source text. Therefore, authors developed an attention mechanism to address this issue. Figure 2.5 shows an architecture of the system for NMT proposed by authors.

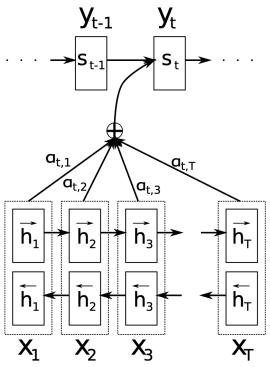


Figure 2.5 Illustration of the proposed system during generation of the target word y_t based on the source sentence $(x_1, ..., x_T)$. Taken from [3]

Encoder Encoder reads an input sequence of tokens x, starting from the first token x_1 to the last token x_T. However, here authors instead of standard RNN used a bidirectional RNN (BiRNN) [44] to take into account information not only from the previous words, but also from the following words. BiRNN consists of forward RNN and backward RNN. Forward RNN reads input sequence from x_1 to x_T and computes hidden states $(\overrightarrow{h_1}, ..., \overrightarrow{h_T})$. Backward RNN goes through the sequence in the reverse order from x_T to x_1 and

outputs hidden states $(\overleftarrow{h_1}, ..., \overleftarrow{h_T})$. Finally, the hidden state for each token x_i is obtained by concatenating corresponding forward and backward hidden states, i.e. $h_i = [\overrightarrow{h}_i^T, \overleftarrow{h}_i^T]$.

Decoder Decoder is used to predict next word y_i based on the context vector c_i and all the previously predicted words $(y_1, ..., y_{i-1})$. The context vector c_i is computed based on weighted sum of the hidden states from encoder, so that

$$c_i = \sum_{j=1}^{T} \alpha_{ij} h_j, \tag{2.15}$$

where α_{ij} are attention weights and h_j are hidden states from encoder for word j. Attention weights are calculated based on

$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T} exp(e_{ik})}, \tag{2.16}$$

where e_{ij} is the alignment model which scores how the input words around index j and the output at index j match. This score is computed based on the RNN decoder hidden state s_{i-1} and encoder hidden state h_j by

$$e_{ij} = h_j \cdot s_{i-1}. \tag{2.17}$$

The likelihood α_{ij} determines the importance of the annotation h_j concerning the preceding hidden state s_{i1} in generating next state s_i and output y_i. This establishes an attention mechanism within the decoder. This approach enables the decoder to determine specific segments of the source sentence to focus on. By incorporating an attention mechanism into the decoder we relieve the encoder from the task of encoding all details from the source sentence into a static, predefined vector. This updated method allows the information to be distributed across the sequence of annotations, which can then be selectively accessed by the decoder.

Transformers

In this section we describe in details the architecture of Transformers. Transformer architecture introduced in the paper "Attention is All You Need" by Vaswani et al. revolutionized sequence-to-sequence learning using self-attention mechanisms without recurrent or convolutional layers — a key breakthrough in transformer models. Self-attention allows a model to directly access and utilize information from expansively large contexts, without the necessity of routing it through the recurrent connections, like in RNNs. The model consists of encoder and decoder components, each comprising multiple layers [47]. Figure 2.6 shows an architecture of Transformer.

Encoder Encoder block is consists of a stack of $N = 6$ identical layers. Each layer has two sub-layers: multi-head self-attention mechanism and position-wise fully connected feed-forward network. In addition, authors employed the residual connection around each sub-layer, following by layer-normalization. The output of each sub-layer is computed in the following way: $output = LayerNorm(x + SubLayer(x))$, where $SubLayer(x)$ is the operations performed by sub-layers.

Positional Encoding To incorporate sequence order information, positional encodings are added to the input embeddings. They provide the model with information about the position of tokens in the sequence. A commonly used positional encoding approach for position pos and dimension i is represented by

$$PE(pos, 2i) = sin(pos/10000^{(2i/d_{model})}), \tag{2.18}$$

$$PE(pos, 2i+1) = cos(pos/10000^{(2i/d_{model})}), \tag{2.19}$$

where $d_{model} = 512$.

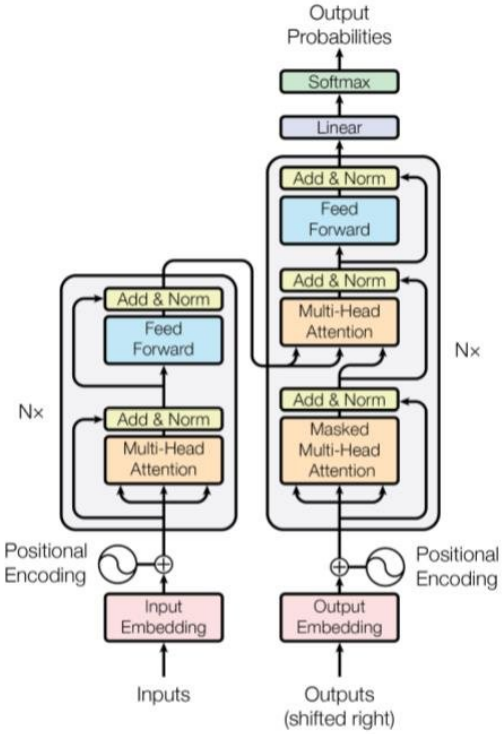


Figure 2.6 Architecture of Transformer model. Taken from [47].

Multi-Head Self-Attention The Multi-Head Self-Attention mechanism in Transformers allows the model to focus on different parts of the input sequence in parallel, therefore capturing various types of contextual dependencies between words. Authors found it beneficial to perform $h = 8$ times self-attention function. Each self-attention function computes the attention scores between all positions in the input sequence. This is done by

$$Q_i = XW_{Q_i}, \tag{2.20}$$
$$K_i = XW_{K_i}, \tag{2.21}$$
$$V_i = XW_{V_i}, \tag{2.22}$$
$$Head_i = Attention(Q_i, K_i, V_i) = softmax(\frac{Q_i K_i^T}{\sqrt{d_{model}}})V_i, \tag{2.23}$$
$$MultiHead(Q, K, V) = Concat(Head_1, ..., Head_h)W_O, \tag{2.24}$$

where X is the input sequence after positional encodeing, W are learnable weight matrices and $d_{model} = 512$. Architectures of scaled dot-product attention and multi-head attention are presented on Figure 2.7.

Feed-Forward Neural Network (FFN) The output of the multi-head self-attention layer passes through a position-wise feed-forward neural network, which consists of fully connected layers. This is described by

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2, \tag{2.25}$$

where x is the output of multi-head self-attention layer, W and b are learnable weight matrices and biases.

Decoder The decoder block also consistst of a sequence of $N = 6$ identical layers. Alongside the two sub-layers present in encoder block, the decoder introduces an additional third sub-layer, which performs

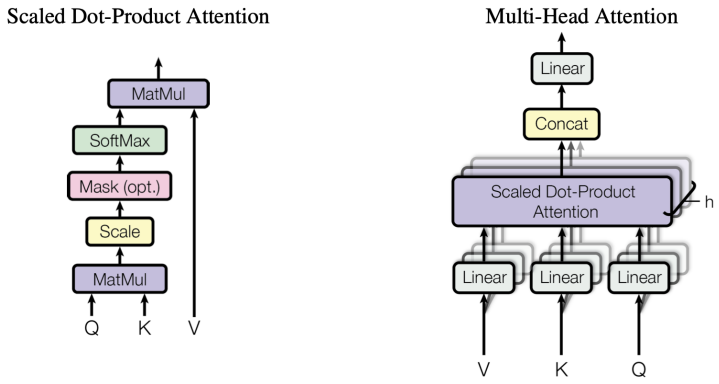


Figure 2.7 (left) Operations of Scaled Dot-Product Attention. (right) Operations of Multi-Head Attention. Taken from [47].

the multi-head self-attention over the output from the encoder block to focus on relevant parts of the input sequence. As in the encoder block, authors incorporated residual connections around each of the sub-layers and normalization layers. In addition, authors modified the self-attention layer to prevent positions to attend subsequent positions. This approach is called Masked Multi-Head Self-Attention. This masking, coupled with the displacement of output embeddings by one position, guarantees that predictions for a particular position i are exclusively rely only on the known outputs at positions less than i. This is implemented inside of scaled dot-product attention by seting to $-\infty$ all values in the input of the softmax which correspond to illegal connections.

Finally, outputs from the decoder block additionally passes through the linear and softmax layer to output final probabilities.

Transformers have emerged as a paradigm-shifting architecture, revolutionizing sequence-to-sequence learning by leveraging self-attention mechanisms. Their ability to efficiently capture long-range dependencies without recurrent connections has significantly enhanced the performance of various natural language processing tasks. The transformative impact of Transformers lies in their their capability to handle extensive contextual information, making them a cornerstone in modern machine learning and language understanding.

2.2 Dynamical Systems

In this section we explore theory and basic concepts of dynamical systems. Dynamical systems theory provides a powerful framework for understanding the evolution of complex systems over time, offering insights into the behaviors and patterns that emerge from underlying dynamics.

2.2.1 Basic Concepts of Dynamical Systems

In this subsection, we describe fundamental concepts that constitute the backbone of dynamical systems theory. Dynamical systems, as a theoretical framework, offer a profound understanding of how systems evolve over time. Here we list the fields where dynamical systems can be applied and explain basic concepts such as state space, evolution law, fixed points, linear and nonlinear dynamical systems.

Dynamical systems are mathematical models used to describe the time-dependent evolution of systems governed by certain rules. They are used in a multitude of scientific fields from mathematics and physics to biology, chemistry, engineering, economics and even medicine. Some examples of the dynamical systems are:

1. **Pendulum System**: A simple pendulum is a classic example of a dynamical system. The state of the pendulum at any given time can be described by its angle and angular velocity. The evolution of the system is governed by the second law of motion and the gravitational force [5].

2. **Population Dynamics**: In biology dynamical systems are used to model the growth and interaction of populations. For instance, the Lotka-Volterra equations model the interaction of two species: a predator and its prey. The state of the system is given by the sizes of the predator and prey populations and the evolution of the system is determined by the birth and death rates of both species [17].

3. **Economic Systems**: In economics dynamical systems can be used to model the evolution of economic indicators over time. For example, a simple macroeconomic model might treat the inflation rate and unemployment rate as the state of the system with the evolution of the system determined by monetary policy and aggregate demand [11].

4. **Fluid Dynamics**: In fluid dynamics, the motion of a fluid can be modeled as a dynamical system. The state of the system can be described by the velocity and pressure at each point in the fluid, and the evolution of the system is governed by the Navier-Stokes equations [26].

5. **Neural Networks**: In neuroscience and artificial intelligence the activity of a neural network can be modeled as a dynamical system. The state of the system is given by the firing rates of the neurons in the network and the evolution of the system is determined by the connections between neurons and their activation functions [16].

The basic concepts of dynamical systems theory revolve around the idea of determinism, which suggests that the future behavior of a system can be fully determined by its current state. A deterministic dynamical system is typically described by two main components: the phase space and the evolution law. The phase space X contains vectors that quantitatively determine all possible states of the system. The states of the system are usually described by a d-dimensional vector $x \in X$, whose d components $x_1, .., x_d$ are called as degrees of freedom pf the system. The evolution law $f : X \to X$ is a rule that allows us to determine the state of the system at a future time given its current state. This is usually represented by

$$x_{t+1} = f(x_t), \tag{2.26}$$

where $x_t \in X$ is d-dimensional vector which represents the state of the system at time step t and f is an arbitrary map [6]. For each initial state $x_0 \in X$ the map f provides us with the new state of the system after one step of time or one application of the mapping f

$$x_1 = f(x_0). \tag{2.27}$$

As $x_1 \in X$ we can again apply the map f to the state x_1 and obtain the new state $x_2 = f(x_1)$ which is the state of the system from x_0 after two time steps or two applications of the mapping f. In general, we can apply mapping f any number of times and we can define the n'th iterate of f by

$$f^n = f \odot ... \odot f, \tag{2.28}$$

where $\odot$ is the composition of maps f. Thus, starting from the initial state of the system $x_0 \in X$, the state of the system after n time steps can be obtained by

$$x_n = f^n(x_0). \tag{2.29}$$

Figure 2.8 shows an example of dynamical system, where P is a map and $P(1)$, $P(2)$ and $P(3)$ are the states of the system at time steps 1,2 and 3.

Theory of dynamical systems have concepts of fixed and periodic points. x_0 is a fixed point if $f(x_0) = x_0$. It is clear that if x_0 is a fixed point of the system then $x_1 = f(x_0) = x_0$ and therefore $x_n = f^n(x_0) = x_0$ for

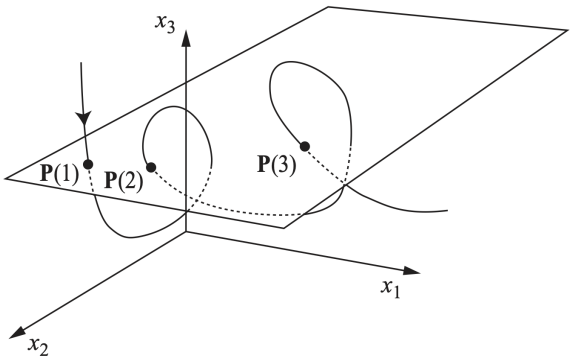


Figure 2.8 Example of dynamical system. Taken from [6].

all $n \in \mathbb{N}$. x_0 is a periodic point for period $k \geq 1$ if $x_k = f^k(x_0) = x_0$.

Dynamical systems can be divided into linear and nonlinear ones. Linear dynamical systems can be described by

$$x_{t+1} = Ax_t, \tag{2.30}$$

where A is the constant matrix. These systems obey superposition and homogeneity. More generally, nonlinear systems can be described by

$$x_{t+1} = f(x_t), \tag{2.31}$$

where f is the nonlinear mapping. These systems encompass intricate behavior that often defies direct analytical solutions.

2.2.2 Nonlinear Dynamical Systems

In this subsection, we explore nonlinear dynamical systems, a critical dimension within the framework of dynamical systems theory. Unlike their linear counterparts, nonlinear dynamical systems exhibit intricate behaviors that arise from complex interactions and feedback loops among system components. This subsection aims to delve into the fundamental principles governing nonlinear systems and describe their challenges.

Nonlinear dynamical systems represent a vast spectrum of phenomena where relationships between variables exhibit nonlinear interactions, challenging the traditional linear modeling approaches. These systems, prevalent in nature and various disciplines, unveil rich and complex behaviors that go beyond the predictable patterns of linear systems.

