Université de Montréal

Beyond the Horizon: Improved Long-range Sequence Modeling, from Dynamical Systems to Language

par

Mahan Fathi

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en Informatique

January 25, 2024

Université de Montréal

Faculté des arts et des sciences

Ce mémoire intitulé

Beyond the Horizon: Improved Long-range Sequence Modeling, from Dynamical Systems to Language

présenté par

Mahan Fathi

a été évalué par un jury composé des personnes suivantes :

Guillaume Rabusseau

(président-rapporteur)

Pierre-Luc Bacon

(directeur de recherche)

Guillaume Lajoie

(membre du jury)

Résumé

Cette thèse est ancrée dans deux aspirations principales: (i) l'extension des longueurs de séquence pour une fidélité de prédiction supérieure pendant les phases d'entraînement et de test, et (ii) l'amélioration de l'efficacité computationnelle des modèles de séquence. Le défi fondamental de la modélisation de séquences réside dans la prédiction ou la génération précise sur de longs horizons. Les modèles traditionnels, tels que les Réseaux Neuronaux Récurrents (RNN), possèdent des capacités intrinsèques pour la gestion de séquences, mais présentent des lacunes sur de longues séquences. Le premier article, "Correction de Cours des Représentations de Koopman," introduit le Réencodage Périodique pour les Autoencodeurs de Koopman, offrant une solution à la dérive dans les prédictions à long horizon, assurant la stabilité du modèle sur de longues séquences. Les défis subséquents des RNN ont orienté l'attention vers les Transformateurs, avec une longueur de contexte bornée et un temps d'exécution quadratique. Des innovations récentes dans les Modèles d'Espace d'État (SSM) soulignent leur potentiel pour la modélisation de séquences. Notre second article, "Transformateurs d'État-Block," exploite les puissantes capacités de contextualisation des SSM, fusionnant les forces des Transformateurs avec les avantages des SSM. Cette fusion renforce la modélisation linguistique, surtout dans les contextes exigeant une large inference et contexte. En essence, cette thèse se concentre sur l'avancement de l'inférence de séquence à longue portée, chaque article offrant des approches distinctes pour améliorer la portée et la précision de la modélisation prédictive dans les séquences, incarnées par le titre "*Au-delà de l'Horizon.*"

Mots-clés: Koopman, Systèmes Dynamiques, Algorithmes en Temps d'Inférence, Inférence à Longue Portée, Grand Modélisation Linguistique, Transformers, Modèles d'Espace d'État, Modélisation Linguistique à Longue Portée

Abstract

This thesis is anchored in two principal aspirations: (i) the extension of sequence lengths for superior prediction fidelity during both training and test phases, and (ii) the enhancement of computational efficiency in sequence models. The fundamental challenge in sequence modeling lies in accurate prediction or generation across extended horizons. Traditional models, like Recurrent Neural Networks (RNNs), possess inherent capacities for sequence management, but exhibit shortcomings over extended sequences. The first article, "Course Correcting Koopman Representations," introduces Periodic Reencoding for Koopman Autoencoders, offering a solution to the drift in long-horizon predictions, ensuring model stability across lengthy sequences. Subsequent challenges in RNNs have shifted focus to Transformers, with a bounded context length and quadratic runtime. Recent innovations in State-Space Models (SSMs) underscore their potential for sequence modeling. Our second article, "Block-State Transformers," exploits the potent contextualization capabilities of SSMs, melding Transformer strengths with SSM benefits. This fusion augments language modeling, especially in contexts demanding extensive range inference and context. In essence, this thesis revolves around advancing long-range sequence inference, with each article providing distinctive approaches to enhance the reach and accuracy of predictive modeling in sequences, epitomized by the title "*Beyond the Horizon.*"

Keywords: Koopman, Dynamical Systems, Inference-time Algorithms, Long-range Inference, Large Language Modeling, Transformers, State-Space Models, Long-range Language Modeling

Contents

List of tables

List of figures

List of Abbreviations

BRecT	Block-Recurrent Transformer
BRT	Block-Recurrent Transformer
BST	Block-State Transformer
CPU	Central Processing Unit
DMD	Dynamic Mode Decomposition
eDMD	extended Dynamic Mode Decomposition
EMA	Exponentially Moving Average
FFT	Fast Fourier Transformer
GAU	Gated Attention Unit
GPU	Graphics Processing Unit

GRU	Gated Recurrent Unit
GSS	Gated State-Space Models
IFFT	Inverse Fast Fourier Transformer
LLM	Large Language Modeling
LM	Language Modeling
LRA	Long Range Arena
LSTM	Long Short-Term Memory
LDS	Linear Dynamical System
LTI	Linear Time Invariant
MF	Multi Filter
MH	Multi Head
MLP	Multilayer Perceptron
MPC	Model Predictive Control

MSE	Mean Square Error
MuJoCo	Multi-Joint dynamics with Contact
NLDS	Nonlinear Dynamical System
NLP	Natural Language Processing
PG	Project Gutenberg
RL	Reinforcement Learning
RNN	Recurrent Neural Network
S4	Structured State-Space Sequence Model
SH	Single Head
SSM	State-Space Model
TPU	Tensor Processing Unit
TRF	Transformer

Acknowledgements

Doing my Master's at Mila and Google has been a pivotal period, not only greatly impacting my academic life, but also shaping my perspective and personal growth. I am deeply grateful to all who made this possible.

First and foremost, I express my deepest appreciation to my supervisors, Pierre-Luc Bacon and Ross Goroshin. Their care and guidance mirrored the supportive presence of elder brothers I never had. In moments when disappointment threatened to overwhelm me, the drive to make you proud compelled me to rise and persevere. You have been role models to whom I will forever be indebted and continue to admire throughout my life.

Beyond the academic sphere, I am deeply appreciative of the unwavering support from friends and family. Nima, my brother, having family nearby is a blessing, but having a friend in that family is a treasure. To me, you are both and so much more, beyond what words can capture. Iman, Arash, and Ali, I say with a light heart that my love for you comes close to that which I hold for Nima.

To my parents, the recipients of my deepest gratitude, this final sentiment belongs to you. Mom and Dad, my love for you transcends all else. Your upbringing and unconditional love are gifts I carry with me always. Mom, in the sanctuary of my heart, your love shines brightest. Dad, your influence has sculpted my character with love's own hand. Boundlessly and eternally, my love for you both endures.

Thank you all for being the foundation upon which I build this milestone.

Mahan.

Introduction

Background

Sequence modeling, a pivotal task in machine learning, finds applications across diverse domains such as natural language processing, time-series forecasting, and dynamical system analysis. Central to this task is the ability to understand, generate, and predict data points sequentially.

Traditional models like RNNs and their variants (LSTMs, GRUs) have been the mainstay due to their capacity to maintain state across sequences. However, their known limitations in handling long-term dependencies due to issues like vanishing and exploding gradients have necessitated the exploration of alternative architectures. Moreover, unrolling a nonlinear RNN can only be carried out sequentially, as opposed Transformers that offer high parallelizability.

On the other hand, from an expressivity standpoint, *linear* Recurrent Neural Networks (RNNs) have demonstrated promising results in modeling extensive-range dependencies [**Orvieto et al., 2023**], provided specific initialization criteria are met. Moreover, recent advancements leverage hardware parallelism to enhance the computational efficiency of linear RNNs. Driven by these observations, we aim to closely study and integrate linear RNNs into challenging tasks such as long-range language modeling.

In this segment, we will address the foundational aspects of the following model categories:

- **Koopman Theory** establishes a one-to-one correspondence between the latent space linear dynamics and that of the original space – possibly nonlinear. This is an existence theorem, and does not guide as to how the linear dynamics can be deduced.
- **Koopman Autoencoders** can be characterized as RNNs with a singular linear recurrent layer. The field of dynamical systems, has offered intriguing insights into the linear representation of nonlinear dynamical systems, thus we intend to examine Koopman Autoencoders in this context, anticipating that these insights will pertain to more broader RNN contexts.

- **State-Space Models (SSMs)** can be viewed as linear Recurrent Neural Networks (RNNs), and achieve remarkable performance on tasks that require long-range modeling, e.g. Long Range Arena (LRA) [**Tay et al., 2020**]. The success of SSMs in information retention can be attributed to their specific initialization, provided by HiPPO framework [**Gu et al., 2020**].
- **Transformers** [**Vaswani et al., 2023**] marked a paradigm shift, particularly in language modeling, due to their ability to process sequences in parallel. However, the fixed-length context window in practice limits its capacity to infer over extended horizons.

This thesis builds on these foundational concepts, introducing innovative techniques and hybrid models that aim to push the boundaries of long-range sequence modeling further than previously achievable.

Koopman Theory Overview

The Koopman operator [**Koopman, 1931**] provides a distinctive approach to examining nonlinear dynamical systems through linear methodologies. This is achieved by projecting the dynamical system's state into an infinite-dimensional functional space. Consider a discrete-time nonlinear dynamical system described as:

$$x_{t+1} = F(x_t), \tag{0.0.1}$$

where $x_t \in \mathbb{R}^d$ represents the state at time t and $F : \mathbb{R}^d \to \mathbb{R}^d$ denotes the state transition function. The Koopman operator, represented by $\mathcal{K}$, operates on functions $g : \mathbb{R}^d \to \mathbb{C}$ (referred to as observables of the system), such that:

$$\mathcal{K}g(x_t) := g \circ F(x_t) = g(x_{t+1}) \tag{0.0.2}$$

This implies that the Koopman operator propels the function g one temporal step by the evolution of the dynamical system. A salient observation is that, while the dynamical system might manifest nonlinear characteristics in its inherent state space, the evolution of observables under the influence of the Koopman operator remains linear. Such linearization offers the advantage of employing linear methodologies to investigate the attributes and behavior of the primal nonlinear system. The Koopman theory is revisited with mathematical rigor in the appendices of the first article.

Koopman Autoencoders

A Koopman Autoencoder is a neural network architecture that seeks to integrate the principles of the Koopman operator theory into the framework of autoencoders to learn representations of dynamical systems. Koopman Autoencoders consist of three trainable constituent parts: (1) The Encoder ϕ; this component takes in a state (or observation) of the system and encodes it into a lower-dimensional latent space representation. This latent representation ideally captures the key features or dynamics of the system. (2) The Koopman transition matrix $\mathbf{K}$; Within the latent space, the dynamics are modeled. Typically, the dynamics are represented as a linear transformation, consistent with the Koopman operator's notion of linearly evolving observables. (3) The Decoder ψ; The latent representation is then decoded back into the original state space. This step ensures that the latent space representation maintains the crucial features necessary to recreate the state.

The relationship between the original and latent states is then established as follows:

$$z_t = \phi(x_t), \quad z_{t+1} = \mathbf{K} z_t, \quad x_t = \psi(z_t), \tag{0.0.3}$$

where $z \in \mathbb{R}^n$ represents the latent states. Moreover, Koopman autoencoders can be naturally extended to accommodate controlled dynamical systems. When control inputs are present, the original dynamics can be expressed as:

$$x_{t+1} = F(x_t, u_t). \tag{0.0.4}$$

To incorporate control within the Koopman perspective, we expand the observable function to integrate the control inputs. Formally, we define:

$$\xi_t = \begin{bmatrix} x_t \\ u_t \end{bmatrix}.$$

Hence, our objective within the Koopman framework becomes finding an operator $\mathbf{K}$ such that:

$$\phi(\xi_{t+1}) = \mathbf{K}\phi(\xi_t),$$

where ϕ denotes the observable function on the joint state-control vector ξ_t.

When the Koopman Autoencoder's observable function and the linear operator are distilled to their core functionality, the Koopman Autoencoder can be perceived as an RNN with a singular linear recurrent layer. The linearity of the Koopman operator mirrors the linear transformation of an RNN's recurrent layer, albeit with the theoretical underpinning of Koopman theory in dynamical systems.

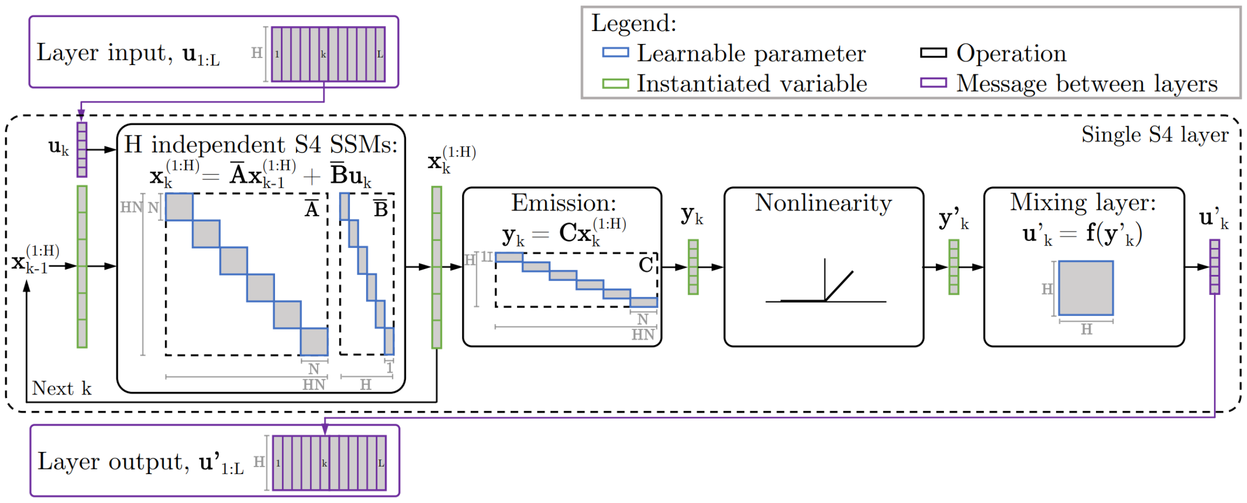


Fig. 0.1. S4 layer, adopted from [**Smith et al., 2023**]. S4 model is a stack of multiple S4 layer. The outputs of the convolutions are mixed using feed-forward networks at every layer.

State-Space Models

State-Space Models employ a structured convolutional kernel initialization, derived from the unrolling of a linear time-invariant (LTI) dynamical system presented as:

$$\begin{aligned} x_k &= \mathbf{A}x_{k-1} + \mathbf{B}u_k \ , \\ y_k &= \mathbf{C}x_k + \mathbf{D}u_k \ . \end{aligned} \tag{0.0.5}$$

This system is defined by its state matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$, and vectors $\mathbf{B} \in \mathbb{R}^{N\times 1}$, $\mathbf{C} \in \mathbb{R}^{1\times N}$, and $\mathbf{D} \in \mathbb{R}^{1\times 1}$. Within this context, the SSM transforms a 1-D input signal u_k into a 1-D output signal y_k. The SSM first elevates the input to an internal N-D representation, denoted as x_k, and subsequently projects it to a scalar via the $\mathbf{C}$ matrix. The component $\mathbf{D}u_k$ acts like a direct path or bypass and will not be considered in the ensuing discussions.

By examining the recurrent relation, the output y_k can be depicted as a discrete convolution by explicitly unrolling the sequence:

$$\begin{aligned} \text{Set} \quad x_{-1} &:= \vec{0} \ , \\ y_k &= \sum_{j=0}^{k} \mathbf{C}\mathbf{A}^j\mathbf{B} \cdot u_{k-j} \ . \end{aligned} \tag{0.0.6}$$

To construct the SSM filter $\mathbf{K} \in \mathbb{R}^L$, we gather entries from $\mathbf{CA}^k\mathbf{B}$. The convolution can then be articulated as:

$$\begin{aligned} \mathbf{K} &= (\mathbf{CB}, \mathbf{CAB}, \ldots, \mathbf{CA}^{L-1}\mathbf{B}) \ , \\ y_k &= \sum_{j=0}^{k} \mathbf{K}_j \cdot u_{k-j} \ , \\ y &= \mathbf{K} * u \ . \end{aligned} \tag{0.0.7}$$

The time complexity of directly computing the convolution is $\mathcal{O}(L^2)$, where L is the length of the signals. The Convolution Theorem asserts that convolution in the time domain is equivalent to multiplication in the frequency domain, mathematically represented as:

$$\mathcal{F}\{\mathbf{K} * u\} = \mathcal{F}\{\mathbf{K}\} \cdot \mathcal{F}\{u\}$$

Utilizing this theorem, the convolution can be computed through the following procedure:

(1) Compute the Fast Fourier Transform (FFT) of signals $\mathbf{K}$ and u to transition to the frequency domain. This operation exhibits a time complexity of $\mathcal{O}(L \log L)$.
(2) Perform element-wise multiplication of the resulting frequency representations, an $\mathcal{O}(L)$ operation.
(3) Use the inverse FFT (IFFT) to revert the result to the time domain, also an $\mathcal{O}(L \log L)$ operation.

This results in an overall time complexity of $\mathcal{O}(L \log L)$, offering a potential computational advantage over direct convolution for large L.

The FFT algorithm can be executed in parallel, particularly when implemented via a divide-and-conquer strategy (similar to the Cooley-Tukey radix-2 algorithm [**Cooley and Tukey, 1965**]). Each stage of the FFT is amenable to parallel computation, making it apt for execution on architectures like multi-core processors, GPUs, and TPUs. Moreover, the element-wise multiplication in the frequency domain can also be parallelized as each product computation is independent.

The remaining computational bottleneck at this point is the rematerialization of the convolutional kernel from its constituent parts, i.e. $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$. Naive computation of the kernel involves L successive multiplication by $\mathbf{A}$, resulting in $\mathcal{O}(N^2L)$ operations. A key contribution of S4 features a specific way to overcome this bottleneck, which is beyond the scope of this introductory section.

Until now, we have not addressed the underlying principles that empower SSMs to excel in memorization tasks. The memorization aspect is ensured through the use of specific parameterizations for $\mathbf{A}$ and $\mathbf{B}$ matrices, based on the HiPPO framework [**Gu et al., 2020**]. The progression of the states, denoted by x_k in the State-Space Model, as dictated by the provided matrices, aligns with real-time modifications to the coefficients of a particular polynomial set.

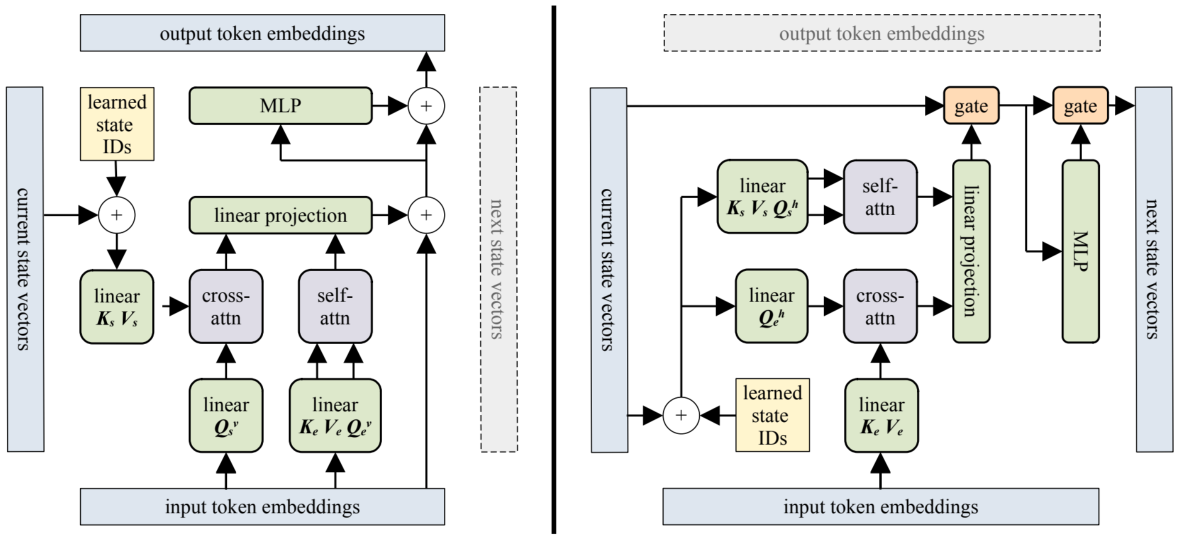


Fig. 0.2. Block-Recurrent Transformer layer, adopted from [**Hutchins et al., 2022**].

The inputs can then be reconstructed under an importance measure function by expanding the polynomial bases using their corresponding coefficients; hence, the coefficients can be considered a representation of the "memory." We direct readers seeking a comprehensive grasp on the topic to [**Gu et al., 2020**].

Block Transformers

For brevity, we presuppose the reader's familiarity with the attention mechanism and focus exclusively on the architectural nuances of **Block Transformers**, as presented by citeintro:hutchins2022block (see Figure 0.2). Block-Recurrent Transformer (BRT) was proposed as an efficient way to extend the context length of the Transformer architecture. Naively feeding a sequence of tokens of length L, into a Transformer would incur $\mathcal{O}(L^2)$ operations. By chunking the sequence of inputs into smaller windows of length W, Block-Recurrent Transformers render the computational requirement linear with respect to L, $\mathcal{O}(L \cdot W)$. This results directly from the execution of Transformer blocks across the segments, with a complexity of $\mathcal{O}(W^2)$, iterated L/W times, i.e. the number of chunks existing in the sequence.

Block-Recurrent Transformers employ recurrence to facilitate communication between Transformer blocks (see Figure 0.2). Executing a Transformer block yields a collection of output tokens, which are subsequently supplied to the subsequent layer–vertical direction, in conjunction with a set of ensuing hidden states. These states are then inputted to the next Transformer that processes the succeeding token chunk–horizontal direction.

Motivation

This thesis is driven by two core motivations: (i) extending the sequence lengths both at training and test time for improved prediction fidelity and accuracy, (ii) improving the computational efficiency of sequence models.

In sequence modeling, accurately predicting or generating sequences over extended horizons is a fundamental challenge. Traditional models, particularly Recurrent Neural Networks (RNNs), have shown promise in their capacity to handle sequences, owing to their recursive nature and theoretically unbounded context length. However, when these models are unrolled beyond the horizons they were trained for, they tend to exhibit diverging behaviors, leading to significant degradation in prediction quality or even complete loss of coherent structure [**Fathi et al., 2023**].

The first article in this thesis, "*Course Correcting Koopman Representations*," addresses this precise challenge. We introduce "*Periodic Reencoding*," a novel technique designed for Koopman Autoencoders [**Brunton et al., 2021**]—a specialized instance of RNNs characterized by a single linear recurrent layer. This method significantly mitigates the inherent drift issue in long-horizon predictions by strategically reencoding the hidden states at periodic intervals, thereby preserving the model's stability and prediction fidelity over extended sequences.

On the other hand, while RNNs possess theoretical advantages for long-range sequence modeling due to their recursive nature, in practice, they often suffer from issues such as vanishing/exploding gradients. These challenges have prompted a shift to Transformers as other general-purpose alternatives for sequence modeling. Transformers have bounded context length and quadratic runtime with respect to the context length. Thus the efforts have been concentrated on expanding the context length by enhancing the efficiency of the attention mechanism, either via hardware-aware implementations or approximation [**Wang et al., 2020**]. Recent advancements in State-Space Models (SSMs), particularly the S4 model [**Gu et al., 2021**], have reinvigorated interest in RNNs, highlighting their potential for effective sequence modeling through improved state representation. State-Space Models handle long-range dependencies, excel at memorization tasks, and can be run in parallel in subquadratic time. Our second article, "*Block-State Transformers*," heavily uses the powerful contextualization abilities of State-Space Models, proposing a hybrid model that integrates the strengths of Transformers with those of SSMs. This synergy enhances language modeling capabilities, particularly in scenarios requiring long-range inference.

Both articles presented in this thesis share a common objective: significantly enhancing long-range sequence inference—hence the title "*Beyond the Horizon*". The first achieves this by introducing stability to the inference mechanisms of fully-observable dynamical systems through Periodic Reencoding. In contrast, the second demonstrates that our hybrid model can generalize effectively beyond the training sequence length, a critical milestone in language modeling.

Thesis Structure

This thesis is articulated around two primary research articles, each addressing unique challenges in long-range sequence modeling:

- The first article, "*Course Correcting Koopman Representations*," focuses on enhancing the stability of long-range predictions in dynamical systems. By introducing the concept of "Periodic Reencoding" within the framework of Koopman Autoencoders, we demonstrate marked improvements in the model's ability to infer extended sequences without the characteristic divergence observed in traditional latent dynamics layers.
- Our second article, "*Block-State Transformers*," pivots towards the domain of language modeling. Here, we propose a novel hybrid model that combines the strengths of State-Space Models and Transformers. This model, characterized by its enhanced ability to handle long-range dependencies, not only excels in standard language modeling benchmarks but also shows remarkable generalization capabilities beyond the sequence lengths observed during training.

Each article is presented as a self-contained chapter with its own introduction, methodology, experiments, results, and conclusion, allowing for a focused exploration of the respective topics.

Contributions

In the first article, we highlight the occurrence of "drift" in models that employ latent dynamics modeling. To address this, we introduce "Periodic Reencoding," a proficient inference-time technique devised to rectify such drift while capitalizing on the computational advantages of linear operations between reencoding stages. While our primary inspiration stemmed from models with linear latent dynamic components, i.e., Koopman Autoencoders, we also demonstrate the method's applicability to nonlinear scenarios and the presence of drift in those cases. Moreover, we show that integrating Periodic Reencoding during training can further enhance model performance.

In the second article, we integrate Transformers and State-Space Models in novel ways to enhance language modeling quality while achieving $10\times$ speed-up compared to Block-Recurrent Transformers layers. We present evidence that the model can generalize well beyond the lengths of sequences used during training. Furthermore, we demonstrate that the BST layers are favorable when scaling the model, up to over a billion parameters.

References

[Brunton et al., 2021] Brunton, S. L., Budišić, M., Kaiser, E., and Kutz, J. N. (2021). Modern koopman theory for dynamical systems. *arXiv preprint arXiv:2102.12086*.

[Cooley and Tukey, 1965] Cooley, J. and Tukey, J. (1965). An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301.

[Fathi et al., 2023] Fathi, M., Gehring, C., Pilault, J., Kanaa, D., Bacon, P.-L., and Goroshin, R. (2023). Course correcting koopman representations.

[Gu et al., 2020] Gu, A., Dao, T., Ermon, S., Rudra, A., and Re, C. (2020). Hippo: Recurrent memory with optimal polynomial projections.

[Gu et al., 2021] Gu, A., Goel, K., and Ré, C. (2021). Efficiently modeling long sequences with structured state spaces. *CoRR*, abs/2111.00396.

[Hutchins et al., 2022] Hutchins, D., Schlag, I., Wu, Y., Dyer, E., and Neyshabur, B. (2022). Block-recurrent transformers. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K., editors, *Advances in Neural Information Processing Systems*.

[Koopman, 1931] Koopman, B. O. (1931). Hamiltonian systems and transformation in hilbert space. *Proceedings of the National Academy of Sciences*, 17(5):315–318.

[Orvieto et al., 2023] Orvieto, A., Smith, S. L., Gu, A., Fernando, A., Gulcehre, C., Pascanu, R., and De, S. (2023). Resurrecting recurrent neural networks for long sequences.

[Smith et al., 2023] Smith, J. T. H., Warrington, A., and Linderman, S. W. (2023). Simplified state space layers for sequence modeling.

[Tay et al., 2020] Tay, Y., Dehghani, M., Abnar, S., Shen, Y., Bahri, D., Pham, P., Rao, J., Yang, L., Ruder, S., and Metzler, D. (2020). Long range arena: A benchmark for efficient transformers.

[Vaswani et al., 2023] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2023). Attention is all you need.

[Wang et al., 2020] Wang, S., Li, B. Z., Khabsa, M., Fang, H., and Ma, H. (2020). Linformer: Self-attention with linear complexity.

Chapter 1 First article

Course Correcting Koopman Representations

by

Mahan Fathi[1], Clement Gehring[2], Jonathan Pilault[3],
David Kanaa[4], Pierre-Luc Bacon[5], and Ross Goroshin[6]

([1]) Google DeepMind, Mila, Université de Montréal
([2]) Mila, Université de Montréal
([3]) Mila, Polytechnique Montréal
([4]) Mila
([5]) CIFAR AI Chair, Mila, Université de Montréal
([6]) Google DeepMind

This article was submitted in International Conference on Learning Representations 2024 – (ICLR'24).

The main contributions of Mahan Fathi for this articles are presented.
I, Mahan Fathi, led the project, encompassing the conception of the primary idea, writing the code, conducting experiments, overseeing the majority of the manuscript composition and logistical operations, such as submission, publication, and presentation.

Ross and Pierre-Luc collaboratively supervised this research endeavor. It was Ross who guided the overarching focus towards Koopman Autoencoders and studied the related works

thoroughly. Clement participated in pivotal meetings that culminated in the paper's key discovery and offered valuable feedback on the manuscript. Jonathan and David helped with writing parts of the appendices.

Résumé. Les représentations de Koopman visent à apprendre les caractéristiques des systèmes dynamiques non linéaires (SDNL) qui conduisent à une dynamique linéaire dans l'espace latent. Théoriquement, de telles caractéristiques peuvent être utilisées pour simplifier de nombreux problèmes liés à la modélisation et à la commande des SDNL. Dans ce travail, nous étudions les formulations d'autoencodeurs de ce problème, ainsi que les différentes manières dont elles peuvent être utilisées pour modéliser la dynamique, notamment en ce qui concerne la prédiction des états futurs sur de longues périodes. Nous découvrons plusieurs limitations liées à la prédiction des états futurs dans l'espace latent et proposons un mécanisme d'inférence que nous appelons Réencodage Périodique, conçu pour capturer fidèlement la dynamique à long terme. Nous justifions cette méthode à la fois sur le plan analytique et empirique grâce à des expérimentations menées dans des SDNL de faible et de haute dimension.
Mots clés : Koopman, Systèmes Dynamiques, Algorithmes en Temps d'Inférence, Inférence à Longue Portée

Abstract. Koopman representations aim to learn features of nonlinear dynamical systems (NLDS) which lead to linear dynamics in the latent space. Theoretically, such features can be used to simplify many problems in modeling and control of NLDS. In this work we study autoencoder formulations of this problem, and different ways they can be used to model dynamics, specifically for future state prediction over long horizons. We discover several limitations of predicting future states in the latent space and propose an inference-time mechanism, which we refer to as Periodic Reencoding, for faithfully capturing long term dynamics. We justify this method both analytically and empirically via experiments in low and high dimensional NLDS.
Keywords: Koopman, Dynamical Systems, Inference-time Algorithms, Long-range Inference

1.1. Introduction

Recent research has shown a growing interest in learning representations of nonlinear dynamical systems (NLDS) in which the dynamics become linear in the latent space. Linear dynamics in the latent space offer distinct advantages, including the capability to derive closed-form solutions to optimal control problems using LQR solvers [**Kalman, 1960**]. System identification and interpretability are also greatly simplified for linear dynamical systems. From a computational standpoint, advancing a linear system forward can be executed more efficiently by leveraging parallelism [**Gu et al., 2021a, Smith et al., 2023**]. An example of this approach is presented in S4 [**Gu et al., 2021a**], a carefully designed deep state-space model (SSM) that takes advantage of the parallelism offered by discrete linear dynamics.

Given the advantages of linear systems, a challenge in the study of dynamics and data-driven modelling resides in extracting global linear representations of nonlinear systems. In this regard, Koopman Theory [**Koopman, 1931, Koopman and v. Neumann, 1932**] provides a framework in which nonlinear dynamics can be cast into linear ones in a space of measurements, spanned by a basis of characteristic functions, which are the eigenfunctions of Koopman's composition operator. However, identifying such characteristic functions as well as the structure of the linear transition operator can prove very difficult, in general, as they depend heavily on the qualitative dynamical properties of the system under study, necessitating the need to resort to approximate methods.

Recently, several studies have explored the integration of deep learning architectures with Koopman operator theory, primarily in the context of small scale well-behaved problems. [**Azencot et al., 2020**] train a backward-compatible Koopman matrix in conjunction with its original forward counterpart, by assuming reversible dynamics, ensuring stability during training by driving the eigenvalues of the operators to be close to one. [**Frion et al., 2023**] adopt a similar strategy by promoting orthogonality of the Koopman matrix through an auxiliary loss. [**Lusch et al., 2018**] learn generalized Koopman representations for systems including those exhibiting continuous spectra. [**Mondal et al., 2023**] employ a Koopman-based model in the context of Model-based Reinforcement Learning as a short-horizon (20-step) planner.

In this work, we further explore the use of principles outlined by Koopman theory [**Brunton et al., 2021**], as a guiding framework to obtain linear representations of NLDS within an autoencoder framework. We observe that unrolling the linear dynamics in latent space leads to long term drift in trajectories when mapped back to the space of observables (state space). Prior works achieved stable unrolls at inference time by restricting the eigenvalues of the Koopman/linear operator, and training on long sequences. We posit that such requirements, i.e. long training sequences, are unnecessary, especially when modelling a fully observable dynamical system. In this work we uncover two limitations of generating trajectories in the latent space: (i) long horizon trajectories cross, violating uniqueness of solutions of dynamical systems, and (ii) latent space trajectory generation is unable to capture switching dynamics between fixed points [**Lan and Mezić, 2013**]. To overcome these issues, we introduce a simple inference method called *Periodic Reencoding* that produces high accuracy predictions over long horizons. We derive analytic expressions and present a special case example (Appendix 1.5) that provides intuition about why this method better captures long-term dynamics. Finally, we empirically validate our approach on a set of established NLDSs, and also demonstrate these findings on offline reinforcement learning datasets in more complex environments within D4RL [**Fu et al., 2020**].

1.2. Deep Koopman Autoencoders

Given a nonlinear dynamical system, Koopman theory attempts to approximate or "explain" nonlinear transitions with a linear dynamical system. For example, Dynamic Mode Decomposition (DMD) [**Brunton et al., 2021**] tries to approximate the infinite dimensional Koopman operator by fitting discrete transition data collected from a nonlinear dynamical system. More specifically, using an integration scheme in time a discrete nonlinear dynamical system can be obtained from its continuous time counterpart, $x_{t+1} = \mathbf{F}(x_t)$. DMD then solves the following optimization problem over the dataset of all transition pairs, $\mathcal{D} = \{(x_t, x_{t+1}) \mid x_t, x_{t+1} \in \mathbb{R}^d\}$:

$$\mathbf{K}^{\text{DMD}} = \arg\min_{\hat{\mathbf{K}}} \sum_{\mathcal{D}} \|x_{t+1} - \hat{\mathbf{K}} x_t\|_2, \tag{1.2.1}$$

where the $\mathbf{K}^{\text{DMD}} \in \mathbb{R}^{d\times d}$ matrix is the finite-dimensional approximation to the linear Koopman operator. Thus DMD treats the state itself as the features, by simply fitting the Koopman matrix directly. Extensions to this approach include "extended" DMD, eDMD [**Williams et al., 2015a, Williams et al., 2015b, Schütte et al., 2016**], which augments the state with fixed nonlinear transformations of the state. In the spirit of end-to-end learning, we are interested in data driven approaches that learn nonlinear transformations of the state. Koopman features can be learned in an autoencoder setting by minimizing:

$$\min_{\hat{\mathbf{K}},\phi,\psi} \sum_{\mathcal{D}} \|x_t - \psi(\phi(x_t))\|_2 + \lambda \cdot \|\phi(x_{t+1}) - \hat{\mathbf{K}}\phi(x_t)\|_2 \tag{1.2.2}$$

where the encoder $\phi : \mathbb{R}^d \to \mathbb{R}^n$, decoder $\psi : \mathbb{R}^n \to \mathbb{R}^d$ are parameterized function approximators, and $\lambda > 0$ is a scalar. The feature vector $z_t \in \mathbb{R}^n$, representing the Koopman embedding for which the dynamics are linear, obeys the following relations:

$$z_t \approx \phi(x_t), \quad z_{t+1} \approx \mathbf{K} z_t, \quad x_t \approx \psi(z_t) \tag{1.2.3}$$

These relations are approximate because, for example, reconstructed states are approximations of the corresponding true states. Once trained, the encoder, decoder, and $\mathbf{K} \in \mathbb{R}^{n\times n}$ comprise a complete model of the dynamical system, potentially capable of performing long-range state prediction over arbitrarily long time horizons.

Though other feature learning methods, such as contrastive learning, are possible [**Lyu et al., 2023**], we study the autoencoder formulation because it is more pervasive in the literature. Furthermore, the learned decoders from this formulation allow for solving the control problems in latent space and mapping control signals back to phase space. This is non-trivial using contrastive learning which only trains an encoder.

1.3. Method

1.3.1. Training Sequence

In this section we formally outline the training objective for sequential data. We start by using the continuous parameterization of the Koopman dynamics, for *controlled systems*. An autonomous controlled system is described by $\dot{x} = f(x, u)$, where u is an exogenous control input. Assuming that a bounded Koopman matrix, $K \in \mathbb{R}^{n\times n}$, can be approximated for measurements $z = \phi(x) \in \mathbb{R}^n$, the linear dynamics are then prescribed by:

$$\frac{\mathrm{d}}{\mathrm{d}t}\phi(x) = K\phi(x) + L\omega(u), \tag{1.3.1}$$

where $L \in \mathbb{R}^{n\times m}$ represents the controlled latent dynamics for external coded input $\upsilon = \omega(u) \in \mathbb{R}^m$. We optimize the objective by taking gradient steps directly over the continuous parameterization of the Koopman dynamics, i.e. K and L, which we discretize via the bilinear method [**Tustin, 1947**], over a timestep of δ:

$$z_{t+1} = \mathbf{K}z_t + \mathbf{L}\upsilon_t, \tag{1.3.2}$$

$$\text{where} \quad \mathbf{K} = \big(I - \frac{\delta}{2}K\big)^{-1}\big(I + \frac{\delta}{2}K\big) \quad \text{and} \quad \mathbf{L} = \big(I - \frac{\delta}{2}K\big)^{-1}\delta L. \tag{1.3.3}$$

We treat δ as a trainable variable as well, and assume that samples are uniformly distributed in time. The Koopman Autoencoder architecture consists of the following trainable components, (i) the latent dynamics K, L and δ (ii) the state encoder ϕ (iii) the action encoder ω (iv) and the state decoder ψ. The training data consists of an initial state, x_t, and a sequence of following actions and states, i.e. $(u_t, x_{t+1}, \cdots, u_{t+T-1}, x_{t+T})$ of length T. The model takes in the initial state and the sequence of actions as input and is tasked to predict the sequence of future states. To respect the Koopman dynamics, i.e. ensuring that $\mathbf{K}$ and $\mathbf{L}$ are the only means for advancing the dynamics, we minimize the "Aligment", "Reconstruction", and

"Prediction" losses that prevent trivial solutions.

$$\mathcal{L}_{\text{Align}} = \sum_{i=1}^{T} \|\hat{z}_{t+i} - \phi(x_{t+i})\|_2 \tag{1.3.4}$$

$$\mathcal{L}_{\text{Reconst}} = \sum_{i=0}^{T} \|x_{t+i} - \psi(z_{t+i})\|_2 \tag{1.3.5}$$

$$\mathcal{L}_{\text{Pred}} = \sum_{i=1}^{T} \|x_{t+i} - \psi(\hat{z}_{t+i})\|_2 \tag{1.3.6}$$

In the above equations, $\hat{z}_t$ denotes the resultant latent code after one or more applications of Koopman dynamics, while z_t represents the resultant latent state immediately after encoding (see Figure 1.1).

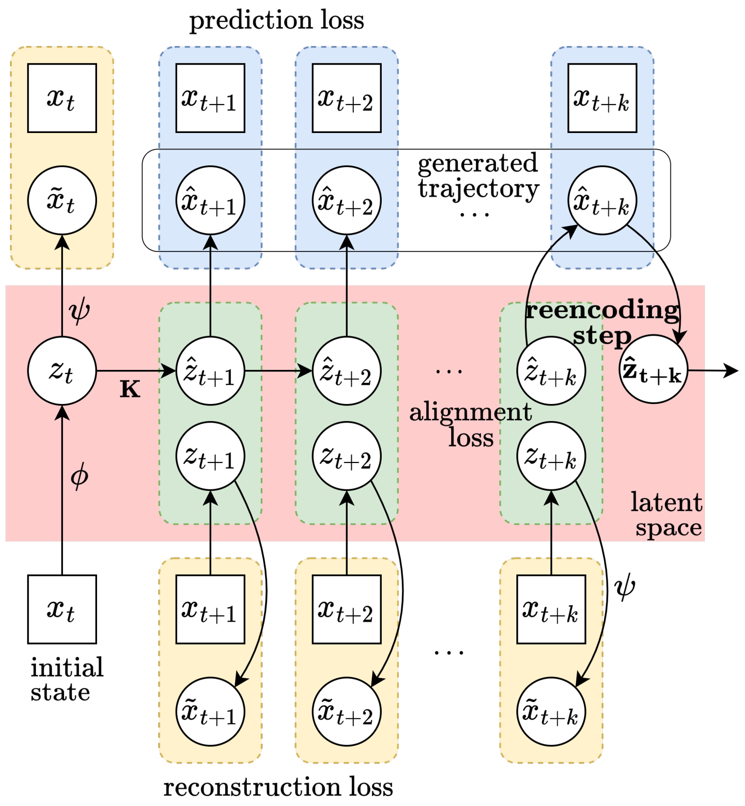


Fig. 1.1. Course Correcting Koopman Autoencoder Unrolls via Periodic Reencoding. Unrolls are generated by first encoding the initial condition, then linearly advancing the dynamics in latent space, and finally decoding the trajectory back to the original space. Values enclosed within squares represent ground truth, while those enclosed within circles are inferred by the model. The objective consists of an alignment loss (shown in green), a reconstruction loss (in yellow), a prediction loss (in blue). We propose Periodic Reencoding; at every k steps in the figure. The latent state, $\hat{\mathbf{z}}_{\mathbf{t+k}}$, is decoded and subsequently reencoded. Because the encoder/decoder are not exact inverses, the reencoded feature vector is different from the original. Control inputs are omitted for simplicity.

1.3.2. Trajectory Generation

The quality of a model of a dynamical system can be assessed by generating trajectories in the state space of the original dynamical system, starting from some initial condition. There are two methods for generating trajectories.

Without Reencoding. This method generates the entire trajectory in the latent space and only uses the encoder to obtain the initial condition. The decoder ψ is applied to every point on generated trajectory to obtain the corresponding *curve* in state space. More specifically, given:

$$\dot{z} = Kz|_{z_0=\phi(x_0)} \quad \text{and} \quad x = \psi(z),$$

we can write the explicit solution in state space as:

$$x(t) = \psi(e^{Kt}\phi(x_0)). \tag{1.3.7}$$

It is the solution to a linear dynamical system, given by the matrix exponential, in latent space which is then mapped to the original state space using ψ. Under this setting, the relationship between z–space and x–space is established by mapping *trajectories* generated in the z–space via linear dynamics, to *curves* in the x–space.

In the standard autoencoder setup, the mappings between the original and latent spaces are not strictly one-to-one. This characteristic becomes especially significant when the dimensionality of the latent space is substantially larger than that of the original space, i.e. $n \gg d$, as is typical in Koopman autoencoders – the inverse function theorem requires that $n = d$. Because of this, the curves generated without reencoding could potentially intersect with themselves, and therefore won't faithfuly capture the characteristics of a *trajectory*, generated by a dynamical system. This directly arises from the existence of multiple points in z–space that lie on the same trajectory generated by the linear system, that are mapped to the same point in x–space, due to lack of injectivity in the decoder, ψ. This is clearly illustrated in Figure 1.4 (a) where phase lines intersect. Intersecting phase lines are impossible to generate with continuous dynamical systems because they imply that multiple solutions exist corresponding to the unique initial condition given by the cross-over point.

With Reencoding. This method uses the encoder/decoder to iteratively generate a trajectory in state-space. The encoder, Koopman operator, and decoder effectively define a dynamical system in the state space, x. More specifically, given:

$$\dot{z} = Kz|_{z_0=\phi(x_0)}, \quad z = \phi(x), \quad \text{and} \quad x = \psi(z),$$

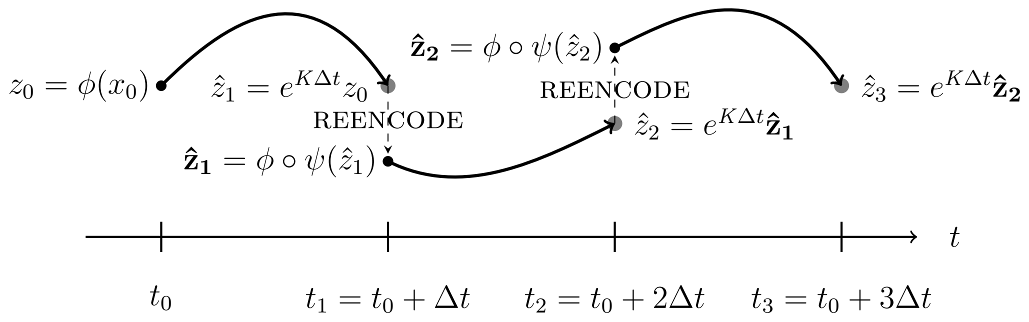
we can express the dynamical system that implicitly defines the solution in state space as:

$$\dot{x} = J_\psi K \phi(x) \tag{1.3.8}$$

Where J_ψ is the Jacobian of ψ. When generating trajectories without reencoding, linearity in the latent space is assumed to hold for all times, i.e. globally. In contrast, when generating trajectories with reencoding, the linearity property is assumed to hold only locally. Equation 1.3.8 gives rise to a *dynamical system with feedback*, thus the relationship between $z-$space and $x-$space is governed by point-wise mapping of dynamics.