Nonlinear dynamical systems find applications across diverse domains, such as weather forecasting, population dynamics, neuroscience, and economics. However, modeling nonlinear systems poses significant challenges due to the complexity and lack of simple analytical solutions. Understanding nonlinear dynamical systems not only provides insights into natural phenomena but also poses challenges and opportunities for modeling and predicting complex real-world systems. Despite their challenges, these systems remain a crucial area of study, uncovering intricate behaviors and shaping various scientific and engineering fields.

Nonlinear dynamical systems are characterized by nonlinear equations governing their behavior, often described as $x_{t+1} = f(x_t)$, where x_t represents the state of the system at time step t. The function f introduces nonlinear terms, leading to intricate system dynamics that can be described by straightforward analytical solutions.

The inherent complexity of nonlinear systems gives rise to diverse behaviors, including bifurcations, chaos, and sensitive dependence on initial conditions, commonly known as the "butterfly effect." Small alterations in initial states can lead to vastly different outcomes, rendering long-term predictions challenging. Bifurcations in nonlinear systems signify qualitative changes in their behavior as system parameters vary. These changes often lead to the emergence of new stable states, limit cycles, or chaotic regimes. Phase space portraits aid in visualizing system trajectories, revealing attractors and their basin of attractions. Chaotic behavior, a hallmark of nonlinear systems, manifests as seemingly random yet deterministic trajectories, bounded within a deterministic system. The Lorenz system is a classic example exhibiting chaotic behavior, illustrating the system's sensitivity to initial conditions [49].

Understanding nonlinear systems often involves numerical simulations to capture their behaviors comprehensively. Tools like bifurcation diagrams, Poincaré maps and Lyapunov exponents aid in characterizing the system's behavior, identifying stability regions, chaotic regimes, and bifurcation scenarios.

2.2.3 Koopman Operator

In this subsection we provide a detailed exploration of the Koopman operator, a powerful mathematical for the analysis and modeling of nonlinear dynamical systems. The Koopman operator offers a unique perspective by transforming the evolution of observables in a dynamical system into an infinite-dimensional linear operator, facilitating the study of complex nonlinear dynamics in a linear framework. This subsection aims to describe the theoretical foundations of the Koopman operator, including its eigenfunctions, spectral properties and challenges. By delving into the intricacies of the Koopman operator, we aim to provide readers with a comprehensive understanding of its utility and significance in analyzing nonlinear dynamical systems.

Non-parametric Koopman Operator

The Koopman operator was named after Bernard O. Koopman, a mathematician known for his contributions to mathematical physics and control theory. In 1931, Koopman, along with mathematician John von Neumann, introduced what is now referred to as the Koopman–von Neumann theory. This theory laid the foundation for the Koopman operator, which is fundamental in the analysis of dynamical systems, particularly in nonlinear dynamics and control theory. The Koopman operator is a powerful mathematical tool used to analyze and understand the behavior of nonlinear dynamical systems. It provides an alternative perspective by mapping functions of the state space to functions that evolve linearly in a higher-dimensional space [21].

The Koopman operator linearizes the dynamics of the system in the function space, providing a linear representation of the evolution of observables. This linear perspective enables the analysis of nonlinear systems through linear methods. The Koopman operator has several important properties:

1. It is linear. This property is a direct consequence of the definition of the Koopman operator and it is what allows for powerful analytical techniques to be applied to the analysis of nonlinear dynamical systems.

2. It operates on the infinite-dimensional space. While the original dynamical system might live in a finite-dimensional state space, the operator acts on the infinite-dimensional space of observables. This is one of the key challenges in practical applications of Koopman theory.

3. Its eigenfunctions and eigenvalues provide valuable insights into the dynamics of the system. In particular, each eigenfunction defines a coherent structure in the system and the corresponding eigenvalue gives the frequency of oscillation or decay rate of that structure.

Consider a dynamical system described by $x_{t+1} = f(x_t)$, where $x_t \in X$ represents the state of the system at time step t, X is the state space of the system and f is nonlinear function. The Koopman operator $\mathbb{K}$

is an infinite-dimensional linear operator that acts on observables or functions $\Phi(x)$ defined on the state space

$$\mathbb{K}\Phi(x_t) = \Phi(f(x_t)). \tag{2.32}$$

As $x_{t+1} = f(x_t)$, we can rewrite above equation as

$$\mathbb{K}\Phi(x_t) = \Phi(f(x_t)) = \Phi(x_{t+1}). \tag{2.33}$$

This operator $\mathbb{K}$ maps functions from the state space X to functions $\Phi(x)$ that evolve linearly in a higher-dimensional function space, providing an alternative linear perspective to the dynamics of nonlinear systems. Fig 2.9 shows an example of such transformation.

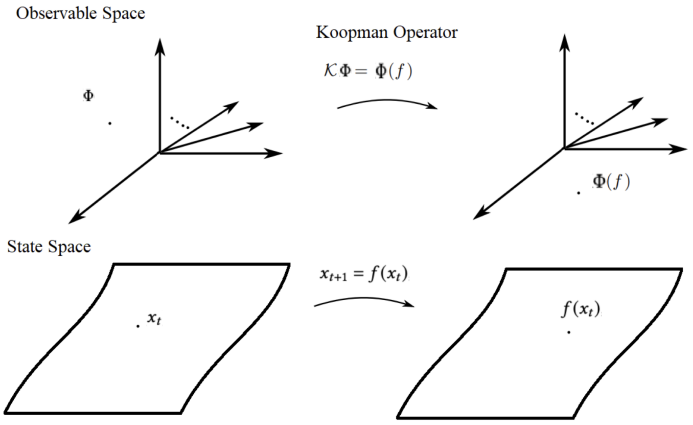


Figure 2.9 $\Phi(x)$ transforms the state space to the higher-dimensional observable space, where dynamics becomes linear. Adapted from [2].

The Koopman operator $\mathbb{K}$ possesses eigenfunctions $\phi_i(x)$ and associated eigenvalues λ_i, forming a spectral decomposition

$$\mathbb{K}\phi_i(x) = \lambda_i\phi_i(x). \tag{2.34}$$

These eigenfunctions serve as a basis in the function space, enabling the representation of nonlinear dynamics as a linear combination of these functions.

The Koopman operator finds applications in various fields, including control theory, nonlinear system identification, and data-driven modeling. Its ability to linearize nonlinear systems offers advantages in analyzing and predicting complex dynamical behaviors.

Despite its advantages, estimating the Koopman operator from data may face challenges in high-dimensional systems, noisy observations and computational complexity due to the infinite-dimensional nature of the operator. In practice, estimating the Koopman operator from data involves learning the dynamics of the system by observing trajectories. Techniques like Dynamic Mode Decomposition (DMD) and Extended Dynamic Mode Decomposition (EDMD) approximate the Koopman operator from data, extracting eigenfunctions and eigenvalues to understand the system's behavior.

In summary, Koopman operator provides a powerful mathematical framework for understanding the behavior of complex nonlinear dynamical systems by linearizing their dynamics in an infinite-dimensional function space, offering insights and analytical tools for studying nonlinear systems from a linear perspective. Its eigenvalues and eigenfunctions provide key insights into the dynamics of the system and practical methods such as DMD allow for its computation from data.

Parametric Koopman Operator

In the case of parametric Koopman Operator, dynamical system additionally depends on some set of static or time-varying parameters u [12]. By incorporating parameters, the dynamics can be represented more flexibly, allowing for better modeling of complex systems. Our dynamical system will be defined as

$$x_{t+1} = f(x_t, u), \tag{2.35}$$

where $x_t \in X$ is d-dimensional vector which represents the state of the system at time step t and $u_t \in U$ is a static q-dimensional vector. Or in the case where u_t changes dynamically in discrete steps

$$x_{t+1} = f(x_t, u_t), \tag{2.36}$$

where $x_t \in X$ is d-dimensional vector which represents the state of the system and $u_t \in U$ is q-dimensional vector at time step t.

The observation function can be outlined as follows

$$\Phi(x_t, u_t) = [\phi_1(x_t), ..., \phi_k(x_t), u_t], \tag{2.37}$$

where ϕ_i is i-th component of the observation function [23].

The Koopman Operator $\mathbb{K}$ approximates dynamics of the lifted system in the linear form as

$$\mathbb{K}\Phi(x_t, u_t) = \Phi(x_{t+1}, u_{t+1}). \tag{2.38}$$

We can also incorporate parameter u to approximate Koopman operator $\mathbb{K}$ as in [12]

$$\mathbb{K}(u_t)\Phi(x_t) = \Phi(x_{t+1}), \tag{2.39}$$

where we want to minimize the loss of the form

$$\mathbb{L} = ||\Phi(x_{t+1}) - \mathbb{K}(u_t)\Phi(x_t)||^2. \tag{2.40}$$

2.2.4 Approximating Koopman Operator

In this subsection, we research methodologies aimed to approximate the Koopman operator. The Koopman operator, while offering powerful insights into the dynamics of complex systems, often requires approximation methods to handle high-dimensional and nonlinear systems efficiently. We delve into three prominent approaches: Dynamic Mode Decomposition (DMD), Extended Dynamic Mode Decomposition (EDMD) and Neural Network-based methods. DMD extracts dominant modes of system behavior from data, EDMD extends this approach to handle nonlinearities and Neural Network methods leverage the capacity of deep learning models for learning complex system dynamics. Through a nuanced examination of these techniques, we aim to elucidate their principles, strengths and limitations.

Dynamic Mode Decomposition

Dynamic Mode Decomposition was introduced by Peter Schmid in 2010. Dynamic Mode Decomposition is a data-driven method used to extract spatial and temporal coherent structures and modes from high-dimensional and time-varying data. It approximates the modes of the Koopman operator, which represents nonlinear dynamics, and computes eigenvalues and eigenvectors of a linear model that approximates the underlying dynamics [42].

The step-by-step process of DMD is decribed in Algorithm 1.

Algorithm 1: Dynamic Mode Decomposition

Data: Snapshot data $X = [x_1, x_2, ..., x_N]$, where x_i represents the state of the system at time step t_i

Step 1: Snapshot Data Collection. Arrange snapshots into a data matrix X;

Step 2: Construct Data Matrices. Divide snapshots into two consecutive matrices X_1 and X_2 by extracting consecutive snapshots;
$X_1 = [x_1, x_2, ..., x_{N-1}]$ and $X_2 = [x_2, x_3, ..., x_N]$;

Step 3: Singular Value Decomposition (SVD) of X_1. Compute SVD of $X_1 = U\Sigma V^T$, where U and V contain left and right singular vectors, and Σ is a diagonal matrix of singular values;

Step 4: Compute Matrix A. Matrix A is approximated by linear mapping from X_1 to X_2 using least-squares method: $A = X_2 X_1^+$, where X_1^+ is the Moore-Penrose pseudo-inverse of matrix X_1;

Step 5: Eigenvalue Decomposition of Matrix A. Compute eigenvalues $\Lambda = diag(\lambda_1, ..., \lambda_N)$ and eigenvectors $W = [w_1, ..., w_N]$ of matrix A: $AW = \Lambda W$;

Step 6: Reconstruction of Dynamics. Reconstruct dynamics using the equation:
$x_k \approx A^k x_1 = (W\Lambda W^{-1})^k x_1 = W\Lambda^k W^{-1} x_1 = W\Lambda^k b$, where $b = \sum_{j=1}^{N} \lambda_k b_j v_j$, and $b = W^{-1} x_1$ [**20**].

Dynamic Mode Decomposition (DMD) offers several advantages in the analysis of dynamic systems. A data-driven approach, it applicable to handle high-dimensional and noisy datasets without requiring prior knowledge of the governing equations of the system. DMD excels in capturing the dominant modes and coherent structures inherent in dynamical systems, providing a clear representation of their essential features. Additionally, it stands out for offering valuable insights into system dynamics without relying on explicit model equations, making it particularly useful in scenarios where obtaining precise mathematical formulations may be challenging.

However, it is crucial to consider the limitations of DMD. Its effectiveness can be sensitive to the choice of snapshots and parameters, potentially impacting the accuracy of the extracted modes. Interpreting the extracted modes accurately may require domain knowledge, necessitating users to possess a certain level of understanding of the specific field or system under investigation. Furthermore, DMD assumes linearity in the relationship between snapshots, constraining its applicability to scenarios where the snapshots are already linearly related and limiting its effectiveness in capturing nonlinear dynamics.

Dynamic Mode Decomposition is a valuable tool for extracting spatial and temporal patterns and understanding the dominant modes governing the dynamics of complex systems. Its ability to analyze data-driven systems without a priori models makes it widely applicable in various scientific fields. However, careful consideration of parameters and interpretation of results are crucial for its effective application.

Extended Dynamic Mode Decomposition

Extended Dynamic Mode Decomposition is an extension of Dynamic Mode Decomposition. It was introduced in 2015 by M.O. Williams. Standard Dynamic Mode Decomposition works if the snapshots are already linearly related. EDMD addresses this limitation of standard DMD by incorporating nonlinear observables or features derived from the state variables. By transforming the state space into a higher-dimensional feature space, EDMD can linearize nonlinear dynamical systems [50].

Extended Dynamic Mode Decomposition is represented in Algorithm 2.

Extended Dynamic Mode Decomposition (EDMD) emerges with distinctive advantages in nonlinear dynamical systems analysis. Still a data-driven approach, EDMD operates efficiently without demanding a prior understanding of the governing equations of the system. Notably, it excels in enhancing accuracy

Algorithm 2: Extended Dynamic Mode Decomposition

Data: Snapshot data $X = [x_1, x_2, ..., x_N]$, representing the state of the system at different time instances.

Step 1: Snapshot Data Collection. Gather a sequence of snapshot data $X = [x_1, x_2, ..., x_N]$;

Step 2: Feature Engineering. Map the state space X to a higher-dimensional space by defining nonlinear observables or features: $\Phi(X) = [\phi(x_1), \phi(x_2), ..., \phi(x_N)]$;

Step 3: Construct Data Matrices. Construct two matrices Φ_1 and Φ_2 using the transformed feature vectors: $\Phi_1 = [\phi(x_1), \phi(x_2), ..., \phi(x_{N-1})]$ and $\Phi_2 = [\phi(x_2), \phi(x_3), ..., \phi(x_N)]$;

Step 4: Linear Mapping in Feature Space. Approximate the linear mapping matrix A by linearly relating Φ_1 to Φ_2 using a least-squares approach: $A = \Phi_2\Phi_1^+$, where Φ_1^+ is the Moore-Penrose pseudo-inverse of matrix Φ_1;

Step 5: Eigenvalue Decomposition of Matrix A**.** Compute eigenvalue $\Lambda = diag(\lambda_1, ..., \lambda_N)$ and eigenvectors $W = [w_1, ..., w_N]$ of matrix A: $AW = \Lambda W$;

Step 6: Reconstruct Dynamics in Feature Space. Reconstruct dynamics using the equation: $\phi(x_k) \approx A^k\phi(x_1) = (W\Lambda W^{-1})^k\phi(x_1) = W\Lambda^k W^{-1}\phi(x_1) = W\Lambda^k b$, where $b = \sum_{j=1}^{N} \lambda_k b_j v_j$, and $b = W^{-1}\phi(x_1)$;

Step 7: Reconstruct Dynamics in State Space. If required, transform the reconstructed dynamics $\phi(x_k)$ back to the original state space using the inverse transformation ϕ^{-1}

when capturing nonlinear dynamics, marking a significant improvement in its applicability to systems manifesting complex nonlinearities. This signifies EDMD as a robust choice for scenarios where traditional methods may fall short.