In theory, the trajectories generated using this method can faithfully capture the dynamics and will not produce invalid trajectories that cross, however, reencoding at every step poses two significant drawbacks: (i) repeated applications of the encoder can potentially accumulate errors much faster than repeated applications of $\mathbf{K}$, as seen in Figure 1.4 (b), and (ii) it is not computationally efficient because unrolling must be performed sequentially due to presence of nonlinear operations at every step, unlike the parallelizability of linear operations.

We introduce ***Periodic Reencoding***, a technique spanning the middle ground between trajectory generations with and without reencoding. When generating trajectories with periodic reencoding, we decode and reencode periodically, in continuous time at Δt intervals (Figure 1.2), and in discrete time every k steps (Figure 1.1), treating Δt or k as hyper-parameters. Given that the encoder and decoder are not perfect inverses of one another, the output of the reencoding step is going to be different from the original point. In other terms, the mapping between the original and latent space lacks *bijectivity*.



$z_0 = \phi(x_0)$
$\hat{z}_1 = e^{K\Delta t} z_0$
REENCODE
$\hat{\mathbf{z}}_\mathbf{1} = \phi \circ \psi(\hat{z}_1)$
$\hat{\mathbf{z}}_\mathbf{2} = \phi \circ \psi(\hat{z}_2)$
REENCODE
$\hat{z}_2 = e^{K\Delta t} \hat{\mathbf{z}}_\mathbf{1}$
$\hat{z}_3 = e^{K\Delta t} \hat{\mathbf{z}}_\mathbf{2}$
t
t_0
$t_1 = t_0 + \Delta t$
$t_2 = t_0 + 2\Delta t$
$t_3 = t_0 + 3\Delta t$

Fig. 1.2. Periodic Reencoding in continuous time for trajectories in latent space. In this figure, the unrolled latent before and after reencoding are denoted as $\hat{z}_i$ and $\hat{\mathbf{z}}_\mathbf{i}$, respectively.

By Periodic Reencoding, we expand and harness the applicability of local linear Koopman dynamics around the initial state all the while mitigating the accumulation of encoding errors. We observe that periodically applying reencoding is an effective way to mitigate the drift accumulated by generating trajectories in the latent space. We term this property **"course correction."** Using our method, we are able to generate stable, plausible, and accurate unrolls over extended horizons, while remaining computationally efficient. In Section 1.4, we empirically establish the effectiveness of periodic reencoding for stable and accurate long-range predictions, in contrast to unrolls without reencoding or reencoding at every step.

Furthermore, we demonstrate that periodic reencoding can be beneficially integrated into the training process to achieve further improvements. [**Lan and Mezić, 2013**] showed that NLDS with multiple fixed points can only be linearized within the basin of attraction of each fixed point. This reveals another limitation of unrolls without reencoding – they cannot capture the switching dynamics between multiple fixed points (see the example in Appendix 1.5). It is important to make the distinction between reencoding and "teacher forcing" [**Bengio et al., 2015**]. Teacher forcing periodically uses ground truth during training to avoid error accumulation. Reencoding never uses ground truth data.

1.4. Results

We empirically evaluated our proposed approach on a number of highly nonlinear environments with varying dimensionality. We begin by modelling the forward dynamics of well known, nonlinear, low dimensional dynamical systems. We further extend the results to more practical, higher dimensional, robotic environments implemented in MuJoCo. We use the D4RL dataset [**Fu et al., 2021**] to train our Koopman autoencoder. Lastly we employ our proposed approach as an open-loop controller for locomotion tasks in D4RL.

1.4.1. Dynamical Systems

We use well-established dynamical systems as benchmarks for forward dynamics modeling. Despite their low dimensionality, these systems display interesting nonlinear dynamics, including multiple fixed points. Nonetheless, their low dimensionality enables the generation of informative visual representations in 2D/3D phase plots. We briefly review the environments used in this section.

Parabolic Attractor, adopted from [**Tu et al., 2014, Brunton et al., 2016**], is a dynamical system with a single fixed point at the origin, known for its closed-form Koopman embedding solution. Governed by the following equations:

$$\dot{x}_1 = \mu x_1, \quad \dot{x}_2 = \lambda(x_2 - x_1^2), \tag{1.4.1}$$

the system admits a solution that is asymptotically attracted to the parabolic manifold, given by $x_2 = x_1^2$, for $\lambda < \mu < 0$. The Koopman embedding, z, that adheres to globally linear dynamics, can be coded by augmenting the state with the additional nonlinear measurement of $z_3 = x_1^2$:

$$\dot{z} = \begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \\ \dot{z}_3 \end{bmatrix} = \begin{bmatrix} \mu & 0 & 0 \\ 0 & \lambda & -\lambda \\ 0 & 0 & 2\mu \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} \quad \text{for} \quad \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \end{bmatrix} \tag{1.4.2}$$

We set $\lambda = -1.0$ and $\mu = -0.1$ and we sample initial conditions uniformly from x_1,$x_2 \in [-1, 1]$.

Duffing Oscillator follows a nonlinear second order differential equation $\ddot{x} = x - x^3$, which represents a model for the motion of a damped and force-driven particle. This particular instance admits two stable fixed points at $(x, \dot{x}) = (\pm 1, 0)$, and an unstable fixed point at the origin, $(x, \dot{x}) = (0, 0)$. Initial conditions are sampled uniformly from $x_1 \in [-2, 2]$ and $x_2 \in [-1, 1]$.

Lotka-Volterra represents the population evolution of biological systems, based on a predator-prey interactions, by the following equations:

$$\dot{x}_1 = \alpha x_1 - \beta x_1 x_2, \quad \dot{x}_2 = \delta x_1 x_2 - \gamma x_2. \tag{1.4.3}$$

The system is known for its abrupt switch in population growth and admits two fixed points, one at the origin (extinction), and a center point at $(x_1, x_2) = (\gamma/\delta, \alpha/\beta)$. We set $\alpha = \beta = \gamma = \delta = 0.2$ and uniformly sample initial conditions from $x_1, x_2 \in [0.02, 3.0]$.

Pendulum represents a freely swinging pole. The initial conditions indicate the states from which the pole is released, deviating slightly from the inverted position by $\pm 10°$. The state consists of the angle and the angular velocity and we report errors in radians.

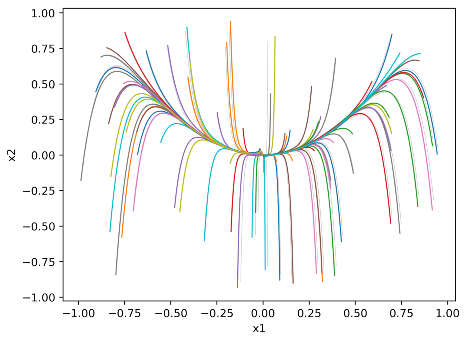
Lorenz System is a chaotic dynamical system [**Lorenz, 1963**]. It features equilibrium points, some stable and some unstable, and is renowned for the "butterfly effect" arising from its sensitivity to initial conditions. The governing equations are as follows:

$$\dot{x}_1 = \sigma(x_2 - x_1), \quad \dot{x}_2 = x_1(\rho - x_3) - x_2, \quad \dot{x}_3 = x_1 x_2 - \beta x_3 \tag{1.4.4}$$

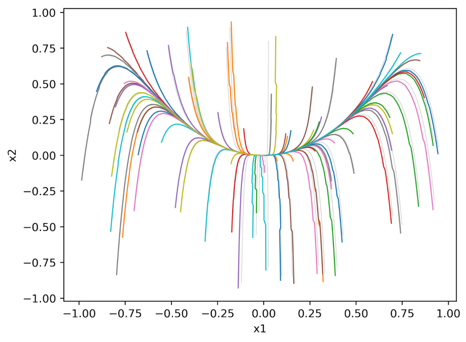
We use the original parameters from Lorenz'63 system. Initial conditions are generated by perturbing the point $(0, 1, 1.05)$ with Gaussian-distributed noise having a standard deviation of 1.

To demonstrate data-efficiency, we applied our proposed Koopman Autoencoder to modest datasets of trajectories gathered from each of the aforementioned dynamical systems, comprising 100 trajectories for Lorenz systems and 50 trajectories for non-chaotic ones. For all dynamical systems except Lorenz'63, the model is trained using the first 500 steps of the trajectories. Nevertheless, during inference, we unroll the models for up to 1000 steps, demonstrating the capability of our approach to accurately capture the underlying dynamics and generalize to unseen regions of the state space. We employed timesteps of 0.01 for forward integration in all environments, with the exception of Lorenz, for which we used a timestep of 0.02.

Results are presented in Table 1.1. We conducted experiments involving both linear and nonlinear decoders, linear and nonlinear latent dynamics, as well as experiments with and without periodic reencoding. In the nonlinear latent dynamics setting we use an MLP, instead of K, to drive the latent state forward. We maintain consistent encoder and decoder



((**a**)) Without reencoding.



((**b**)) With reencoding.

Fig. 1.3. Parabolic Attractor. Phase plots are generated by unrolling the model both with and without Periodic Reencoding. The phase plots are not noticeably different and this is as expected, due to the existence of closed-from Koopman codes for this particular environment.

capacities across different models applied to the same environment. We train a standard MLP for single step dynamics prediction as baseline, with capacity roughly equivalent to that of the encoders.

Periodic reencoding consistently improves the accuracy of the predictions. These improvements also extend to the setting where we permit nonlinear dynamics in the latent space, even though our method was motivated by learning linear latent dynamics. The only exception is the Parabolic Attractor environment (see Figure 3(a)). This is expected because this system admits a simple feature transformation that achieves *globally* linear dynamics. The representation can be decoded, with a linear decoder (Equation 1.4.2) by simply selecting the first two elements, without the need for course correction. Furthermore, it should be noted that the best prediction results use a linear decoder, rather than a nonlinear one. This observation holds true, regardless of the type of dynamics assumed in the latent space or the specific environment. The results also demonstrate that Koopman autoencoders with linear dynamics and decoders, unrolled using periodic reencoding, consistently outperform widely used nonlinear dynamics models.

1.4.2. D4RL: State Prediction

In this section, we train our proposed Koopman Autoencoder on continuous control tasks from D4RL benchmark by [**Fu et al., 2021**]. Again, the goal is to predict the future states over long time horizons. We choose a number of locomotion tasks as our main testbed, namely the `Hopper-v2`, `HalfCheetah-v2`, and `Walker2d-v2` environments. The curated datasets are generated by driving the simulated robots forward from a stationary position, using policies with varying degrees of optimality, i.e. `expert`, `medium-expert`, `medium`, and `medium-replay`.

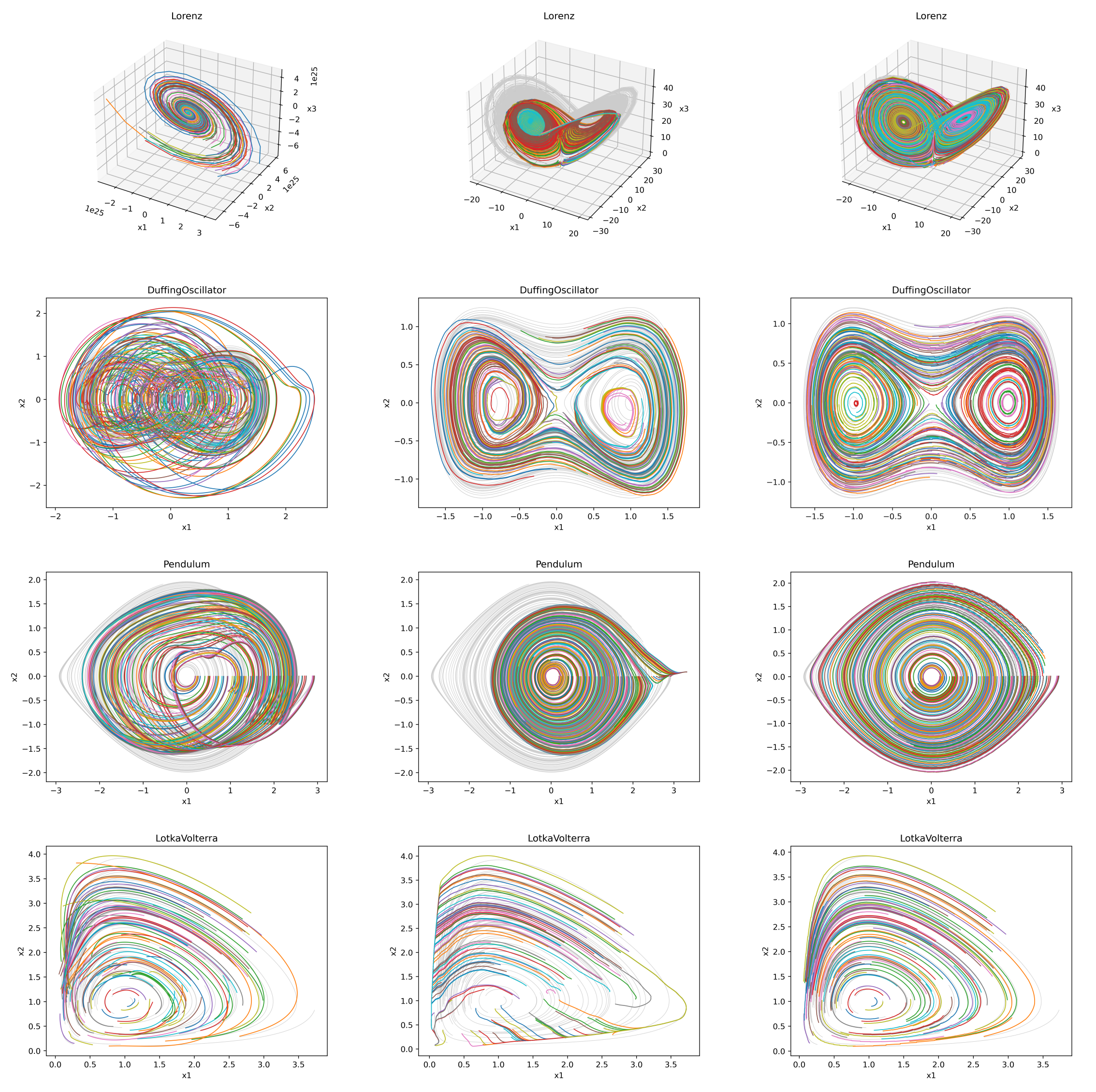
Model	Koopman (Linear Latent Dynamics)				NonLinear Latent Dynamics				MLP
Decoder Type	Linear		NonLinear		Linear		NonLinear		-
Periodic Reenc. (✗, ✓)	✗	✓	✗	✓	✗	✓	✗	✓	-
Environment	*MSE over **100** steps*								
Parabolic Attractor	**0.0205**	**0.0292**	0.1465	0.0758	0.0739	0.0496	0.0727	0.0547	0.2674
Pendulum	0.0512	0.0042	0.0648	0.0181	0.0288	**0.0025**	0.0242	0.0034	0.7442
Duffing Oscillator	0.1152	0.0112	0.1512	0.0512	0.0450	**0.0022**	0.0450	0.0112	0.4050
Lotka-Volterra	0.0113	0.0072	0.0145	0.0098	0.0128	**0.0040**	0.0112	0.0060	1.4450
Lorenz'63	✗	11.162	✗	12.569	18.985	**7.265**	19.051	**7.110**	88.565
Environment	*MSE over **1000** steps*								
Pendulum	9.2021	0.1818	14.5800	0.6612	10.7184	**0.0841**	10.5832	0.2964	55.281
Duffing Oscillator	20.5440	1.0658	15.0701	2.5312	5.7851	**0.5725**	9.2451	0.93845	22.445
Lotka-Volterra	1.6292	0.3961	1.6203	0.2812	0.7261	**0.2888**	0.7281	0.4324	83.205
Lorenz'63	✗	78.980	✗	64.838	✗	**59.262**	✗	**54.793**	133.509

Table 1.1. Mean Squared Error of state prediction for a number of dynamical systems. The entries in the table are scaled up by a factor of 100×, except for Lorenz system. Evaluation is done via sampling unseen initial points from the dynamical system. The cross marks and check marks in the Periodic Reenc. rows indicate, respectively, absence and presence of periodic reencoding mechanism during inference time. When enabled, the errors are reported by searching over reencoding periods of $(1, 10, 25, 50, 100)$. The cross marks in place of table entries indicate exploded values. Underlined values denote best performance for Koopman-based models, where bold numbers represent best performance across all models. We exclude the Parabolic Attractor environment when reporting errors over the 1000-step horizon since trajectories would almost perfectly merge onto the parabolic manifold and reach the origin in less time.

Each individual dataset comprises 1 million transitions, organized as trajectories with a maximum length of 1,000 steps. We use 80% of the trajectories for training and evaluate on the remainder 20%, consisting of trajectories of length 300. Table 1.2 provides a breakdown of the MSE over a 300-step horizon.

The D4RL benchmark differs from the dynamical systems discussed in the previous section in three fundamental ways: the dynamics are notably more challenging due to the presence of collisions, the dimensionality is higher, and the systems are subject to control inputs. Here we use the extended version of the Koopman autoencoder model designed for controlled systems (see Section 1.3.1). For training, we solely rely on the observations and avoid incorporating additional training signals, such as rewards.

Similar to the previous section, a multi-step MLP is trained as a baseline. To ensure a fair comparison, we set the size of the MLP to be equal to the combined size of the Koopman



((a)) No Reencoding **((b))** Reencode every step **((c))** Periodic Reencoding

Fig. 1.4. Phase portraits of the trained Koopman Autoencoders (w/linear decoder) for unrolls of 1000 steps. We use different different reencoding schemes for unrolling the same model, namely (a) w/o reencoding, (b) reencoding at every step, and (c) periodic reencoding (our proposed approach). The grey lines in the background represent ground truth phase portraits.

encoder and decoder. In our implementation, this equivalent MLP is implemented by *setting the reencoding period to 1 during training* and deactivating the loss terms associated with the Koopman autoencoder, specifically the “alignment” and “reconstruction” losses, *optimizing only the “prediction” loss* as the objective (see Figure 1.1). Refer to Appendix 1.5 for details and specifics.

We utilize training sequences with a length of 100. The results presented in Table 1.2 demonstrate that through periodic reencoding, we can consistently unroll our model over

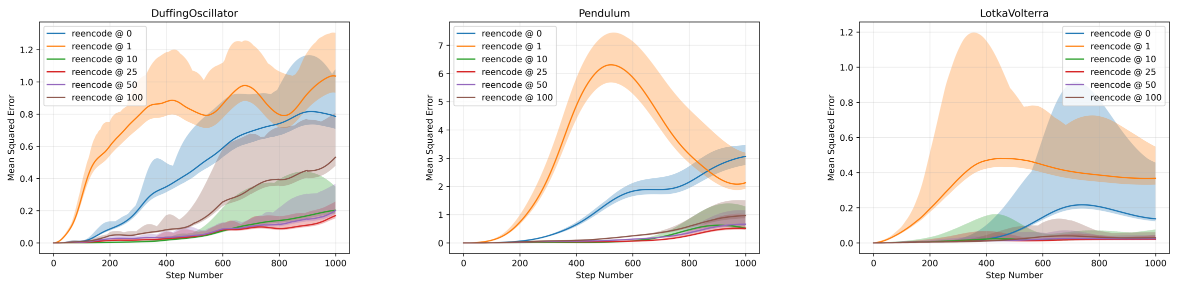


Fig. 1.5. Mean Squared Error (MSE) over the unrolling horizon, under varying reencoding schemes. We use 100 freshly sampled (unseen) initial points for evaluation. `` `reencode @ 0` `` legend indicates unrolling without ever reencoding. We observe that Periodic Reencoding robustly improves the quality of long-range modeling of dynamics.

extended horizons, surpassing the training length (in this case, over a 300-step horizon). Furthermore, the use of a nonlinear decoder proves to be crucial for accurate prediction. In our experience, training the MLP becomes unstable for horizons longer than 10 steps, and its performance is significantly inferior. However, it fulfills its role as a baseline.

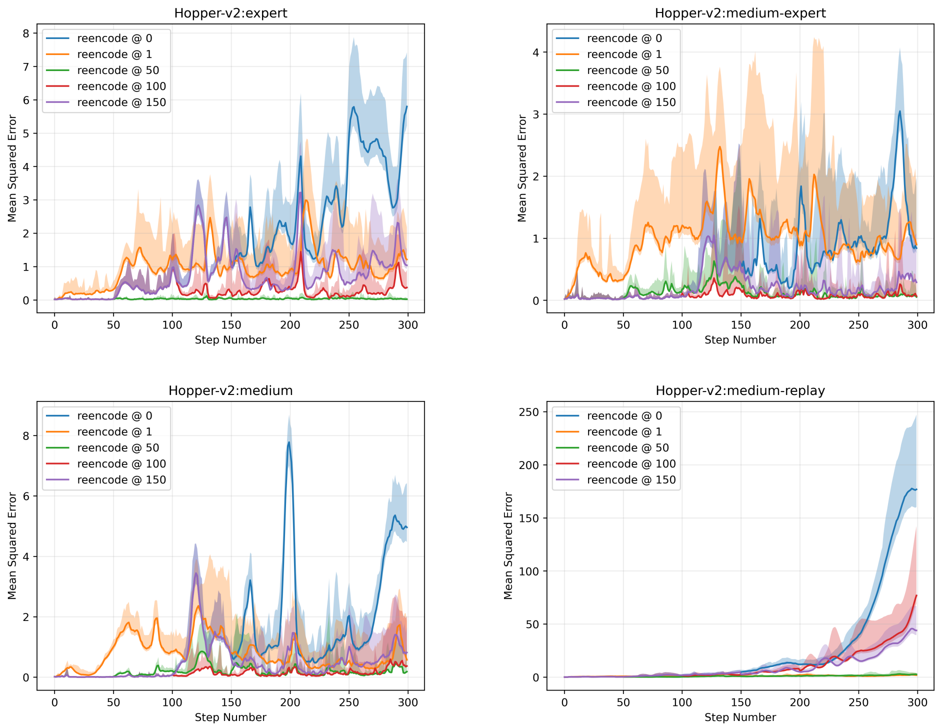


Fig. 1.6. Mean Squared Error (MSE) for Hopper-v2 environment over the unrolling horizon of 300 steps, under varying reencoding schemes. Periodic Reencoding generates more accurate trajectories compared to no reencoding and every-step reencoding schemes.