Nevertheless, it is essential to acknowledge the challenges associated with EDMD. The selection of nonlinear features proves critical and a suboptimal choice can lead to inaccurate results, emphasizing the importance of careful consideration during the modeling process. Additionally, EDMD may become computationally expensive when applied to high-dimensional systems or when dealing with large dictionaries of observables, potentially posing constraints on its scalability. Users may also find that expertise in feature selection and engineering becomes a prerequisite for maximizing the effectiveness of EDMD in practical applications. These considerations underscore the need for a balanced evaluation of its advantages and limitations in specific analytical contexts.

EDMD extends the capabilities of DMD by incorporating nonlinear features, enabling a more accurate approximation of nonlinear dynamical systems. While it offers significant advantages in capturing complex dynamics, it requires careful selection and engineering of nonlinear features and involves increased computational complexity compared to traditional DMD.

Neural Network Approaches

One of the disadvantages of Extended Dynamic Mode Decomposition is that it requires manual feature engineering and selection. Approximating the Koopman operator using neural network approaches offers several advantages compared to Extended Dynamic Mode Decomposition:

1. Neural network approaches are inherently designed to capture and represent nonlinear dynamics more effectively [24].

2. Neural networks provide flexibility in feature representation. These methods can automatically learn and extract relevant features from the data, enabling adaptive representations that might be more descriptive of the system's behavior than manually chosen features in EDMD.

3. Neural networks have the capacity to learn complex mappings between state spaces, allowing for more accurate approximations of the Koopman operator compared to DMD and EDMD methods.

For instance, one of the improvements over EDMD is Extended Dynamic Mode Decomposition with dictionary learning. In this approach Extended Dynamic Mode Decomposition is combined with a trainable dictionary represented by artificial neural network. Authors used a simple feed-forward 3-layer neural network as the approximator for Φ. Figure 2.10 shows an architecture of this neural network.

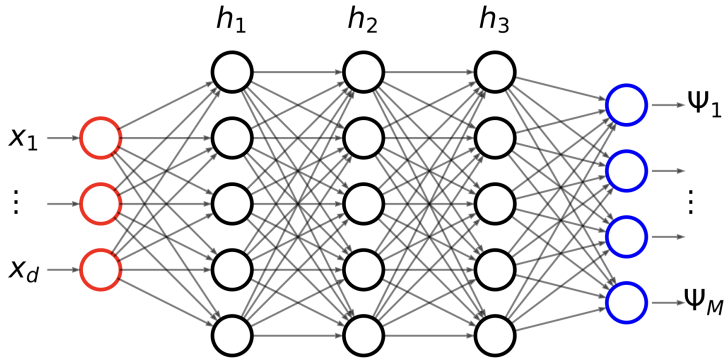


Figure 2.10 Architecture of neural network which approximates trainable dictionary $\Phi(x)$. Taken from [24].

This neural network is described by

$$\Phi(x) = W_{out} h_3 + b_{out}, \tag{2.41}$$

$$h_{k+1} = tanh(W_k h_k + b_k), k = 0, 1, 2, \tag{2.42}$$

where $h_0 = x$, W and b are trainable parameters. This approach enables a minimal set of refined dictionary functions to cover a linear subspace where an accurate approximation of the Koopman operator can be achieved.

An extension of [24] is parametric Koopman decomposition with neural networks (PK-NN). PK-NN applies previous algorithm to the parametric case where $x_{t+1} = f(x_t, u_t)$ and $\Phi(x_{t+1}) = K(u_t)\Phi(x_t)$. Here $U \subseteq R^{N_u}$ is a set of parameters and $u_t \in U$ can remains static or changes dynamically with time steps. The observable functions Φ is defined as a neural network and the parameters are tuned with training data to achieve high prediction accuracy [12]. Figure 2.11 shows a graphical representation of the PK-NN algorithm.

Here both the projected Koopman operator and the state dictionary are parameterized using neural networks. Both of these components are trained simultaneously using trajectory data. This allows to handle high-dimensional and strongly nonlinear data, making it particularly suitable for large-scale data-driven prediction and control problems.

2.3 Related Work

In this section, we research the application of the Koopman operator in the analysis of neural networks. The Koopman operator, traditionally employed in dynamical systems theory, finds innovative utility in deciphering the underlying dynamics of neural networks. By leveraging the Koopman operator, we aim to unravel the intricate transformations and representations within neural networks, shedding light on the underlying mechanisms that govern their functionality. This section delves into practical implications of utilizing the Koopman operator to analyze neural network behavior, offering a unique perspective that enhances our understanding of the complex dynamics inherent in these sophisticated learning architectures.

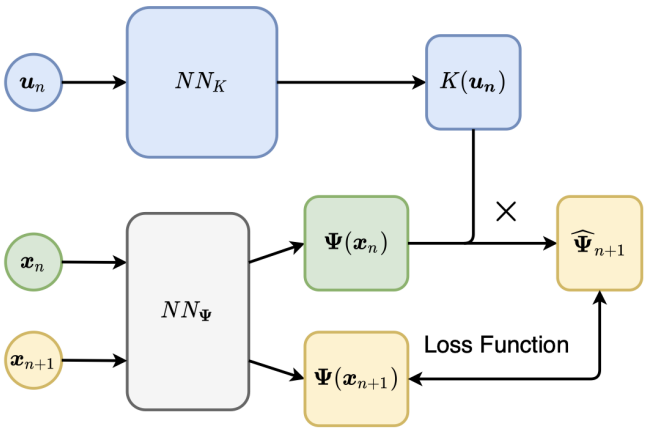


Figure 2.11 Architecture of neural network in the PK-NN algorithm. Trainable parameters of Koopman operator $K(u, W_k)$ are learned via NN_K neural network and network NN_Φ is used to obtain dictionaries $\Phi(x; W_\phi)$. Taken from [12].

Through this exploration, we seek to bridge the realms of dynamical systems theory and neural networks.

Recently there were a lot of approaches to describe neural networks as dynamical systems via Koopman operator. In [8] authors represent training process of neural network as a discrete dynamical flow with the number of iterations acting as the discrete time parameter. In this flow the loss function L (or its independent parts) and the individual weights of the neural network are some of the dynamical quantities of interest. The aim of the training process of neural network is to find a set of weights that minimize the loss L to the best possible value. From the prospective of dynamical systems theory the value of loss function L is depends on the evolution of different weights of neural network (state space parameters) as the number of training iterations increases (temporal parameter of the system). The goal is to describe the standard training process as dynamical system using tools of Koopman Operator theory. Let F be the discrete mapping governing the dynamics of weights w with the training iteration t being a temporal parameter. Therefore, the training rule K for a neural network is a discrete dynamical map on L, describing the evolution of loss value during training process

$$L(t) = K^t L(w_0) = L(F^t(w_0)), \tag{2.43}$$

where w_0 is the set of weights for the neural network at first iteration and t is the number of training iterations from the original point. Also the values of weights w at each iteration t of the training proccess can be governed by some training rule K. Let now K represent the discrete map for weights w and F discrete mapping describing the dynamics of loss L. We have

$$w(t) = K^t w(L_0) = w(F^t(L_0)), \tag{2.44}$$

where L_0 is the original value of the loss component at the beggining of training process. These equations clearly demonstrate that the training of the neural network can be represented as dynamical system and its evolution can be described via Koopman operator theory.

In [25] authors address the problem of credit assignment (CAP) of neural networks from a linear dynamics perspective via Koopman operator theory. The credit assignment problem in neural networks deals with determining the contribution of each component of the network to the final outputs. This involves assessing how much each part of the network impacts the end results. Authors define a general neural network N with l layers. This neural network can be described by the composition of transformation of each layer

$$f(x) = f_l \circ f_{l-1} \circ ... \circ f_2 \circ f_1(x) = \sigma_l(W_l \cdots \sigma_2(W_2\sigma_1(W_1 x + b_1) + b_2) \cdots + b_l), \tag{2.45}$$

where W_i and b_i are weight matrices and biases of the i-th layer and σ_i represents nonlinear activation function, such as ReLU, Sigmoid, Tanh or other. This neural network is partitioned into m blocks $B_1, B_2, ..., B_m$, where $1 \leq m \geq l$and each block contains the mappings represented by a network layer or composition of layers. Suppose the block B_i contains the j-th to k-th layers, the function of block B_i is be defined as

$$f_i = \sigma_k(W_k \cdots \sigma_j(W_j\sigma_{j-1}(W_{j-1}x_{j-1} + b_{j-1}) + b_j) \cdots + b_k). \tag{2.46}$$

Each block B_i can be viewed as dynamical subsystem and each corresponding transformation function f_i can be linearized via Koopman operator by

$$y_{k+1} = f_i(y_k), \tag{2.47}$$

$$y_{k+1} \approx K_i y_k, \tag{2.48}$$

where $y(k)$ is the output of block B_{i-1} and K_i is the Koopman operator for a block B_i, obtained via DMD. Thus, the transformation of all blocks can be defined as

$$K = K_m K_{m-1} \cdots K_2 K_1. \tag{2.49}$$

Finally, we need to measure the contribution of each K_i to K. By employing backward propagation and the Jacobian matrix, we can calculate the partial derivative of K with respect to K_i. The absolute value of its determinant may be interpreted as the block sensitivity of K_i, which quantifies the impact of the change of Ki on K and can be represented as follows

$$BS_i = abs\left(\left|\frac{\partial K}{\partial K_i}\right|\right). \tag{2.50}$$

This approach provides a credit assessment of specific layers of neural network or its modules.

In [9] Koopman operator was also employed to predict the evolution of weights and biases for neural networks. In this case neural network was represented as discrete dynamical system

$$w_{t+1} = f(w_t), \tag{2.51}$$

where w_t are weights and biases of neural network at iteration t and their evolution is governed by a dynamical map f. The Koopman operator K was utilized to predict values of weights and biases for the next iteration $t + 1$

$$w_{t+1} = Kw_t. \tag{2.52}$$

Through several experiments authors verified that Koopman training is able to correctly approximate the action of standard training algorithms such as gradient descent. Morevover, the analysis of complexity showed that Koopman training is much faster, making it a compelling alternative to conventional optimization techniques.

In [27] authors also considered a process of training neural network as discrete dynamical system and successfully applied Koopman operator and Dynamic Mode Decomposition to determine when to terminate training, prune network weights without losing performance and determine the required number of layers in Hierarchical SVR model for a mulitscale signal.

Koopman theory was utilized in [33] for analysis of neural networks. Particularly, authors investigated sequential neural models via Koopman operator and its practical applications. Their technique employs a hidden state representation that reduces dimensionality and calculates a linear mapping from the current to the next hidden state. Authors determined linear estimates of the hidden state paths through basic

matrix-vector multiplications. Furthermore, they pinpoint the main characteristics of the dynamic system and analyze their influence on inference and prediction. Their findings on sentiment analysis task and ECG classification challenge offer straightforward yet precise explanations of the underlying dynamics and behavior of the recurrent neural models.

A new point of view on language modeling was presented in [29]. LLMs are considered as dynamical systems, where the next state depends on the previous states. Techniques such as One-Hot-Encoding are used to represent each words as a unit vector in a space whose dimension is matching the vocabulary size. In the Koopman operator framework, these vectors become indicator observables for a set of words. These observables-features are combinations of time-delayed indicator observables. The transformer block in LLMs works on a time-ordered feature matrix, sequentially transforming individual feature sequences. Finally, these transformed sequences are nonlinearly combined.

This aligns precisely with the Koopman operator framework's principles: embedding abstract elements into a Euclidean space and identify functions within this embedding to efficiently predict how the dynamical system evolves over time. For instance, employing time-delayed observables, a common approach in LLMs, allows for filtering and creating linear combinations of these observables, mirroring the initial phase in the transformer model. The subsequent stage aims for a nonlinear transformation of these observables, resulting in a linear representation when discrete spectrum eigenvalues are identified and a nonlinear representation for continuous spectra. Unlike LLMs, Koopman operator-based architectures often maintain computational efficiency due to some predefined transformations, contrasting with learned transformations in LLMs.

3 Explaining Transformer Operations Through Koopman Operator Theory

In this chapter, we discuss the exploration of Transformer operations through Koopman Operator theory. We analyze the language sequence as a time-ordered set of tokens, akin to a state space evolving over discrete time steps. The transition between tokens mirrors state transitions in dynamical systems, encapsulating inherent dependencies and relationships. Our focus lies on the decoder block within the Transformer architecture. First, we explore its operations in a next token prediction task using Koopman Operator theory. Specifically, we investigate whether the self-attention mechanism can be viewed as a dictionary function, and the Multi-Layer Perceptron (MLP) layer as a potential inverse function. Subsequently, we extend our analysis to the encoder-decoder architecture in a basic sequence-to-sequence task, leveraging parametric Koopman Operator theory. We then describe practical experiments conducted to validate our theoretical framework, focusing first on training the modified Transformer model on a specific task and evaluating alignment with the Koopman Operator framework. Results provide insights into modification effectiveness and performance impact. Following this, we detail experiments aimed at refining modifications to better align with Koopman Operator theory. Through iterative experimentation and parameter adjustment, we aim to improve model performance while maintaining fidelity to the theoretical framework. We conclude with a discussion of limitations and future work, contributing to ongoing research on understanding and enhancing Transformer capabilities through Koopman Operator theory.

3.1 Explaining Decoder architecture through Koopman Operator Theory

In this section, we delve into the operations of the decoder block through the lens of Koopman Operator theory. Our focus on the decoder architecture stems from its pivotal role in generating output sequences in sequence-to-sequence tasks, making it a crucial component of many natural language processing and machine translation models. Unlike the encoder, which primarily focuses on encoding input sequences into fixed-length representations, the decoder operates in a conditional manner, dynamically generating each output token based on previous predictions and encoded input representations. This dynamic nature introduces unique challenges and opportunities for analysis, as the decoder's behavior is intricately tied to its interactions with both the input sequence and previously generated output tokens. By examining the decoder's operations within the framework of Koopman Operator theory, we aim to uncover fundamental insights into its dynamic behavior and information processing mechanisms.