Model		Koopman Autoencoder				MLP
Decoder Type		Linear		NonLinear		-
Periodic Reencoding (✗, ✓)		✗	✓	✗	✓	-
Environment	Dataset	*MSE over **300** steps*				
Hopper$^{\text{v2}}$	expert	0.250 ± 0.05	0.079 ± 0.03	0.353 ± 0.05	$\mathbf{0.012 \pm 0.00}$	0.561
	medium-expert	0.486 ± 0.07	0.102 ± 0.04	0.719 ± 0.10	$\mathbf{0.015 \pm 0.00}$	0.592
	medium	0.624 ± 0.05	0.102 ± 0.03	0.889 ± 0.08	$\mathbf{0.008 \pm 0.00}$	0.533
	full-replay	1.052 ± 0.12	0.570 ± 0.11	2.254 ± 0.14	$\mathbf{0.158 \pm 0.01}$	0.815
	medium-replay	1.190 ± 0.22	0.776 ± 0.18	✗	$\mathbf{0.296 \pm 0.07}$	0.936
HalfCheetah$^{\text{v2}}$	expert	0.645 ± 0.05	0.602 ± 0.09	0.622 ± 0.07	$\mathbf{0.227 \pm 0.03}$	1.359
	medium-expert	1.141 ± 0.05	1.076 ± 0.08	0.813 ± 0.12	$\mathbf{0.391 \pm 0.07}$	1.481
	medium	1.362 ± 0.04	1.456 ± 0.08	1.816 ± 0.16	$\mathbf{0.809 \pm 0.09}$	1.861
	full-replay	1.262 ± 0.10	1.252 ± 0.17	1.668 ± 0.17	$\mathbf{0.816 \pm 0.14}$	1.994
Walker2d$^{\text{v2}}$	expert	0.364 ± 0.07	0.302 ± 0.08	0.544 ± 0.05	$\mathbf{0.072 \pm 0.03}$	0.285
	medium-expert	0.755 ± 0.02	0.602 ± 0.08	0.796 ± 0.10	$\mathbf{0.198 \pm 0.08}$	1.295
	medium	0.825 ± 0.08	0.718 ± 0.11	1.822 ± 0.18	$\mathbf{0.404 \pm 0.13}$	0.821
	full-replay	1.676 ± 0.19	1.413 ± 0.13	✗	$\mathbf{0.867 \pm 0.15}$	1.291
	medium-replay	✗	2.379 ± 0.20	✗	$\mathbf{2.077 \pm 0.26}$	1.917

Table 1.2. Mean Squared Error of state prediction for D4RL robotic locomotion tasks. Evaluation is done under a held-out set of trajectories. The cross marks in place of table entries indicate exploded or almost exploded values (large errors).

1.4.3. D4RL: Semi-Open-Loop Control

In this section, we utilize our proposed model as an open-loop controller for locomotion tasks within the D4RL framework to showcase long-term stability, generalization capability, and prediction quality. This approach allows us to present another informative metric, the total reward achieved by a semi-open-loop controller (with sparse feedback). In this context, the model is trained using transitions generated by an optimally trained policy, specifically the `expert` datasets.

The objective is to produce the sequence of states and actions that follow the initial state which is provided as input to the model. We train a Koopman autoencoder, jointly with an independent decoder head trained using a behaviour cloning loss to output actions, which receives the encoded states as input. We assess the quality of the predictions by evaluating the optimality of the replicated "expert" behavior. This is done by executing the generated sequence of actions in the environments and recording the total reward achieved. To underscore the stability of our approach, *we ensure that the model plans the motion of the robot for the next 100 steps*, by generating a sequence of actions before receiving any sort of feedback form the environment. By removing feedback at test time, i.e. by providing the ground truth state as input to the model only every 100 steps, we demonstrate the model's capability to generate stable motion plans, particularly when periodic reencoding is utilized.

Model		Koopman Autoencoder		MLP	BC
State Feedback (✗, ✓)		✗	✗	✗	✓
Periodic Reencoding (✗, ✓)		✗	✓	-	-
Environment	Dataset	*total reward until termination*			
Hopper^{v2}	expert	18.40 ± 6.2	$\mathbf{53.5 \pm 14.5}$	18.9 ± 5.9	96.05 ± 4.3
HalfCheetah^{v2}	expert	15.06 ± 5.3	$\mathbf{64.2 \pm 12.9}$	19.5 ± 7.9	82.91 ± 5.9
Walker2d^{v2}	expert	18.46 ± 8.1	$\mathbf{61.9 \pm 14.2}$	11.4 ± 2.5	98.73 ± 1.5

Table 1.3. Semi-open-loop reward achieved. The total rewards are normalized according to D4RL random and expert scores. The Koopman Autoencoder uses a nonlinear decoder.

Table 1.3 provides the results for the designed open-loop control problem. As baseline, we train an MLP that takes the state as input and outputs the next state along with the optimal action. The MLP can then be unrolled autoregressively at test time similarly in an open-loop fashion. Furthermore, for comparison, we train a standard behavior cloning (BC) model that we run *without* state obfuscation, and allow to observe state at every step. This is the upper bound of performance of the policy run with state obfuscation.

We demonstrate that our approach is capable of making sufficiently accurate predictions well into the future when the state is withheld from the model over 100-step horizons. This is evident from the agent's ability to sustain its motion without falling, relying solely on preplanned motion.

1.5. Conclusion

Our study of Koopman autoencoders for modeling nonlinear dynamical systems lead us to explore various inference schemes for generating predictions over long horizons. We showed that generating trajectories exclusively in the latent space presents two potential difficulties: (i) the inability to capture switching behaviour between multiple fixed points, and (ii) violation of the existence and uniqueness theorem of initial value problems. We showed, through theory and experiment, that trajectories generated with reencoding do not suffer from these limitations. Finally we introduced periodic reencoding as a method that bridges the gap between no reencoding and reencoding at every step, and achieves the best results in practice.

Acknowledgments

We would like to thank Daniel Worrall for their careful review of the manuscript. We would also thank Peter Battaglia and Hugo Larochelle for their prompt approval of the work.

References

[Arjovsky et al., 2015] Arjovsky, M., Shah, A., and Bengio, Y. (2015). Unitary evolution recurrent neural networks. *CoRR*, abs/1511.06464.

[Azencot et al., 2020] Azencot, O., Erichson, N. B., Lin, V., and Mahoney, M. W. (2020). Forecasting sequential data using consistent koopman autoencoders.

[Bengio et al., 2015] Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. (2015). Scheduled sampling for sequence prediction with recurrent neural networks. *Advances in neural information processing systems*, 28.

[Bradbury et al., 2018] Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. (2018). JAX: composable transformations of Python+NumPy programs.

[Brunton et al., 2016] Brunton, S. L., Brunton, B. W., Proctor, J. L., and Kutz, J. N. (2016). Koopman invariant subspaces and finite linear representations of nonlinear dynamical systems for control. *PLOS ONE*, 11(2):e0150171.

[Brunton et al., 2021] Brunton, S. L., Budišić, M., Kaiser, E., and Kutz, J. N. (2021). Modern koopman theory for dynamical systems. *arXiv preprint arXiv:2102.12086*.

[Frion et al., 2023] Frion, A., Drumetz, L., Mura, M. D., Tochon, G., and Bey, A. A. E. (2023). Leveraging neural koopman operators to learn continuous representations of dynamical systems from scarce data.

[Fu et al., 2020] Fu, J., Kumar, A., Nachum, O., Tucker, G., and Levine, S. (2020). D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*.

[Fu et al., 2021] Fu, J., Kumar, A., Nachum, O., Tucker, G., and Levine, S. (2021). D4rl: Datasets for deep data-driven reinforcement learning.

[Gu et al., 2021a] Gu, A., Goel, K., and Ré, C. (2021a). Efficiently modeling long sequences with structured state spaces. *CoRR*, abs/2111.00396.

[Gu et al., 2021b] Gu, A., Johnson, I., Goel, K., Saab, K., Dao, T., Rudra, A., and Ré, C. (2021b). Combining recurrent, convolutional, and continuous-time models with linear state-space layers.

[Kalman, 1960] Kalman, R. (1960). On the general theory of control systems.

[Koopman, 1931] Koopman, B. O. (1931). Hamiltonian systems and transformation in hilbert space. *Proceedings of the National Academy of Sciences*, 17(5):315–318.

[Koopman and v. Neumann, 1932] Koopman, B. O. and v. Neumann, J. (1932). Dynamical systems of continuous spectra. *Proceedings of the National Academy of Sciences*, 18(3):255–263.

[Kutta, 1901] Kutta, M. W. (1901). Beitrag zur näherungsweisen integration totaler differentialgleichungen. *Zeitschrift für Mathematik und Physik*, 46(1):435–453.

[Lan and Mezić, 2013] Lan, Y. and Mezić, I. (2013). Linearization in the large of nonlinear systems and koopman operator spectrum. *Physica D: Nonlinear Phenomena*, 242(1):42–53.

[Lorenz, 1963] Lorenz, E. N. (1963). Deterministic Nonperiodic Flow. *Journal of Atmospheric Sciences*, 20(2):130–148.

[Loshchilov and Hutter, 2017] Loshchilov, I. and Hutter, F. (2017). Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101.

[Lusch et al., 2018] Lusch, B., Kutz, J. N., and Brunton, S. L. (2018). Deep learning for universal linear embeddings of nonlinear dynamics. *Nature Communications*, 9(1).

[Lyu et al., 2023] Lyu, X., Hu, H., Siriya, S., Pu, Y., and Chen, M. (2023). Task-oriented koopman-based control with contrastive encoder. In *7th Annual Conference on Robot Learning*.

[Martin and Cundy, 2018] Martin, E. and Cundy, C. (2018). Parallelizing linear recurrent neural nets over sequence length. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

[Mondal et al., 2023] Mondal, A. K., Panigrahi, S. S., Rajeswar, S., Siddiqi, K., and Ravanbakhsh, S. (2023). Efficient dynamics modeling in interactive environments with koopman theory.

[Monfared et al., 2021] Monfared, Z., Mikhaeil, J. M., and Durstewitz, D. (2021). How to train rnns on chaotic data? *CoRR*, abs/2110.07238.

[Orvieto et al., 2023] Orvieto, A., Smith, S. L., Gu, A., Fernando, A., Gulcehre, C., Pascanu, R., and De, S. (2023). Resurrecting recurrent neural networks for long sequences.

[Pearlmutter, 1990] Pearlmutter, B. A. (1990). Dynamic recurrent neural networks.

[Runge, 1895] Runge, C. (1895). Über die numerische auflösung von differentialgleichungen. *Mathematische Annalen*, 46(2):167–178.

[Schütte et al., 2016] Schütte, C., Koltai, P., and Klus, S. (2016). On the numerical approximation of the perron-frobenius and koopman operator. *Journal of Computational Dynamics*, 3(1):1–12.

[Smith et al., 2023] Smith, J. T. H., Warrington, A., and Linderman, S. W. (2023). Simplified state space layers for sequence modeling.

[Trischler and D'Eleuterio, 2016] Trischler, A. and D'Eleuterio, G. M. (2016). Synthesis of recurrent neural networks for dynamical system simulation.

[Tu et al., 2014] Tu, J. H., , Rowley, C. W., Luchtenburg, D. M., Brunton, S. L., and and, J. N. K. (2014). On dynamic mode decomposition: Theory and applications. *Journal of Computational Dynamics*, 1(2):391–421.

[Tustin, 1947] Tustin, A. (1947). A method of analysing the behaviour of linear systems in terms of time series. *Journal of the Institution of Electrical Engineers - Part IIA: Automatic Regulators and Servo Mechanisms*, 94:130–142(12).

[Vlachas et al., 2021] Vlachas, P. R., Arampatzis, G., Uhler, C., and Koumoutsakos, P. (2021). Multiscale simulations of complex systems by learning their effective dynamics.

[Williams et al., 2015a] Williams, M. O., Kevrekidis, I. G., and Rowley, C. W. (2015a). A data–driven approximation of the koopman operator: Extending dynamic mode decomposition. *Journal of Nonlinear Science*, 25(6):1307–1346.

[Williams et al., 2015b] Williams, M. O., Rowley, C. W., and Kevrekidis, I. G. (2015b). A kernel-based approach to data-driven koopman spectral analysis.

First Article Appendices

Koopman Theory Overview

Let us consider an autonomous dynamical system defined over an open set $\mathfrak{D}$ of $\mathbb{R}^n$ by a system of first order differential equations:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) \tag{1.5.1}$$

This system induces a flow $\Phi : \mathfrak{D} \times \mathbb{R} \times \mathbb{R} \to \mathfrak{D}$ which, for a given time t, an observed value of the state of the system $\mathbf{x}_t$ and a duration Δt, maps to the state of the system evolved by said duration

$$\Phi_{t \to t+\Delta t}(\mathbf{x}_t) = \mathbf{x}_t + \int_t^{t+\Delta t} \mathbf{f}(\mathbf{x}_\tau)\ \mathrm{d}\tau = \mathbf{x}_{t+\Delta t} \tag{1.5.2}$$

One of the challenges in modern dynamical system theory revolves around establishing transformations which maps such nonlinear dynamics into linear ones, as there exist myriad of tools and frameworks facilitating the study of linear systems, in contrast to their nonlinear counterparts.

In his seminal work [**Koopman, 1931**] proposed an alternative perspective of general nonlinear dynamics in which he considered *measurement functions* $\varphi : \mathfrak{D} \to \mathbb{R}$ s.t. $\varphi \in \mathfrak{L}^2$, the set of square-integrable functions w.r.t. Lebesgue's measure—*which constitutes a Hilbert space whose metric is determined by the inner product* $(\varphi, \psi) \mapsto \int_{\mathfrak{D}} \varphi(x)\psi^*(x)\mathrm{d}\mu(x)$. A *measurement function* maps the actual state of the system to the measurement or observable data to which one would have access. He introduced an operator $\mathcal{K}$ acting upon the set of measurement functions and which advances the measurement such that

$$\forall\ \varphi \in \mathfrak{L}^2(\mathfrak{D}),\ \forall \mathbf{x} \in \mathfrak{D},\ \forall t, s \in \mathbb{R}\ \text{ s.t. }\ t > s,\ \mathcal{K}_{s \to t}(\varphi)(\mathbf{x}) = \varphi\left(\Phi_{s \to t}(\mathbf{x})\right) \tag{1.5.3}$$

It is linear by definition, and the family of operator $\{\mathcal{K}_{0 \to t}\}_{t \in \mathbb{R}}$ constitute a one-parameter group in Hilbert space, which admits as an infinitesimal generator the *Lie derivative operator* $\mathcal{T}$ such that

$$\begin{aligned} \mathcal{L}(\varphi)(\mathbf{x}(t)) &= \lim_{\tau \to 0} \frac{\mathcal{K}_{t \to t+\tau}(\varphi)(\mathbf{x}(t)) - \varphi(\mathbf{x}(t))}{\tau} \\ &= \lim_{\tau \to 0} \frac{\varphi(\mathbf{x}(t+\tau)) - \varphi(\mathbf{x}(t))}{\tau} = \frac{\mathrm{d}}{\mathrm{d}t}\Big\{\varphi(\mathbf{x}(t))\Big\} = \left(\nabla\varphi\right)\left(\mathbf{x}(t)\right) \cdot \mathbf{f}(\mathbf{x}(t)) \end{aligned} \tag{1.5.4}$$

However, the resulting linear differential equation on measurement functions is of little use in practical settings, as it is of infinite dimension. To circumvent this, it is customary to resort to orthogonal decomposition of the Hilbert space of measurement functions $\mathfrak{L}^2$ into subspaces invariant by application of the Lie (and thus Koopman) operator [**?, ?**].

$$\mathfrak{L}^2 = \bigoplus_{k=1}^{\infty} \mathfrak{E}_k \tag{1.5.5}$$

By truncation, it possible to operate on an invariant subspace $\mathfrak{H} = \bigoplus_{k=1}^{q} \mathfrak{E}_k$, thus providing a finite-dimensional representation of the operator $\mathcal{T}$ (resp. $\mathcal{K}$), restricted to said subspace, into a matrix $\mathbf{T}$ (resp. $\mathbf{K}$) acting on the vector space $\mathbb{R}^q$. The restriction of a measurement function $\overline{\varphi}^{\mathfrak{H}}$ thus fulfils the linear dynamics

$$\frac{\mathrm{d}}{\mathrm{d}t}\left\{\overline{\varphi}^{\mathfrak{H}}(\mathbf{x}(t))\right\} = \mathbf{T}\ \overline{\varphi}^{\mathfrak{H}}(\mathbf{x}(t)) \tag{1.5.6}$$

or equivalently

$$\overline{\varphi}^{\mathfrak{H}}(\mathbf{x}(t)) = \exp\{\mathbf{T}\ (t - t_0)\} \cdot \overline{\varphi}^{\mathfrak{H}}(\mathbf{x}(t_0)) = \mathbf{K}^{(t-t_0)}\ \overline{\varphi}^{\mathfrak{H}}(\mathbf{x}(t_0)) \tag{1.5.7}$$

Implementation Details

State Prediction Tasks. To minimize the alignment loss, the encoder may generate latent codes with arbitrarily small values. We regard these solutions as degenerate, and to discourage this behavior, we normalize the weight columns of the decoder. Moreover, we employ AdamW optimizer by [**Loshchilov and Hutter, 2017**] with a slight weight decay of value 1e−4 and learning rate of 1e−4. We utilize a custom learning rate of 1e−5 for the dynamics components, which are either the Koopman matrices or 3-layer MLPs. This encourages the encoder to follow the dynamics prescribed by Koopman and stabilizes training. We use the continuous Koopman parameterization and apply bilinear descretization [**Tustin, 1947**] whenever control inputs are present. In the absence of control inputs, we allow the model to unroll by solving an initial value problem (IVP), using more sophisticated integrators implemented by JAX [**Bradbury et al., 2018**]. During both training and inference, we make heavy use of the `jax.experimental.ode.odeint()` which relies on an adaptive stepsize (Dormand-Prince) Runge-Kutta integrator [**Runge, 1895, Kutta, 1901**]. We find incorporating the prediction loss to be detrimental to the overall performance when training on the dynamical systems listed in Section 1.4.1. Inspired by the model described in Appendix 1.5, we use a small sparsity inducing L_1 loss, 1e−3, applied to the Koopman embeddings to encourage region-dedicated dynamics, and use ReLU activations for all experiments to ensure sparse activations. The encoders used for dynamical systems and D4RL are 4-layer and 6-layer standard MLPs, respectively. We use training sequence lengths of 10 and 100, for the dynamical systems and D4RL state prediction tasks, respectively. Moreover, we observe that using a reencoding training scheme of 50 steps is slightly beneficial for D4RL state prediction tasks, compared to training w/o reencoding. Interestingly, presence of control inputs in D4RL dataset makes the training more stable over longer horizons, compared to the dynamical systems we use as benchmark. This is because in the absence of control inputs, the latent state can be arbitrarily scaled up by the eigenvalues of the $\mathbf{K}$ matrix. We find that normalizing the observations and actions as a preprocessing step negatively impacts performance, especially for D4RL tasks. We choose an embedding size of 128 for all dynamical systems, 512 for D4RL states, and 128 for D4RL action embeddings. The δ stepsize is trained in log space, initialized from that of the environment.

D4RL: Semi-Open-Loop Control. In this subsection we review the implementation details that pertain specifically to the open-loop control task. Given that the actions in the `expert` datasets are generated via a fixed expert policy, which is a function of the states, denoted as $u_t = \pi(x_t)$, we can transform the dynamics from their original form, $x_{t+1} = \mathbf{F}(x_t, u_t)$, more compactly into, $x_{t+1} = \mathbf{F}(x_t, \pi(x_t)) = \mathbf{F}_\pi(x_t)$. Therefore, we employ the standard Koopman formulation, i.e. $z_{t+1} = \mathbf{K}z_t$, which does not involve exogenous controls (see Section 1.3.1). Due to the absence of controls as direct inputs to the model, dictated by the nature of this problem, the training process becomes unstable for long sequences, in comparison to the D4RL state prediction tasks. We train a separate decoder head for action prediction, which takes the encoded states as input. We utilize training sequences with a length of 20 steps, and likewise, employ a periodic reencoding scheme of 20 steps during test time. As previously mentioned, we restrict ourselves to observing the ground truth state values only at intervals of 100 steps, hence the designation "Semi-Open-Loop." Additionally, for improved prediction accuracy, we utilize precise adaptive-stepsize integrators implemented in JAX, i.e. `odeint()`, both during the training and inference stages.

Switching Dynamics

To explore another limitation of trajectories generated without reencoding, it is instructive to study the special case where $dim(z) > dim(x)$ and $\psi(z) = Wz$, i.e. the decoder is an over-complete dictionary, e.g. a linear *operator*. Therefore the trajectories generated without reencoding correspond to *a linear projection of the solution of a linear dynamical system, e.g.* $x(t) = We^{Kt}\phi(x_0)$. Such trajectories exhibit limited expressiveness; for instance, they cannot capture transitions between multiple distinct fixed points. For example, consider a scenario where $\mathbf{x} \in \mathbb{R}^2$ is governed by a NLDS with multiple fixed points, such as the Duffing Oscillator. Let $\mathbf{z} \in \mathbb{R}^4$. It is possible for z to represent multiple, distinct, attractors of x but not the switching behaviour between them that occurs in the Duffing oscillator for some initial conditions. Let these attractors be located in two regions of state space, R_1 and R_2.

Now, let $\mathbf{z} = [z_1, z_2, z_3, z_4]$ and:

$$\phi(x) = \{[z_1, z_2, 0, 0] \quad \text{if } \mathbf{x} \in R_1, \quad [0, 0, z_3, z_4] \quad \text{if } \mathbf{x} \in R_2\}. \tag{1.5.8}$$

These can be interpreted as sparse codes with distinct supports corresponding to R_1 and R_2, inferred using ϕ. Furthermore, if we let K be block diagonal then:

$$\dot{z} = \begin{bmatrix} K_1 & 0 \\ 0 & K_2 \end{bmatrix} z. \tag{1.5.9}$$

Phase space trajectories are synthesized by linearly combining the trajectories in the latent space.

$$x = \left\{ z_1 \begin{bmatrix} | \\ w_1 \\ | \end{bmatrix} + z_2 \begin{bmatrix} | \\ w_2 \\ | \end{bmatrix} \text{if } x \in R_1, \quad z_3 \begin{bmatrix} | \\ w_3 \\ | \end{bmatrix} + z_4 \begin{bmatrix} | \\ w_4 \\ | \end{bmatrix} \text{if } x \in R_2 \right\} \tag{1.5.10}$$

The dynamics of $\mathbf{x}$ can be approximated by two distinct linear systems. Furthermore, the columns of W can shift the origin to represent two distinct fixed points. Note that, without reencoding (Equation 1.3.7), the dynamics of x are *restricted to follow the same linear system, determine by the initial condition $z_0 = \phi(x_0)$, for all time.* In other words, this scheme does not allow for switching between linear systems, represented by K_1 and K_2, after $t = 0$. Indeed this is what is observed in Figure 1.4 (a), the dynamics captured without reencoding resemble the phase diagram of the superposition of several LDS. In contrast, trajectories generated with reencoding, have the capacity to switch between supports (non-zero elements of z) and therefore switch between the dynamics defined by K_1 and K_2. Indeed it was shown that non-linear systems can only be linearized within a basin of attraction of a fixed point [**Lan and Mezić, 2013**], implying that the best we can hope to achieve is to partition the phase space into regions each of which can be well approximated by LDS, as illustrated in the example above.

Efficiency

Producing full trajectories for T discrete points in a nonlinear dynamical system of the form $x_{t+1} = \mathbf{F}(x_t)$ requires the repeated recurrent application of the nonlinear transformation $\mathbf{F}$. For the Koopman Autoencoder, trajectories for T discrete points are first generated in the Koopman linear space $z_{t+1} = \mathbf{K}z_t + \mathbf{L}v_t$ and followed by a time-agnostic decoding step $x = \psi(z)$. The linear recurrence in the latent space allows for a highly parallelizable unrolling of the predicted sequence $\hat{z}$ using parallel scans [**Martin and Cundy, 2018**]. Compared to nonlinear recurrences, training time is greatly improved. Periodic reencoding does add a nonlinear step to the Koopman autoencoder recurrence. However, since we decode and reencode periodically every k steps, we can parallelize the predicted trajectories for k discrete points and still obtain much improved training efficiencies. Furthermore, considering that our method can provide accurate predictions over short horizons (where reencoding is not needed), it is valuable in situations requiring fast look-ahead capabilities, such as model predictive control (MPC) or n–step value function bootstrapping in Reeinforcement Learning.

Additional Results

Figure 1.7 show the error plots for `HalfCheetah-v2` and `Walker2d-v2` environements – `Hopper-v2` error plots can be found in the main text. For all D4RL experiments, during training time, we either do not use reencoding at all, or we use reencoding periods of 20 or 50 steps. The plots correspond to the best performing trained models reported in Table 1.2. Periodic Reencoding consistently achieves the best results and is robust to the reencoding period.

Figure 1.8, 1.9, and 1.10 shows the error plots of the Koopman Autoencoder models that DO NOT employ reencoding (periodic reencoding) during training time. We still observe drift in latent space despite the absence of periodic reencoding at training time. Employing periodic reencoding at test time can still mitigate the drift and generate stable, long unrolls.

Nonlinear Latent Dynamics for D4RL. Table 1.4 presents the results for a latent dynamics model which employs **nonlinear** transition functions. This model maps the current latent state and encoded action to the next latent state, denoted as $(z_t, \nu_t) \to z_{t+1}$, by processing the concatenated vector of encoded states and control inputs through a wide 3-layer MLP. The observations made in the linear scenario, i.e. the Koopman framework, are found to be extensible to nonlinear instances as well – periodic reencoding greatly improves the prediction quality over long horizons. Table 1.4 also reports the effect of periodic reencoding during training time in isolation – compare the columns which do and do not use periodic reencoding during training.

Experiments with Structured Koopman Matrix. Additionally, we attempted experiments with a diagonal Koopman transition matrix but were unsuccessful. We suspect that this lack of success may be due to additional requirements imposed by the Koopman linear space. Achieving complete disentanglement of features in this context might necessitate an exponentially larger latent state dimension. In an effort to attain training stability and mitigate the issue of scaling up and down the outputs in linear dynamics steps, we conducted experiments using a skew-symmetric matrix, denoted as K. Skew-symmetric transition matrices perform rotations without scaling the space. The experiments with the skew-symmetric matrix yielded unsuccessful results, similar to those with the diagonal

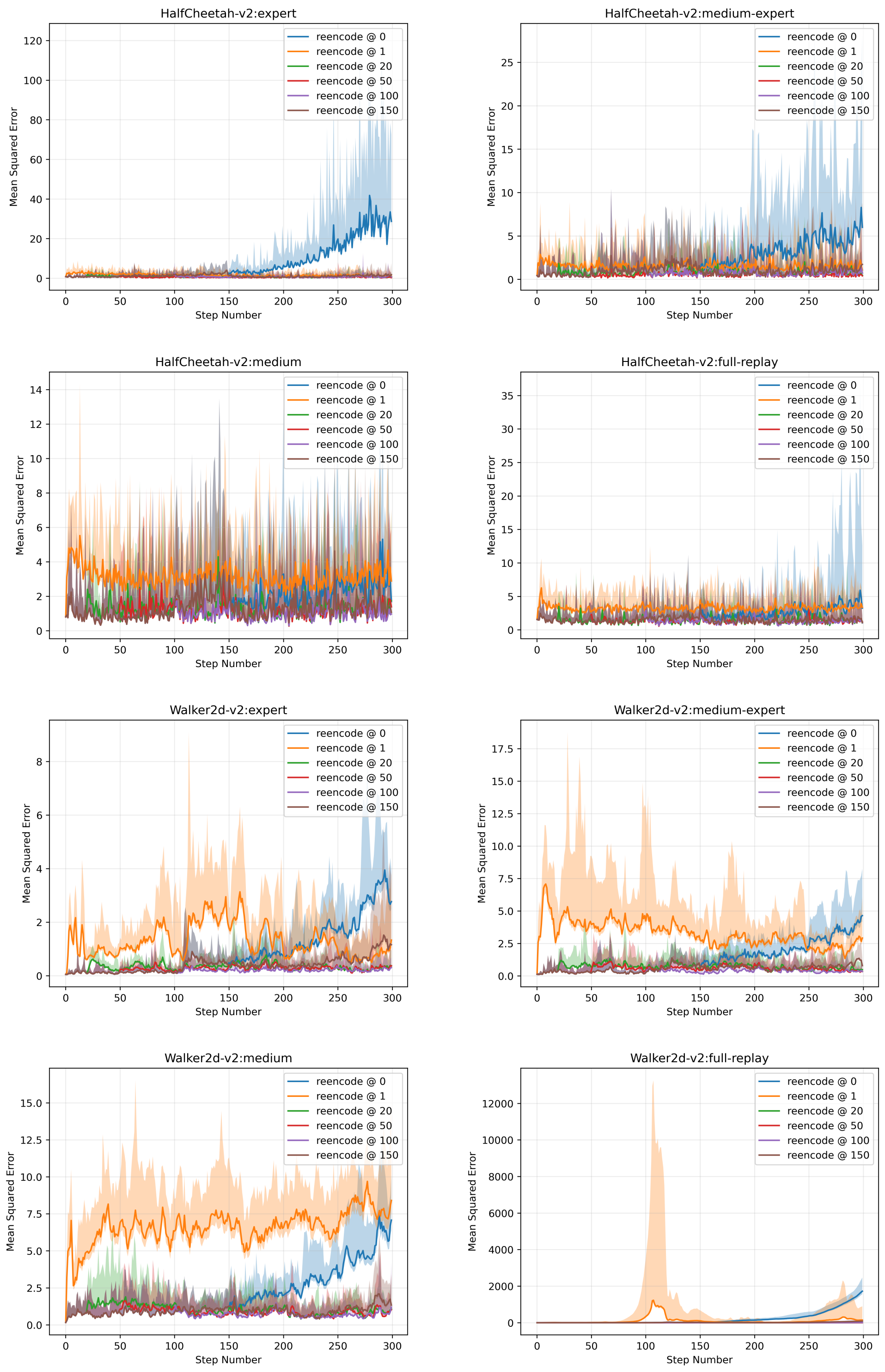


Fig. 1.7. Mean Squared Error (MSE) for HalfCheetah^{v2} and Walker2d^{v2} environments over the unrolling horizon of 300 steps, under varying reencoding schemes. The results are for Koopman Autoencoder models.

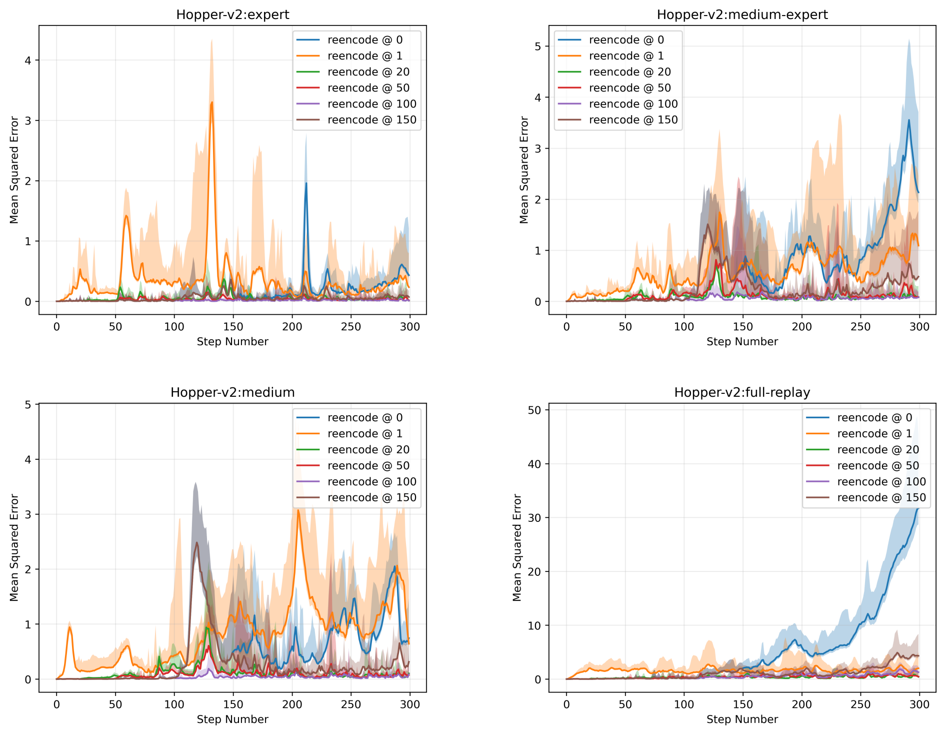


Fig. 1.8. Mean Squared Error (MSE) for `Hopper-v2` D4RL environment over the unrolling horizon of 300 steps, under varying reencoding schemes. The plots belong to models that <u>DO NOT</u> make use of reencoding during training. The errors reported in Table 1.2, under PERIODIC REENCODING: (✗), corresponds to the blue lines.

matrix. We speculate that this lack of success may be attributed to a lack of expressivity.

Low Dimensional Latents and Deep Latent-to-Latent Mappings. So far, we have assumed a high-dimensional latent space, where $dim(z) \gg dim(x)$, as in theory, Koopman operator is an infinite-dimensional operator. This enables us to learn rich representations of the dynamics, with the information bottleneck being the linearity requirement by Koopman. We conducted experiments using lower dimensional state embeddings, $dim(x) > dim(z)$, and deep latent-to-latent transition networks. In this context, the information bottleneck more closely resembles that found in tranditional, "static" autoencoders, where the projection into the latent space represents a form of compression. We observed extreme instability when unrolling over long horizons, resulting in poor prediction quality. We also discovered that Periodic Reencoding fails to improve the stability of these models.

Single Step Koopman Objective. Rather than training the Koopman objective over multiple steps, we focused on minimizing the losses as defined in Equation 1.3.4 at individual

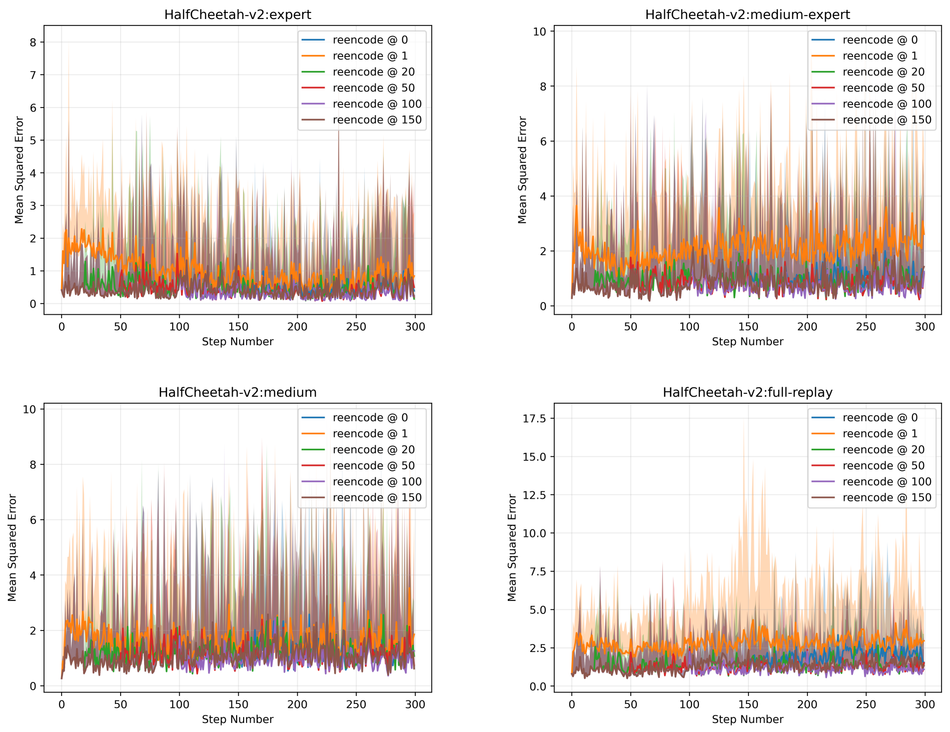


Fig. 1.9. Mean Squared Error (MSE) for D4RL `HalfCheetah-v2` environment over the unrolling horizon of 300 steps, under varying reencoding schemes, trained w/o reencoding.

timesteps. We implemented this approach with the aim of perfectly aligning the training objective with the inference time scenario, where reencoding occurs at every step. However, this resulted in high inference-time instability and subpar performance, even worse than the MLP baseline, regardless of the utilized inference mechanism, e.g. with or without reencoding. This suggests that the Koopman objective is indeed leveraging its extensibility within basins of attraction, as indicated by [**Lan and Mezić, 2013**].

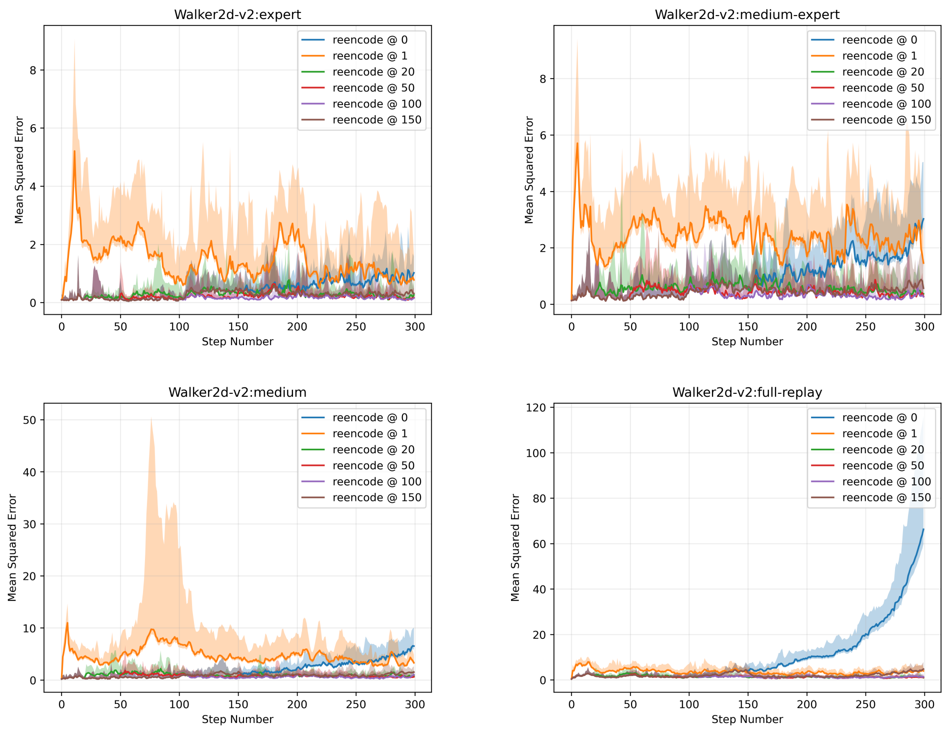


Fig. 1.10. Mean Squared Error (MSE) for D4RL `Walker2d-v2` environment over the unrolling horizon of 300 steps, under varying reencoding schemes, trained w/o reencoding.

Connection to RNNs

There exists an extensive body of literature on training RNNs on nonlinear dynamical systems [**Pearlmutter, 1990, Trischler and D'Eleuterio, 2016, Vlachas et al., 2021**]. Koopman autoencoders can be regarded as RNN models with a single linear recurrent layer. From an efficiency perspective, linear recurrent units are more advantageous as they can be executed in parallel rather than sequentially. Furthermore, from a representation learning perspective, one could argue that linearly evolving features extracted from a NLDS would result in a more meaningful and interpretable representation of the system in question. State-space Models (SSMs) [**Gu et al., 2021a, Gu et al., 2021b**] can also be regarded as linear RNNs, achieving impressive performance in modeling long-range dependencies. A recent finding by [**Orvieto et al., 2023**] highlights that linear RNN layers surpass tuned nonlinear RNN variants in performance. This result is also partially attributable to the fact that the behavior of linear layers can be designed and engineered more effectively, thanks to the well-studied nature of this problem. This enables effective regulation of linear layers, ensuring the stability of the training process over long sequences. A similar approach was previously adopted by [**Arjovsky et al., 2015**], wherein the eigenvalues of the RNN transition matrix were constrained to lie on the unit circle. The drawback of imposing structure and constraints

MODEL		NONLINEAR LATENT DYNAMICS					
DECODER TYPE		LINEAR			NONLINEAR		
		PERIODIC REENCODING					
TRAINING (✗, ✓)		✗	✗	✓	✗	✗	✓
INFERENCE (✗, ✓)		✗	✓	✓	✗	✓	✓
Environment	Dataset	*MSE over **300** steps*					
Hopper^{v2}	expert	0.190	0.165	0.054	0.172	<u>0.011</u>	**0.009**
	medium-expert	0.533	0.287	0.093	0.607	<u>0.031</u>	**0.013**
	medium	0.668	0.349	0.085	0.428	<u>0.027</u>	**0.009**
	full-replay	1.043	0.698	0.425	2.178	<u>0.262</u>	**0.118**
	medium-replay	2.787	0.785	0.785	3.163	<u>0.412</u>	**0.328**
HalfCheetah^{v2}	expert	0.621	0.600	0.503	0.374	<u>0.265</u>	**0.142**
	medium-expert	1.189	1.141	1.061	0.809	<u>0.569</u>	**0.394**
	medium	1.538	1.412	1.380	1.213	<u>0.798</u>	**0.641**
	full-replay	1.457	1.177	1.494	1.376	<u>0.655</u>	**0.616**
Walker2d^{v2}	expert	0.388	0.322	0.307	0.377	<u>0.115</u>	**0.099**
	medium-expert	0.739	0.669	0.563	0.794	<u>0.230</u>	**0.191**
	medium	0.796	0.920	0.782	1.202	<u>0.419</u>	**0.162**
	full-replay	1.500	1.509	1.415	✗	<u>0.767</u>	**0.507**
	medium-replay	✗	2.479	2.519	✗	**2.163**	<u>2.202</u>

Table 1.4. Mean Squared Error of state prediction for D4RL locomotion tasks when <u>nonlinear</u> latent dynamics are allowed. The results show further improvements over the linear case when periodic reencoding is employed.

on the transition matrices is that it results in a loss of expressivity [**Monfared et al., 2021**]. By introducing periodic reencoding, we eliminate the necessity of training on extended sequences altogether (for fully-observable systems) and avoid the requirement to impose any form of structure onto the transition matrices. An interesting follow-up work could involve reformulating periodic reencoding for RNNs with multiple recurrent layers.

Chapter 2 Second article

Block-State Transformers

by

Mahan Fathi*[1], Jonathan Pilault*[2], Orhan Firat[3],
Christopher Pal[4], Pierre-Luc Bacon[5], and Ross Goroshin[6]

([1]) Google DeepMind, Mila, Université de Montréal
([2]) Google DeepMind, Mila, Polytechnique Montréal
([3]) Google DeepMind
([4]) Mila, Polytechnique Montréal, CIFAR
([5]) Mila, Université de Montréal, CIFAR
([6]) Google DeepMind

This article was submitted in Conference on Neural Information Processing Systems 2023 – (NeurIPS'23).

The main contributions of Mahan Fathi for this articles are presented. Jonathan and I, Mahan Fathi, shared equal responsibilities and contributions to this endeavor. We jointly led the project, partaking in coding, overseeing experiments, and handling logistics. To delineate further, I predominantly took charge of coding and manuscript composition, while Jonathan led the majority of the experimental runs. Ross and Pierre-Luc provided supervision for the research. Ross contributed greatly to manuscript refinement.

Résumé. Les modèles d'espace d'état (State Space Models, SSM) ont montré des résultats impressionnants sur des tâches nécessitant la modélisation de dépendances à longue portée et s'échelonnent efficacement sur de longues séquences grâce à leur complexité temporelle sous-quadratique. À l'origine conçus pour les signaux continus, les SSM ont affiché des performances supérieures sur une multitude de tâches en vision et en audio ; cependant, les SSM accusent encore un retard par rapport aux performances des Transformers dans les tâches de modélisation linguistique. Dans ce travail, nous proposons une couche hybride nommée "Block-State Transformer" (BST), qui combine internement une sous-couche SSM pour la contextualisation à longue portée et une sous-couche Transformer par bloc pour la représentation à court terme des séquences. Nous étudions trois variantes différentes, totalement parallélisables, intégrant des SSM et une attention par bloc. Nous montrons que notre modèle surpasse des architectures similaires basées sur les Transformers en termes de perplexité en modélisation linguistique et généralise à des séquences plus longues. De plus, le Block-State Transformer présente une augmentation de vitesse au niveau de la couche de plus de *dix fois* par rapport au Block-Recurrent Transformer lorsque la parallélisation du modèle est utilisée.
Mots clés : Grand Modélisation Linguistique, Transformers, Modèles d'Espace d'État, Modélisation Linguistique à Longue Portée

Abstract. State space models (SSMs) have shown impressive results on tasks that require modeling long-range dependencies and efficiently scale to long sequences owing to their subquadratic runtime complexity. Originally designed for continuous signals, SSMs have shown superior performance on a plethora of tasks, in vision and audio; however, SSMs still lag Transformer performance in Language Modeling tasks. In this work, we propose a hybrid layer named "Block-State Transformer" (BST), that internally combines an SSM sublayer for long-range contextualization, and a Block Transformer sublayer for short-term representation of sequences. We study three different, and completely parallelizable, variants that integrate SSMs and block-wise attention. We show that our model outperforms similar Transformer-based architectures on language modeling perplexity and generalizes to longer sequences. In addition, the Block-State Transformer demonstrates more than *tenfold* increase in speed at the layer level compared to the Block-Recurrent Transformer when model parallelization is employed.
Keywords: Large Language Modeling, Transformers, State-Space Models, Long-range Language Modeling

2.1. Introduction

Transformers have shown impressive performance on a wide range of natural language processing (NLP) tasks. While they have been primarily used for language modeling the Transformer architecture [**Vaswani et al., 2017**] has also been successfully applied to other tasks outside of the NLP and have mostly replaced Recurrent Neural Networks (RNNs). Several factors contribute to this success, including computational efficiency and architectural inductive biases that are well-suited for training on natural language tasks at scale. On the computational upside, Transformers are able to process tokens of a given input sequence in parallel, making the most of modern accelerator hardware. Moreover, the attention mechanism enables Transformers to find relationships in longer sequences by providing ready access to all the extracted information from past tokens when inferring the next token. Compared to RNNs and LSTMs [**Hochreiter and Schmidhuber, 1997**], the benefits of self-attention are two-fold: (i) the capacity of what could be stored and directly accessible as context is drastically increased, and (ii) training on longer sequences is more stable [**Hochreiter, 1998, Khandelwal et al., 2018**].