Model Simplifications

In this section we assume that we have only decoder block from the Transformer model. We also simplify the decoder block by following considerations:

1. We remove normalization layers from the decoder block. This simplification is justified by the observation that normalization layers do not fundamentally alter the topology of the data manifold being processed. While in practice, normalization layers contribute to stabilizing training and improving convergence by controlling the scale and mean values of activations, their removal in our simplified model is motivated by the desire to streamline the architecture without the added complexity of normalization.

2. We remove residual connections in the decoder block. The rationale behind this simplification lies in the theoretical equivalence between directly learning the mapping $g(x)$ and approximating it as

$x + f(x)$. While residual connections facilitate gradient flow and ease the optimization process, particularly in deeper architectures, their exclusion simplifies the model by reducing parameter dependencies and computational overhead.

3. We have only one attention head in the first self-attention block. We opt for a single attention head to simplify the attention mechanism and reduce parameter overhead. While multi-head attention offers increased modeling capacity and facilitates capturing diverse interaction patterns, a single attention head suffices for basic sequence modeling tasks and helps maintain model simplicity.

4. We remove the second self-attention block from the decoder. This simplification is based on the consideration that in original architecture in the second self-attention block the result from the encoder acts as keys and values. In our settings we do not have encoder block, therefore it is sufficient to keep only first self-attention block in the decoder, which encodes input sequence.

5. All the weights of the self-attention and feed-forward layers are shared between each stack of the decoder. Thus, the operations inside decoder block will act in a recurrent way. This is necessary to describe the dynamics inside decoder block through Koopman Operator theory. Thus, we need same dictionary function and same inverse function across all stacks of decoder.

6. We omitted the bias term before the nonlinearity in the feed-forward layer of the decoder block. This simplification reduces the number of parameters in the model and allows us to focus on the core operations of the feed-forward layer without the additional complexity introduced by bias terms.

Hence, in our decoder we have a single-head self-attention, represented by equations 2.20 - 2.24 following by feed-forward layer, represented by equation 2.25. Figure 3.1 shows architecture of our decoder block.

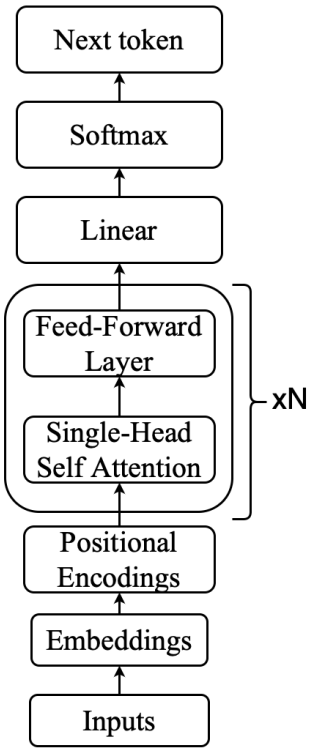


Figure 3.1 Decoder block.

Exploring Dynamics inside Decoder Block through Koopman Operator Theory

From the dynamical system point of view we have a following problem setting for the next token prediction task:

1. At each time step s our state space x_s is defined by the sequence of k tokens $x_s = (t_1, ..., t_k)$
2. Our goal is to predict the next state space of the system x_{s+1} which is also defined by a sequence of k tokens $x_{s+1} = (t_2, ..., t_{k+1})$, where t_{k+1} is the next token and we need to predict this token.

Thus, we will have a sequence of k preceding tokens $x_s = (t_1, ..., t_k)$ at each time step s as input to our decoder block. Each token in the sequence is initially represented as a vector through an embedding layer. This layer maps each token to a high-dimensional vector space, allowing the model to capture semantic relationships between tokens. We assume that each token is encoded into d-dimensional vector via embedding layer. In our Transformer model, devoid of recurrence and convolutional layers, leveraging the sequential order of tokens within the input sequence poses a unique challenge. Without these traditional architectural elements that inherently encode sequential information, such as hidden states in recurrent networks or convolutional filters in convolutional neural networks, our model lacks explicit knowledge of token order. Consequently, to enable our model to effectively utilize the order of the sequence, we must inject positional information about the relative or absolute position of tokens. To address this requirement, we incorporate "positional encodings" into the input embeddings at the bottoms of both the decoder block. These positional encodings are designed to augment the embeddings with information regarding the position of each token in the sequence. Crucially, the positional encodings possess the same dimensionality as the embeddings, facilitating seamless integration by simply summing the two vectors. Thus, after the embedding and positional encoding layers each token t_i is mapped to a higher-dimensional vector in R^d. These steps aligns with the lifting of initial state of dynamical system to a higher-dimensional state space in Koopman Operator theory. Thus, embedding and positional encoding layers can be considered as a lifting function Φ_1, mapping tokens to a dictionary space. This can be represented by

$$\Phi_1 = \text{Embed}(t_i) + \text{PE}(i), \tag{3.1}$$

where *Embed* is the embedding layer and *PE* is positional encoding layer, described by equation 2.18. Thus, we have $z_s = \Phi_1(x_s)$, where $z_s \in R^{k \times d}$.

In the next step we input our state of the system z_s into single-head self-attention mechanism. Our self-attention mechanism processes input

$$z_{SHA} = \text{softmax}(z_s W_Q W_K^T z_s^T) z_s W_V, \tag{3.2}$$

where $z_s \in R^{k \times d}$ is state of our system in a higher-dimensional space and $W_Q, W_K, W_V \in R^{d \times d}$ are learnable weight matrices. It is easy to see that $z_{SHA} \in R^{k \times d}$ as

$$\underset{k\times d}{\mathrm{z_s}}\ \underset{d\times d}{\mathrm{W_Q}} = \underset{k\times d}{\mathrm{Q}}, \tag{3.3}$$

$$\underset{d\times d}{\mathrm{W_K^T}}\ \underset{d\times k}{\mathrm{z_s^T}} = \underset{d\times k}{\mathrm{K^T}}, \tag{3.4}$$

$$\underset{k\times d}{\mathrm{Q}}\ \underset{d\times k}{\mathrm{K^T}} = \underset{k\times k}{\mathrm{QK^T}}, \tag{3.5}$$

$$\underset{k\times d}{\mathrm{z_s}}\ \underset{d\times d}{\mathrm{W_V}} = \underset{k\times d}{\mathrm{V}}, \tag{3.6}$$

$$\underset{k\times k}{\mathrm{softmax(QK^T)}}\ \underset{k\times d}{\mathrm{V}} = \underset{k\times d}{\mathrm{z_{SHA}}}. \tag{3.7}$$

Self-attention mechanism operates on a sequence of input embeddings, assigning varying degrees of importance to each token embedding based on its relevance to the context. We can see self-attention

mechanism as another lifting function Φ_2. The main aim of the lifting function in the Koopman Operator Theory is to expand the dimensionality of the state space. This expansion allows for a more comprehensive representation of the system's dynamics, capturing complex relationships and interactions among variables. The lifting function may enhance interactions between variables by introducing higher-order terms or transformations. This enhancement enables the representation of nonlinear dynamics and dependencies that may be present in the system. This is what self-attention mechanism actually does. The primary function of the self-attention mechanism is to enhance and supress interactions between different tokens in the input sequence. This is done by assigning attention weights to each token based on its relevance to other tokens in the input sequence.

By attending to different parts of the input sequence, the self-attention mechanism enables tokens to interact and exchange information. Moreover, the attention scores computed by the self-attention mechanism are non-linear functions of token embeddings. By attending to different parts of the input sequence with varying degrees of emphasis, the self-attention mechanism introduces nonlinear transformations that capture complex relationships and dependencies among tokens. Self-attention mechanism does not alter the underlying temporal or sequential structure of the input sequence, but rather enhances interactions and dependencies between tokens. Thus,

$$\Phi_2(z_s) = \text{softmax}(z_s W_Q W_K^T z_s^T) z_s W_V \tag{3.8}$$

and $z_{SHA} \in R^{k \times d}$ will be a representation of z_s in the "lifted" space.

Next we have a feed-forward layer

$$z_{FFL} = max(0, z_{SHA} W_1) W_2 + b_2, \tag{3.9}$$

where $z_{SHA} \in R^{k \times d}$ and $W_1 \in R^{d \times d_f}$ and $W_2 \in R^{d_f \times d}$ are learnable weight matrices.

If we consider z_{SHA} as the initial state of our system in the dictionary space lifted by $\Phi_2(z_s)$ and write it as z^0_{SHA}, we can see the dot product of z^0_{SHA} and W_1 as the prediction of the future state of the system in the dictionary state space. This can be described by

$$z^0_{SHA} W_1 = z^1_{SHA}. \tag{3.10}$$

However, we need to notice that the result of dot product of the z^0_{SHA} and W_1 will be in $R^{k \times d_f}$

$$\underset{k \times d}{z^0_{\text{SHA}}} \underset{d \times d_f}{W_1} = \underset{k \times d_f}{z^1_{\text{SHA}}}. \tag{3.11}$$

To align with the theory of Koopman Operator, we need z^0_{SHA} and z^1_{SHA} to be in the same dictionary space. Therefore, we need d equals to d_f. In this case $z^0_{SHA} \in R^{k \times d}$ and $z^1_{SHA} \in R^{k \times d}$.

Given that, we can consider the rest of the equation 3.9 as $\Psi_2(z_{SHA}) = max(0, z^1_{SHA}) W_2 + b_2$. This is the inverse function of $\Phi_2(z_s)$, mapping z^1_{SHA} back to the original state space $z_s \in R^{k \times d}$.

In terms of Koopman Operator theory we have a following dynamical system inside decoder block

$$\begin{aligned} z_s^{i+1} &= f_{dec}(z_s^i), && (3.12)\\ \Phi_2(z_s^i) W_1 &= \Phi_2(f_{dec}(z_s^i)) = \Phi_2(z_s^{i+1}), && (3.13)\\ z_s^{i+1} &= \Psi_2(\Phi_2(z_s^i) W_1), && (3.14) \end{aligned}$$

where $z_s^0 = \Phi_1(x_s)$.

If we have N stacks of decoder then

$$z_s^N = f_{dec} \overbrace{\odot ... \odot}^{Ntimes} f_{dec}(z_s^0), \tag{3.15}$$

where $z_s^0 = \Phi_1(x_s)$. And evolution inside the decoder block through Koopman Operator theory is represented in Algorithm 3.

Algorithm 3: Evolution inside decoder block through Koopman Operator theory

$i \leftarrow 0$;
$z_s^i \leftarrow \Phi_1(x_s)$;
while $i < N$ **do**
  $z_{SHA}^i \leftarrow \Phi_2(z_s^i)$;
  $z_{SHA}^{i+1} \leftarrow z_{SHA}^i W_1$;
  $z_s^{i+1} \leftarrow \Psi_2(z_{SHA}^{i+1})$;
  $i \leftarrow i + 1$
end

After N iterations of decoder we are transforming our state z_s^N into original state space by applying affine and softmax layers to it. This is represented by

$$z_s^{\text{linear}} = \text{Linear}(z_s^N) = z_s^N W_{\text{linear}} + b_{\text{linear}}, \tag{3.16}$$

where $W_{\text{linear}} \in R^{d \times v}$ and v is the size of vocabulary and

$$(z_s^{\text{softmax}})_{ij} = \frac{e^{(z_s^{\text{linear}})_{ij}}}{\sum_{j=1}^{v} e^{(z_s^{\text{linear}})_{ij}}}, \tag{3.17}$$

where $z_s^{\text{linear}} \in R^{k \times v}$ is the matrix obtained after the final affine layer.

Finally, we are going to the next state in the initial state space

$$(x_{s+1})_i = \underset{j}{\text{argmax}}((z_s^{\text{softmax}})_{ij}) \tag{3.18}$$

where $z_s^{\text{softmax}} \in R^{k \times v}$ is the matrix obtained after the softmax layer.

Therefore, equations 3.16 - 3.18 can be considered as the Ψ_1 - inverse function of Φ_1.

In other words here we have two dynamical systems:

1. The outer dynamical systems which acts in the state space of token sequences $x_s = (t_1, ..., t_k)$ with lifting function to a higher dimensional space Φ_1
2. The inner dynamical system which acts inside decoder block in the state space of token embeddings $z_s \in R^{k \times d}$ with lifting function $\Phi_2(z_s)$

After N steps of inner dynamical function we go back to the state space of outer dynamical function. The evolution of the decoder block through Koopman Operator theory is described in Algorithm 4.

3.2 Explaining Encoder-Decoder Architecture through Parametric Koopman Operator

In this section, we embark on an exploration of the encoder-decoder architecture within the Transformer model through the lens of Koopman Operator theory. The encoder-decoder architecture lies at the heart

Algorithm 4: Evolution for the next token prediction through Koopman Operator theory

Assume we want to predict m future tokens.
$x_s \leftarrow (t_1, ..., t_k)$ - is the state of outer dynamical system at time step s, described by k preceding tokens. k - is the context window of the model;
while $s < (s + m)$ **do**
  $z_s^0 \leftarrow \Phi_1(x_s)$ - is the initial state of inner dynamical system;
  $i \leftarrow 0$.
  while $i < N$ **do**
    $z_{SHA}^i \leftarrow \Phi_2(z_s^i)$;
    $z_{SHA}^{i+1} \leftarrow z_{SHA}^i W_1$;
    $z_s^{i+1} \leftarrow \Psi_2(z_{SHA}^{i+1})$;
    $i \leftarrow i + 1$.
  end
  $x_{s+1} \leftarrow \Psi_1(z_s^N)$;
  $s \leftarrow s + 1$.
end

of the Transformer model, enabling it to perform a wide array of sequence-to-sequence tasks, including machine translation, text summarization, and question answering. While the Transformer architecture has proven to be remarkably successful in capturing long-range dependencies and contextual information through self-attention mechanisms, understanding the dynamics of information flow between the encoder and decoder remains a challenging yet crucial endeavor. By applying Koopman Operator theory to analyze the encoder-decoder interactions, we can gain deeper insights into the underlying principles governing the transformation of input sequences to output sequences.

In various tasks employing the Transformer model, the encoder and decoder modules are fundamental components for processing input data and generating output sequences. Initially, the encoder receives the input data and transforms it into a sequence of high-dimensional vectors. These vectors encode the semantic and contextual information of the input data through self-attention mechanisms and feed-forward neural networks. Subsequently, these encoded representations are passed to the decoder, which is responsible for generating the output sequence. Leveraging the information from the encoder and its own previously generated tokens, the decoder predicts each token in the output sequence sequentially. This prediction process is facilitated by a masked multi-head attention mechanism, allowing the decoder to focus on relevant parts of the input sequence while generating the output. By ensuring that predictions rely solely on preceding tokens within the sequence, this mechanism maintains coherence during generation. This iterative process continues until the entire output sequence is generated, with the decoder's attention mechanism facilitating an understanding of complex relationships within the data, thereby enabling the generation of accurate and contextually relevant outputs.

Therefore, we have a following problem setting in the realm of dynamical system theory:

1. At each time step s our state space x_s^{target} is defined by k preceding tokens in the output sequence $x_s^{\text{target}} = (t_1, ..., t_k)$ generated by decoder and a sequence of tokens $x_s^{\text{source}} = (t_1, ..., t_l)$ from the input sequence.

2. Our goal is to predict the next state space of the system x_{s+1}^{target} which is also defined by a sequence of k tokens $x_{s+1}^{\text{target}} = (t_2, ..., t_{k+1})$, where t_{k+1} is the next token in the target sequence and we want to predict this token.

Model Architecture

In this section we have encoder-decoder blocks architecture of the Transformer model. We simplify encoder and decoder blocks as earlier by following considerations:

1. We remove normalization layers from the encoder and decoder blocks. This decision is supported by the understanding that normalization layers do not inherently modify the underlying structure of the data manifold under consideration. Although in practical scenarios, normalization layers play a role in enhancing training stability and facilitating convergence by regulating activation scale and mean values, their omission in our simplified model is driven by the intention to simplify the architecture without introducing the additional intricacies associated with normalization.

2. We remove residual connections from the encoder and decoder blocks. This decision is based on the conceptual equivalence between directly learning the mapping $g(x)$ and approximating it as $x+f(x)$. Although residual connections aid in maintaining smooth gradient flow and simplifying optimization, especially in deeper architectures, their removal simplifies the model by reducing dependencies on parameters and computational complexity.

3. We have only one attention head in the self-attention layers of the encoder and decoder blocks. This choice is made to simplify the attention mechanism and decrease the number of parameters required. While employing multiple attention heads enhances the model's ability to capture complex interaction patterns and increases modeling capacity, using a single attention head is adequate for basic sequence modeling tasks and contributes to maintaining the simplicity of the model.

4. We remove the first self-attention block from the decoder block. Removing the first self-attention layer in the decoder aligns with the principle of focusing computational resources on the most relevant and informative parts of the model architecture. In many cases, the encoder's self-attention layers are sufficient for capturing the contextual information necessary for generating high-quality output sequences. By eliminating redundant self-attention computations in the decoder, we optimize the model's efficiency and improve training and inference speed without sacrificing performance.

5. The weights within both the self-attention and feed-forward layers are shared across each stack of the decoder. Consequently, the operations within the decoder block exhibit a recurrent behavior. This sharing of weights is essential for characterizing the dynamics within the decoder block using Koopman Operator theory. Therefore, it is imperative to have consistent dictionary and inverse functions across all stacks of the decoder to effectively analyze its dynamics.

6. We excluded the bias term preceding the nonlinearity in the feed-forward layer of the decoder block. This simplification decreases the model's parameter count and enables a more concentrated examination of the fundamental operations of the feed-forward layer, free from the added intricacies associated with bias terms.

Thus, our encoder and decoder blocks have single-head self-attention layers, represented by equations 2.20 - 2.24 following by feed-forward layers, represented by equation 2.25. The result of the encoder block serves as the keys and values to the self-attention mechanism in the decoder block. Figure 3.2 shows architecture of our simplified Transformer model.

Investigating Encoder-Decoder Block Dynamics via Parametric Koopman Operator Theory

In the Transformer model utilized for various tasks, both the encoder and decoder blocks are responsible for handling input sequences. The encoder block of the Transformer model receives an input sequence, which is tokenized into individual units. These tokens undergo transformation through an embedding layer, converting them into high-dimensional vectors to capture semantic relationships. Subsequently, positional encodings are incorporated into the token embeddings to denote each token's position within the sequence. The encoder block then processes this amalgamation of token embeddings and positional encodings to generate contextualized representations for each token. Through self-attention mechanisms,

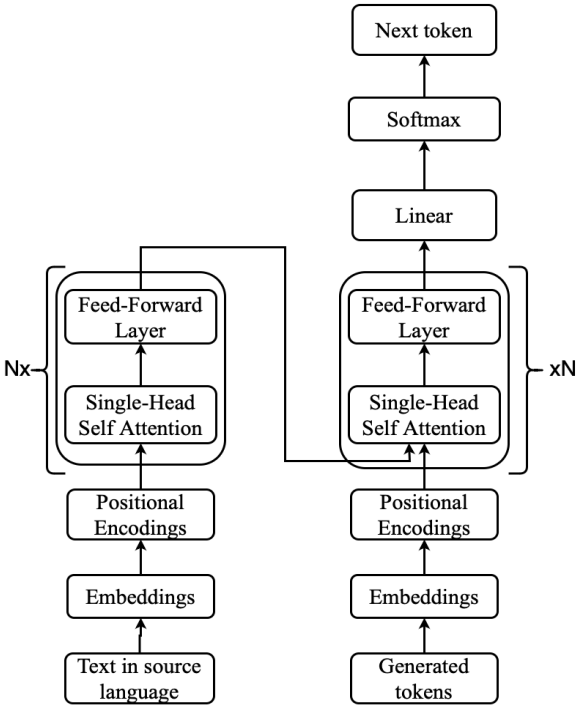


Figure 3.2 Reduced architecture of the Transformer model.

the encoder captures dependencies and interrelations between tokens within the input sequence, creating a comprehensive contextualized representation. This robust encoding of the input sequence is pivotal for subsequent processing in various tasks, as it enables the model to approximate and comprehend semantic nuances and contextual information necessary for accurate processing and generation.

The Transformer model operates on input text by sequentially processing each token. Initially, the encoder segments the input text into individual tokens and encodes them into a higher-dimensional space using embedding and positional encoding layers by

$$u_s = \Phi_{enc}(x_s^{\text{source}}) = \text{Embed}(x_s^{\text{source}}) + \text{PE}(x_s^{\text{source}}), \tag{3.19}$$

where Embed is the embedding layer and PE is positional encoding layer, described by equation 2.18. Therefore, after these steps we have a representation $u_s \in R^{l \times d}$ of our input text in a higher-dimensional space.

Next we have N stacks of self-attention mechanism and feed-forward layers, where output from one layer serves as input to the other layer. The final result of the encoder block is represented by

$$u_s^{\text{enc}} = f_{enc} \overbrace{\odot \ldots \odot}^{Ntimes} f_{enc}(u_s), \tag{3.20}$$

where $u_s = \Phi_{enc}(x_s^{\text{source}})$ and f_{enc} is described by equations 2.20-2.25. As a result we have a hidden state $u_s^{\text{enc}} \in R^{l \times d}$ of the input text which aids the decoder in comprehending the nuanced meaning and contextual details necessary for producing accurate outputs during each iteration.

In the subsequent phase, the decoder commences processing the previously generated text. Similar to the encoder, the initial step involves mapping the input text to a higher-dimensional vector space using embedding and positional encoding layers by

$$z_s = \Phi_{dec}(x_s^{\text{target}}) = \text{Embed}(x_s^{\text{target}}) + \text{PE}(x_s^{\text{target}}), \tag{3.21}$$

where $x_s^{\text{target}} = (t_1, ..., t_k)$ is the sequence of already generated k preceding tokens, *Embed* is the embedding layer and *PE* is positional encoding layer, described by equation 2.18. Therefore, after this step we have a sequence of already processed tokens embedded into a higher-dimensional space $z_s \in R^{k\times d}$.

In the following phase of the decoder block, we employ a single-head self-attention mechanism. However, it differs in that it receives two inputs: the encoded representation of the input text, denoted as u_s^{enc}, and the encoded representation of the already generated text, denoted as z_s. In this configuration, u_s^{enc} serves as the keys and values, whereas z_s serves as the queries. Consequently, the self-attention mechanism within the decoder processes inputs by

$$z_{SHA} = \text{softmax}(z_s W_Q W_K^T u_s^{\text{enc}^T}) u_s^{\text{enc}} W_V, \tag{3.22}$$

where $z_s \in R^{k\times d}$ is state of our system in a higher-dimensional space, $u_s^{\text{enc}} \in R^{l\times d}$ is additional parameter in our system at time step s and $W_Q, W_K, W_V \in R^{d\times d}$ are learnable weight matrices. Again, it's straightforward to confirm that $z_{SHA} \in R^{k\times d}$

$$\underset{k\times d}{\mathrm{z_s}}\ \underset{d\times d}{\mathrm{W_Q}} = \underset{k\times d}{\mathrm{Q}}, \tag{3.23}$$

$$\underset{d\times d}{\mathrm{W_K^T}}\ \underset{d\times l}{\mathrm{u_s^{enc^T}}} = \underset{d\times l}{\mathrm{K^T}}, \tag{3.24}$$

$$\underset{k\times d}{\mathrm{Q}}\ \underset{d\times l}{\mathrm{K^T}} = \underset{k\times l}{\mathrm{QK^T}}, \tag{3.25}$$

$$\underset{l\times d}{\mathrm{u_s^{enc}}}\ \underset{d\times d}{\mathrm{W_V}} = \underset{l\times d}{\mathrm{V}}, \tag{3.26}$$

$$\underset{k\times l}{\mathrm{softmax(QK^T)}}\ \underset{l\times d}{\mathrm{V}} = \underset{k\times d}{\mathrm{z_{SHA}}}. \tag{3.27}$$

We employ the encoder block's output as keys and values in the self-attention layer of the decoder block for several pivotal reasons. Initially, the encoder block captures contextual information from the input text, which is crucial for comprehending the subtleties within the input sequence. By utilizing these outputs as keys and values, the decoder can effectively focus on pertinent segments of the input sequence, facilitating precise predictions. Additionally, leveraging the encoder's outputs facilitates information exchange between the encoder and decoder, fostering coherence and alignment between the input and target sequences. In essence, integrating the encoder's output as keys and values enhances the decoder's capacity to generate coherent and contextually precise predictions.

As in the previous part, we can see the self-attention layer as the lifting function for z_s, but now it is also parameterized by the output of the encoder block u_s. Hence, in our case lifting function is described by

$$\Phi_1(z_s, u_s^{\text{enc}}) = \text{softmax}(z_s W_Q W_K^T u_s^{\text{enc}^T}) u_s^{\text{enc}} W_V. \tag{3.28}$$

In the next stage feed-forward layer processes output from the self-attention mechanism

$$z_{FFL} = max(0, z_{SHA} W_1) W_2 + b_2, \tag{3.29}$$

where $z_{SHA} \in R^{k\times d}$ and $W_1 \in R^{d\times d_f}$ and $W_2 \in R^{d_f\times d}$ are learnable weight matrices.

As in the previous subsection, here we can again view z_{SHA} as the initial state of our system in the dictionary space lifted by $\Phi_1(z_s, u_s^{\text{enc}})$ and rewrite it as z_{SHA}^0. From the point of view of Koopman Operator theory, the dot product of z_{SHA}^0 and W_1 is the prediction of the future state of the system in the dictionary state space. This evolution of the system is described by

$$z_{SHA}^0 W_1 = z_{SHA}^1. \tag{3.30}$$

Once again, we must pay attention that the result of the dot product of z_{SHA}^0 and W_1 will be in $R^{k \times d_f}$

$$\underset{k \times d}{\mathrm{z}_{\mathrm{SHA}}^0} \underset{d \times d_f}{\mathrm{W}_1} = \underset{k \times d_f}{\mathrm{z}_{\mathrm{SHA}}^1}. \tag{3.31}$$

And to stay within the frames of Koopman Operator Theory we must have z_{SHA}^0 and z_{SHA}^1 in the same dictionary space. Thus, we need W_1 to be in $R^{d \times d}$.

We define the rest of the equation 3.29

$$\Psi_1 = max(0, z_{SHA}^1) W_2 + b_2 \tag{3.32}$$

as the inverse function of $\Phi_1(z_s, u_s^{\text{enc}})$, mapping z_{SHA}^1 back to the original state space $z_s \in R^{k \times d}$. Therefore, in the realm of Koopman Operator theory, we encounter a following dynamical system within the decoder block represented by

$$\begin{aligned} z_s^{i+1} &= f_{dec}(z_s^i, u_s^{\text{enc}}), && (3.33)\\ \Phi_1(z_s^i, u_s^{\text{enc}}) W_1 &= \Phi_1(f_{dec}(z_s^i, u_s^{\text{enc}})) = \Phi_1(z_s^{i+1}, u_s^{\text{enc}}), && (3.34)\\ z_s^{i+1} &= \Psi_1(\Phi_1(z_s^i, u_s^{\text{enc}}) W_1), && (3.35) \end{aligned}$$

where $z_s^0 = \Phi_{dec}(x_s^{\text{target}})$.

With N stacks of decoder block we have

$$z_s^N = f_{dec} \overbrace{\odot \ldots \odot}^{Ntimes} f_{dec}(z_s^0, u_s^{\text{enc}}), \tag{3.36}$$

where $z_s^0 = \Phi_{dec}(x_s^{\text{target}})$. Evolution inside the decoder block through parametric Koopman Operator theory is represented in Algorithm 5.

Algorithm 5: Evolution inside decoder block through parametric Koopman Operator

$u_s^{\text{enc}} \leftarrow \text{Encoder}(x_s^{\text{source}})$;
$i \leftarrow 0$;
$z_s^i \leftarrow \Phi_{dec}(x_s^{\text{target}})$;
while $i < N$ **do**
  $z_{SHA}^i \leftarrow \Phi_1(z_s^i, u_s^{\text{enc}})$;
  $z_{SHA}^{i+1} \leftarrow z_{SHA}^i W_1$;
  $z_s^{i+1} \leftarrow \Psi_1(z_{SHA}^{i+1})$;
  $i \leftarrow i + 1$
end

After completing N iterations of decoding, we again convert our state z_s^N back to the original state space by employing affine and softmax layers to obtain next token prediction for the target sequence from the model. This process is expressed by

$$z_s^{\text{linear}} = \text{Linear}(z_s^N) = z_s^N W_{linear} + b_{linear}, \tag{3.37}$$

where $W_{\text{linear}} \in R^{d \times v}$ and v is the size of vocabulary of the target sequence, and

$$(z_s^{\text{softmax}})_{ij} = \frac{e^{(z_s^{\text{linear}})_{ij}}}{\sum_{j=1}^{v} e^{(z_s^{\text{linear}})_{ij}}}, \tag{3.38}$$

where $z_s^{\text{linear}} \in R^{k \times v}$ is the matrix obtained after the final affine block.

After that we generate next token in the target sequence

$$(x_{s+1}^{\text{target}})_i = \underset{j}{\text{argmax}}((z_s^{\text{softmax}})_{ij}), \tag{3.39}$$

where $z_s^{softmax} \in R^{k \times v}$ is the matrix obtained after the softmax layer.