Given the remarkable achievements of Transformers in language modeling tasks, and their improved performance at scale on hard NLP tasks such as reasoning and question answering [**Brown et al., 2020, Thoppilan et al., 2022, Chowdhery et al., 2022**], the demand for deploying even deeper and larger networks is greater than ever before. An orthogonal scaling dimension, which could be potentially even more consequential, is the size of the input sequence. Despite the several advantages of Transformers over RNNs, it is still problematic to scale the input sequence length, again for both computational performance and quality reasons. Further, the Transformer's runtime is quadratic with respect to the input sequence length, which makes training these models increasingly expensive. Furthermore, Transformers with attention, that is local [**Dai et al., 2019**], sparse [**Child et al., 2019, Zaheer et al., 2020, Tay et al., 2020a**], low-rank approximated [**Wang et al., 2020**] or linearlized via kernel methods [**Choromanski et al., 2020, Katharopoulos et al., 2020**], notoriously struggle on long-input classification tasks [**Tay et al., 2020b**] and are highly unstable when trained on long sequences [**Zhang et al., 2022**].

A emerging body of research suggests that State Space Models (SSMs) can serve as an alternative to Transformers because they are able to capture dependencies in extremely long sequences, while being more computationally efficient and parallelizable [**Gu et al., 2022a**]. While still falling into the category of autoregressive sequence models, the underlying linear time-invariant dynamical system of SSMs allows the efficient processing of sequences using parallelizable convolution operators with the Fast Fourier Transform (FFT) [**Cooley and Tukey, 1965**], with $\mathcal{O}(L \log L)$ complexity, where L is the length of the sequence. Moreover, retention of past information over long sequences, up to thousands of steps, can be ensured by deriving recurrent update rules by borrowing ideas from online function approximation [**Chihara, 2011, Gu et al., 2020**]. SSMs have recently outperformed Transformers on long-range dependency benchmarks by a large margin [**Tay et al., 2020b**]. Despite their success on long-range classification tasks, SSMs have not yet completely matched Transformers as an off-the-shelf sequence model for general language modeling tasks [**Fu et al., 2023a**].

Recent findings suggest that Transformers and SSMs are complementary models for the purpose of language modeling [**Mehta et al., 2023**]. In this work, we propose an architecture that integrates a strong local attention-based inductive bias with the long-term context modeling abilities of SSMs into a single layer, that we call *Block-State Transformer* (BST). Our model is able to process long input sequences, while still incorporating an attention mechanism to predict next tokens. BST is fully parallelizable, scales to much longer sequences, and offers a 10× speedup compared to comparable Transformer-based layers.

In every BST layer, an SSM takes the entire sequence as input and maps it into a "context" sequence of the same length. The SSM sublayer takes advantage of FFT-based convolutions. This sequence of context is then divided into blocks of equal size, i.e. window length (W), and each context block is then fed to a Block Transformer layer, that attends to the subsequences of size W as defined in [**Hutchins et al., 2022**]. The block of input token embeddings are then cross-attended to the corresponding block of context states; see Figure 2.1. Note that by introducing SSMs as a means of contextualization, we completely remove the need for sequential recurrences and we are able to run our hybrid SSM-Transformer layer fully in parallel. The resulting runtime complexity can be expressed as the sum of $\mathcal{O}(W^2) + \mathcal{O}(L \log L)$, where the first term represents the time complexity of the Transformer sublayer, while the second term represents the time complexity of the SSM sublayer. This is a major improvement over $\mathcal{O}(LW)$ of Block-Recurrent Transformer, so long as hardware to support parallel computation is available. Moreover, due to hardware imposed restrictions, the runtime complexity of the SSM on a full sequence is comparable to that of Block Transformer on a block of tokens, which further implies the absence of a speed bottleneck in the BST layer, empirically validated for sequences containing hundreds of thousand of tokens. This is evident by observing that the bottom-most two lines on the left of Figure 2.5 are almost overlapping.

2.2. Related Work

This work is primarily related to two branches of recent research: (i) combining local attention with recurrent networks in order to extend their capacity to capture long-range

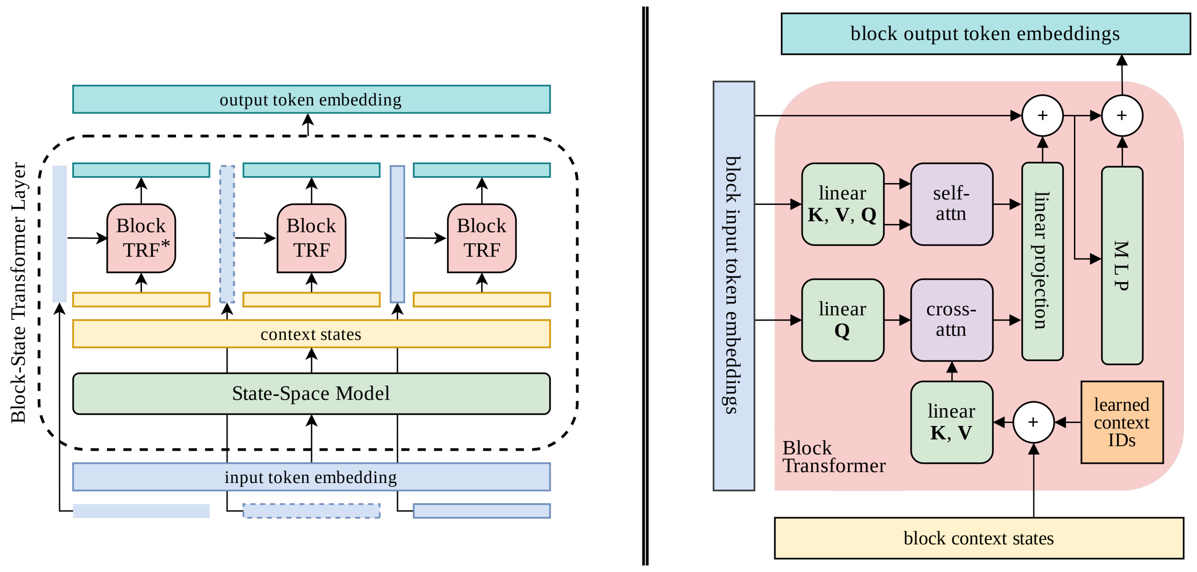


Fig. 2.1. Block-State Transformer layer. The BST-SH layer is illustrated on the left, and includes a state space model (SSM, in green) and Block Transformers (in red). For demonstration purposes the sequence is divided into 3 blocks in the picture. The details of the Block Transformer sublayer are on the right. *TRF = Transformer.

dependencies, beyond the length of the attention window size, and (ii) State Space Models (SSMs) which describe sequences via linear dynamical systems whose outputs can be computed in parallel. Block-Recurrent Transformer (BRecT) [**Hutchins et al., 2022**] uses a recurrent memory mechanism to extend the theoretical context length of the Transformer. In the recurrent unit of the BRecT cell, the updates made to the "recurrent state vectors," are extracted by employing a cross-attention mechanism over a block/window of input token embeddings. Different from their work, we use linear state space models instead of recurrent cells to maintain context states. We also conduct a more extensive exploration of maintaining and updating context states. Earlier works augment transformers with a non-differentiable external memory include the Memorizing Transformer [**Wu et al., 2022**]. Transformer-XL [**Dai et al., 2019**] was an early work that combined recurrent memory with Transformers. Our work can be seen as a continued evolution of those models incorporating state-of-the-art recurrent memory models inspired by SSMs.

State space models can be considered as linear RNNs [**Gu et al., 2020**]. This simplicity facilitates their analysis and even enables analytical derivation of recurrent weights for optimally representing arbitrarily long sequences. The linear property also allows the recurrence to be unrolled and parallelized during training and inference [**Gu et al., 2022a**]. Our work combines these state-of-the art models, enabling Transformers to leverage theoretically infinite context.

Other works have attempted to replace Transformers, and their attention mechanism with SSMs [**Mehta et al., 2023, Ma et al., 2023, Fu et al., 2023a, Poli et al., 2023**], however despite recent progress, the performance achieved by the Transformer architecture remains unparalleled in language. Nevertheless, SSMs are able to capture longer range dependencies than Transformers in both theory and practice, while also being highly parallelizable [**Cooley and Tukey, 1965, Fu et al., 2023b**]. We therefore elect to combine the best aspects of SSMs and Transformers into a single model. The idea of communication across blocks, similar to GSS [**Mehta et al., 2023**], was later implemented by MEGA [**Ma et al., 2023**], through an Exponentially Moving Average (EMA) update rule instead of SSMs[1]. However, both GSS and MEGA use a single-head Gated Attention Unit (GAU) [**Hua et al., 2022**]. MEGA further mixes layer inputs, GAU outputs and EMA outputs via two gating mechanisms. Our method uses a simpler architecture to mix signals from local attention and SSM outputs via cross-attention, allowing us to plug any out-of-the-box SSMs or attention layers. Further, we investigate three ways to mix SSM signals with attention as outlined in Section 2.3.3.

2.3. Method

We consider the problem of next token prediction via a decoder-only language model. This seemingly simple pretext task has led to spectacular progress in language understanding [**Devlin et al., 2018, Brown et al., 2020, OpenAI, 2023**]. During training, the decoder takes in a sequence of length L of tokens embeddings and is tasked to generate the next token at every step in the sequence.

We start by a brief review of SSMs that are essential for understanding the Block-State Transformer layer (2.3.1). Our full Block-State Transformer architecture is outlined in Section 2.3.2. Section 2.3.3 describes three approaches for integrating SSM states into the attention mechanism. Important implementation details are described in Section 2.3.4.

2.3.1. State Space Preliminaries

State space models can be divided into two categories:

State Spaces: Structured Kernels. S4 [**Gu et al., 2022a**], S5 [**Smith et al., 2023**], S4D [**Gu et al., 2022b**], DSS [**Gupta et al., 2022**], follow a structured initialization of the convolutional kernel by unrolling a linear time-invariant (LTI) dynamical system of the

[1]The authors in [**Ma et al., 2023**] show a mathematical form of EMA that has a state transition and also derive a convolution kernel to efficiently compute EMA similarly to S4.

following form:

$$\begin{aligned} x_k &= \mathbf{A}x_{k-1} + \mathbf{B}u_k \ , \\ y_k &= \mathbf{C}x_k + \mathbf{D}u_k \ . \end{aligned} \tag{2.3.1}$$

The system is parameterized by a state matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$, vectors $\mathbf{B} \in \mathbb{R}^{N \times 1}$, $\mathbf{C} \in \mathbb{R}^{1 \times N}$, and $\mathbf{D} \in \mathbb{R}^{1 \times 1}$, the SSM maps a 1-D input signal u_k, to a 1-D output signal y_k. Internally, the SSM projects the input signal to an N-D representation state x_k, before mapping it down to a scalar using the $\mathbf{C}$ matrix. The term $\mathbf{D}u_k$ can be thought of as a skip connection and will be omitted for the remainder of the discussion for convenience. The output of the above recurrent equation, y_k, can be computed as a discrete convolution, by realizing that the recurrence can be explicitly unrolled:

$$\begin{aligned} \text{Let} \quad x_{-1} &:= \vec{0} \ , \\ y_k &= \sum_{j=0}^{k} \mathbf{C}\mathbf{A}^j\mathbf{B} \cdot u_{k-j} \ . \end{aligned} \tag{2.3.2}$$

The $\mathbf{C}\mathbf{A}^k\mathbf{B}$ entries are collected to create the SSM kernel $\mathbf{K} \in \mathbb{R}^L$, and the convolution could be expressed as:

$$\begin{aligned} \mathbf{K} &= (\mathbf{CB}, \mathbf{CAB}, \dots, \mathbf{CA}^{L-1}\mathbf{B}) \ , \\ y_k &= \sum_{j=0}^{k} \mathbf{K}_j \cdot u_{k-j} \ , \quad y \ = \ \mathbf{K} * u \ . \end{aligned} \tag{2.3.3}$$

Given an input sequence $u \in \mathbb{R}^L$, it is possible to compute the output $y \in \mathbb{R}^L$ sequentially through the recurrence in Equation (2.3.1). While this property is useful for autoregressive decoding, sequential computation is prohibitively slow to train with long inputs and, instead, the convolution from the Equation (2.3.3) can be used to compute all elements of y in parallel. This is done via Fast Fourier Transform (FFT) [**Cooley and Tukey, 1965**], provided we have already computed $\mathbf{K}$.

Additional inductive biases have been imposed on SSMs by *analytically deriving* closed-form expressions for the matrices $\mathbf{A}$ and $\mathbf{B}$ using the HiPPO framework [**Gu et al., 2020**]. In this framework, the state x_t represents the coefficients of polynomials that approximate the sequence u_t.

Explicitly Parameterized Filters. In contrast to structured kernels, one can parameterize the convolution kernel, as trainable weights and optimize them, $\bar{\mathbf{K}} \in \mathbb{R}^L$. However, this would result in poor performance unless certain types of regularization are applied to the kernel. [**Fu et al., 2023b**] simply makes use of squashing the kernel weights, and subsequently applying a smoothing technique. Trainable kernels are also used in attention-free alternative models to Transformers, such as Hyena [**Poli et al., 2023**], which involves exponentially decaying the weights along the kernel:

$$\bar{\mathbf{K}}_t \quad = \quad e^{-\alpha t} \cdot \big(\texttt{FFN} \circ \texttt{PositionalEncoding}\big)(t) \ , \tag{2.3.4}$$

where $\bar{\mathbf{K}}_t$ is an entry in the filter at location t, and `FFN` is a feed-forward network used for decoupling the parameter count from the seuqnece length.

2.3.2. Block-State Transformer (BST) Layer

We now introduce the Block-State Transformer layer, which combines SSMs with Block Transformers. At each training iteration, a sequence of L tokens, is sampled from a longer document. The tokens are then embedded and fed to the model. Our model consists of a stack of Block-State Transformer layers. Each BST layer optionally includes an SSM sublayer that is responsible for providing long-range context to the Block Transformer layer, which operate similarly to a Block-Recurrent Transformer (BRecT) cell. The SSM sublayer takes the sequence of token embeddings from the previous layer as input, and produces a sequence of the same length L as the output.

The output of the SSM is contextually encoded, meaning that entries at every time-step, potentially include information about all the time steps preceding elements in the sequence. We collect a number of "context states," S, from the context sequence, and we set $S \ll L$. In order to prevent the model from accessing future information, we only allow the model to access context states that precede the current token. Various ways to gather context states from the context sequence are discussed in section 2.3.3 in detail.

The context states are fed to the Block Transformer, in place of what was referred to as "recurrent state vectors" in Block-Recurrent Transformer [**Hutchins et al., 2022**]. The subsequent operations, shown on the right side of Figure 2.1, are kept unaltered, except that we no longer need to run the recurrent unit of the BRecT cell since we are maintaining the context via an SSM. In addition to the context states, the Block Transformer also receives a block/window of length W of token embeddings as input, which are cross-attended to the context states. The output of the cross-attention operation is then concatenated with that of self-attention over the input embeddings, followed by a simple projection.

In addition to the ability of SSMs to retain information over longer time horizons compared to Transformers and RNNs, using the SSM to maintain context states as a replacement for recurrent cells makes for a more computationally efficient layer. Removing recurrence by integrating SSMs into Transformer layers, allows the Block-State Transformer layer to be fully parallelizable, whereas the Block-Recurrent architecture processes blocks of tokens sequentially using a for-loop.

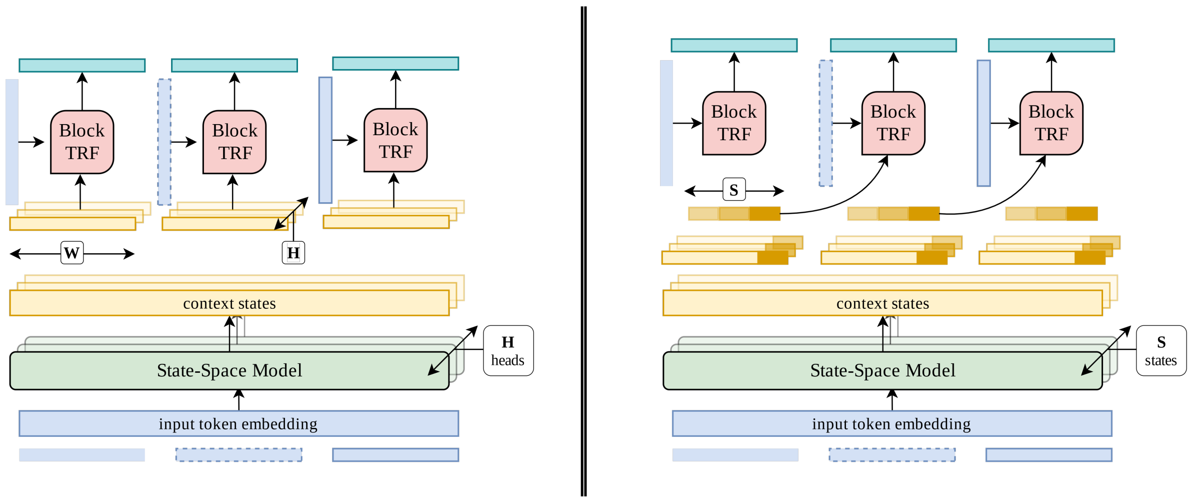


Fig. 2.2. Summarizing our approaches. The left side shows the cases where the SSM is required to output Multi-Head (**MH**) contexts. On the right Multi-Filter (**MF**) approach is depicted where the last entries from the previous window are concatenated into a set of context states of size S. Dashed lines represent the current block.

2.3.3. Context States

Although the latest SSM output technically contains information about the entire sequence, retrieving individual tokens from only the final state may not be feasible. To compensate, we concatenate a sequence of states, corresponding to the latest block of tokens. This is also analogous to the approach taken by BRecT. This representation ensures *retrievability* and ease of access, through *redundancy*. It is redundant because adjacent states are highly correlated, however this also makes it possible to easily recover the current block of tokens, if necessary.

In our approach, the context states are constructed from the output of the SSM and fed to the attention heads of the Transformer. These context states can be constructed in various ways. To guide these design decisions we consider each of the below proposed schemes as introducing *retrievability* at the cost of *redundancy*. The shape of the output of a single SSM layer is $(B \times L \times D)$, where B is the batch size, L is the number of the tokens processed, and D is the embedding dimension. When doing cross-attention in the Transformer cell with H different heads, this tensor needs to be transformed into a context tensor of shape $(B \times S \times D \times H)$, where S is the number of context states; we usually set $S \ll L$ and $S = W$ similar to Block-Recurrent Transformers (BRecT).

We now discuss the three different approaches that we evaluate to generate a context tensor for each block sequence:

SH: Single-Head. The first approach constructs the context tensor by sequentially concatenating the S states from the SSM with a single filter (each of size D). Note that because the SSM captures information from preceding blocks, the context state also captures information about blocks that preceded the current block. The resulting context vector is highly *retrievable* and *redundant*, as defined above. As in typical Transformers, fully connected layers are used to project each context vector to H different heads of size D. Note that in the cross-attention operation, context states that correspond to future tokens from the current block need to be causally masked out. In this case we set $S = W$, and we pick the window of SSM outputs that correspond to the current block, and a triangular mask is used to implement causal masking of context states. This approach is shown in Figure 2.1.

MH: Multi-Head. This approach differs from Single-Head (SH) in that here the SSM is tasked to generate a separate output for different heads. We use separate $[\mathbf{C}_1, \mathbf{C}_2, ..., \mathbf{C}_H]$ matrices, to produce context states that are fed to the attention heads. This enables the SSM to extract complementary features from the summarized history. The conceptual difference is that the $\mathbf{C}$ matrix, from Equation (2.3.1), has direct access to the full memory state of the SSM (x_k), that in theory could be thought of as a compact representation of the history, before it gets mapped down to a scalar. The Multi-Head (MH) approach is illustrated on the left side of Figure 2.2. Because the H different $\mathbf{C}$ matrices may extract complementary information, the context vector constructed by this method is theoretically less *redundant* compared to the single-head method described above.

MF: Multi-Filter. In this approach the SSM sublayer produces S context states, which we set to be independent from W. This is done by convolving the sequence of embeddings with S different kernels/filters. The output of each convolution operation, corresponding to a specific filter, is a tensor of shape $(B \times L \times D)$. After convolving the input with all the filters, the context states of size D that correspond to the *last token* from the previous window are stacked together to make a $(B \times S \times D)$ tensor. Feed forward networks are then used to lift this tensor to different heads, $(B \times S \times D \times H)$. Different from the previous two approaches, the context is formed by taking only the *last* S context states, from the previous window, outputted by the S SSMs. The context is less redundant because it no longer consists of adjacent SSM states. Since the context is taken from the entries of the previous window, cross-attention masking is no longer required, as shown on the right of Figure 2.2.

The memory states of the Multi-Filter (MF) approach is least *redundant*, while Multi-Head (MH) strikes a middle ground, and Single-Head (SH) has the most redundancy. The incorporation of *redundancy* in these approaches aims to facilitate *retrievability* of the most recent context captured by the SSM, albeit at the expense of potentially inefficient *utilization* of the network capacity. The last approach attains highest *utilization*, as the cross-attention is done in the space of unique features extracted by specialized filters.

2.3.4. Implementation Details

Context IDs & Positional Embedding. To allow distinction between the entries supplied to the attention mechanism, a positional embedding is commonly added to the inputs. When using the Multi-Filter (MF) approach, the collected context states correspond to different features extracted from the sequence, hence we add a set of unique learned "context IDs" to the context states, before using them as input to cross-attention. However, in the cases where the context states correspond to different time-steps along the sequence, namely Single-Head (SH) and Multi-Head (MH) approaches, inherent positional encoding is incorporated into the context states, due to the incremental nature of convolutions; as such, we find the addition of context IDs to be unnecessary. We also realize that we do not need to add global positional bias to the token embeddings, and use a T5-style relative position bias [**Raffel et al., 2019**] instead, as the SSM does also encode positional information into the context.

Down-sampling. Consistent with findings in [**Mehta et al., 2023**], we find FFT operations to be the main source of bottleneck when training SSMs on TPUs. We project the input embeddings to a lower-dimensional space, that is a quarter of embedding size in our experiments, this reduces the required total number of FFTs by a factor of 4. The output of the SSM, i.e. the context states, are later lifted to the original embedding size before being passed to the Block Transformer.