Hence, equations from 3.37 to 3.39 can be viewed as the inverse function Ψ_{dec} of Φ_{dec}.

We are again dealing with two dynamical systems:

1. The external dynamical system operates within the realm of token sequences, denoted as $x_s^{\text{target}} = (t_1, ..., t_k)$ with lifting function to a higher dimensional space Φ_{dec}.

2. The internal dynamic system operates within the decoder block, functioning in the space of token embeddings $z_s \in R^{k \times d}$ and is elevated by a lifting function Φ_1.

After N steps of the internal dynamic function, we return to the space governed by the external dynamic function to obtain next token prediction for the target sequence. The complete procedure for our simplified Transformer architecture is represented in Algorithm 6.

Algorithm 6: Evolution of the encoder-decoder Transformer architecture through Koopman Operator theory

Assume we have an input sequence x and wish to generate an output sequence y.
$x \leftarrow (t_0^{\text{source}}, ..., t_l^{\text{source}})$ - Sequence of tokens representing the input data;
$t_0^{\text{target}} \leftarrow [BOS]$ - Beginning of sequence token, initializing the state of the outer dynamical system at time step 0;
while $t_s \neq [EOS]$ **do**
  $u \leftarrow \Phi_{enc}(x)$ - Representation of input tokens in a higher-dimensional space;
  $u_{enc} \leftarrow f_{enc} \overbrace{\odot ... \odot}^{N \text{ times}} f_{enc}(u)$ - Parameter for the internal dynamical system;
  $y \leftarrow (t_0^{\text{target}}, ..., t_{s-1}^{\text{target}})$ - State of the dynamical system at time step s, defined by the sequence of already generated tokens;
  $z^0 \leftarrow \Phi_{dec}(y)$ - Initial state of the internal dynamical system;
  while $i < N$ **do**
    $z^i SHA \leftarrow \Phi_1(z^i, u_{enc})$;
    $z^{i+1} SHA \leftarrow z^i SHA W_1$;
    $z^{i+1} \leftarrow \Psi_1(z^{i+1} SHA)$.
  end
  $t^s \leftarrow \Psi_{dec}(z^N)$.
end

3.3 Experiments and Results

This section presents the results of the experiment validating the connection between operations inside Transformer block and Koopman operator theory. It starts by outlining the dataset employed in this experiment, underscoring its significance and unique attributes. Following that, it clarifies the quantitative measures and qualitative validation techniques utilized to check the correctness of the proposed method. Afterward, it presents the experimental setup and procedure, providing a detailed account of the steps taken to evaluate the method. Finally, it presents the outcomes of this experiment and an analysis of the results. The section concludes with a discussion of the limitations of the method.

3.3.1 Dataset

The Anki English-French dataset is a comprehensive collection of bilingual sentence pairs designed to facilitate language learning and translation tasks between English and French [28]. Compiled from various sources, this dataset encompasses a diverse range of linguistic contexts, covering everyday phrases, idiomatic expressions, and specialized vocabulary. With a total of 232,736 sentence pairs, the dataset provides ample data for training and evaluating machine translation models, as well as for linguistic analysis and research purposes. In our experiments we will use a modified version of this dataset, which consists of 175,621 sentence pairs [19].

Data Composition and Characteristics

The Anki English-French dataset comprises a rich collection of bilingual sentence pairs, meticulously curated to facilitate language learning and translation tasks between English and French. Each sentence pair encapsulates a diverse range of linguistic contexts, covering various topics, styles, and grammatical structures.

Characteristics:

1. **Bilingual Content.** The dataset consists of parallel sentences in English and French, fostering bidirectional translation tasks between these two languages. This bilingual nature enables learners and researchers to explore language dynamics and nuances across English and French.

2. **Sentence Variability.** Spanning a wide spectrum of sentence lengths and complexities, the dataset encapsulates both simple, everyday phrases and more intricate linguistic constructs. This variability challenges translation models to handle diverse linguistic patterns and capture nuanced meanings effectively.

3. **Real-World Relevance.** The sentence pairs in the dataset are derived from real-world language usage, ensuring authenticity and relevance to practical communication scenarios. As a result, learners and researchers can glean insights into natural language usage and cultural nuances in both English and French.

4. **Quality Assurance.** The dataset undergoes rigorous quality assurance measures to ensure accuracy and coherence in the sentence pairs. Each sentence pair is carefully reviewed and curated to maintain linguistic fidelity and alignment between English and French counterparts.

5. **Data Diversity.** Covering a broad spectrum of topics, including everyday conversations, academic discourse, and specialized domains, the dataset encapsulates diverse linguistic contexts. This diversity enriches the learning and research experience, catering to a wide range of language learners and application domains.

Usage and Impact

The Anki English-French dataset has been utilized in various research projects, particularly in the field of machine translation and language learning technologies. One notable example is a project on GitHub, where the dataset was used for training a deep learning model for language translation [34]. This project aimed to translate English to French, leveraging the Anki dataset to enhance the model's understanding and translation capabilities between the two languages.

Additionally, the Anki English-French dataset has been instrumental in the development of educational tools and applications designed to aid in the learning process. For instance, the French B2 Vocabulary Anki Deck, which is part of the broader Anki dataset, has been used to create a personalized practice experience for learners. This deck contains 3172 flashcards across various chapters, including pronouns, conjunctions, prepositions, and determiners, providing a comprehensive resource for learning French B2 vocabulary. The deck's structure, which includes images, IPA notation, and audio files, has been designed to enhance memory retention and pronunciation accuracy, making it an effective tool for memorizing and understanding French vocabulary in context [10].

These examples highlight the Anki English-French dataset's significance in both academic research and practical applications, showcasing its impact on the field of language learning and technology.

Data Exploration

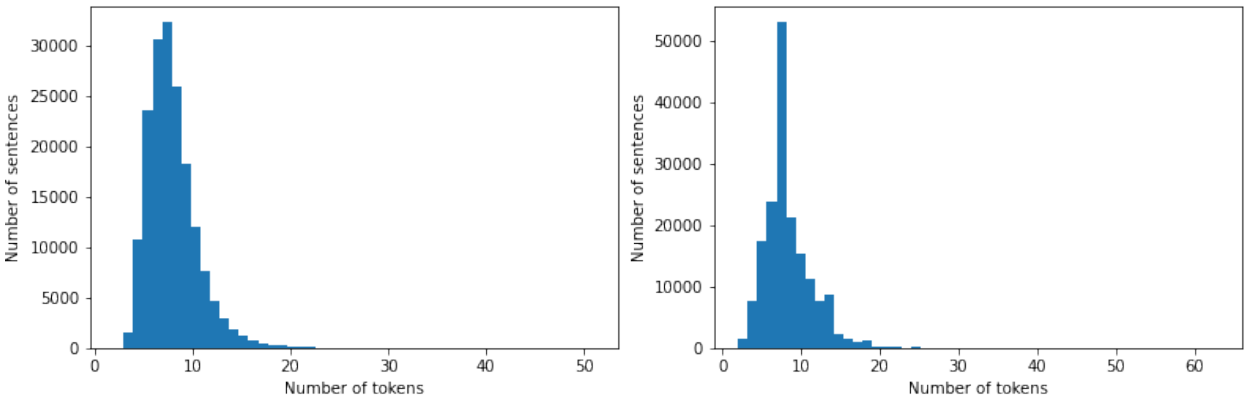
The Anki English-French dataset comprises two columns: the first column contains text in English, and the second column contains the corresponding French translations. In total, the dataset consists of 175,621 pairs of bilingual sentence pairs. Table 3.1 shows a ten randomly sampled bilingual sentence pairs. To explore the statistics of the text, we tokenized the sentences using the spacy package, which provides pre-trained word embeddings for various languages, including English and French.

English text	French text
I wonder what all of them have in common.	Je me demande ce qu'ils ont tous en commun.
There are only three options.	Il n'y a que trois options.
You take the money.	Prends l'argent.
The soccer game will be played, even if it rains.	Le match de foot sera disputé, même s'il doit pleuvoir.
Are you sure that you want to do this?	Êtes-vous sûr de vouloir faire cela?
Can anyone here speak French?	Quelqu'un ici sait-il parler français?
I was ten minutes late for school.	Je suis arrivé 10 minutes en retard à l'école.
He will end up in jail.	Il finira en prison.
Don't you ever get tired?	Tu ne te fatigues jamais?
Why are you so cheerful?	Pourquoi es-tu si joyeux?

Table 3.1 Example of 10 randomly chosen bilingual sentence pairs

Figure 3.3a illustrates the distribution of tokens for English texts in the dataset. The mean number of tokens for English sentences is 7.6, with a standard deviation of 2.66, indicating that the texts are relatively short yet varied. The distribution ranges from a minimum of 2 tokens to a maximum of 51 tokens, showcasing the diversity in sentence lengths. Table 3.2a provides detailed statistics on token counts, including minimum, maximum, median, and quartile values, offering a comprehensive overview of the dataset's token distribution. Additionally, Figure 3.4a showcases the top 20 most frequent tokens in the English sentences, shedding light on the vocabulary composition and prevalent linguistic patterns within the dataset.

Figure 3.3b demonstrates the distribution of tokens within French texts. The average number of tokens per French sentence is 8.4, with a standard deviation of 3.12, suggesting that French translations are gen-



(a) Token distribution for English texts.

(b) Token distribution for French texts.

Figure 3.3 Token frequencies for English and French sentences.

(a) Top 20 most frequent tokens for English sentences.

(b) Top 20 most frequent tokens for French sentences.

Figure 3.4 Top 20 most frequent tokens for English and French sentences.

	Value
Min	2.0
25th percentile	6.0
Mean	7.6
Median	7.0
75th percentile	9.0
Max	51.0
Std	2.66

(a) Token statistics for English texts.

	Value
Min	2.0
25th percentile	6.0
Mean	8.4
Median	8.0
75th percentile	10.0
Max	63.0
Std	3.12

(b) Token statistics for French texts.

Table 3.2 Detailed token statistics for English and French sentences.

erally concise yet varied in length, ranging from a minimum of 2 tokens to a maximum of 63 tokens per sentence. Table 3.2b provides detailed statistics on token counts, including minimum, maximum, median, and quartile values, offering a comprehensive overview of the dataset's token distribution in French texts. Additionally, Figure 3.4b presents top 20 most popular tokens in the French sentences, providing insights into the prevalent linguistic patterns and vocabulary composition within the dataset.

Data Preparation

For data preparation in model training and evaluation, we initially refrained from conducting any data cleaning procedures as the dataset was already curated and prepared. With a focus on training and evaluating the model effectively, we partitioned the dataset into two subsets: 90% of the data was allocated for model training, while the remaining 10% was reserved for model evaluation. This division ensures that the model learns from a sufficiently large portion of the data while also allowing for robust evaluation of its performance on unseen data.

3.3.2 Model Architecture

Figure 3.2 illustrates the comprehensive architecture of our proposed model. Our custom Transformer model begins its journey with an Embedding layer, responsible for generating 192-dimensional token embeddings for both the source and target languages. These embeddings serve as rich representations of individual tokens, crucial for subsequent processing.

Following the embedding stage, a positional encoding layer injects positional information into the embeddings, facilitating the model's understanding of token positions within sequences. This step is particularly vital in preserving the sequential order of input tokens, which is essential for capturing contextual relationships effectively.

Moving forward, the model features an encoder block comprising three layers, each equipped with six heads in its self-attention mechanism. This design allows each token in the sequence to attend to multiple positions, capturing diverse contextual dependencies. The multi-headed attention mechanism enables the model to efficiently process and encode information from various perspectives, enhancing its ability to capture intricate patterns within the data. Subsequently, the output of the self-attention mechanism traverses through fully connected feed-forward layers. The result of the encoder block serves as keys and values for the self-attention mechanism in the decoder block.

Transitioning to the decoder block, a similar architecture is employed, albeit with a single head in its self-attention mechanism. This adjustment ensures that the decoder can selectively focus on relevant por-

tions of the input sequence while generating output, a crucial aspect in sequence-to-sequence tasks such as translation. The decoder block utilizes a masked self-attention layer, allowing tokens in the output sequence to attend only to previous positions, preventing information leakage from future tokens during training. Furthermore, the decoder layers share weights and biases across all layers, a design choice motivated by the need to align the dynamics within the decoder block with Koopman Operator theory. In our experiment, the self-attention mechanism viewed as the dictionary function, capturing the system's state evolution, while the first matrix in the feed-forward layer acts as the Koopman operator matrix, encoding the system's transition dynamics and the rest of feed-forward layer considered as inverse of the dictionary function. By sharing weights and biases across layers, we ensure consistency in these transformation functions throughout the decoder block, facilitating the application of Koopman Operator theory to analyze the model's behavior. In a departure from traditional transformer architectures, the first self-attention block in the decoder block is omitted, streamlining the model's processing pipeline. Furthermore, normalization layers are excluded from the decoder, aligning with the model's overarching goal of simplicity and efficiency.

Throughout both the encoder and decoder blocks, the attention and feed-forward layers maintain a consistent dimensionality of 192×192, ensuring uniformity in the model's representation and transformation capacities. This coherence facilitates seamless information flow and transformation across different layers, enhancing the model's overall performance and robustness.

In summary, our custom Transformer model embodies a carefully crafted architecture tailored to the specific requirements of description operations inside decoder block through parametric Koopman Operator theory. By incorporating strategic design choices and parameter configurations, the model strikes a balance between complexity and effectiveness, offering a potent framework for various natural language processing applications.

3.3.3 Evaluation Methods

In our study, our primary objective is to examine whether operations inside the decoder block of our custom Transformer model align with the Koopman Operator theory. Specifically, we aim to determine if the self-attention mechanism, as depicted in equation 3.28, operates as a dictionary function, with the subsequent multi-layer perceptron from equation 3.29, excluding the first weight matrix, serving as its inverse. Additionally, we seek to explore the role of the first matrix in the feed-forward layer as a potential Koopman Operator matrix, facilitating transition to the next state within the dictionary space.

To conduct our investigation, we utilize the Anki English-French dataset for training and evaluation, allocating 10% of the dataset for hypothesis testing. Our experimental design begins with training an end-to-end implementation of our custom Transformer model on the Anki English-French dataset, focusing specifically on the task of machine translation. Our primary evaluation metric will be the cross-entropy loss for the translated tokens.

Cross-Entropy Loss

Cross-entropy loss measures the dissimilarity between the predicted probability distribution and the actual distribution of the target labels. Mathematically, it is defined as

$$\text{CrossEntropyLoss} = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(p_i), \tag{3.40}$$

where N is the number of samples, y_i is the true label, and p_i is the predicted probability of the corresponding class. Lower cross-entropy loss indicates better model performance in classification tasks.