Caching. In addition to the current block of token embeddings, the Block Transformer layer also process keys and values from the previous window stored in a differentiable cache. This is implemented similarly to the sliding window attention pattern suggested in [**Hutchins et al., 2022**] and was originally introduced by Transformer-XL [**Dai et al., 2019**]. Using a causal mask, at every token inference step, we only attend to W most recent tokens, which partially extend to the cached keys and values from the previous block. The cache becomes non-differentiable when it is carried from one sequence to the next. Additionally, we do not use a cache for context states, since to compute the output of the convolution, we need access to the whole sequence. In [**Hutchins et al., 2022**] the last recurrent states of a sequence are stored in a non-differentiable cache and fed to the next training step on the following sequence in the document as a warm-start. We do not find the improvements we report to be attributable to dependencies captured over longer horizons than the training sequence length, mainly due to the nature of the target datasets.

2.4. Results

Our results are presented in Table 2.1. We conduct experiments with BST on three different datasets, PG19, arXiv and GitHub, allowing us to test our method on a suite of varying documents lengths composed of English texts, latex scientific articles and source code.

PG19 dataset is from a large collection of full-length books from Project Gutenberg [**Rae et al., 2020**]. All extracted 28,602 books were published prior to 1919 and contain 6,966,499 English language words. When tokenized, each PG19 book has between 50k-100k tokens. PG19 has become a popular benchmark for measuring progress on long-range language modeling performance. We report the "test" split evaluation performance.

arXiv dataset is a corpus containing scientific and technical articles on the subject of Mathematics [**Wu et al., 2022**]. The arXiv dataset contains latex source code as well as items such as theorems, citations, definitions that are referenced and discussed over long ranges of text. Using the same vocabulary as in [**Wu et al., 2022**] and [**Hutchins et al., 2022**] for a fair comparison, many special characters are broken up into small subwords. As a result, the number of tokens per paper in the arXiv dataset is approximately equal to the number of tokens per book in PG19. We report perplexity on "test" split.

GitHub dataset [**Wu et al., 2022**] is the largest of the three datasets and was assembled by extracting GitHub code repositories with open-source licences. Files were filtered to only contain the following programming languages: C, C++, Java, Python, Go and Typescript. While code files are relatively small, there are many import dependencies between each file. By traversing the directory tree and concatenating all code files along the path, a single document that preserves a repository's structure and dependencies is created. We report performance on the "validation" split.

For a fair comparison with the baselines, we keep the vocabularies consistent as used by [**Hutchins et al., 2022**] and [**Mehta et al., 2023**]. Specifically, we used a pretrained T5 vocab with 32k tokens for PG19 [**Raffel et al., 2020**] and LaMDA vocab with 32k tokens [**Thoppilan et al., 2022**] for both arXiv and GitHub datasets. Due to the long training times and large number of experiments, we only provide error bars for the PG19 ∼200M parameter models by running our models with three different random seeds. BRecT:fixed:skip error bars are from [**Hutchins et al., 2022**].

Model	eval seq. length	window length	number params	TPUv4 hours (k) PG19/arXiv/GitHub	PG19	arXiv	GitHub
SLIDE:12L	4096	512	190M	0.5 / 0.5 / 1.8	12.12	2.69	2.28
TRSF-XL:2048	2048	2048	190M	0.8 / 0.8 / 3.0	11.96	2.48	2.01
BRECT:FIXED:SKIP	4096	512	196M	0.8 / 0.8 / 3.0	11.55 ±1.1	**2.36**	2.04
BST:SH:S4			202M	0.5 / 0.5 / 1.8	11.57 ±1.1	2.51	2.14
BST:MH:S4			218M	0.8 / 0.8 / 1.8	11.60 ±1.1	2.52	2.15
BST:MF:S4			217M	0.5 / 0.5 / 1.8	11.63 ±1.2	2.48	2.07
BST:SH:UNSTRUCT			206M	**0.5** / **0.5** / **1.8**	**11.52** ±1.1	2.49	2.09
BST:MF:UNSTRUCT			221M	**0.5** / **0.5** / **1.8**	11.56 ±1.2	2.44	**2.03**
GSS-HYBRID-L	4096	512	373M	0.8 / 0.8 / 1.8	10.52	2.51	1.88
BST:SH:S4-L			366M	0.8 / 0.8 / 1.8	10.47	2.49	1.86
BST:MF:S4-L			383M	0.8 / 0.8 / 1.8	10.52	2.46	1.84
BST:SH:UNSTRUCT-L			371M	0.8 / 0.8 / 1.8	**10.37**	2.46	1.85
BST:MF:UNSTRUCT-L			388M	0.8 / 0.8 / 1.8	10.42	2.41	**1.83**

Table 2.1. Perplexity of each model. The results for XL:2048, SLIDE:12L and BRECT:FIXED:SKIP are from [**Hutchins et al., 2022**] by converting $\log_2$ of perplexity to raw perplexity. GSS-HYBRID-L performance was taken from [**Mehta et al., 2023**]. Results with $\pm$ are average scores and error bars of runs with three different random seeds. For a *smaller computational budget*, BST provides a small perplexity improvement compared to BRECT on PG19 and GitHub. For the same computational budget, BST outperforms GSS-HYBRID-L across datasets by 1.5% to 4%.

2.4.1. Comparing our Baselines and Models

We experiment three different types Block-State Transformer (BST) models: BST-SH, BST-MH and BST-MF as described in Section 2.3.3. Our models do not use global learned positional embeddings but encode positional awareness with an SSM at the first layer, right after the word embedding layer. We organize models into two groups: (i) *fixed window size* have either a 512 or a 2048 token training window size; and (ii) *fixed parameter count* have either a ~200M or ~400M total parameters. We run experiments with two types of SSMs:

BST:{SH,MH,MF}:S4: encode long context using a Structured State Space Model (S4) [**Gupta et al., 2022**]. As described in Equation (2.3.3), S4 kernel matrix **K** is compiled from matrices **A**, **B** and **C** and is independent of the length of the input evaluation sequence length. We show that the structured parameterization of **K** allows our BST models to generalize to longer lengths. We refer the reader to section 2.4.2 for results on length generalization. We only run one BST:MH using S4 since the model requires 8% more parameters while performing on par with the faster BST:SH variant. BST:MF also has 8% more parameters but performs better on arXiv and GitHub compared to SH. Interestingly, SH performs better than MF on the PG19, a dataset where local context is more important to predict the next token compared to arXiv and GitHub. We posit that this is likely due to the ability of the SH model to *retrieve* the most recent context captured by the SSM.

BST:{SH,MF}:UNSTRUCT: are based of unstructured parameterized convolution filters, inspired by the Hyena Hierarchies [**Poli et al., 2023**] convolutional kernel. We exclude the utilization of the multiplicative gating mechanism employed in Hyena Hierarchies and solely apply the regularizations implemented on the parameterized kernel, denoted as $\bar{\mathbf{K}}$ in Equation (2.3.4). This formulation has two important advantages over S4: (1) the $\bar{\mathbf{K}}$ kernel does not need to be recompiled, allowing speedups when using multiple filters; (2) $\bar{\mathbf{K}}$ has more free parameters because it is no longer restricted by $\mathbf{A}$, $\mathbf{B}$ matrices in equation 2.3.3, potentially providing richer representations that can explain the improved perplexity scores over S4 variants. Nonetheless, UNSTRUCT kernel $\bar{\mathbf{K}}$ relies on learned positional encoding which makes the method less extendable to larger length sequences at inference..

We compare the Block-State Transformer to four different baselines:

TRSF-XL:2048: [**Dai et al., 2019**] is a Transformer with a training window size of 2048. As expected, increasing the window size improves perplexity, especially on the arXiv and GitHub datasets. However, this model performs worse than BST:SH:HYENA on PG19 and is much slower, bottlenecked by the attention layer on higher sequence lengths.

SLIDE:12L: [**Hutchins et al., 2022**] This model is almost identical to TRSF-XL:2048. It uses however a sliding window of size 512 over a segment of 4096 tokens. The sliding window is differentiable over two blocks, while TRSF-XL does not backpropagate through the cached keys and values from the previous window. This simple baseline is closest in terms of training speed to BST:SH. The perplexity scores show that integrating a representation of the past, as with BRECT and BST, positively impacts LM performance.

BRECT:FIXED:SKIP: [**Hutchins et al., 2022**] is the strongest performing and fastest Block-Recurrent Transformer architecture in [**Hutchins et al., 2022**]. This architecture is very similar to SLIDE:12L. There is however a sequential recurrent "skip" configuration, a simple linear layer gating mechanism that combines current block hidden representation with past information from the previous blocks.

GSS-HYBRID-L: [**Mehta et al., 2023**] is the closest SSM-Transformer hybrid model that was tested on long-range language modeling tasks. GSS-HYBRID-L is based on the Diagonal State Space (DSS) [**Gupta et al., 2022**]. DSS and S4 are similar in performance and architecture, only differing on the initialization of the kernel $\mathbf{K}$ [**Gu et al., 2022b**].

[**Gupta et al., 2022**] further improves on DSS for LM tasks by introducing a Gated State Space version called GSS, which performs better on PG19, arXiv and GitHub. Unlike our method, GSS-Hybrid-L does not directly integrate SSMs states into the attention mechanism but only interleaves 32 GSS layers with Transformer layers. It must be noted that the GSS-Hybrid-L scores were obtained after grid searching over four learning rates $\{6.4,3.2,1.6,0.8\}\times 10^{-3}$ and used a different learning rate and weight decay for the SSM layer and the Transformer layer to avoid training instabilities. In our experiment, we did not use grid search and used the same learning rate for all layers. BST results demonstrate that integrating SSM states into the Transformer attention provides larger benefits than interleaving SSM and attention layers as in GSS-Hybrid-L.

Fixed compute budget. As seen in Table 2.1, we track the exact amount of compute in TPUv4 hours that was spent training each model. The training TPUv4 hours for Slide:12L, Trsf-XL:2048, BRecT:fixed:skip and GSS-Hybrid-L were taken from [**Mehta et al., 2023**]. The TPUv4 hours metric measures the compute cost of training models. For our experiments, we align our training times with GSS-Hybrid-L for a fair comparison. Smaller parameter models all have 12 layers, 8 heads of size 128, embedding vectors of size 1024, an MLP with a hidden layer size of 4096 with ReLU activation functions. For larger BST models, we double the intermediate layer size from 4096 to 8192 and increase the number of attention heads to 12.

Training details. We use the same training setup as [**Hutchins et al., 2022**] and we perform our experiments using the Meliad library[2] in JAX/Flax [**Bradbury et al., 2018, Heek et al., 2023**]. We use the Adam optimizer [**Kingma and Ba, 2015**] and a batch size of 32 and a sequence length L of 4k for training. Using a structured SSM's recurrence (such as S4) in the first layer allows us to extend the positional encoding to various lengths at inference. Smaller BST models have Block-State layer integrated in Transformer layers {1, 7, 9} and larger BST models at layers {1, 5, 7, 9}. Since our datasets contain long documents, it is possible to train on larger sequence lengths L. Training on 4k sequence lengths allows us to test length generalization since the convolution kernel $\mathbf{K}$ in Equation (2.3.3) can be extended to any sequence length L. However, since we show in Section 2.4.2 that our model works well when extended to unseen lengths, we did not find it necessary to run expensive experiments with higher sequence lengths. For the MF model variants, we lower the SSM state dimension D by an additional factor of two to improve FFT efficiency. The state dimension reduction has negligible impact to perplexity. The MF models have $S = 32$ filters while the larger MF models have $S = 64$ filters.

[2]https://github.com/google-research/meliad

2.4.2. Evaluating Length Generalization capabilities

We present our length generalization analysis and report perplexity in Figure 2.3. Our models and baselines all have ∼400M parameters, are trained on a sequence length of 4k and tested on sequences with *lower* and *higher* sequence lengths of {512, 16k, 65k}.

We notice that all models have similar perplexity for sequence lengths of 512. Both BST:SH:S4-L and GSS-Hybrid-L generalize well on 16k and 65k sequence lengths for PG19 and GitHub. For arXiv, GSS-Hybrid-L and BST:MF:unstruct-L perplexities increase drastically, potentially due to noise in the arXiv dataset (as indicated by variation in perplexity metric over time). [**Mehta et al., 2023**] also reported that larger GSS models had difficulty generalizing to higher lengths. Interestingly, for arXiv again, BRecT:fixed:skip-L performs very well at higher sequence lengths. We hypothesize that the Block-Recurrent model's access to the entire past during training, via a non-differentiable cache of representations across sequences, helps retain a "memory" of dependencies between key items in an arXiv article allowing the model to access past symbols, definitions, theorems or equations beyond the 4k training sequence length. We also note that BST:MF:unstruct-L and BRecT:fixed:skip-L outperform other methods on PG19 up to a sequence length of 16K. Perplexity performance on PG19 is perhaps less reliant on long term relationships between tokens, which can explain the performance of models that have no explicit built-in mechanisms for length generalization.

The analysis also allows us to draw a clear distinction between *structured* and *unstructured* SSMs integrated in hybrid architectures. As previously mentioned in Section 2.3.1, SSMs such as DSS and S4 use a structured kernel $\mathbf{K}$, built from learned matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ for any sequence length L in Equation 2.3.3. Since $\mathbf{K}$ is extendable to any arbitrary sequence length L, both BST:SH:S4-L and GSS-Hybrid-L have a build-in mechanism for length generalization that the unstructured BST:MF:unstruct-L model does not. BST:MF:unstruct-L performs best on the training sequence of 4K and is on-par for 512 with perplexity increasing for unseen 16K and 65K sequence lengths. **BST:SH:S4-L has by far the best perplexity for 65K sequence lengths on PG19, GitHub and arXiv.**

2.4.3. Ablation Studies

In the following section, we perform ablations to investigate (1) the placement of a *single* SSM layer in Table 2.2 in the overall architecture, (2) the effects of the number of SSM layers added in Table 2.3, and (3) the size D of the SSM state in Table 2.4. For the ablations, we use the ∼200M parameter BST:SH:S4, since it is the fastest model, and assess various configurations on PG19.

In Table 2.2, we experiment adding a single BST layer at layer indices $3, 6, 9, 12$. We notice that a single BST layer with state size $D = 16$ located closer to the middle of the

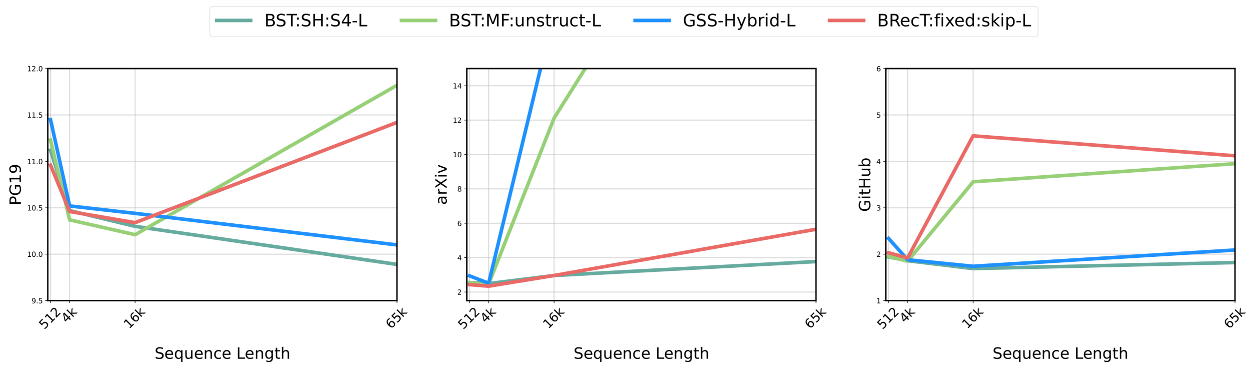


Fig. 2.3. Length Generalization for sequence lengths {512, 16k, 65k} on PG19 (left), arXiv (middle) and GitHub (right). BST:SH:S4-L generalizes better than any other baseli nes, including GSS-Hybrid-L that uses GSS, a structured SSM. GSS-Hybrid-L numbers are from [**Mehta et al., 2023**].

Table 2.2. A single BST at various layer index.

Layer index	Perplexity
3	12.41
7	11.92
9	11.88
12	12.03

Table 2.3. Multiple BST layers at various locations.

Num layers	Perplexity
2	11.69
3	11.57
4	11.21
5	11.20

Table 2.4. Increasing BST's S4 model state size D.

State Size	Perplexity	Step Time
8	11.95	$\times 0.7$
16	11.57	$\times 1.0$
32	11.55	$\times 1.8$
64	11.54	$\times 3.2$

whole Block Transformer stack, at index $= 9$, has the greatest effect on perplexity. This finding is inline with findings in prior work [**Wu et al., 2022, Hutchins et al., 2022**].

In Table 2.3, we test if adding multiple BST layers yields improvements on performance. We start with BST layers with state size $D = 16$ at indices $0, 9$. We follow by adding another BST layer at index 7 for a total of three BST layers and then another at index 5, followed by another at index 12. Adding more BST layers lowers perplexity. However, the results seem to plateau at 5 BST layers. We note also that there is a 3.5% training step time increase for each added layer.

In Table 2.4, we train our models with different state sizes D. For the state size ablation, we use three BST layers at indices $0, 7, 9$. We find that increasing D improves perplexity to the detriment of training speed (step time). For this reason, we chose $D = 16$ for Table 2.1 BST results.

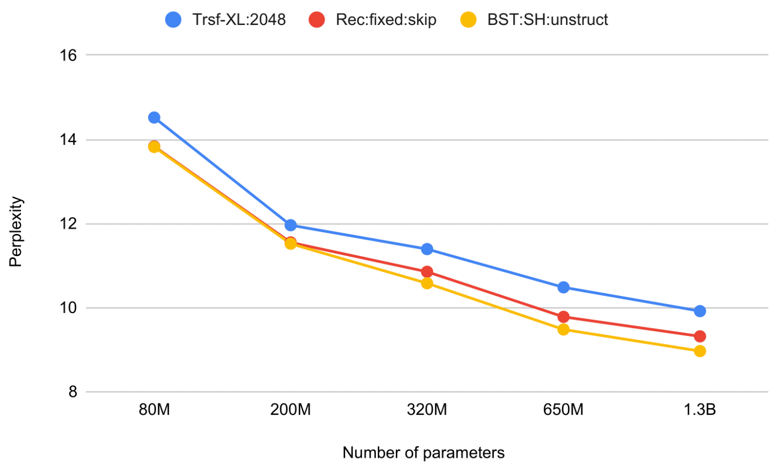


Fig. 2.4. Scaling properties, BST vs BRecT vs Trsf-XL on PG-19. Red: 12-layer Block-Recurrent Transformer (Rec:fixed:skip), Yellow: 12-layer Block-State Transformer (BST:SH:unstruct), and Blue: 13-layer Transformer-XL (Trsf-XL-2048).

2.4.4. Language Modeling at Scale

In this section, we compare how BST scales compared to Transformer-XL with $4\times$ the window size and BRecT. In Figure 2.4, we see that at lower scales, from 80M to 200M, BRecT and BST have very similar performances. Beyond 200M, the perplexity performance percentage gap between BRecT and BST increases from 2.5% at 200M paramaters to 4.0% at 1.3B parameters. The perplexity performance percentage gap between BRecT and Trsf-XL is even more pronounced as it starts at 7.6% at 200M parameters to 10.6% at 1.3B parameters.

2.4.5. Long Range Arena (LRA)

While the main focus of our research was to demonstrate that hybrid Transformer-SSM models are efficient and perform well on long context autoregressive LM, we also evaluate our method on standard classification task where long range dependencies in a sequence are important to capture. In Table 2.5, we present our results on the Long Range Arena (LRA) benchmark [**Tay et al., 2020b**] which incorporates three different modalities including text, images, and mathematical expressions. The LRA dataset also tests models on various sequence lengths from 1K to 16K.

2.4.6. Efficiency

The improvement over Block-Recurrent Transformers, with time complexity of $\mathcal{O}((W^2 + S^2 + 2SW) \cdot L/W) \approx \mathcal{O}(L \cdot W)$, follows from the ability to run the Block Transformer's cells in parallel. The time complexity of the Block-State Transformer layer is comprised of the

Model	ListOpts	Text	Retrieval	Image	Pathfinder	Path-X	Avg
Transformer	36.37	64.27	57.46	42.44	71.40	✗	53.66
Linear Transformer	16.13	65.90	53.09	42.34	75.30	✗	50.46
Reformer	37.27	56.10	53.40	38.07	68.50	✗	50.56
Performer	18.01	65.40	53.82	42.77	77.05	✗	51.18
BigBird	36.05	64.02	59.29	40.83	74.87	✗	54.17
Mega	**63.14**	**90.43**	91.25	**90.44**	**96.01**	97.98	**88.21**
S4D	60.47	86.18	89.46	88.19	93.06	91.95	84.89
S4	59.60	86.82	90.90	88.65	94.20	96.35	86.09
S5	62.15	89.32	**91.40**	88.00	95.33	**98.58**	87.46
Methods with chunked input sequences							
BRecT:fixed:skip	37.29	66.14	58.76	50.41	76.33	75.89	60.80
Mega-chunk	58.76	**90.19**	**90.97**	85.80	94.41	93.81	85.66
BST:SH:S4 (ours)	**61.49**	87.63	90.51	**91.07**	**95.75**	**95.28**	**86.96**

Table 2.5. Performance on Long Range Arena (LRA). For a fair comparison, we adjust the number of layers and model dimensions on each task so that BST and BRecT have similar number of parameters with S4 and Mega-chunk. BRecT results are from our own runs and all other baselines are from published results.

time complexity of the state space model sublayer, $\mathcal{O}(D \cdot L \log L)$, in addition to the time complexity required to execute the Transformer over the given context chunks (blocks) in parallel, $\mathcal{O}(W^2)$.