Subsequently, we plan to evaluate the functionality of the self-attention mechanism and the feed-forward layer as dictionary and inverse functions, along with the first matrix of the feed-forward layer being a Koopman Operator matrix by mean squared error.

Mean Squared Error (MSE) Loss

Mean squared error is a measure of the average squared difference between the predicted values and the actual values. In the context of our investigation, we utilize MSE to quantify the discrepancy between embeddings before and after specific transformations within the decoder block. Mathematically, it is expressed as

$$\text{MSE}(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{3.41}$$

where y represents the true values, $\hat{y}$ denotes the predicted values, and n is the number of samples. A lower MSE indicates that the predicted values are closer to the true values, signifying better model performance.

To assess whether the feed-forward layer acts as an inverse to the self-attention mechanism, during training and evaluation stages we will capture the embeddings prior to the self-attention mechanism application. Following this, we will apply both the self-attention mechanism and the feed-forward layer (excluding the first matrix) to these embeddings. We will then compute the mean squared error between the initial embeddings and the embeddings after the application of both mechanisms, evaluating if they are sufficiently close. In other words, for each layer of the decoder block, we will compute the following mean squared error

$$MSE(z^i, \Psi_1(\Phi_1(z^i, u^{enc}))), \tag{3.42}$$

where z^i is the embedding matrix which acts as the input to the ith layer of the decoder block, u^{enc} is the result of the encoder block, Φ_1 is the dictionary function from equation 3.28, and Ψ_1 is the inverse function from equation 3.32. In the case if Ψ_1 is the inverse function of $\Phi_1(z^i, u^{enc})$, this mean squared error should be close to zero.

In order to examine the function of the initial matrix in the feed-forward layer as a Koopman Operator, we will first capture the embeddings after the self-attention mechanism and the W_1 matrix have been applied during both the training and evaluation phases. Additionally, we will record embeddings solely after the self-attention mechanism has been applied. Subsequently, we will calculate the mean squared error between the first embeddings and the second embeddings of the following layer to assess the effectiveness of the first matrix in the feed-forward layer in transitioning to the next state within the dictionary space. This process is defined by

$$MSE(\Phi_1(z^i, u^{enc})W_1, \Phi_1(z^{i+1}, u^{enc})), \tag{3.43}$$

where z^i is the embedding matrix which acts as the input to the ith layer of the decoder block, u^{enc} is the result of the encoder block, and Φ_1 is the dictionary function from equation 3.28.

BLEU Score

Additionaly we will track BLEU score of our predicted translations. BLEU (Bilingual Evaluation Understudy) score is a metric commonly used to evaluate the quality of machine-translated text by comparing it to one or more reference translations [35]. It computes the precision of n-grams (contiguous sequences of n items, such as words or tokens) in the candidate translation compared to the reference translations. Mathematically, BLEU score is represented by

$$\text{BLEU} = \text{BP} \times \exp\left(\sum_{n=1}^{N}\frac{1}{N}\log p_n\right), \tag{3.44}$$

where BP is the brevity penalty to account for short translations, N is the maximum order of n-grams considered, and p_n is the precision of n-grams. Higher BLEU scores indicate better agreement between the candidate and reference translations.

In the case that our initial experiment indicates the inefficacy of the Koopman framework within the decoder block, we propose to modify our approach. Specifically, we will augment our loss function with the two mean squared errors discussed above. This adjustment aims to compel the feed-forward layer to effectively serve as an inverse to the self-attention mechanism and the first matrix in the feed-forward layer as a Koopman Operator. Additionally, it will be intriguing to compare the BLEU scores of predictions from both the first and second experiments to ensure that the modifications do not result in a performance drop in translations during the second experiment.

Through these investigations, we aim to deepen our understanding of the Transformer model architecture and its alignment with the theoretical underpinnings of the Koopman framework.

3.3.4 Results and Discussion

First experiment. Basic training of Transformer model.

Here we detail the initial experiment conducted to investigate the alignment of the operations within the decoder block of our custom Transformer model with the theoretical framework of the Koopman Operator theory. The primary objective was to explore whether the self-attention mechanism and the feed-forward layer functioned as dictionary and inverse functions, respectively, and if the first matrix in the feed-forward layer acted as a Koopman Operator matrix.

The experiment utilized the Anki English-French dataset for training and evaluation, with 10% of the dataset reserved for hypothesis testing. An end-to-end implementation of our custom Transformer model was trained on this dataset, focusing specifically on the task of machine translation. Training was performed for 500 epochs with batch size 192 on a Tesla V100 32 GB GPU, with each epoch taking approximately 115 seconds.

The experiment commenced with a notably higher cross-entropy loss on the training set, approximately around 6, gradually diminishing to 0.68 as the training progressed. Similarly, the cross-entropy loss on the validation set started at around 4.5 and decreased to approximately 1.43 by the end of the training period. Despite this reduction, the validation loss remained comparatively higher, suggesting potential room for further optimization. Figure 3.5 illustrates the dynamic evolution of the cross-entropy loss for both the training and validation datasets over the course of training. This trend highlights the progressive convergence of the model's predictions towards the ground truth labels, albeit with some fluctuations and residual error on the validation set.

We reject the hypothesis that the feed-forward layer serves as the inverse function for the self-attention mechanism in the decoder block, as the mean squared error (MSE) between the initial embedding and its inverted computation consistently fell within the range between 200 to 400 across both the training and validation sets. Figure 3.6 presents the fluctuating changes in the mean squared error loss for both the training and validation datasets throughout the training process. This MSE observation, averaging around 300 during both training and validation stages, signifies a notable departure from the anticipated results according to the Koopman Operator theory. Such findings underscore a misalignment between the operations within the decoder block of the Transformer model and the theoretical framework originally hypothesized.

We also dismiss the hypothesis regarding the first matrix of the feed-forward layer functioning as the Koopman Operator matrix, as the mean squared error (MSE) between the multiplication of this matrix with current state in the dictionary space and the next state in the dictionary space consistently ranged

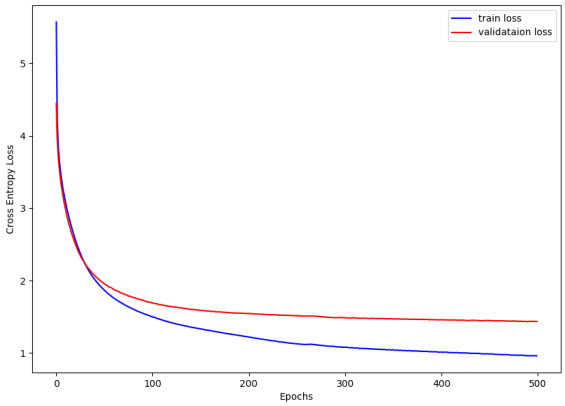


Figure 3.5 Dynamics of the Cross-Entropy loss

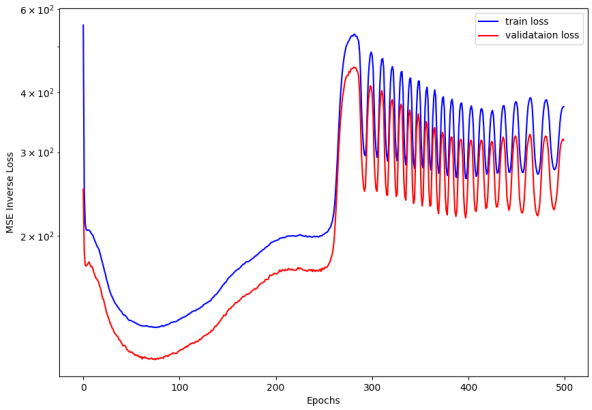


Figure 3.6 Dynamics of the Mean Squared Error loss for the inverse function.

between 100 and 250 across both the training and validation sets. Figure 3.7 depicts the dynamics of the mean squared error loss concerning both the training and validation datasets throughout the training process. This MSE observation suggests a departure from the expected outcomes based on the Koopman Operator theory. Therefore, it appears that the first matrix in the feed-forward layer does not effectively facilitate the transition to the next state within the dictionary space, as initially proposed.

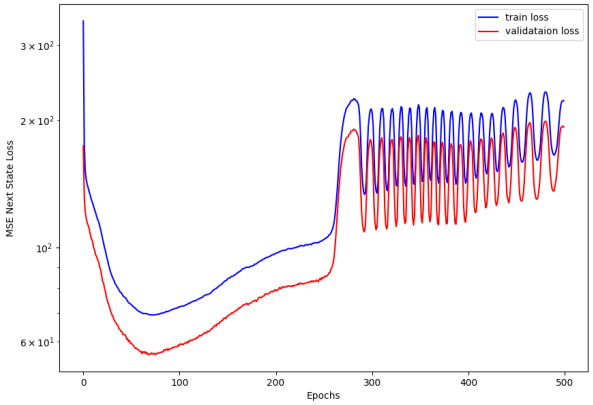


Figure 3.7 Dynamics of the Mean Squared Error loss for the next state prediction in the dictionary space.

Table 3.3 presents detailed metrics for train and validation data at the end of the experiment.

Dataset	CE Loss	MSE Inverse	MSE Next State	BLEU
Train Dataset	0.68	318.52	192.08	24.3
Valid Dataset	1.43	318.02	192.29	18.07

Table 3.3 Metrics for train and validation data at the end of the training.

The initial experiment provided valuable insights into the behavior of the Transformer model's decoder block in relation to the Koopman Operator theory. Despite the observed misalignment between theoretical expectations and empirical results, the experiment lays the groundwork for future investigations aimed at refining the training process of the Transformer model. Through continued experimentation and iterative refinement, we aim to bridge the gap between theory and practice, advancing our understanding of Transformer model through point of view of dynamical systems.

Second experiment. Adjusted training of Transformer model.

In this experiment, we aimed to refine the operations within the decoder block of our custom Transformer model to align with the theoretical framework of the Koopman Operator theory. Specifically, we enforced the feed-forward layer to act as the inverse function of the self-attention mechanism and the first weight matrix in the feed-forward layer to facilitate transition to the next state in the dictionary space. This was achieved by designing a complex loss function comprising cross-entropy loss and two mean squared errors, which optimized the desired behavior during backpropagation.

The training process resulted in significant improvements. The cross-entropy loss for the training data decreased from approximately 7.0 to 0.68 after 500 epochs. Similarly, the validation cross-entropy loss decreased from around 5 to nearly 1.4 over the training period. Figure 3.8 depicts the dynamics of the cross-entropy loss for both the training and validation datasets throughout the training stage. This pat-

tern underscores the gradual alignment of the model's predictions with the ground truth labels, although occasional fluctuations and residual errors were also observed, particularly on the validation set. This observation was consistent with our findings in the first experiment.

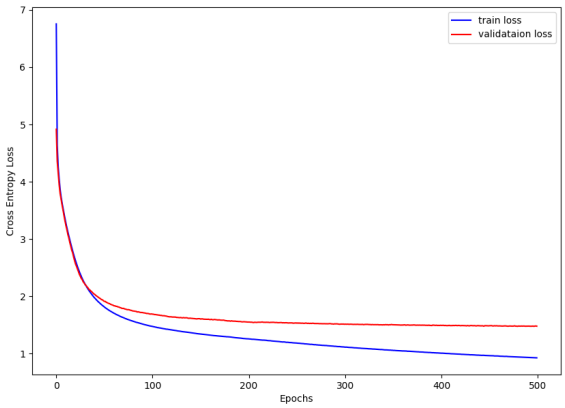


Figure 3.8 Dynamics of the mean Cross-Entropy loss per epoch.

Remarkably, there was a substantial reduction in the mean squared error associated with the inverse function, decreasing significantly from above 20 to 0.012 for both the training and validation datasets, and this reduction persisted consistently throughout the training process. The initial sharp decline in the mean squared loss of the inverse function can be attributed to its initial disparity compared to the cross-entropy loss. Consequently, during backpropagation, the optimizer prioritized optimizing the mean squared error over the cross-entropy loss. The variation in the mean squared error for the inverse function across the training and validation datasets is depicted in Figure 3.9, providing a visual representation of its dynamic changes during training. This outcome underscores the feed-forward layer's effectiveness in accurately mapping the system state from the dictionary space back to the original state space.

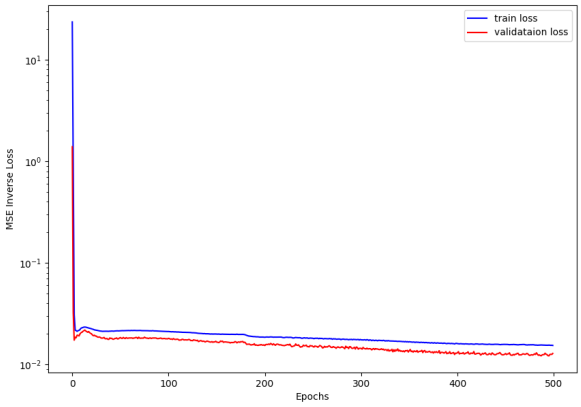


Figure 3.9 Dynamics of the mean MSE loss per epoch for the inverse function.

Moreover, there was a significant decrease in the mean squared error associated with the next state transition in the dictionary space, plummeting from around 15 to 0.012 for both the training and validation datasets, and this reduction remained consistent throughout the training phase. The fluctuation in the mean squared error for the next state transition in the dictionary space across the training and validation datasets is illustrated in Figure 3.10, offering a visual representation of its dynamic changes during train-

ing. This outcome emphasizes the effectiveness of the weight matrix in facilitating accurate transitions between states within the dictionary space.

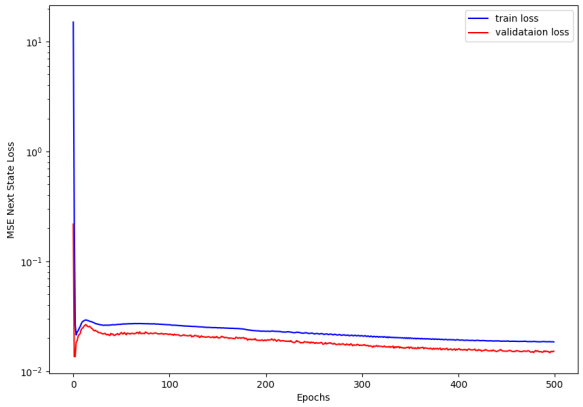


Figure 3.10 Dynamics of the mean MSE loss per epoch for the next state prediction in the dictionary space.

The detailed metrics for train and validation data are presented in Table 3.4. Additionally, it is noteworthy that with the adjusted training approach, we also achieved slightly higher BLEU scores for translations in the second experiment.

Dataset	CE Loss	MSE Inverse	MSE Next State	BLEU
Train Dataset	0.68	0.0127	0.01528	24.55
Valid Dataset	1.48	0.0128	0.01526	18.21

Table 3.4 Metrics for train and validation data at the end of the training.

Overall, these results demonstrate the successful training of our custom Transformer model with the feed-forward layer acting as the inverse function of the self-attention mechanism and the first weight matrix serving as a transition to the next state in the dictionary space. Importantly, these modifications were achieved while maintaining the cross-entropy loss at a level comparable to that of the initial experiment. Furthermore, it is noteworthy that with the adjusted training approach, we also observed elevated BLEU scores for translations in the second experiment. This success paves the way for further exploration and refinement, enhancing our understanding of Transformer model dynamics through the lens of the Koopman Operator theory.

Limitations and Future Work

The experiments conducted to investigate the alignment of the operations within the decoder block of our custom Transformer model with the Koopman Operator theory have provided valuable insights into the model's behavior. However, several limitations and areas for future research have been identified, which warrant further exploration and refinement.

Limitations:

1. **Simplified Model Dynamics.** The experiments were conducted using a simplified representation of the Transformer model architecture. While this approach allowed for a targeted analysis, it may not fully capture the complex interactions and dynamics present in the entire model. Future re-

search could explore the integration of the proposed modifications within the entire Transformer architecture to assess their impact comprehensively.

2. **Assumption of Linearity.** The Koopman Operator theory relies on the assumption of linearity, which may not hold true for all aspects of the Transformer model's operations. The observed deviations from theoretical expectations in both experiments suggest potential nonlinearities or complexities that need to be accounted for in future investigations.

3. **Limited Dataset and Task.** The experiments were conducted using the Anki English-French dataset for the task of machine translation. While this dataset served as a suitable starting point, its size and complexity may have constrained the generalizability of the findings. Future research could explore the applicability of the proposed modifications across different datasets and tasks to assess their robustness and effectiveness under varying conditions.

4. **Lack of Exploration in Hyperparameter Space.** The experiments were conducted with a fixed set of hyperparameters, limiting the exploration of the model's performance across different configurations. Future research could explore a wider range of hyperparameter settings to understand their impact on the effectiveness of the proposed modifications and to identify optimal configurations for improved performance.

5. **Limited Analysis of Model Robustness.** The experiments focused primarily on training dynamics and performance metrics, with limited analysis of the model's robustness to variations in input data or perturbations in model parameters. Future research could investigate the robustness of the modified Transformer model through techniques such as adversarial testing or input perturbation analysis to assess its resilience to unforeseen challenges.

Future Work:

1. **Model Refinement.** Building upon the insights gained from the experiments, future work could focus on refining the proposed modifications to enhance their effectiveness and applicability. This could involve exploring alternative loss functions, regularization techniques, or architectural adjustments to better align the model's behavior with the theoretical expectations of the Koopman Operator theory.

2. **Evaluation on Real-World Tasks.** Extending the evaluation of the model to real-world tasks and applications could provide further insights into its practical utility and effectiveness. Future research could explore the performance of the modified Transformer model on a diverse range of tasks, such as natural language understanding, generation, or multimodal tasks, to assess its versatility and generalization capabilities.

3. **Theoretical Investigations.** Delving deeper into the theoretical underpinnings of the observed model behaviors could provide valuable insights into the relationship between neural network architectures and dynamical systems theory. Future research could explore theoretical frameworks and methodologies to formalize and analyze these relationships, advancing our understanding of complex neural architectures like the Transformer model.

In conclusion, while the experiments have shed light on the alignment of the Transformer model's operations with the Koopman Operator theory, there remain several avenues for future research to explore and refine these findings. Addressing the identified limitations and pursuing the suggested future work could contribute to the development of more effective and theoretically grounded neural network architectures.

4 Conclusion

Throughout this thesis, we have embarked on a journey to elucidate the operations of Transformer blocks from the perspective of Koopman Operator theory, aiming to provide a deeper understanding of their dynamics. The advent of the Transformer architecture has reshaped the landscape of natural language processing, revolutionizing various tasks such as machine translation, sentiment analysis, and text summarization. Despite their remarkable performance, the inner workings of Transformer models, particularly their self-attention mechanisms, have often been regarded as opaque due to their complex nature.

Motivated by the desire to demystify these operations, we turned to Koopman Operator theory, a powerful mathematical framework for analyzing the evolution of nonlinear dynamical systems. By casting the operations of Transformer blocks in the language of Koopman Operators, we sought to uncover underlying structures and dynamics that govern the processing of sequential data.

In our exploration, we first delved into the operations of the decoder Transformer block, treating the self-attention mechanism as a dictionary function and the feed-forward layer as an inverse function. Through meticulous analysis, we identified the first weight matrix in the feed-forward layer as a Koopman Operator, responsible for transitioning the system to the next state in the dictionary space. Building upon this foundation, we extended our investigation to the encoder-decoder architecture, incorporating the encoder's output as an additional parameter in the self-attention mechanism. Leveraging parametric Koopman Operator theory, we elucidated the dynamics of information flow within the Transformer block, shedding light on how it processes input sequences and generates output sequences.

Notably, our theoretical conjectures were not merely speculative; they underwent empirical validation via practical experiments by training our Transformer model for the task of machine translation using the Anki dataset comprising English-French sentence pairs. While the initial experiment involving the basic training of our Transformer model highlighted a divergence in the operations within the decoder block from the Koopman Operator theory, subsequent endeavors were directed towards resolving this inconsistency. By devising custom loss functions and meticulously adjusting model parameters, we effectively realigned the operations of Transformer blocks with the tenets of Koopman Operator theory. Importantly, these refinements did not compromise performance, as evidenced by the sustained preservation of the BLEU score for machine translations. This achievement underscores the efficacy and feasibility of our methodology in augmenting the interpretability and controllability of Transformer models.

Looking ahead, there are several promising avenues for future research. Firstly, our exploration has primarily focused on understanding Transformer blocks in the context of machine translation tasks. Extending this analysis to other NLP tasks, such as text summarization or question answering, could provide further insights into the versatility and applicability of Koopman Operator theory in natural language processing. Additionally, investigating the implications of our findings on model optimization techniques and architectural design choices could lead to the development of more efficient and interpretable Transformer variants.

In conclusion, this thesis represents a step towards bridging the gap between machine learning and dynamical systems theory. By leveraging Koopman Operator theory to unravel the dynamics of Transformer blocks, we have laid the groundwork for a deeper understanding of these ubiquitous architectures. We believe that this interdisciplinary approach holds great promise for advancing the field of natural language processing and paving the way for the development of more interpretable and controllable machine learn-

ing models.

Bibliography

[1] S. Abdulwahab, M. Jabreel, and D. Moreno. "Deep Learning Models for Paraphrases Identification". In: (Sept. 2017).

[2] H. Arbabi. "Introduction to Koopman operator theory of dynamical systems". In: 2018.

[3] D. Bahdanau, K. Cho, and Y. Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Y. Bengio and Y. LeCun. 2015.

[4] P. Bevanda, S. Sosnowski, and S. Hirche. "Koopman operator dynamical models: Learning, analysis and control". In: *Annual Reviews in Control* 52 (2021), pp. 197–212. ISSN: 1367-5788.

[5] H. Broer and F. Takens. *Dynamical Systems and Chaos*. Vol. 172. Jan. 2011. ISBN: 978-1-4419-6869-2.

[6] P. Castiglione et al. *Chaos and Coarse Graining in Statistical Mechanics*. Cambridge University Press, 2008.

[7] K. Cho et al. "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by A. Moschitti, B. Pang, and W. Daelemans. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734.

[8] A. S. Dogra. *Dynamical Systems and Neural Networks*. 2020. arXiv: 2004.11826 [math.DS].

[9] A. S. Dogra and W. Redman. "Optimizing Neural Networks via Koopman Operator Theory". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 2087–2097.

[10] *French B2 Vocabulary Anki*. Accessed: 2024-04-21. 2024.

[11] G. Gandolfo. *Economic Dynamics*. Springer Berlin Heidelberg, 2010. ISBN: 9783642038716.

[12] Y. Guo et al. *Learning Parametric Koopman Decompositions for Prediction and Control*. 2023. arXiv: 2310.01124 [math.OC].

[13] H. Hewamalage, C. Bergmeir, and K. Bandara. "Recurrent Neural Networks for Time Series Forecasting: Current status and future directions". In: *International Journal of Forecasting* 37.1 (2021), pp. 388–427. ISSN: 0169-2070.

[14] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (1997), 1735–1780. ISSN: 0899-7667.

[15] C.-Z. A. Huang et al. "Music Transformer". In: *International Conference on Learning Representations*. 2019.

[16] E. M. Izhikevich. *Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting*. The MIT Press, July 2006. ISBN: 9780262276078.

[17] T. Jackson and A. Radunskaya. *Applications of Dynamical Systems in Biology and Medicine / edited by Trachette Jackson, Ami Radunskaya*. eng. 1st ed. The IMA Volumes in Mathematics and its Applications, 158. New York, NY: Springer New York : Imprint: Springer, 2015. ISBN: 1-4939-2781-7.

[18] D. Jurafsky and J. H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 1st. USA: Prentice Hall PTR, 2000. ISBN: 0130950696.

[19] C. Kelly. *Language Translation (English-French)*. 2020.

[20] S. Khan et al. “Transformers in Vision: A Survey”. In: *ACM Comput. Surv.* 54.10s (2022). ISSN: 0360-0300.

[21] B. O. Koopman. “Hamiltonian Systems and Transformation in Hilbert Space”. In: *Proceedings of the National Academy of Sciences* 17.5 (1931), pp. 315–318. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.17.5.315.

[22] Q. Le and T. Mikolov. “Distributed Representations of Sentences and Documents”. In: *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*. ICML’14. Beijing, China: JMLR.org, 2014, II–1188–II–1196.

[23] C. Lee, K. Park, and J. Kim. *Parameter-Varying Koopman Operator for Nonlinear System Modeling and Control*. 2023. arXiv: 2309.10278 [eess.SY].

[24] Q. Li et al. “Extended dynamic mode decomposition with dictionary learning: A data-driven adaptive spectral decomposition of the Koopman operator.” In: *Chaos* 27 10 (2017), p. 103111.

[25] Z. Liang et al. “Credit Assignment for Trained Neural Networks Based on Koopman Operator Theory”. In: *Front. Comput. Sci.* 18.1 (2023). ISSN: 2095-2228.

[26] G. Łukaszewicz and P. Kalita. *Navier–Stokes Equations: An Introduction with Applications*. Advances in Mechanics and Mathematics. Springer International Publishing, 2016. ISBN: 9783319277585.

[27] I. Manojlovic et al. “Applications of Koopman Mode Analysis to Neural Networks”. In: *ArXiv* abs/2006.11765 (2020).

[28] ManyThings.org. *Tab-delimited English-French Bilingual Sentence Pairs*. Accessed: 2024-04-20. 2024.

[29] I. Mezić. *Operator is the Model*. 2023. arXiv: 2310.18516 [math.DS].

[30] T. Mikolov et al. “Distributed Representations of Words and Phrases and their Compositionality”. In: *Advances in Neural Information Processing Systems*. Ed. by C. Burges et al. Vol. 26. Curran Associates, Inc., 2013.

[31] T. Mikolov et al. “Efficient Estimation of Word Representations in Vector Space”. In: *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*. 2013.

[32] Y. Ming et al. “Understanding Hidden Memories of Recurrent Neural Networks”. In: (Oct. 2017).

[33] I. Naiman and O. Azencot. “An Operator Theoretic Approach for Analyzing Sequence Neural Networks”. In: *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*. AAAI’23/IAAI’23/EAAI’23. AAAI Press, 2023. ISBN: 978-1-57735-880-0.

[34] OValery16. *Language-Translation-with-deep-learning-*. Accessed: 2024-04-21. 2024.

[35] K. Papineni et al. “Bleu: a Method for Automatic Evaluation of Machine Translation”. In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Ed. by P. Isabelle, E. Charniak, and D. Lin. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, July 2002, pp. 311–318.

[36] J. Pennington, R. Socher, and C. Manning. “GloVe: Global Vectors for Word Representation”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by A. Moschitti, B. Pang, and W. Daelemans. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543.

[37] D. Phuc and N. T. K. Phung. “Using Naïve Bayes Model and Natural Language Processing for Classifying Messages on Online Forum”. In: *2007 IEEE International Conference on Research, Innovation and Vision for the Future*. 2007, pp. 247–252.

[38] L. Rabiner and B. Juang. “An introduction to hidden Markov models”. In: *IEEE ASSP Magazine* 3.1 (1986), pp. 4–16.

[39] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pp. 533–536.

[40] H. Salehinejad et al. “Recent Advances in Recurrent Neural Networks”. In: *CoRR* abs/1801.01078 (2018). arXiv: 1801.01078.

[41] “TF–IDF”. In: *Encyclopedia of Machine Learning*. Ed. by C. Sammut and G. I. Webb. Boston, MA: Springer US, 2010, pp. 986–987. ISBN: 978-0-387-30164-8.

[42] P. J. SCHMID. “Dynamic mode decomposition of numerical and experimental data”. In: *Journal of Fluid Mechanics* 656 (2010), 5–28.

[43] R. M. Schmidt. “Recurrent Neural Networks (RNNs): A gentle Introduction and Overview”. In: *CoRR* abs/1912.05911 (2019). arXiv: 1912.05911.

[44] M. Schuster and K. Paliwal. “Bidirectional recurrent neural networks”. In: *IEEE Transactions on Signal Processing* 45.11 (1997), pp. 2673–2681.

[45] E. Shareghi et al. “Show Some Love to Your n-grams: A Bit of Progress and Stronger n-gram Language Modeling Baselines”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Ed. by J. Burstein, C. Doran, and T. Solorio. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4113–4118.

[46] S. Varsamopoulos, K. Bertels, and C. Almudever. “Designing neural network based decoders for surface codes”. In: (Nov. 2018).

[47] A. Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017.

[48] J. Weizenbaum. “ELIZA—a Computer Program for the Study of Natural Language Communication between Man and Machine”. In: *Commun. ACM* 9.1 (1966), 36–45. ISSN: 0001-0782.

[49] S. Wiggins. *Introduction To Applied Nonlinear Dynamical Systems And Chaos*. Vol. 4. Jan. 2003. ISBN: 0-387-00177-8.

[50] M. Williams, I. Kevrekidis, and C. Rowley. “A Data-Driven Approximation of the Koopman Operator: Extending Dynamic Mode Decomposition”. In: *Journal of Nonlinear Science* 25 (Aug. 2014).

[51] S. Yun et al. “Graph Transformer Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019.

[52] D. Zhang et al. “Simulating Reservoir Operation Using a Recurrent Neural Network Algorithm”. In: *Water* 11 (Apr. 2019), p. 865.

List of Figures

List of Algorithms

List of Tables