In spite of the superlinear growth of the SSM sublayer, our experiments indicate that significant performance improvements, up to a factor of 6, remain evident for sequences as long as 65k tokens, the point at which hardware saturation began to occur. When using a structured SSM, the computational complexity is closely tied to the internal memory state size of the SSM, N – specifics may vary depending on the exact type of the SSM. We set $N = 16$ when reporting performance. Left side of Figure 2.5 shows the results of benchmarking the forward-pass of a Block-State Transformer layer on GPU. Our proposed layer runs almost 6-11$\times$ faster than Block-Recurrent Transformers (including recurrent units), and yields comparable performance to a Slide:12L layer, i.e. BRecT without the recurrence. At 4k sequence length, which is mostly used during training, BRecT layer runs almost 15$\times$ slower than Slide:12L with the same window size. We manage to reduce this gap to less than 2$\times$ with BST layer. To reflect a realistic model, for these experiments we use a fixed window length of 128, an internal state size of 16 for the SSM, and 16 heads. Moreover, to highlight the performance gains that are only due to parallelization made possible by

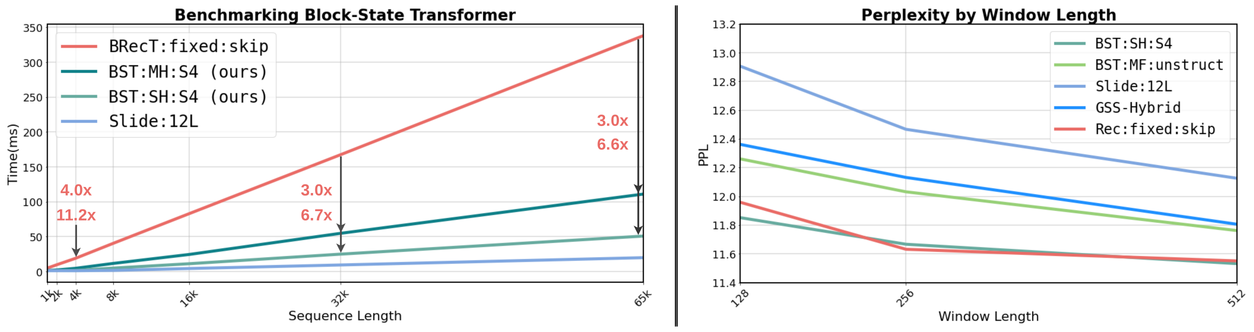


Fig. 2.5. Left: The forward-pass computation time of a BST layer is compared against a layer of BRecT and Slide:12L. These experiments were executed on GPU, to demonstrate and exploit the parallelizability of BST layers. BST:SH is 6-11× faster than BRecT while BST:MH is 3-4× faster. **Right**: Perplexity of the trained models using different window lengths. The figure shows that increasing the training window length results, as expected, in better perplexity scores. We find however that both BST:MF:Hyena and BRecT:fixed:skip are the least impacted by decreasing window lengths.

our framework, we use same embedding size as input to the SSM, which is 512. Note that we use the vanilla implementation of FFT and inverse FFT operations provided by JAX [**Bradbury et al., 2018**]. However, we believe that the speed of our method can be further improved with recent and faster hardware-specific I/O-aware implementations introduced in other auto-diff frameworks.

2.5. Limitations

While BST's SSM layer allows the model to unroll and parallelize the recurrence that models long-term context between blocks of tokens, the SSM variants are reliant on efficient FFT operations. We have found that the FFT operation is an important speed bottleneck on TPUs that needs to be resolved to better scale BST to many layers and larger models. While we are still investigating the reasons, we found that JAX FFT was 4× faster on GPUs. Further, new SSM variants such as S5 [**Smith et al., 2023**] bypass FFT operations using a binary associative operator[3]. Our implementation is modular enough that we can simply plug in S5 or use other FFT implementations.

One of our assumptions is that BST's SSM layer is able to capture the right long-term dependencies for each block. The SSM recurrence at step $T = t$ provides a summarized representation of previous steps for $T = 0$ to $T = t$. However, a single vector representation may not be powerful enough to support all important long-term dependencies. Despite the perplexity improvements on long-range language modeling tasks, this assumption needs to be

[3]In JAX, this is equivalent to using *jax.lax.associative_scan*.

tested on other long-range classification tasks such as Long Range Arena [**Tay et al., 2020b**] as well. It is possible that our model can perform better if we feed to the attention layer $k = W$ SSM representations that are chosen by a top-k retrieval operation, similar to the one in Memorizing Transformer [**Wu et al., 2022**].

2.6. Conclusion

We have introduced a model that combines the attention mechanism of Transformers with the long-range memory mechanism, and parallelism afforded by State Space Models. We explored several memory state variants that make different trade-offs between *redundancy* and *retrievability*. Experiments show that our model can minimize perplexity on par with and often improves upon recent competing baselines, while achieving up to more than $10\times$ speedups at the layer level, provided there is hardware support to fully take advantage of parallelism. This is an appealing property for scaling up BST which makes the addition of SSMs into Transformers computationally appealing. We show that integrating SSM states into the Transformer attention provides larger benefits than simply interleaving SSM and attention layers. Finally, we show that the model generalizes to longer sequences than it was trained.

Acknowledgments

We would like to thank Caglar Gulcehre and Albert Gu for helpful discussions and support with the S4 codebase. We would also like to express our gratitude to Delesley Hutchins for providing valuable guidance throughout the project, as well as Xavier Garcia and Courtney Paquette for their careful review of the manuscript, where they identified and rectified several errors.

References

[Bradbury et al., 2018] Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. (2018). JAX: composable transformations of Python+NumPy programs.

[Brown et al., 2020] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. *CoRR*, abs/2005.14165.

[Chihara, 2011] Chihara, T. (2011). *An Introduction to Orthogonal Polynomials*. Dover Books on Mathematics. Dover Publications.

[Child et al., 2019] Child, R., Gray, S., Radford, A., and Sutskever, I. (2019). Generating long sequences with sparse transformers. *CoRR*, abs/1904.10509.

[Choromanski et al., 2020] Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlós, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., Belanger, D., Colwell, L. J., and Weller, A. (2020). Rethinking attention with performers. *CoRR*, abs/2009.14794.

[Chowdhery et al., 2022] Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., Schuh, P., Shi, K., Tsvyashchenko, S., Maynez, J., Rao, A., Barnes, P., Tay, Y., Shazeer, N., Prabhakaran, V., Reif, E., Du, N., Hutchinson, B., Pope, R., Bradbury, J., Austin, J., Isard, M., Gur-Ari, G., Yin, P., Duke, T., Levskaya, A., Ghemawat, S., Dev, S., Michalewski, H., Garcia, X., Misra, V., Robinson, K., Fedus, L., Zhou, D., Ippolito, D., Luan, D., Lim, H., Zoph, B., Spiridonov, A., Sepassi, R., Dohan, D., Agrawal, S., Omernick, M., Dai, A. M., Pillai, T. S., Pellat, M., Lewkowycz, A., Moreira, E., Child, R., Polozov, O., Lee, K., Zhou, Z., Wang, X., Saeta, B., Diaz, M., Firat, O., Catasta, M., Wei, J., Meier-Hellstern, K., Eck, D., Dean, J., Petrov, S., and Fiedel, N. (2022). Palm: Scaling language modeling with pathways.

[Cooley and Tukey, 1965] Cooley, J. and Tukey, J. (1965). An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301.

[Dai et al., 2019] Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., and Salakhutdinov, R. (2019). Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*.

[Devlin et al., 2018] Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

[Fu et al., 2023a] Fu, D. Y., Dao, T., Saab, K. K., Thomas, A. W., Rudra, A., and Ré, C. (2023a). Hungry hungry hippos: Towards language modeling with state space models.

[Fu et al., 2023b] Fu, D. Y., Epstein, E. L., Nguyen, E., Thomas, A. W., Zhang, M., Dao, T., Rudra, A., and Ré, C. (2023b). Simple hardware-efficient long convolutions for sequence modeling.

[Gu et al., 2020] Gu, A., Dao, T., Ermon, S., Rudra, A., and Re, C. (2020). Hippo: Recurrent memory with optimal polynomial projections.

[Gu et al., 2022a] Gu, A., Goel, K., and Ré, C. (2022a). Efficiently modeling long sequences with structured state spaces.

[Gu et al., 2022b] Gu, A., Gupta, A., Goel, K., and Ré, C. (2022b). On the parameterization and initialization of diagonal state space models.

[Gupta et al., 2022] Gupta, A., Gu, A., and Berant, J. (2022). Diagonal state spaces are as effective as structured state spaces.

[Heek et al., 2023] Heek, J., Levskaya, A., Oliver, A., Ritter, M., Rondepierre, B., Steiner, A., and van Zee, M. (2023). Flax: A neural network library and ecosystem for JAX.

[Hochreiter, 1998] Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(2):107–116.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

[Hua et al., 2022] Hua, W., Dai, Z., Liu, H., and Le, Q. (2022). Transformer quality in linear time. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S., editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 9099–9117. PMLR.

[Hutchins et al., 2022] Hutchins, D., Schlag, I., Wu, Y., Dyer, E., and Neyshabur, B. (2022). Block-recurrent transformers. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K., editors, *Advances in Neural Information Processing Systems*.

[Katharopoulos et al., 2020] Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. (2020). Transformers are RNNs: Fast autoregressive transformers with linear attention. In III, H. D. and Singh, A., editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5156–5165. PMLR.

[Khandelwal et al., 2018] Khandelwal, U., He, H., Qi, P., and Jurafsky, D. (2018). Sharp nearby, fuzzy far away: How neural language models use context. *CoRR*, abs/1805.04623.

[Kingma and Ba, 2015] Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y., editors, *ICLR (Poster)*.

[Ma et al., 2023] Ma, X., Zhou, C., Kong, X., He, J., Gui, L., Neubig, G., May, J., and Zettlemoyer, L. (2023). Mega: Moving average equipped gated attention.

[Mehta et al., 2023] Mehta, H., Gupta, A., Cutkosky, A., and Neyshabur, B. (2023). Long range language modeling via gated state spaces. In *The Eleventh International Conference on Learning Representations*.

[OpenAI, 2023] OpenAI (2023). Gpt-4 technical report.

[Poli et al., 2023] Poli, M., Massaroli, S., Nguyen, E., Fu, D. Y., Dao, T., Baccus, S., Bengio, Y., Ermon, S., and Ré, C. (2023). Hyena hierarchy: Towards larger convolutional language models.

[Rae et al., 2020] Rae, J. W., Potapenko, A., Jayakumar, S. M., Hillier, C., and Lillicrap, T. P. (2020). Compressive transformers for long-range sequence modelling. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

[Raffel et al., 2019] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683.

[Raffel et al., 2020] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67.

[Smith et al., 2023] Smith, J. T. H., Warrington, A., and Linderman, S. W. (2023). Simplified state space layers for sequence modeling.

[Tay et al., 2020a] Tay, Y., Bahri, D., Yang, L., Metzler, D., and Juan, D.-C. (2020a). Sparse Sinkhorn attention. In III, H. D. and Singh, A., editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 9438–9447. PMLR.

[Tay et al., 2020b] Tay, Y., Dehghani, M., Abnar, S., Shen, Y., Bahri, D., Pham, P., Rao, J., Yang, L., Ruder, S., and Metzler, D. (2020b). Long range arena: A benchmark for efficient transformers.

[Thoppilan et al., 2022] Thoppilan, R., Freitas, D. D., Hall, J., Shazeer, N., Kulshreshtha, A., Cheng, H., Jin, A., Bos, T., Baker, L., Du, Y., Li, Y., Lee, H., Zheng, H. S., Ghafouri, A., Menegali, M., Huang, Y., Krikun, M., Lepikhin, D., Qin, J., Chen, D., Xu, Y., Chen, Z., Roberts, A., Bosma, M., Zhou, Y., Chang, C., Krivokon, I., Rusch, W., Pickett, M., Meier-Hellstern, K. S., Morris, M. R., Doshi, T., Santos, R. D., Duke, T., Soraker, J., Zevenbergen, B., Prabhakaran, V., Diaz, M., Hutchinson, B., Olson, K., Molina, A., Hoffman-John, E., Lee, J., Aroyo, L., Rajakumar, R., Butryna, A., Lamm, M., Kuzmina, V., Fenton, J., Cohen, A., Bernstein, R., Kurzweil, R., Aguera-Arcas, B., Cui, C., Croak, M., Chi, E. H., and Le, Q. (2022). Lamda: Language models for dialog applications. *CoRR*, abs/2201.08239.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

[Wang et al., 2020] Wang, S., Li, B. Z., Khabsa, M., Fang, H., and Ma, H. (2020). Linformer: Self-attention with linear complexity. *CoRR*, abs/2006.04768.

[Wu et al., 2022] Wu, Y., Rabe, M. N., Hutchins, D., and Szegedy, C. (2022). Memorizing transformers. In *International Conference on Learning Representations*.

[Zaheer et al., 2020] Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., et al. (2020). Big bird: Transformers for longer sequences. *Advances in Neural Information Processing Systems*, 33.

[Zhang et al., 2022] Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., Mihaylov, T., Ott, M., Shleifer, S., Shuster, K., Simig, D., Koura, P. S., Sridhar, A., Wang, T., and Zettlemoyer, L. (2022). Opt: Open pre-trained transformer language models.

Second Article Appendices

More detailed comparisons with existing baselines

This section provides the reader with a more in-depth comparison with similar architectures. We cover BRecT [**Hutchins et al., 2022**] in Section 2.6 and GSS-Hybrid [**Mehta et al., 2023**] in Section 2.6.

Comparison with Block Recurrent Transformer (BRecT). The Block Transformer sublayer (i.e Slide:12L) processes keys and values from the previous window stored in a differentiable cache. This is implemented similarly to the sliding window attention pattern suggested in [**Hutchins et al., 2022**] and was originally introduced by Transformer-XL [**Dai et al., 2019**]. Using a causal mask, at every token inference step, the attention mechanism is applied to blocks of tokens of size W and is partially extended to the cached keys and values from the previous block with the sliding window. BRecT, as explained in [**Hutchins et al., 2022**], uses a non-differentiable cache that is carried from one sequence of size L to the next[4]. The last recurrent states of a sequence are stored in a non-differentiable cache and fed to the next training step on the following sequence in the document as a warm-start. We do not pass such a representation, since to compute the output of the convolution, we need access to the whole sequence. We believe that this is one advantage that BRecT has over our method, especially for very long examples that split into ordered sequences of length L, since the cache carried from one sequence to the next can provide very useful long-range information and (weak) access to the whole past. Since we need the whole sequence to compute SSM states, history beyond L may be lost in the process. We believe that BST can further be improved by adding non-differentiable sequence cache for very long documents.

While in other architectures, the history between blocks of tokens is not modeled, both BST and BRecT use a mechanism to model previous block context. The authors of BRecT experiment with various sequential gating mechanisms to condense the information from past blocks. With BST, we use SSM to provide context from previous blocks to the current block as explained in Section 2.3.2.

Comparison with the Transformer GSS-Hybrid. GSS-Hybrid [**Mehta et al., 2023**] is a SSM-Transformer hybrid architecture that we first describe in Section 2.4.1. The architecture is significantly different from BST. GSS-Hybrid is primarily composed of Gated State Space (GSS) layers and has a few interleaved Transformer layers at every 4th layer starting with

[4]In our work and in [**Hutchins et al., 2022**], a document is split into multiple sequences of size L and each sequence is split into multiple blocks of size W

the 2nd layer. BST on the other hand is mainly composed of Block Transformer layers and has Block-State Transformer layers at positions {1, 7, 9} for the ∼200M model and {1, 5, 7, 9} for the ∼400M model. Our hybrid does not stack SSM and Transformer layers like the GSS-Hybrid but rather replaces the recurrence in BRecT with an SSM such as S4. In BST, the SSM generates states for each Block Transformer representations and we then use cross-attention to mix the states and the self-attention outputs. The authors in [**Mehta et al., 2023**] initially built GSS, a gated version of DSS [**Gupta et al., 2022**], to (1) reduce SSM parameter dimensions, (2) stabilize training of the SSM and (3) allow better length generalization. However, when experimenting with SSMs such as S4 or DSS, we found that the gating was not necessary to achieve all three objectives stated above. We decided that using GSS's Gated Attention Unit [**Hua et al., 2022**] was therefore not needed when integrating SSM states into the attention mechanism. We also reiterate that the authors in [**Mehta et al., 2023**] used hyperparameter search to get the best performance while we did not.

JAX Implementation of BST

Pseudocode 2.1 contains a function that implements convolution of multiple filters over the same input sequence using FFT and inverse FFT operations. Pseudocodes 2.2, 2.3 and 2.4 respectively implement context state collection of `BST` variants: Single-Head (`SH`), Multi-Head (`MH`) and Multi-Filter (`MF`). Finally, Pseudocode 2.5 runs the Block Transformer sublayer in parallel by feeding the context states to their corresponding block.

```
"""Unstructured filters and convolutions."""

import jax
from jax import numpy as jnp
from einops import rearrange

win_length = 512    # (w)
seq_length = 4096   # (l)

def get_filters_unstruct(channels):
    """Returns trainable filters and biases.

    Args:
        channels: number of filters.

    Returns:
        h: filter of shape (seq_length, channels, dim)
        b: bias of shape (channels, dim)
    """
    t = jnp.linspace(0.0, 1.0, seq_length)
    h = jnp.exp(- alpha * t) * dense(positional_emb(t))
    b = get_bias()
    return h, b

def multichannel_convolution(u, h, b):
    """Multichannel convolution function.

    Args:
        u: input of shape (seq_length, dim)
        h: filters of shape (seq_length, channels, dim)
        b: bias of shape (channels, dim)
    """
    h = rearrange(h, "l c d -> c d l")

    fft_size = seq_length * 2
    u_f = jnp.fft.rfft(x, n=fft_size)
    h_f = jnp.fft.rfft(h, n=fft_size)

    y = jnp.fft.irfft(h_f * x_f, n=fft_size, norm="forward")[
            ..., :seq_length]       # (c, d, l)
    y = y + x * b[..., None]        # (c, d, l)
    y = rearrange(y, "c d l -> l d c")
    return y
```

Pseudocode 2.1. Unstructured filters and convolutions.

```
"""Context state collection for BST-SH variant."""

num_heads = 8       # (h)
num_states = 32     # (s)

# (SH): Single-Head
def SH_context_states(u):
    """Single-Head Context Collection."""
    h, b = get_filters_[unstruct/s4](channels=1)
    y_1 = multichannel_convolution(u, h, b)  # y_1: (l, d, 1)

    # lift to multiple heads
    y_h = dense(y_1)  # y_h: (l, d, h)

    context_states = jnp.split(
            y_h, seq_length // win_length, axis=0)
    return context_states # (l/w, w, d, h)
```

Pseudocode 2.2. Context state collection for `BST-SH` variants.

```
"""Context state collection for BST-MH variant."""

# (MH): Multi-Head
def MH_context_states(u):
    """Multi-Head Context Collection."""
    h, b = get_filters_[unstruct/s4](channels=num_heads)
    y_h = multichannel_convolution(u, h, b)  # y_h: (l, d, h)

    context_states = jnp.split(
            y_h, seq_length // win_length, axis=0)
    return context_states # (l/w, w, d, h)
```

Pseudocode 2.3. Context state collection for `BST-MH` variants.

```
"""Context state collection for BST-MF variant."""

# (MF): Multi-Filter
def MF_context_states(u):
    """Multi-Filter Context Collection."""
    h, b = get_filters_[unstruct/s4](channels=num_states)
    y_s = multichannel_convolution(u, h, b)  # y_s: (l, d, s)
    context_states = jnp.split(
            y_s, seq_length // win_length, axis=0)
    # context_states: (l/w, w, d, s)

    # collect the last context states
    context_states = context_states[:, -1, ...] # (l/w, d, s)
    context_states = rearrange(
            context_states, "lw d s -> lw s d")

    # shift context states corresponding to windows
    context_states = jnp.roll(context_states, 1, axis=1)

    # replace the initial window with trainable weights
    init_context = get_init_context(num_states) # (d, s)
    context_states[0] = init_context

    # lift to multiple heads
    context_states = dense(context_states)

    return context_states # (l/w, s, d, h)
```

Pseudocode 2.4. Context state collection for BST-MF variants.

```
"""Block-State Transformer Layer."""

# Block Transformers are non-recurrent and parallelizable.
block_transformer = jax.vmap(BRecT.nonrecurrent_cell)

def BST(u):
    """Block-State Transformer Layer."""
    global MF # True if Multi-Filter, False otherwise (SH/MH)

    # split inputs into windows (l/w, w, d)
    u = jnp.split(u, seq_length // win_length, axis=0)

    # collect context states from SSM outputs
    context_states = [SH/MH/MF]_context_states(u)

    # pass the contexts in place of recurrent states
    y = block_transformer(
            token_embeddings=u,
            recurrent_state=context_states,
            use_cross_attn_causal_mask=not MF,
            use_cross_positional_emb=MF, # context IDs
    )

    return rearrange(y, "lw w d -> (lw w) d") # (l, d)
```

Pseudocode 2.5. Block-State Transformer Layer.

Conclusion

This thesis, mainly structured in two articles, tackles the challenge of extending sequence modeling lengths during both training and inference phases.

In the first article, *Course Correcting Koopman Representations* [**Fathi et al., 2023a**], we explore fully-observable, deterministic dynamical systems where the "hidden" state in recurrence directly corresponds to the system's state. We identify the issue of drift in latent dynamics models and establish the necessity of a reencoding mechanism during inference. To address this issue, we propose periodic reencoding as a simple yet effective solution. Periodic reencoding is necessary at inference time, however, training time usage also enhances the prediction quality. We also show that the findings in the linear case translate to scenarios in which a wide MLP is used as the latent dynamics component. Through periodic reencoding, we enable the unrolling of latent dynamics models over indefinitely extended horizons, effectively circumventing the problem of drift. Although our focus was on Koopman autoencoders, we highlight their architectural equivalence to single-layer linear recurrent models. These models can be considered as fundamental components of State-Space Models.

In the second article, *Block-State Transformers* [**Fathi et al., 2023b**], we combine State-Space Models (SSMs) and Transformers. This combination aims to harness the strengths of both, resulting in a model that surpasses other baselines in tasks involving long-range language modeling and the Long Range Arena benchmark. BST can be viewed as an evolution of "Block-Recurrent Transformers" [**Hutchins et al., 2022**]. In BRT, the interaction between transformer blocks occurs in a slow, sequential manner through recurrence. A notable limitation of this approach is the delayed processing of information by subsequent blocks. BST rectifies this limitation, offering a more efficient information processing methodology ($10\times$ speedup), through contextualizing the sequence via SSMs. We show that SSMs, when combined with transformers, can be unrolled over horizons much longer than those used during training. This property, which we term "length generalization capability," is lacking in both Block-Recurrent Transformers (BRT) and standalone SSMs.

In summary, this thesis bridges theory and practice in extending sequence modeling lengths, articulated through two main articles. The first article adopts a theoretical perspective, focusing on identifying the fundamental issues in latent dynamics sequence models. In contrast, the second article takes a more practical, hardware-aware approach, concentrating specifically on large language models. Both articles in this thesis are closely linked to State-Space Models (SSMs), increasingly recognized for surpassing Transformers in various benchmarks – including language, progressively challenging their dominance. The contributions of this thesis underscore the increasing importance of State-Space Models (SSMs), while shedding light on the core challenges associated with these models. This work not only advances our understanding of SSMs but also paves the way for further exploration and refinement in the field.

References

[Fathi et al., 2023a] Fathi, M., Gehring, C., Pilault, J., Kanaa, D., Bacon, P.-L., and Goroshin, R. (2023a). Course correcting koopman representations.

[Fathi et al., 2023b] Fathi, M., Pilault, J., Firat, O., Pal, C., Bacon, P.-L., and Goroshin, R. (2023b). Block-state transformers.